2013

Senior Software Consultant

Burak Selim Şenyurt

[BURAK SELİM ŞENYURT - ALMANAC 2013 - V2]
www.buraksenyurt.com sitesinde yayınlamış olduğum teknik içerikli post' ların toplamını içeren derlenmiş dökümandır.

Contents

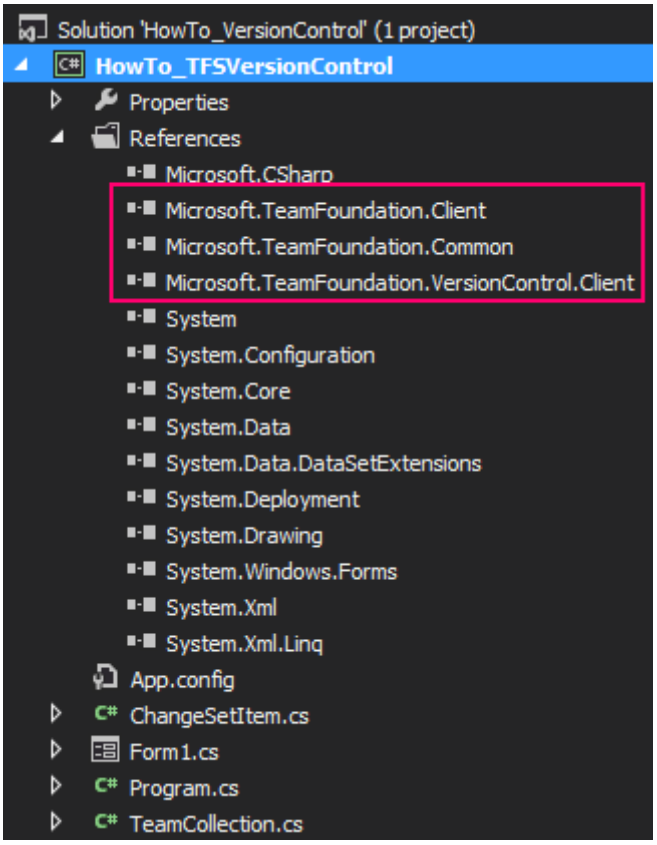
TFS Version Control Hizmetine Kısa Bir Bakış

Pazartesi, 28 Ekim 2013 22:55 by bsenyurt

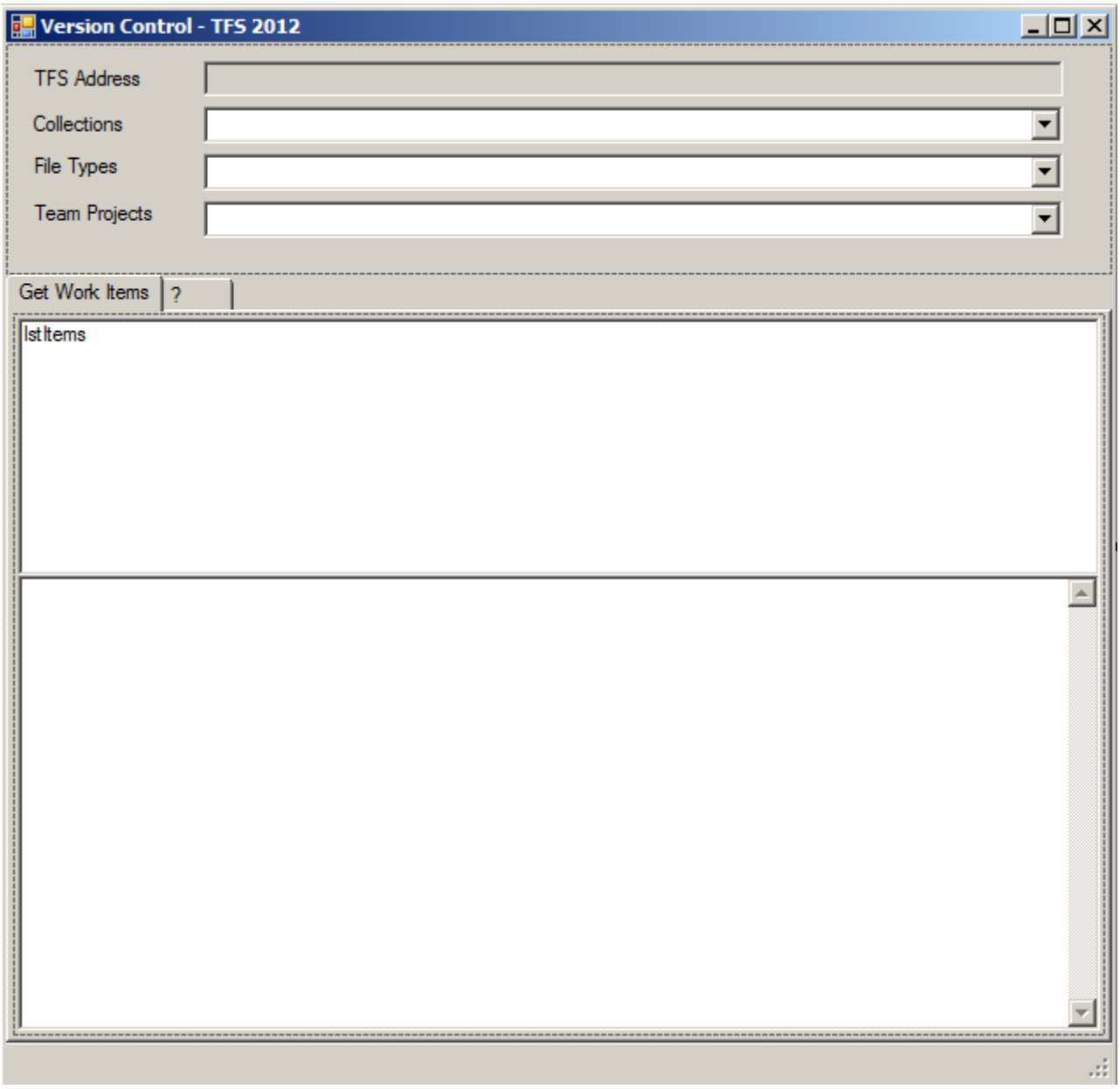
Merhaba Arkadaşlar,
TFS Web Services kullanımlarını incelediğimiz bu yazımızda, en popüler hizmetlerden birisi olan**Work Item Tracking** servisine odaklamıştık. Bu servisten yararlanarak özellikle template bazlı öğelerin*(Task, Bug, Product Back Log Item gibi)* nasıl okunabileceğini öğrenmiştik. Çok doğal olarak daha pek çok servis kullanımı söz konusudur. Önemli olan nokta ise, ilgili servislerin **TFS Client Object Model** üzerinden kullanılabileceğidir.
İşte bu yazımızda örnek bir servis kullanımını daha incelemeye çalışıyor alacağız. Amacımız **VersionControl** hizmetini ele alarak bir **Team Project** içerisinde yer alan ve **source control**'e dahil edilmiş çeşitli içerikleri elde edebilmek. Ağırlıklı olarak kod dosylarının içeriğini görmeyi hedeflediğimizi söyleyebiliriz. Örneğimizi bir **Windows Forms** uygulaması olarak geliştirebiliriz. Dilerseniz hiç vakit kaybetmeden ilgili **TFS Assembly** dosyalarını referans ederek işe başlayalım.*(Tabi kaynak kodun versiyon kontrolü denilince aklımıza aşağıdakinden daha iyi bir çözüm geliyordur diye var sayıyorum)*

Başlangıç

İlk olarak uygulamaya **Microsoft.TeamFoundation.Client**, **Microsoft.TeamFoundation. Common** ve **Microsoft.TeamFoundation. VersionControl.Client assembly** dosyalarının referans ederek başlayabiliriz. Örneğimize konu olan **VersionControlServer** tahmin edileceği üzere**Microsoft.TeamFoundation.VersionControl.Client assembly**' ının bir parçasıdır.

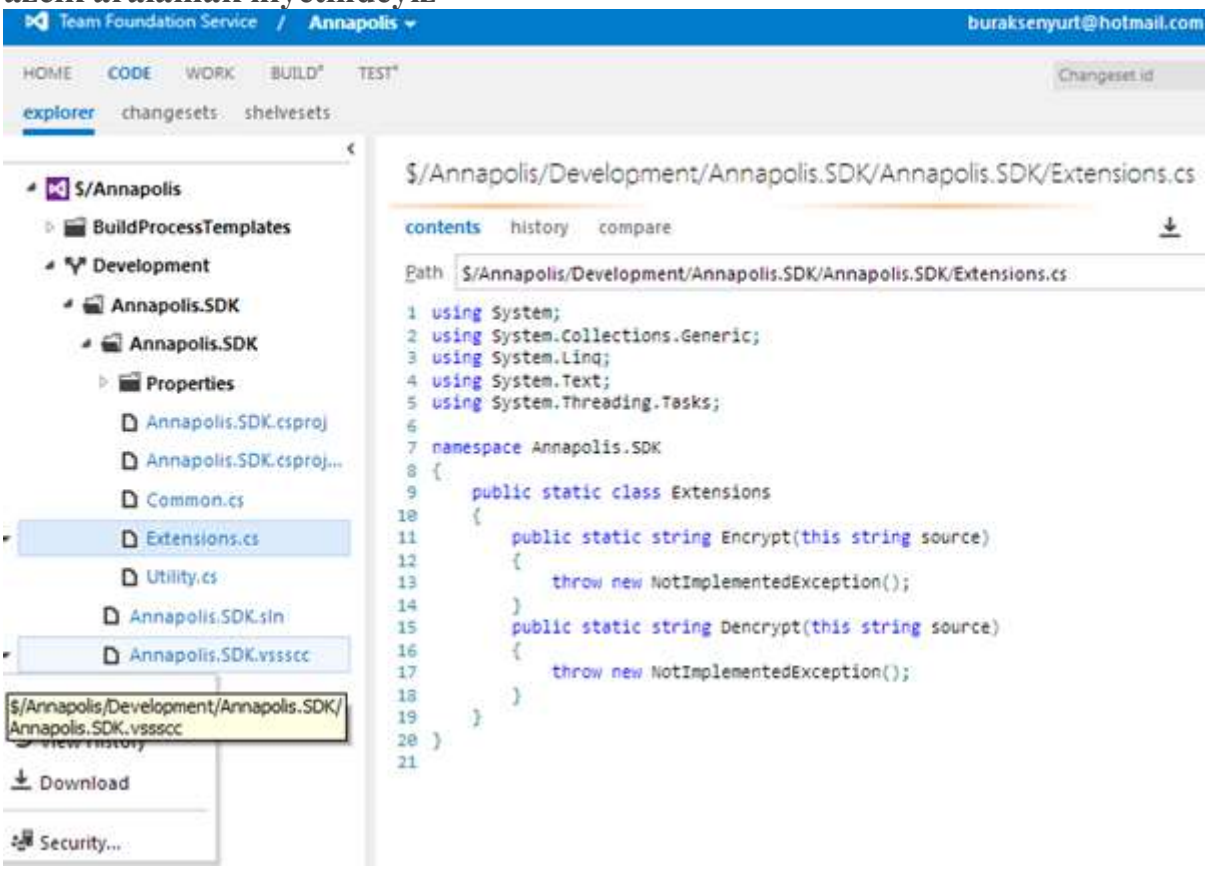


Window uygulamasına ait **Form** ise aşağıdaki gibi tasarlanabilir.



Kullanıcılar **app.config** dosyasında belirtilen **TFS** sunucusuna bağlanabilecektir.
Bu **TFS** sunucusu çok doğal olarak kendi içerisinde **n sayıda Team Project Collection barındırabilir**. İlgili **TeamProject Collection**' a ait bazı bilgilerin **Collections** bileşeni yanındaki **ComboBox** kontrolüne doldurulması sağlanmalıdır. Kullanıcı, bir **Team Project Collection** seçimi yaptığında ise, buna bağlı **Team Project** listesinin de ilgili **ComboBox** kontrolüne eklenmesi gerekmektedir. En azından bağlı olan **Team Project** adlarının listelenmesi yeterli olacaktır. Kullanıcının yapacağı bir diğer seçim de, **source control** üzerinden çekilmek istenen dosyaların tipleridir. Örneğin **C# , VB** gibi kod dosyaları olabileceği gibi, **XAML** içerikli dosyalar da göz önüne alınabilir. Örnekte kısıtlı bir küme kullanılmıştır ancak bu genişletilebilir.
Kendi çalışmalarınızı yaparken Team Foundation Server' ın tfs.visualstudio.com adresinden sunulan hizmetini göz önüne almanızı ve o ortamda sunulan Code penceresini incelemenizi öneririm. Code penceresine gelindiğinde aslında bu, bir Team Project Collection' daki bir Team Project içerisindeyiz anlamına gelir.

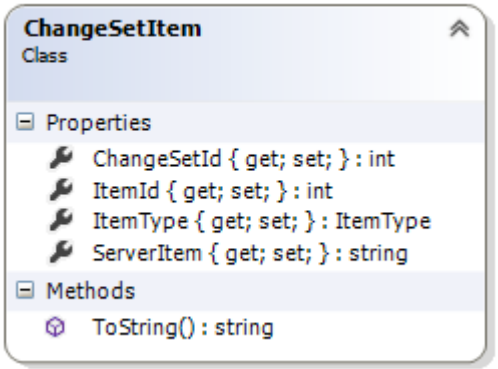
Dolayısıyla bu Team Project içerisindeyken source control üzerine atılan ne kadar içerik varsa görülebilir. Biz çok daha kısıtlı bir örneğini yapıyoruz. Aslında kapıyı azcık aralamak niyetindeyiz

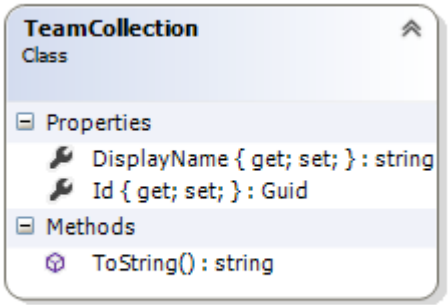


Dosya tipi seçimi de belli olduğunda aşağıdaki listeye, söz konusu uzantıya sahip dosyaların bazı bilgileri gelecektir. Eğer herhangibir dosya seçilirse de, bu dosyanın içeriği gösterilecektir.

Yardımcı Sınıflar

Örnekte işleri biraz olsun kolaylaştırmak adına iki yardımcı **POCO*(PlainOldClrObject)*** tipi kullanılmıştır. Bu tipler içerisinde bir **Team Project Collection**' ın ve **Change Set**' in temel bilgileri tutulmaktadır.





TeamCollection sınıfı içeriği;

```
using System;
namespace HowTo_TFSVersionControl
{
   public class TeamCollection
   {
      public string DisplayName { get; set; }
      public Guid Id { get; set; }
      public override string ToString()
      {
         return string.Format("[{0}]-{1}", Id.ToString(), DisplayName);
      }
   }
}
```

ve **ChangeSetItem** sınıfı içeriği;

```
using Microsoft.TeamFoundation.VersionControl.Client;
namespace HowTo_TFSVersionControl
{
   public class ChangeSetItem
   {
      public ItemType ItemType { get; set; }
      public string ServerItem { get; set; }
      public int ItemId { get; set; }
      public int ChangeSetId { get; set; }
      public override string ToString()
      {
         return string.Format("[{0}]-[{1}]-{2}"
            , ChangeSetId.ToString()
            , ItemId.ToString()
            , ServerItem);
      }
   }
}
```

Kod Tarafı

Dilerseniz öncelikle **Form1**' e ait kod içeriklerini üretelim ve neler yaptığımızı açıklamaya çalışalım.

```
using Microsoft.TeamFoundation.Client;
using Microsoft.TeamFoundation.Framework.Client;
using Microsoft.TeamFoundation.Framework.Common;
using Microsoft.TeamFoundation.VersionControl.Client;
```

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.IO;
using System.Windows.Forms;
namespace HowTo_TFSVersionControl
{
   public partial class Form1
      : Form
   {
      #region Common Fields
      TfsTeamProjectCollection tfs = null;
      VersionControlServer vcServer = null;
      ItemSet itemSet = null;
      List<ChangeSetItem> csitems = null;
      IReadOnlyCollection<CatalogNode> teamCollectionNodes = null;
      TfsTeamProjectCollection selectedCollection = null;
      #endregion Common Fields
      public Form1()
      {
         InitializeComponent();
         cmbFileTypes.Items.Add("cs|C# Files");
         cmbFileTypes.Items.Add("vb|Visual Basic Files");
         cmbFileTypes.Items.Add("xaml|XAML Files");

         txtTFSAddress.Text = ConfigurationManager.AppSettings["TfsAddress"];
         FillTeamCollections();
      }
      private void FillTeamCollections()
      {
         tfs = new TfsTeamProjectCollection(new Uri(txtTFSAddress.Text));
         teamCollectionNodes=tfs.ConfigurationServer.CatalogNode.QueryChildren(
            new[] {CatalogResourceTypes.ProjectCollection },
            false,
            CatalogQueryOptions.None);
         foreach (var teamCollectionNode in teamCollectionNodes)
         {
            TeamCollection teamCollection = new TeamCollection();
            teamCollection.DisplayName= teamCollectionNode.Resource.DisplayName;
            teamCollection.Id
=Guid.Parse(teamCollectionNode.Resource.Properties["InstanceId"]);
```

```
            cmbTeamCollections.Items.Add(teamCollection);
        }
    }
    private void lstItems_SelectedIndexChanged(object sender, EventArgs e)
    {
        ChangeSetItem selectedChangeSetItem=lstItems.SelectedItem as
ChangeSetItem;
        Item item=vcServer.GetItem(selectedChangeSetItem.ItemId,
selectedChangeSetItem.ChangeSetId);
        using(Stream stream = item.DownloadFile())
        {
            using(StreamReader reader = new StreamReader(stream))
            {
                txtItemContent.Text=reader.ReadToEnd();
            }
        }
    }
    private void cmbTeamCollections_SelectedIndexChanged(object sender, EventArgs
e)
    {
        cmbTeamProjects.Items.Clear();
        lstItems.Items.Clear();
        TeamCollection selectedTeamCollection=cmbTeamCollections.SelectedItem as
TeamCollection;
        selectedCollection=tfs.ConfigurationServer.
GetTeamProjectCollection(selectedTeamCollection.Id);
        var teamProjectNodes=selectedCollection.CatalogNode.QueryChildren(
            new[] { CatalogResourceTypes.TeamProject },
            false, CatalogQueryOptions.None);

        foreach (var teamProjectNode in teamProjectNodes)
        {
            cmbTeamProjects.Items.Add(teamProjectNode.Resource.DisplayName);
        }
    }
    private void cmbTeamProjects_SelectedIndexChanged(object sender, EventArgs e)
    {
        lstItems.Items.Clear();
        txtItemContent.Clear();
        vcServer=selectedCollection.GetService<VersionControlServer>();
        string itemPath = string.Format("$/{0}/*.{1}"
```

```
            , cmbTeamProjects.SelectedItem
            ,cmbFileTypes.SelectedItem.ToString().Split('|')[0]);
          itemSet = vcServer.GetItems(itemPath, RecursionType.Full);
         csitems = new List<ChangeSetItem>();
          foreach (Item item in itemSet.Items)
          {
             ChangeSetItem csi = new ChangeSetItem();
             csi.ItemId = item.ItemId;
             csi.ChangeSetId = item.ChangesetId;
             csi.ItemType = item.ItemType;
             csi.ServerItem = item.ServerItem;
             lstItems.Items.Add(csi);
          }
          lblStatus.Text = itemSet.Items.Length.ToString();
       }
    }
}
```

TFS üzerindeki **Collection** nesnelerini elde etmek için **TfsTeamProjectCollection** örneği üzerinden önce **ConfigurationServer**, ardından da **CatalogNode** özelliğine gidilmekte ve **ChildNode** içerikleri sorgulanmaktadır. Bu sorgu içerisinde **CatalogResourceTypes enum** sabitinin**ProjectCollection** değeri kullanıldığından, **TFS** sunucusu üzerindeki **Team Project Collection**' ların içerikleri elde edilecektir.

Bir **Team Project Collection** sorgulanırken **Guid** tipinden olan **ID** değeri önem arz eder. Bu yüzden sorgu sonucu elde edilen **Node**' lar arasında dolaşılırken **Resorce** özelliklerinden yararlanılarak iki değerin alınması sağlanmıştır. **DisplayName** ve **Properties["InstanceId"]** yardımıyla **TeamProject Collection'** ı benzersiz olarak niteyen **Guid** değeri. İlgili özelliklerin toplandığı**TeamCollection** nesne örnekleri de **cmbTeamCollections** isimli **ComboBox** kontrolünün **Items**koleksiyonuna eklenmektedir.

Kullanıcı bu **ComboBox** bileşeninden bir öğe seçtiğinde ise, ilgili **Team Project Collection**altındaki **Team Project** örneklerinin, en azından adlarının çekilmesi gerekmektedir. Nitekim bu**Team Project** adı ilgili **Version Control** servisi tarafından, **Change Set** öğelerinin çekilmesi sırasında ele alınacaktır.

Seçilen öğenin karşılığı olan **TfsTeamProjectCollection** nesnesine ulaşmak için**ConfigurationServer** örneğine ait **GetTeamProjectCollection** metoduna başvurulur. Bu metod parametre olarak seçili öğenin **Id** değerini alır. Tahmin edeceğiniz üzere bu değeri, **ComboBox**içerisinde **TeamCollection** nesne örneklerinde birer özellik olarak kullanmıştık. Seçilen öğenin karşılığı olan **TfsTeamProjectCollection** yakalandıktan sonra ise **CatalogNode** üzerinden tekrar**Child Node**' lara gidilecek şekilde bir çağrı yapılır.

Lakin bu kez **CalatogResourceTypes enum** sabitinin **TeamProject** değeri kullanılır. Buna göre seçili olan **Team Project Collection**'a bağlı olan **Team Project** listesi, birer **CatalogNode** olarak elde edilebilir. Sonrasında ise liste dönülür ve her bir **Team Project**' in **Resource** özelliği üzerinden yakalanacak **DisplayName** değeri, **cmbTeamProjects** bileşeninin **Items** koleksiyonuna eklenir.

Yazımıza konu olan **Verison Control** servisi ise bu noktadan sonra devreye girecektir. İlgili servis örneği kullanıcının yaptığı **Team Project** seçimi sonrasında önemlidir. Nitekim Change Set' lerin ver içeriklerinin elde edilmesi için ilgili operasyon desteğini sunmaktadır.

İlk olarak güncel **Team Project Collection** tespit edilerek generic **GetService** metoduna bir çağrıda bulunulur ve **VersionControlServer** tipinden bir referans alınır. Bu referansın **GetItems**metodu kilit noktamızdır. İlk parametre ile bir **string** verilmektedir ama yazım şekli özeldir.

$/{0}/*.{1}

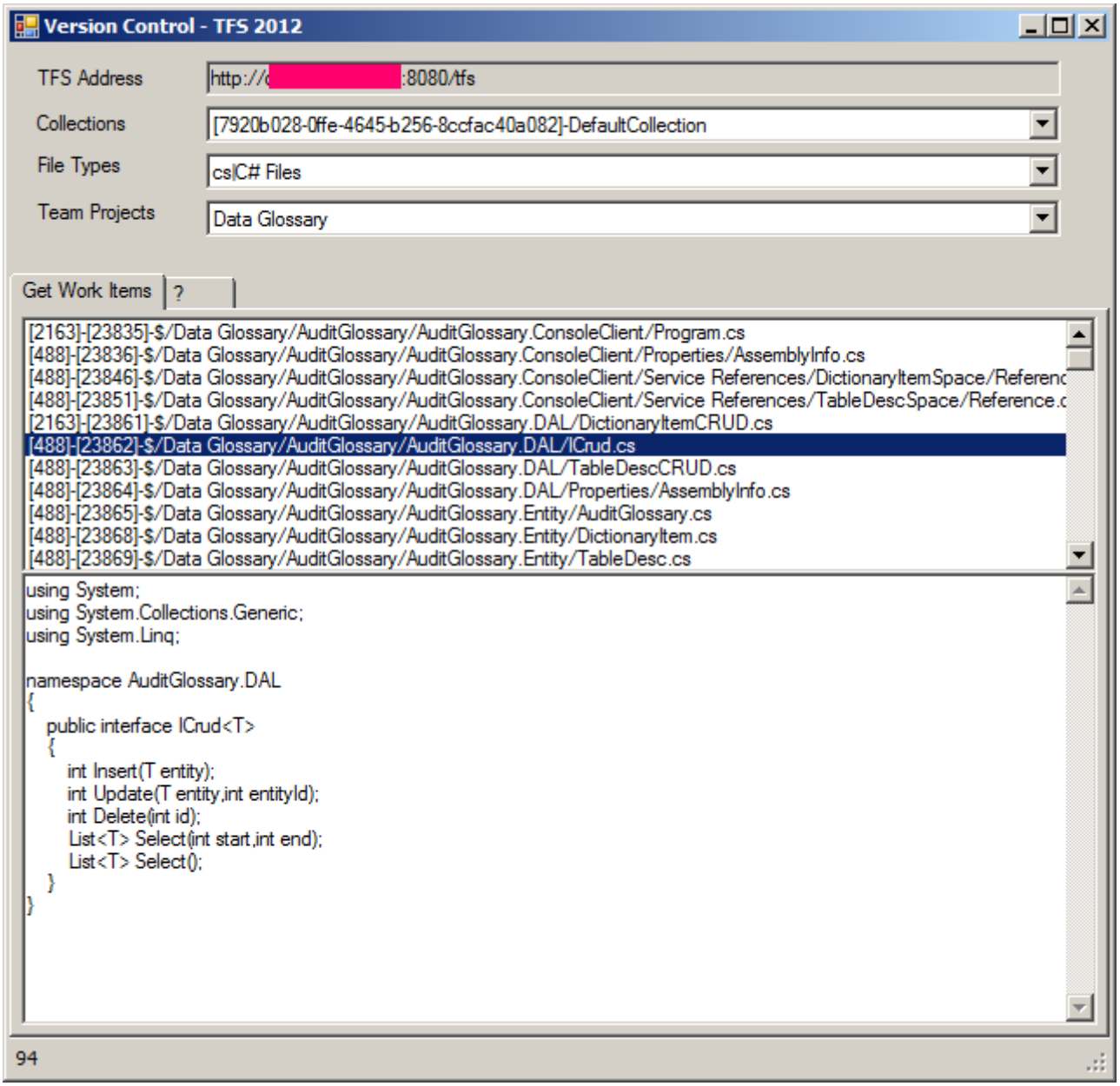
ifadesinde **{0}** yerine **Team Project** adı, **{1}** yerine ise talep edilen dosya formatının uzantısı gelir.**GetItems** metodu esas itibariyle bir **ItemSet** örneği döndürür ve aslında beklediğimiz dosyalar bu tipin **Items** koleksiyonunca alınır. Örnekte **Items** özelliği dolaşılmakta olup her biri için bir**ChangeSetItem** örneklenir. **ChangeSetItem** sınıfında basit bir kaç temel özellik bulunmaktadır.**ItemId**, **ChangeSetId**, **ItemType** ve **ServerItem**.

Bu aşamadan sonra kullanıcının bir **ChangeSetItem'** ı listeden seçmesi halinde içeriğinin gösterilmesi adımına gelinir. Bu sefer de **VersionControlServer** örneğine ve **GetItem** isimli metoduna başvurulur. Örnekte **GetItem** metoduna iki parametre girmektedir. **Item** ve **Change Set**numaraları. Çok doğal olarak bir **Change Set** altında farklı numaralara sahip birden fazla öğe yer alabilir. Bu sebepten hangi **Change Set**'teki hangi öğeyi alacağımızı bildirmemiz gerekir.

Elde edilen **Item** öğesinin **DownloadFile** isimli bir fonksiyonu da bulunmaktadır. Bu metod sayesinde, sunucudaki içeriğin **Stream** olarak elde edilmesi mümkündür. Örnek kod parçasında**StreamReader** sınıfından yararlanılmış ve ilgili sunucu içeriğinin **txtItemContent** isimli **TextBox**kontrolüne basılması sağlanmıştır.

Çalışma Zamanı Sonuçları

Örneği çalıştırdığımızda ve sırasıyla **Team Project Collection**, **File Type** ve **Team Project**seçimlerine yaptığımızda, aşağıdakine benzer sonuçlar ile karşılaşabiliriz.



Dikkat edileceği üzere **Default Collection** altındaki bir **Team Project**' in içerisinde yer alan **C#**kod dosyalarının temel bilgileri çekilebilmiştir. Bu bilgiler arasında, **Item** ve **Change Set** numarası ile **Server** üzerinde tutulan **Full Path** bilgisi yer almaktadır. Hatta istenirse son **Check-In** bilgisi bile alınabilir*(Belki de alınamaz. Neden araştırmıyorsunuz*  *)* İşin güzel yanı söz konusu öğelerden herhangibirisine tıklandığında, içeriğinin de görüldüğüdür.

Uygulamada Neleri Yapmadık?

Version Control hizmetini kullanarak, **Source Control** içeriğini çekelebilmek oldukça kolaydır. Bu felsefeden yola çıkarak kendi Source Control arabirimlerinizi yazabilirsiniz. Örneğin Windows Phone 8 tabanlı çalışacak bir istemci söz konusu olabilir. Pek tabi yapabileceğimiz daha pek çok şey var. Örneğin,

- Source Control' den çekilen bir içeriği güncelleyebilir ve sunucuya kaydedebiliriz.

- Yeni bir öğeyi Source Control'a ekleyebiliriz.
- Bir öğeyi silebiliriz.
- Klasöre veya Branch gibi öğeler oluşturabiliriz.

ve benzerleri…

Bu konuları örneğimizde ele almadık ancak **VersionControlServer** tipinin işaret ettiği servis metodlarını inceleyerek nasıl yapabileceğinizi araştırmaya başlayabilirsiniz. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

[TFS 2013 Kullanımında Farklılıklar Olabilir. Araştırmanız öneririm]

HowTo_VersionControl.zip (218,31 kb)

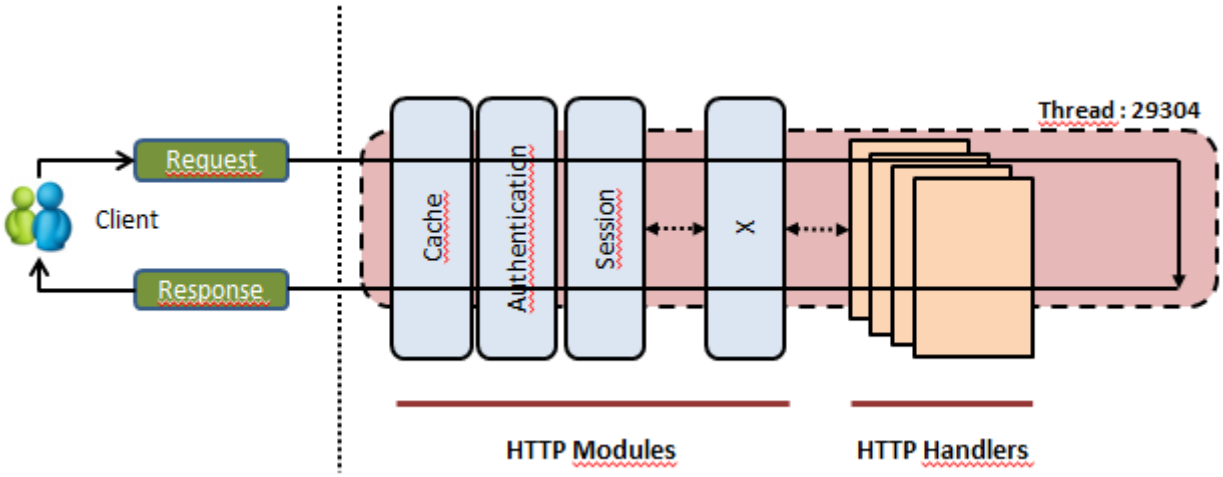
Asp.Net 4.5–Asenkron HTTP Module Geliştirmek

Pazartesi, 14 Ekim 2013 08:00 by bsenyurt

Merhaba Arkadaşlar,
Bir çoğunuz gibi ben de düzenli olarak bazı dergilerin abonesiyim ve her ay onları alıp biraz karıştırdıktan sonra arşive*(yani çalışma odasındaki kütüphaneye)* kaldırmaktayım.
Tabi gün oluyor pek çoğuna dönüp bakmıyorum bile. Hatta kimisinin rengi sararıp soluyor bir köşede mazlum mazlum kalıyor. Ama eminim ki geride kalanlar için bazıları güncel içeriklere sahip iken, bazıları da tam anlamıyla bir Retro havası veriyor. Ve hatta çoğu, yıllar geçtikçe daha fazla değer kazanıyor.
Söz gelimi yandaki basılı medya görselinde 1974 model bir Pioneer ses sisteminin reklamı yer almakta. Bu pek çoğumuzun daha henüz hayatta bile olmadığı, olsa da pek hatırlamadığı bir yıl belki de? Sanıyorum blog tutan bizlerin yazıları da, zaman içerisinde bu şekilde eskiyor. Hele de yazılım teknolojileri gibi çok çabuk gelişen konular ele alınıyorsa.
Ben bazen geçmişte yazmış olduğum yazılara bakıyorum ve ne kadar da çabuk eskidiklerini görüyorum. Ancak bir yandan da, "acaba yeni sürümde bu konuda neler yapılmış?" sorusunun cevabını da kurcalamaya çalışıyorum.
Nostalji : 2006 dan bir makale. O zamanlar Asp.Net Pipeline' ın önemli parçaları olan HttpModule ve HttpHandler kavramlarını incelemeye çalışmıştım.
İşte bu günkü konumuzda **HttpModule** tipleri içerisindeki işlemleri asenkron olarak nasıl yaptırabileceğimizi incelemeye çalışıyor olacağız. Bildiğiniz üzere **.Net Framework 4.0** ile hayatımıza giren **Task** ve doğal olarak **Task Parallel Library** kavramı, **4.5** sürümünde gelen **async**ve **await** anahtar kelimeleri ile birlikte alt yapının pek çok noktasında daha sık görülmeye başladı. Bu açıdan bakıldığında **Asp.Net 4.5** tarafında da ilgili anahtar kelime ve **Task** tiplerini kullanarak bazı senkronize edilmiş işlemlerin asenkron hale getirilmesi sağlanabilmekte.
Asp.Net ve asenkron çalışma diyince bir duraksayıp düşünmek gerekir. Asp.Net aslında client-server modelini baz alarak çalışır ve stateless bir ortam sunar. Bu yüzden istemciden gelen her talep(Request) sonrası, uygulama ve talep gören sayfanın Lifecycle' ının tekrar işletilmesi gerekir. Web tarafında asenkronluk diyince bunu belki de iki taraflı olarak düşünmek gerekebilir. Sunucu tarafında çalışan asenkron işlemler ve istemcinin kendi açısından asenkron olarak başlatıp, Response' u beklemeye gerek kalmadan başka talepleri de gönderebildiği işlemler*(Genelde AJAX tarzı çağrılar ile hallettiğimiz çalışma şekli)*

Module ve Handler gibi tipler sunucu tarafında çalışan yapılar olduklarından, yazıda bahsedeceğimiz asenkron çalışma aslında, sunucu üzerinde söz konusu olan ve LifeCycle yaşamı boyunca ele alınan yapılar olarak düşünülmelidirler.
Bu işten nasibini alan kısımlardan ikisi de **Asp.Net** uygulamalarının yaşam döngüsünde önemli yere sahip olan **Module** ve **Handler'** lardır. Aslında bir Web sayfasını talep ettiğimizde, sunucu tarafında gerçekleşen bir dizi işlem söz konusudur. Ayrıca yaşam döngüsüne ait süreçte devreye giren bir kanal yapısı mevcuttur. Bu kanal yapısına göre istemciden gelen **Request'** in önce **Http** bazlı**Module'** lerden geçmesi ve ardından da **Handler'** lar tarafından değerlendirilmesi söz konusudur. Klasik çalışma modeline baktığımızda aşağıdaki grafikte yer alan ve özetle ifade edilmeye çalışılan işleyişin gerçekleştiğini söyleyebiliriz.

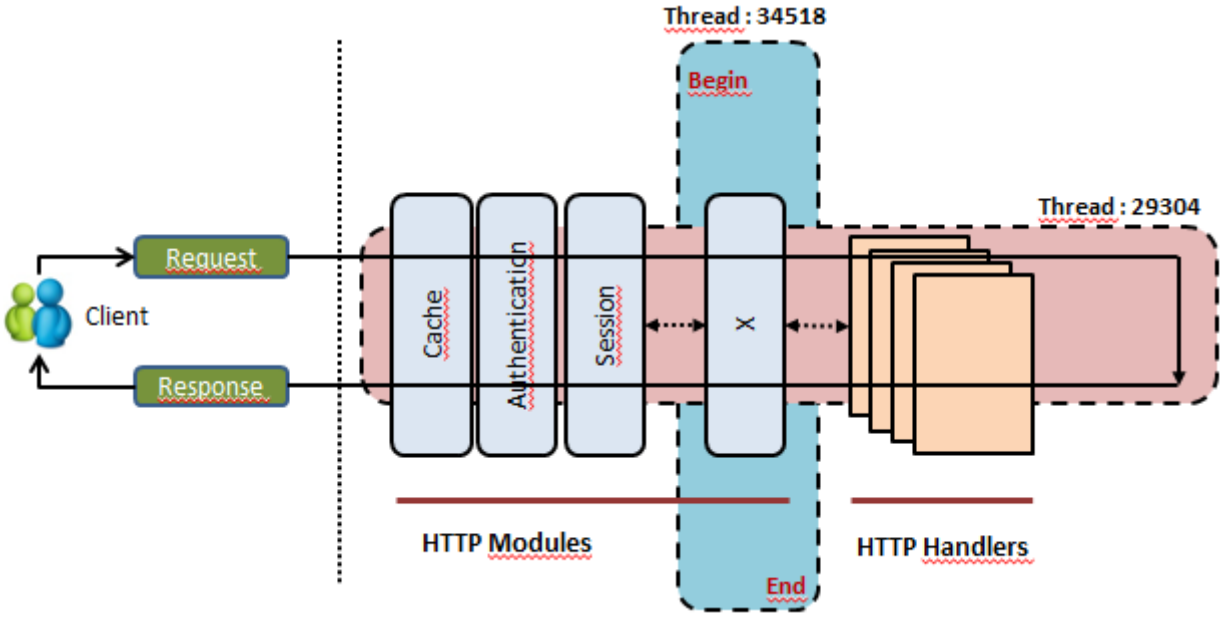
Klasik Senkron Çalışma Modeli



Görüldüğü gibi gelen talep, **Built-In** bazı **Http Module'** lerinden geçmekte ve **Handler** seviyesinde de değerlendirildikten sonra geriye döndürülmektedir. *(Bu çizelgede talep edilen içeriğe ait sayfa veya user control' un iç çalışma şekli göz ardı edilmiştir)* Dikkat edilmesi gereken husus, **Module** ve**Handler'** lardan oluşan bu kanal yapısının **Thread** havuzundan çekilen tek bir **Thread** içerisinde uçtan uca çalışıyor olmasıdır. Dolayısıyla bir **module'** den diğerine olan geçiş sırasında, işlemekte olan module' ün işini tamamlamış olması gerekmektedir. Aynı durum doğal olarak **Handler'** lar için de geçerlidir.

Asenkron Çalışma Modeli

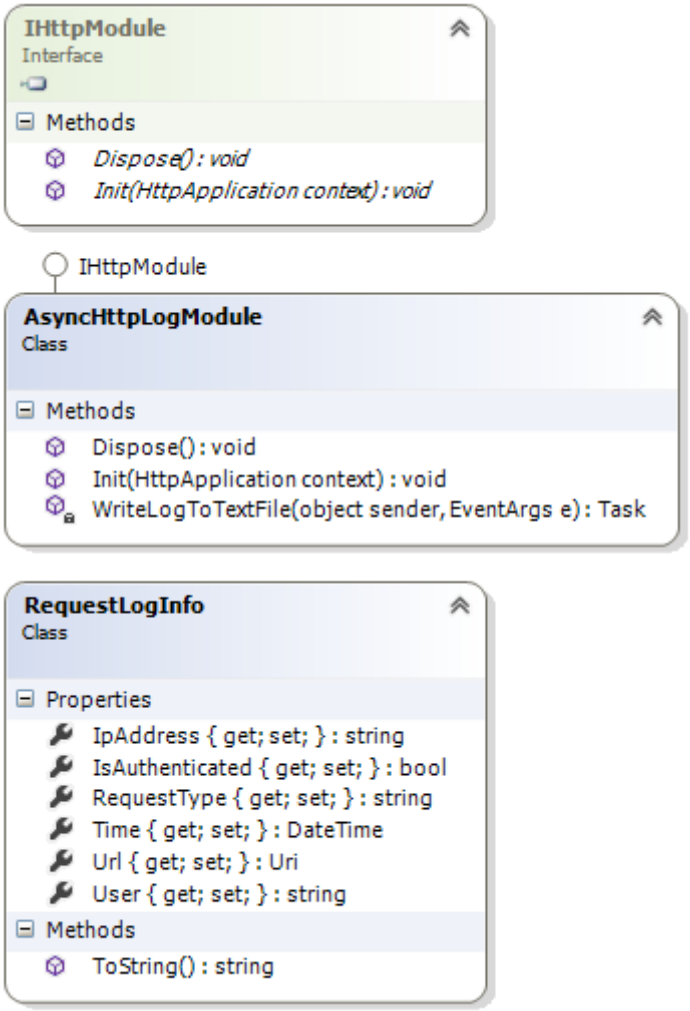
Asenkron çalışma modeline göre ise, bir **Module** veya **Handler'** ın kendi çalışmasını farklı bir**Thread'** e yıkması ve daha sonra ana **Thread'** e sonuç dönerek işleyişi uzun süre duraksatmamasıhedeflenmektedir. Bu, özellikle geliştirici tarafından yazılan **Module** veya **Handler** tipleri için kullanılabilecek bir stratejidir. Aşağıdaki grafikte bu yaklaşım modeli özetlenmeye çalışılmıştır.



Dikkat edileceği üzere **X Module** tipi kendi işleyişini farklı bir **Thread** altında yapmaktadır. Dolayısıyla işlenmekte olan talebin akışı sırasında **X** modülüne gelindikten sonra, sıradaki **Module**veya takip eden **Handler'** lara geçilmesi için bir duraksama söz konusu değildir. İlgili **Module**çalışmasını tamamladığında, kanal içerisindeki işleyişe otomatikman dahil olacaktır.

Asp.Net 4.5 için Örnek Uygulama

Peki bu modeli **Asp.Net 4.5** içerisinde nasıl gerçekleştirebiliriz? Gelin basit ve klasik bir örnek üzerinden ilerleyelim. Bu amaçla **Visual Studio 2012** ortamında **Asp.Net Empty Web Application** tipinden bir proje açalım. Bir **Web** uygulamasında veya bir **Application Pool** içerisinde kendi **Http Module'** lerimizi kullanabilmenin yolu **IHttpModule** arayüzünden türeyen sınıflar geliştirmekten geçmektedir. **IHttpModule** kendi içerisinde **Asenkron** kullanım için doğal bir metod içermemektedir. Bunun yerine ezilmesi gereken **Init** ve **Dispose** metodlarını sunmaktadır. **Init** fonksiyonu tahmin edileceği üzere **Module** devreye girdiğinde yapılacak işlemleri içermekte ve icra ettirmektedir. Biz burada asenkron işleyişlere yer verebiliriz. Nasıl mı? İşte uygulama içerisindeki sınıf çizelgesi ve kod yapısı.



RequestLogInfo POCO tipinin içeriği;

```
using System;
namespace HowTo_AsyncModules
{
   public class RequestLogInfo
   {
      public string IpAddress { get; set; }
      public DateTime Time { get; set; }
      public bool IsAuthenticated { get; set; }
      public string User { get; set; }
      public string RequestType { get; set; }
      public Uri Url { get; set; }
      public override string ToString()
      {
         return string.Format("{0}|{1}|{2}|{3}|{4}|{5}"
            ,Time.ToLongTimeString()
            ,IpAddress
```

```
            ,IsAuthenticated.ToString()
            ,User
            ,RequestType
            ,Url.ToString()
            );
      }
   }
}
```

RequestLogInfo örnek senaryoda kullanacağımız ve içerisinde istemciden gelen **talep*(Request)***ile ilişkili bir kaç basit bilgiyi içeren **POCO*(Plain Old CLR Object)*** sınıfımızdır. Gelelim**AsyncHttpLogModule** sınıfının içeriğine.

```
using System;
using System.Configuration;
using System.IO;
using System.Threading.Tasks;
using System.Web;
namespace HowTo_AsyncModules
{
   public class AsyncHttpLogModule
      :IHttpModule
   {
      public void Dispose()
      {
       // Do Something
      }
      public void Init(HttpApplication context)
      {
         EventHandlerTaskAsyncHelper eventHandler =new
EventHandlerTaskAsyncHelper(WriteLogToTextFile);
         context.AddOnPostAuthorizeRequestAsync(
            eventHandler.BeginEventHandler
            , eventHandler.EndEventHandler
            );
      }
      private async Task WriteLogToTextFile(object sender, EventArgs e)
      {
         var context = HttpContext.Current;
         string filePath =
```

```
HttpContext.Current.Server.MapPath(ConfigurationManager.AppSettings["LogFileAddres
s"]);
            RequestLogInfo requestLog = new RequestLogInfo()
            {
               Time = DateTime.Now,
               IpAddress = context.Request["REMOTE_ADDR"],
               User = context.User.Identity.Name,
               IsAuthenticated = context.User.Identity.IsAuthenticated,
               RequestType = context.Request.RequestType,
               Url = context.Request.Url
            };
            using (StreamWriter streamWriter = File.AppendText(filePath))
            {
               await streamWriter.WriteLineAsync(requestLog.ToString());
            }
         }
      }
}
```

Örnek kod parçasında dikkat edileceği üzere **Init** metodu içerisinde gerçekleştirilen olay bazlı bir asenkron çalıştırma kayıt bildirimi söz konusudur. **Init** metodu içerisinde kullanılan**EventHandlerTaskAsyncHelper** sınıfı, **TaskEventHandler temsilci(delegate)** tipinden bir parametre almaktadır.

Aslında bu temsilci geriye **Task** örneği döndüren ve **object** ile **EventArgs** tipinden parametre alan metodları işaret edebilir ki örneğimizde bu fonksiyon **WriteLogToTextFile**' dır. Bu metod **async**anahtar kelimesi ile işaretlendiğinden kendi içerisinde **awaitable** operasyonları da içerebilmektedir. Örnekte bunu sembolize etmek için **StreamWriter** tipinin asenkron çalışabilen **WriteLineAsync**metodu kullanılmıştır.

Çok doğal olarak bir Module her zaman asenkron çalışacak şekildetasarlanmamalıdır. Nitekim içerisindeki işlemlerin sonucuna göre sonraki Module' lere bilgi taşınması veya çalışmanın kesilerek anında bir Exception döndürülmesi vb durumlar söz konusu olabilir.

İlgili asenkron işleyişlerde **text** tabanlı olarak fiziki bir dosyaya **log** atma işlemi sembolize edilmiştir. Pek tabi bunun yerine **log** içeriğinin veritabanına aktarılması, bir servise çağrıda bulunulması, harici 3ncü parti bir arayüz ile konuşulması*(Örneğin bir Bussines Process Management sistemine mesaj gönderilmesi)* ve benzeri zaman alabilecek ve dolayısıyla asenkron hale getirilebilecek işlemler ele alınabilir. Uygulamayı teste tabi tutmadan

önce **Web.config** dosyası içerisinde, tasarlanmış olan yeni **HttpModule** tipine ait bildirimin de yapılması gerekmektedir ki çalışma zamanı var olan **Module'** lere ek olarak **AsyncHttpLogModule** örneğini de devreye alabilsin. Bunun için **system.webServer** elementi içerisinde aşağıdaki **module** tanımlaması yapılmalıdır.

Config Ayarlarmaları

```
<?xml version="1.0"?>
<configuration>
   <system.web>
     <compilation debug="true" targetFramework="4.5" />
     <httpRuntime targetFramework="4.5" />
   </system.web>
   <system.webServer>
     <modules>
       <add name="AsyncLogModule"
type="HowTo_AsyncModules.AsyncHttpLogModule,HowTo_AsyncModules"/>
     </modules>
  </system.webServer>
  <appSettings>
    <add key="LogFileAddress" value="~/App_Data/Logs.txt"/>
  </appSettings>
</configuration>
```

modules elementi içerisinde dikkat edilmesi gereken nokta **type** niteliğine atanan değerdir. Burada**[Namespace Adı].[Module Tip Ad],[Assembly Adı]** notasyonundan yararlanılmıştır. Bu, özellikle ilgili **Module** tipinin harici bir **Class Library** içerisinde olduğu durumlarda ön plana çıkan önemli kurallardan birisidir.

Test

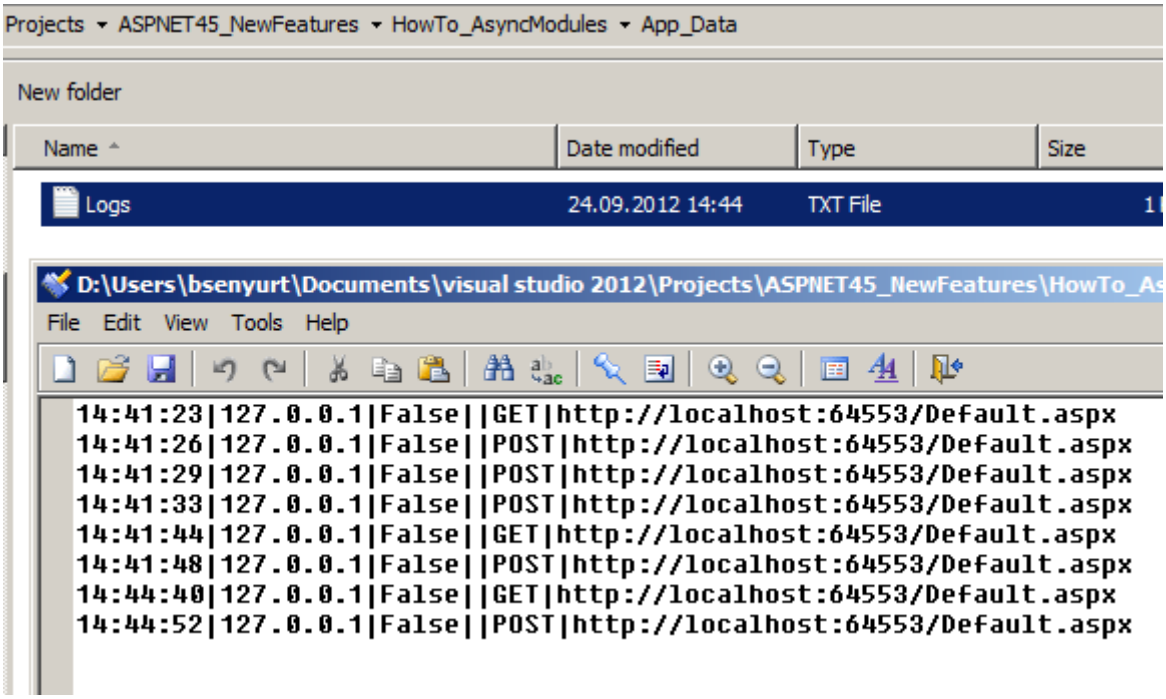
Test için dilerseniz çok basit bir de **aspx** sayfası hazırlayalım. Böylece basit anlamda **Get**, **Post** gibi aksiyonları simüle ettirmiş oluruz.

Örnek aspx içeriği;

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="HowTo_AsyncModules.Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
```

```
    <form id="form1" runat="server">
    <div>
    Mesajınız : <asp:TextBox ID="txtMessage" runat="server" /><br />
        <asp:Button ID="btnSendMessage" runat="server" Text="Send My Message"
OnClick="btnSendMessage_Click"/>
    </div>
    </form>
</body>
</html>
```

WebForm içerisinde bir **TextBox** ve **Button** kontrolü bulunmaktadır. Kullanıcı sayfayı ilk kez talep ettiğinde **HTTP Get** ve **Button'** a her bastığında da **HTTP Post** tipinden talepler üretilmesine neden olmaktadır. Çalışma zamanında yapılan çeşitli aksiyonlar sonrası oluşan örnek bir **log** içeriği ise aşağıda görüldüğü gibidir.



Dikkat edileceği üzere geliştirici tanımlı bir **HttpModule** tipinin işleyişinin asenkron hale getirilmesi mümkündür. Aynı prensiplerden yola çıkarak bir **HttpHandler** tipinin de asenkron çalışacak hale getirilmesi söz konusu olabilir elbette. Lakin **Asp.Net 4.5, HttpHandler** tiplerinin asenkron yazılabilmesi için daha güçlü bir yol sunmaktadır. Bu yolun başında **HttpTaskAsyncHandler** isimli**soyut sınıf*(Abstract Class)*** yer almaktadır. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HowTo_AsyncModules.zip (25,83 kb)

Dosya Satır Sayısını Bulmak

Pazartesi, 7 Ekim 2013 09:02 by bsenyurt

Merhaba Arkadaşlar,
Eğer sizde zamanında benim gibi bankaların teknoloji departmanlarında çalışmış ve yazılım geliştirmişseniz, eminimki hayatınızın bir döneminde büyük boyutlu **Text**dosyaları ile çalışmak zorunda kalmışsınızdır.

Malum Bankaların sistemleri halen daha eski olabildiğinden, bölümler arası veya uygulamalar arası veri aktarmanın en popüler yollarından birisi olarak **Text** tabanlı dosya formatları göz önüne alınmaktadır. Bazen onlarca **megabyte'** ı aşan ve milyonlarca satırdan oluşabilen düzenli text dosyaları söz konusu olur ve bunların bir şekilde uygulamaların konuştuğu veritabanı ortamlarına işlenerek, ilişkisel veri bütünlüğü içerisinde yerlerini alması beklenir.

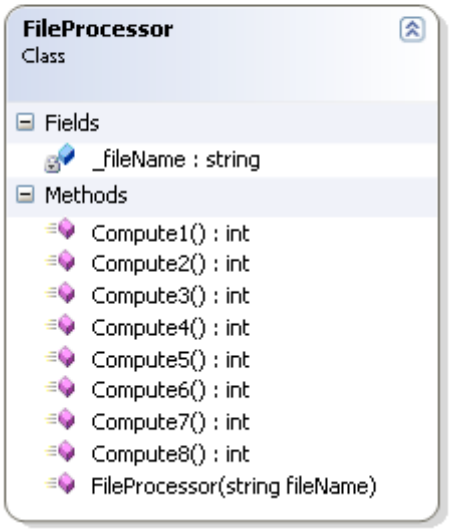
Vaka

Aktarım işlemleri sırasında çoğunlukla **SSIS(Sql Server Integration Services)** paketlerinden yararlanılmaktadır veya ekstra 3ncü parti araçlar kullanılır. Lakin bazen kendi uygulamalarımız içerisinde bu tip dosyaların kod yardımıyla ayrıştırılması ve işlenmesi de gerekebilir. Böyle bir halde ise son kullanıcının durumdan haberdar edilmesi ve özellikle işlem çok uzun sürecekse bir **Progress Bar** bileşeni ile*(En azından Windows Forms tarafı için)* anlık ilerleme durumunun gösterilmesi uygun olabilir. Tabi anlık durumun bir dosya için gösterilmesi söz konusu ise, dosyanın toplam satır sayısının da bilinmesi gerekecektir.

Hımmm…Bir dosyanın toplam satır sayısı nasıl bulunabilir peki? Bunun için aklımıza gelecek ve uygulayabileceğimiz bir çok yol bulunmakta. Ancak hangisinin daha efektif olduğunu bir şekilde tespit etmemiz ve görmemiz önemlidir. Dolayısıyla bu yazımızda, bir text dosyanın satır sayısını bulmak için kullanabileceğimiz metodlardan bazılarını ve bu fonksiyonların toplam çalışma sürelerini hesaplatacağız. Tabi alacağımız sonuçlar sistemden sisteme farklılık gösterebilirler.

Hazırlıklar

İlk olarak operasyonel işlemleri üstlenen tipimizi ve test kodumuzu geliştirip üzerinde kısaca konuşalım. Uygulamamıza ait sınıf diagramı ve kod içeriği aşağıdaki gibidir.




```
using System;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Reflection;
using System.Text.RegularExpressions;
namespace FindLineCountsApp
{
   class Program
   {
      static void Main(string[] args)
      {
         // Satır sayısı hesap edilecek olan dosya adresi alınır
         string file = Path.Combine(Environment.CurrentDirectory, "Content.txt");
         // Hesaplama işlemlerini üstlenen tipimiz
        FileProcessor prcsr=new FileProcessor(file);
         // Reflection' dan yararlanarak, FileProcessor tipi içerisinde LineTest niteliği ile
işaretlenmiş ne kadar metod varsa çağırıyor ve her birinin hesaplama sürelerini buluyoruz.
         Type t = prcsr.GetType();
         foreach (var method in t.GetMethods())
         {
           if (method.GetCustomAttributes(false).Length == 1)
            {
               var lAttr = (method.GetCustomAttributes(false)[0]) as
LineTestAttribute;
```

```
                if (lAttr != null)
                {
                    // Kronometremizi her metod çağrısı içerisinde oluşturuyoruz
                    Stopwatch watcher = new Stopwatch();
                   watcher.Start(); // Kronometre start
                    var result = method.Invoke(prcsr, BindingFlags.InvokeMethod, null,
null, null);
                       // Metod çağırılıyor
                    watcher.Stop(); // Kronometre stop
                    Console.WriteLine("Method {0}\tLineCount {1}\tDuration {2} milisaniye",
method.Name,  result.ToString(), watcher.ElapsedMilliseconds.ToString());
                }
            }
        }
    }
}
class FileProcessor
{
    private string _fileName;
    public FileProcessor(string fileName)
    {
        _fileName = fileName;
    }
    // Hesaplama metodlarımızdan birisi File tipinin ReadAllLines metodunu kullanarak
dönen dizinin uzunluğuna bakar
    [LineTest]
    public int Compute1()
    {
        int lineCount = 0;
        lineCount=File.ReadAllLines(_fileName).Length;
        return lineCount;
    }
    // Dosya baştan sona okunurken Alt Satıra geçme karakteri araştırılır '\n' bu -1
olmadığı sürece yani dosya sonuna gelinmediği sürece satır sayısı ve pozisyon arttırılarak
devam edilir.
   [LineTest]
    public int Compute2()
    {
        int lineCount = 0;
        int position = 0;
        while ((position = File.ReadAllText(_fileName).IndexOf('\n', position)) != -1)
```

```
        {
            lineCount++;
            position++;
        }
        return lineCount+1;
    }
    // RegEx ifadesinden yararlanılarak \n' lerin sayısı hesaplanır.
    [LineTest]
    public int Compute3()
    {
        int lineCount = 0;
        Regex regEx = new Regex("\n", RegexOptions.Multiline);
        MatchCollection matchCollection = regEx.Matches(File.ReadAllText(_fileName));
        lineCount=matchCollection.Count;
        return lineCount+1;
    }
    // Bu sefer \n karakterlerinin tespiti için Linq sorgusundan yararlanılmaktadır.
    [LineTest]
    public int Compute4()
    {
        int lineCount = 0;
        lineCount=(from ch in File.ReadAllText(_fileName)
         where ch == '\n'
         select ch).Count();
        return lineCount+1;
    }
    // Bu kez dosya içeriği bir char dizisine atanır ve tüm dizide dönülerek \n' lerin toplam
sayısı bulunur
    [LineTest]
    public int Compute5()
    {
        int lineCount = 0;
        string text = File.ReadAllText(_fileName);
        int posMax = text.Length;
        char[] a = text.ToCharArray();
        for (int position = 0; position < posMax; )
            if (a[position++] == '\n')
                lineCount++;
        return lineCount+1;
    }
```

```
        // Bu seferki metodumuzda dosyanın uzunluğundan, dosya içerisinde new line karakterlerinin kaldırılması sonucu oluşan içeriğin uzunluğunu çıkartarak hesaplama yaptırıyoruz
        [LineTest]
        public int Compute6()
        {
            int lineCount = 0;
            string text = File.ReadAllText(_fileName);
            lineCount=text.Length - text.Replace("\n", "").Length;
            return lineCount+1;
        }
        // Count Extension Method' unun kullanılması ve new line' ların sayılarının bulunarak satır sayısının hesap edilmesi işlemini üstlenir
        [LineTest]
        public int Compute7()
        {
            int lineCount = 0;
            lineCount=File.ReadAllText(_fileName).Count(c => (c == '\n'));
            return lineCount+1;
        }
        // Bu sefer dosyanın her bir karekteri foreach döngüsü ile dolaşılıyor ve new line karakterlerinin sayısı hesap ediliyor
        [LineTest]
        public int Compute8()
        {
            int lineCount = 0;
            foreach (char c in File.ReadAllText(_fileName))
                if (c == '\n')
                    lineCount++;
            return lineCount+1;
        }
    }
    // Satır sayısını hesaplama ile ilişkili test metodlarımızı belirtmek amacıyla basit bir attribute tanımlıyoruz
    [AttributeUsage(AttributeTargets.Method)]
    class LineTestAttribute
        :Attribute
    {
    }
}
```

Kod Ne Yapıyor?

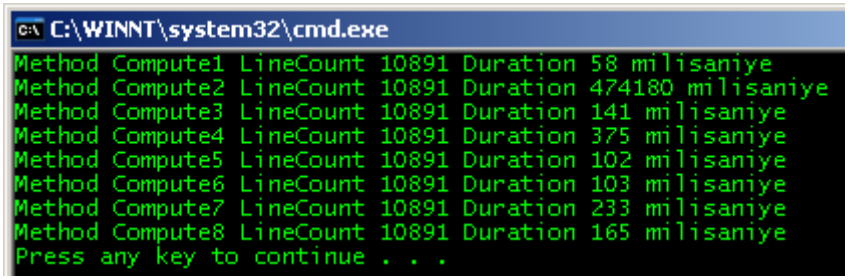
FileProcessor isimli sınıfımız **ComputeX** isimli 8 adet metod içermektedir. Her bir metod yorum satırlarında belirtildiği üzere, ilgili dosyanın satır sayısını bulmak için farklı bir yöntem kullanılmaktadır. **Constructor(Yapıcı Metod)** teste tabi tutulacak olan dosya adını alır.

Dikkat edilmesi gereken noktalardan birisi de, **ComputeX** metodlarının **LineTest** isimli bir**nitelik(Attribute)** ile imzalanmış olmalarıdır. Bu niteliği metodları çalışma zamanında test için çağırırken kullanacağımız **Reflection** bazlı kodlarda değerlendireceğiz. *(Nitekim her test metodunu**Main** içerisinde arka arkaya yazıp, her biri için süre hesabını o kod satırlarında yazıp, satır sayısını arttırmak istemiyorum. **Maintability** ve **Readability** açısından biraz daha yüksek bir kod olsun niyetindeyim)*

Main metodu içerisinde dikkat edileceği üzere **Reflection**' dan yararlanılarak **FileProcessor** tipinin üye metodları arasında gezilmekte ve **LineTest** niteliği uygulanmış olanların çağırılması sağlanmaktadır.

Testler

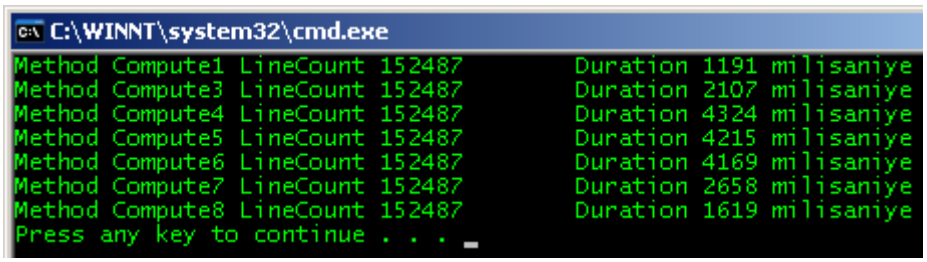
Teste tabi tutacağımız **text** dosyası içerisinde **Lorem Ipsum** metinlerinden bolca yer almaktadır. İlk testler için dosyamızda **10891** satır yer alıyor. Buna göre ilk sonuçlar aşağıdaki gibidir.



Şu hemen dikkatinizi çekmiş olmalıdır. **Compute2** metodu inanılmaz derecede yavaştır. En hızlı metod ise **Compute1** olmuştur ki içerisinde **File** tipinin **static** **ReadAllLines** metodunu kullandığımızı belirtmek isterim. Bu sonuçlara göre **Compute2** metodunu doğrudan elemeliyiz. İkinci testimizde bu metodu göz ardı edeceğiz. Şimdi dosyanın satır sayısını belirgin ölçüde arttırdığımızı düşünelim. Örneğin **152487** satır olsun.

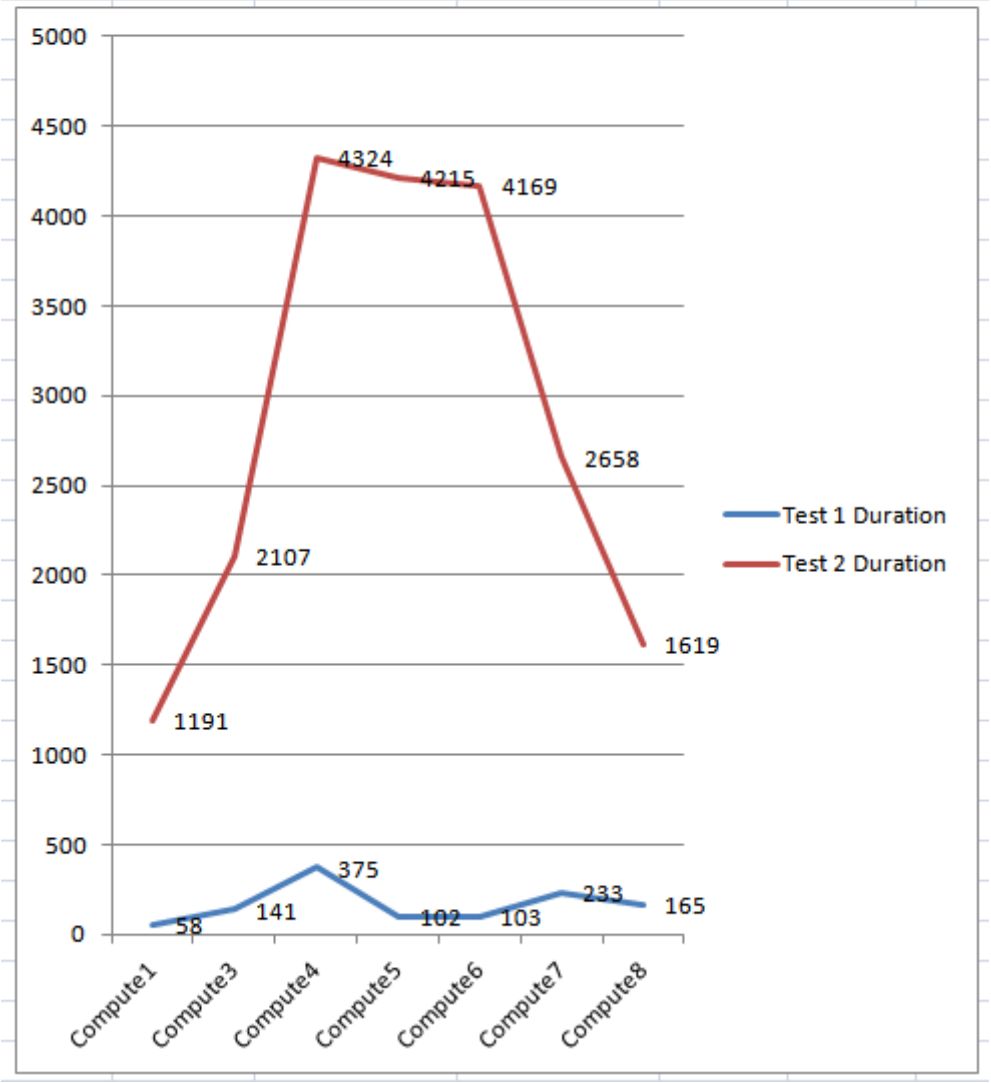
Çok mu sizce? Ben milyonları gördüğüm için bana normal geliyor aslında . Gerçi bu durumu **Visual Studio** bile yadırgadı. **50 Mb**' a yaklaşan boyutuyla **Text** dosyasını sadece **Notepad++** ile açabildim. Neyse tekrar konumuza dönelim. Amacımız boyutun

artması halinde, hesaplama metodlarının aynı performans istikrarını sağlayıp sağlamadığını görebilmek. İşte sonuçlar;



Compute2 metodunu hariç tuttuğumuzda **1nci** ve **2nci** testlerin sonuçlarını aşağıdaki tablo grafiğinde görüldüğü gibi yorumlayabiliriz.

	Milisaniye	
	Test 1 Duration	Test 2 Duration
Compute1	58	1191
Compute3	141	2107
Compute4	375	4324
Compute5	102	4215
Compute6	103	4169
Compute7	233	2658
Compute8	165	1619



Sonuçlar

Görüldüğü üzere sonuçlar gayet açık ve net. **File.ReadAllLines** metodunun eline pek su döken olmadı. Ancak yaklaşabilenler var. Peki bu metod kendi içerisinde nasıl bir çalışma modeline sahipte bu kadar hızlı sonuç döndürülmesine olanak sağlıyor?
private **static string[] InternalReadAllLines(string path, Encoding encoding)**

```
{
   List<string> list = new List<string>();
   using (StreamReader reader = new StreamReader(path, encoding))
   {
     string str;
      while ((str = reader.ReadLine()) != null)
      {
         list.Add(str);
      }
  }
   return list.ToArray();
}
```

Dikkat edileceği üzere **ReadAllLines** metodu aslında içeride başka bir metodu tetiklemekte*(InternalReadAllLines)*. Bu metod içerisinde **StreamReader**' dan yararlanılarak bir okuma işlemi yapılmakta ve satırların generic bir **List** koleksiyonunda toplanması sağlanmakta. Söz konusu metod, **generic list** koleksiyonun **Array**' e indirgenmiş halini döndürmektedir. Sonrasında zaten bu dönüş dizisinin eleman sayısını bulmamız yeterlidir.

Tabi satır sayısını bulmak için alternatif yollarda düşünebilir ve mutlaka vardır. Söz gelimi **Pointer**kullanımı daha hızlı sonuçlar alabilmemizi sağlayabilir. Ya da paralel programlama desenlerinden yararlanarak özellikle çok büyük boyutlu bir dosyanın parçalara bölünmek suretiyle satır sayısının hesaplanması ve **Reduction** ile kümülatif toplamın ortaya konulması düşünülebilir. Ancak önerilen yol çok büyük boyutlu dosyalarda **File.ReadAllLines** metodunun kullanılmasıdır. Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

FindLineCountsApp.zip (32,62 kb)

Sıralama Algoritmaları - Hangisi Daha Hızlı(Bubble,Quick,Insertion,Selection,Shell,Merge,Heap)

Çarşamba, 25 Eylül 2013 14:08 by bsenyurt

Merhaba Arkadaşlar,
Evimdeki çalışma odasında yer alan kütüphanemi zaman zaman gelen yeni kitaplar ve afacan S(h)arp Efe' nin haylazlıkları nedeni ile darma duman halde bulabiliyorum. Hal böyle olunca çoğu zaman kitaplıkta yer alan onlarca kitabı tekrardan düzenlememem ve uygun bir sırada dizmem gerekebiliyor. Hatta bunu kitapların tozunu almak için hepsini yerlere indirdikten sonra da yaşayabiliyorum. Aslına bakarsanız her seferinde farklı bir kategorilendirme yapıyor ve buna göre bir sıralama işlemi icra etmeye çalışıyorum. Tabi el çabukluğu dışında akıllı düşününce sıralamak ve yerleştirmek kısa sürede bitebiliyor. Ama bazen de kafa bulanık olunca bu işlem sandığımdan da uzun sürüp bir işkence haline gelebiliyor.
Üniversite yıllarında özellikle programlama derslerinde buna benzer şekilde sıralama algoritmaları ile haşır neşir olmayanımız yoktur eminim ki. Hatta çoğu sınavın korkulu rüyası sorularının kaynağını teşkil etmektedir ki hocalarımız genellikle bunları kağıt kalem kullanarak çözmemizi isterler*(En azından benim zamanında böyleydi)*
Özellikle bu algoritmaların dil bağımsız olan **Pseudo Code** içeriklerinden yararlanarak her hangi bir dile uygulanmaya çalışılması üzerine epeyce kafa yormuşuzdur. **C, C++, Java, Basic, Pascal** ve benzeri temel programlama derslerinde edindiğimiz bilgiler ile bu algoritmaları yazmak için çokça uğraşmışızdır. Tabi üzülerek söylemeliyim ki ben üniversite yıllarındayken **Bubble** ve **Quick Sort**sıralama algoritmalarından fazlasını ne yazık ki göremedim.
Dolayısıyla o zamanlarda iş başa düşmüş ve diğer algoritmaları kendi başıma araştırıp öğrenmeye çalışmıştım. Çok şükür ki günümüzde Internet elimizin altında ve her ne kadar bilgi kirliliği olması söz konusu ise de her çeşit bilgiye kolayca ulaşmamız mümkün. Dolayısıyla sıralama algoritmalarının fazlasını herhangibir dil için kolayca bulabiliyoruz.
Gelelim bu yazımızın konusuna. Geçtiğimiz günlerde yine boş anlarıma denk gelen bir zaman diliminde, sıralama algoritmaları arasındaki hız farklarını incelemeye çalışmaktaydım. İş bir süre sonra **Console** uygulamasının **Main** metodu içerisine hapsolmanın ötesine geçti. Dilerseniz hiç vakit kaybetmeden kodlarımızı değerlendirmeye başlayalım. Amacımız bir performans test uygulaması geliştirmek olacak.
Ele almayı düşündüğüm bir kaç sıralama algoritması söz konusu idi. Kimisinin yazımı oldukça kolay, kimisin ise inanılmaz karmaşıktı. Örnekte **Bubble, Heap, Insertion, Merge, Quick, Selection** ve**Shell** sıralama algoritmalarını ele aldım. Aslında amacım algoritmaların yazılmasından ziyade, bunları aynı dizi içerikleri ile test edebilmenin kolay

bir yolunu bulmaktı. Bu noktada aklımdaki yapıda tek bir **Execute** metodunun, kendisine parametre olarak gelen herhangibir sıralama algoritmasını, belli bir dizi için yapması gerektiğini düşünmekteydim. Bu tip bir test ortamını kurmak çok zor değildir aslında. Aynı metodun kendisine parametre olarak gelen herhangibir sıralama algoritmasını icra etmesini planlıyorsak bir **arayüz(Interface)** tipinden yararlanabiliriz.
Hatırlayacağınız üzere*(Hatta hatırlamanız gerektiği üzere)* Interface' ler ile, kendilerini uygulayan diğer tiplere icra etmesi gereken zorunlulukları bildirebilir ve diğer yandan çok biçimli olarak hareket etmelerini sağlayabiliriz. *(Özellikle Plug-Intabanlı programlamada çok işe yaradığını unutmayalım)*
Örneğimizde son derece basit bir arayüz tipi kullanılmaktadır. Aşağıdaki kod parçasında görüldüğü gibi.

```
namespace SortingAlgorithm
{
   public interface ISorter
   {
      string Description { get; }
      void Execute(int[] Array);
   }
}
```

ISorter arayüz tipi basit olarak iki **üye(Member)** bildirimi içermektedir. **Execute** metodu parametre olarak gelen **Array** dizisi üzerinde uygulanacak sıralama işlemini icra eden fonksiyon bildirimidir. Diğer yandan **Description** özelliği ile de sıralama algoritması için kısa bir açıklama verilmesi sağlanabilir. Dolayısıyla sıralama işlerini üstlenecek olan tiplerin bu arayüzü uygulamasını sağlayarak yola devam edebiliriz. *(Hem de tüm sıralama algoritmalarının C# tarafındaki kod karşılıklarını tek bir yazı içerisinde toplamış oluruz)* Öyleyse hiç vakit kaybetmeden sıralama algoritma tiplerini yazmaya başlayalım.

Bubble Sort

```
namespace SortingAlgorithm.Sorters
{
   public class Bubble
      :ISorter
   {
      #region ISorter Members

      public string Description
      {
         get { return "Bubble Sort Sıralama Algoritması"; }
      }

      public void Execute(int[] Array)
```

```
        {
            int i;
            int j;
            int TempValue;

            for (i = (Array.Length - 1); i >= 0; i--)
            {
                for (j = 1; j <= i; j++)
                {
                    if (Array[j - 1] > Array[j])
                    {
                        TempValue = Array[j - 1];
                        Array[j - 1] = Array[j];
                        Array[j] = TempValue;
                    }
                }
            }
        }

        #endregion
    }
}
```

Quick Sort

```
namespace SortingAlgorithm.Sorters
{
    public class Quick
        :ISorter
    {
        #region ISorter Members

        public string Description
        {
            get { return "Quick Sort Sıralama Algoritması"; }
        }

        public void Execute(int[] Array)
        {
            Sort(0, Array.Length - 1,Array);
        }
```

```
#endregion

private void Sort(int LeftValue, int RightValue,int[] Array)
{
   int PivotValue, LeftHoldValue, RightHoldValue;

   LeftHoldValue = LeftValue;
   RightHoldValue = RightValue;
   PivotValue = Array[LeftValue];

   while (LeftValue < RightValue)
   {
      while ((Array[RightValue] >= PivotValue) && (LeftValue < RightValue))
      {
         RightValue--;
      }

      if (LeftValue != RightValue)
      {
         Array[LeftValue] = Array[RightValue];
         LeftValue++;
      }

      while ((Array[LeftValue] <= PivotValue) && (LeftValue < RightValue))
      {
         LeftValue++;
      }

      if (LeftValue != RightValue)
      {
         Array[RightValue] = Array[LeftValue];
         RightValue--;
      }
   }

   Array[LeftValue] = PivotValue;
   PivotValue = LeftValue;
   LeftValue = LeftHoldValue;
   RightValue = RightHoldValue;

   if (LeftValue < PivotValue)
```

```
            {
                Sort(LeftValue, PivotValue - 1,Array);
            }

            if (RightValue > PivotValue)
            {
                Sort(PivotValue + 1, RightValue,Array);
            }
        }
    }
}
```

Insertion Sort

```
namespace SortingAlgorithm.Sorters
{
    public class Insertion
            : ISorter
    {
        #region Sorter Members

        public string Description
        {
            get { return "Insertion Sort Sıralama Algoritması"; }
        }

        public void Execute(int[] Array)
        {
            int i;
            int j;
            int IndexValue;

            for (i = 1; i < Array.Length; i++)
            {
                IndexValue = Array[i];
                j = i;

                while ((j > 0) && (Array[j - 1] > IndexValue))
                {
                    Array[j] = Array[j - 1];
                    j = j - 1;
                }
```

```
            Array[j] = IndexValue;
         }
      }

      #endregion
   }
}
```

Selection Sort

```
namespace SortingAlgorithm.Sorters
{
   public class Selection
      :ISorter
   {
      #region ISorter Members

      public string Description
      {
         get { return "Selection Sıralama Algoritması"; }
      }

      public void Execute(int[] Array)
      {
         int i, j;
         int MinValue, TempValue;

         for (i = 0; i < Array.Length - 1; i++)
         {
            MinValue = i;

            for (j = i + 1; j < Array.Length; j++)
            {
               if (Array[j] < Array[MinValue])
               {
                  MinValue = j;
               }
            }

            TempValue = Array[i];
            Array[i] = Array[MinValue];
```

```
            Array[MinValue] = TempValue;
          }

        }

        #endregion
    }
}
```

Shell Sort

```
namespace SortingAlgorithm.Sorters
{
   public class Shell
     :ISorter
   {
      #region ISorter Members

     public string Description
      {
         get { return "Shell Sıralama Algoritması"; }
      }

      public void Execute(int[] Array)
      {
         int i, j, Increment, TempValue;

         Increment = 3;

         while (Increment > 0)
         {
            for (i = 0; i < Array.Length; i++)
            {
               j = i;
               TempValue = Array[i];

               while ((j >= Increment) && (Array[j - Increment] > TempValue))
               {
                  Array[j] = Array[j - Increment];
                  j = j - Increment;
               }
```

```
                Array[j] = TempValue;
            }

            if (Increment / 2 != 0)
            {
                Increment = Increment / 2;
            }
            else if (Increment == 1)
            {
                Increment = 0;
            }
            else
            {
                Increment = 1;
            }
        }

    }

    #endregion
  }
}
```

Merge Sort

```
namespace SortingAlgorithm.Sorters
{
  public class Merge
    :ISorter
  {
    int[] Array2;

    #region ISorter Members

    public string Description
    {
      get { return "Merge Sıralama Algoritması"; }
    }

    public void Execute(int[] Array)
    {
      Array2 = new int[Array.Length];
```

```
    Sort(0, Array.Length - 1,Array);
}

#endregion

private void Sort(int LeftValue, int RightValue,int[] Array)
{
    int mid;

    if (RightValue > LeftValue)
    {
        mid = (RightValue + LeftValue) / 2;
        Sort(LeftValue, mid,Array);
        Sort(mid + 1, RightValue,Array);

        DoMerge(LeftValue, mid + 1, RightValue,Array);
    }
}

private void DoMerge(int LeftValue, int MiddleValue, int RightValue,int[] Array)
{
    int i, LeftEnd, NumberOfElements, TempPosition;

    LeftEnd = MiddleValue - 1;
    TempPosition = LeftValue;
    NumberOfElements = RightValue - LeftValue + 1;

    while ((LeftValue <= LeftEnd) && (MiddleValue <= RightValue))
    {
        if (Array2[LeftValue] <= Array[MiddleValue])
        {
            Array2[TempPosition] = Array[LeftValue];
            TempPosition = TempPosition + 1;
            LeftValue = LeftValue + 1;
        }
        else
        {
            Array2[TempPosition] = Array[MiddleValue];
            TempPosition = TempPosition + 1;
            MiddleValue = MiddleValue + 1;
        }
```

```
            }

            while (LeftValue <= LeftEnd)
            {
               Array2[TempPosition] = Array[LeftValue];
               LeftValue = LeftValue + 1;
               TempPosition = TempPosition + 1;
            }

            while (MiddleValue <= RightValue)
            {
               Array2[TempPosition] = Array[MiddleValue];
               MiddleValue = MiddleValue + 1;
               TempPosition = TempPosition + 1;
            }

            for (i = 0; i < NumberOfElements; i++)
            {
               Array[RightValue] = Array2[RightValue];
               RightValue = RightValue - 1;
            }
         }
      }
   }
```

Heap Sort

```
namespace SortingAlgorithm.Sorters
{
   public class Heap
     :ISorter
   {
      #region ISorter Members

      public string Description
      {
         get { return "Heap Sıralama Algoritması"; }
      }

      public void Execute(int[] Array)
      {
         for (int i=(Array.Length-1)/2; i >= 0;i--)
```

```
            Adjust (Array, i, Array.Length - 1);

        for (int i = Array.Length - 1; i >= 1; i--)
        {
            int Temp = Array[0];
            Array[0] = Array[i];
            Array[i] = Temp;
            Adjust (Array, 0, i - 1);
        }
    }

    #endregion

    private void Adjust(int[] Array, int i, int m)
    {
        int TempValue = Array[i];
        int j = i * 2 + 1;

        while (j <= m)
        {
            if (j < m)
                if (Array[j] < Array[j + 1])
                    j = j + 1;

            if (TempValue < Array[j])
            {
                Array[i] = Array[j];
                i = j;
                j = 2 * i + 1;
            }
            else
            {
                j = m + 1;
            }
        }

        Array[i] = TempValue;
    }
  }
}
```

Vowwww!!! Ne çok kod, ne çok algoritma, ne çok matematik...

Örneğimizde bir de Performans testini gerçekleştirecek yardımcı bir tipe ihtiyacımız olacaktır. Bu sınıfı da aşağıdaki gibi tasarladığımızı düşünebiliriz.

```
using System;
using System.Diagnostics;
using System.IO;

namespace SortingAlgorithm
{
   public class PerformanceTester
   {
      public void Execute(ISorter Sorter, int[] Array)
      {
         Stopwatch watcher = new Stopwatch();
         watcher.Start();

         Sorter.Execute(Array);

         watcher.Stop();

         File.AppendAllText(
            Path.Combine(Environment.CurrentDirectory, "ExecutionLog.txt")
            , String.Format("{0} Sıralama Algoritması için Toplam Çalışma Süresi : {1} milisaniyedir. Dizi boyutu {2}|"
            , Sorter.Description
            , watcher.ElapsedMilliseconds.ToString()
            ,Array.Length)
            );
      }
   }
}
```

Execute metoduna dikkat edilecek olursa **ISorter** interface tipinden bir parametre aldığı görülmektedir. Dolayısıyla bu metoda, yukarıda tanımlanmış olan sıralama sınıflarından herhangibiri atanabilir. Çünkü hepsi bu **arayüzü(Interface)** implemente etmektedir. Diğer yandan **Execute**metodu, **çalışma zamanında(Runtime), Sorter** isimli değişkenin büründüğü sıralama sınıfının**Execute** metodunu yürütmekte ve ayrıca bu sıralama algoritması için bir **Text** dosyaya **log**atmaktadır. Metodumuzun önemli özelliklerinden birisi de, **Stopwatch** tipini kullanarak sıralama işlemi için gerekli çalışma süresi farkını hesaplıyor ve bunu yine **Text** dosyası içerisine raporluyor olmasıdır.

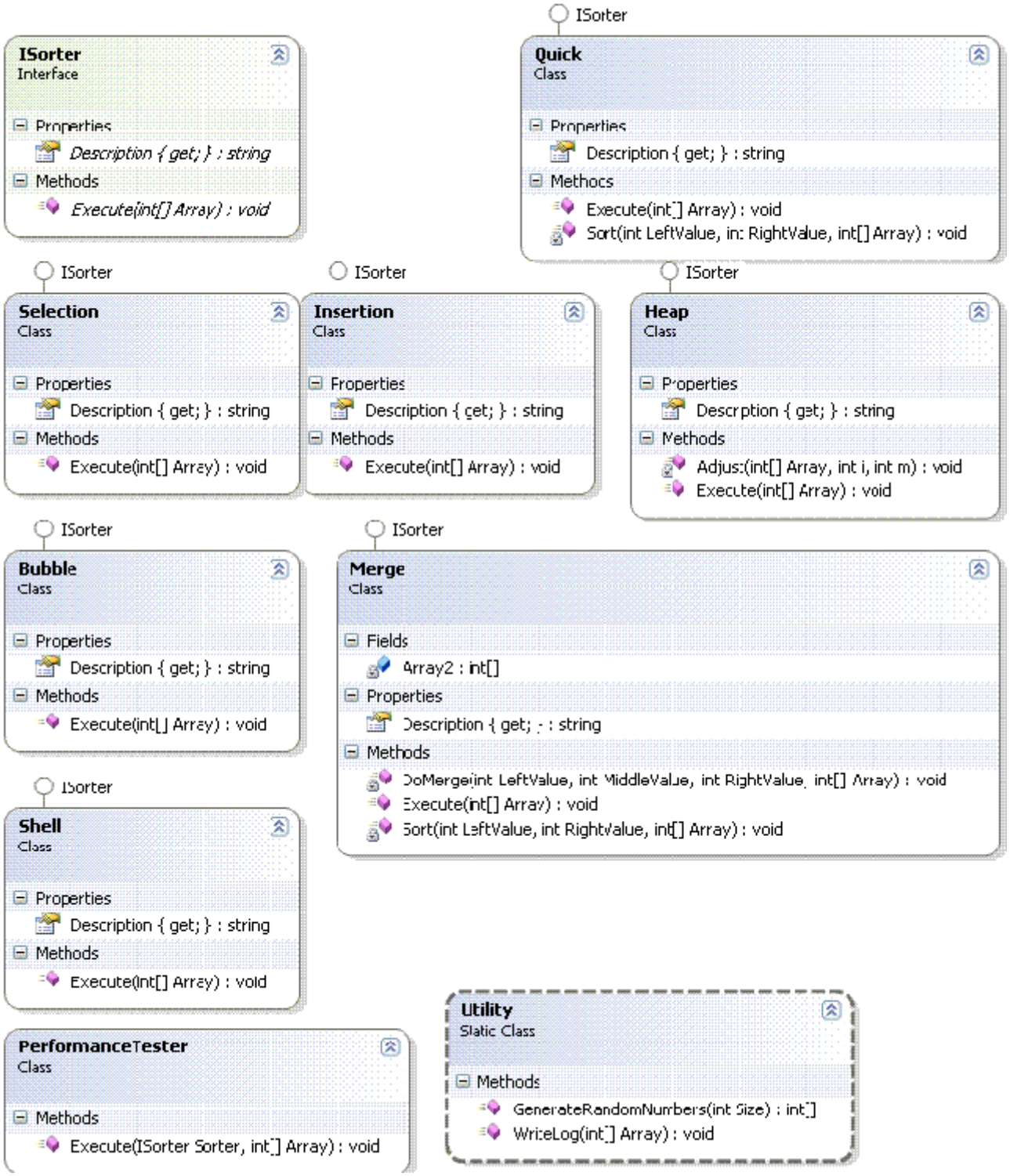
Uygulamanın Testi

Örnek uygulamamız aslında bir **Test** uygulamasıdır. Amacı çeşitli sıralama algoritmalarını, içerikleri karışık tamsayılardan oluşan birden fazla boyuttaki dizi için çalıştırmak ve çalışma zamanındaki toplam icra süresi farklarını görmektir. Dolayısıyla bize yardımcı bir kaç fonksiyonellik daha gerekmektedir. Örneğin belirtilen boyutta rastgele **int** tipinde sayılardan oluşacak bir dizi üretimi fonksiyonelliği ve hatta bu dizinin ham ve sıralanmış hallerinin karşılıklarını yine **log** amaçlı olarak**Text** dosyasına yazdıracak bir metod düşünülebilir. Bunun için uygulamamıza **Utility** isimli **static**bir tip ilave edebiliriz.

```
using System;
using System.IO;
using System.Text;

namespace SortingAlgorithm
{
   public static class Utility
   {
      public static int[] GenerateRandomNumbers(int Size)
      {
         int[] numbers = new int[Size];
         Random rnd = new Random();
         for (int i = 0; i < Size; i++)
         {
            numbers[i] = rnd.Next(1, Size);
         }

         return numbers;
      }

      public static void WriteLog(int[] Array)
      {
         StringBuilder builder = new StringBuilder();

         foreach (int element in Array)
         {
            builder.AppendLine(String.Format("{0} ", element.ToString()));
         }

         File.AppendAllText(
            Path.Combine(Environment.CurrentDirectory, "ExecutionLog.txt")
            , builder.ToString()
            );
      }
```

```
    }
}
```

GenerateRandomNumbers metodu testler için gerekli **n boyutlu int tipinden dizi** üretimini üstlenmektedir. Söz konusu kobay diziler rastgele **int** sayılarından oluşmaktadır. Diğer yandan**WriteLog** metodu' da **int** tipinden herhangibir dizinin içeriğini*(Sıralı veya değil)* **ExecutionLog**isimli text tabanlı dosyaya yazmaktadır.Buraya kadar yazmış olduğumuz tipleri aşağıdaki sınıf diagramında daha net bir şekilde görebilirsiniz.



Artık yazmış olduğumuz tipleri kullanarak ilgili test işlemlerini gerçekleştirecek olan uygulama kodunu geliştirmeye başlayabiliriz.

```
using System;
using System.IO;
using SortingAlgorithm.Sorters;

namespace SortingAlgorithm
{
   class Program
   {
      #region Program değişlenleri

      static PerformanceTester neco = new PerformanceTester();
      static Insertion insertion = new Insertion();
      static Bubble bubble = new Bubble();
      static Heap heap = new Heap();
      static Quick quick = new Quick();
      static Selection selection = new Selection();
      static Shell shell = new Shell();
      static Merge merge = new Merge();
      static Tuple<int[], int[], int[], int[], int[], int[], int[]> numbers;

      #endregion

      static void Main(string[] args)
      {
         Console.WriteLine("Testler başladı. Lütfen bekleyiniz");

         numbers = Load(100);
         Execute(numbers);

         numbers = Load(500);
         Execute(numbers);

         numbers = Load(1000);
         Execute(numbers);

         numbers = Load(5000);
         Execute(numbers);
```

```
            numbers = Load(10000);
            Execute(numbers);
            numbers = Load(50000);
            Execute(numbers);
            numbers = Load(100000);
            Execute(numbers);
            numbers = Load(500000);
            Execute(numbers);

            Console.WriteLine("Testler tamamlandı. {0}");

            string Result = File.ReadAllText(Path.Combine(Environment.CurrentDirectory,
"ExecutionLog.txt"));
            Console.WriteLine(Result);
        }

        private static Tuple<int[], int[], int[], int[], int[], int[], int[]> Load(int MaxValue)
        {
            int[] randomNumbers = Utility.GenerateRandomNumbers(MaxValue);
            numbers = new Tuple<int[], int[], int[], int[], int[], int[], int[]>(
                    randomNumbers
                    , (int[])randomNumbers.Clone()
                    , (int[])randomNumbers.Clone()
                    , (int[])randomNumbers.Clone()
                    , (int[])randomNumbers.Clone()
                    , (int[])randomNumbers.Clone()
                    , (int[])randomNumbers.Clone()
            );
            return numbers;
        }

        private static void Execute(Tuple<int[], int[], int[], int[], int[], int[], int[]>
numbers)
        {
            neco.Execute(bubble,numbers.Item1);
            neco.Execute(heap, numbers.Item2);
            neco.Execute(insertion, numbers.Item3);
            neco.Execute(merge, numbers.Item4);
            neco.Execute(quick, numbers.Item5);
            neco.Execute(selection, numbers.Item6);
            neco.Execute(shell, numbers.Item7);
```

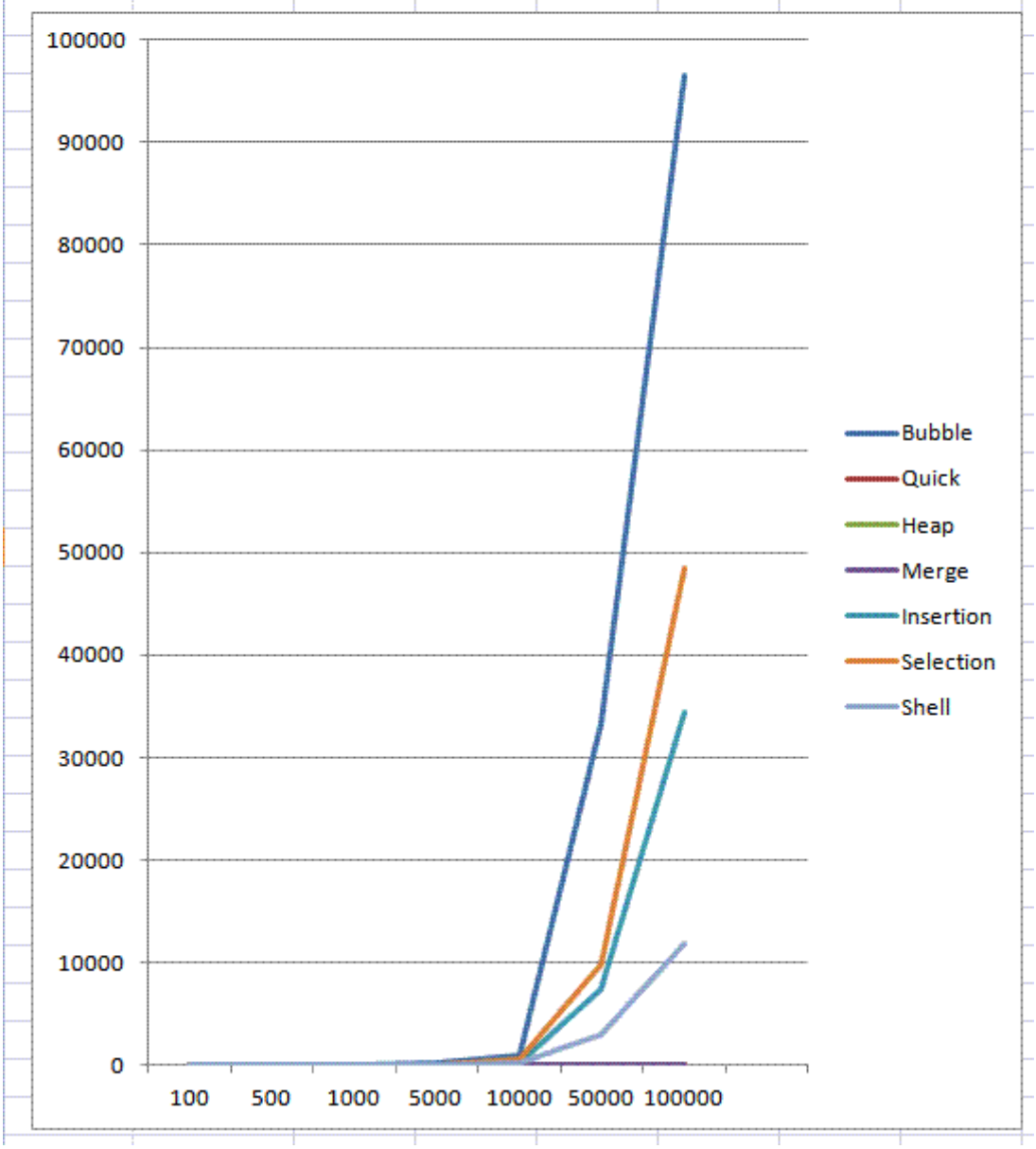
```
      }
   }
}
```

Program kodumuzda kritik olan noktalardan birisi, üretilen 7 adet dizinin **Tuple<>** tipine yüklenişi sırasında ortaya çıkmaktadır. Diziler bildiğiniz üzere referans tipleridir. Dolayısıyla bir metoda parametre olarak atandıklarında ve içeride değişikliğe uğradıklarında, bu metodun çağırıldığı yerdeki orjinal dizinin içeriğinin de değişmesi söz konusudur. Çünkü her ikiside zaten aynı adresi işaret eden bir pointer' dır. Bu sebepten **Tuple<>** içerisindeki dizileri set ederken klonlamamız gerekmektedir. Aksi durumda ilk dizinin sıralanmasını takiben diğer metodların kullanacağı tüm dizilerin zaten sıralanmış olarak işleme alındığı görülecektir.
Demek ki basit bir Performans Test uygulaması yazarken dahi çok dikkali davranmalı ve özellikle Debug işlemlerine ağırlık vermeliyiz.

Sonuçlar

Artık uygulamamızı çalıştırabilir ve sonuçları irdeleyebiliriz. Dizilerimize ait değer aralıkları **100, 500, 1000 , 5000, 10000, 50000 ve 100000'** dir. Çok doğal olarak son değer aralıklarının büyüklüğü nedeni ile uygulama ilgili sıralama algoritmalarını oldukça zorlayacaktır ki bu istediklerimizden birisidir. Ben şahsen uygulamayı denediğimde bir kaç saat çalıştığına şahit oldum. Tabi burada işi yavaşlatan farklı faktörler ve etkenlerde var. Ancak sonuç olarak aşağıdaki test değerlerini elde ettim.

	Değer Aralığına Göre Toplam Çalışma Süresi						
Algoritma	**100**	**500**	**1000**	**5000**	**10000**	**50000**	**100000**
Bubble	0	2	10	216	1005	33329	96600
Quick	1	0	0	1	2	16	51
Heap	1	0	0	2	4	24	83
Merge	1	0	0	2	4	21	82
Insertion	0	1	2	105	292	7412	34338
Selection	0	1	4	103	482	9907	48485
Shell	0	0	1	27	164	2945	11911



Görüldüğü üzere değer aralığı büyüdükçe en hızlı sonuçlar **Quick Sort** algoritmasından gelmektedir. Diğer yandan **Bubble Sort** algoritmasının çok fazla maliyet getirdiği ve uzun sürdüğü ortadadır.**Heap** ve **Merge** algoritmaları birbirlerine yakın süreler vermiştir. Insertion, Selection ve Shell algoritmaları tatmin edici hızlarda değildir. Tabi buradaki süreler **Milisaniye** cinsinden olup alsında çok fazla önemli görünmeyebilir. Ancak

matematiksel hesaplamaların yoğun yapıldığı bilimsel uygulamalarda sıklıkla kullanıldıkları ve ihtiyaç duyuldukları da bilinmektedir.
Elbette çok daha etkili ve faydalı bir test uygulaması yazılabilir.
Mesela **http://www.sorting-algorithms.com** adresinde bunun web tabanlı güzel bir örneği bulunmaktadır. Diğer yandan akılda oluşması gereken önemli sorulardan birisi de şudur. Acaba bu sıralamaların paralelize edilmiş versiyonları var mıdır? Olay sadece **Parallel.ForEach** veya **Parallel.For** metodlarını kullanmak kadar basit olabilir mi? Yoksa dikkat edilmesi gereken başka unsurlar da var mıdır? Bu soruların cevaplarını ilerleyen yazılarımızda bulmaya çalışıyor olacağız. Diğer yandan bu yazımızda sadece kullandığımız sıralama algoritmaları hakkında en iyi bilgilere wikipedia üzerinden ulaşabilirsiniz.(Bubble Sort,Insertion Sort, Selection Sort, Heap Sort, Quick Sort, Merge Sort, Shell Sort) Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

SortingAlgorithm.rar (40,36 kb)

Sıfır Sabit Değeri ve Enum Sorunu

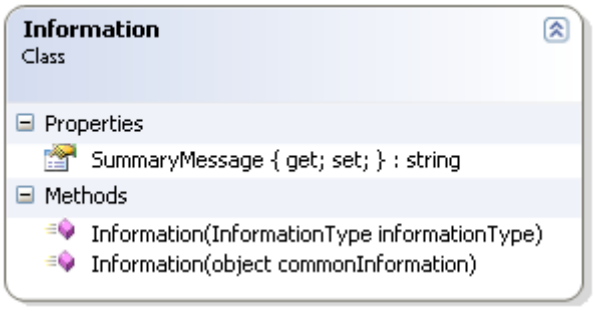
Cuma, 20 Eylül 2013 07:48 by bsenyurt

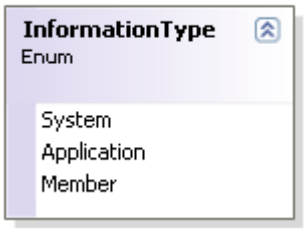
Merhaba Arkadaşlar,
C# konulu yeni bir bilmece ile karşı karşıyayız. Bu sefer kolay kolay fark edemeyebileceğimiz, basit ama irdelenmesi gereken bir vakayı göz önüne alacağız.

Bir uygulamayı geliştirirken, Developer olarak son derece dikkatli olmalı ve testlerimizi gerçekleştirirken de tüm senaryoları göz önüne almalıyız. Bu anlamda gerek dilin gerek**.Net Framework** alt yapısının tüm unsurlarına hakim olmak da son derece önemli.
Nitekim bazı noktalarda profesyonel bir geliştiricinin dahi kestiremeyeceği sorunsallar yaşanabiliyor. Yazımızın ilerleyen kısımlarında bu perspektiften bakarak sorun olarak da görülebilecek bir konuyu masaya yatıracağız.

Senaryo

İlk olarak **C#** dili ile geliştireceğimiz aşağıdaki basit kod parçasını göz önüne alalım.






```
using System;
using System.Collections.Generic;
namespace EnumAndZeroConstant
{
   class Program
   {
      static void Main(string[] args)
      {
         List<Information> infos = new List<Information>()
```

```
                        {
                            new Information("Hata mesajı"),
                            new Information(InformationType.Application),
                                    new Information(InformationType.Member),
                                    new Information(InformationType.System),
                                    new Information(0),
                                    new Information(1),
                                    new Information(2),
                                    new Information(3)
                        };
        foreach (Information info in infos)
        {
            Console.WriteLine(info.SummaryMessage);
        }
    }
}
enum InformationType
{
    System,
    Application,
    Member
}
class Information
{
    public string SummaryMessage { get; private set; }
    public Information(object commonInformation)
    {
        SummaryMessage = commonInformation.ToString();
    }
    public Information(InformationType informationType)
    {
        switch (informationType)
        {
            case InformationType.Application:
                SummaryMessage = "Uygulama bilgisi";
                break;
            case InformationType.System:
                SummaryMessage = "Sistem bilgisi";
                break;
            case InformationType.Member:
                SummaryMessage = "Üyeden";
```

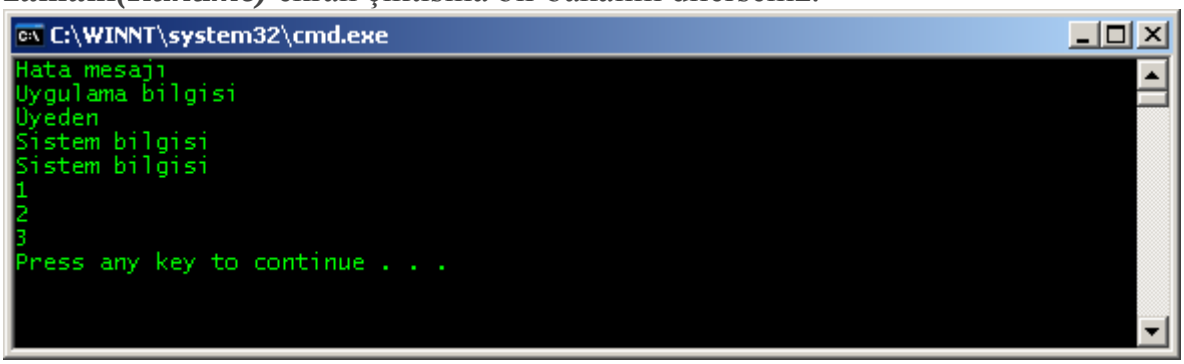
```
                break;
            default:
                SummaryMessage = "Bilinmeyen kaynak";
                break;
        }
    }
  }
}
```

Uygulamamızda **InformationType** isimli bir **Enum** sabiti ve bu tipi kullanan **Information** isimli bir sınıf(Class) tanımlanmıştır. **Information** sınıfı içerisinde yer alan **SummaryMessage** isimli**özelliğimiz(Property)** dikkat edileceği üzere **Read Only** belirtilmiştir(*set keyword' ü başındaki private kullanımına dikkat)*
Information sınıfı içerisinde kullanılan **SummaryMessage** özelliğinin değerini set etmek için,**yapıcı metod(Constructor)** parametrelerinden yararlanılmaktadır. İki adet **aşırı yüklenmiş yapıcı metod(Overload Contstructor)** mevcuttur**.** Bunlardan birisi **InfortmationType enum**sabiti tipinden bir değişken almakta ve bunun değerine göre **SummaryMessage** özelliğine **string**bir değer atamaktadır. Bu anlamda diğer aşırı yüklenmiş versiyon ise, **object** tipinden gelen referans değişkenin, **ToString** metodunun sonucunu kullanmaktadır.
Main metodu içerisinde test amaçlı olaraktan**, Information** tipinden **generic** bir **List** koleksiyonu değerlendirilmektedir. Dikkat edileceği üzere, koleksiyonun **Information** tipinden olan her bir nesne örneği üretilirken, parametre olarak farklı referanslar gönderilmektedir. Test amacıyla **String, int, InfortmationType** türünden değişken değerleri ele alınmaktadır.
"Peki bu kod parçasında nasıl bir tuzak olabilir? Ya da başka bir deyişle bir tuzak var mıdır? Gözümüzden kaçırdığımız bir nokta söz konusu ise nedir? "

Düşünsel ve Gerçek Çalışma Sonuçları

Bu sorulara cevap bulabilmek için öncelikli olarak kodun nasıl bir sonuç üreteceğini düşünmeye çalışalım. Aslında beklentimiz, uygulamanın çıktısının aşağıdaki temsilde olduğu gibi üretilmesidir.
Hata mesajı
Uygulama bilgisi
Üyeden
Sistem bilgisi
0
1
2
3

Ancak, yapıcı metoda gönderilen **0** değerinden pis kokular gelmektedir. Ne demek istediğimi daha net bir şekilde ifade edebilmek için uygulamanın **çalışma zamanı*(Runtime)*** ekran çıktısına bir bakalım dilerseniz.



Uppsss! Bir fark görebildiniz mi? Biz yapıcı metodumuza **0** değişkenini göndermemize rağmen, ekrana **InformationSystem enum** sabitindeki ilk değere karşılık gelen **string** mesajı döndürülmüştür. Gerçekten de uygulamayı **debug** ettiğimizde, **0** değerini yapıcı metoda gönderdikten sonra **Enum** sabitini parametre olarak kullanan versiyonuna gittiğimizi görürüz. Kodu biraz değiştirip **Debug** işlemimizi icra edelim ve bu durumu çalışma zamanında görelim.

```
        {
            Information infoZero=new Information(0);
            List<Information> infos = new List<Information>()
                                        {
                                            new Information("Hata mesajı")
                                            new Information(InformationTyp
                                                            new Informatio
                                                            new Informatio
                                                            infoZero,
                                                            new Informatio
                                                            new Informatio
                                                            new Informatio
                                        };

            foreach (Information info in infos)
            {
                Console.WriteLine(info.SummaryMessage);
            }

        }
    }

    enum InformationType
    {
        System,
        Application,
        Member
    }

    class Information
    {
        public string SummaryMessage { get; private set; }

        public Information(object commonInformation)
        {
            SummaryMessage = commonInformation.ToString();
        }

        public Information(InformationType informationType)
        {
            switch (informationType)
```

informationType | System

infoZero isimli **Information** tipinden olan değişken örneklenirken, yapıcı metoda **0 integer** değeri gönderilmiştir. Ancak kod bu satırdan sonra **object** tipinden parametre alan versiyon yerine**InformationType** tipinden referans kullanan **yapıcı metoda** sıçramıştır. Üstelik **0 integer** değer,**bilinçsiz bir şekilde(Implicitly) InfortmationType.System** sabit değerine dönüştürülmüştür.
Tabi bizim buradaki asıl beklentimiz **0** gibi bir sayısalın **object** gibi algılanması ve uygun olan yapıcı metoda gitmesi yönündedir. Ne varki bu gerçekleşmemiştir. Bu aslında **0** sayısal değerinin bilinçsiz olarak bir **Enum** sabiti formunda algılanıyor olmasıdır. Yani sıfır tabanlı sabit olarak algınanan yapıcı metod parametresi, bilinçsiz olarak **Enum** karşılığına dönüştürülmüştür.

IL Tarafındaki Görüntü

Zaten uygulamamızın **IL*(Intermediate Language)*** tarafındaki çıktısına baktığımızda, **0** atamasının yapıldığı yerde **InformationType enum** sabitinin kullanıldığı versiyonun çağırılacağı da önceden belirtilmiştir.

```
.method private hidebysig static void  Main(string[] args) cil managed
{
  .entrypoint
  // Code size       191 (0xbf)
  .maxstack  3
  .locals init ([0] class EnumAndZeroConstant.Information infoZero,
           [1] class [mscorlib]System.Collections.Generic.List`1<class
EnumAndZeroConstant.Information> infos,
           [2] class EnumAndZeroConstant.Information info,
           [3] class [mscorlib]System.Collections.Generic.List`1<class
EnumAndZeroConstant.Information> '<>g__initLocal0',
           [4] valuetype [mscorlib]System.Collections.Generic.List`1/Enumerator<class
EnumAndZeroConstant.Information> CS$5$0000,
           [5] bool CS$4$0001)
  IL_0000:  nop
  IL_0001:  ldc.i4.0
  IL_0002:  newobj     instance void
EnumAndZeroConstant.Information::.ctor(valuetype
EnumAndZeroConstant.InformationType)
```

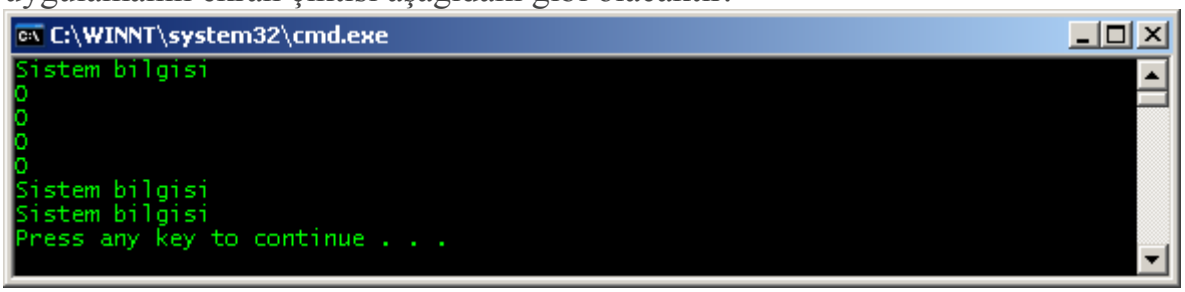
Sorunun Çözümü

Bu gözden kolayca kaçabilecek bir noktadır. Hani test süreçlerinde **Zero-Data** diye bir senaryo vardır gerçi ama bu vaka zaten onunla bir değildir. Sadece bir isim benzerliği olduğunu ifade edebiliriz. Bizim burada sorunu nasıl çözebileceğimize bir bakmamız gerekiyor. Mademki ortada**bilinçsiz bir tür dönüşümü var(Implicitly Type Conversion)** bu durumda dönüşümü bilinçli hale getirmeyi deneyebiliriz**(Explicitly Type Conversion)**. Aşağıdaki kod parçasındaki denemeleri yaptığımızı düşünelim.

```
Information infoZero1 = new Information((int)0);
var zeroVar = 0;
Information infoZero2=new Information(zeroVar);
object zeroObj = 0;
Information infoZero3 = new Information(zeroObj);
int zeroInt = Convert.ToInt32("0");
Information infoZero4 = new Information(zeroInt);
double zeroDbl = 0.0;
Information infoZero5 = new Information(zeroDbl);
Information infoZero6 = new Information(0.0);
var infoZero7 = new Information(0);
```

```
List<Information> infos = new List<Information>()
                    {
                                infoZero1,
                                infoZero2,
                                infoZero3,
                                infoZero4,
                                infoZero5,
                                infoZero6,
                                infoZero7
                    };
```

Burada farklı şekillerde **0** değerinin gönderilmesi söz konusu olmuştur. İlk olarak **cast** operatörü göz önüne alınarak **0** değerinin **int** tipinden ele alınması denenmiştir. İkinci olarak **0** değişkenini taşıyan bir **var** tanımlaması yapılmaktadır. Devam eden kısımda sırasıyla **Object** tipine atanarak yapıcı metoda gönderilme, **Convert** tipinin **static Int32** metodundan yararlanma, **double** bir değişken atama, yapıcı metoda parametre olarak **0** gönderirken **var** kullanarak ilerleme seçenekleri de denenmiştir. Bu durumda uygulamanın ekran çıktısı aşağıdaki gibi olacaktır.

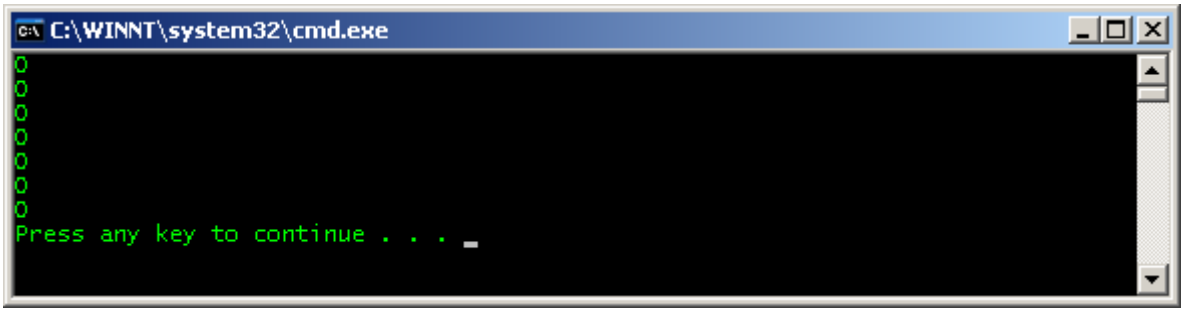


Dikkat edileceği üzere yapıcı metoda **0** değeri atama işlemini, dışarıda tanımlanan bir değişken üzerinden gerçekleştirdiğimizde **object** tipini kullanan yapıcı metoda gidilmiştir. Tabi bu durumda**Information** tipini kullanan geliştiricinin bu durumu göz önüne alması ve öncesinde gerekli dönüşüm işlemlerini yapması gibi bir sorumluluk ortaya çıkmaktadır. Oysaki geliştirici veya **Object User** çok kolayca durumu gözden kaçırabilir. Daha kalıcı bir çözüm ile ilerlemek gerekmektedir. Bunun için **Information** tipinin yapıcı metodlarının sayısını arttırıp tipe göre bir aksiyon alınması sağlanabilir. Aşağıdaki kod parçasın görüldüğü gibi.

```
class Information
{
   public string SummaryMessage { get; private set; }
   public Information(int info)
   {
      SummaryMessage = info.ToString();
   }
   public Information(double info)
   {
      SummaryMessage = info.ToString();
```

```
    }
    public Information(object commonInformation)
    {
        SummaryMessage = commonInformation.ToString();
    }
    public Information(InformationType informationType)
    {
        switch (informationType)
        {
            case InformationType.Application:
                SummaryMessage = "Uygulama bilgisi";
                break;
            case InformationType.System:
                SummaryMessage = "Sistem bilgisi";
                break;
            case InformationType.Member:
                SummaryMessage = "Üyeden";
                break;
            default:
                SummaryMessage = "Bilinmeyen kaynak";
                break;
        }
    }
}
```

Dikkat edileceği üzere **int** ve **double** tiplerini parametre olarak alan iki ek yapıcı metod daha ilave edilmiştir. Bu durumda **0** ve **0.0** değerleri için uygun olan yapıcı metodlar devreye girecek, bir başka deyişle **InformationType enum** sabitin kullanan yapıcı metod göz ardı edilecektir. Tabi yine de çok şık bir çözüm olmadığını ifade etmemiz gerekiyor. Nitekim **0** değerini taşıyabilecek pek çok sayısal tip mevcuttur. **Byte, short, float** vb…Dolayısıyla bunların her biri için bir aşırı yüklenmiş yapıcı metod yazmak çok da yerinde olmayabilir. Yine de **Information** tipini kullanacak olan geliştirici için hata riski bu şekilde azaltılabilir. Uygulamayı bu haliyle çalıştırdığımızda aşağıdaki ekran çıktısını elde ederiz.



İstediğimiz sonuca ulaştık. Görüldüğü üzere **Enum** sabitlerini yapıcı metodlara parametre olarak geçirdiğimizde, göndereceğimiz **0** sabit değeri otomatik olarak bir **enum** sabiti tipine dönüştürülmektedir. Dolayısıyla söz konusu sınıf içerisinde **object** tipini kullanan başka bir yapıcı metod varsa, tedbir almadığımız durumda fark etmeyeceğimiz bir çalışma zamanı sonucu üretilebilir. Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.
EnumAndZeroConstant.zip (1,97 mb)

En Kısa Metni Bulmak

Pazartesi, 2 Eylül 2013 17:45 by bsenyurt

Merhaba Arkadaşlar,

Uzun zamandır makale yazmaya çalışmakta ve öğrendiklerimi, edindiğim tecrübeleri sizlere aktarmaktayım. Tabi zaman ilerledikçe yazacak konu bulmakta da bir hayli zorlanıyor insan. Bu noktada öğrenmeninin sınırının olmadığını hepimiz biliyoruz. Olaya bu açıdan baktığımızda yazılmaya ve araştırılmaya değer binlerce konu olduğunu gönül rahatlığıyla ifade edebilirim. Yazma hevesli bir birey olarak bu benim için gerçekten önemli 🙂

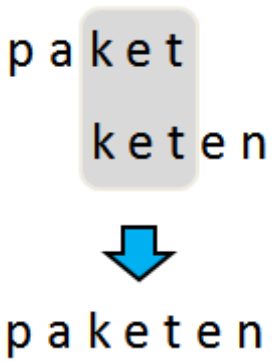
İşte geçtiğimiz hafta içerisinde de Internet üzerinden araştırma yaparken enteresan bir konu ile karşılaştım. Aslında konuyu isimlendirmek oldukça zor ama bir optimizasyon işlemi olduğunu ifade edebilirim. Sorun n sayıda kelimenin saklanmak istendiği bir durumda ortaya çıkıyor. İstenen, bu kelimeleri birleştirerek saklamak ancak bunu yaparkende olabilecek en kısa cümleyi elde ederek ilgili depolama işlemini gerçekleştirmek. Öyleki, üretilen cümle hem çok kısa olmalı hem de tüm kelimeleri içermeli.

Dilerseniz önce sorunu ve ulaşmak istediğimiz hedefi bir örnek üzerinden ifade etmeye çalışalım. Elimizde 4 adet kelime olduğunu farz edelim. **Enginar, Arpa, Keten** ve **Paket.** Normal şartlarda bu 4 kelimeyi sırasız olarak birleştirirsek aşağıdaki gibi bir **string** katarının oluştuğunu görürüz.

EnginarArpaKetenPaket

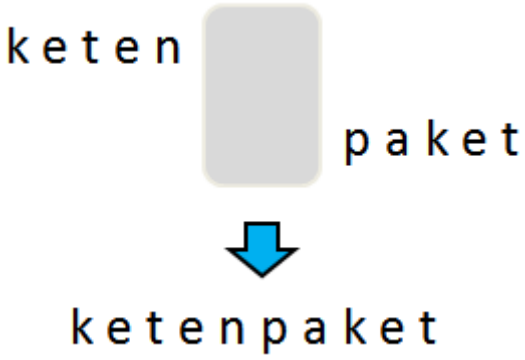
Bu son derece doğal bir sonuç tabi 😉 Ancak amacımız bu metni mümkün olduğu kadar kısaltmak. Peki bunu nasıl yapabiliriz?

Aslına bakarsanız bir şekilde kelimeler arasındaki ilişkileri matematiksel olarak anlamlandırmak ve olabilecek en iyi kombinasyonları seçerek ilerlemeye çalışmak yerinde olacaktır. Bunu düşünerek **Paket** ile **Keten** kelimelerini göz önüne alalım.



Dikkat edileceği üzere **paket** ile **keten** kelimelerini yan yana getirdiğimizde, **KET** hecesinin ortak olmasından dolayı **paketen** şeklinde daha kısa bir

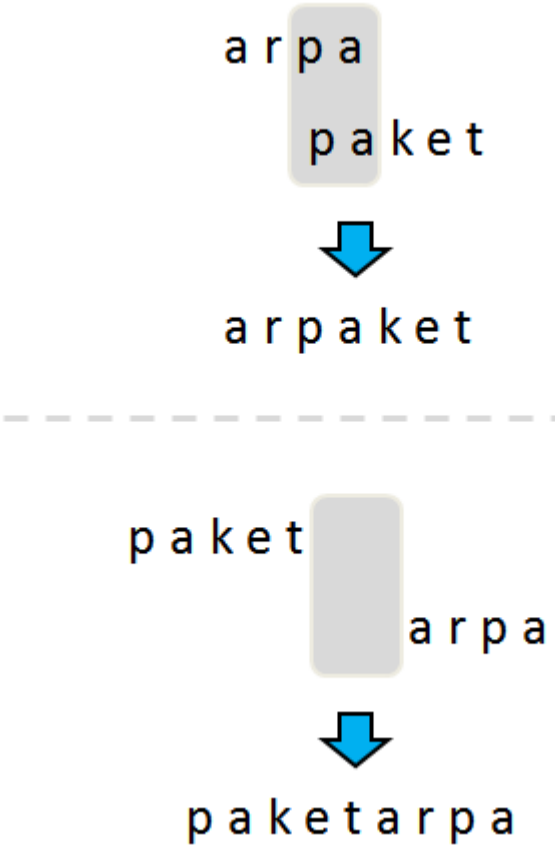
birleşim elde etmiş durumdayız. Ancak önce keten sonra paket kelimesini yan yana getirirsek, bu durumda bir kısaltma söz konusu olmayacaktır.



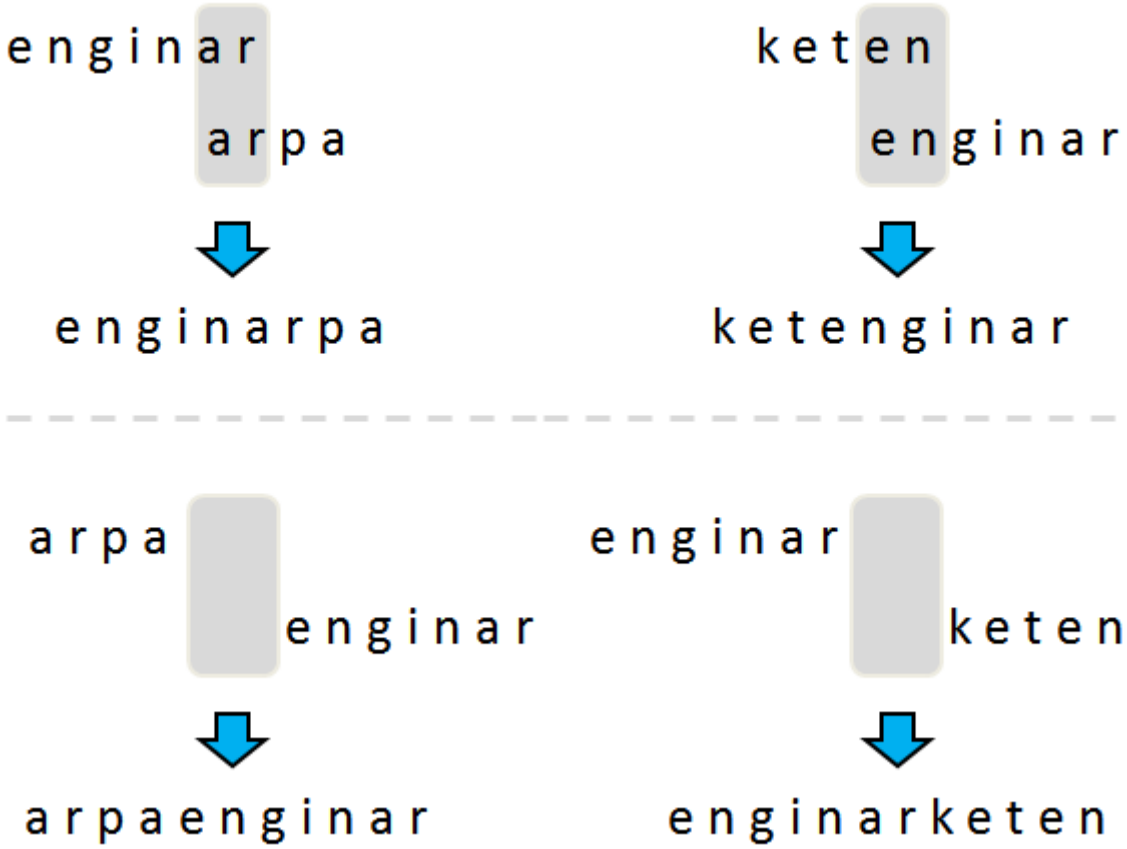
Dolayısıyla paket ile keten kelimeleri arasındaki ilişkiyi sayısal olarak anlamlandırmaya çalıştığımızda şunları söyleyebiliriz.

- **Keten Paket** sırası düşünüldüğünde arada hiç ortak birleşim harfi veya hecesi yoktur. Dolayısıyla **Keten**kelimesinden **Paket** kelimesine geçişinin değeri **0** olarak nitelendirilebilir.
- **Paket Keten** sırasına baktığımızda ise, **ket** hecesinin ortak olduğu bir durum söz konusudur. **ket** hecesi teke indirgendiğinde, iki kelime birleşimi önemli ölçüde kısaltılmış olmaktadır. Buna göre **Paket**kelimesinden **Keten** kelimesine geçişin maliyeti **3** olarak düşünülebilir*(3 harfli bir kısım kesilmiş olduğu için)*

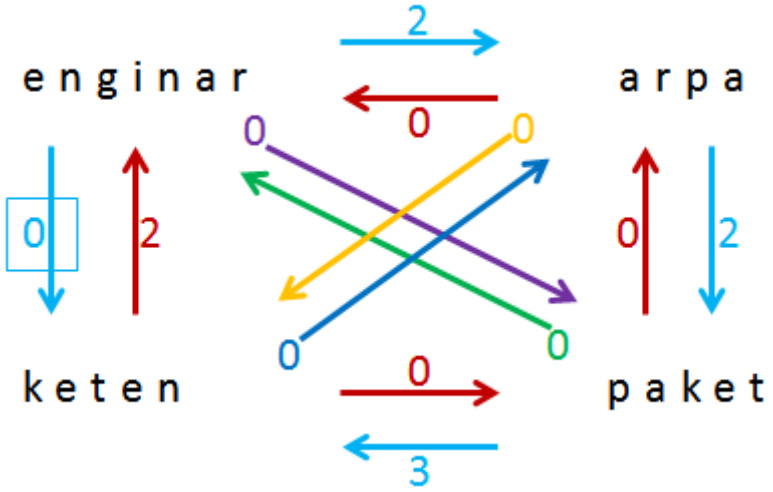
Diğer kelimeleri de göz önüne alarak devam edelim. İlk olarak **Arpa** ile **Paket**' e bir bakalım. Yukarıdaki tekniği göz önüne alacak olursak aşağıdaki görsellerde yer alan ilişkileri kurabiliriz.



Buna göre **paket** -> **arpa** geçişinin değeri **0** iken, **arpa** -> **paket** geçişinin değeri **pa** hecesinin değişmesi nedeniyle **2** olarak hesaplanabilir.
Şimdi de **Enginar**, **Arpa** ve **Keten** kelimeleri arasındaki ilişkiye bir bakalım. Çünkü bu 3 kelime arasında aynı değerlere sahip bir ilişki durumu söz konusudur. Aşağıdaki şekilde bu durum ifade edilmeye çalışılmıştır.



Burada **enginar** -> **arpa** birleşimi ile **keten** -> **enginar** birleşimlerinin sayısal ağırlıkları aynıdır(2). Karar vermek çok önemli değildir aslında. Nitekim ağırlık puanları eşittir ve iki birleşiminde tüm sözcük dizimine etkisi aynı olacaktır. Elimizde bulunan 4 kelimeyi ve aralarındaki ilişkileri düşündüğümüzde aşağıdaki şekilde olduğu gibi bir ağırlıklandırma yapabiliriz.

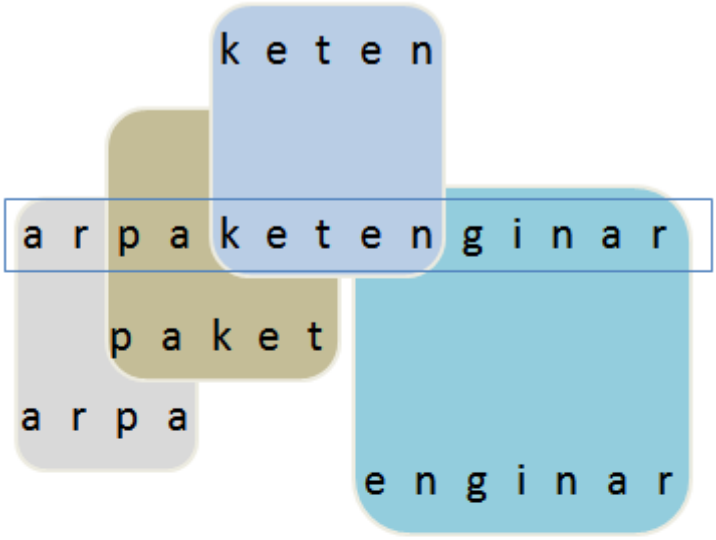


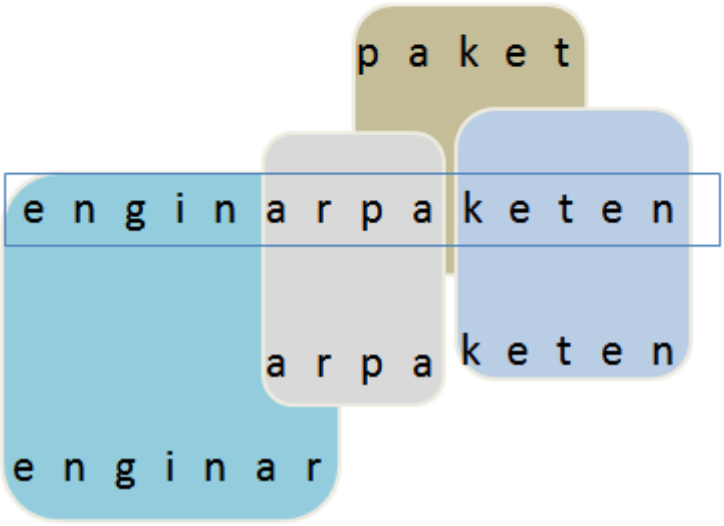
Sanırım bu şekil yardımıyla kelimeler arasındaki ilişkileri daha net görebilmekteyiz. Biraz renkli oldu ama olsun Sonuç olarak aşağıdaki iki kelime katarından birisini üretebiliriz.

arpaketenginar

veya

enginarpaketen



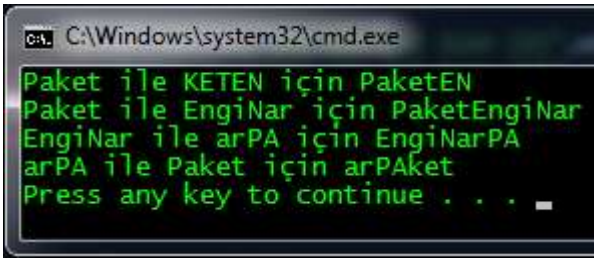


Herşey iyi güzel. Kafamızda kelimeleri bir şekilde birbirlerine bağladık. Peki ama bunun kodlamasını nasıl yapacağız? Sonuçta en önemli nokta aslında bu sıkıştırma şeklini bir şekilde kod tarafında üretebilmemizdir. İşe ufak bebek adımları ile başlamakta yarar var. Örneğin çok temel bir**genişletme metodu(Extension Method)** ile iki **string**' i ağırlık derecesine göre uygun bir biçimde birleştirmeyi düşünelim. İşte kod parçamız.

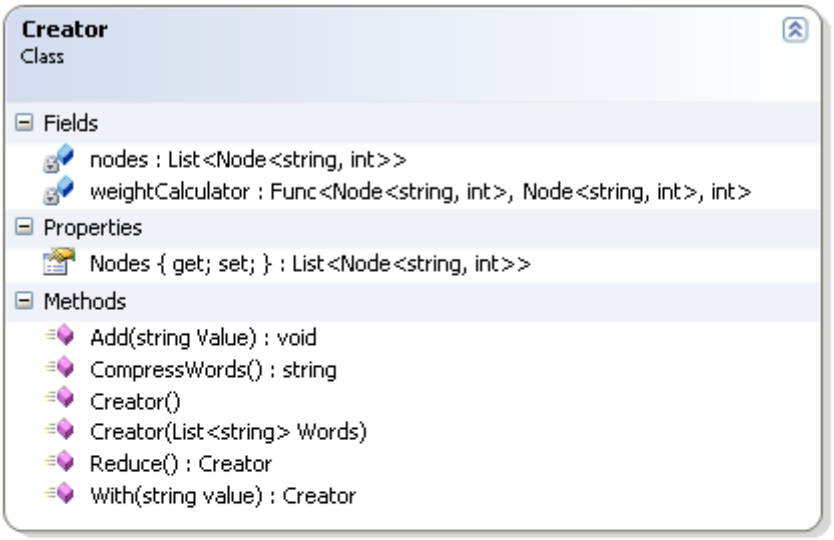
```
using System;
using System.Linq;
using System.Collections.Generic;
namespace EnKisaCumle
{
   class Program
   {
      static void Main(string[] args)
      {
         string[] words = { "Paket", "KETEN", "EngiNar", "arPA","demir" };
```

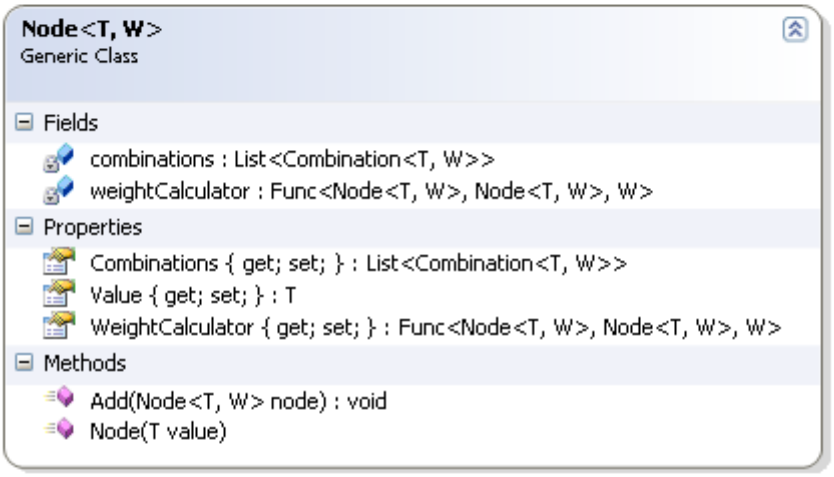
```
            Console.WriteLine("{0} ile {1} için {2}",words[0],words[1],
words[0].Combine(words[1]));
            Console.WriteLine("{0} ile {1} için {2}",words[0], words[2],
words[0].Combine(words[2]));
            Console.WriteLine("{0} ile {1} için {2}", words[2], words[3],
words[2].Combine(words[3]));
            Console.WriteLine("{0} ile {1} için {2}", words[3], words[0],
words[3].Combine(words[0]));
        }
    }
    public static class Extensions
    {
        public static string Combine(this string LeftWord,string RightWord)
        {
            int weight = FindWeight(LeftWord, RightWord);
            string result =string.Format("{0}{1}", LeftWord, RightWord.Substring(weight,
RightWord.Length - weight));
            return result;
        }
        private static int FindWeight(string wordLeft, string wordRight)
        {
            int maxLength = wordLeft.Length < wordRight.Length ? wordLeft.Length :
wordRight.Length;
            for (int i = 0; i < maxLength; i++)
            {
                if (wordLeft.EndsWith(wordRight.Substring(0, maxLength - i), true, null)) //
büyük küçük harf ayrımı yapmasın
                {
                    return maxLength - i;
                }
            }
            return 0;
        }
    }
}
```

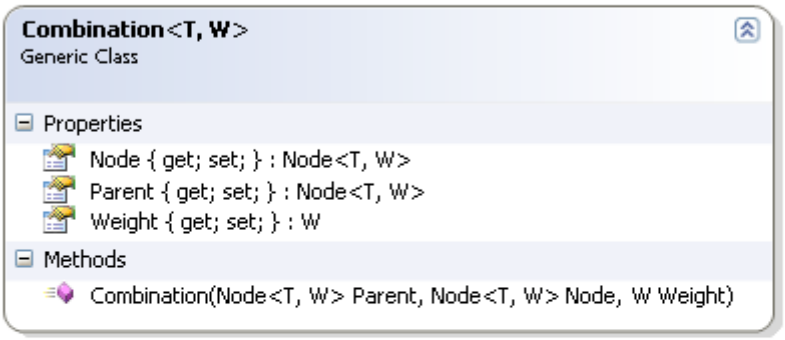
Uygulamamızda basit bir extension metod ile string birleştirme işlemi gerçekleştirilmektedir. Buna göre örneğin sonucu aşağıdaki gibi olacaktır.

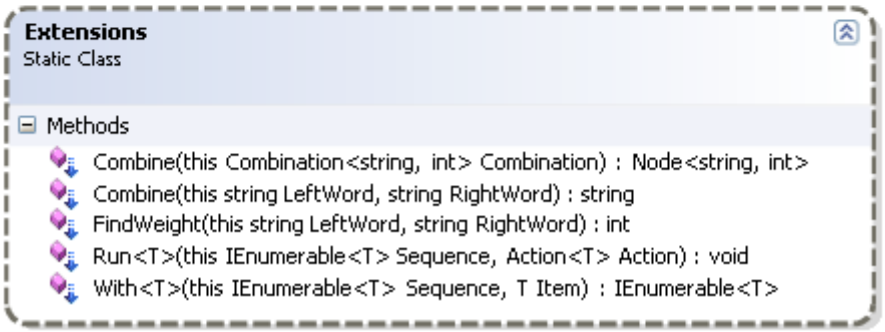


Görüldüğü üzere ağırlık derecelerine göre bazı kelimelerin birleşimi daha kısa olurken bazıları ise arka arkaya gelmektedir, nitekim ağırlıkları **0** dır.
Yazmış olduğumuz bu temel fonksiyon iki kelime işin içerisinde olduğunda işe yaramaktadır. Lakin kelime dizisinin tamamının ele alınması biraz daha farklı bir durumdur. Daha fazla kod ve daha karmaşık bir algoritma gerekmektedir. Yaptığım araştırmalar sonucunda aşağıdaki gibi bir kod parçasını toparlamayı başarabildim.


Creator
Class
Fields
nodes : List<Node<string, int>>
weightCalculator : Func<Node<string, int>, Node<string, int>, int>
Properties
Nodes { get; set; } : List<Node<string, int>>
Methods
Add(string Value) : void
CompressWords() : string
Creator()
Creator(List<string> Words)
Reduce() : Creator
With(string value) : Creator


Node<T, W>
Generic Class
Fields
combinations : List<Combination<T, W>>
weightCalculator : Func<Node<T, W>, Node<T, W>, W>
Properties
Combinations { get; set; } : List<Combination<T, W>>
Value { get; set; } : T
WeightCalculator { get; set; } : Func<Node<T, W>, Node<T, W>, W>
Methods
Add(Node<T, W> node) : void
Node(T value)


Combination<T, W>
Generic Class
Properties
Node { get; set; } : Node<T, W>
Parent { get; set; } : Node<T, W>
Weight { get; set; } : W
Methods
Combination(Node<T, W> Parent, Node<T, W> Node, W Weight)


Extensions
Static Class
Methods
Combine(this Combination<string, int> Combination) : Node<string, int>
Combine(this string LeftWord, string RightWord) : string
FindWeight(this string LeftWord, string RightWord) : int
Run<T>(this IEnumerable<T> Sequence, Action<T> Action) : void
With<T>(this IEnumerable<T> Sequence, T Item) : IEnumerable<T>

Sınıf diagramında(Class Diagram) da görüleceği üzere, kelimeler arası ilişkileri tasvir edebilmek adına bir **Node** listesini modelemeye çalışıyoruz. Bu **Node** listesi, bir kelime ve buna bağlı ne kadar diğer kelime kombinasyonu varsa, ağırlıkları ile birlikte **cover** etmeye çalışmaktadır. Diğer yandan**Node** listesini kullanarak uygun olabilecek bir birleşimi üretme fonksiyonelliğini, **Creator** isimli tip karşılamaktadır. Kodun biraz karmaşık olduğunu biliyorum. Lakin bu konu ile ilişkili olarak yaptığım araştırmalarda, en basite indirgeyebildiğim sıkıştırma algoritması bu oldu. İnanın bu anlamda ele alınan diğer algoritmalar *(NP Algoritmaları özellikle)* inanılmaz derecede karışık geliyor bana Ama yılmak yok! Lafı fazla uzatmadan kodumuzu paylaşalım.

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace EnKisaCumle
{
   class Program
   {
      static void Main(string[] args)
      {
         string[] words = { "Paket", "KETEN", "EngiNar", "arPA" };

         Creator graph = new Creator(new List<string>(words));
         Console.WriteLine(graph.CompressWords());
      }
   }

   // Yardımcı olacak genişletme metodlarımız
   public static class Extensions
   {
      // İki kelimeyi ağırlıkları mertebesinde birleştirir.
      public static string Combine(this string LeftWord, string RightWord)
      {
         int weight = FindWeight(LeftWord, RightWord);
         string result = string.Format("{0}{1}", LeftWord, RightWord.Substring(weight,
RightWord.Length - weight));
         return result;
      }

      // İki kelimenin birleşme ağırlık değerini hesaplar
      public static int FindWeight(this string LeftWord, string RightWord)
      {
```

```
            int maxLength = LeftWord.Length < RightWord.Length ? LeftWord.Length :
RightWord.Length;
            for (int i = 0; i <= maxLength; i++)
            {
               if (LeftWord.EndsWith(RightWord.Substring(0, maxLength - i), true, null)) //
büyük küçük harf ayrımı yapmasın
               {
                  return maxLength - i;
               }
            }
            return 0;
         }

         // Belirtilen Sequence koleksionu üzerindeki her bir elemanda Action temsilcisi ile
belirtilen fonksiyonelliğin çalıştırılmasını sağlar
         public static void Run<T>(this IEnumerable<T> Sequence, Action<T> Action)
         {
            foreach (var item in Sequence)
               Action(item);
         }

         // Reduce metodu içerisinde devreye girer
         public static IEnumerable<T> With<T>(this IEnumerable<T> Sequence, T
Item)
         {
            foreach (var t in Sequence)
               yield return t;

            yield return Item;
         }

         // Bir kelime kombinasyonu içerisindeki Node ve Parent' ı arasındaki birleşimi üretir
ve yeni bir Node olarak elde etmemizi sağlar
         public static Node<string, int> Combine(this Combination<string, int>
Combination)
         {
            return new Node<string,
int>(Combination.Parent.Value.Combine(Combination.Node.Value));
         }
      }
```

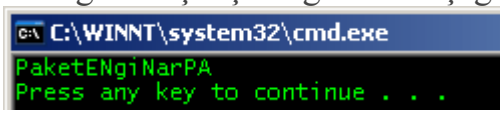
```
   // Bir kelime ve bu kelime ile diğerleri arasındaki ilişkiler ile ağırlıkları tutan temel tipimizdir
   public class Node<T, W>
   {
      private List<Combination<T, W>> combinations = new List<Combination<T, W>>();
      Func<Node<T, W>, Node<T, W>, W> weightCalculator;

      public Func<Node<T, W>, Node<T, W>, W> WeightCalculator
      {
         get
         {
            if (weightCalculator == null)
               weightCalculator = (n1, n2) => default(W);
            return weightCalculator;
         }
         set
         {
            weightCalculator = value;
         }
      }
      public List<Combination<T, W>> Combinations
      {
         get
         {
            return combinations;
         }
         set
         {
            combinations = new List<Combination<T, W>>(value);
         }
      }
      public T Value { get; private set; }
      public Node(T value)
      {
         Value = value;
      }

      public void Add(Node<T, W> node)
      {
```

```
            var combination = new Combination<T, W>(this, node, WeightCalculator(this,
node));
            combinations.Add(combination);
        }
    }

    // Kelime kombinasyonlarını Parent ve Current Node bazında ağırlıkları ile birlikte
saklar.
    public class Combination<T, W>
    {
        public Node<T, W> Parent { get; private set; }
        public Node<T, W> Node { get; private set; }
        public W Weight { get; private set; }

        public Combination(Node<T, W> Parent, Node<T, W> Node, W Weight)
        {
            this.Parent = Parent;
            this.Node = Node;
            this.Weight = Weight;
        }
    }


    // Asıl işlemleri üstlenen tipimiz. Kelimelere ait node listesine yeni örnekler eklenmesi,
sıkıştırma yapılması, olası kombinasyonların azaltılması gibi fonksiyonellikleri üstlenir.
    public class Creator
    {
        List<Node<string, int>> nodes = new List<Node<string, int>>();
        Func<Node<string, int>, Node<string, int>, int> weightCalculator;

        public Creator()
        {
            weightCalculator = (node, newNode) =>
            {
                return node.Value.FindWeight(newNode.Value);
            };
        }

        public Creator(List<string> Words)
        {
            Words.Run(Add);
```

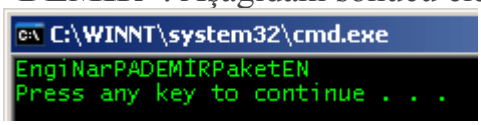
```
}

public List<Node<string, int>> Nodes
{
   get
   {
      return nodes;
   }
   set
   {
      nodes = new List<Node<string, int>>(value);
   }
}

public void Add(string Value)
{
   if (!nodes.Exists(n => n.Value == Value))
   {
      var newNode = new Node<string, int>(Value)
      {
         WeightCalculator = weightCalculator
      };

      nodes.Run(node =>
      {
         newNode.Add(node);
         node.Add(newNode);
      });

      nodes.Add(newNode);
   }
}

public Creator With(string value)
{
   Add(value);
   return this;
}

public Creator Reduce()
{
```

```
            if (nodes.Count <= 1)
                return this;

            var combinations = from n in nodes
                               from e in n.Combinations
                               select e;
            var combination = combinations.First(n => n.Weight == combinations.Max(e =>
e.Weight));

            return new Creator(nodes.Select(n => n.Value)
                                    .Where(str => str != combination.Parent.Value && str !=
combination.Node.Value)
                                    .With(combination.Combine().Value).ToList());
        }

        public string CompressWords()
        {
            Creator graph = this;
            while (graph.nodes.Count > 1)
            {
                graph = graph.Reduce();
            }

            return graph.nodes[0].Value;
        }
    }
}
```

Örneğimizi çalıştırdığımızda aşağıdaki gibi bir sonuç elde ederiz.



Görüldüğü üzere kelimeleri istediğimiz şekilde birleştirebilidik.
Uygulama kodunu kavrayabilmek adına **Debug** ederek adım adım ilerlemenizi öneririm. Lakin içeride makaleyi yazdığım tarih itibariyle halen çözemediğim bazı **bug**' lar bulunmakta. Söz gelimi kelime dizimize **5nci** bir içeriği eklediğimizi düşünelim. "**DEMİR"**. Aşağıdaki sonucu elde ederiz.



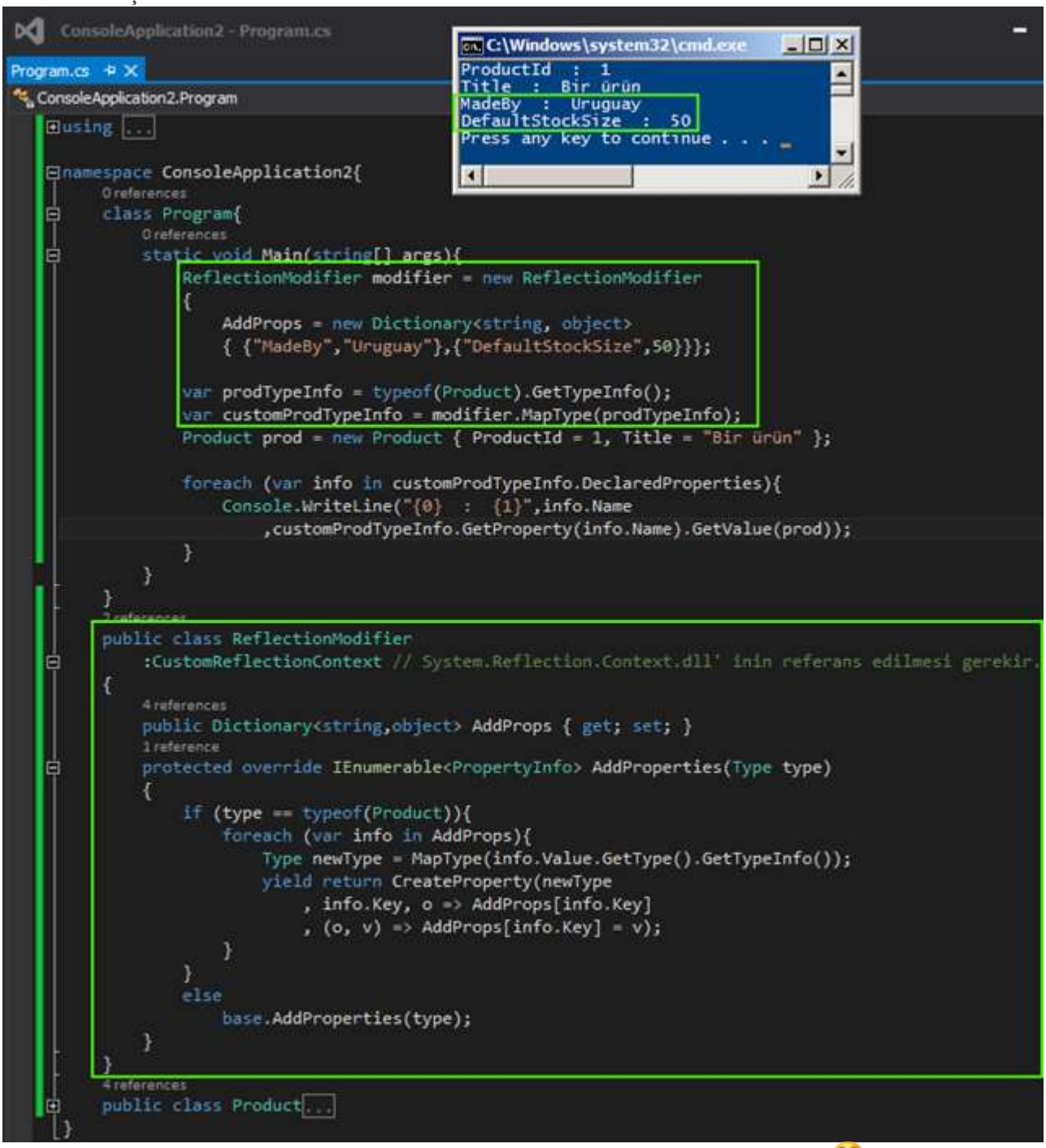
Başarılı bir birleştirme işlemi yapılmış gibi görünüyor değil mi? Aslında biraz daha dikkat edersek, hiç bir kelime ile ortak notkası olmayan **DEMİR** sözcüğünün en sona veya en başa alınmasının daha doğru olduğunu görebiliriz. Çünkü bu durumda

EngiNarPAketENDEMİR veya DEMİREngiNarPAKetEN
sonuçlarını elde edebiliriz. Yani **PA** hecesinin teke indirgenmesi söz konusudur. Dolayısıyla kodun gözden geçirilmesi, algoritmanın temizlenerek en uygun sonucun elde edilmesi için gerekli müdahalelerin yapılması gerekmekte. Burada iş biraz da sizlere düşüyor. Her ne kadar mükemmel şekilde çalışan bir algoritma olmasa da, kendi adıma kelime sıkıştırma konusunda epey bir fikir verdiğini, en azından kağıt üzerinde kelimeler arası ilişkilerin bulunması noktasında nasıl hareket edilebileceğini hem kendi adıma hem de değerli okurlarım adına anlamış bulunmaktayım. Tabi önmeli bir eksiğimiz daha var. Sıkıştırılan metni nasıl geri çözümleyeceğimiz. Buna bir çözüm getirebilir misiniz? Tekrardan görüşünceye dek hepinize mutlu günler dilerim.
EnKisaCumle.zip (59,15 kb)

Tek Fotoluk İpucu 104 : CustomReflectionContext ile Tipe Özellike Kazandırmak

Perşembe, 29 Ağustos 2013 19:18 by bsenyurt

Bir tipin çalışma zamanında **Reflection** ile yakalanabilen özelliklerine ilaveler yapmak ister miydiniz? Aslında bunun çok kolay bir yolu var. Tek yapmanız gereken **CustomReflectionContext** tipinden yeni bir sınıf üretmek ve bunu aşağıdakine benzer bir şekilde kullanmak



Bir başka ipucunda görüşmek dileğiyle, hepinize mutlu günler dilerim

TFS OData Desteği

Pazartesi, 26 Ağustos 2013 07:35 by bsenyurt

[Orjinal Yazım Tarihi 3/20/2013]

Merhaba Arkadaşlar,

Çoğu zaman geliştirilen yazılım ürünleri ile farklı profilden insanları ortak bir payda da buluşturmayı hedefleriz. Farklı özelliklere sahip insanları, ürüne nasıl katabileceğimizi keşfetmeye çalışırız. Tabi geliştirilen ürünün hedef kitlesi de burada önemli bir rol oynar.

Bazı ürünlerin arayüzlerinin son derece basit tasarlanması yeterli iken bazılarında ise tam tersi bir durum söz konusudur.

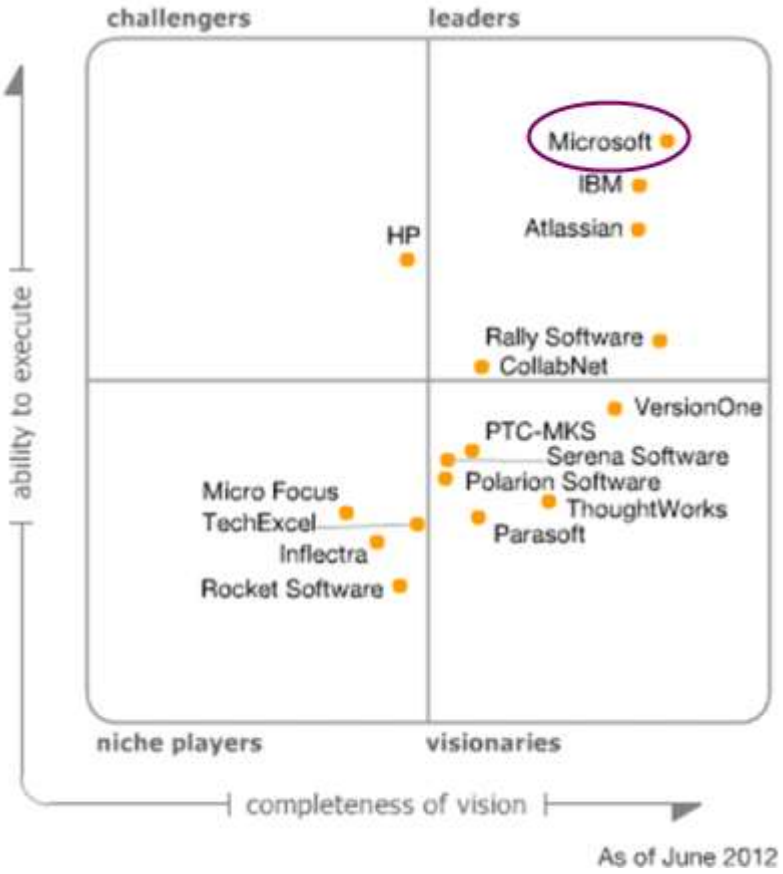
Hangisi olursa olsun kullanıcı bir insan olarak düşünüldüğünde çok da fazla zorlanmamalı veya kolayca adapte olabilmelidir. Ne kadar kolay kullanılırsa, hedef kitle içerisinde o kadar fazla sayıda farklı profile de ulaşılabilinir. Ancak bazı hallerde ürünün hedef kitlesi o kadar dağınıktır ki, hepsini çekebilmek ya da bir başka deyişle kazanabilmek için yapılan genişletmeler yeterli gelmeyebilir. Böyle bir durumda çevreye şu mesajı vermeniz gerekebilir;

Ey ahali…Bu gördüğünüz, ürünümüzün dışarıya açılmış olan servisi/servisleri/sdk'sı/api'si. Buyrun istediğiniz gibi uyarlayın, kullanın. Sonuçta ürünümüzün yaşamının bir parçası olabileceksiniz

Özellikle **ALM*(Application Lifecycle Management)*** gibi geniş konuların uygulandığı ürünlerin değerlendirildiği firmalar ve kalabalık ekipleri düşünüldüğünde, bu heterojenlik kendini iyiden iyiye hissettir. Dolayısıyla ürünün geliştiriciler açısından ne kadar ve nasıl **genişletilebileceği** önem kazanır.

Ekipleriniz içindeki profilleri düşünün! Yazılımcılar IDE' leri, iş analistleri Word dokümanlarını, Müdür' ler web browser üzerinden erişilebilen raporları, Release Manager' lar Team Explorer' ı, Proje Yöneticileri Ms Project' i, CIO' lar ise ürünlerinin hangi sprint' ler de olup ne kadarlık işlerinin kaldığını okuyabildikleri e-postaları, sever. Listeyi uzatmak mümkün

Bu felsefeden baktığımızda bence **Microsoft**' un **Team Foundation Server** ürünü epey önemli bir noktada yer alıyor. Hatta **Gartner**' ın **Application Lifecycle Management** konusundaki bi raprunda yer alan **Magic Quadrant** grafiği de, bunu doğrular nitelikte. Kabiliyet ve sunulan vizyon açısından **Microsoft** liderler arasında en iyi noktada yer alıyor diyebiliriz *(Rapor hakkında detaylı bilgiye bu adresten ulaşabilirsiniz)*



Peki TFS takımı bunu nasıl başarıyor?

Bildiğiniz üzere **Team Foundation Server** ailesi içerisinde, **.Net** tabanlı kullanılabilen **ObjectModeller*(Client Object Model, Server Object Model, Build Process Object Model)***, yabancı ürünlerin entegre olabilmesini sağlayan **Provider**' lar ***(MSSCCI Provider ve Team Explorer Everywhere)***, servis bazlı entegrasyon için **XML Web hizmetleri** var. Ayrıca bilindiği üzere **TFS** in**Cloud** tabanlı çalışan bir başka verisyonu daha bulunmakta.

Cloud Tabanlı TFS

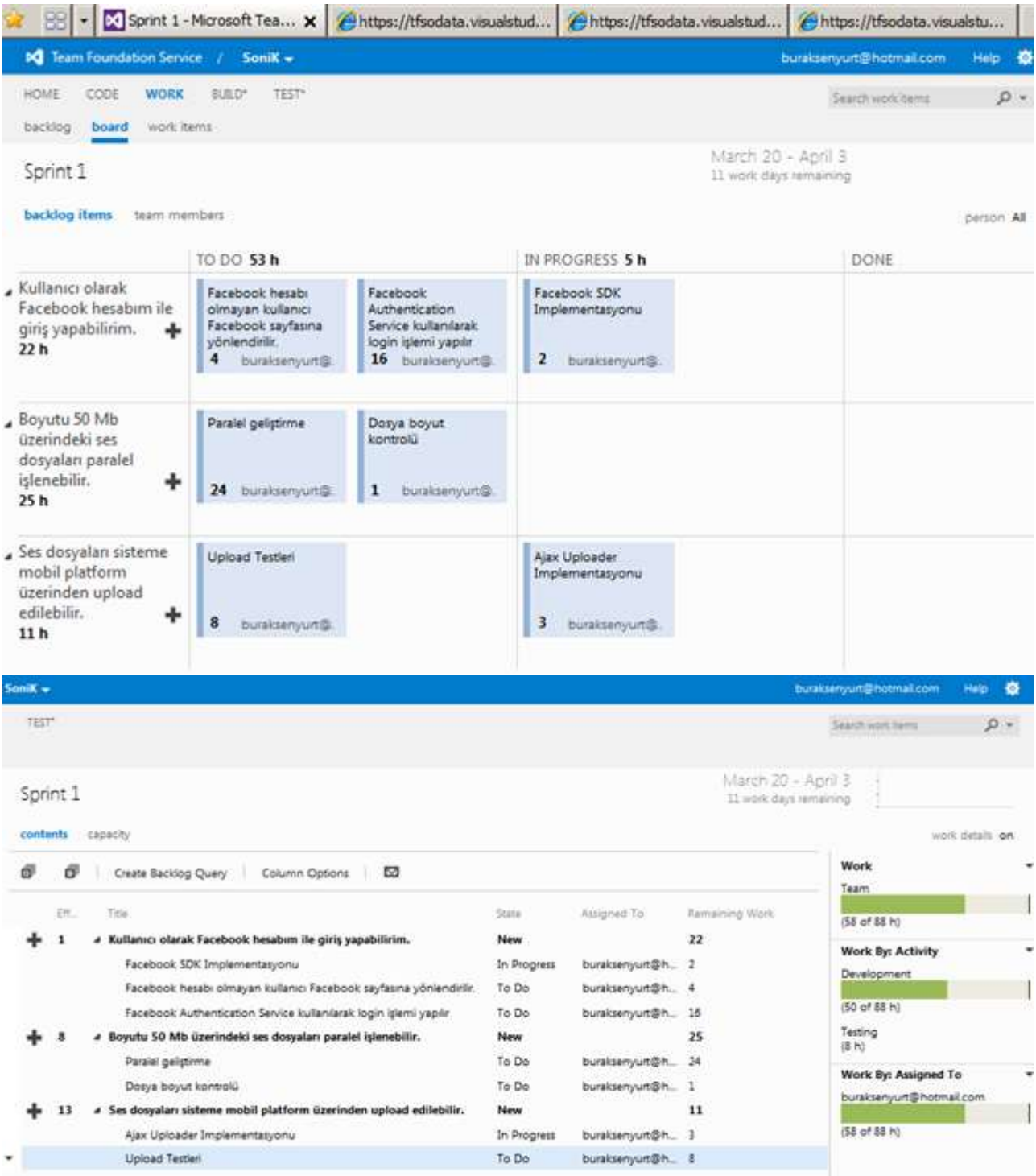
Team Foundation Server bilindiği üzere bir süredir **Cloud Service** olarak da hizmet vermekte. **5 Windows Live ID** hesabına kadar ücretsiz kullanılabilen hizmet, sunucu modelli kurulum sonucu yapılabilen hemen herşeyi karşılamakta. **Scrum, MSF, CMMI** gibi şablonları doğrudan destekleyen servis, **TFS Web Access** arayüzü ile de oldukça kolay bir kullanıma sahip. **Visual Studio** ailesine,**Excel** gibi ofis ürünlerine kolayca bağlanabilmekte. Hatta son zamanlarda **Git** ile olan entegrasyonu sayesinde, **Git** fanatiği geliştiricilerin de dikkatini çekmeyi başardı.
Bir bulut servisi olduğu için, **TFS** ortamının kurulumunu düşünmemize gerek yok. Sadece abone oluyor veya lisanslı kullanıcı iseniz kira bedelini ödüyorsunuz. Kurulumu düşünmüyor olmanız beraberinde **ölçeklenebilirlik*(Scalability)*,** sunucu performansı,

donanım alımı, personel istihdamı gibi mevzuları da düşünmemize gerek olmadığı anlamına gelmekte.
Pek tabi bazı kurumlar halen daha bulut servislerine temkinli yaklaşmakta ve hatta doğrudan geri çevirmekte. Örneğin ülkemizdeki BDDK gibi kurumlar, bankaların bu tip bulut tabanlı yapılar ile çalışmalarını ve bilgi alışverişinde bulunmalarını epeyce sorgulamakta ve kolay kolay izin vermemekte.
Açıkçası askeri ve stratejik açıdan düşünürsek ülke ekonomisine ait değerli bilgilerin 3ncü parti sunucularda, ülke dışında tutmak çok da doğru değil bana kalırsa.
http://tfs.visualstudio.com adresinden ulaşılabilen hizmete son aylarda eklenen ve halen geliştirilme aşamasında olan önemli bir yenilikte **OData*(Open Data Protocol)*** servis desteği. **XML*(eXtensible Markup Language)*** üzerine oturan ve **URL** bazlı sorgulama yetenekleri tanıyan bu protokol, bir dünya standardı. Standardın hedefi ise veri odaklı yayınlayıcılar***(Publishers)***.
Hal böyle olunca herhangibir veri kaynağının, özellikle internet ortamı üzerinden **OData** protokolüne göre sorgulanabilmesi mümkün hale geliyor. Bunun tam karşılığı ise, platform bağımsızlıktan başka bir şey değil. Ama bu son derece önemli bir yetenek. Nitekim bulut üzerinde koşan **TFS** hizmetini kullanabilecek istemcileri her hangi bir platform için geliştirebileceğimiz anlamına gelmekte.
OData protokolünün ön gördüğü veri odaklı sorgular bilindiği üzere tamamen **URL** bazlı olarak çalışmakta. Bu nedenle bulut **TFS** servisi üzerinde duran bilgileri basit bir tarayıcı uygulamayı kullanarak sorgulayabilirsiniz de. Bunun sonucu olarak dilerseniz **TFS** projenizin bazı raporlarını veya yönetsel arabirimi özelliklerini, örneğin mobil cihazınıza kadar indirebilirsiniz. *(Bu konu ile ilişkili olarak Nisha Singh' in Windows 8 uygulamasına bir göz atmanızı öneririm)*
Gelelim bu yazımızdaki konumuza. Hiç kod yazmayacak ve geliştirme yapmayacağız aslında  Bir tarayıcı uygulama, **tfs.visualstudio.com** üzerindeki hesabımız ve **OData** servislerinden yararlanarak sorgulamalar gerçekleştireceğiz.

Seddulbahir

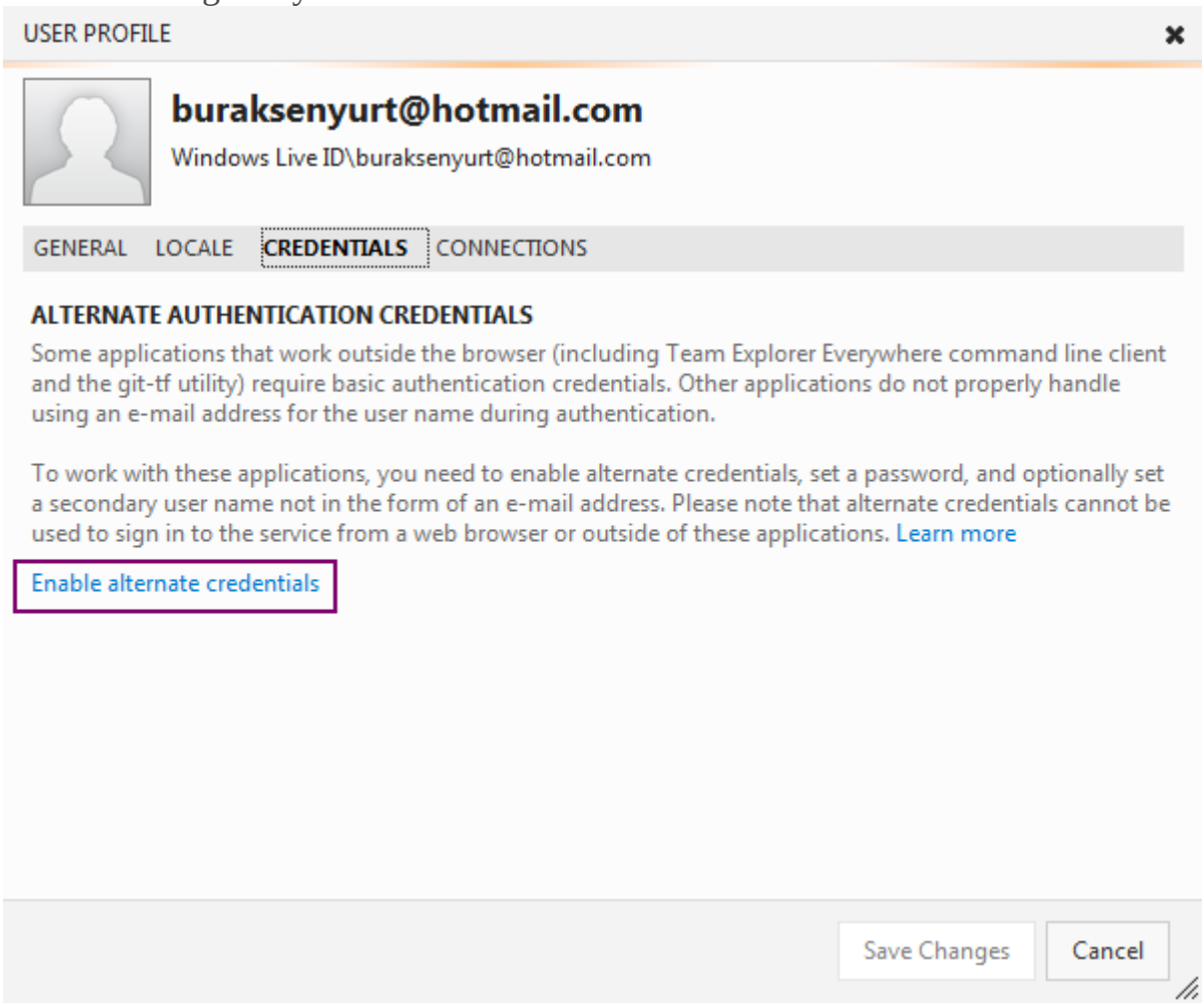
Bu amaçla ben **seddulbahir.visualstudio.com** adresinde konuşlandırılmış olan ve sahibi olduğum**Team Project Collection** alanını kullanıyor olacağım. **OData** örnekleri için çok basit olarak **SoniK**isimli uydurmasyon bir **Team Project** oluşturdum. Söz konusu proje **Scrum 2.2** şablonuna göre kullanılmakta. Şimdilik tek üyesi benim ve tüm **Task**' lar üzerimde 
tfs.visualstudio.com servisinin en güzel yanlarından birisi de, son güncellemeleri haberiniz olmasa dahi hızla ve ilk elden implemente ediyor oluşu. Söz gelimi Scrum' ın yeni bir versiyonunun çıktığını fark etmemiş olabilirsiniz. Ama bulut üzerinde bu güncelleme çıktığı gibi entegre edilmiştir de.
Tabi **OData** servisleri ile bir **Team Project'** in sorgulanması denince dikkatler hemen **Work Item**içeriklerine çevrilecektir. Yani **Product Backlog**

Item, **Task**, **Test Case**, **Bug**, **Impediment** gibi öğelere*(Scrum için söz konusu olan bu Work Item çeşitleri, seçilen süreç şablonuna göre elbetteki değişiklik gösterebilir)* Bu nedenle **SoniK** isimli proje içerisine aşağıdaki ekran görüntüsünde yer alan bazı **Work Item'** ları ekledim ve bunları şimdilik, 2 haftalık süreye sahip olan **Sprint 1** içerisinde değerlendirmeye aldım. Görüldüğü gibi **TO DO**' dan **IN PROGRESS**'e aldığım iki **Task**' ım var



Sorgulama işlemine başlamadan önce yapılması gereken küçük bir hazırlık daha var. **OData**servislerini etkinleştirmek için **TFS** hesabımızın profil özelliklerinden **Enable**

Aternate Credentials seçeneğini aktif hale getirmemiz ve bir kullanıcı adı ile şifre belirlememiz gerekiyor.



Bu işlemin ardından https://tfsodata.visualstudio.com/defaultcollection adresine girerek başlama vuruşunu yapabiliriz  **domainAdı\kullanıcıAdı** ve **şifre** ile giriş yapabiliriz. Örneğin
benim **TFS**projem **seddulbahir.visualstudio.com** olduğundan, **seddulbahir\AlternatifKullanıcıAdı** ve**şifre** ile giriş yapmam gerekiyor. **defaultcollection** adresini sorguladığımızda standart bir **OData**servisinden beklediğimiz sonuçlar ile karşılaşırız. Bize **TFS** hizmeti için sorgulanabilir olan **Entity**adlarının adreslerini içeren bir sayfa üretilecektir. Aşağıdaki ekran görüntüsündeki gibi.

Dikkat edileceği üzere kullanıcılardan **Area**' lara, **Work Item**' lardan, **Team Project Collection**içerisindeki **Team Project**' lere, **Build** tanımlamalarından, **Iteration**' lara kadar sorgulanabilecek oldukça geniş bir yelpaze söz konusudur. Şimdi dilerseniz örnek projemiz için bir kaç **OData**sorgusu icra edelim ve sonuçları görmeye çalışalım.

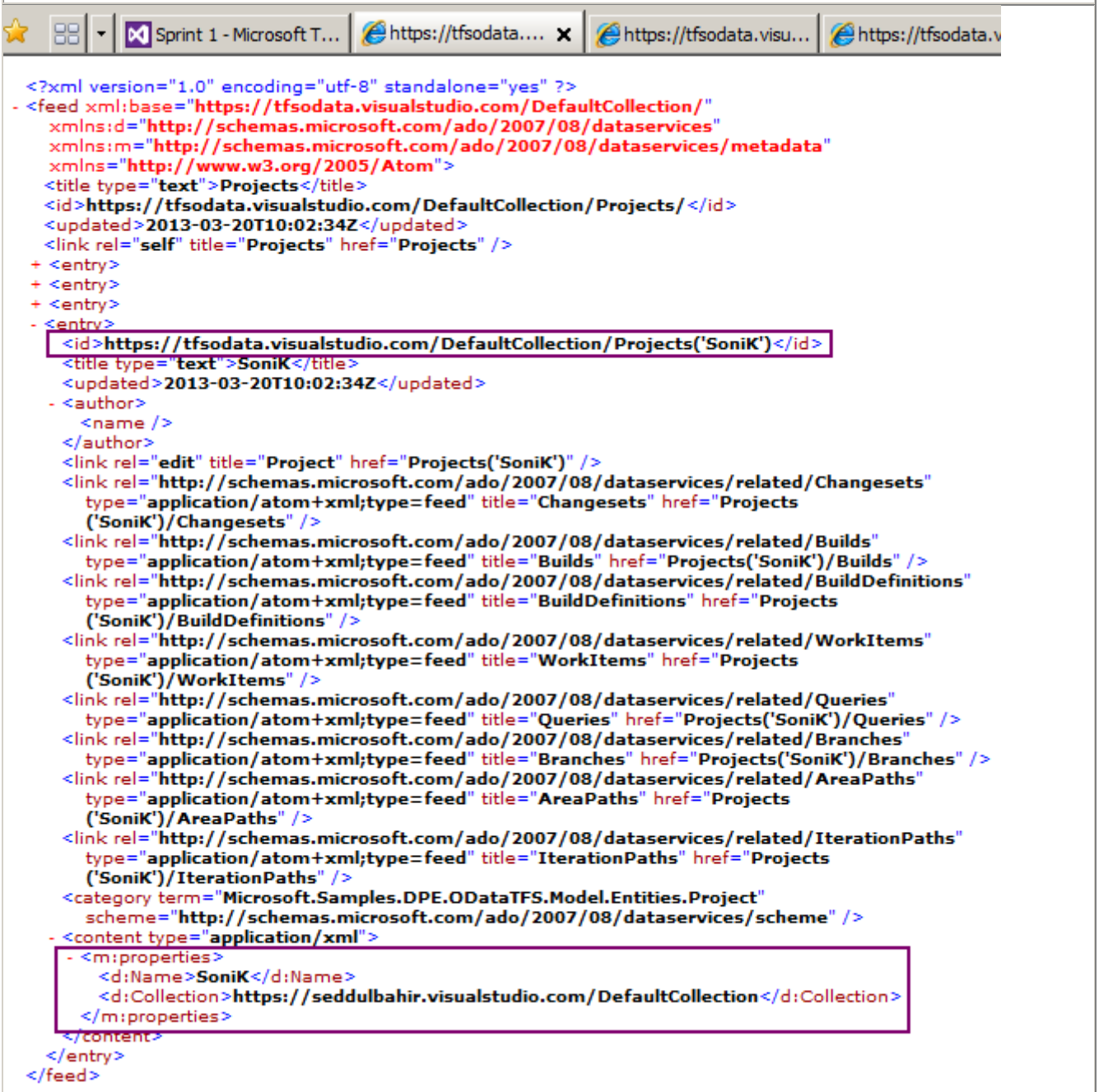
Örnek Sorgular

Aşağıdaki tabloda sorguya ait **URL** ifadeleri, bazı kısa açıklamalar ve **SoniK** için üretilen sonuçlara ait ekran görüntüleri yer almaktadır.

Team Project Collection içerisinde yer alan Team Project öğelerinin çekilmesi

https://tfsodata.visualstudio.com/DefaultCollection/Projects

Yukarıdaki sorgu hesabımıza ait Team Project Collection içerisinde ne kadar Team project var ise döndürür. Dikkat çekici **entry** özelliklerinden birisi, belirli bir projeye erişebilmek için **id** elementi ile gelen **URL**değeridir.

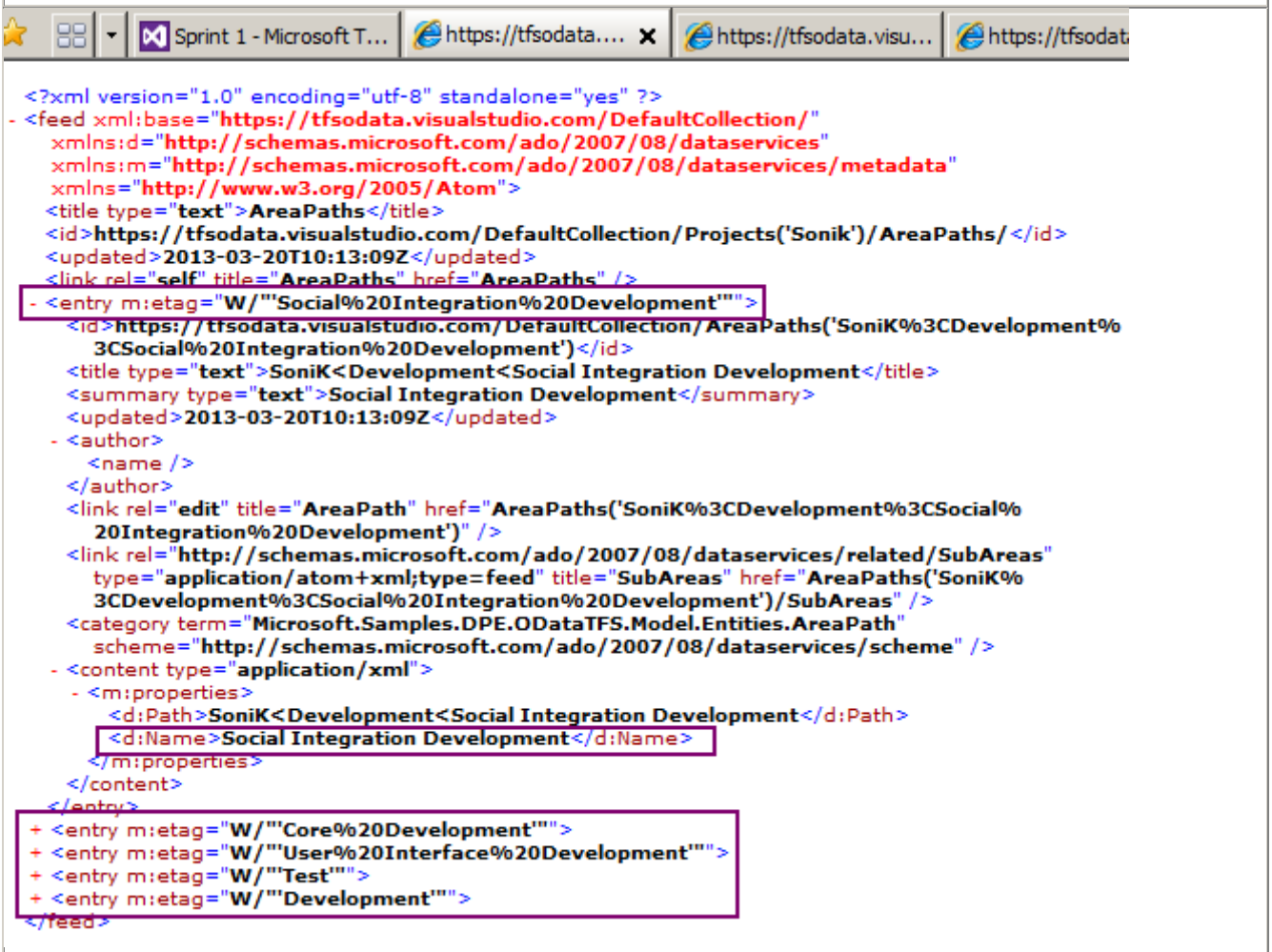


Belirli bir Team Project içerisinde tanımlanmış olan Area bilgilerinin çekilmesi

https://tfsodata.visualstudio.com/DefaultCollection/Projects('Sonik')/AreaPaths

Bir proje içerisinde pek çok **Area** tanımlanmış olabilir. Genellikle aynı projede birden fazla takımın kendilerine ait **Area**' lar üzerinde yetkilendirilerek çalışması gibi ihtiyaçlarda ideal

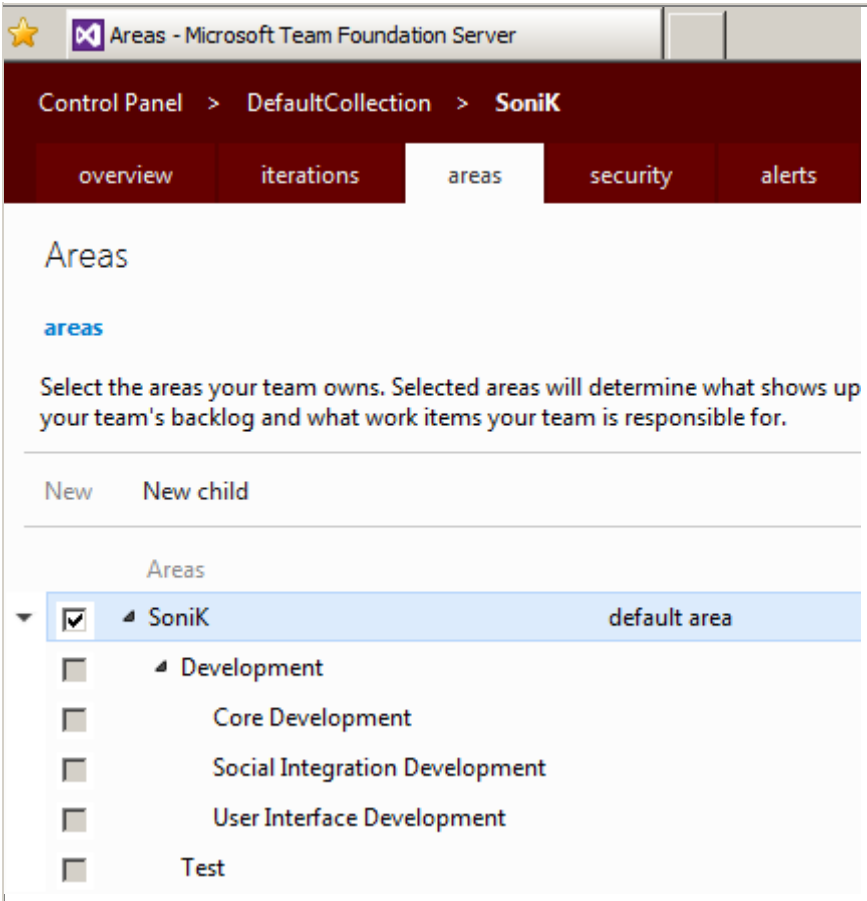
bir çözümdür. Sorguda önce Projects üzerinden SoniK' e gidilmiş ve **AreaPaths** alt entity içeriği talep edilmiştir.

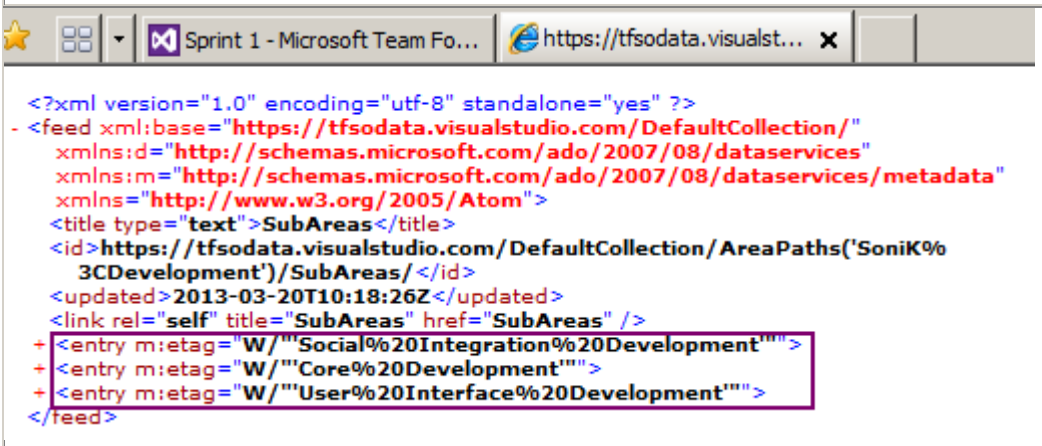


Bir Area' nın altında yer alan alt Area' ların çekilmesi

https://tfsodata.visualstudio.com/DefaultCollection/AreaPaths('SoniK%3CDevelopment')/SubAreas

Bazı durumlarda bir **Area** altında birden fazla **Child Area** açılmış olabilir. Hatta bu, bir kaç seviyelendirme şeklinde yapılmış da olabilir. Örnek projede aşağıdaki gibi bir **Area** yapısı söz konusudur ve yukardaki**OData** sorgusu ile bu içerik **XML** formatında elde edilebilir.

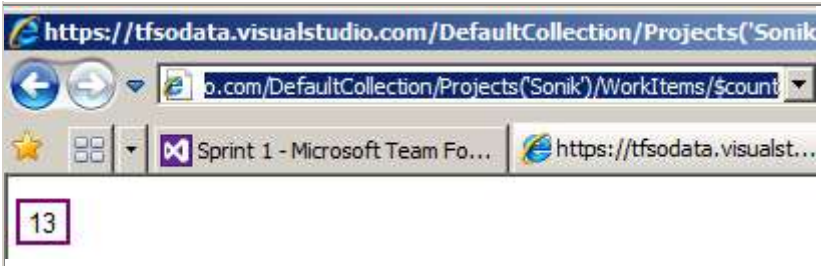




Bir Team Project için söz konusu olan Work Item Öğelerinin toplam sayısını bulmak

https://tfsodata.visualstudio.com/DefaultCollection/Projects('Sonik')/WorkItems/$count

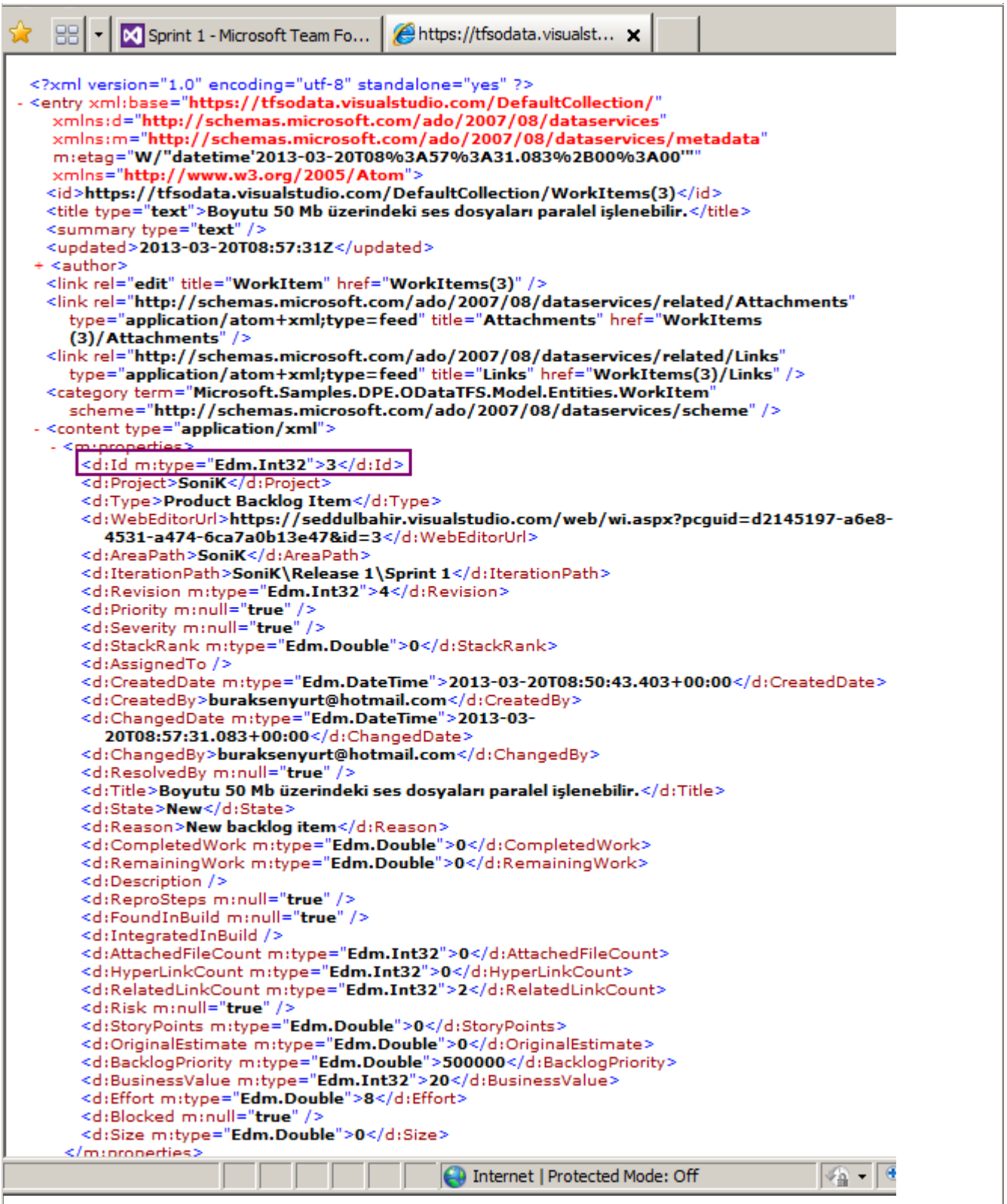
Örneğin **Sonik** isimli **Team Project** içerisinde yer alan toplam **work item** sayısını **count ODataf**onksiyonundan yararlanarak bulabiliriz.



Belirli bir Id değerine sahip Work Item' ın elde edilmesi

https://tfsodata.visualstudio.com/DefaultCollection/Projects('Sonik')/WorkItems(3)

Bu sorgu ile **3 numaralı ID değerine sahip Work Item** bilgisi elde edilmektedir. Tabi çok fazla özellik olduğundan element sayısı da epeyce fazladır. Azaltmak için bir sonraki örnekte olduğu gibi **projectiontype** kullanımı tercih edilebilir.



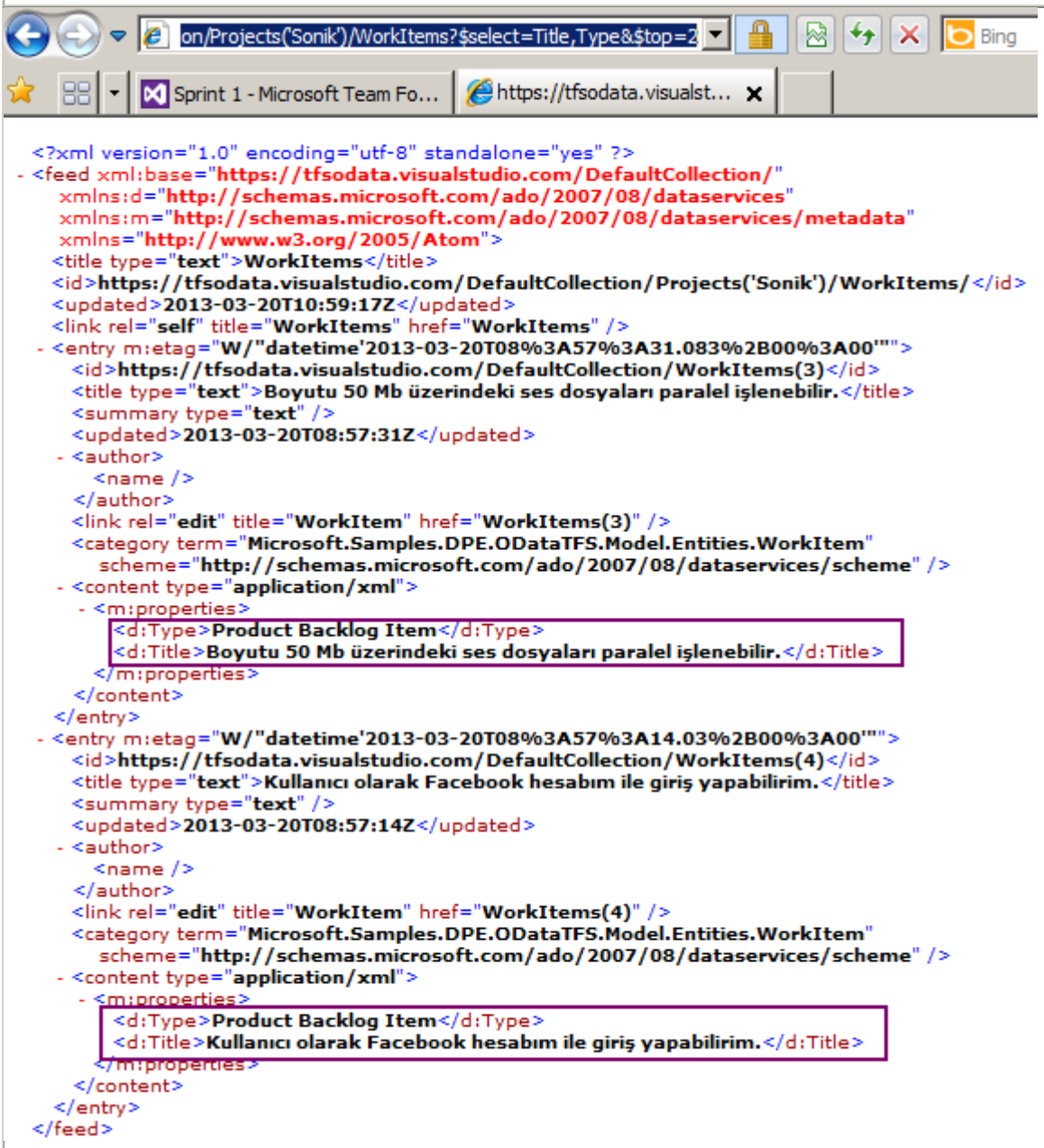

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <entry xml:base="https://tfsodata.visualstudio.com/DefaultCollection/"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    m:etag="W/"datetime'2013-03-20T08%3A57%3A31.083%2B00%3A00'""
    xmlns="http://www.w3.org/2005/Atom">
  <id>https://tfsodata.visualstudio.com/DefaultCollection/WorkItems(3)</id>
  <title type="text">Boyutu 50 Mb üzerindeki ses dosyaları paralel işlenebilir.</title>
  <summary type="text" />
  <updated>2013-03-20T08:57:31Z</updated>
 + <author>
  <link rel="edit" title="WorkItem" href="WorkItems(3)" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Attachments"
    type="application/atom+xml;type=feed" title="Attachments" href="WorkItems
    (3)/Attachments" />
  <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Links"
    type="application/atom+xml;type=feed" title="Links" href="WorkItems(3)/Links" />
  <category term="Microsoft.Samples.DPE.ODataTFS.Model.Entities.WorkItem"
    scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
 - <content type="application/xml">
   - <m:properties>
      <d:Id m:type="Edm.Int32">3</d:Id>
      <d:Project>SoniK</d:Project>
      <d:Type>Product Backlog Item</d:Type>
      <d:WebEditorUrl>https://seddulbahir.visualstudio.com/web/wi.aspx?pcguid=d2145197-a6e8-
        4531-a474-6ca7a0b13e47&id=3</d:WebEditorUrl>
      <d:AreaPath>SoniK</d:AreaPath>
      <d:IterationPath>SoniK\Release 1\Sprint 1</d:IterationPath>
      <d:Revision m:type="Edm.Int32">4</d:Revision>
      <d:Priority m:null="true" />
      <d:Severity m:null="true" />
      <d:StackRank m:type="Edm.Double">0</d:StackRank>
      <d:AssignedTo />
      <d:CreatedDate m:type="Edm.DateTime">2013-03-20T08:50:43.403+00:00</d:CreatedDate>
      <d:CreatedBy>buraksenyurt@hotmail.com</d:CreatedBy>
      <d:ChangedDate m:type="Edm.DateTime">2013-03-
        20T08:57:31.083+00:00</d:ChangedDate>
      <d:ChangedBy>buraksenyurt@hotmail.com</d:ChangedBy>
      <d:ResolvedBy m:null="true" />
      <d:Title>Boyutu 50 Mb üzerindeki ses dosyaları paralel işlenebilir.</d:Title>
      <d:State>New</d:State>
      <d:Reason>New backlog item</d:Reason>
      <d:CompletedWork m:type="Edm.Double">0</d:CompletedWork>
      <d:RemainingWork m:type="Edm.Double">0</d:RemainingWork>
      <d:Description />
      <d:ReproSteps m:null="true" />
      <d:FoundInBuild m:null="true" />
      <d:IntegratedInBuild />
      <d:AttachedFileCount m:type="Edm.Int32">0</d:AttachedFileCount>
      <d:HyperLinkCount m:type="Edm.Int32">0</d:HyperLinkCount>
      <d:RelatedLinkCount m:type="Edm.Int32">2</d:RelatedLinkCount>
      <d:Risk m:null="true" />
      <d:StoryPoints m:type="Edm.Double">0</d:StoryPoints>
      <d:OriginalEstimate m:type="Edm.Double">0</d:OriginalEstimate>
      <d:BacklogPriority m:type="Edm.Double">500000</d:BacklogPriority>
      <d:BusinessValue m:type="Edm.Int32">20</d:BusinessValue>
      <d:Effort m:type="Edm.Double">8</d:Effort>
      <d:Blocked m:null="true" />
      <d:Size m:type="Edm.Double">0</d:Size>
    </m:properties>
```


Work Item' lardan ilk 2sinin sadece Title ve Type bilgilerini çekmek

https://tfsodata.visualstudio.com/DefaultCollection/Projects('Sonik')/WorkItems?$select=Title,Type&$top=2

Burada aslında bir **projection** kullanımı söz konusudur. **select** anahtar kelimesini takiben,

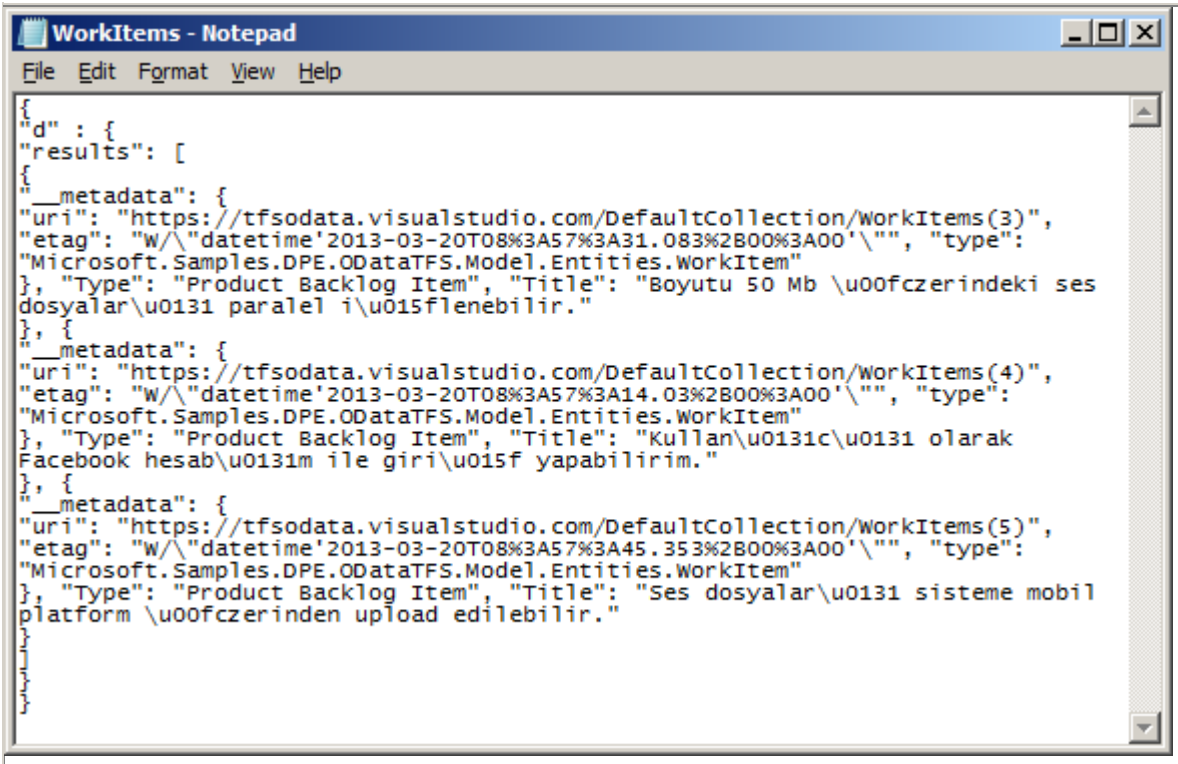
çekilmek istenen özelliklerin adları verilmiştir. **top** anahtar kelimesi ile de kaç tane element çekileceği ifade edilir.



Sorgu çıktısını JSON(JavaScript Object Notation) formatında elde etmek

https://tfsodata.visualstudio.com/DefaultCollection/Projects('Sonik')/WorkItems?$select=Title,Type&$top=3&$format=json

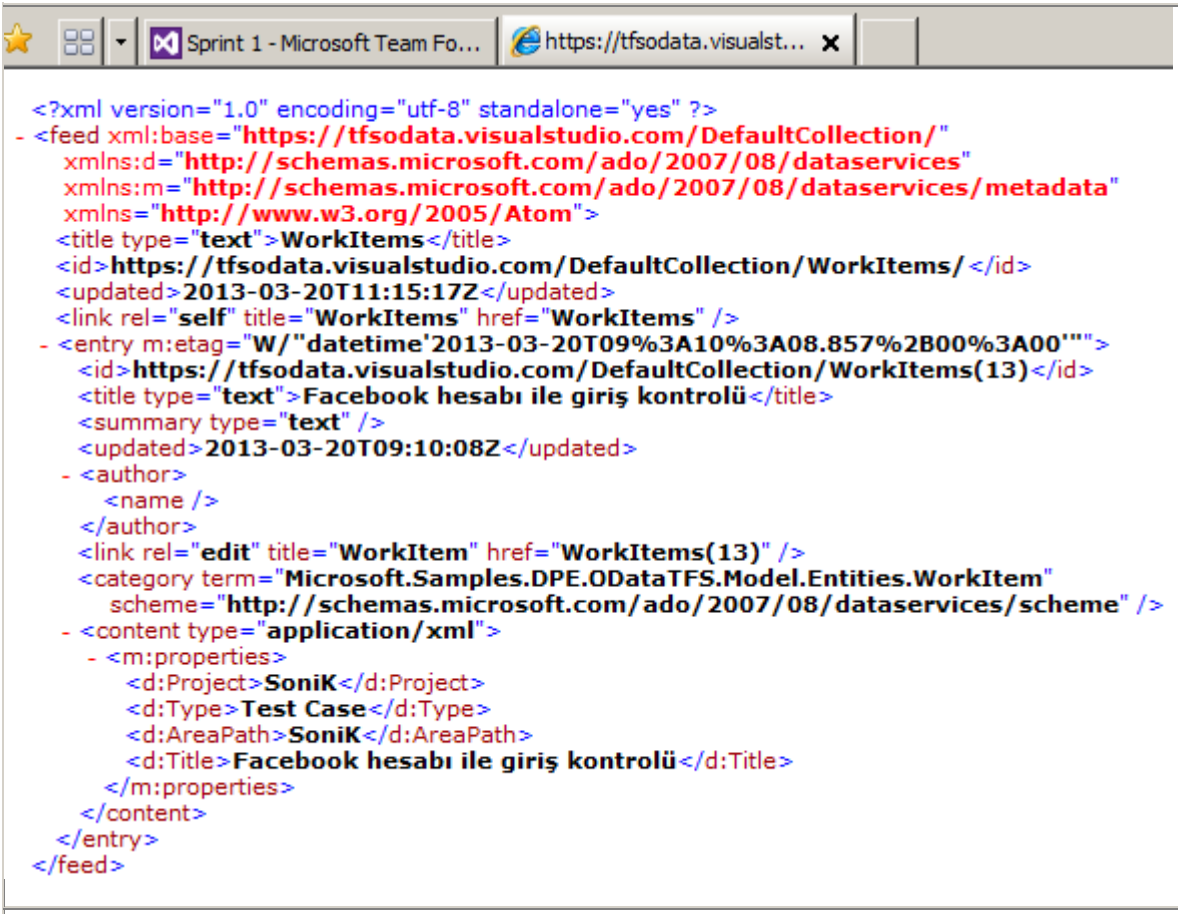
Çıktının sadece **XML** formatında elde edilmesi şart değildir. Örneğin daha az yer tutan **JSON** tipinde bir çıktı üretilmesi için **format** anahtar kelimesinden yararlanılabilir. Yukarıdaki sorguda **Sonik** projesindeki ilk 3 Work Item' ın **Title** ve **Type** değerlerini içeren **JSON** çıktısı talep edilmektedir. Sonuç aşağıdaki gibidir.



Bir projedeki belirli bir Work Item tipine ait öğeleri çekmek

https://tfsodata.visualstudio.com/DefaultCollection/WorkItems?$filter=Project eq 'Sonik' and Type eq 'Test Case'&$select=Project,Type,Title,AreaPath

Örnek sorguda **filter** ve **select** anahtar kelimelerinden yararlanılmış olup, **Sonik** isimli projede yer alan **TestCase** tipindeki **Work Item**'
ların **Project**, **Type**, **Title** ve **AreaPath** bilgileri talep edilmiştir.



Belirli bir Sprint içindeki Work Item bilgilerinin çekilmesi

https://tfsodata.visualstudio.com/DefaultCollection/WorkItems?$filter=Project eq 'Sonik' and Type eq 'Product Backlog Item' and IterationPath eq 'SoniK\Release 1\Sprint 1'&$select=Project,Type,Title,AreaPath,IterationPath

Örnekte **Sprint 1** içerisine alınmış olan **Product Backlog Item** öğelerinin **Project**, **Type**, **Title**, **AreaPath**,**IterationPath** değerleri sorgulanmaktadır. **filter** anahtar kelimesi ile birden fazla kriterin hesaba katılması noktasında **and** ve **or** operatörlerinden yararlanılır.


```
- <feed xml:base="https://tfsodata.visualstudio.com/DefaultCollection/"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns="http://www.w3.org/2005/Atom">
  <title type="text">WorkItems</title>
  <id>https://tfsodata.visualstudio.com/DefaultCollection/WorkItems/</id>
  <updated>2013-03-20T11:29:23Z</updated>
  <link rel="self" title="WorkItems" href="WorkItems" />
- <entry m:etag="W/"datetime'2013-03-20T08%3A57%3A31.083%2B00%3A00'"">
    <id>https://tfsodata.visualstudio.com/DefaultCollection/WorkItems(3)</id>
    <title type="text">Boyutu 50 Mb üzerindeki ses dosyaları paralel işlenebilir.</title>
    <summary type="text" />
    <updated>2013-03-20T08:57:31Z</updated>
  + <author>
    <link rel="edit" title="WorkItem" href="WorkItems(3)" />
    <category term="Microsoft.Samples.DPE.ODataTFS.Model.Entities.WorkItem"
      scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  - <content type="application/xml">
    - <m:properties>
        <d:Project>SoniK</d:Project>
        <d:Type>Product Backlog Item</d:Type>
        <d:AreaPath>SoniK</d:AreaPath>
        <d:IterationPath>SoniK\Release 1\Sprint 1</d:IterationPath>
        <d:Title>Boyutu 50 Mb üzerindeki ses dosyaları paralel işlenebilir.</d:Title>
      </m:properties>
    </content>
  </entry>
- <entry m:etag="W/"datetime'2013-03-20T08%3A57%3A14.03%2B00%3A00'"">
    <id>https://tfsodata.visualstudio.com/DefaultCollection/WorkItems(4)</id>
    <title type="text">Kullanıcı olarak Facebook hesabım ile giriş yapabilirim.</title>
    <summary type="text" />
    <updated>2013-03-20T08:57:14Z</updated>
  + <author>
    <link rel="edit" title="WorkItem" href="WorkItems(4)" />
    <category term="Microsoft.Samples.DPE.ODataTFS.Model.Entities.WorkItem"
      scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  - <content type="application/xml">
    - <m:properties>
        <d:Project>SoniK</d:Project>
        <d:Type>Product Backlog Item</d:Type>
        <d:AreaPath>SoniK</d:AreaPath>
        <d:IterationPath>SoniK\Release 1\Sprint 1</d:IterationPath>
        <d:Title>Kullanıcı olarak Facebook hesabım ile giriş yapabilirim.</d:Title>
      </m:properties>
    </content>
  </entry>
- <entry m:etag="W/"datetime'2013-03-20T08%3A57%3A45.353%2B00%3A00'"">
    <id>https://tfsodata.visualstudio.com/DefaultCollection/WorkItems(5)</id>
    <title type="text">Ses dosyaları sisteme mobil platform üzerinden upload edilebilir.</title>
    <summary type="text" />
    <updated>2013-03-20T08:57:45Z</updated>
  + <author>
    <link rel="edit" title="WorkItem" href="WorkItems(5)" />
    <category term="Microsoft.Samples.DPE.ODataTFS.Model.Entities.WorkItem"
      scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
  - <content type="application/xml">
    - <m:properties>
        <d:Project>SoniK</d:Project>
        <d:Type>Product Backlog Item</d:Type>
        <d:AreaPath>SoniK</d:AreaPath>
```


Şu an için **TFS Odata** servis sorgularında **filter**, **count**, **select**, **orderby**, **top**, **skip**, **format** ve**callback** anahtar kelimeleri kullanılabilmektedir. Ancak bu anahtar kelime seti artabilir. Sorgular sırasında **?** ve **$** harflerine de dikkat edilmelidir. Tüm **OData** komutlarının önünde dikkat

edileceği üzere **$** harfi yer almaktadır. Her ne kadar örneklerde ağırlıklı olarak **Work Item**' lar üzerinde durulmuş olsa da **DefaultCollection** altında sunulan **Entity**' lerin çoğu üzerinde sorgulamalar yapılabilir. Bunu denemenizi öneririm 
Görüldüğü üzere söz konusu **OData** sorgularını kullanarak farklı platformlar üzerinde çalışacak istemci uygulamaların geliştirilmesinin önü son derece açıktır. Şu anda halen geliştirilmekte olan TFS OData servislerinin kullanımına ilişkin detaylı bilgileri bu adresten takip edebilirsiniz. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.
[Orjinal Yazım Tarihi 3/20/2013]

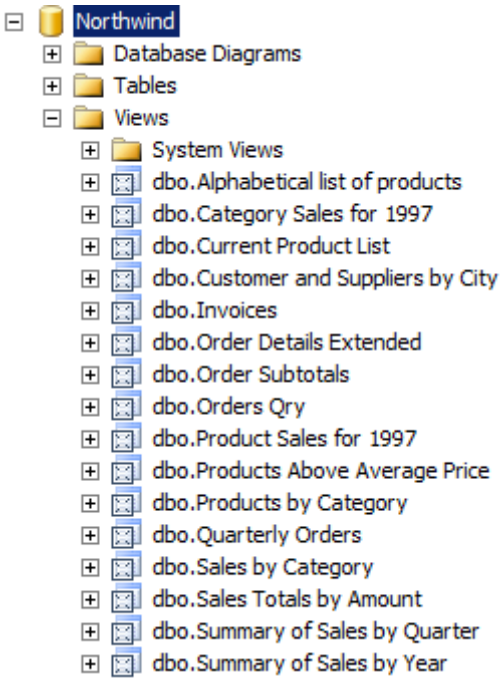
MigraDoc ile PDF Rapor Üretimi - Hello World

Pazartesi, 19 Ağustos 2013 08:48 by bsenyurt

"Mösyö Reno" dedi, oturduğu yerden Jimmy Carl. Uzun süredir bu dev şirketi yönetiyordu. Son zamanlarda teknolojiye büyük yatırım yapan firmanın, bundan en iyi şekilde yararlanabilmesini isteyenlerin başında geliyordu. Oldukça meraklı biri olan Carl, bilgisayarına bakarken içeriye orta boylarda, saçlarının bir kısmı ağırmış, numarası büyük olduğu belli olan kalın çerçeveli gözlüklü, hafif de göbekli ama güler yüzlü birisi girdi. Üstünde rengarenk bir hawai t-shirt, altında bermuda şort ve parmak arası terlikleri ile.
"Buyrun" dedi Reno, nefes nefese kalmış bir halde.
Fransız, yazılım alanında çift doktora yapmış birisiydi. Şirketin en kilit projelerinde görev almıştı. Bu yüzden Carl' ın da bir numaralı adamıydı.
"Sizin için ne yapabilirim?" diyerek devam etti sözlerine Fransız.
"Şu son satış rakamlarına ait raporlı diyorum Mösyö; acaba bunları PDF dosyasına kayıt edebilir miyiz?"
Gülümsedi Fransız Reno.
"Neden olmasın? Bana mesai bitimine kadar müddet verin lütfen."
Merhaba Arkadaşlar,
Özellikle **veri odaklı*(Data-Centric)*** çalışan uygulamalar düşünüldüğünde çeşitleri ne olursa olsun raporlama, işin oldukça önemli bir parçasını oluşturmaktadır. Ağırlıklı olarak rapora ihtiyaç duyan pozisyonlar, söz konusu raporları çeşitli ortamlarda görmek isteyen elemanlardır. Örneğin bunları Web arayüzünde açabilmeyi, **Excel** veya **Word** formatındaki dosyalara çıktı olarak alabilmeyi ve mobil cihazlarından takip edebilmeyi isterler. Günümüzün pek çok modern uygulaması zaten bu tip çıktıların alınmasını standart olarak olanak sunmaktadır.
Elbette çok farklı istekler de gelebilmektedir. Söz gelimi çıktı olarak basılacak veya bir dergi içerisinde kullanılması düşünülen raporlar için **PDF, XPS** gibi dosya formatlarında üretilmeleri istenebilir. İşte bu yazımızda bir rapor içeriğinin, **PDF** formatında nasıl oluşturulabileceğini basit bir**Hello World** uygulaması ile anlamaya çalışacağız. Örnek uygulamamızda açık kaynak olarak sunulan **MigraDoc** kütüphanelerinden yararlanacağız. **MigraDoc aslında, PDFSharp and MigraDoc Foundation isimli ürün ailesinin bir parçasıdır. Bu ürünlere ait kaynak kodları veya derlenmiş Binary dosyalarını[Codeplex üzerinden](#) indirebilirsiniz.**

Senaryo

İlk olarak örnek senaryomuzu ele alalım. Bilindiği üzere **Northwind** veritabanında aşağıdaki ekran görüntüsünde yer alan standart **View** nesneleri varsayılan olarak yer almaktadır.



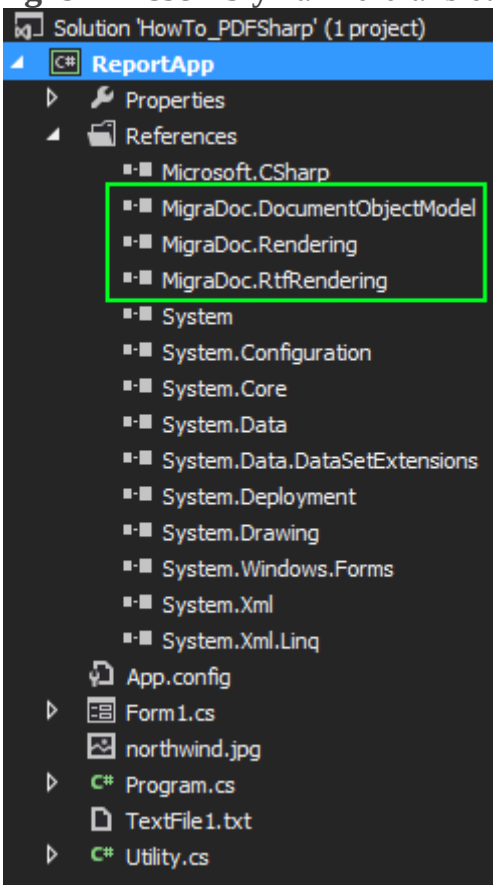
Bu **View** nesnelerinde örnek pek çok rapor içeriği bulunmaktadır. Söz gelimi *1997 yılına ait kategori bazlı satışları* veya *yıldan yıla satış rakamlarının özetini* görebilir, *listede yer alan tüm ürünlerimizi kategori bazlı* olarak elde edebiliriz. Dolayısıyla bu veriler raporlanabilir nitelikte olup çıktı şeklinde değerlendirilebilirler.

Senaryomuza göre basit bir **Windows Forms** uygulamasında, **Northwind** veritabanı içerisinde yer alan **View** nesnelerini kullanıcıya seçilebilir halde sunuyor olacağız. Bu **View** nesnelerinden her hangi biri seçildiğinde ise, veri içeriğini barındıran bir **PDF** dokümanının üretilmesini sağlayacağız. Örneğimizde ulaşmak istediğimiz hedef aşağıdaki ekran görüntüsündekine benzer olacaktır.

Sol üst köşede şirkete ait bir logo, **Footer** ve **Header** kısımlarında açık gri formatta bir bilgi, raporun alındığı **View** nesnesinin adı, üretildiği tarih, detay için **URL** adresine gönderme yapan bir link ve çok doğal olarak verinin kendisini içeren bir tablo. Peki bu içeriği nasıl üretiyor olacağız?

Kodlama Zamanı

Örneğimiz basit bir **Windows Forms** uygulaması şeklinde geliştirilecektir. İlk etapta uygulamaya aşağıdaki görselde yer alan **MigraDoc.DocumentObjectModel**, **MigraDoc.Rendering** ve**MigraDoc.RtfRendering** isim **Assembly**' ları referans etmemiz gerekiyor.



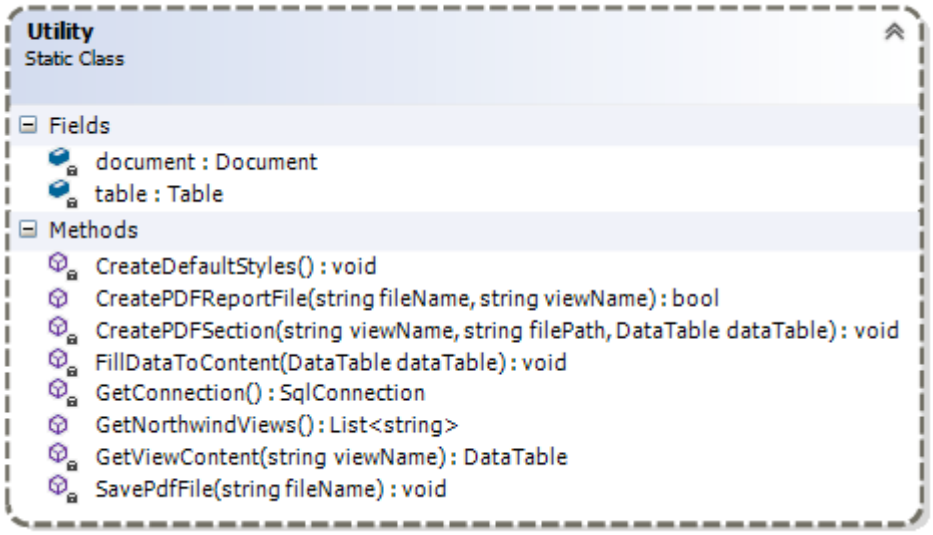
İlgili referansların eklenmesini takiben, aşağıdaki ekran görüntüsünde yer alan **Form** içeriğini tasarlayarak devam edebiliriz.

Pek tabi uygulamamız **Northwind** veritabanına bağlanacağından bazı **SQL** işlemlerini de icra ediyor olacak. Söz gelimi **View** adlarını, **sys.Views metadata** içeriğini kullanarak çekeceğiz. Diğer yandan bir **View** nesnesinin sunduğu veriyi almak için de **SQL** sorgusuna da ihtiyacımız olacak. Uzun lafın kısası temelde bir **Connection String** bilgisi gerekiyor İşte o bilgiyi **App.config** dosyasından tedarik edebiliriz.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
```

```
   <add name="Northwind"
      connectionString="data source=localhost;initial catalog=Northwind;integrated
security=SSPI"
      providerName ="System.Data.SqlClient"
      />
  </connectionStrings>
</configuration>
```

Genel fonksiyonelliklerimizi **Utility** isimli yardımcı bir **static** sınıf içerisinde tutabiliriz. Sınıf içeriği biraz uzun olduğundan adım adım ilerlemeye çalışmanızı öneririm.
İşte **Utility** içeriği;




```
using MigraDoc.DocumentObjectModel;
using MigraDoc.DocumentObjectModel.Shapes;
using MigraDoc.DocumentObjectModel.Tables;
using MigraDoc.Rendering;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Data.SqlClient;
using System.IO;
namespace ReportApp
{
   public static class Utility
   {
      #region Genel değişkenler
      static Document document = null;
      static Table table = null;
      #endregion Genel değişkenler
```

```
        // Hazır bir SqlConnection nesnesini bize verecek olan private fonksiyonumuz
        static SqlConnection GetConnection()
        {
            SqlConnection conn = new
SqlConnection(ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString)
;
            if (conn.State == ConnectionState.Closed)
                conn.Open();
            return conn;
        }
        // Rapor almak için kullanacağımız View bilgilerini çektiğimiz fonksiyon
       public static List<string> GetNorthwindViews()
        {
            List<string> viewNames = new List<string>();
            string query = "Select name from sys.views order by name";
            using(SqlConnection conn=GetConnection())
            {
                using(SqlCommand cmd=new SqlCommand(query,conn))
                {
                    SqlDataReader reader = cmd.ExecuteReader();
                    while (reader.Read())
                    {
                        viewNames.Add(reader["name"].ToString());
                    }
                    reader.Close();
                }
            }
            return viewNames;
        }
        // PDF Dosyasını oluşturacak olan metodumuz
        public static bool CreatePDFReportFile(string fileName,string viewName)
        {
            bool result = false;

            document = new Document();
            DataTable dataContent=GetViewContent(viewName);
            CreateDefaultStyles();
            CreatePDFSection(viewName,fileName, dataContent);
           FillDataToContent(dataContent);
            SavePdfFile(fileName);
            result = true;
```

```
        return result;
    }
    // İlgili View içeriğini bir DataTable nesnesi olarak geriye döndüren private
metodumuz
    static DataTable GetViewContent(string viewName)
    {
        string query = string.Format("select * from [{0}]", viewName);
        DataTable table = new DataTable();
        using (SqlConnection conn = GetConnection())
        {
            using (SqlDataAdapter adapter = new SqlDataAdapter(query, conn))
            {
                adapter.Fill(table);
            }
        }
        return table;
    }
    // PDF içeriğinde kullanılacak olan stiller belirlenir. Bu Style tiplerini HTML
içeriğindeki style kavramına benzetebiliriz.
    static void CreateDefaultStyles()
    {
        Style style = document.Styles["Normal"];
        style.Font.Name = "Calibri";
        style = document.Styles[StyleNames.Header];
        style.ParagraphFormat.AddTabStop("12cm", TabAlignment.Right);
        style = document.Styles[StyleNames.Footer];
        style.ParagraphFormat.AddTabStop("8cm", TabAlignment.Center);
        // Table isimli bir style oluşturuyoruz. Normal isimli Style' dan türemekte
        style = document.Styles.AddStyle("Table", "Normal");
        style.Font.Name = "Calibri";
        style.Font.Size = 9;
        // Normal isimli Style' ı baz alan Reference isimli bir Style oluşturuyoruz
        style = document.Styles.AddStyle("Reference", "Normal");
        style.ParagraphFormat.SpaceBefore = "3mm";
        style.ParagraphFormat.SpaceAfter = "3mm";
        style.ParagraphFormat.TabStops.AddTabStop("12cm",
TabAlignment.Right);
    }
    // PDF Sayfası üretilir.
    static void CreatePDFSection(string viewName,string filePath,DataTable
dataTable)
```

```
        {
            string info = string.Format("{0} raporu - Northwind Tarafından Üretilmiştir - Her
Hakkı Saklıdır {1}"
                , viewName
                , DateTime.Now.Year);
            // Landscape olacak şekilde dokümanın yönünü belirliyoruz
            document.DefaultPageSetup.Orientation = Orientation.Landscape;
            // İçeriğimizi koyacağımız bir Section oluşturuyoruz
            Section section = document.AddSection();
            #region Firma Logosunun Eklenmesi
            Image image =
section.AddImage(Path.Combine(Environment.CurrentDirectory,
"northwind.jpg"));
            image.Top = ShapePosition.Top;
            image.Left = ShapePosition.Left;
            #endregion Firma Logosunun Eklenmesi
            #region Header kısmı
            Paragraph paragraph = section.Headers.Primary.AddParagraph();
            paragraph.AddText(info);
            paragraph.Format.Font.Size = 10;
            paragraph.Format.Font.Color = Colors.LightGray;
            paragraph.Format.Alignment = ParagraphAlignment.Left;
            #endregion Header kısmı
            #region Footer kısmı
            paragraph = section.Footers.Primary.AddParagraph();
            paragraph.AddText(info);
            paragraph.Format.Font.Size = 10;
            paragraph.Format.Font.Color = Colors.LightGray;
            paragraph.Format.Alignment = ParagraphAlignment.Left;
            #endregion Footer kısmı
            #region Adres bildirimi

            paragraph = section.AddParagraph();
            paragraph.Format.SpaceBefore = "3cm";
            paragraph.Style = "Reference";
            paragraph.AddFormattedText(viewName, TextFormat.Italic);
            paragraph.AddTab();
            paragraph.AddText("Rapor Tarihi, ");
            paragraph.AddDateField("dd.MM.yyyy");
            paragraph.AddLineBreak();
            paragraph.AddText("Rapor, Migra Document API ile üretilmiştir");
```

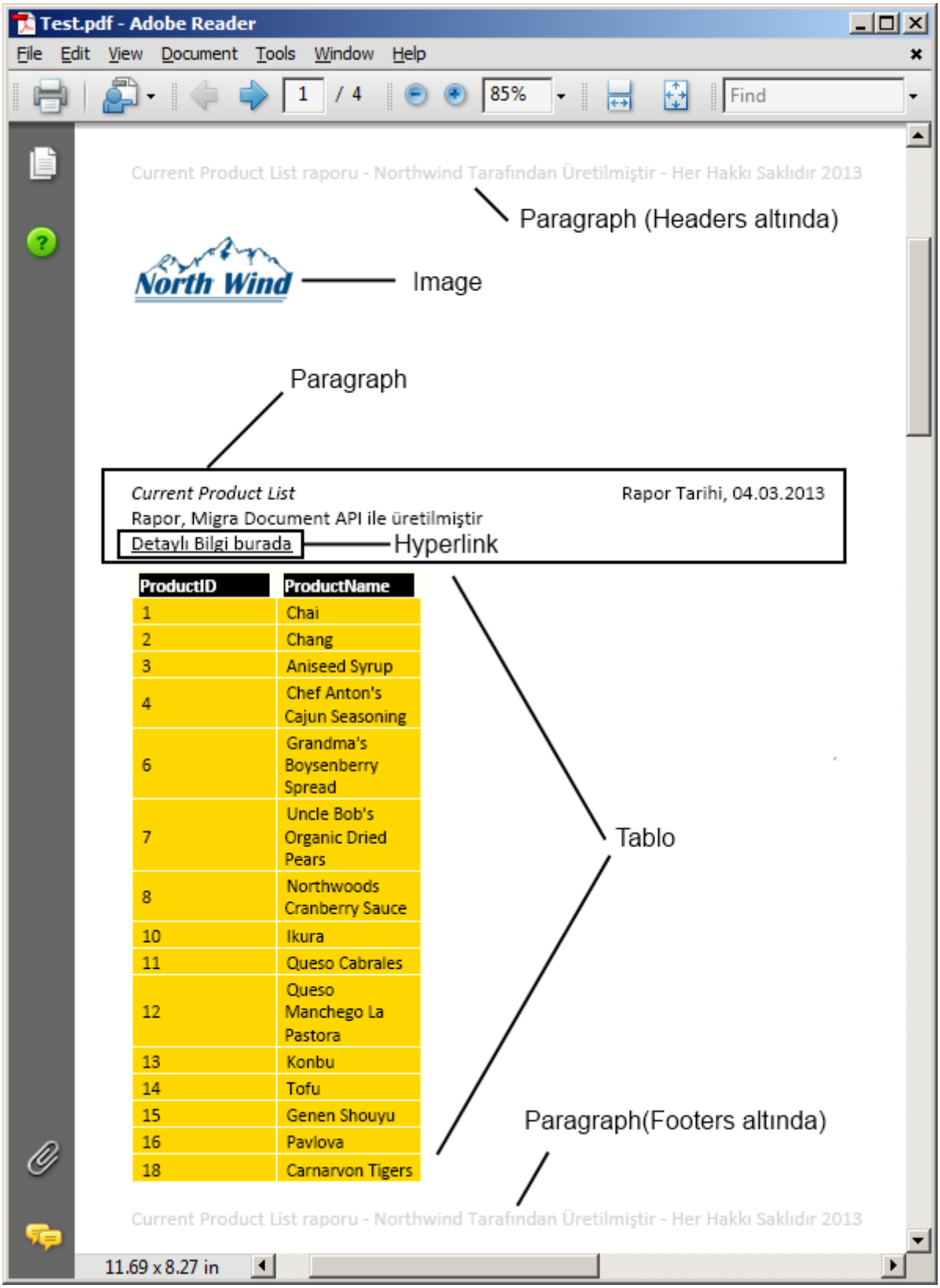
```
            paragraph.AddLineBreak();
            Hyperlink link = paragraph.AddHyperlink("http://www.buraksenyurt.com");
            link.Type = HyperlinkType.Url;
            link.Font.Underline = Underline.Single;
            link.AddText("Detaylı Bilgi burada");
            #endregion Adres bildirimi
            #region View içeriğinin basılacağı Table ve öğelerinin üretimi
            table = section.AddTable();
            table.Style = "Table";
            table.Borders.Color = Colors.LightYellow;
            table.Borders.Width = 0.25;
            table.Borders.Left.Width = 0.5;
            table.Borders.Right.Width = 0.5;
            table.Rows.LeftIndent = 0;
            Column column;
            foreach (DataColumn col in dataTable.Columns)
            {
                column = table.AddColumn(Unit.FromCentimeter(2.5));
                column.Format.Alignment = ParagraphAlignment.Center;
            }
            // Tablonun Header kısmı üretiliyor
            Row row = table.AddRow();
            row.HeadingFormat = true;
            row.Format.Alignment = ParagraphAlignment.Center;
            row.Format.Font.Bold = true;
            for (int i = 0; i < dataTable.Columns.Count; i++)
            {
                row.Cells[i].AddParagraph(dataTable.Columns[i].ColumnName);
                row.Cells[i].Format.Font.Color = Colors.White;
                row.Cells[i].Format.Shading.Color = Colors.Black;
                row.Cells[i].Format.Font.Bold = true;
                row.Cells[i].Format.Alignment = ParagraphAlignment.Left;
                row.Cells[i].VerticalAlignment = VerticalAlignment.Bottom;
            }
            table.SetEdge(0, 0, dataTable.Columns.Count, 1, Edge.Box,
BorderStyle.Single, 0.75, Color.Empty);
            #endregion View içeriğinin basılacağı Table ve öğelerinin üretimi
        }
        // Sayfa içeriği parametre olarak gelen DataTable içeriğine göre doldurulur
        static void FillDataToContent(DataTable dataTable)
        {
```

```
        Row newRow;
        for (int i = 0; i < dataTable.Rows.Count; i++)
        {
           newRow = table.AddRow();
           newRow.TopPadding = 1.5;
           for (int j = 0; j < dataTable.Columns.Count; j++)
           {
              newRow.Cells[j].Shading.Color = Colors.Gold;
              newRow.Cells[j].VerticalAlignment = VerticalAlignment.Center;
              newRow.Cells[j].Format.Alignment = ParagraphAlignment.Left;
              newRow.Cells[j].Format.FirstLineIndent = 1;
              newRow.Cells[j].AddParagraph(dataTable.Rows[i][j].ToString());
              table.SetEdge(0, table.Rows.Count - 2, dataTable.Columns.Count, 1,
Edge.Box, BorderStyle.Single, 0.75);
           }
        }
     }
     // PDF Dosyası parametre olarak belirtilen adrese kayıt edilir
     static void SavePdfFile(string fileName)
     {
        PdfDocumentRenderer pdfRenderer = new
PdfDocumentRenderer(true);
        pdfRenderer.Document = document;
        pdfRenderer.RenderDocument();
        pdfRenderer.Save(fileName);
     }
  }
}
```

Aslında fonksiyonları dikkatlice incelediğimizde **PDF** içeriğini oluşturma işleminin oldukça basit olduğunu görebiliriz. Baş kahraman **MigraDoc.DocumentObjectModel** içerisinde yer alan**Document** tipidir. Tahmin edileceği üzere **MigraDoc**, **PDF** içeriğine ait **Document Object Model**' i kullanmaktadır.

Document tipi aslında sayfa içerisinde kullanılacak **global Style**' leri ve **Section**' ları oluşturmak için kullanılır. Bu nedenle **PDF** içersine bir **paragraf, tablo, resim, link** ve benzeri materyalleri eklemek istediğimizde bir **Section** tipinden yararlanmalıyız. Örnekte dikkat edileceği üzere **Logo**'nun eklenmesi için **section** nesne örneği üzerinden **AddImage** metodu çağırılmaktadır. Diğer yandan bir paragraf eklenmek istendiğinde yine **section** nesne örneği üzerinden ama bu kez**AddParagraph** fonksiyonuna çağrıda bulunulmaktadır.

Örneğimizin kalbi rapor içeriklerinin bir tablo içerisinde gösterilmesidir. Bu sebepe bir **Table** tipi kullanılmış ve örneklenmesi için yine **section** nesnesi üzerinden hareket edilerek, **AddTable**metoduna başvuruda bulunulmuştur. Bir **Table** örneklendikten sonra içerisinde yer alan**satırların*(Rows)*** veya **hücrelerin*(Cells)*** doldurulması işi, aslında bir **HTML Table** içeriğinin dinamik olarak üretilmesinden pek de farklı değildir. Yeterki doküman nesne modeline hakim olalım 
Örneğimizin daha kolay anlaşılması açısından aşağıdaki görseli dikkate alabilirsiniz.



Kodlama sırasında keşfetmemiz gereken veya bizleri zorlayabilecek olan noktalar genellikle ilgili içeriğin*(paragraf, tablo, resim vb)* sayfa içerisindeki yerleşimlerinin ayarlanmasıdır. Aslında bir kaç deneme yanılma ile düzgün bir şablon oturtabiliriz. Ancak yine de dikkate alınması gereken bazı hususlar vardır. Söz gelimi **View**' un döndürdüğü sonuç kümesinde yer alan kolon sayısının çok fazla olması halinde sayfaya sığmayacak ve

yatay olarak görüntü kayıpları yaşanacaktır ki bu **Jimmy Carl'** ın pek de hoşuna gitmeyecektir. Bu gibi ileri seviye sayılabilecek hususlar örneğimizde ele alınmamıştır. Dolayısıyla siz kendi örneklerinizi icra ederken daha dikkatli davranmalısınız.

Satır sayısı çok fazla olan bir içeriğin PDF' e yazılması oldukça uzun sürebilir/sürdüğü gözlemlenmiştir. Bu sebepten ilgili dosya kaydetme operasyonunun aslında asenkron bir düzenek ile icra edilmesi çok daha uygun olabilir. Hatta büyük boyutlu raporlar için bir Progress Bar ile durum bildirimi bile yapabilirsiniz *(async ve await kullanmayı deneyiniz)*

Form içeriğindeki kodlarımız ise aşağıdaki gibidir. **Utility** tipi pek çok ağır fonksiyonelliği kapsüllediğinden bu kısımın okunurluğu çok daha kolaydır.

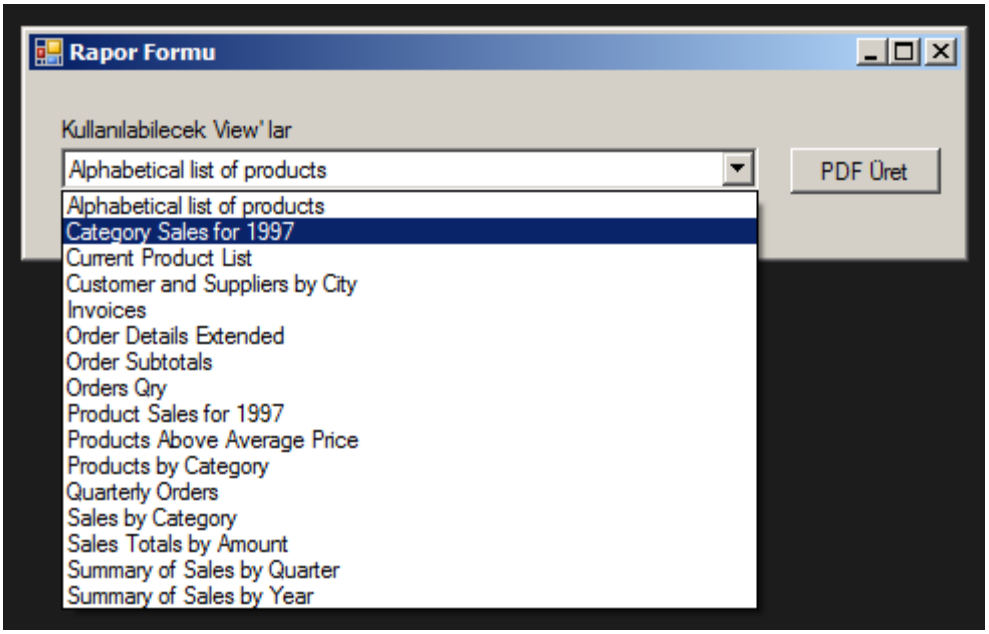
```
using System;
using System.Diagnostics;
using System.Windows.Forms;
namespace ReportApp
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
         // Northwind veritabanında yer alan View isimleri ComboBox kontrolüne basıyoruz.
         cmbViews.DataSource = Utility.GetNorthwindViews();
      }
      private void btnCreatePDF_Click(object sender, EventArgs e)
      {
         // Kullanıcıdan bir PDF dosya adı istiyoruz
         if(sfdReportFile.ShowDialog()== DialogResult.OK)
         {
            // Dosyanın var olması halinde kullanıcının tepkisini bekliyoruz. Yes düğmesine basarsa üzerine yazacak
            if(sfdReportFile.CheckFileExists==true)
                  return;
            string pdfFileName = sfdReportFile.FileName;
            // PDF oluşturma işlemini üstlenen Utility tipini çağırıyoruz.
            Utility.CreatePDFReportFile(pdfFileName, cmbViews.SelectedValue.ToString());
            // Üretilen PDF dosyasını sistemde PDF Read edebileceğimiz bir uygulama olduğunu var sayarak Process tipi yardımıyla açtırıyoruz.
            // Bu sayede üretilen raporu da görebiliriz.
            Process.Start(pdfFileName);
```

```
            }
        }
    }
}
```

Dikkat edilmesi gereken noktalardan birisi de **PDF** dosyasının kayıt edilmesinden sonra**Process.Start** operasyonu ile ilgili içeriğin otomatik olarak açılmasıdır. Bu sayede sonuçları anında görebiliriz.

Çalışma Zamanı

Örneğimizi çalıştırdığımızda **Select name from sys.views order by name** sorgusunun bir sonucu olarak tüm **View** nesnelerinin elde edilebildiği görülecektir. Bu şekilde bir kullanım nedeni ile,**Northwind** veritabanına eklenecek olan yeni **View**' ları da **PDF** üretimi sürecine katabiliriz.

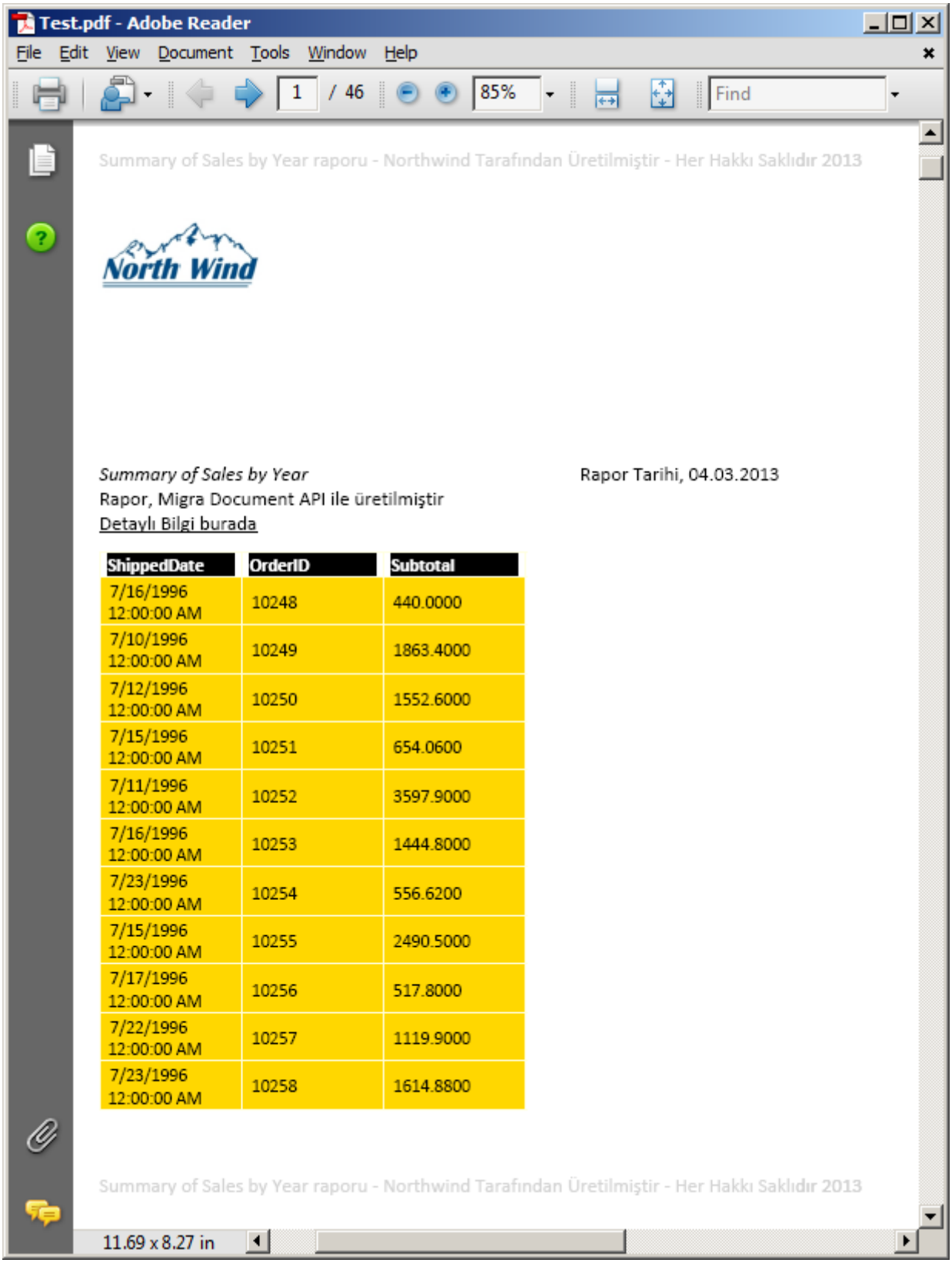


ve bir kaç rapor örneğine ait ekran çıktısına yer vererek devam edelim.

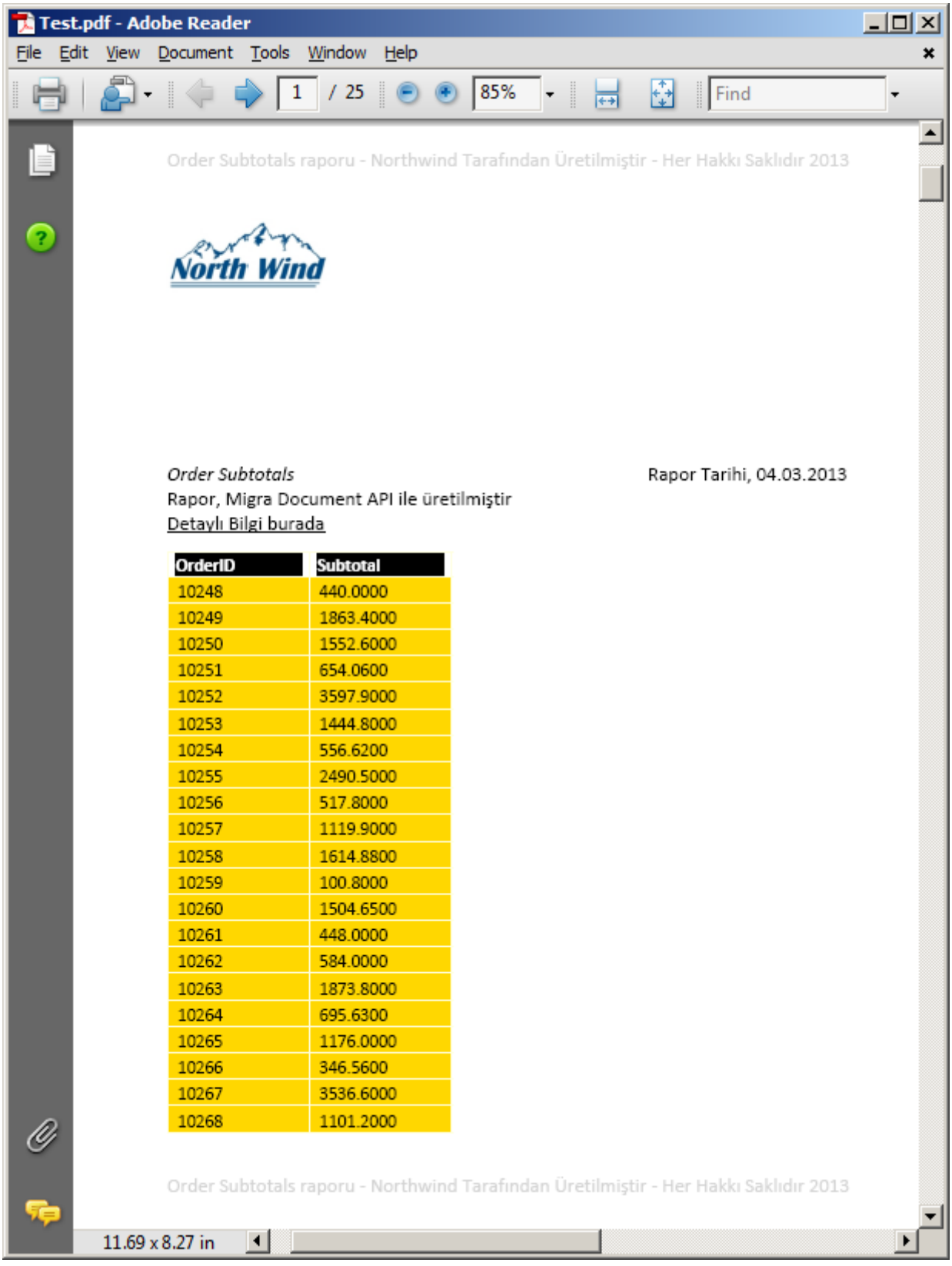
Örnek PDF Çıktıları

Products by Category View' u için örnek ekran çıktısı

Summary of Sales By Year View' u için örnek ekran çıktısı



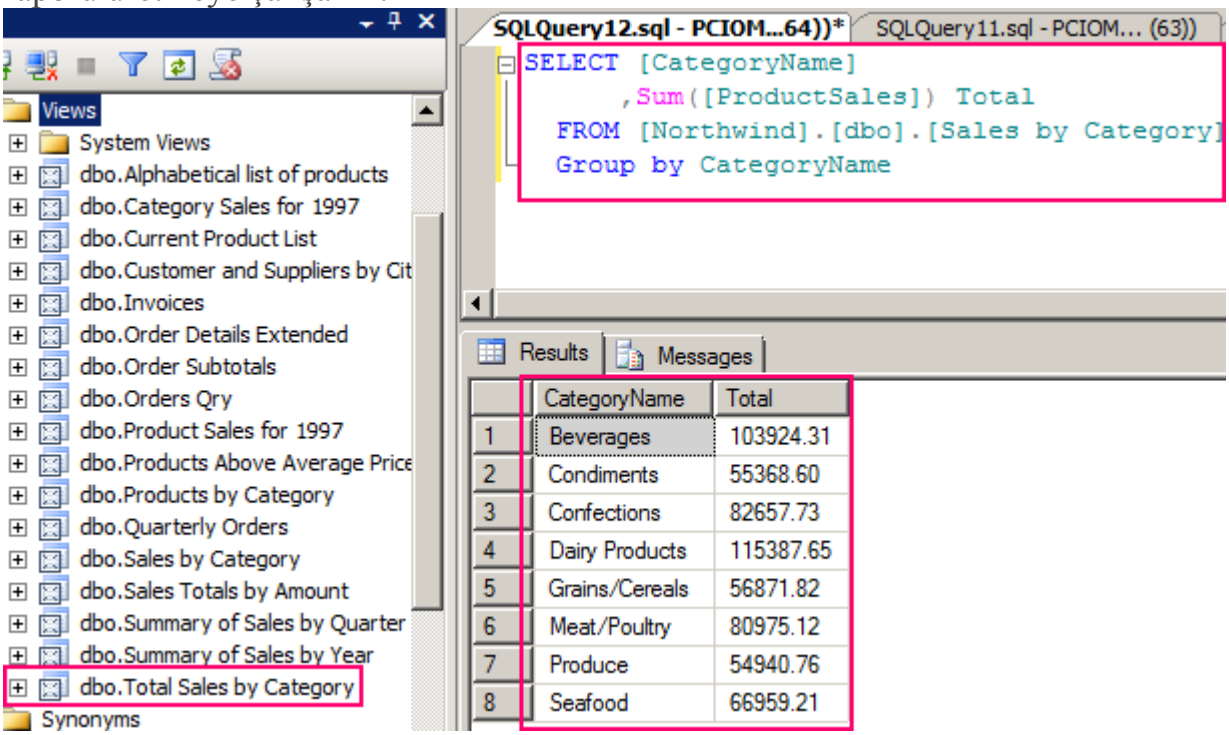
Order Subtotals View' u için örnek ekran çıktısı



PDF içinde Chart Üretimi

Elbette **PDF** dosyasına çıktı olarak verilen bu raporlar arasında en etkileyici olanlarından birisi de**Chart** tipindekilerdir. Şimdi örnek senaryomuzda aşağıdaki görsel de yer alan ve

kategori bazlı toplam satış rakamlarını gösteren **View** nesnesini kullanarak **Line** tipinde bir raporu üretmeye çalışalım.



İşte kodlarımız

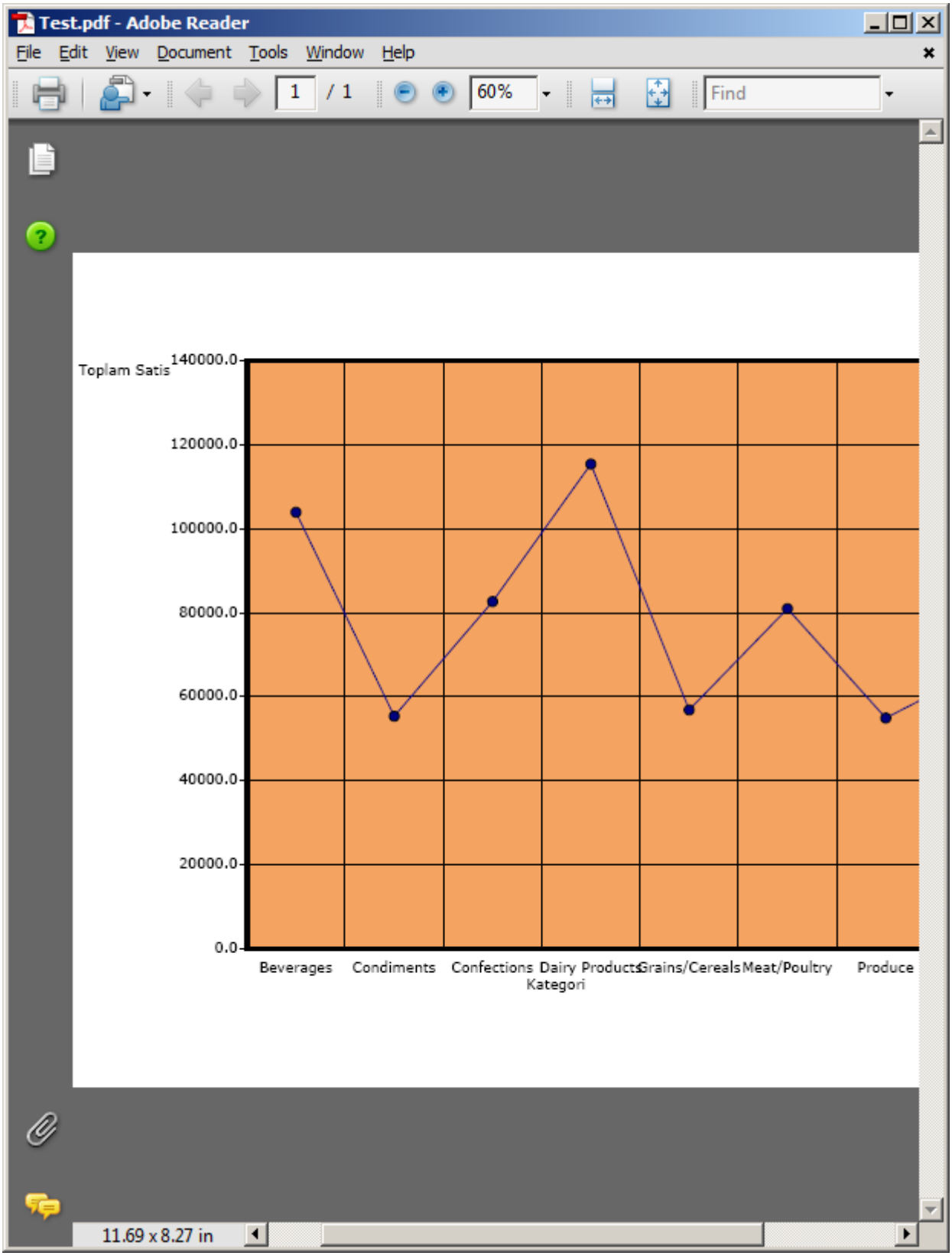
```
public static bool CreateChart(string fileName)
{
   bool result=false;
   document = new Document();
   document.DefaultPageSetup.Orientation = Orientation.Landscape;
   DataTable dataContent = GetViewContent("Total Sales By Category");
  DrawChart(dataContent);
   SavePdfFile(fileName);
   result = true;
   return result;
}
private static void DrawChart(DataTable dataTable)
{
   List<double> serieValues = new List<double>();
   List<string> xAxisValues = new List<string>();
   Section section = document.AddSection();
   Chart chart = new Chart();
   chart.Left = 0;
   chart.Width = Unit.FromCentimeter(24);
   chart.Height = Unit.FromCentimeter(16);
   Series series = chart.SeriesCollection.AddSeries();
   series.ChartType = ChartType.Line;
```

```
    foreach (DataRow row in dataTable.Rows)
    {
        serieValues.Add(Convert.ToDouble(row["Total"]));
        xAxisValues.Add(row["CategoryName"].ToString());
    }
    series.Add(serieValues.ToArray());
    XSeries xSeries = chart.XValues.AddXSeries();
    xSeries.Add(xAxisValues.ToArray());
    chart.XAxis.Title.Caption = "Kategori";
    chart.XAxis.HasMajorGridlines = true;
    chart.YAxis.Title.Caption = "Toplam Satış";
    chart.YAxis.HasMajorGridlines = true;
    chart.PlotArea.FillFormat.Color = Colors.SandyBrown;
    chart.PlotArea.LineFormat.Width = 3;
    section.Add(chart);
}
```

Baş rol oyuncusu bu kez **Chart** tipinden olan nesne örneğidir. Bu nesne örneği bir **Section** içerisinde yer almalıdır. Aynen **Excel Chart** nesnelerinde olduğu gibi **X** ve **Y** eksenleri ve bu eksenlere dizilmiş değerler bulunmaktadır. Dolayısıyla bu serileri veri ile doldurmak önemlidir. Bu sebepten **DataTable**içeriğinde dolaşılmış ve **toplam satış rakamları** ile **kategori adları**, sırasıyla **XAxis** ve **YAxis**özelliklerine ait koleksiyonlara yerleştirilmiştir. Bu fonksiyonellikleri yeni bir **Button** arkasında aşağıdaki kod parçasında olduğu gibi deneyebiliriz.

```
private void btnCreateChart_Click(object sender, EventArgs e)
{
    if (sfdReportFile.ShowDialog() == DialogResult.OK)
    {
        if (sfdReportFile.CheckFileExists == true)
            return;
        string pdfFileName = sfdReportFile.FileName;
        Utility.CreateChart(pdfFileName);
        Process.Start(pdfFileName);
    }
}
```

Uygulamayı çalıştırdığımızda aşağıdakine benzer bir sonuç ile karşılaşırız.



Sonuç

Sonuç olarak en azından işe yarar **PDF** içeriklerini kolayca üretebildiğimize şahit olduk. Örnekleri geliştirmek tamamen sizin elinizde. Söz gelimi **Chart** bileşenini kullandığımız

senaryoyu daha da ileri götürebilir, örneğin **Elma Dilimi** raporları işin içersine katarak daha etkileyici çıktılar sunabilirsiniz*(Üstelik bu tip görsel raporlar **Jimmy Carl'** ın da çok hoşuna gidecektir)* Buna ilaveten bir ön iletişim kutusundan yararlanarak görsellik üzerine detay bilgilerini*(örneğin tablonun arka plan rengi, font büyüklükleri, logonun gösterilip gösterilmeyeceği, bir özet bilginin konulup konulmayacağı, header veya footer kısımlarında ne yazılması istendiği vb)* kullanıcıdan alabilirsiniz. Biz bu yazımızda sadece **Hello World** demeye çalıştık

PDF Sharp ve Migra kütüphanelerinin detaylı kullanımı ile ilişkili olarak bu Wiki sayfasından da yararlanabilirsiniz. Oldukça geniş kullanım örnekleri olduğunu ifade edebilirim.

Böylece geldik bir makalemizin daha sonuna. Bir diğer yazımızda görüşünce dek hepinize mutlu günler dilerim.

HowTo_PDFSharpV2.zip (555,18 kb)

Nedir Bu MSBuild?

Cuma, 2 Ağustos 2013 07:47 by bsenyurt

[Orjinal Yazım Tarihi – 25/02/2013]

Merhaba Arkadaşlar,

Yıllar öncesinde bir kaç seneliğine de olsa saygın bir eğitim kurumunda eğitmen olarak görev alma şansını yakalamıştım. Özellikle **C#**' ın öğretilmeye çalışıldığı başlangıç niteliğindeki seanslarda dilin temel özelliklerini anlatırken, tüm dış çevre ile olan bağlantıyı kesip, sadece anahtar kelime(keyword), ifade ve materyale odaklanmaya çalışırdık. Bu sebepten genellikle ilk örneklerimiz ve**Hello World** uygulamamız, **Notepad** gibi bir program ve komut satırındaki **csc*(C# Compiler)*** ile inşa edilirdi.

O zamanlar bu bizim için yeterli görünüyordu ama tabi **.Net Framework 2.0** ile birlikte hayatımıza yeni bir inşa süreci de girdi. Aslında bu günkü konumuzda da, **Notepad*(tam olarak Notepad 2)***ve komut satırı aracını kullanarak ilerlemeye çalışıyor olacağız. Amacımız **MSBuild** platformunu çok kısaca tanımaya ve anlamaya çalışmak.

Microsoft Build Engine aslında başlı başına bir platformdur. Kısaca **MSBuild** olarak anılmaktadır ve bir uygulamanın inşa edilmesi noktasında devreye giren **XML*(eXtensible Markup Language)***tabanlı bir **Script** bütününü esas alır. Kısacası uygulamanın inşa edilmesi sırasındaki aşamalar **XML**tabanlı bir akış olarak ifade edilebilmektedir. **MSBuild** platformunun en önemli özelliği ise, inşa sürecinde **Visual Studio** gibi bir araca ihtiyaç duymuyor oluşudur.

Evet **Visual Studio**' nun kendisi, **Build** işlemlerinde bu platformu kullanmaktadır doğru ama, tam tersi durum geçerli değildir. Yani istersek **MSBuild** aracını kullanarak, bir uygulamanın veya uygulama ortamının üretilmesi sırasındaki aşamaları, basit bir **Notepad** aracı ile tasarlayabilir ve**MSBuild.exe**' den yararlanarak hayata geçirebiliriz*(Ancak bu gün şanslısınız çünkü **Notepad2**isimli ürünü kullanacağız. Sourceforge adresinden indirebilirsiniz)*

Bu fikir tabi ki otomatize edilmiş **Build** işlemlerinin de icra edilebileceği anlamına gelmektedir. Ki**Team Foundation Server** ürünü de **MSBuild**' un etkin bir şekilde kullanılmasına olanak tanımaktadır. Özellikle **Team Foundation Build** olarak anılan platform içerisinde **Build Server**' un kurulduğu ortam **MSBuild Script**' lerini kullanarak üretim işlemlerini gerçekleştirmektedir. Söz gelimizi **TFS** tarafında eğer **Continous Integration** gibi bir strateji tercih edilmişse, kodlamacıların**Check-In** işlemleri sonrası **Team Foundation Build** devreye girecek ve **MSBuild script**' leri otomatik olarak çalıştırılarak inşa işlemleri icra edilecektir. *(**Team Foundation Build** ayrıca incelenmesi gereken bir konu olduğundan bu yazımızda detaylandırılmamıştır)*

Visual Studio ortamında geliştirdiğimiz projeleri göz önüne aldığımızda, oluşturulan proje dosyaları içerisinde, **MSBuild**' un kullanacağı **ayarlar*(Settings)*** ve bazı koşul bağlı

işlevsellikler konuşlandırılmaktadır. Tabi geliştirici olarak bu kısımlar ile pek fazla uğraşmayız ve nihayetinde**Visual Studio** bizim için bu akışları otomatik olarak inşa eder. Ancak **Release Manager** gibi pozisyonlar özellikle bu proje dosya içeriklerini değerlendirerek genişletmeler yapabilir ve **MSBuild**sürecini farklılaştırabilirler.
Visual Studio ile geliştirilen proje dosyaları C# tarafı için csproj, Visual Basic tarafı için vbproj, Managed C++ tarafı içinse vcxproj uzantılı olanlardır.

Hangi Durumlarda MSBuild

Peki **MSBuild** ağırlık olarak hangi hallerde ele alınır.

- En bilinen sebep elimizde **Visual Studio** olmayan bir ortamda inşa etme işlemlerinde değerlendirilebiliyor olmasıdır.
- **Compiler** devreye girmeden önce bazı dosyaların **Process** edilmesi gereken durumlarda ele alınabilir.*(Pre-Processing Steps)*
- Benzer şekilde **Process** sonrası yapılması istenen işlemler içinde kullanılabilir*(Post-Processing Steps)*
- **Build** işlemi sonucu çıktıların farklı bir klasöre taşınması sağlanabilir. *(Ki bu klasör pek çok projenin ortaktaşa kullandığı bir assembly için paylaşımdaki bir makine bile olabilir)*
- Çıktıların sıkıştırılması istendiği bir durumda ele alınabilir.
- İnşa işleminin birden fazla **Process'** e bölünerek daha hızlı tamamlanması istendiği durumlarda değerlendirilebilir. *(Özellikle Enterprise çözümlerde, n sayıda Branch' in ve dolayısıyla çıktının söz konusu olabileceği senaryolarda, uzun sürebilecek Build işlemleri için kritik bir özelliktir)*
- **MSBuild 64bitlik** bir sistem için inşa işlemini icra edebilir.
- **Build** işleminin herhangibir noktasında harici bir aracın kullanılması istendiği hallerde göz önüne alınabilir.

ve benzeri pek çok durumda **MSBuild**' u açık bir şekilde özelleştirerek kullanabiliriz.

Proje Dosyasının İçine Bakalım

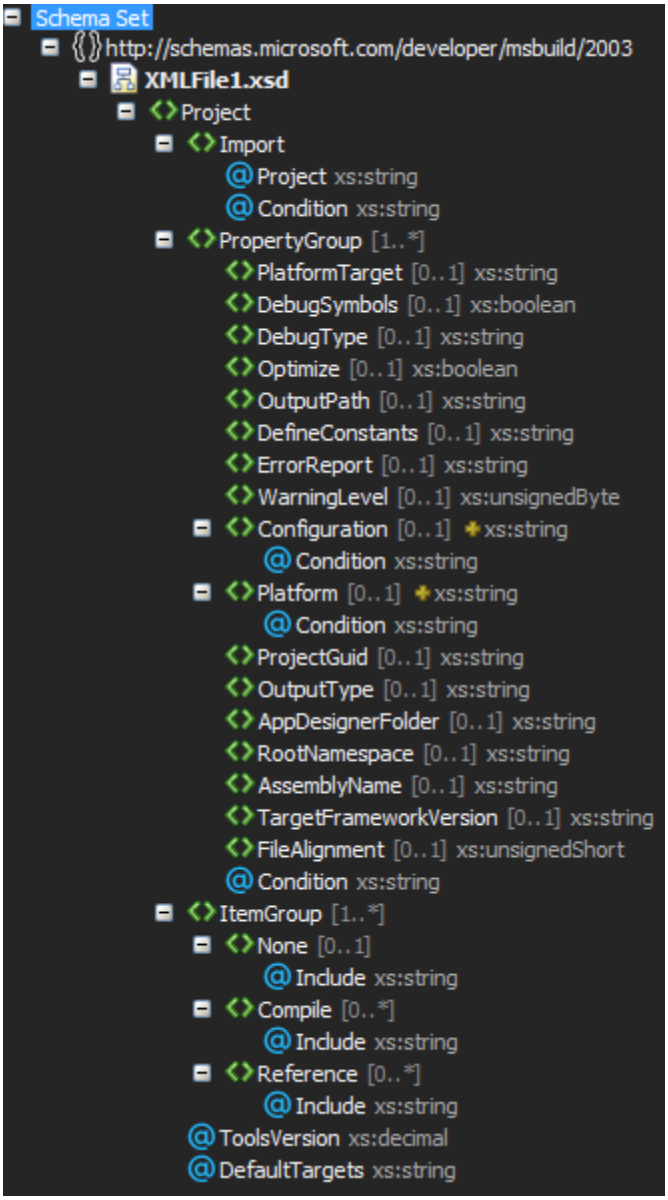
Hızlı bir şekilde uygulama geliştirme işine giren pek çok yazılımcı çoğunlukla **MSBuild** gibi programların ürettiği çıktıları göz ardı etmektedir. Nasıl olsa binlerce dolar verilerek satın alınan**Visual Studio** bizim yerimize pek güzel bu işi halletmektedir. Ancak gerçek hayatta öyle vakalar ve senaryolara vuku bulmaktadır ki, bunların üstesinden gelebilmek için özelleştirmelere gidilmesi şart olmaktadır. Bu özelleştirme konsepti **MSBuild** tarafı için de geçerlidir. Bu sebepten proje dosyalarının içeriğinin az da olsa bilinmesi en azından şema yapısının anlaşılması yararlıdır. Tipik olarak bir **Console** uygulaması dahi açsak, üretilen proje dosyası içerisinde*(ki örneğimiz **csproj**uzantılı olandır)* aşağıdakine benzer bir içerik oluştuğu görülecektir.

```
<?xml version="1.0" encoding="utf-8"?>
<Project ToolsVersion="4.0" DefaultTargets="Build"
xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
```

```
  <Import Project="$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Com
mon.props"
Condition="Exists('$(MSBuildExtensionsPath)\$(MSBuildToolsVersion)\Microsoft.Comm
on.props')" />
  <PropertyGroup>
    <Configuration Condition=" '$(Configuration)' == " ">Debug</Configuration>
    <Platform Condition=" '$(Platform)' == " ">AnyCPU</Platform>
    <ProjectGuid>{21E86FBF-5A78-4175-8522-A6FC854BB637}</ProjectGuid>
    <OutputType>Exe</OutputType>
    <AppDesignerFolder>Properties</AppDesignerFolder>
    <RootNamespace>ConsoleApplication33</RootNamespace>
    <AssemblyName>ConsoleApplication33</AssemblyName>
    <TargetFrameworkVersion>v4.5</TargetFrameworkVersion>
    <FileAlignment>512</FileAlignment>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Debug|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugSymbols>true</DebugSymbols>
    <DebugType>full</DebugType>
    <Optimize>false</Optimize>
    <OutputPath>bin\Debug\</OutputPath>
    <DefineConstants>DEBUG;TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <PropertyGroup Condition=" '$(Configuration)|$(Platform)' == 'Release|AnyCPU' ">
    <PlatformTarget>AnyCPU</PlatformTarget>
    <DebugType>pdbonly</DebugType>
    <Optimize>true</Optimize>
    <OutputPath>bin\Release\</OutputPath>
    <DefineConstants>TRACE</DefineConstants>
    <ErrorReport>prompt</ErrorReport>
    <WarningLevel>4</WarningLevel>
  </PropertyGroup>
  <ItemGroup>
    <Reference Include="System" />
    <Reference Include="System.Core" />
    <Reference Include="System.Xml.Linq" />
    <Reference Include="System.Data.DataSetExtensions" />
    <Reference Include="Microsoft.CSharp" />
    <Reference Include="System.Data" />
```

```
    <Reference Include="System.Xml" />
  </ItemGroup>
  <ItemGroup>
    <Compile Include="Program.cs" />
    <Compile Include="Properties\AssemblyInfo.cs" />
  </ItemGroup>
  <ItemGroup>
    <None Include="App.config" />
  </ItemGroup>
  <Import Project="$(MSBuildToolsPath)\Microsoft.CSharp.targets" />
  <!-- To modify your build process, add your task inside one of the targets below and
uncomment it.
       Other similar extension points exist, see Microsoft.Common.targets.
  <Target Name="BeforeBuild">
  </Target>
  <Target Name="AfterBuild">
  </Target>
  -->
</Project>
```

Bu **XML** içeriğinin şema yapısına bakıldığında aşağıdaki grafikte görülen iskeletin söz konusu olduğu görülecektir.



Aslında içerik okunduğunda, **Visual Studio Proje Özelliklerinden** de ayarladığımız pek çok öğenin buraya yazıldığı görülebilir.

Örneğin **PropertyGroup** elementlerinde **Debug**, **Debug-Any CPU** ve **Release-Any CPU** için bazı atamalar söz konusudur.

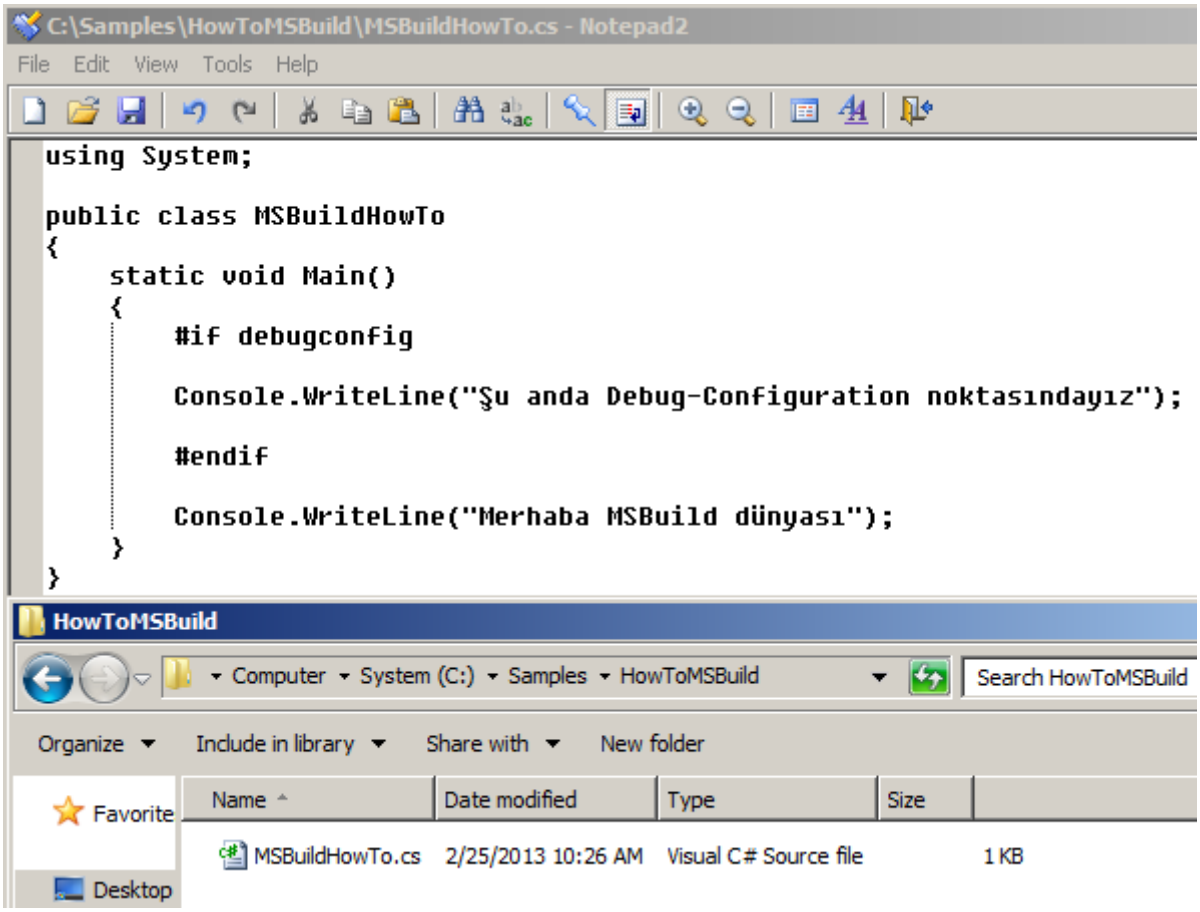
Debug ile alakalı **PropertyGroup**' a bakıldığında uygulamanın vereceği çıktının **exe** olacağı, **.NetFramework 4.5** platformunu hedef aldığı, **Assembly** adının **ConsoleApplication33** olduğu ve buna bağlı olarak da **Root Namespace'** in yine **ConsoleApplication33** şeklinde set edildiği görülebilir. **Condition** niteliklerinde belirtilen kısımlar, **MSBuild** uygulamasına bazı kriterlere göre nasıl çıktı üretmesi gerektiğini de söylemektedir. Örneğin uygulama **Release** modda inşa edildiğinde, çıktının **bin\Release** klasörüne doğru yapılması gerektiği yine bir **PropertyGroup**içerisinde ifade edilmiştir.

Condition niteliklerinde kullanılan $ notasyonu mutlaka dikkatinizi çekmiştir. Aslında burada $[PropertyName] şeklinde bir kullanım söz konusudur. Burada inanılmaz geniş bir esneklik vardır.

Örneğin $(registery:Hive\SomeKey\SomeSubKey@Value) gibi bir ifade ile Registery' deki bir değeri okuyabilir ve Build içerisinde kullanabiliriz. Ya da $([System.DateTime]::Now.ToString("yyyy.MM.dd")) gibi bir kullanım ile o anki zamanı istediğimiz formatta elde edebiliriz vb…
[PropertyName] yerine gelecek MSBuild özelliklerinin neler olabileceğini MSDN adresinden öğrenebilirsiniz.
ItemGroup elementleri içerisine bakıldığında, uygulamanın referans ettiği diğer **Assembly'** ların adları, **Compile** işlemi sırasında derlemeye tabi olacak **C#** dosyaları gibi bilgiler yer almaktadır.**Visual Studio'** nun ürettiği **XML** içeriğine bakıldığında, son kısımda yorum satırları içerisine dahil edilmiş **Target** isimli bir element daha olduğu görülmektedir. Bu elementi kullanarak **MSBuild** için bazı **Task'** lar tanımlanabilir ve inşa işlemi sırasında devreye girmeleri sağlanabilir.
Task olarak yapılan tanımlamalar paylaşılabilir ve farklı geliştirme **Build'** larında da kullanılabilir. Bu yüzden bir yazılım evinin **Build** işlemlerinde standart olarak gerçekleştirdiği bazı yürütmeler, **Task**olarak tanımlanıp ilgili **Build** paketlerine gömülebilir. Ayrıca **Task'** lar **Reusable** özelliği taşımaktadır. *(**Target** elementleri genellikle yazıldıkları sırada çalışmaktadır. Yani belirli bir sırada icra edilmesi istenen **Task'** lar var ise, **Target** elementinin buna uygun olacak şekilde kullanılması gerekir)*
Schema yapısı bu kadar basit değildir. Kullanılabilecek tüm elementler için MSDN üzerindeki şu adrese gitmenizi öneririm.

Klavye Başına

Yazımızın bu bölümünde basit bir örnek geliştirmeye çalışıyor olacağız. Amacımız temel seviyede**MSBuild** aracını kullanmak ve konsepti anlamaya çalışmak olacaktır. İlk olarak basit bir **C#** koduna ihtiyacımız var. Bu amaçla
örneğin **C:\Samples\HowToMSBuild** isimli klasör altında, aşağıdaki içeriğe sahip bir **C#** dosyası oluşturduğumuzu düşünelim.



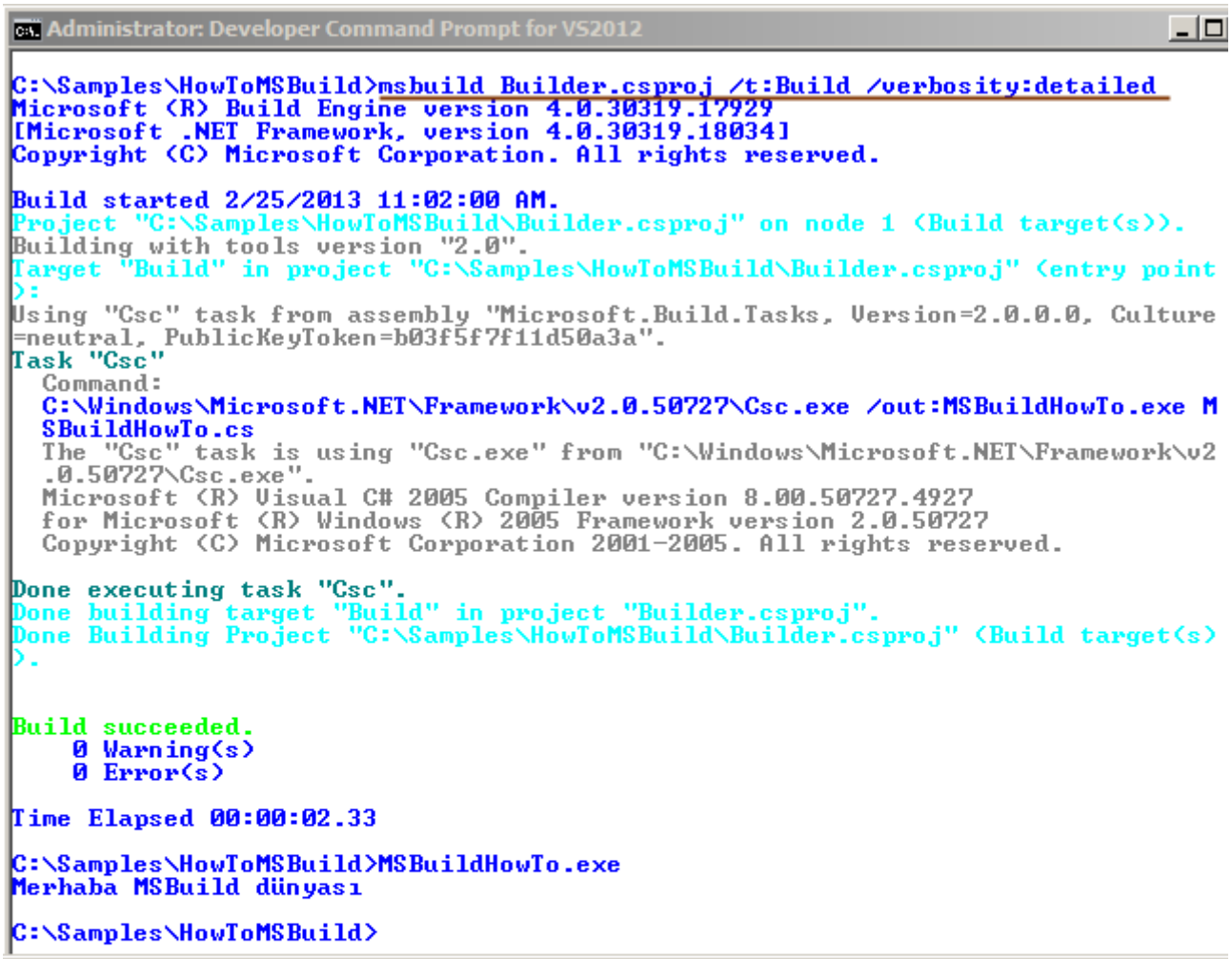
Bu adımdan sonra yine **Notepad2** aracını kullanarak aşağıdaki içeriğe sahip bir **csproj** dosyası üreterek devam edelim.

```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <ItemGroup>
    <Compile Include="MSBuildHowTo.cs" />
  </ItemGroup>
  <Target Name="Build">
    <Csc Sources="@(Compile)"/>
  </Target>
</Project>
```

Project içerisinde bir adet **ItemGroup** ve **Target** elementi yer almaktadır. **ItemGroup** içerisinde yer alan **Compile** elementi, **Include niteliğinde*(attribute)***, derleme işlemine tabi olacak dosyayı belirtmektedir. **Target** altındaki **Csc** elementinin **Sources** niteliğinde ise **Compile** isimli bir komut yer almaktadır. Dolayısıyla bu **Task**, **MSBuild** aracına, **Compile** elementine dahil edilen dosyanın derlenmesi gerektiğini söylemektedir.

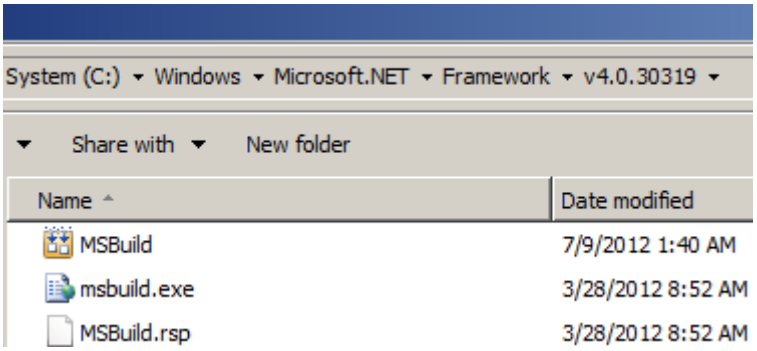
Bu içeriği **Builder.csproj** adı ile kaydettikten sonra ise sıradaki operasyon, **MSBuild** komut satırı aracını kullanarak inşa işlemini yürüttürmektir. Bunun için **MSBuild'** u aşağıdaki gibi kullanabiliriz.

msbuild Builder.csproj /t: Build /verbosity: detailed



Görüldüğü gibi çalışma sonrasında **MSBuildHowTo** isimli bir **exe** dosyası oluşmuştur. Söz konusu**exe** dosyasını doğrudan çalıştırdığımızda ise, **pre-precessor** direktifi dışında kalan kod parçasının yürütüldüğü gözlemlenecektir.

MSBuild özellikle kurulu olan .Net Framework versiyonuna bağlı olaraktan Microsoft.Net\Framework\vX.X.XXXXX altında yer almaktadır. Örneğin ben kendi sistemimde aşağıdaki klasörde yer alan sürümü kullandım.



Bunu sistem' de Path olarak belirtebiliriz ama dilerseniz doğrudan Visual Studio Command Prompt' tan da yararlanabiliriz.

CSPROJ İçeriğini Genişletelim

Şimdi **csproj** dosyasının içeriğini biraz daha genişletelim ve aşağıdaki hale getirelim.

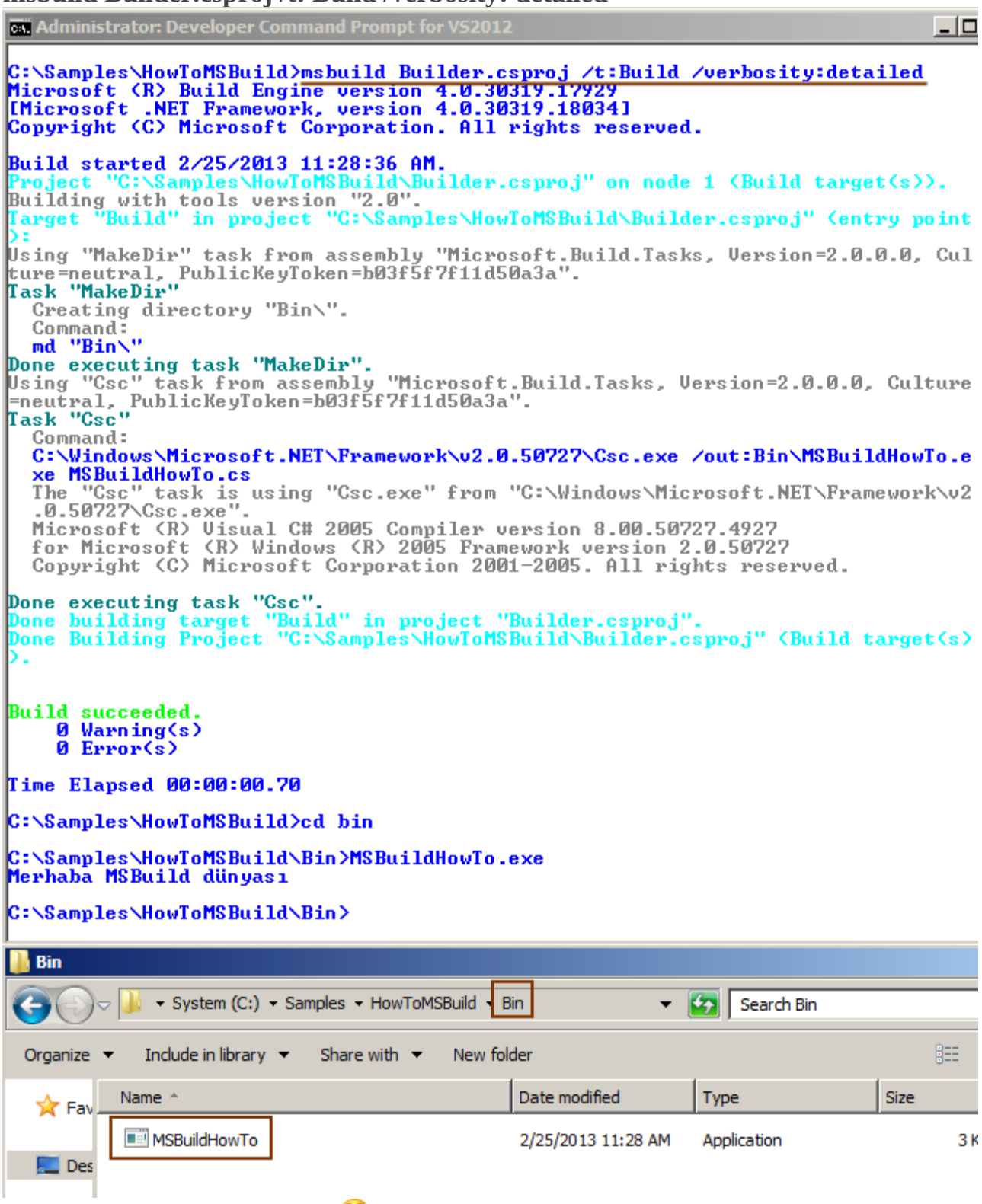
```
<Project xmlns="http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <AssemblyName>MSBuildHowTo</AssemblyName>
    <OutputPath>Bin\</OutputPath>
  </PropertyGroup>
  <ItemGroup>
    <Compile Include="MSBuildHowTo.cs" />
  </ItemGroup>
  <Target Name="Build">
    <MakeDir Directories="$(OutputPath)" Condition="!Exists('$(OutputPath)')" />
    <Csc Sources="@(Compile)"
OutputAssembly="$(OutputPath)$(AssemblyName).exe"/>
  </Target>
</Project>
```

Bu sefer **PropertyGroup** içerisinde **assembly** adını***(AssemblyName elementi)*** ve **build** işlemi sonrası ortaya çıkacak olan çıktının konuşlandırılacağı klasörü de belirttik***(OutputPath)***. Diğer yandan bu örneğimizde yer alan en önemli kısım **Build** isimli **Target** elementinin içeriğidir. Dikkat edilecek olursa ilk sırada **MakeDir** isimli element yer almaktadır. Burada **$(OutputPath)** ile çıktının yapılacağı klasör işaretlenirken aslında **PropertyGroup>OutputPath** elementinin içeriği ifade edilmektedir. Önemli olan kısım ise **Condition** niteliğinde yazılan değerdir.**!Exists('$(OutputPath)')** ifadesi ile şu söylenmektedir. **"OutputPath yoksa…"**. Buna göre eğer**Condition** sağlanırsa, **MakeDir OutputPath**' in oluşturulması gerektiğini belirtecektir.

Csc elementinde ise derleme işlemi için kullanılacak olan kaynak **Sources** elementi ile ifade edilirken, aslında **ItemGroup** içerisindeki **Compile** elementinde yer alan **Include** niteliğinin değeri işaret edilmektedir. **OutputAssembly** niteliğine atanan değer ise, **OutputhPath** özelliğinin belirttiği klasörün altına **AssemblyName'** in işaret

ettiği isimle bir **exe** üretilmesi gerektiğini belirtmektedir. Buna göre komut satırından yapılan **MSBuild** çağrısı sonucu aşağıdaki gibi olacaktır.

msbuild Builder.csproj /t: Build /verbosity: detailed



Oldukça zevkli öyle değil mi? 😊

Target Belirtmek

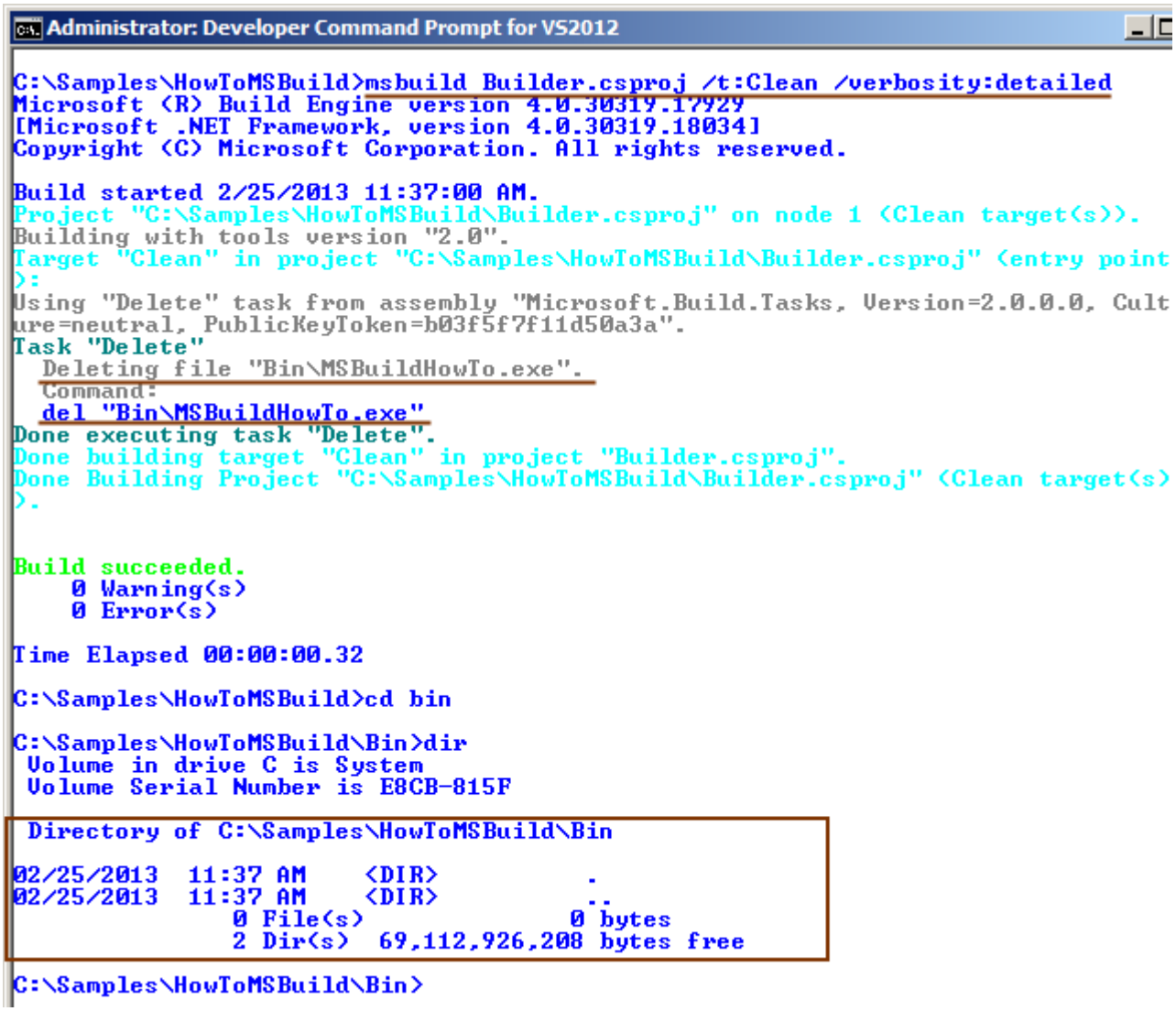
Öyleyse gelin olayı biraz daha genişleterek devam edelim.

```
<Project DefaultTargets="Build" xmlns="
http://schemas.microsoft.com/developer/msbuild/2003">
  <PropertyGroup>
    <AssemblyName>MSBuildHowTo</AssemblyName>
    <OutputPath>Bin\</OutputPath>
  </PropertyGroup>
  <ItemGroup>
    <Compile Include="MSBuildHowTo.cs" />
  </ItemGroup>
  <Target Name="Build">
    <MakeDir Directories="$(OutputPath)" Condition="!Exists('$(OutputPath)')" />
    <Csc Sources="@(Compile)"
OutputAssembly="$(OutputPath)$(AssemblyName).exe" />
  </Target>
  <Target Name="Clean" >
    <Delete Files="$(OutputPath)$(AssemblyName).exe" />
  </Target>
  <Target Name="Rebuild" DependsOnTargets="Clean;Build" />
</Project>
```

Bu seferki çalışmada **3 adet Target** elementi söz konusudur. Ancak her birinin **Name** niteliğinin değerleri farklıdır. Buna göre, **Build, Clean** ve **Rebuild** isimli **Target'** lar olduğunu ve aslında bunların **3 ayrı Task**' ı ifade ettiğini düşünebiliriz. Yani **MSBuild** aracını kullanırken **/t: [TargetNameValue]** şeklinde bir yaklaşım ile, istediğimiz **Target'** ın yürütülmesini sağlayabiliriz. Söz gelimi;

msbuild Builder.csproj /t: Clean /verbosity: detailed

şeklinde yapılan çağrı sonucunda, proje dosyasındaki **Target** elementlerinden **Clean** isimli olanı çalıştırılacaktır. Bu elementte ise **Delete** isimli bir alt element yer almakta olup **Files** niteliğinde belirtilen kritere göre, **Bin** klasöre altında **assembly** adı ile duran **exe** dosyasının silinmesi gerektiği ifade edilmektedir.

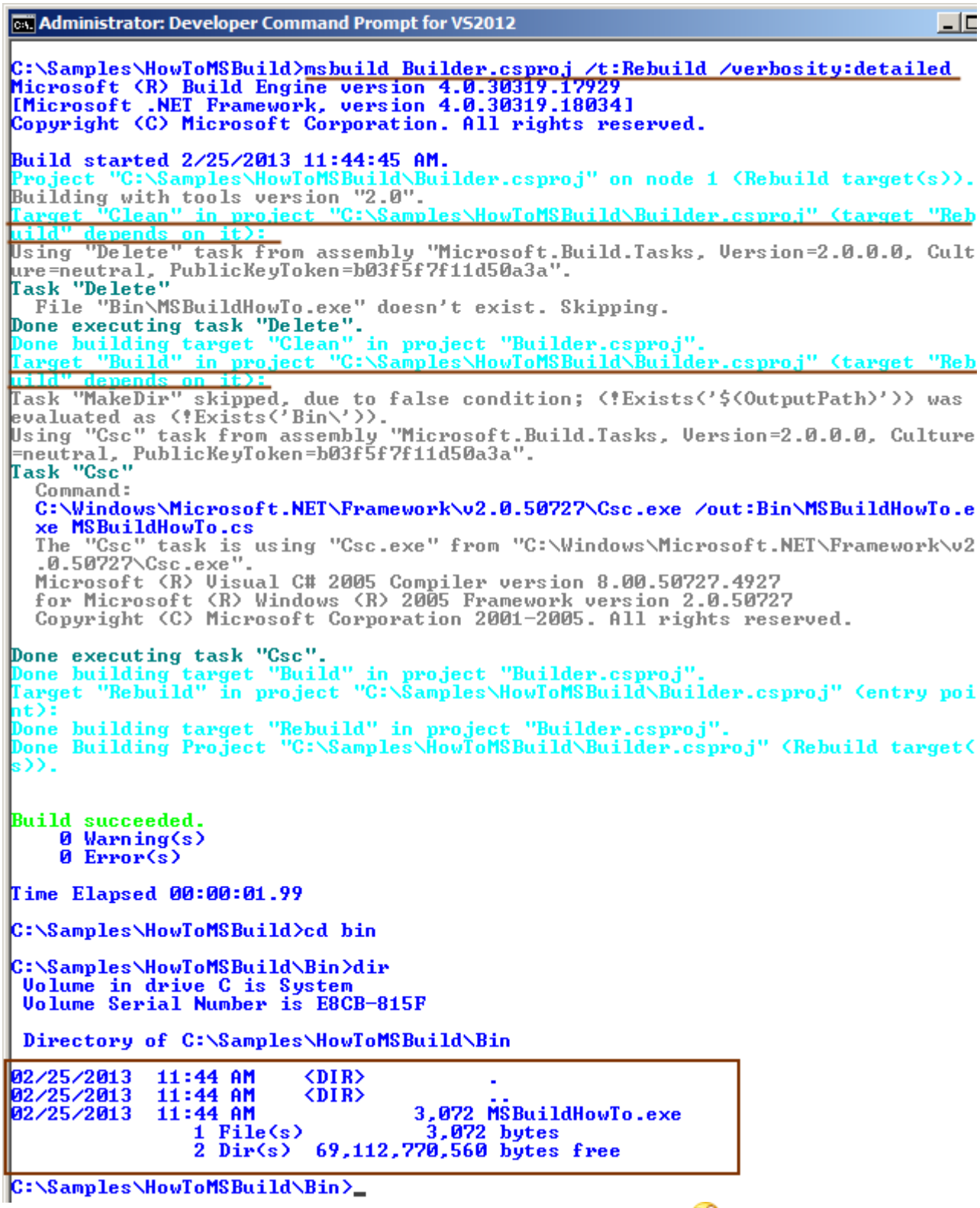


Eğer **Rebuild** takısını kullanırsak bu durumda **Target->Name** niteliği **Rebuild** olan bölüm devreye girecektir. Bu vakada dikkat edilmesi gereken ise **DependsOnTargets** niteliği içerisinde yazılan**Clean;Build** ifadesidir. Yani şunu ifade etmiş oluruz;

Ey MSBuild!…Önce Clean isimli Target, sonrasında ise Build isimli Target içeriğini icra et

ki bu durumda **Bin** klasörü içeriği silinecek ve tekrardan bir derleme işlemi yapılarak, orada ilgili **exe**çıktısının üretilmesi sağlanacaktır.

msbuild Builder.csproj /t: Rebuild /verbosity: detailed



Örnekler daha da çoğaltılabilir. Yapılabilecek pek çok şey var Bunun için mutlaka **MSDN** orjinli bir kaynağa başvurmanızı veya yazımızın sonunda belirttiğimiz tarzdaki bir kitabı tedarik etmenizi öneririm. Ancak temel mantığı ifade edebildiğimizi varsayıyorum. Bundan sonrasında **Condition'**lara ve kullanılabilecek **element/attribute** tiplerine bakılması yeterli olur düşüncesindeyim.

.Net Framework 4.5 ile Gelen Yenilikler

MSBuild ürününe **Framework 4.5** sürümü ile birlikte bazı geliştirmeler ve yeni özellikler de katılmıştır. Henüz inceleme fırsatı bulamadığım bu özellikleri aşağıdaki maddeler ile özetleyebiliriz.

- **ARM*(Advanced RISC Machine)*** desteği gelmiştir.
  Yani **Build** çıktıları **ARM** işlemcilerini hedef alacak şekilde üretilebilir.
- Bir **Task** *(Target elementi ile belirtilen bir görev diyelim)* süreç dışı***(Out of Process)*** modda çalışmaya zorlanabilir.
- Yeni bazı **XML element** ve **nitelikleri*(Attributes)*** gelmiştir.
- Klasik bir cümle olacak
  ama, **Performans*(Performance)*** ve **Ölçeklendirme*(Scabilitiy)*** noktasında iyileştirmeler vardır

MSBuild' un etkili kullanımı üzerine geliştirilmiş bazı araçlar da vardır. Görsel arabirimleri olan bu araçlar yardımıyla MSBuild akışlarını daha kolay yönetebiliriz. Attrice firmasının bu alanda ön plana çıkan Microsoft Build Sidekick isimli ürünü gibi.

Öneri Kitap

Microsoft' un **Visual Studio** ailesi ve geliştirme platformu oldukça geniş bir alana yayılmakta olup, pek çok notkasında uzmanlık gerektiren yapılar içermektedir. Bu sebepten söz konusu yapılara yönelik pek çok yayın da*(kitap, official site, blog vb)* mevcuttur.
Örneğin **MSBuild** tarafında daha önceden yayınlanmış olan **Inside The Microsoft Build Engine** isimli kitabın **Nisan** ayı içerisinde yayınlanan yeni bir tamamlayıcı baskısı mevcuttur. Yaklaşık olarak 120 sayfalık bir kitap olmasına rağmen odaklandığı konu özünde **MSBuild** ürünüdür.
Kitaba Amazon üzerinden bu adres yardımıyla erişebilirsiniz.
Böylece geldik bir yazımızın daha sonuna. Bu yazımızda kısada olsa, **MSBuild** platformunu ve **XML**tarafındaki **betikleri*(Scripts)*** anlamaya çalıştık. Ağırlıklı olarak bir inşa sürecine müdahale edebildiğimizi, bunun için platformun sunduğu bazı standart element ve niteliklerin olduğunu gördük. Devamı sizde artık Bir başka yazımızda görüşünceye dek hepinize mutlu günler dilerim.
[Orjinal Yazım Tarihi – 25/02/2013]
[Güncel bilgi –> MS Build is now part of Visual Studio!]

STSdb ile Hello World

Perşembe, 25 Temmuz 2013 09:24 by bsenyurt

[İlk Yazım Tarihi 2013-01-15]
Merhaba Arkadaşlar,
Matrix Reloaded' ı seyrettiğim zamanları düşündüğümde, anımsadıklarım arasında heyecanlı aksiyon sahnelerinde yer alan ve eski Amerikan stilini de yansıtan kocaman otomobiller vardı. *(Hatta bildiğim kadarı ile ikinci dünya savaşı sonrası çelik stoklarının fazlalığı nedeniyle Amerikan otoları hep kocaman olmuşlardı)*
General Motors firmasına ait olan otomobillerden birisi de, **Cadillac STS**' in farklı bir versiyonu olan **CTS** idi. Tabi ben konuyu bir şekilde bu günkü yazının konusu olan **STSdb**' ye getirmek istediğimden Cadillac STS' e ait bir fotoğrafa yer vermek istedim Öyleyse vakit kaybetmeden konumuza geçelim.
Bilindiği üzere bir süredir **NoSQL(Not only SQL)** veritabanlarını incelemeye*(öğrenmeye)* çalışıyoruz. Daha önceki yazılarımızda **Apache Cassandra, RavenDB** ve **DEX** ürünlerine bir göz atmış ve **.Net**uygulamalarında nasıl kullanılabileceklerini görmüştük. Klasik olarak yaptığımız Hello World uygulamalarının bir benzerini de bu gün inceleyeceğimiz **STSdb** için gerçekleştiriyor olacağız.
Bu veritabanı için bazı kaynaklarda **Revolutionary*(Devrimci, devrimsel, devrim niteliğinde)***sıfatı kullanılmış. Gerçekten böyle bir sistem olduğunu ispat etmemiz oldukça zor tabi. Yine de bu konu ile ilişkili yapılmış bir kaç veritabanı testine bakmak az da olsa fikir verebilir. Söz gelimi kaynağının ne kadar güvenilir olduğunu tam olarak bilmediğim bu adreste, karşılaştırıldığı veritabanlarına göre en hızlısı olduğu iddia edilmiş. Aslında bizim temel amacımız **key-value** modeline göre çalışan **NoSQL** veritabanlarından bir diğerini incelemektir Dilerseniz **STSdb** ürününü kısaca mercek atlına almaya başlayalım.
STSdb ile ilişkili olarak karşılaştığım ilk sıkıntı dokümantasyonun yeteri kadar doyurucu olmamasıydı. MSDN benzeri API dokümantasyonu ve bir kaç örnek kod parçası haricinde, forumlarda da yeteri kadar doyurucu içeriğe rastlayamadım. Amazon.com' da konu ile ilişkili bir kitabı da ne yazık ki bulamadım. Pek tabi zaman ilerledikçe bu durum değişebilir.
Hal böyle olunca biraz dene-keşfet modeline göre öğrenilmeye çalışılan bir ürün oldu benim için. Kaldı ki API dokümantasyonuna baktığımızda gerçekten çok önemli görünen pek çok fonksiyonellik var. Örneğin Snapshot almak için XTable sınıfına ait bir metod var ama örnek bir kod parçası yok. Dolayısıyla biraz kurcalanması gereken bir ürün olarak karşımıza çıkmakta.
STSdb, Key-Value modeline göre çalışan açık kaynak*(GPL 2 ve GPL 3. Ancak Community lisanslaması da yapılabiliyor)* **NoSql** depolama **API**' lerinden birisidir. **.Net**

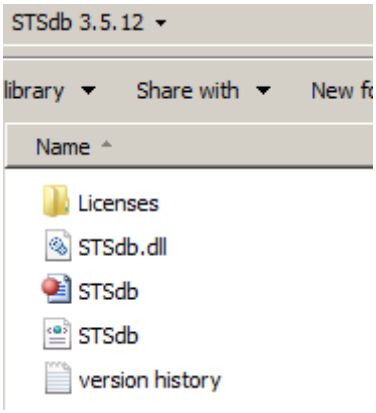
Framework üzerinde **C#**programlama dili kullanılarak geliştirilmiş gömülü*(Embedded)* bir sistem olarak karşımıza çıkıyor. Her platformu destekleyebilir ilkesini ilgili platformlarda **Mono** yüklü olması halinde karşılamakta*(Tam bir plaform bağımsızlık olduğunu ifade edemeyiz ama Mono ile Linux, MacOS X ve Unix gibi sistemlere de entegre edilebilir)*

Genel Özellikler

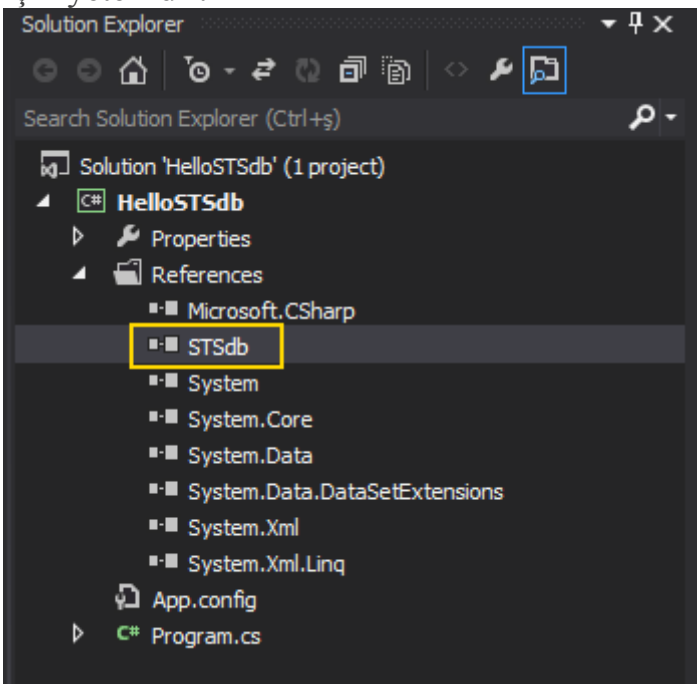
Genel özelliklerini ise aşağıdaki maddeler halinde ifade edebiliriz.

- **.Net Framework**' e gömülü olarak geliştirildiğinden doğrudan uygulama ile ilişkilendirilip çalıştırılabiliyor. Yani ek bir konfigurasyon hazırlığına ihtiyaç yok.
- Veriyi **disk** üzerinde, **bellekte*(In-Memory)*** veya her ikisinin kombinasyonu olacak şekilde hibrid bir yapıda tutabilir.
- Kullanılabilirlik alanları oldukça geniştir. İnsan Makine Arayüzleri*(Human Machine Interface)*, Süreç kontrol sistemleri*(Process Control System)*, Otomotiv Endüstürisi, çeşitli tipte finansal uygulamalar vb...
- Mantıksal modeli iki katmandan oluşmaktadır. Birinci kat dahili **dosya sistemidir*(File System)***. Bu katman 2üzeri64 dosyaya*(Her biri ile de 2üzeri64 byte' a çıkılabilir)* kadar destek vermektedir.
  İkinci katman ise tabloların tutulduğu **veritabanıdır*(Database Layer)***. Burada tablo yönetimine ilişkin işlemler yapılır. Tabloların oluşturulması, silinmesi, tablolar üzerinde ileri/geri hareket edilmesi, veya düzenlenmesi bu operasyonlar arasında sayılabilir.
- **Transaction** desteği vardır. Bu nedenle **Atomicity, Consistency, Isolation** ve **Durability** olarak bilinen **ACID** prensiplerini karşılamaktadır.
- Karmaşık bir **.Net** tipi, **primary key** olarak kullanılabilir.
- **Snapshot** özelliği sayesinde **gerçek zamanlı yedekleme*(Real-time Backup)*** imkanı da sunar.
- Çok doğal olarak **.Net Framework** üzerinde geliştirilmiş olduğundan **LINQ*(Language INtegrated Query)*** desteğine sahiptir.
- Veriyi bir algoritma yardımıyla sıkıştırmaktadır. Sıkıştırma önemli ölçüde yer kazanımına da imkan sağlamaktadır.
- Belki de en önemli özelliklerinden birisi tutabileceği tablo veya kayıt için bir üst limit değerinin olmayışıdır. Bu nedenle çok büyük boyutlu veri kümeleri için tercih edilebilir.

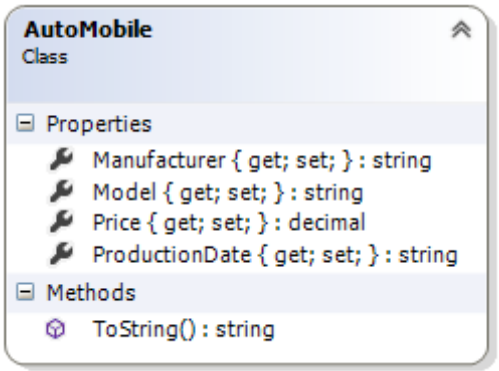
İlk olarak ürünü tedarik etmemiz gerekiyor tabi ki  Bunun için şu adresi kullanabiliriz. İndirilen içerik aşağıdaki gibidir. *(Yazının yayınlandığı tarihi itibariyle 4.0 RC sürümüde mevcut)*



Tahmin edileceği üzere STSdb.dll assembly' ının, projeye referans edilmesi kullanılması için yeterlidir.



Console uygulaması olarak geliştireceğimiz programımızda **Hello World** demek maksadıyla aşağıdaki kod içeriğini yazdığımızı düşünelim. **Table Record** olarak **AutoMobile** isimli bir sınıftan yararlanıyor olacağız.



Veri Ekleme Operasyonu

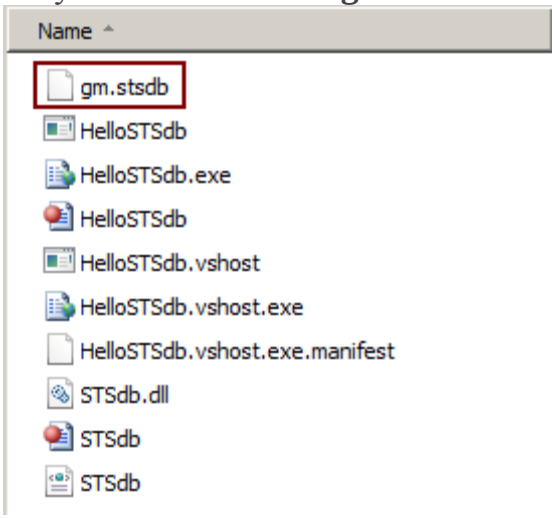
ilk olarak veri ekleme işini sembolize edelim.

```
using STSdb.Data;
using System;
using System.Linq;
namespace HelloSTSdb
{
   class Program
   {
      static void Main(string[] args)
      {
         Add();
      }
      private static void Add()
      {
         var index1 = new XKey<byte[], string>(Guid.NewGuid().ToByteArray(),
"Kadillak STS");
         var index2 = new XKey<byte[], string>(Guid.NewGuid().ToByteArray(),
"Lamborjini Diyablo");
         var index3 = new XKey<byte[], string>(Guid.NewGuid().ToByteArray(),
"Ferrarriim");
         var index4 = new XKey<byte[], string>(Guid.NewGuid().ToByteArray(),
"Doğanım");
         #region Örnek veri kümesi oluşturulması
         using (StorageEngine sEngine = StorageEngine.FromFile("gm.stsdb"))
        {
           var table = sEngine.Scheme
               .CreateOrOpenXTable<XKey<byte[], string>, AutoMobile>(new
Locator("Automobile"));
            table.KeyMap = new XKeyMap<byte[], string>(null, 16, null, 512);
            sEngine.Scheme.Commit();
            table[index1] = new AutoMobile
           {
               Manufacturer = "Ceneral Motor Kampani",
               Model = "STS Turbo CRDI AK/S",
               Price = 90000,
               ProductionDate = "2005"
            };
            table[index2] = new AutoMobile
            {
               Manufacturer = "İtalyan cob",
               Model = "Diablo III",
               Price = 250000,
```

```
            ProductionDate = "2008"
        };
        table[index3] = new AutoMobile
        {
            Manufacturer = "Ferrari",
            Model = "799",
            Price = 350000,
            ProductionDate = "2013"
        };
        table[index4] = new AutoMobile
        {
            Manufacturer = "Türk işi",
            Model = "Doğan SLX/AK 8 ileri",
            Price = 450,
            ProductionDate = "2014"
        };
        table.Commit();
        table.Close();
      }
      #endregion
    }
  }
  public class AutoMobile
  {
    public string Model { get; set; }
    public string Manufacturer { get; set; }
    public string ProductionDate { get; set; }
    public decimal Price { get; set; }
    public override string ToString()
    {
      return string.Format("{0} {1}({2}) by {3}", ProductionDate, Model, Price,
Manufacturer);
    }
  }
}
```

Dikkat edileceği üzere **StorageEngine** ana nesnedir. **using** bloğu ile kullanılabilen, bir başka deyişle **Dispose** edilebilir olan bu nesne örneğinin **FromFile** metodundan yararlanarak depolama işlemini yapan dosya belirtilmektedir. **Key** değerleri için **XKey<byte[],string>** tipinden nesneler örneklenmiştir. Veriyi tutacak olan ise **XTable** tipine ait nesne örnekleridir. Bu tanımlama için**StorageEngine** nesne örneği

üzerinden hareket edilir ve **CreateOrOpenXTable** metoduna başvurulur. Bir başka deyişle bir şema tanımı yaptığımızı ifade edebiliriz.
Dikkat edilmesi gereken noktalardan birisi
de **Key** olarak **Primitive** tip kullanmadığımız için**KeyMap** özelliğine bir değer atama zorunluluğumuz olmasıdır. Bu atama sırasında, **XKey** içerisindeki tip yapısına göre maksimum boyutlar belirtilmek zorundadır. Aksi durumda çalışma zamanında **nullreferans** nedeniyle bir **istisna*(Exception)*** alınacaktır.
Veri ekleme işlemi aslında son derece basittir. **XTable** tipinin indeksleyici operatörüne tanımlanan**Key** örnekleri atanır. Eşitlikten sonra ise yine bildirimi yapılmış olan tipten*(ki örneğimizde AutoMobile sınıfına ait nesnelerdir)* örnekler atanır. Tüm işlemlerin tamamlanmasının ardından bir**Close** ve **Commit** çağrısının yapılması, verilerin kalıcı olarak yazılması açısından kritiktir. **Console**uygulamasını bu şekilde çalıştırdığımızda dosya sistemi üzerinde **gm.stsdb** isimli bir dosya oluşturulduğu gözlemlenir.



Veri Okuma

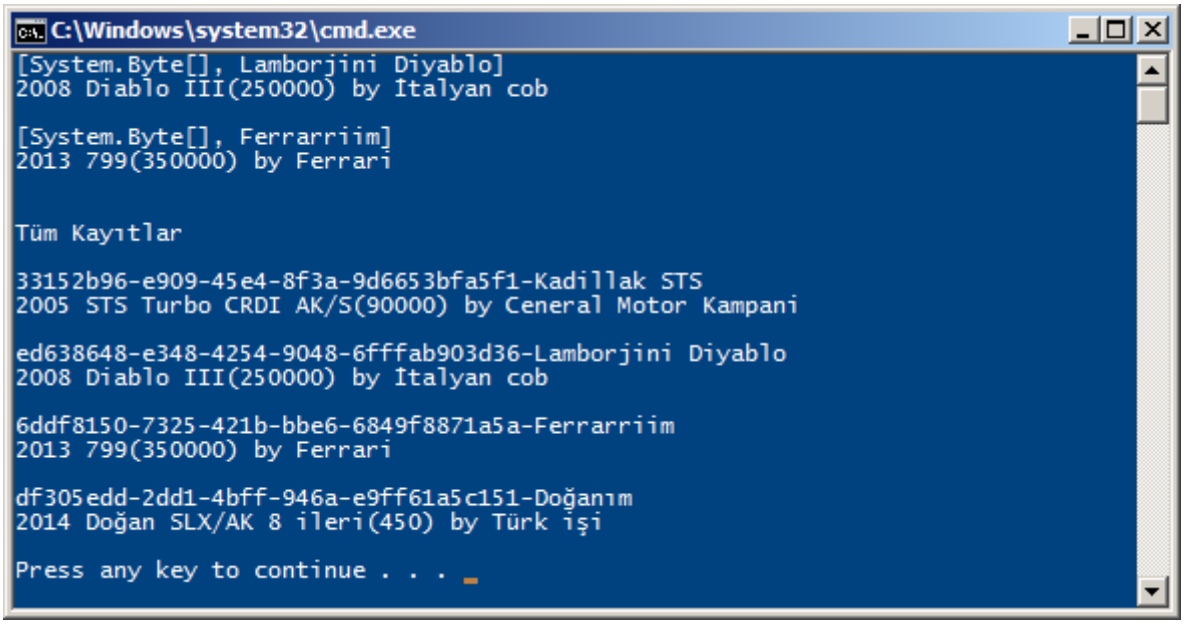
Gelelim veri okuma işine. Örneğin az önce eklemiş olduğumuz XTable içeriğini ekrana yazdırmayı deneyelim. Bu amaçla aşağıdaki Read metodunu ele alabiliriz.

```
private static void Read()
{
   #region Veri okunması
   using (StorageEngine sEngine = StorageEngine.FromFile("gm.stsdb"))
  {
      var table = sEngine.Scheme
        .OpenXTable<XKey<byte[], string>, AutoMobile>(new
Locator("Automobile"));
      foreach (var row in table.Where(r => r.Record.Price > 100000))
      {
         Console.WriteLine("{0}\n{1}\n"
           , row.Key.ToString()
```

```
            , row.Record.ToString()
            );
        }
        Console.WriteLine("\nTüm Kayıtlar\n");
        foreach (var row in table)
        {
           Console.WriteLine("{0}-{1}\n{2}\n"
            , (new Guid(row.Key.SubKey0)).ToString()
            , row.Key.SubKey1
            , row.Record.ToString()
            );
        }
        table.Close();
    }
    #endregion Veri okunması
}
```

Okuma işleminde başrol oyuncusu olarak yine, **StorageEngine** sınıfı devreye girmektedir. **FromFile**metodu ile **sts** veritabanı dosyası yüklendikten sonra ise, veri okuması yapılmak istenen **XTable**' ın açılması gerekmektedir. Bu işlem için **OpenXTable** metodundan yararlanılmaktadır. **Generic**parametreye***(<XKey<byte[], string>, AutoMobile>)*** dikkat edileceği üzere, oluşturulan **XTable**' ın şema yapısına uygun olacak şekilde verildiği görülmektedir.

table nesne örneği elde edildikten sonra basit bir **foreach** iterasyonu ile kayıtlar arasından dolaşılabilinir. Hatta örnekte görüldüğü gibi **Where** genişletme metodundan yararlanılarak bir **LINQ**sorgusunun icra edilmesi de sağlanabilir. **lambda(=>)** operatörü etrafında kullanılan **r** değişkeni,**AutoMobile** sınıfından bir nesne örneğidir. Dolayısıyla özellikleri sorgulamada filtre kriteri olarak kullanılabilir. Uygulamanın çalışma zamanı çıktısı aşağıdaki gibi olacaktır.



Temel olarak **key-value** teorisini baz alarak çalışan **STSdb** sisteminde, tablo anahtar*(Table Key)* ve kayıtlarının*(Table Record)* hangi türlerden oluşabileceği belirlidir. Burada oldukça geniş bir nesne yelpazesinin olduğunu ifade edebiliriz.

Tablo anahtarlarına***(Table Keys)*** baktığımızda aşağıdaki tiplerin desteklendiğini görmekteyiz.

- .Net primitive types *(Boolean, Byte, Char, DateTime, Decimal, Double, Int16, Int32, Int64, TimeSpan, SByte, Single, String, UInt16, UInt32, Uınt64)*
- byte[] dizisi*(Örneğin bir resmin byte içeriğini Key olarak kullanabilirsiniz)*
- STSdb' ye özgün Locator tipi
- Tn primitive olmak suretiyle XKey<T1, T2[, T3[, T4[, T5[, T6[, T7]]]]]>,
- IKeyMap<TKey> arayüzü*(Interface)* türetmeleri

Tablo kayıtlarına***(Table Records)*** baktığımızda ise benzer tiplerin desteklendiğini görürüz.

- .Net primitive types *(Boolean, Byte, Char, DateTime, Decimal, Double, Int16, Int32, Int64, TimeSpan, SByte, Single, String, UInt16, UInt32, Uınt64)*
- byte[] dizisi, enum, Type , Image, Icon, MemoryStream, Guid, XElement gibi diğer tipler
- STSdb' ye özgü Locator ve BlobStream*(Çok büyük boyutlu binary içerikleri alabilirsiniz demek)*
- Serileştirilebilir nesneler(Serializable Objects)
- IBinaryPersist<IIndexer<TRecord>> arayüzünün türevleri *(Özellikle kendi Persistence mekanizmamızı yazmak istediğimiz durumlarda yine bu arayüzden türetilmiş tipler söz konusu olacaktır. Bir başka deyişle Binary olarak yazma ve okuma işlemlerine müdahale edip Persistence şeklini değiştirebiliriz)*

Görüldüğü üzere özellikle **Key** değerlerini karmaşık tipler şeklinde tutmak mümkün. Burada nesne yönelimli dünyanın*(Object Oriented World)* faydalarını da görüyoruz. **Interface** türetmeli tiplerin **Key**veya **Record** olarak kullanılabilmesi bunun en güzel örneği belki de.

STSdb' yi göz önüne alırken **.Net Framework** platformu için garanti, embedded bir **NoSQL** çözümü olduğunu ve özellikle **Cross-Platform** alanında **Mono**' ya bağımlı olduğunu unutmamalıyız. Bu bir kısıtlama gibi görünse de, ürünlerini .Net ortamında geliştirenler için çok da büyük bir sorun değil. Ayrıca sunduğu üst limitsiz depolama alanı, **Transaction**, **Snapshot, sıkıştırma** kabiliyetleri vb diğer imkanlar nedeniyle, özellikle büyük boyutlu veri kümeleri için ideal görünüyor. **STS** firmasının bir ürünü olan bu veritabanı, şirketin uzun yıllardır **FOREX** pazarı üzerine yaptığı uygulama geliştirmelerine ait tecrübelerinin bir çıktısı aslında.

Bu yazımızda kısaca **STSdb** ürününü incelemeye çalıştık. Bir Hello World uygulaması geliştirdik ve veri ekleme ile okuma işlemlerinin nasıl yapılabildiğine baktık. Elbette örnekleri zenginleştirmek ve gerçek saha tecrübesini yapmak sizin elinizde. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

HelloSTSdb.zip (249,52 kb)

[İlk Yazım Tarihi 2013-01-15]

Word Dosyası İçerisinden Entity Framework' e

Perşembe, 18 Temmuz 2013 14:26 by bsenyurt

[Orjinal Yazım Tarihi 03-15-2013]

Merhaba Arkadaşlar,

Geliştirdiğimiz veya kullanmakta olduğumuz yazılım ürünlerine dahil olan, farklı segmentlerden gelen pek çok kullanıcı profili vardır. Farklı profillerin olması, bazı hallerde geliştirilmekte olan ürünlerin başarısını doğrudan etkilemektedir.

Bir fotoğraf işleme programını geliştirirken çoğu zaman annemizin olası kullanıcı profilleri arasına gireceğini pek düşünmeyiz. Genellikle fotoğraf işleme programını kullanacak olanların, en azından temek düzeyde fotoğrafçılık bilgisine sahip olduğunu kabul eder, menü komutlarını buna göre belirler, arayüzü buna göre hazırlarız. Ama bazı uygulamalarda annemizi hedef alır ve çektiği fotoğraflara kolayca efekt uygulamasına bir kaç basit adımda olanak tanır. Örneğin **instagram**' ın **iPhone**uygulamasında olduğu gibi.

Bir yazılım geliştirme ürününü tasarlarken ise, son kullanıcının programcılar olduğunu varsayar ve arabirimin karmaşık olmasının herhangibir sorun oluşturmayacağını düşünürüz. Oysaki **Team Foundation Server** gibi geniş ürün yelpazesine sahip aileler düşünüldüğünde, işe dahil olan farklı profildeki kullanıcılar için işleri kolaylaştırıcı şekilde düşünüldüğüne şahit oluruz.

Söz gelimi proje yöneticisinin, **Ms Project** ürünü ile **TFS**' e entegre olabildiğini, **Scrum Master**' ın isterse tüm **Product Backlog** içeriğini bir **Excel** dosyasına indirip senkronize edebildiğini, **Yazılım Mimarının** ve **Geliştiricilerin, Visual Studio** ile ortama bağlanabildikleri ama birim müdürü için sadece **Team Explorer**' ın kafi gelebildiğini, **İş Analisti** gibi geliştiriciden farklı profile sahip elemanların ise Web arayüzünü kullanarak basit bir şekilde **Requirement** ekleyebildikleri görürüz.

Sonuç olarak geliştirilen ürünün kullanım alanına dahil olan tüm personeli göz önüne alarak hareket etmemiz gerektiğinin farkında olmalıyız. İşte bu yazımızda bu konuyu tecrübe etmeye çalışacağımız bir örnek geliştiriyor olacağız. Şimdi izleyen paragraftaki senaryoyu göz önüne alalım.

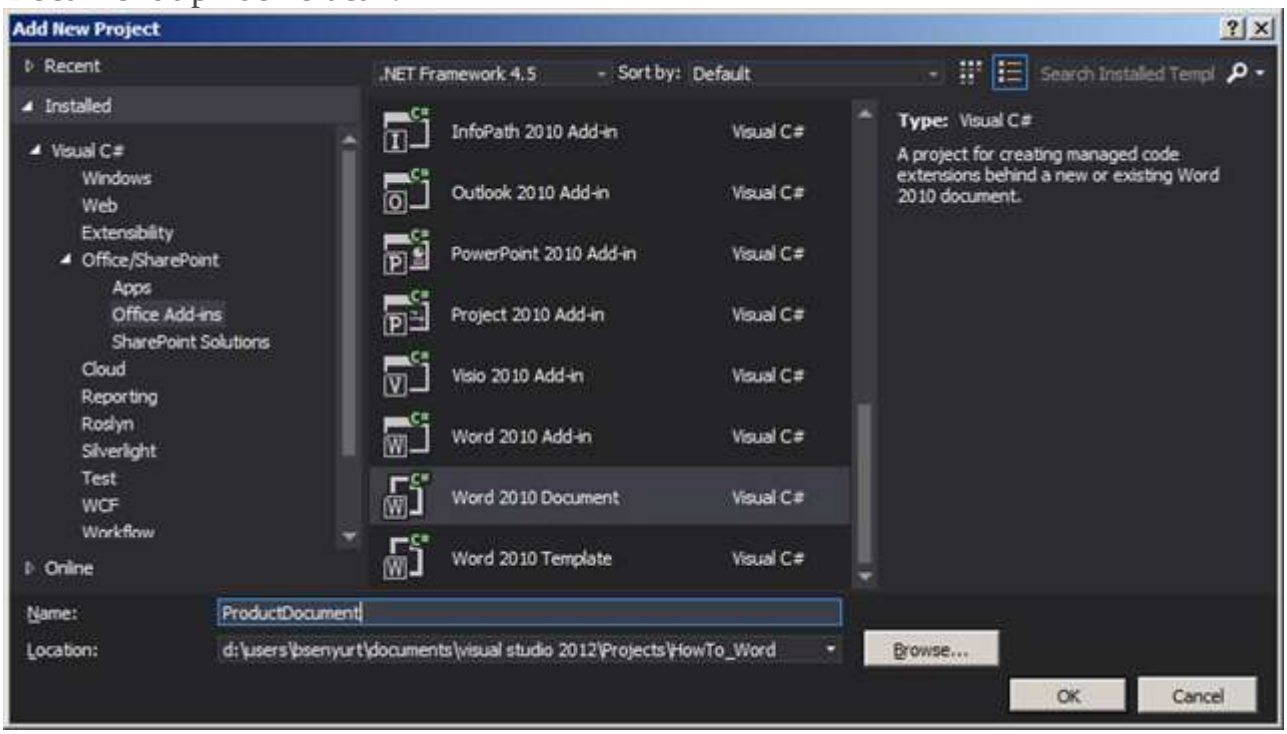
Senaryo

Selim Usta bir oto yedek parçacıdır. İşlerini kolaylaştırmak amacıyla uzun zamandır bilgisayar öğrenmeye çalışmaktadır ve en nihayetinde **Word, Excel** gibi ofis uygulamalarını basit seviyede de olsa kullanmaya başlamıştır. En sık yaptığı işlemlerden birisi ise stoğuna dahil ettiği yeni ürünleri bilgisayarda bir **Word** dosyasına kaydetmektir. Lakin zaman içerisinde **Word** dosyası epeyce şişmiş, içeride bir şey aramak epeyce zorlaşmıştır. Üstelik içeriğin düzgün bir formatı da yoktur.

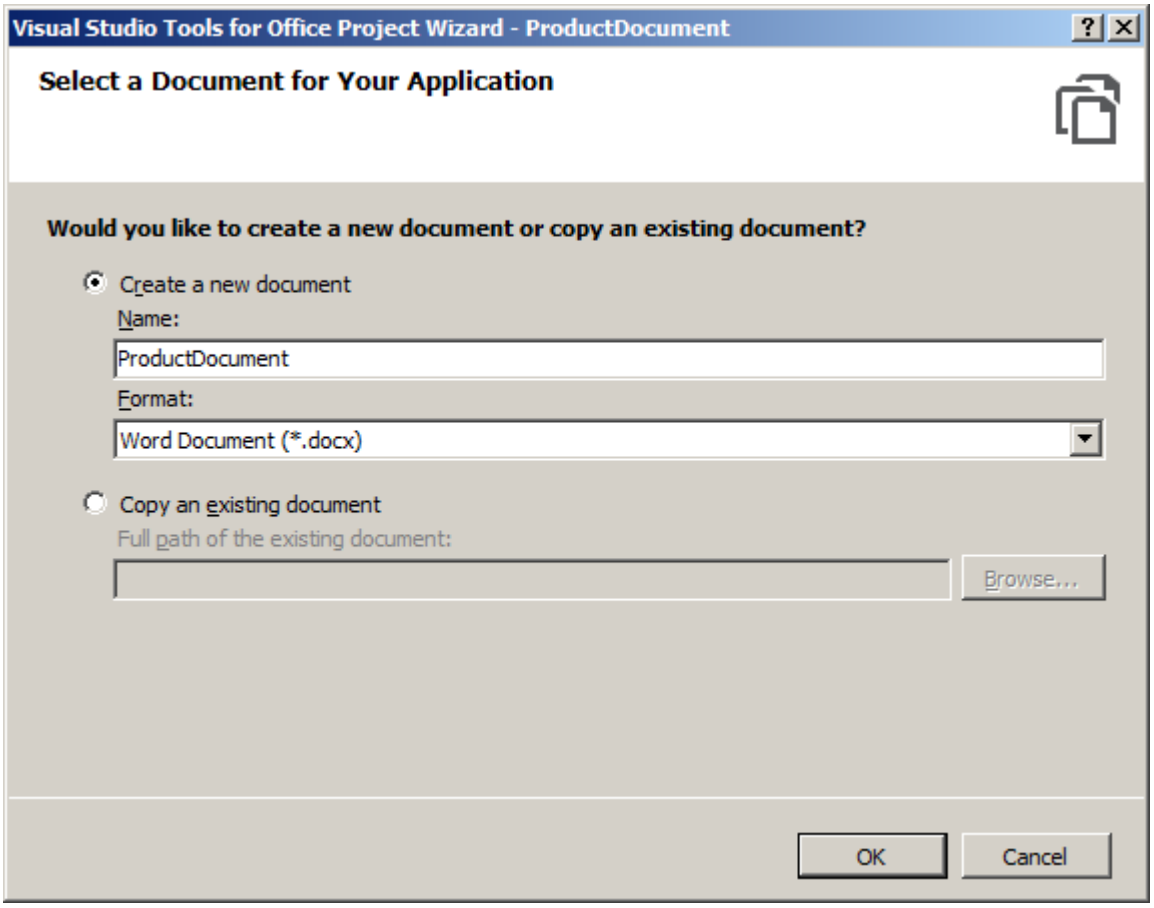
Aslında ihtiyacı olan basit bir arayüz ile stoğunu takip etmektir. **Rukiye**, **Selim Usta**' nın torunudur ve **Üniversite** son sınıftadır. **Matematik Mühendisliği** okumaktadır. Dedesinin işlerini kolaylaştırmak için bir **Web** arayüzü hazırlamıştır. **Asp.Net MVC** kullanmıştır. İyi renkler seçmiş, adımları sadeleştirmiş ve **Selim Usta** için kullanıcı deneyimi*(User Experience)* epeyce yüksek bir arabirim sunmuştur. Herşey yolunda gibidir. Lakin **Selim Usta**' nın vazgeçemediği alışkanlıkları vardır Bir gün torununa şöyle der; **"Kızım keşke şu yeni gelen tamponları bu öğrendiğim Word ile de kayıt edebilsem. Bu internet üzerinden girmek zor geliyor bana…"**
İşte şimdi bizim devreye girme sıramız. Bu yazımızda gerçekleştireceğimiz iş, **bir Word dokümanının içeriğini basit bir şekilde bir veritabanına eklemek** olacak. Bunun için **Word**üzerinde bazı **Windows Form** kontrollerini kullanıyor olacağız ve ayrıca kod yazarak, **Entity Framework** tabanlı giriş işlemleri gerçekleştireceğiz. Dilerseniz hiç vakit kaybetmeden yola koyulalım.

Word Document Projesini Oluşturmak

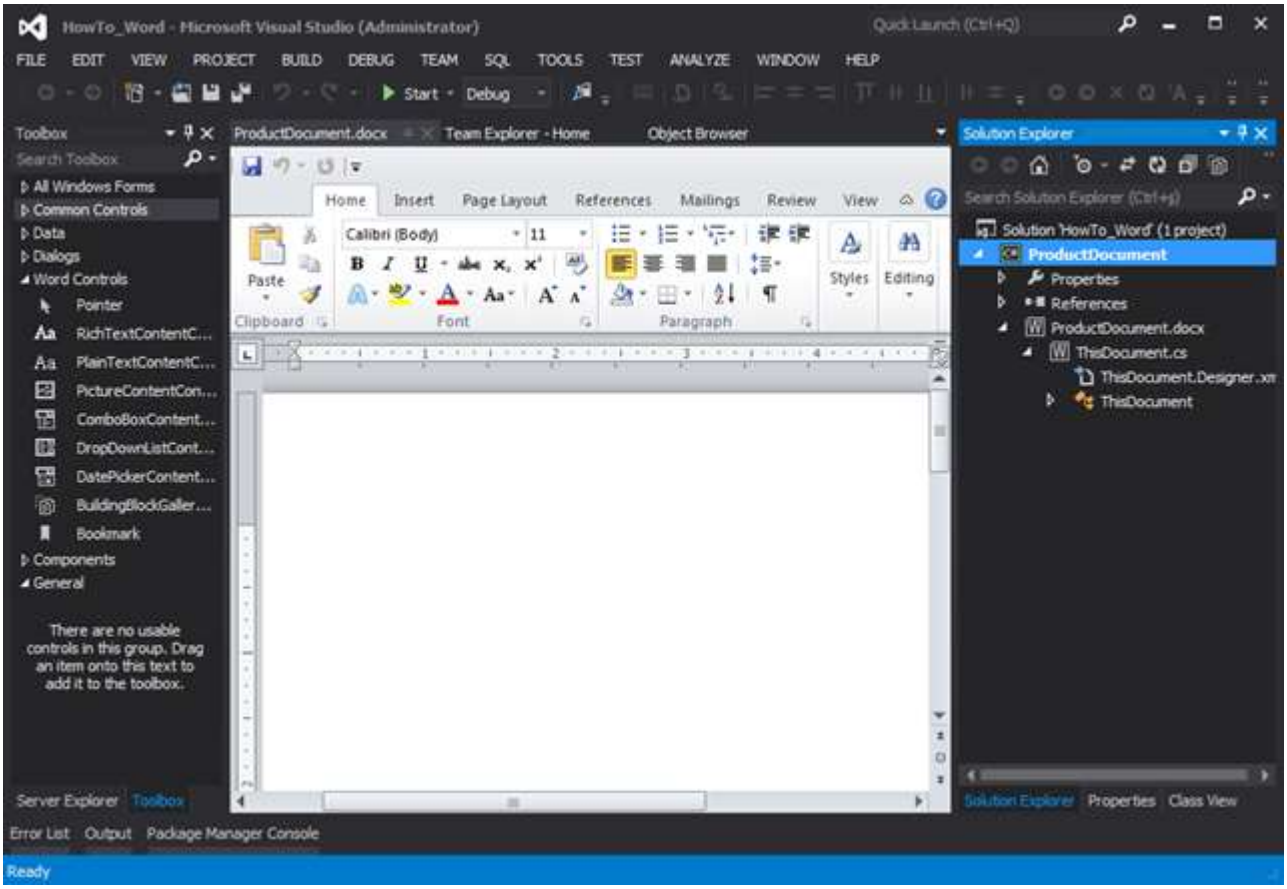
Visual Studio 2012 ortamımızda yeni bir **Solution** açarak işe başlayabiliriz. İlk projemiz **Visual C#->Office/SharePoint/Office Add-ins** sekmesinde yer alan **Word 2010 Document** tipinden olacak.



ProductDocument olarak projeyi isimlendirdikten sonra küçük bir soru ile karşılacağız.



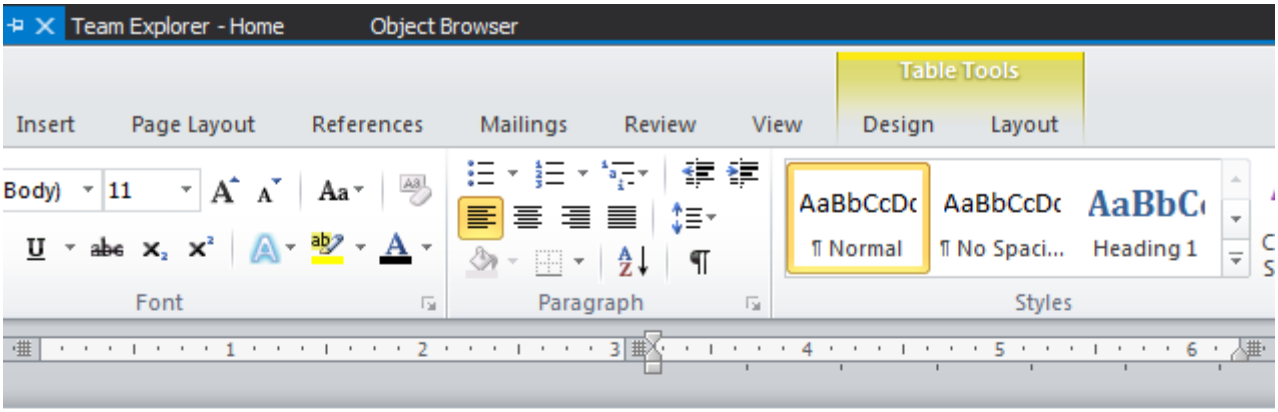
Yeni bir doküman oluşturarak ilerlemeyi tercih edelim. Ama var olan bir dokümanı da kullanabiliriz. Bu işlemlerin sonucunda **Visual Studio** ortamımız aşağıdaki şekle bürünecektir

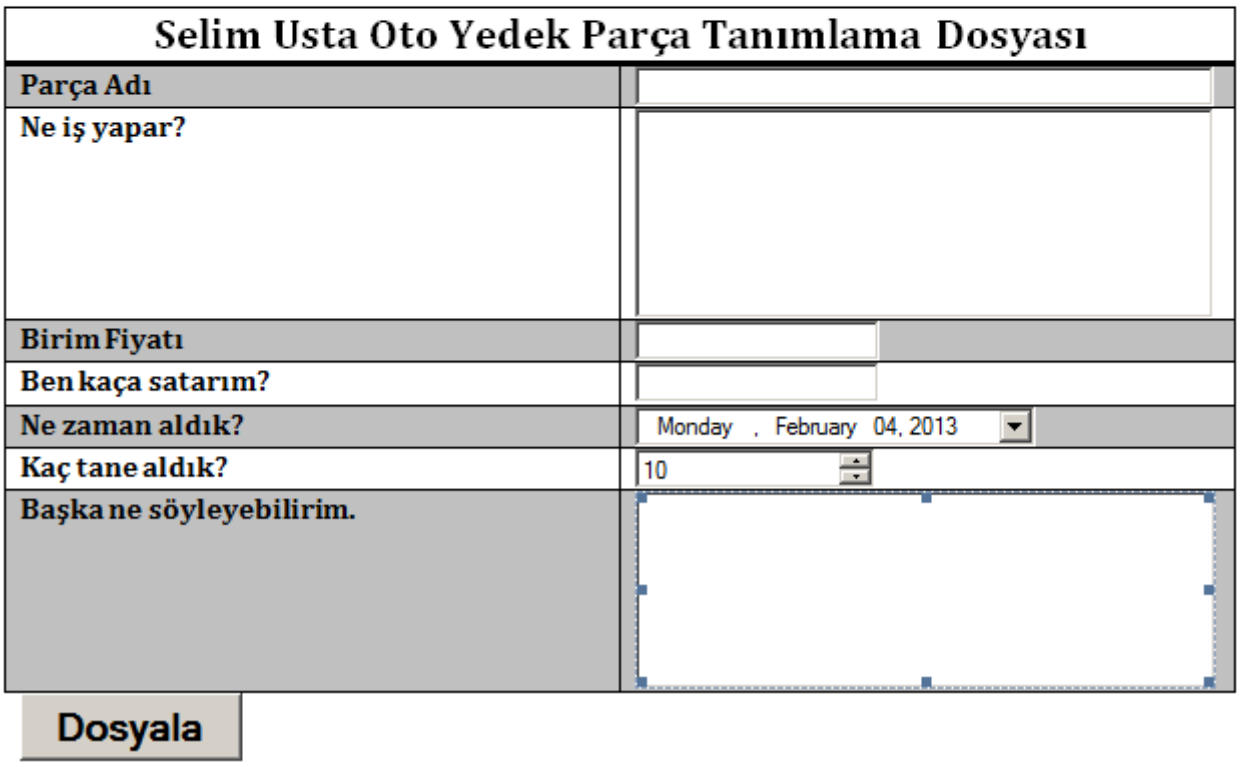


Dikkat edileceği üzere **IDE**' nin göbeğinde **docx** uzantılı bir **Word** belgesi yer almaktadır. Hatta**Solution Explorer** penceresine baktığımızda, **ThisDocument.cs** isimli bir sınıf dosyası olduğunu,**Toolbox** içerisinde ise pek çok **Form** kontrolünün yer aldığını görürüz Bu, kısaca şu anlama gelmektedir; **Word Object Model**' in üzerine **Windows Form** kontrollerini ekleyebilir ve **C#** ile kodlama yapabiliriz. Öyleyse **Selim Usta** için basit bir içerik hazırlayarak ilerleyelim*(Word Object Model ile ilişkili olarak bu adresten daha detaylı bilgi tedarik edebilirsiniz)*

Word Tasarımını Yapalım

Word belgesini **Ribon** tarafında yer alan bileşenler ile donatabileceğimiz gibi **Toolbox** üzerinde yer alan **Component**' leri içermesini de sağlayabiliriz. Bu anlamda melez bir arayüz geliştirme ortamı oluştuğunu ifade edebiliriz. Hem **Word** hem de **Windows Forms** kontrollerini bir arada ele alabilmekteyiz. Bu düşünceler ışığında **Selim Usta** için aşağıdaki formu oluşturduğumuzu düşünelim.

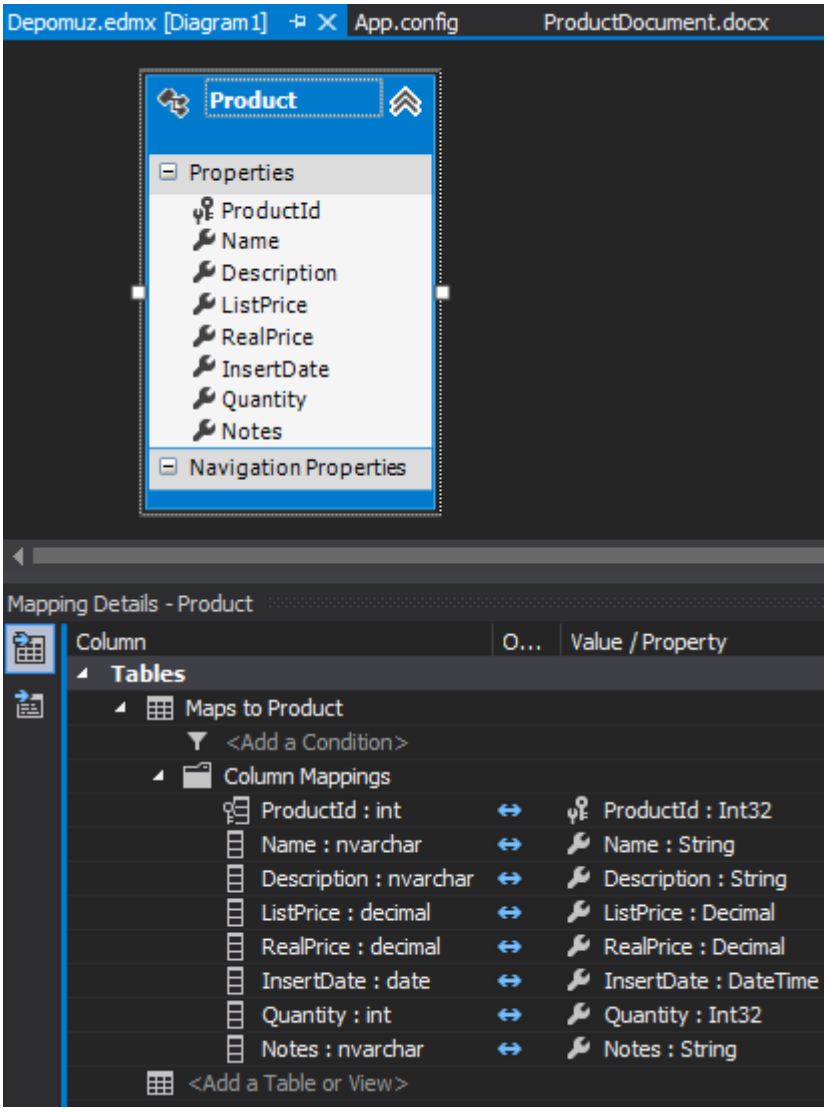




Dikkat edileceği üzere **Word** içeriğine **TextBox, DateTimePicker, NumericUpDown, Button**kontrolleri eklenmiştir. Kontrollere kod tarafı için anlamlı isimler vererek ilerleyelim. Ben örnekte**[KontrolTipi][TabloKolonAdı]** şeklinde bir notasyon kullandım. Yani **Name** alanı için**TextBoxName**, açıklama alanı için **TextBoxDescription** vb…

Entity Framework Çözümünün Eklenmesi

Solution içerisinde **Entity Framework** tabanlı bir **Class Library** projesi de yer almaktadır. Söz konusu proje bir **Ado.Net Entity Model** öğesi bulundurmakta olup aşağıdaki diagramda görülen içeriğe sahiptir.



Product Entity tipi, **Depomuz** isimli **SQL 2008** veritabanında yer alan **Product** tablosunu işaret etmektedir. Otomatik artan ve **Primary Key** olan **ProductId** alanı dışında, **Name**(nvarchar(50)),**Description**(nvarchar(250)), **ListPrice**(decimal(18,0)), **Real Price**(decimal(18,0)), **InsertDate**(Date),**Quantity**(int) ve **Notes**(nvarchar(250)) gibi kolonları da içermektedir.

Pek tabi bir **Word** dokümanından **C#** kodlarını kullanarak farklı bir kütüphaneye erişebildiğimize göre, **WCF** servislerini de çağırmamız mümkündür. Özellikle bir şirketin bu tip bir dokümanı kullanarak, çeşitli **Repository**' lere kayıt atması istenen durumlarda, bir **Servis** kanalına uğranmasını sağlamak yerinde bir davranış olabilir. *(Dolayısıyla senaryonuzu, **Entity Framework**kütüphanesi ile **Word** projesi arasına bir **WCF Servisi** sokarak genişletebilirsiniz)*

Kodlar

Tabiki ilk etapta **Word** projesinin, **Entity Library** kütüphanesini referans etmesi gerekmektedir.

Burada dikkat edilmesi gereken noktalardan birisi de Target Framework seçimidir. Oluşturduğumuz Word Document projesi .Net Framework 4.0 odaklıdır. Bu sebeple Entity Framework tabanlı olan Class Library projesinin de .Net Framework 4.0 odaklı olması gerekmektedir. Aksi durumda Target Framework uyuşmazlığı oluşuacak ve Word Document tipi projeye eklenen referans için bir Warning alınacaktır.

Kod yazımı işin eğlenceli taraflarından birisidir

elbette **Word** projesindeki **docx** dosyası, bir de**C#** kod dosyası içermektedir. Burada **Word**' ün **Object Model**' ine **this** anahtar kelimesi üzerinden kolayca erişilebilinir. Daha da önemlisi bazı olaylar*(Events)* yüklenerek dokümanının davranışları fonksiyonel olarak değiştirilebilir. Örneğin bu **Word** dokümanı kayıt edilmeden önce veya sonra devreye girecek **olay metodları*(Event Handlers)*** dahil edilebilir. Dolayısıyla yapabileceklerimiz oldukça geniş bir yelpazeye yayılmaktadır. Biz örneğimizde oldukça basit ilerledik ve aşağıdaki kod yapısını oluşturduk.

```
using AzonEntityLibrary;
using System;
using System.Windows.Forms;
using Word = Microsoft.Office.Interop.Word;
namespace ProductDocument
{
   public partial class ThisDocument
   {
      private void ThisDocument_Startup(object sender, System.EventArgs e)
      {
         this.Application.DocumentBeforeSave += Application_DocumentBeforeSave;
      }
      private void ThisDocument_Shutdown(object sender, System.EventArgs e)
      {
      }
      void Application_DocumentBeforeSave(Word.Document Doc, ref bool SaveAsUI,
ref bool Cancel)
      {
         SaveToDatabase();
      }
      private bool SaveToDatabase()
      {
         bool result = false;
         try
         {
            using (DepomuzEntities context = new DepomuzEntities())
            {
```

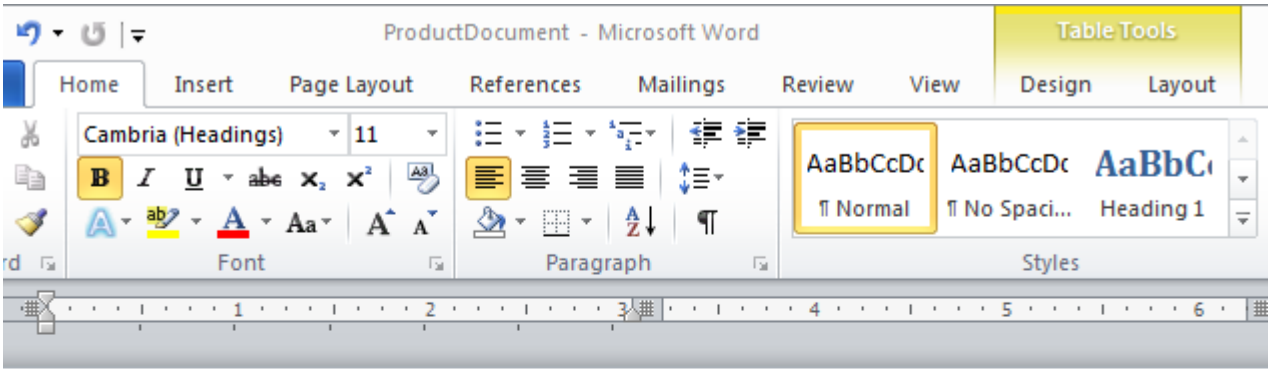
```
                Product newProduct = new Product
                {
                    Name = TextBoxName.Text,
                    Description = TextBoxDescription.Text,
                    InsertDate = DateTimePickerEnterDate.Value,
                    ListPrice = System.Convert.ToDecimal(TextBoxListPrice.Text),
                    RealPrice = System.Convert.ToDecimal(TextBoxRealPrice.Text),
                    Quantity = System.Convert.ToInt32(NumericUpDownQuantity.Value),
                    Notes = TextBoxNotes.Text
                };
                context.Products.Add(newProduct);
                context.SaveChanges();
                result = true;
                MessageBox.Show(
                    string.Format("Selim Ustam ürün {0} numarası ile dosyalandı"
                    ,newProduct.ProductId.ToString()));
            }
        }
        catch(Exception excp)
        {
            MessageBox.Show("Selim Ustam beni arar mısın? Sanırım işlemin sırasında bir
hata oluştu");
            //TODO@Rukiye burada oluşan istisnaları kendime mail atayım ve hatta log
dosyasına yazdırayım
        }
        return result;
    }
    private void ButtonAdd_Click(object sender, EventArgs e)
    {
        SaveToDatabase();
    }
    #region VSTO Designer generated code
    /// <summary>
    /// Required method for Designer support - do not modify
    /// the contents of this method with the code editor.
    /// </summary>
    private void InternalStartup()
    {
      this.ButtonAdd.Click += new System.EventHandler(this.ButtonAdd_Click);
       this.Startup += new System.EventHandler(this.ThisDocument_Startup);
```

```
            this.Shutdown += new System.EventHandler(this.ThisDocument_Shutdown);
        }
        #endregion
    }
}
```

Kod dosyasında dikkat çeken noktalardan birisi default olarak üretilen olay metodlarıdır.**ThisDocument_Startup** ve **ThisDocument_Shutdown**. Bu olay metodları Word dokümanı başlatılırken ve kapatılırken devreye girmektedir.
Örnekte, **ThisDocument_Startup** fonksiyonunda,**DocumentBeforeSave** olay metodunun yüklenmesi işlemi yapılmıştır.
Yeni bir ürünün **Entity Framework** üzerinden veritabanına kayıt edilmesi işlemi iki farklı noktadan tetiklenmektedir. Birincisi, **Word** dokümanının **Save** komutu verildikten ama **Save** operasyonu tamamlanmadan öncedir. Bunun için dikkat edileceği üzere **Application** referansı üzerinden**DocumentBeforeSave** olay metodu yüklenmiştir. İkinci tetikleme noktamız ise **Word** dokümanı üstüne atılmış olan **Button** kontrolünün **Click** olay metodudur. Yani, bir **Word** dokümanının var olan **Object Model'** ine ait olay metodlarından yararlanılabileceği gibi, doküman üzerine bırakılmış**Windows Form** kontrollerinin olay metodlarından da yararlanılabilinir.
SaveToDatabase metodu içerisinde **Context** nesnesi kullanılmaktadır. **Word** dokümanı üzerindeki kontrollere ait bilgiler yeni üretilen **Product** nesne örneğinin özelliklerine set edildikten sonras ise kayıt altına alma işlemi yapılmaktadır.
Yeni **Product** örneği, **Products** özelliği üzerinden **Add**metodu ile **generic DbSet**' e ilave edildikten sonra **SaveChanges** çağrısı ile de veritabanına yazılmaktadır.

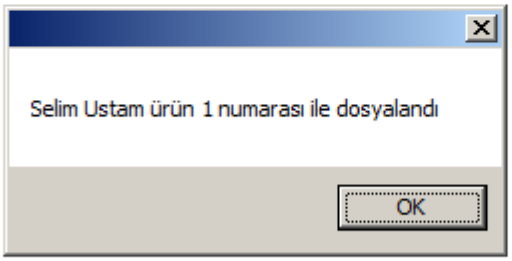
Kaba Testler

Uygulamayı **F5** ile çalıştırdığımızda ve örnek bazı veriler girdiğimizde kayıt işleminin başarılı bir şekilde gerçekleştirildiğine dair bir mesaj kutusu ile karşılaşırız. Aynen aşağıdaki ekran görüntüsündeki gibi.



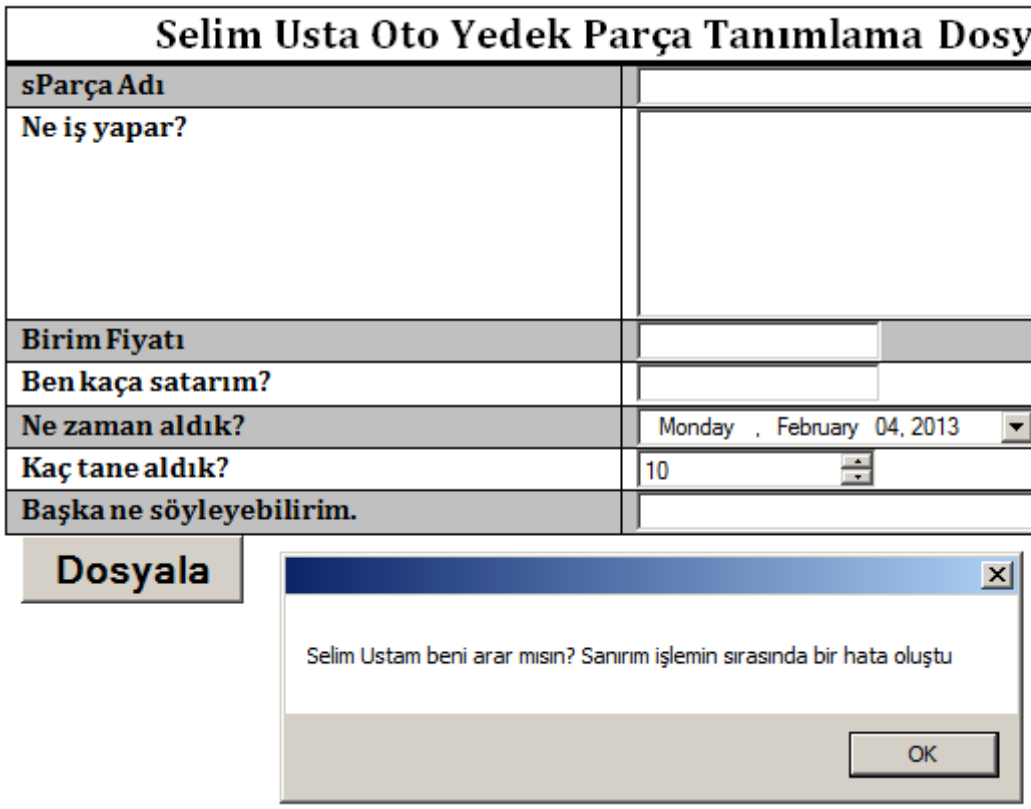
Selim Usta Oto Yedek Parça Tanımlama Dosyası	
Parça Adı	Hyundai Accent Blue Ön Tanpon
Ne iş yapar?	Arabanın ön tanponudur. Darbelere dayanıklıdır. Sağlam aluminyum alaşımlıdır.
Birim Fiyatı	250
Ben kaça satarım?	287
Ne zaman aldık?	Monday , February 04, 2013
Kaç tane aldık?	30
Başka ne söyleyebilirim.	Sadece gri, siyah, beyaz, kırmız getirttik.

Dosyala



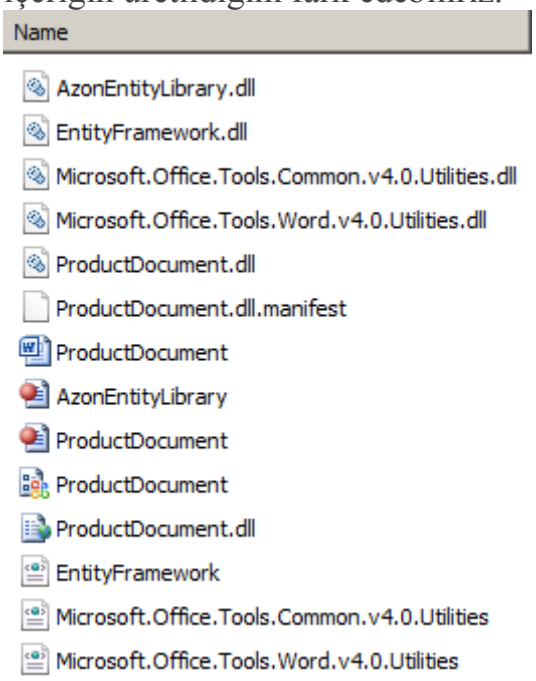
Ki eklenme işleminin başarılı olup olmadığını hemen veritabanına bakarak öğrenebiliriz.

Eğer ters bir durum oluşursa*(örneğin form boş iken Save etmeye çalışmak gibi)*, bu durumda bir çalışma zamanı hatası alınacak ama durum **Selim Usta**' ya yumuşatılarak iletilecektir 



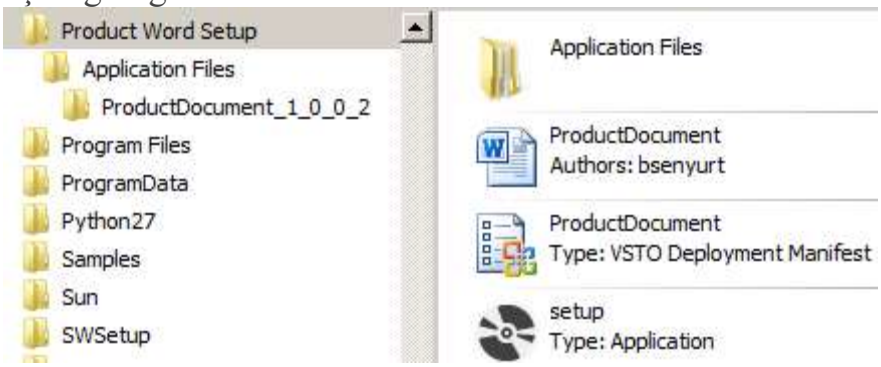
Build Sonrası

Aslında uygulamanın Build sonrası durumuna baktığımızda aşağıdaki şekilde görülen içeriğin üretildiğini fark edebiliriz.



Word Document projesi, **ProductDocument.docx** haricinde, **Solution** da kullanılan başka dosyaları da doğal olarak içerecektir. Tabi burada akla gelen ilk soru bu içeriği nasıl

dağıtılabileceğidir? **Development** yapılan makinede herhangibir sorun olmayacaktır. **Debug** veya**Release** klasöründeki **ProductDocument.dox** dosyasının çalıştırılması yeterlidir. Ancak bu çözümü **Selim Usta**' nın bilgisayarına yüklemek için en azından bir **Setup** paketine sahip olmak dağıtım işini kolaylaştıracaktır.
Bunun için **Publish** işlemi **Word Document** projesi için uygulanabilir. Burada **Click Once**teknolojisinden yararlanılabildiğini de belirtelim. Bir **Publish** paketini söz konusu uygulama için nasıl hazırlayabileceğinizi MSDN üzerindeki bu adresten öğrenebilirsiniz. Detayları ayrı bir makale konusu olacağından burada derinlemesine incelemedik.
İlgili adresteki içeriği dikkatlice okumanızı ve özellikle Post-Deployment kullanımına bakmanızı öneririm. Nitekim Post-Deployment sırasında, ilgili Word dosyasının bir kopyasının istemci bilgisayarda istenen bir klasöre atanması işlemi gerçekleştirilebilmektedir. Ayrıca Publisher' ın Trusted olarak kabul edilebilmesi için gerekli Signing işlemlerini de atlamayın derim
Publish işlemi sonrası aşağıdaki ekran görüntüsündekine benzer bir içerik oluşacaktır. Setup dosyasını kullanarak tipik bir install işlemi gerçekleştirebilirsiniz. Ayrıca bu klasörde yer alan**ProductDocument** çalıştırıldığında doğrudan tasarlamış olduğumuz **Form**' un açıldığını görebiliriz.



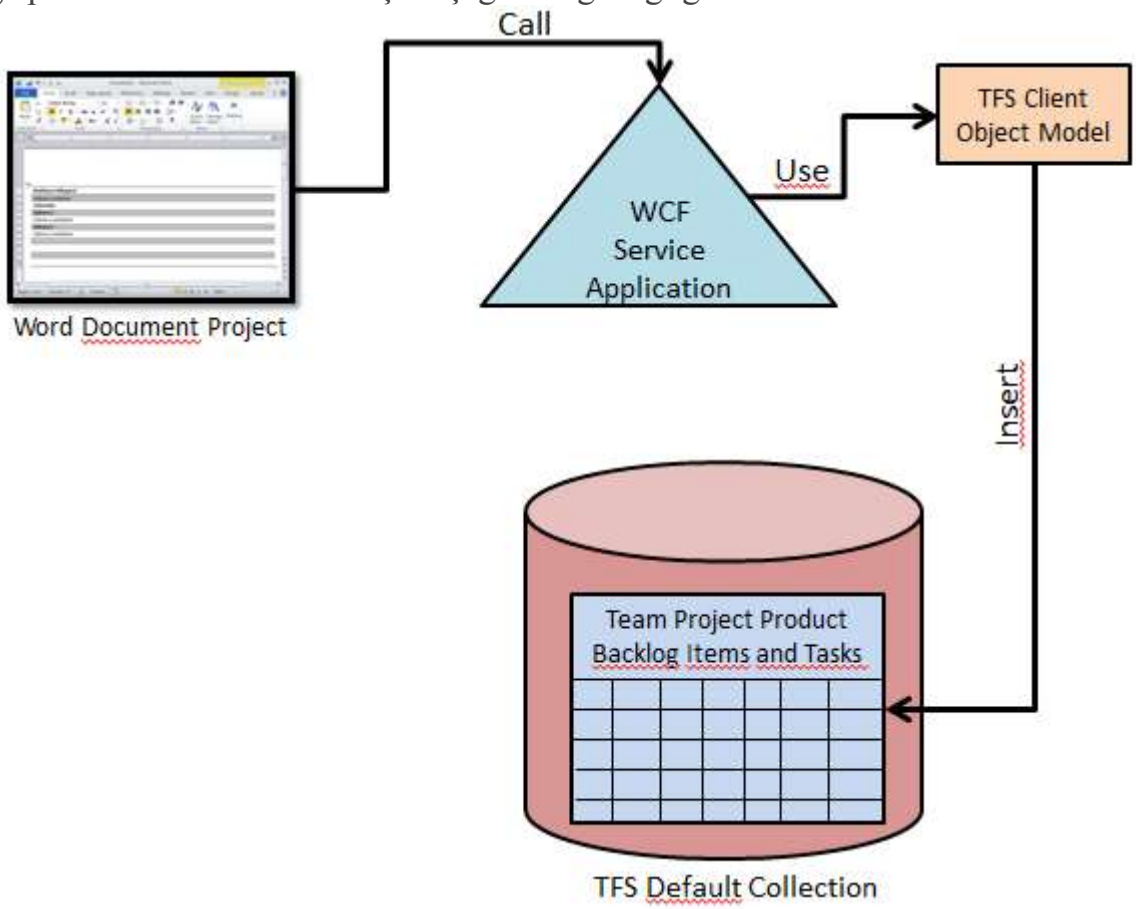
Sonuç

Geliştirdiğimiz örnek aslında **Document Level** tipinden kabul edilen bir proje uygulamasıdır. İstenirse **Application Level** şeklinde bir proje de üretilebilir ki bu durumda tüm **Word** dokümanları için uygulama seviyesinde bir geliştirme yapma şansına sahip olabiliriz. Elbette yazımıza konu olan ve **Selim Usta**' nın işine yarayacağını düşündüğümüz çözümün geliştirilmesi gereken pek çok yeri vardır. Örneğin,

- Form üzerinde bir **doğrulama*(Validation)*** mekanizması mevcut değildir. Çok doğal olarak Selim Usta sayısal olması gereken alanlara metinsel içerik girebilir.
- **Word** dosyası üzerinden yeni bir ürün eklendiğinde belki form temizlenebilir ve hatta belki de alt sayfalarda yer alan bir **DataGridView** kontrolü içerisine, yapılan ekler basılarak özet bir durum raporu sunulabilir.
- **Selim Usta** dalgınlıkla yanlış bir içerik girebilir. Bu durumda ilgili kayıtları silmek isteyecektir. Bu vakanın da doküman yoluyla karşılanması gerekebilir. Ya da mevzunun Rukiye' ye iletilmesi yolu tercih edilebilir

- Bir **WCF** servis katmanının olması da çok daha iyi olabilir. Nitekim bu sayede **Word Document** projesini kullanan herhangibir istemci bilgisayarının sadece servise ulaşması yeterli olacaktır. Ki servis de Host edildiği makinede **Entity Framework'** ü kullanarak bir **Repository'** ye yazma işlemini icra edebilir ve bu sayede **Word** dosyasının olduğu bilgisayar için **Loosely Coupled** bir geçerlilik de sağlanmış olur.

Bu ve benzeri eksiklikleri siz değerli okurlarıma bırakıyorum. Diğer yandan senaryoyu biraz daha farklılaştırabilirsiniz. Söz gelimi **TFS Client Object Model**' i de işin içerisine katabilir ve bir **WordTemplate**' i üzerinden **Product Backlog Item** ve **Task** girişlerini yaptırabilirsiniz. Bu ödev için aşağıdaki grafiği göz önüne alabilirsiniz.



Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim

HowTo_Word.zip (1,80 mb)

[Dosya boyutunun küçülmesi için Packages ve Release klasörleri silinmiştir]

[Orjinal Yazım Tarihi 03-15-2013]

Tek Fotoluk İpucu 103–Database.Query ve dynamic Avantajı

Cuma, 5 Temmuz 2013 18:31 by bsenyurt

Merhaba Arkadaşlar,
SQL gibi bir veri kaynağına erişmek için pek çok yol olduğunu gayet iyi biliyoruz. Hatta bu işi öğrenmeye başladığımız ilk zamanları hatırlayın. **Connection**' ın açılması, **Command** hazırlanması, bir **DataAdapter**' dan yararlanılarak **DataTable/DataSet** doldurulması ve DataReader ile veri setinin ileri yönlü dolaşılması ve benzeri tiplerle uğraşırız. Hatta **Entity Framework** gibi alt yapılar da kendi içlerinde bu temel türlerin ata tiplerinden fazlasıyla yararlanmaktadır.
Son zamanlarda popüler olan*(en azından benim dikkatimi yeni çeken)* kütüphanelerden birisi de**WebMatrix.Data** dır*(Oldukça eğlenceli olduğunu da ifade etmek isterim)* Bu kütüphane içerisinde yer alan **Database** tipi ise tam bir sihirbaz. Özellikle **dynamic** desteği sağlayan metodları.
Söz gelimi **Northwind** veritabanındaki ürünlerin kategori bazlı sayılarını öğrenmek istediniz. Aşağıdaki gibi bir kod parçası pekala işinize yarar.

```
using System;
using WebMatrix.Data;

namespace UsingWebMatirxDatabaseClass
{
    0 references
    class Program
    {
        0 references
        static void Main(string[] args)
        {
            Database db = Database.Open("NorthConStr");
            string commandText=@"Select Count(*) as ProductCount,C.CategoryName
                                 from Categories C
                                 join Products P on C.CategoryID=P.CategoryID
                                 group by C.CategoryName
                                 order by C.CategoryName";
            var resultSet=db.Query(commandText);

            foreach (var result in resultSet)
            {
                Console.WriteLine("{0} -> ({1} Products)"
                    , result.CategoryName
                    , result.ProductCount
                    );
            }
        }
    }
}
```

```
C:\Windows\system32\cmd.exe
Beverages -> (12 Products)
Condiments -> (12 Products)
Confections -> (13 Products)
Dairy Products -> (10 Products)
Grains/Cereals -> (7 Products)
Meat/Poultry -> (6 Products)
Produce -> (5 Products)
Seafood -> (12 Products)
Press any key to continue . . .
```

dynamic türü nedeniyle sorgu sonucu elde edilen liste elemanları üzerinden doğrudan**CategoryName** ve **ProductCount** alanlarına gidilmesi mümkün olmuştur 😉
Bir başka ip ucunda görüşmek dileğiyle.

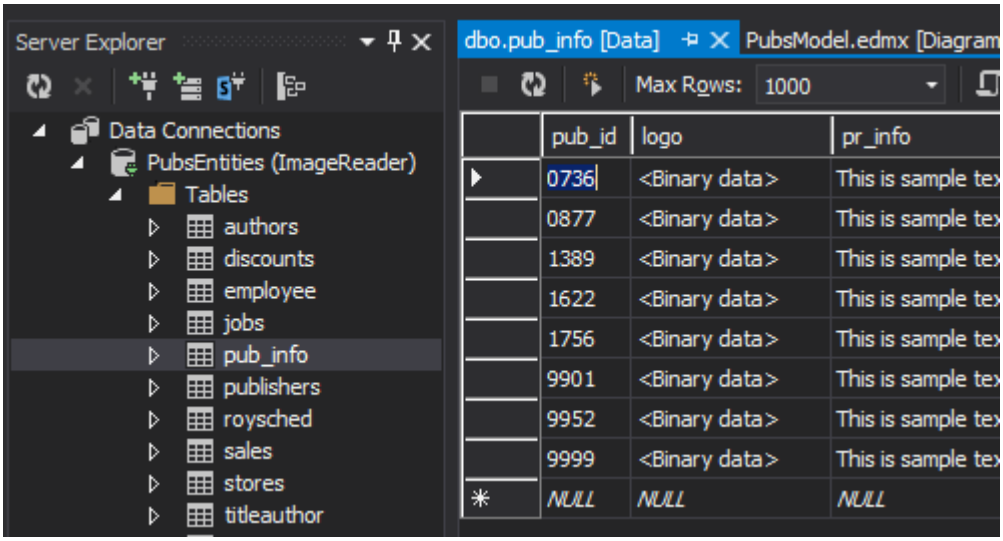
Asp.Net Web API Üzerinden Resim Döndürmek

Çarşamba, 3 Temmuz 2013 19:08 by bsenyurt

Merhaba Arkadaşlar,
Eminim çocukken çizgi filmlerle aranız vardı. Hatta çoğumuz yaşı kaç olursa olsun çizgi filmlere arada sırada da olsa zaman ayırmakta. *(Ben Batman gördüm mü pür dikkat izlerim örneğin)* Keza pek çok büyüğümüz de, eskiden izlediği çizgi filmler ile karşılaştığında taaaa çocukluk yıllarına kadar gidip aynı o zamanki gibi içten gülebiliyorlar da*(Rahmetli babamdan bilirim)*
Aslına bakarsanız bazen teknoloji de bizi aynen bu mantıkta epeyce güldürebiliyor. Örneğin **Microsoft**' un ürünlerini düşünelim. *(Gerçi çok fazlalar ama gene de düşünmeye çalışalım)* Sürekli yenilikler çıkartıyorlar, sürekli verisyon atlatıyorlar ve işin en acı tarafı da koşan Road Runner' a benziyorlar. Biz mi? Biz ise **Road Runner**' ı her fırsatta yakalamaya çalışıp yakaladığını zanneden ama son anda hep elinden kaçıran Coyote' ye Bence bu senaryoda developer' lar biraz daha şanslı. Ya benim gibi düzenli blog tutmaya çalışanlar napsınlar
Sözü fazla uzatmadan ve moralimizi daha da bozmadan konumuza geçelim.
Bu yazımızda **Asp.Net Web API** üzerinden, **SQL** tablolarında **binary** formatta tutulabilen resim içeriklerini nasıl çekebileceğimizi basit bir örnek ile incelemeye çalışıyor olacağız. Örneğimizin özel yanlarından birisi de kısa süre önce yayınlanan **Visual Studio 2013 Preview** ile geliştirilecek olması. Önce senaryomuza bir bakalım.

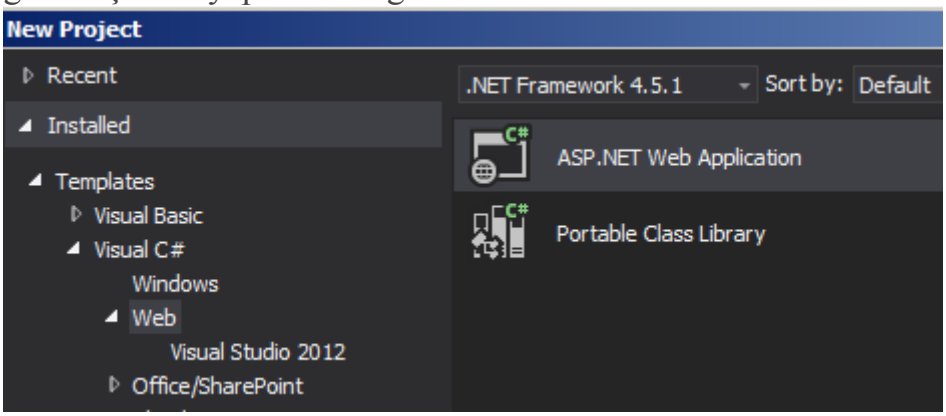
Senaryo

Uzun zamandır uğramadığımız hatta pek çok genç arkadaşımızın belki de adını bile duymadığı bir**Microsoft** veritabanını ele alıyor olacağız. **Pubs, SQL 2000** sürümünde sıklıkla **Northwind** ile birlikte andığımız kobay veritabanlarından birisidir Bu veritabanında yayıncılara ait bazı bilgiler yer almaktadır. Örneğin **pub_info** isimli tablo içerisinde **pub_id, logo** ve pr_**info** isimli 3 adet alan yer almaktadır. Bu alanlardan **logo** tahmin edileceği üzere **Binary** veri tipindedir ve yayıncının firma logosunu tutmaktadır.
Hedefimiz bu **binary** içerikleri*(yani logoları)* bir **Web API** fonksiyonu üzerinden geriye döndürebilmek ve hatta en azından tarayıcı pencresinde resim formatında gösterebilmek olacaktır. O halde projeyi açarak ilk adımımız atalım.

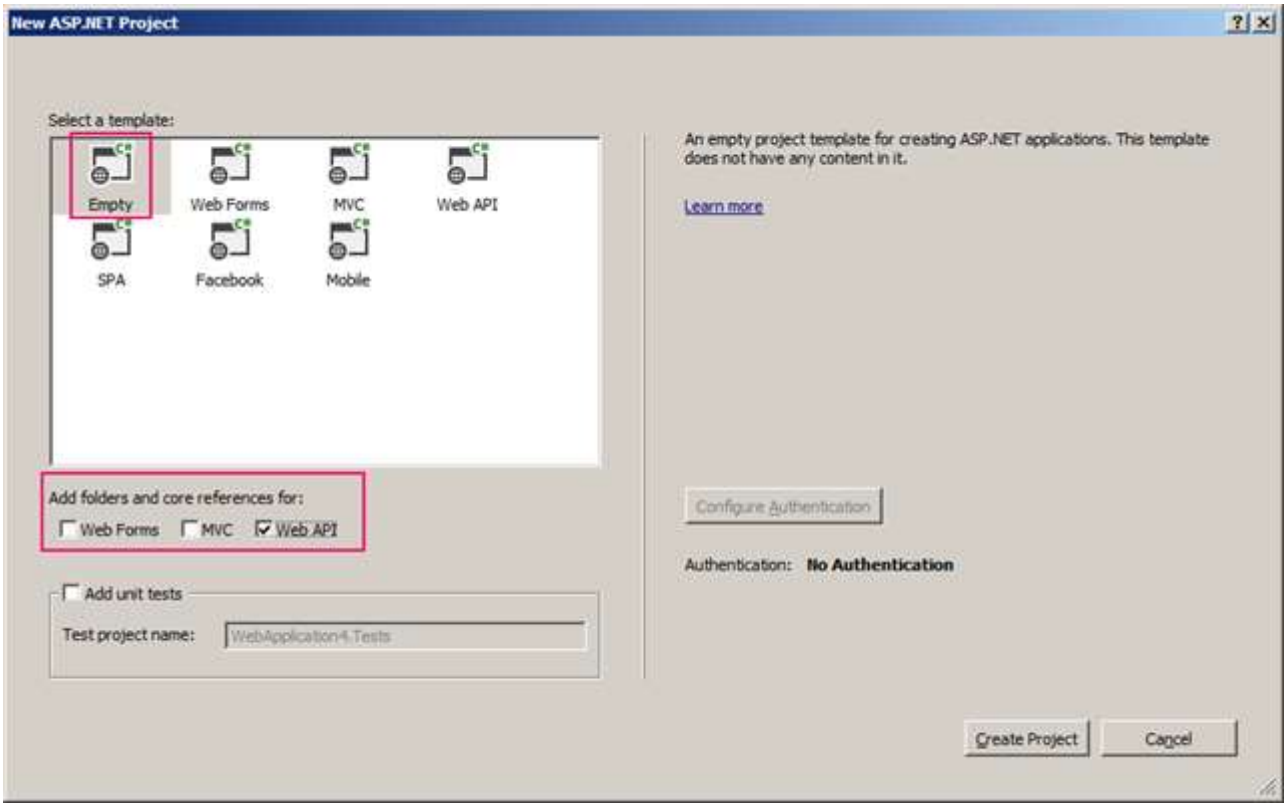


Projenin Oluşturulması

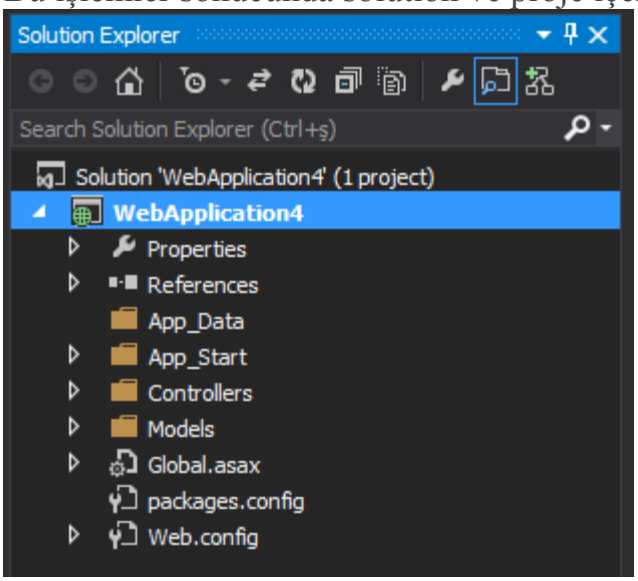
İlk olarak yeni bir **Web** uygulaması oluşturarak işe başlayabiliriz. Pek tabi **Visual Studio 2013 preview** içerisinde görünen önemli özelliklerden birisi de **One Asp.Net** yeteneğidir. Buna göre tek bir **Web** uygulaması şablonu üzerinden hareket edilerek istenen kabiliyetlere göre seçimler yapılması sağlanmaktadır.



Doğruyu söylemek gerekirse Asp.Net tarafındaki proje şablonlarının artması kafa karışıklıkları yanında bir arada kullanmak istediğimiz kabiliyetler olduğunda da sıkıntı yaratmaktaydı. Umarız bu özellik baki olur ve daha da iyileştirilir.
Asp.Net Web Application seçimi sonrasında karşımıza gelen pencereden **Empty template** tipini seçip **Web API** özelliğini etkineleştirebiliriz. Ya da **Web API** özelliğini işaretleyip ilerleyebiliriz. Ben mümkün mertebe sade bir ortam arzu ettiğimden **Empty** template seçip **Web API** kutusunu işaretledim.

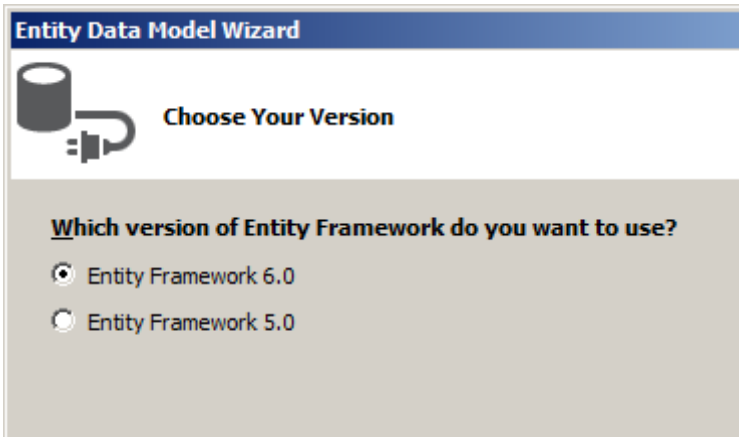


Bu işlemler sonucunda solution ve proje içeriği aşağıdaki gibi oluşacaktır.

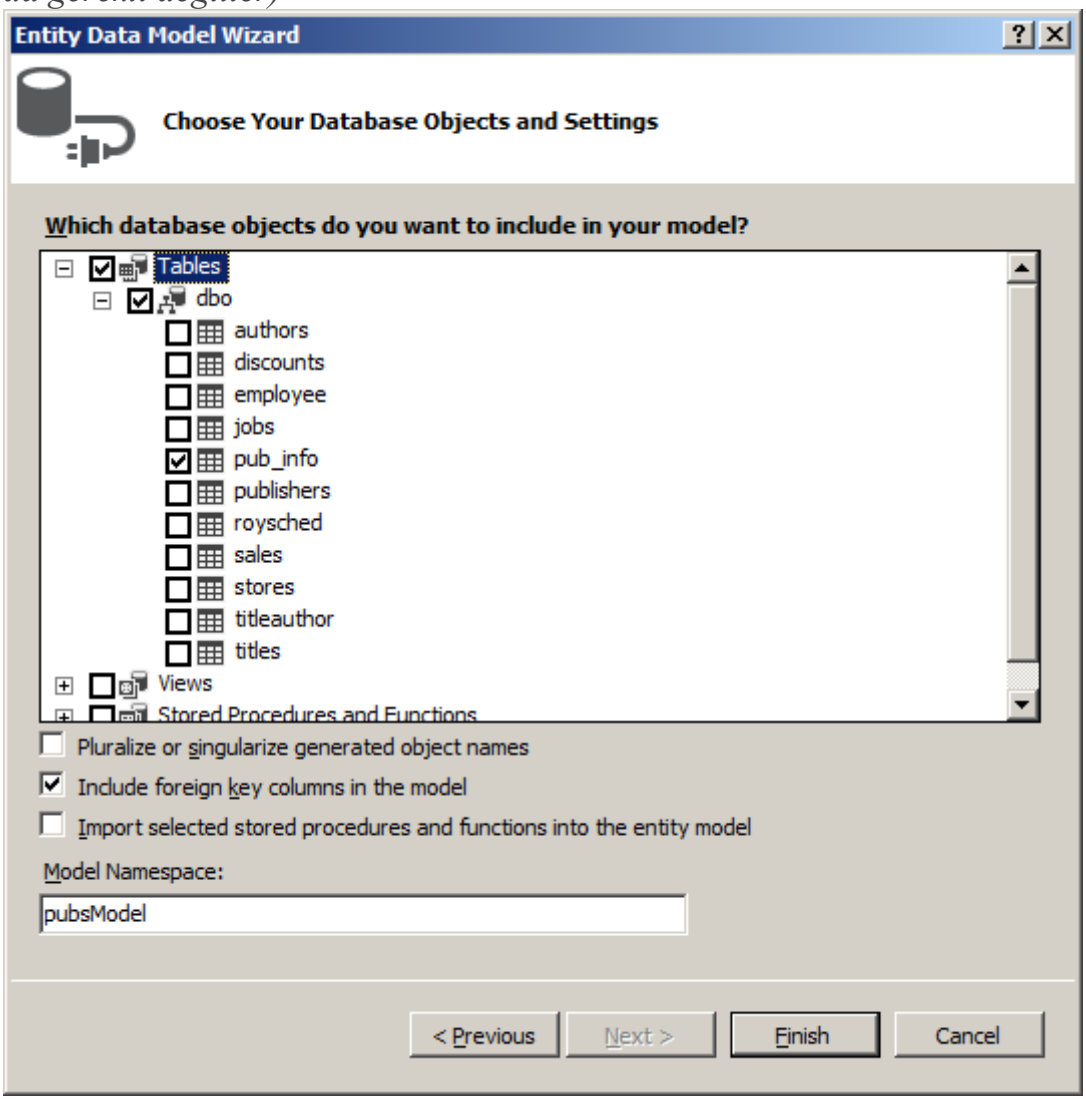


Modelin Eklenmesi

İzleyen adımda modelimizi ilave etmemiz gerekiyor. Tahmin edeceğiniz gibi **Entity Framework** den yararlanıyor olacağız. Projeye yeni bir öğe olarak **Ado.Net Entity Data Model** nesnesi ekledikten sonra klasik adımlarımızla ilerliyoruz*(Model klasörü altına ekleyebilirsiniz)* Lakin **Visual Studio 2013 Preview**' a has bir özellik olarak **Entity Framework** versiyonunu seçebileceğimiz bir ekranla karşılaşacağız*(Sanırım Entity Framework tarafı kadar hızlı versiyon atlatan ürün sayısı nadirdir)* Ben**6.0** sürümünü seçtim ve bunun sonucu olarak **Beta 1'** in kütüphane olarak ilave edildiğini fark ettim.

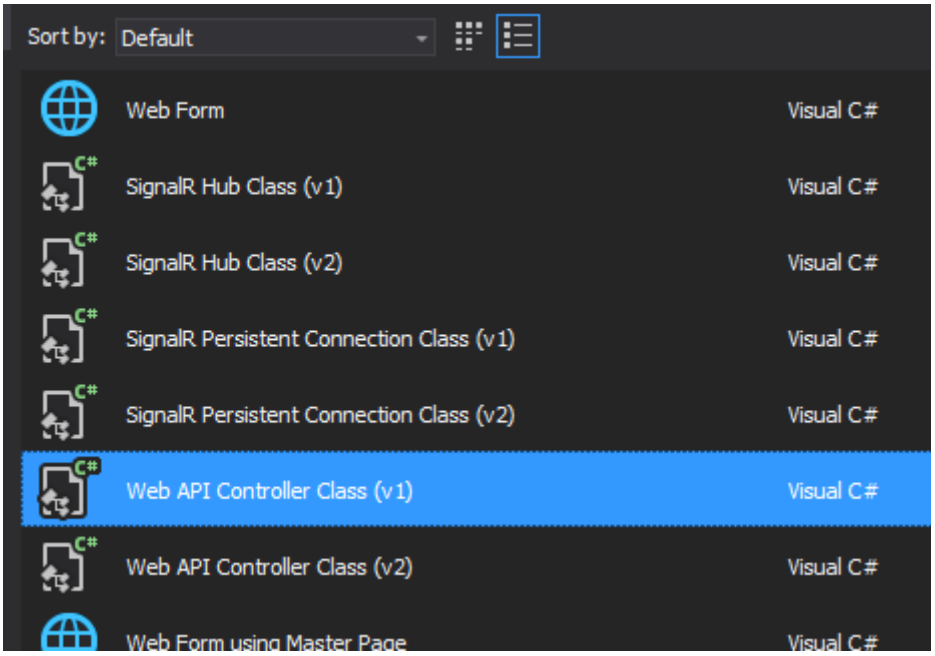


İzleyen kısımda sadece **pub_info** tablosunun eşleniği olan **entity** üretimini yaptırmamız yeterlidir.*(Diğer tablolarıı dilersenize ekleyebilirsiniz ancak şu anki senaryomuz için çok da gerekli değiller)*



Controller Tipinin Yazılması

Web API' nin temel yapı taşı olan **Controller** tipini ekleyerek örneğimize devam edelim.



Web API controller sınıfı için iki farklı versiyon bulunmaktadır. *(Ben **v1**'i seçerek ilerledim ama bunu yaparken iki versiyon arasındaki farkı tam olarak bilmediğimi itiraf etmek isterim)*

LogosController sınıfı içeriği

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Net.Http.Headers;
using System.Web.Http;
using WebApplication4.Models;
namespace WebApplication4.Controllers
{
   public class LogosController
      : ApiController
   {
      public List<string> Get()
      {
         List<string> pubIds = null;
         using (PubsEntities _context = new PubsEntities())
         {
            pubIds = (from p in _context.pub_info
                      select p.pub_id).ToList();
         }
         return pubIds;
      }
```

```
        public HttpResponseMessage Get(string id)
        {
            HttpResponseMessage response = null;
            using (PubsEntities _context = new PubsEntities())
            {
                var pubPicture = (from p in _context.pub_info
                                  where p.pub_id == id
                                  select p.logo).FirstOrDefault();
                if (pubPicture==null)
                {
                    response = new HttpResponseMessage(HttpStatusCode.NotFound);
                }
                else
                {
                    MemoryStream ms = new MemoryStream(pubPicture);
                    response = new HttpResponseMessage(HttpStatusCode.OK);
                    response.Content = new StreamContent(ms);
                    response.Content.Headers.ContentType = new
MediaTypeHeaderValue("image/png");
                }
            }
            return response;
        }
    }
}
```

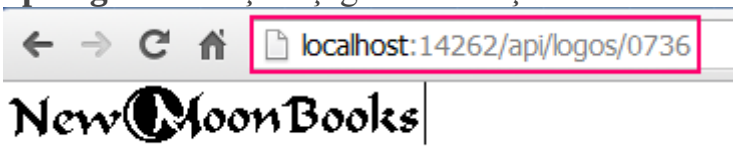
LogosController sınıfı içerisinde iki adet **Get** metodu bulunmaktadır. İstemci tarafından gelecek**HTTP Get** taleplerine cevap verecek olan bu fonksiyonlardan birisi **pub_info** tablosundaki **pub_id**alanlarını liste olarak döndürmektedir. Diğer yandan senaryomuzun can alıcı **Get** metodu ise,**HttpResponseMessage** tipinden bir nesne örneğini döndürmektedir. Bu metod parametre olarak**string** tipinden olan bir **pub_id** değerini alır. İlgili alana eş satırın **logo** içeriğini bulur*(eğer varsa)*. Bu içeriğin **byte[]** tipinden olan karşılığı bir **MemoryStream** referansından yararlanılarak**HttpResponseMessage** örneğinin **Content** özelliğine set edilir.
Bundan sonra yapılması gereken, istemciye dönecek cevap içeriğinin bir **image** olduğunu belirtmektir. **Headers.ContentType** özelliğine bir **MediaTypeHeaderValue** örneğinin atanmasının ve parametre olarak **image/png** verilmesinin sebebi de budur. Çok doğal olarak ilgili **id** değeri yanlış girilebilir ve **LINQ** sorgusu bu durumda **null** değer üretebilir. **Null** değer kontrolü yapılarak böyle bir vaka oluşması halinde **HTTP 404 Not Found** istisnasının döndürülmesi de sağlanmaktadır*(Web' in doğasına ve isteğine uygun şekilde*  *)*

Testler

Uygulama kodunun tamamlanmasını müteakip test çalışmalarına başlanabilir. Her hangi bir tarayıcı uygulama ile bu işlemi yapabiliriz*(Ben tercihimi Google Chrome' dan yana kullandım)* Örneğin**api/logos** şeklinde bir talepte bulunulduğunda aşağıdaki ekran görüntüsüne benzer olacak şekilde**pub_id** bilgilerinin elde edildiği görülür.

Eğer belirli bir **pub_id** değeri için talepte bulunulursa asıl istediğimiz sonuçlara ulaşırız. Yani yayıncının logosuna

api/logos/0736 için aşağıdaki sonuç elde edilirken



api/logos/1756 için

sonucu elde edilir. Çok doğal olarak olmayan bir pub_id için istemci tarafında HTTP 404 hatası dönecektir.

Daha Neler Yapılabilir ve Size Kalan

Senaryomuz sadece yayın evinin logosunu ve yayın evi numaralarını döndürecek fonksiyonelliklere sahip bir **Asp.Net Web API** hizmetini içermektedir. Ancak siz bu senaryoyu daha da geliştirebilirsiniz.

- Örneğin **jQuery** kullanarak yayıncıların listesinin logoları ile birlikte bir **View**' da görünmesini deneyebilirsiniz.
- Kuvvetle muhtemel yukarıdaki maddeyi bir **MVC** projesinde denersiniz. Ama aynısını **Web Forms** tabanlı bir uygulama için de yapmaya çalışabilirsiniz.

- Resimlerin gösterilmesi haricinde istemcilerin yine **Asp.Net Web API**' den yararlanarak **upload** etme işlemlerini yapabilmelerini de sağlayabilirsiniz  Bunu bir araştırmanızı öneririm. **POST** şeklinde bir talebi ele almanız gerektiğini ip ucu olarak verebilirim.
- Bir önceki maddede var olan kısmı birden fazla dosyayı bir seferde yükleme senaryosu için ele alabilirsiniz(Multiple Upload)
- Büyük boyutlu resimleri parça parça atmayı veya okumayı deneyebilirsiniz.
- ve benim aklıma gelmeyen ama sizin ele alacağınız başka bir senaryo da söz konusu olabilir.

Görüldüğü üzere bir **Asp.Net Web API** servisini resim içeriklerinin elde edilmesini konu alan bir senaryo da kullanabildik. Örneğimizi yeni göz bebeğimiz **Visual Studio 2013 Preview** üzerinde geliştirmeye çalıştık ve böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Excel ve Entity Framework Konuşuyor

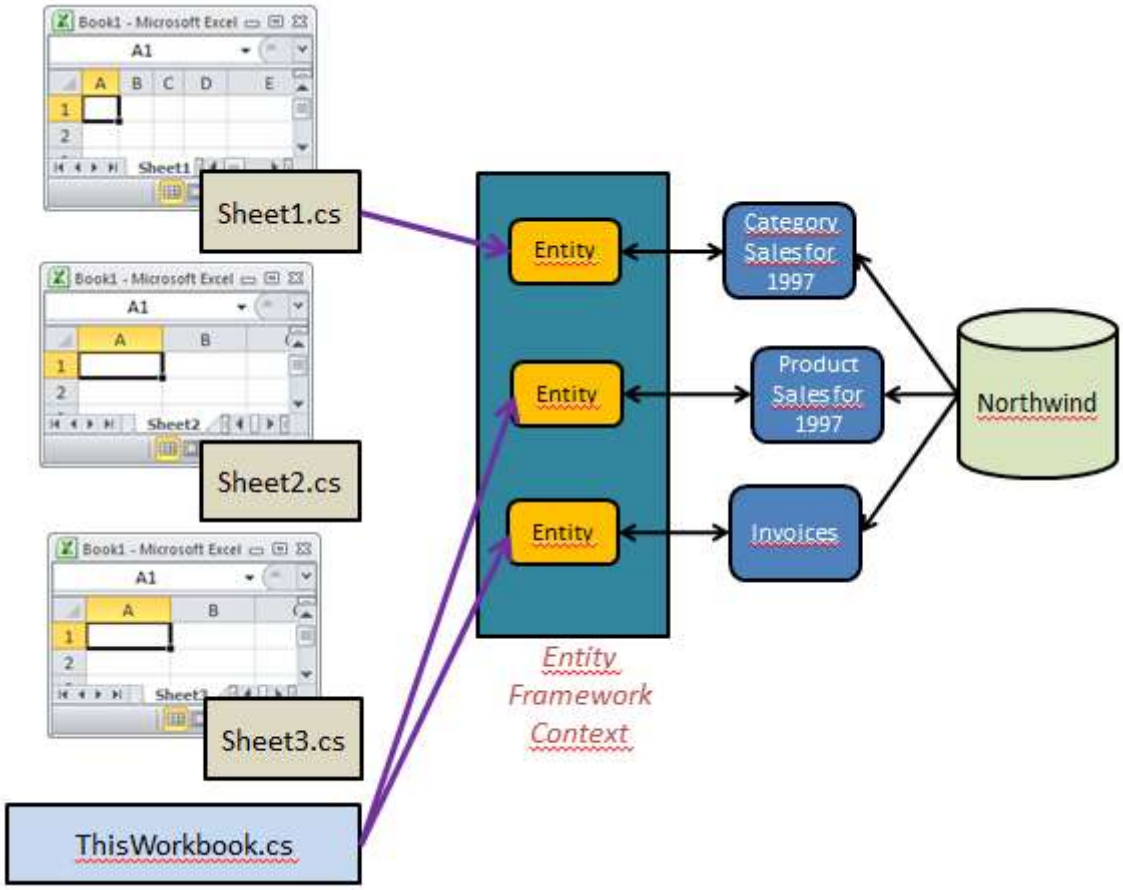
Pazartesi, 1 Temmuz 2013 14:57 by bsenyurt

[Orjinal Yazım Tarihi 03-05-2013]

Merhaba Arkadaşlar,
Artık uygulamaların birbirleri ile konuşmaları çok ama çok kolay. Bu gerçekten önemli bir mesele. Özellikle farklı segmentlerden insanların bir araya geldiği bilişim toplumlarında. Kimi kullanıcı için **Office Excel, Word**veya **Powerpoint** çok şey ifade ederken, kimi kullanıcı içinde **SQL Management Studio** ortamında hazırlanan karmaşık bir sorguya bakmak daha anlamlı olabiliyor. Ya da bir **Web** sayfası üzerinden alınan raporlar şirketin **Muhasebe Şefi** için değerli iken, kimisi **SSRS** ile elde edilen raporları mobil cihazında görmeyi tercih edebiliyor.
Ancak **Developer** gözüyle olaya bakıldığında, her segmenti memnun edecek şekilde geliştirme yapması beklendiği oldukça aşikar. Bu sebepten, farklı uygulamaların birbirleriyle rahatça konuşabilmeleri önemli bir mesele olarak karşımıza çıkıyor. **Visual Studio 2012** tarafında olaya baktığımızda bir **Office** uygulamasının, önceki sürümlere göre **.Net Framework** ile daha yüksek seviyede etkileşime girerek tasarlanabilmesi/geliştirilebilmesi de pekala mümkün.
Özellikle **Sheet**' ler veya **Workbook**' lar kodlanabilir birer **C#(Vb.Net)** dosyası olduğu için, uygulama bazında istediğimiz taklayı atma şansına sahibiz. İşte bu düşünceler ışığında yola çıktığımız ve okumakta olduğunuz yazımızda, **Excel**' i, **Entity Framework'** ü, **C#**' ı işin içerisine katacak ve birbirleri ile konuşmalarını sağlamaya çalışacağız. Haydi hiç vakit kaybetmeden yola koyulalım. Ama önce örnek senaryomuz

Senaryo

Amacımız, **Northwind** veritabanında bulunan 3 adet **View** nesne örneğinin **Excel** dokümanı içerisindeki **Sheet'** ler de gösterilmesini sağlamak. Bu amaçla **Category Sales for 1997**, **Product Sales for 1997** ve **Invoices** isimli **View** nesnelerini kullanıyor olacağız. **Sheet1** içeriğini kendi kod dosyası içerisinde üretmek istiyoruz. **Sheet2** ve **Sheet3** içeriklerini ise, **Workbook**' a ait kod dosyasından besliyor olacağız. Bu noktada ağırlıklı olarak **Workbook** ve **Sheet** sınıflarının ilgili başlangıç noktalarını değerlendireceğiz. Kabaca senaryomuzu işaret eden aşağıdaki görseli göz önüne alabiliriz.



Konuşan Çözüm

Dilerseniz senaryo içerisinde tarafların gözünden olaya bakalım.

Sheet Konuşuyor

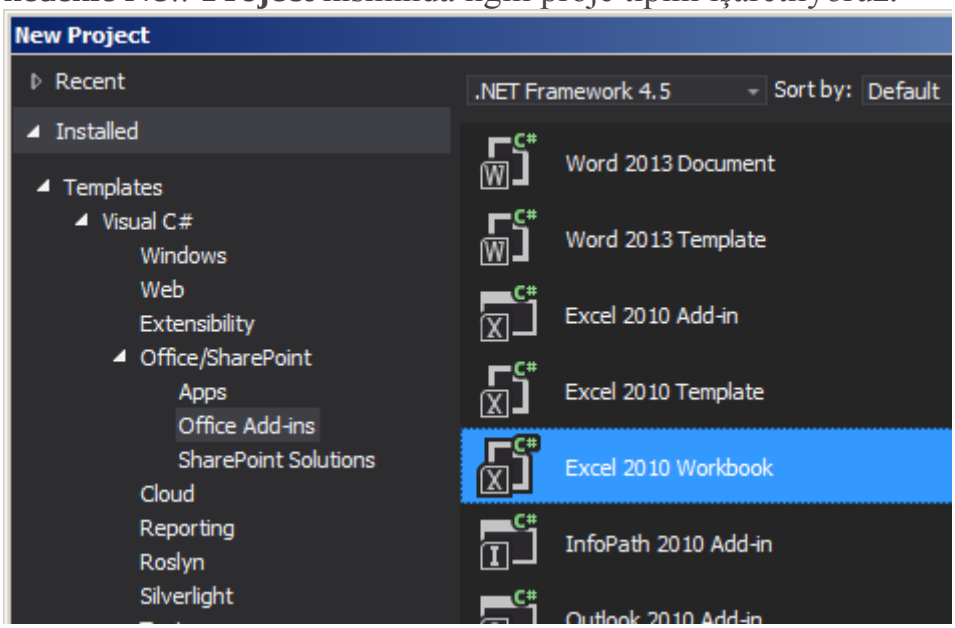
"Merhaba, benim adım **Sheet1**. Çalıştırıldığımda **Sheet1_Startup** olay metoduna bir çağrıda bulunurum. Ey Startup metodu, haydi işini yap derim. O da kendi içinde **Entity Framework** tabanlı**Context** nesnesini kullanır ve **1997** yılına ait kategori bazlı satışların verilerini, sahip olduğum hücrelere teker teker aktarır Onunla çok iyi anlaşırız.

Workbook Konuşuyor

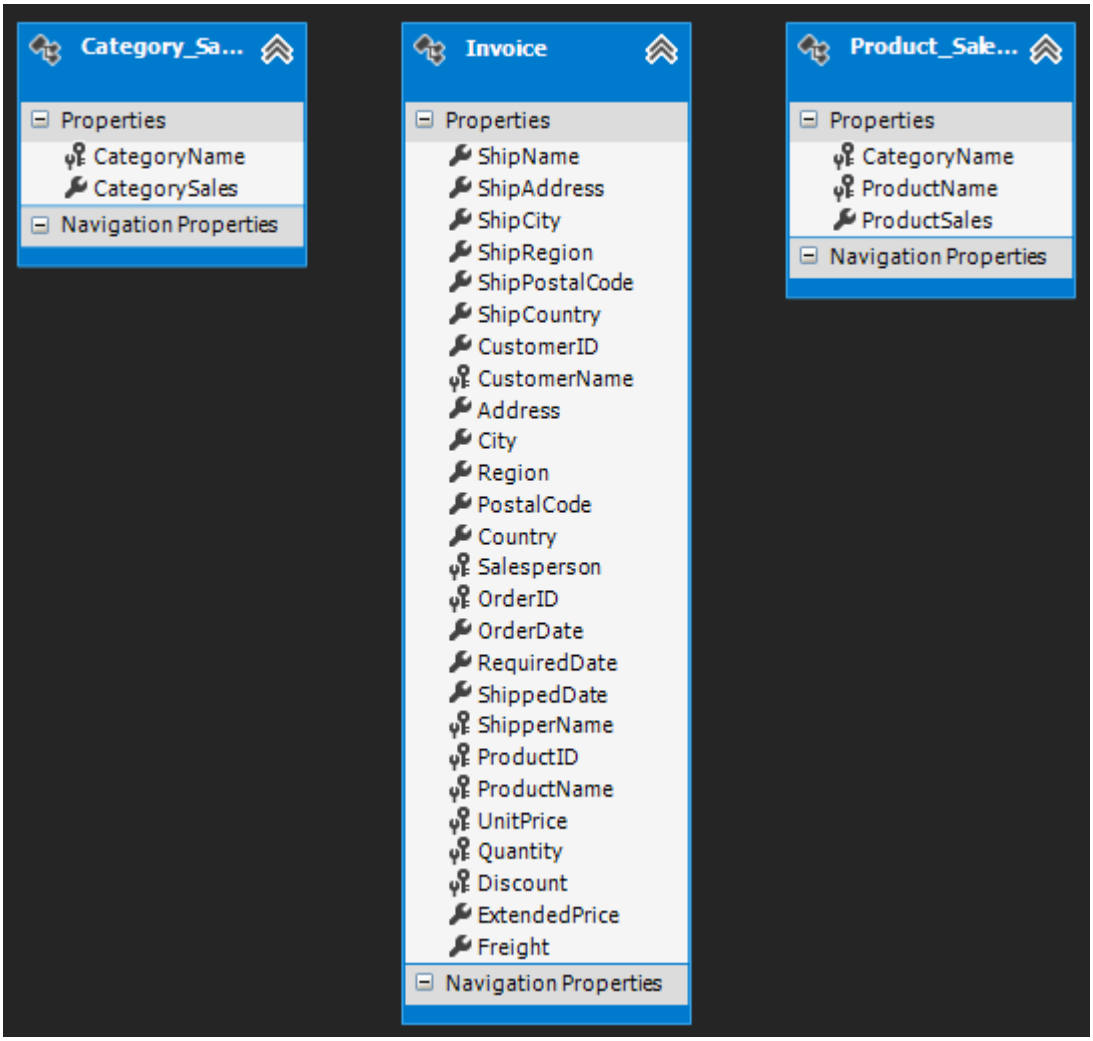
Merhaba, ben **ThisWorkbook**. Ben sahip olduğum tüm **Sheet**' leri yönetebilirim. Örneğin kendi**Startup** metoduma, tüm **Sheet**' leri çeşitli yerlerden topladığı veriler ile doldurmasını söyleyebilirim.**He-Man** ile aramdaki tek fark onun kılıcının olmasıdır.

Hazırlıklar

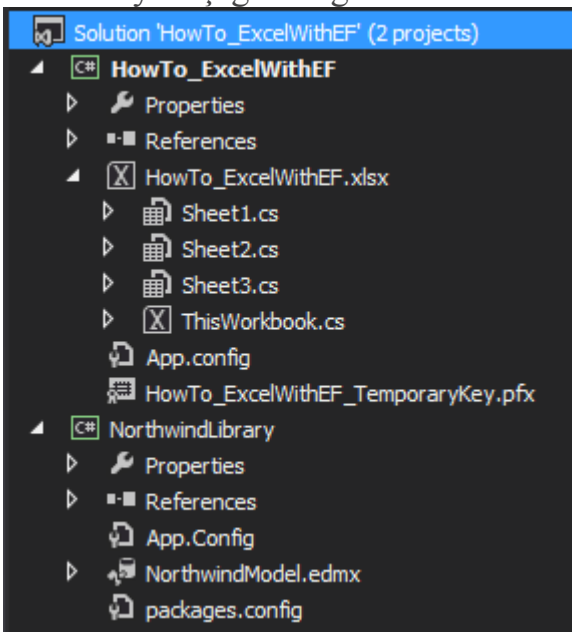
Dilerseniz senaryomuzu nihai sonuca ulaştırmak için adım adım ilerlemeye başlayalım. İlk olarak**Visual Studio 2012** ortamında bir **Excel 2010 Workbook** projesi oluşturmalıyız. Bu nedenle **New Project** kısmında ilgili proje tipini işaretliyoruz.



Bu işlemin ardından **Northwind** veritabanına ait **Entity** içeriğini barındıracak bir **Class Library**projesini aynı **Solution** içerisine ekleyebilir ve aşağıdaki gibi mevzuya konu olan **View** nesnelerini içeren **Entity Data Model** öğesini üretebiliriz.



Çok doğal olarak **Excel** uygulamasının, söz konusu **Class Library**' yi ve ayrıca **EntityFrameworkAssembly**' ını referans etmesi gerekmektedir. **Solution** içeriği şu an itibariyle aşağıdaki gibi olacaktır.



Dikkat edileceği üzere **HowTo_ExcelWithEF** uygulaması içerisinde **Workbook** ve söz konusu**Workbook'** a ait **Sheet**' ler için birer kod dosyası bulunmaktadır. Bu kod dosyaları içerisindeki pek çok olay metoduna müdahale edebilir ve çalışma zamanını kontrol altına alabiliriz ki örneğimizde de benzer bir işi gerçekleştiriyor olacağız.
Kritik noktalardan birisi de, **Entity** kütüphanesini kullanacak çalıştırılabilir uygulamanın,**Connection String** bilgisine sahip olması zorunluluğudur. Bu nedenle **NorthwindLibrary**içerisindeki **App.Config** dosyasına ait **Connection String** bilgisini, **HowTo_ExcelWithEF**uygulamasında da kullanmalıyız.

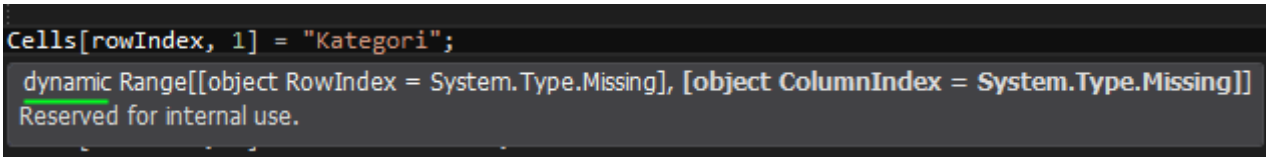
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add
name="NorthwindEntities"connectionString="metadata=res://*/NorthwindModel.csdl|res://*/NorthwindModel.ssdl|
res://*/NorthwindModel.msl;provider=System.Data.SqlClient;provider connection string=&quot;data source=.;initial catalog=Northwind;integrated security=True;MultipleActiveResultSets=True;App=EntityFramework&quot;"
providerName="System.Data.EntityClient" />
  </connectionStrings>
</configuration>
```

Artık kodlama tarafını tamamlayabiliriz. İlk olarak **Sheet1.cs** içeriğini aşağıdaki gibi geliştirdiğimizi düşünelim.

```
using NorthwindLibrary;
using System.Linq;
namespace HowTo_ExcelWithEF
{
   public partial class Sheet1
   {
      private void Sheet1_Startup(object sender, System.EventArgs e)
      {
         using(NorthwindEntities context=new NorthwindEntities())
         {
            var categoryBasedSales = from s in context.Category_Sales_for_1997
                     orderby s.CategorySales descending
                     select new {
                        s.CategoryName, s.CategorySales
                     };
            int rowIndex = 1;

            Cells[rowIndex, 1] = "Kategori";
            Cells[rowIndex, 2]= "Satışlar";
```

```
            Cells[rowIndex, 1].Font.Bold = true;
            Cells[rowIndex, 2].Font.Bold = true;
            foreach (var sale in categoryBasedSales)
            {
               rowIndex++;
               Cells[rowIndex, 1]= sale.CategoryName;
               Cells[rowIndex, 2]= sale.CategorySales;
            }
         }
      }
      private void Sheet1_Shutdown(object sender, System.EventArgs e)
      {
      }
      #region VSTO Designer generated code
      /// <summary>
      /// Required method for Designer support - do not modify
      /// the contents of this method with the code editor.
      /// </summary>
      private void InternalStartup()
      {
         this.Startup += new System.EventHandler(Sheet1_Startup);
         this.Shutdown += new System.EventHandler(Sheet1_Shutdown);
      }
      #endregion
   }
}
```

Startup metodunda **context** nesne örneği kullanılarak ilgili **View**' dan **Anonymous** tipte bir nesne listesi oluşturulmaktadır. Söz konusu liste **Sheet1** içerisindeki hücrelere yerleştirilir. Bunun için**Cells[rowIndex,columnIndex]** ifadesi ele alınmaktadır. Kodu yazarken dikkat edilecek olursa**dynamic** özelliğinin kullanılmakta olduğu görülebilir. Yani noktadan sonrasındaki atamalar ve bunlara ait tipler çalışma zamanında çözümlenecektir. Bu biraz performans kaybına yol açacak olsa da şu anki senaryodaki gibi küçük veri kümelerinde önemsizdir.



1,1 ve **1,2** hücrelerini başlıklar ile doldurduktan sonra*(Excel tarafında ilk hücre satır sütun değeri 0,0 değil 1,1 dir)* bunların **Bold** olması sağlanmıştır. Diğer yandan sonuç listesi üzerinde dönülmekte ve diğer hücreler, **View** dan gelen veriler ile beslenmektedir. Bu kodlamaya göre veri **Sheet1**' in kod dosyası da doldurulmaktadır. Pek tabi **Sheet**' lerin **Workbook**' a ait olay metodlarında şekillendirilmesi de söz

konusudur. **Sheet2** ve **Sheet3** bu amaçla **Workbook.cs** içerisinde ele alınmıştır. Aynen aşağıdaki kod parçasında görüldüğü gibi.

```
using NorthwindLibrary;
using System.Drawing;
using System.Linq;
namespace HowTo_ExcelWithEF
{
   public partial class ThisWorkbook
   {
      private void ThisWorkbook_Startup(object sender, System.EventArgs e)
      {
         using(NorthwindEntities context=new NorthwindEntities())
         {
            #region Entity View içeriklerinin çekilmesi
            var invoices = from i in context.Invoices
                           orderby i.CustomerName
                           select new
                          {
                             From = i.Country + "," + i.City,
                             i.CustomerID,
                             i.CustomerName,
                             i.OrderDate,
                             ShipTo = i.ShipCountry + "," + i.ShipCity
                          };
            var prdoductSalesFor1997 = from s in context.Product_Sales_for_1997
                                       orderby s.ProductSales descending
                                       ,s.CategoryName ascending
                                       ,s.ProductName ascending
                                       select new
                                       {
                                          Product=s.CategoryName+"-"+s.ProductName,
                                          s.ProductSales
                                       };
            #endregion
            #region Sheet2 nin Invoices içeriği ile doldurulması
            int rowIndex = 1;
            Sheets["sheet2"].Cells[rowIndex, 1] = "From";
            Sheets["sheet2"].Cells[rowIndex, 1].Font.Bold = true;
            Sheets["sheet2"].Cells[rowIndex, 2] = "Customer ID";
            Sheets["sheet2"].Cells[rowIndex, 2].Font.Bold = true;
            Sheets["sheet2"].Cells[rowIndex, 3] = "Customer";
```

```
        Sheets["sheet2"].Cells[rowIndex, 3].Font.Bold = true;
        Sheets["sheet2"].Cells[rowIndex, 4] = "Order Date";
        Sheets["sheet2"].Cells[rowIndex, 4].Font.Bold = true;
        Sheets["sheet2"].Cells[rowIndex, 5] = "Ship To";
        Sheets["sheet2"].Cells[rowIndex, 5].Font.Bold = true;
        foreach (var invoice in invoices)
        {
           rowIndex++;
           Sheets["sheet2"].Cells[rowIndex, 1] = invoice.From;
           Sheets["sheet2"].Cells[rowIndex, 2] = invoice.CustomerID;
           Sheets["sheet2"].Cells[rowIndex, 3] = invoice.CustomerName;
           Sheets["sheet2"].Cells[rowIndex, 4] = invoice.OrderDate;
           Sheets["sheet2"].Cells[rowIndex, 5] = invoice.ShipTo;
        }
        #endregion Sheet2 nin Invoices içeriği ile doldurulması
        #region Sheet3 ün 1997 yılına ait satış verileri ile doldurulması
        rowIndex = 1;
        Sheets["sheet3"].Cells[rowIndex, 1] = "Product";
        Sheets["sheet3"].Cells[rowIndex, 1].Font.Bold = true;
        Sheets["sheet3"].Cells[rowIndex, 2] = "Sales";
        Sheets["sheet3"].Cells[rowIndex, 2].Font.Bold = true;
        foreach (var sales in prdoductSalesFor1997)
        {
           rowIndex++;
           Sheets["sheet3"].Cells[rowIndex, 1] = sales.Product;
           if (sales.ProductSales > 30000)
           {
              Sheets["sheet3"].Cells[rowIndex, 2].Interior.Color = Color.Black;
              Sheets["sheet3"].Cells[rowIndex, 1].Interior.Color = Color.Black;
              Sheets["sheet3"].Cells[rowIndex, 2].Font.Color = Color.Gold;
              Sheets["sheet3"].Cells[rowIndex, 1].Font.Color = Color.Gold;
           }
           Sheets["sheet3"].Cells[rowIndex, 2] = sales.ProductSales;
        }
        #endregion Sheet3 ün 1997 yılına ait satış verileri ile doldurulması
     }
}
private void ThisWorkbook_Shutdown(object sender, System.EventArgs e)
{
}
#region VSTO Designer generated code
```

```
        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InternalStartup()
        {
            this.Startup += new System.EventHandler(ThisWorkbook_Startup);
            this.Shutdown += new System.EventHandler(ThisWorkbook_Shutdown);
        }
        #endregion
    }
}
```

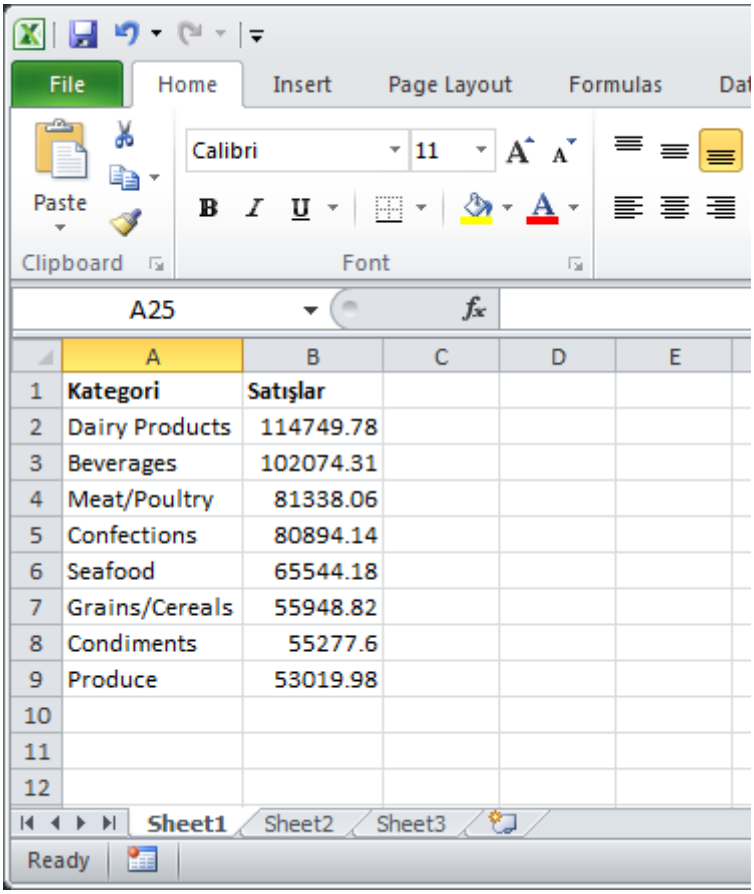
Office uygulamalarını geliştirirken C# 4.0 ile birlikte gelen dynamic, Optional and Named Parameters gibi yeniliklerin işlerimizi nasıl kolaylaştırdığına bir kere daha şahit oluyoruz.

Kodlarda bizi zorlayacak bir karmaşa yoktur. İki **View** sırasıyla sorgulanmakta ve elde edilen listeler ilgili **Sheet**' ler deki hücrelere satır satır doldurulmaktadır. Özellikle **1997 yılına ait satış rakamları** alınırken **30000** birimden büyük olunması halinde, o satırın arka plan ve font renkleri değiştirilerek göze batmaları sağlanmaktadır Dikkate değer noktalar, ilgili hücrelerin nasıl formatlandığı veya belirli bir **Sheet**' e nasıl ulaşıldığı ile alakalıdır.

Sonuçlar

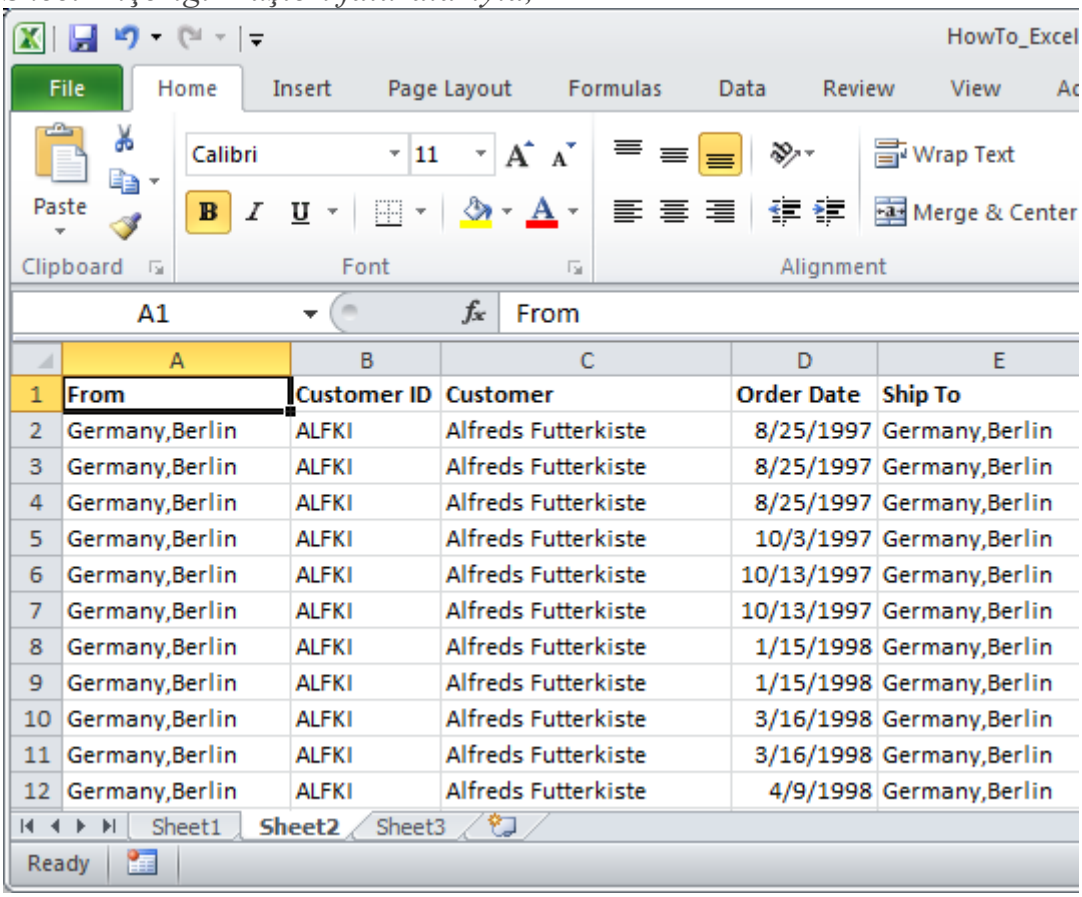
Uygulamayı çalıştırdığımızda çalışma zamanında bir **Excel Workbook**' un açıldığını ve **Sheet1, Sheet2, Sheet3** sayfalarının ilgili **View** içerikleri ile doldurulduğunu görebiliriz.

Sheet1 içeriği kategori bazlı satış verilerl ile



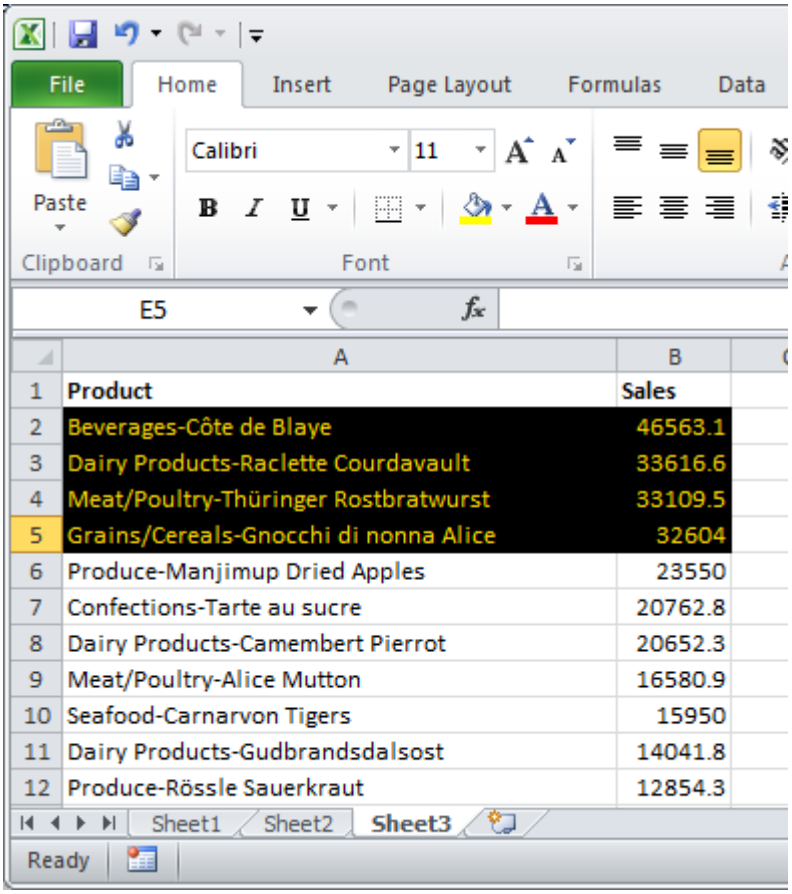
	A	B	C	D	E
1	**Kategori**	**Satışlar**			
2	Dairy Products	114749.78			
3	Beverages	102074.31			
4	Meat/Poultry	81338.06			
5	Confections	80894.14			
6	Seafood	65544.18			
7	Grains/Cereals	55948.82			
8	Condiments	55277.6			
9	Produce	53019.98			
10					
11					
12					

Sheet 2 içeriği Müşteri faturalarıyla,



	A	B	C	D	E
1	**From**	**Customer ID**	**Customer**	**Order Date**	**Ship To**
2	Germany,Berlin	ALFKI	Alfreds Futterkiste	8/25/1997	Germany,Berlin
3	Germany,Berlin	ALFKI	Alfreds Futterkiste	8/25/1997	Germany,Berlin
4	Germany,Berlin	ALFKI	Alfreds Futterkiste	8/25/1997	Germany,Berlin
5	Germany,Berlin	ALFKI	Alfreds Futterkiste	10/3/1997	Germany,Berlin
6	Germany,Berlin	ALFKI	Alfreds Futterkiste	10/13/1997	Germany,Berlin
7	Germany,Berlin	ALFKI	Alfreds Futterkiste	10/13/1997	Germany,Berlin
8	Germany,Berlin	ALFKI	Alfreds Futterkiste	1/15/1998	Germany,Berlin
9	Germany,Berlin	ALFKI	Alfreds Futterkiste	1/15/1998	Germany,Berlin
10	Germany,Berlin	ALFKI	Alfreds Futterkiste	3/16/1998	Germany,Berlin
11	Germany,Berlin	ALFKI	Alfreds Futterkiste	3/16/1998	Germany,Berlin
12	Germany,Berlin	ALFKI	Alfreds Futterkiste	4/9/1998	Germany,Berlin

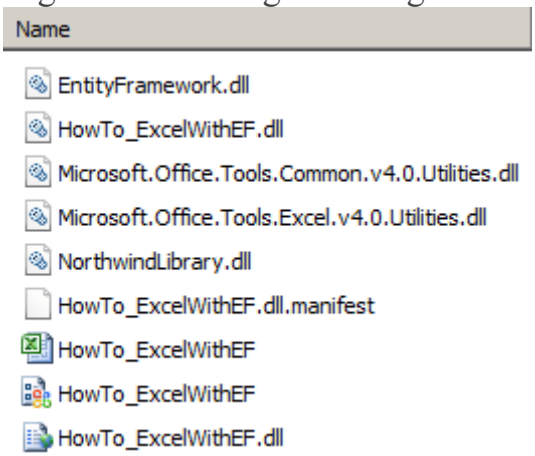
son olarak Sheet3 içeriği de ürün bazlı satış rakamlarıyla doldurulacaktır.



	A	B
1	**Product**	**Sales**
2	Beverages-Côte de Blaye	46563.1
3	Dairy Products-Raclette Courdavault	33616.6
4	Meat/Poultry-Thüringer Rostbratwurst	33109.5
5	Grains/Cereals-Gnocchi di nonna Alice	32604
6	Produce-Manjimup Dried Apples	23550
7	Confections-Tarte au sucre	20762.8
8	Dairy Products-Camembert Pierrot	20652.3
9	Meat/Poultry-Alice Mutton	16580.9
10	Seafood-Carnarvon Tigers	15950
11	Dairy Products-Gudbrandsdalsost	14041.8
12	Produce-Rössle Sauerkraut	12854.3

Görüldüğü üzere bir **Excel** içeriğini **Entity Framework** üzerinden geçerek veri ile doldurmak son derece kolaydır. Elbette gerçek hayat senaryoları düşünüldüğünde, ilgili veri içeriğinin asenkron olarak ve hatta servisler yardımıyla doldurulması daha doğru bir yaklaşım olacaktır. Zaten **Excel**' in konuşabildiği pek çok dış dünya aracı servis bazlı veri sunmaktadır*(OData servislerinden veri çekilmesi ve Excel içerisinde Pivot table olarak gösterilebilmesi konusunu araştırabilirsiniz)*

Tamamlanan uygulamanın herhangibir bilgisayarda veya ortamda çalıştırılabilmesi için aşağıdaki görselde yer alan içeriğin taşınması yeterli olacaktır. Elbette veritabanı bağlantısının olduğunu ve ilgili **Northwind** içeriklerine ulaşılabildiğini varsayıyoruz.



Böylece geldik bir yazımızın daha sonuna. Size düşen örneği daha da zengilenştirmektir. Örneğin,

- n sayıda Sheet' in doldurulması noktasında **asenkron** bir işleyişin ele alınmasını sağlayabilirsiniz.
- Aynı uygulamayı **Office 2013** standartlarında ele alıp, **.Net Framework 4.5** hedefli düşünüp, **async** ve**await** anahtar kelimelerini devreye sokabilirsiniz.
- **Excel**' de yapılacak olan değişikliklerin, **Save** işlemleri sonrasında veri kaynağına doğru yansıtılmasını düşünebilirsiniz.
- Bazı **View**' lardan yararlanarak **Sheet** içerisine çok basit anlamda **Chart***(örneğin bir Pie Chart çok şık durabilir)* çizdirmeyi deneyebilirsiniz

vb Burada hayal gücünüzü kullanmadan önce, bir **Excel** uygulamasının aslında bir **Windows**uygulaması olduğunu ve **Visual Studio 2012** tarafında istenildiği gibi özelleştirilebildiğini düşünmenizde yarar olacaktır. Nitekim gördüğünüz üzere aynen bir **Windows Forms**uygulamasında olduğu gibi olay metodlarına girebiliyor ve **C#** kodlarımızı konuşturabiliyoruz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HowTo_ExcelWithEF.zip (1,20 mb)

[Orjinal Yazım Tarihi 03-05-2013]

WCF Uygulamalarında Enterprise Library Validation Block Kullanımı

Pazartesi, 24 Haziran 2013 12:26 by bsenyurt

[Orjinal Yazım Tarihi 10.09.2012]

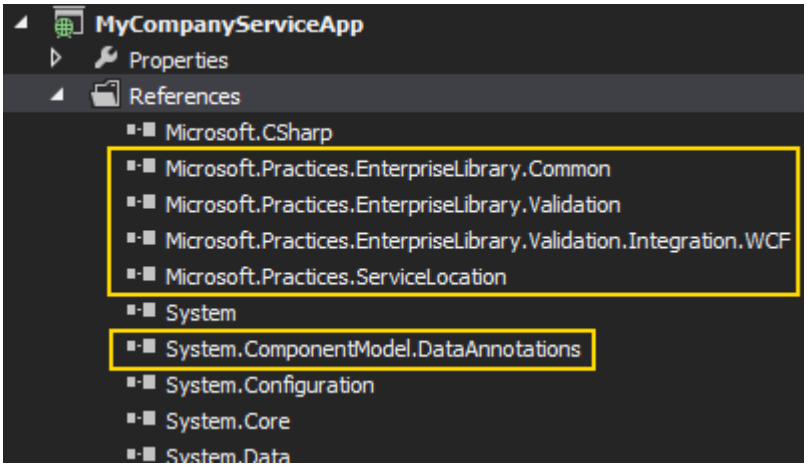
Merhaba Arkadaşlar,

Enterprise Library ve içerisinde yer alan **Application Block'** lar çoğunlukla projelerimizde ihtiyaç duyduğumuz ve**Cross-Cutting** olarak geçen parçaların hızlı ve kolay bir biçimde uygulanmasında kullanılmaktadır. **Cross-Cutting'** ler özellikle birden fazla katmandan oluşan proje bazlı çözümlerde, katmanların pek çok noktasında sıklıkla kullanılabilen*(ihtiyaç duyulabilen)* fonksiyonelliklerdir. Örneğin **Exception Handling, Security, Cryptography, Configuration, Logging, Validation, Caching** vb…Bu tip modüler yapılar çok sık kullanıldıklarından her çözüm için ayrı ayrıgeliştirilmemektedir/geliştirilmemelidir. Bunun yerine yeniden kullanılabilen modüler yapılar olarak ele alınmaları daha doğru bir yaklaşımdır. Örneğin **Enterprise Library**

Biz bu makalemizde **WCF(Windows Communication Foundation)** servislerinde, **ValidationApplication Block'** u nasıl kullanabileceğimizi incelemeye çalışıyor olacağız. Bu block yardımıyla**nitelik bazlı(Attribute Based)** olacak şekilde doğrulama*(Validation)* kontrolleri yapılabilmektedir. Söz konusu doğrulama kontrolleri sınıfların özelliklerine uygulanan nitelikler ile yapılabileceği gibi, metodların parametreleri üzerine de enjekte olabilmektedir. Dilerseniz adım adım senaryomuzu geliştirip konuyu basit seviye de kavramaya çalışalım.

Örnek senaryomuzda **WCF** tabanlı bir servis üzerinden bir metod çağrısı ile **Player** isimli bir nesne örneğinin oluşturulmasını sağlıyor olacağız. **Player** tipinin özelliklerine ait değerler, servis operasyonuna parametre olarak gelecekler. İşte doğrulama kriterlerimiz de bu notkada devreye girecek ve bazı veri giriş ihlallerini kontrol edecekler. Haydi başlayalım

İlk olarak servis tarafını geliştireceğiz. **WCF Service Application** şablonundan üretilen uygulamamızda, aşağıdaki ekran görüntüsünde yer alan referansların bulundurulması gerekmektedir.

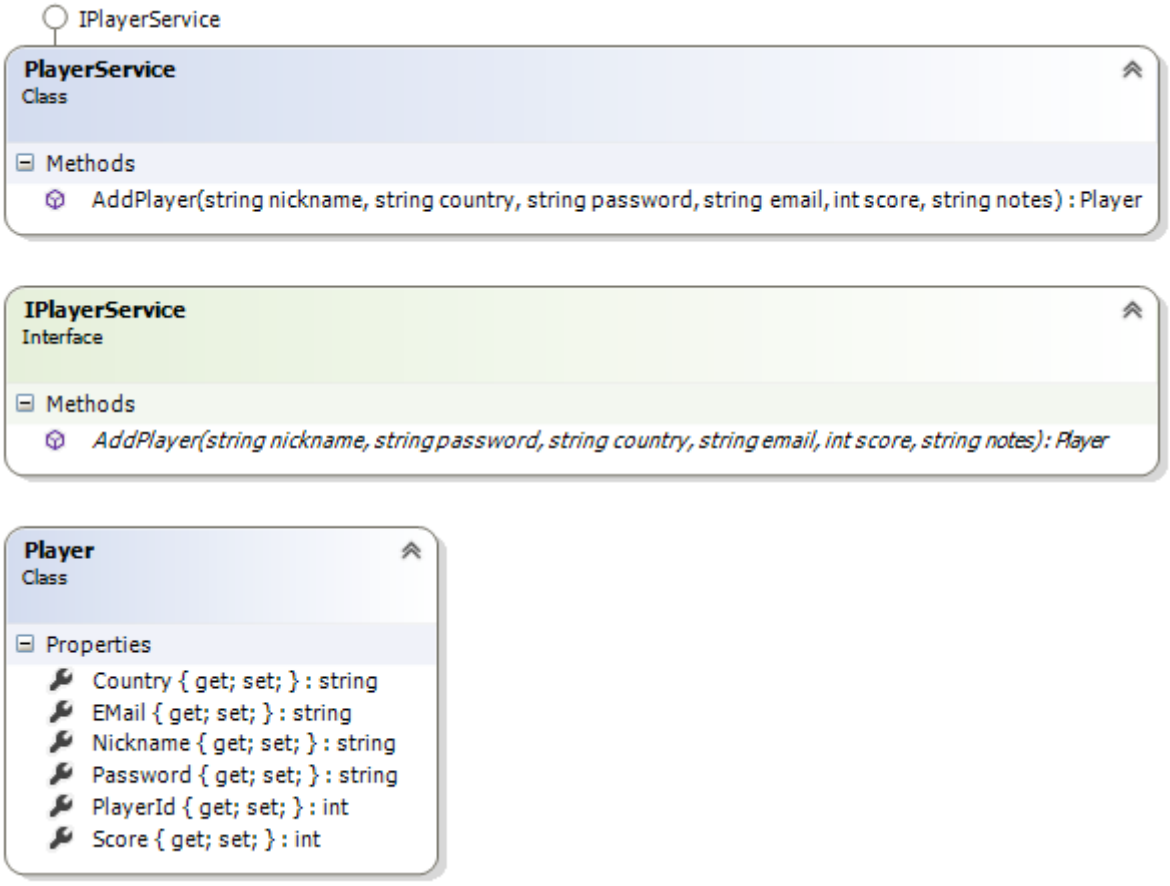


Enterprise Library' sisteme yüklendikten sonra kurulduğu yerdeki **bin** klasöründen

- Microsoft.Practices.EnterpriseLibrary.Common
- Microsoft.Practices.EnterpriseLibrary.Validation
- Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WCF
- Microsoft.Practices.ServiceLocation

assembly' larının yüklenmesi gerekmektedir. Bunlara ek olarak**System.ComponentModel.DataAnnotations.dll assembly'** ının da ayrıca projeye ilave edilmesi gerekecektir.

İlgili **Assembly'** ların referans edilmesinin ardından servis tarafındaki uygulamanın geliştirilmesine başlanabilir. Örnek senaryomuza göre doğrulama denetimi, servis operasyonundaki metod parametreleri seviyesinde yapılacaktır. Şimdi aşağıdaki sınıf diagramında görülen tipleri geliştirmeye başlayalım.



Player tipinin içeriği aşağıdaki gibidir.

```
using System.Runtime.Serialization;
namespace MyCompanyServiceApp
{
   [DataContract]
   public class Player
   {
      [DataMember]
      public int PlayerId { get; set; }

      [DataMember]
      public string Nickname { get; set; }
      [DataMember]
      public string Password { get; set; }
      [DataMember]
      public string Country { get; set; }

      [DataMember]
      public string EMail{ get; set; }
      [DataMember]
      public int Score{ get; set; }
```

```
    }
}
```

Servis sözleşmesini ise aşağıdaki gibi tasarlayacağız.

```
using Microsoft.Practices.EnterpriseLibrary.Validation;
using Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WCF;
using Microsoft.Practices.EnterpriseLibrary.Validation.Validators;
using System.ServiceModel;
namespace MyCompanyServiceApp
{
    [ServiceContract]
    public interface IPlayerService
    {
        [OperationContract]
        [FaultContract(typeof(ValidationFault))]
        Player AddPlayer(
            [ValidatorComposition(CompositionType.And)]
            [StringLengthValidator(5, 10, MessageTemplate = "Nickname en az 5 en fazla
10 karakter olabilir")]
            [NotNullValidator()]
            string nickname,
            string password,

            [ValidatorComposition(CompositionType.And)]
            [StringLengthValidator(3, 30, MessageTemplate = "Ülke bilgisi en az 3 en
fazla 30 karakter olabilir")]
            [NotNullValidator()]
            string country,
            [RegexValidator(@"^(?("")(""".+?""@)|(([0-9a-zA-Z]((\.(?!\.))|[-
!#\$%&'\*\+/=\?\^`\{\}\|~\w])*)(?<=[0-9a-zA-Z])@))(?(\[)(\[(\d{1,3}\.){3}\d{1,3}\])|(([0-
9a-zA-Z][-\w]*[0-9a-zA-Z]\.)+[a-zA-Z]{2,6}))$")]
            string email,
            [RangeValidator(0, RangeBoundaryType.Inclusive, 1000,
RangeBoundaryType.Inclusive, MessageTemplate = "Score bilgisi 0 ile 1000 arasında
olabilir")]
            int score,
            [StringLengthValidator(
                10
                , 100
                , MessageTemplate = "Notlar en az 10 en fazla 100 karakter uzunluğunda
olabilir"
                , ErrorMessageResourceName="Notes"
```

```
            ,ErrorMessageResourceType=typeof(string)
            )]
        string notes
        );
    }
}
```

Dikkat edileceği üzere metod parametrelerinde bazı **nitelikler*(Attribute)*** kullanılmıştır. **Validator**kelimesi ile biten bu nitelik sınıfları yardımıyla bazı doğrulama kriteleri set edilmiştir. Örneğin**nickname** bilgisinin string uzunluğu 5 ile 10 karakter arasında olmalıdır. Benzer durum **notes** ve**country** parametreleri için de geçerlidir. **NotNullValidator**, tahmin edileceği üzere uygulandığı parametreinin **null** olmamasını gerektirmektedir. Özellikle referans tiplerinin*(örneğin string)* null geçilmemesi bir doğrulama kriteri olarak sunulabilir. Kullanımı etkin olan doğrulama tiplerinden birisi de **RegexValidator'** dur**. String** tipinden parametrelere uygulanmakta olup, verinin bir **RegEx** desenine göre doğruluğunun kontrol edilmesini sağlamaktadır. Örnekte **email** verisinin **geçerli bir elektronik posta adresi** olup olmadığının kontrolü yapılmaktadır.

Birden fazla doğrulama kriteri uygulanmak istediğinde ise bu **and'** li veya **or'** lu bir biçimde enjekte edilebilir. Bunun için **ValidatorComposition** niteliğinin kullanılması yeterlidir. Örneklerimizde **And**opsiyonu etkinleştirilmiştir. Yani takip eden tüm **Validator'** ların aynı anda **true** olması halinde bir ihlal söz konusu olmayacaktır. **AddPlayer** isimli servis operasyonuna uygulanan bir diğer nitelikte **FaultContract'** tır. Nitelik,**Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WCF** isim alanı*(namespace)*altında yer alan **ValidationFault** tipini kullanmaktadır. Dolayısıyla bir hata sözleşmesi olarak **WCF'**in **Validation Application Block** için üretilmiş olan **ValidationFault** tipi kullanılacak ve istemci tarafında gönderilecektir.

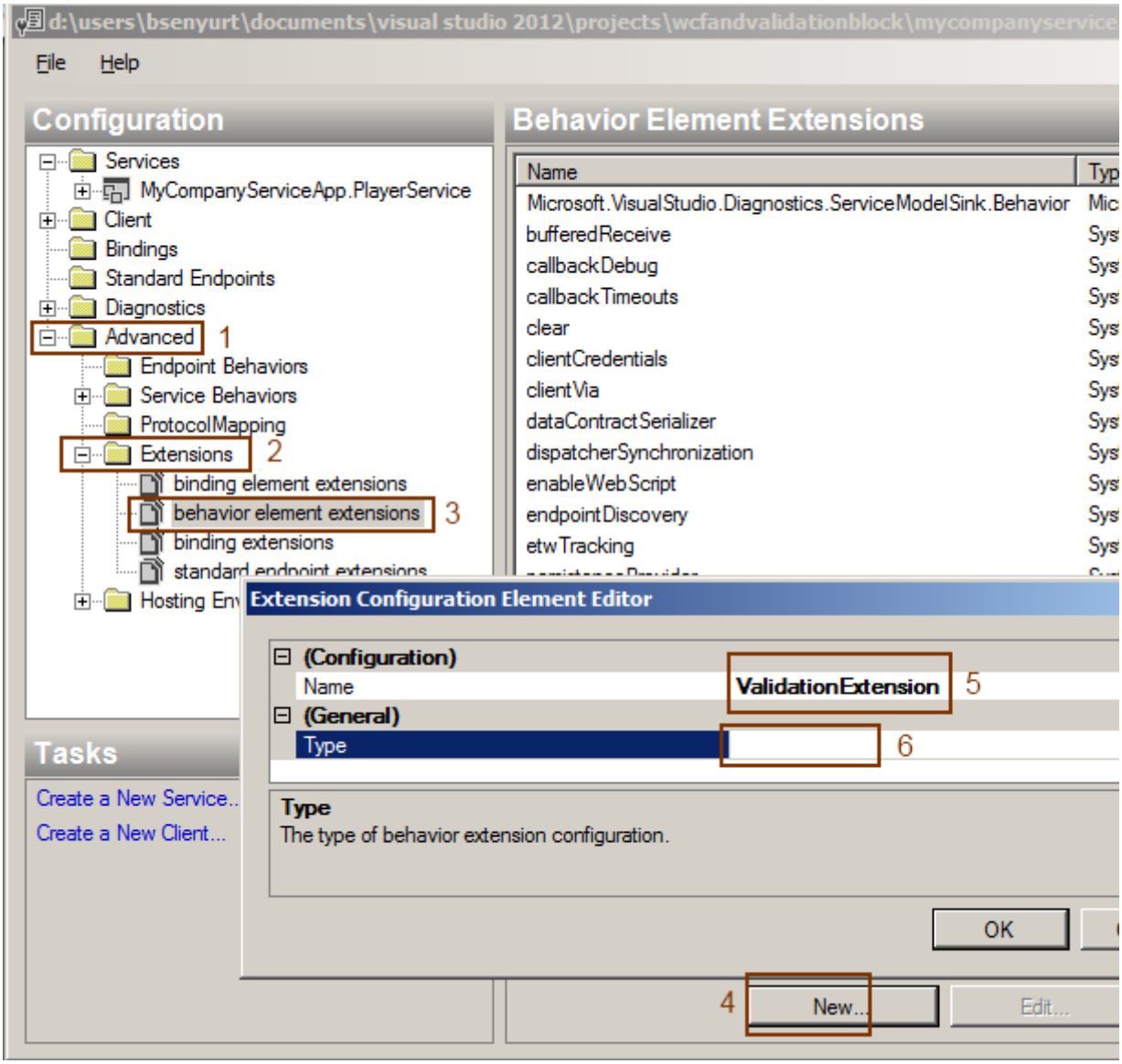
Servis sözleşmesinin uygulandığı sınıf kodları ise aşağıdaki gibidir.

```
using System;
namespace MyCompanyServiceApp
{
    public class PlayerService
        : IPlayerService
    {
        public Player AddPlayer(
            string nickname,
            string country,
            string password,
            string email,
            int score,
            string notes)
        {
```

```
            Random rnd = new Random();
            return new Player
            {
               PlayerId=rnd.Next(1,100),
               Nickname=nickname,
               Password=password,
               EMail = email,
               Score=score,
               Country=country
            };
         }
      }
}
```

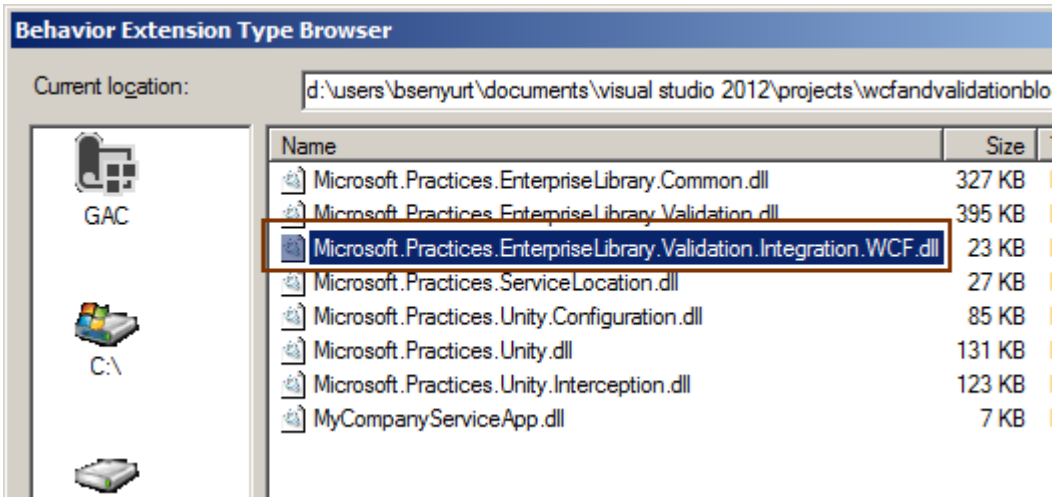
PlayerService sınıfı içerisinde yer alan **AddPlayer** metodu içerisinde çok olağanüstü bir çalışma yoktur. Sadecebir **Player** nesnesi örneklenmekte ve istemci tarafına geri gönderilmektedir. *Tabi herhangibir doğrulama kriterine takılınmadıysa.*
Buraya kadar yapılan hazırlıklar ne yazık ki yeterli değildir. **WCF** çalışma zamanının da söz konusu**Validation Application Block** ile kullanılacağının bir şekilde belirtilmesi gerekir. Bu aslında servis**endpoint'** i için ilave bir **davranış*(Behavior)*** belirtilmesinden başka bir şey değildir. Bunun için**WCF Service Configuration Editor'** ü kullanabiliriz. Şimdi bu adımlarımızı sırasıyla gerçekleştirelim.

Adım 1



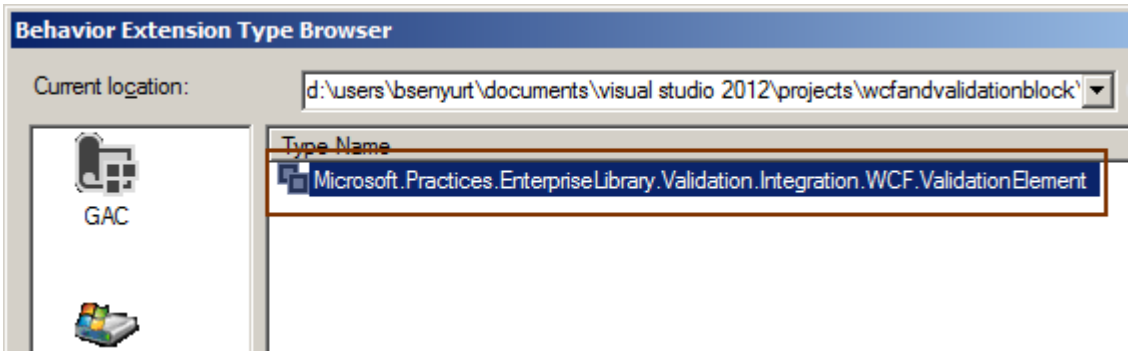
İlk olarak **Advanced->Extensions->behavior element extensions** kısmına gidilir ve buradan**New** düğmesine basıldıktan sonra çıkan iletişim pencersine geçilir. **Name** özelliğine bir değer verdikten sonra ise **Type** özelliğinin karşısında bulunan 3 nokta düğmesine basılır.

Adım 2

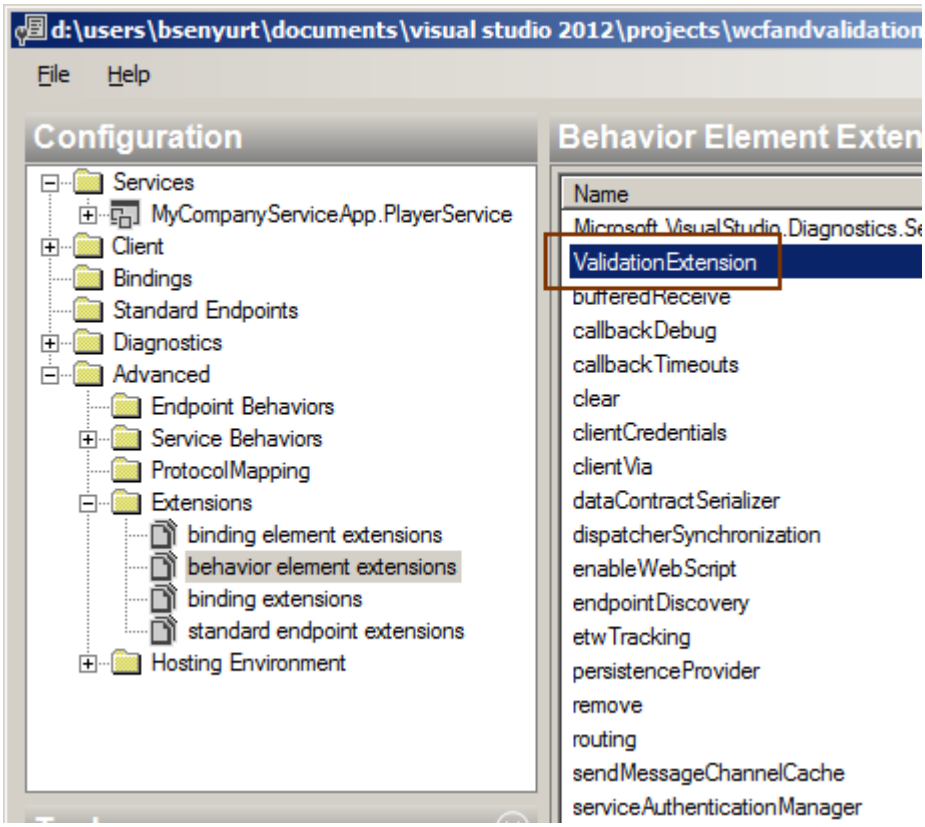


Üç nokta düğmesine basıldıktan sonra ise, projenin **bin** klasörüne eklenmiş olan**Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WCF.dll assembly'** ı seçilir.

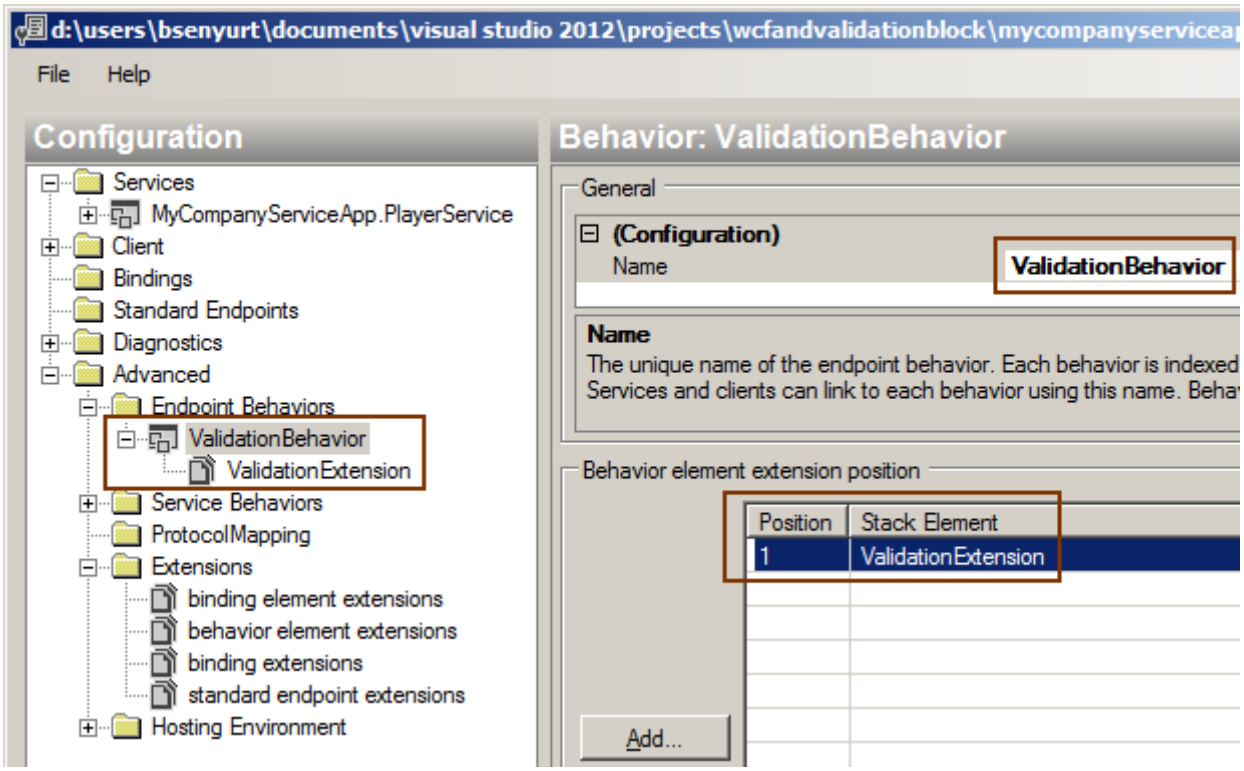
Adım 3



Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WCF.dll assembly' nın seçilmesini takiben gelen ekrandaki tek tip olan **ValidationElement** işaretleniz. Bu durumda aşağıdaki ekran görüntüsünde yer aldığı gibi ilgili elementin, **behavior element extensions**kısmına eklenmiş olduğu görülür.



Adım 4

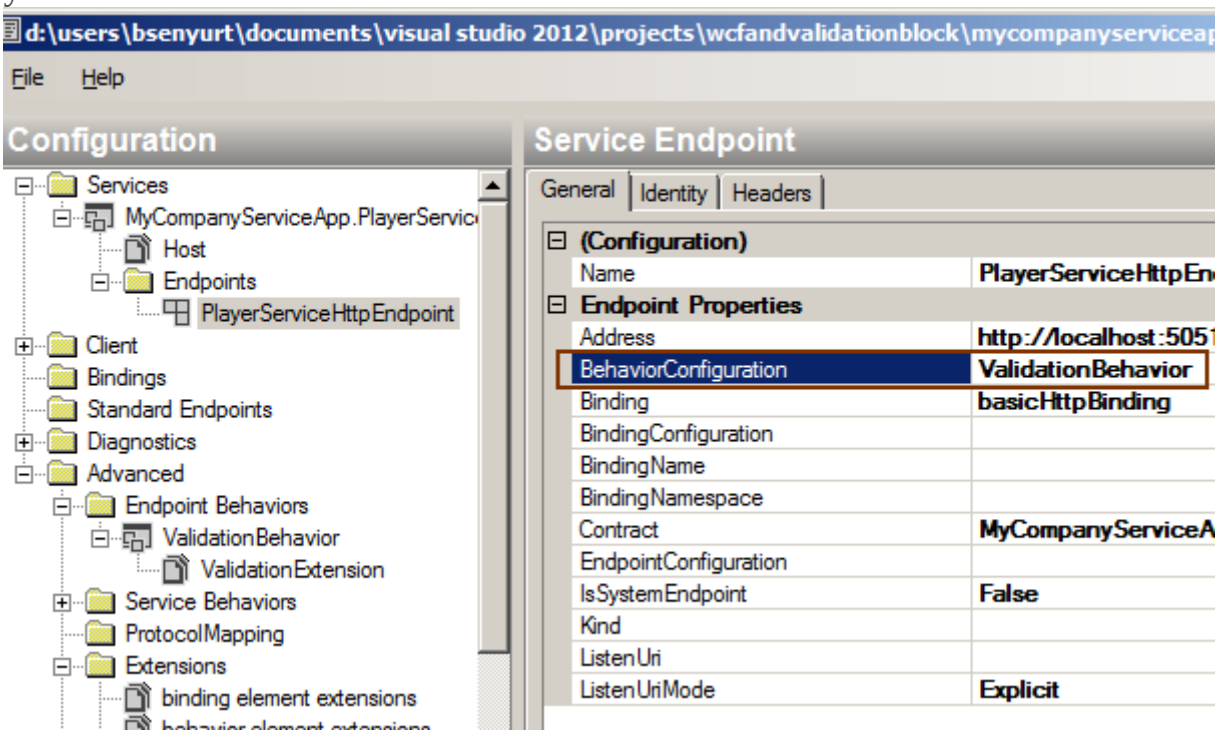


Sıradaki adımda ilgili **ValidationExtension** elementinin bir **EndPoint** davranışı haline getirilmesi yer almaktadır. Bunun için **Advanced->Endpoint Behaviors** kısmına yeni bir davranış ilave edilmelidir. Davranışa örnek bir isim verildikten sonra ise **Add** düğmesi

yardımıyla biraz önce sisteme dahil edilmiş olan **ValidationExtension** tipinin eklenmesi sağlanır.

Adım 5

Son olarak yeni **Endpoint** davranışının, servise ait **Endpoint** ile ilişkilendirilmesi yeterli olacaktır. Bunun için **Services->[Service Adı]->Endpoints->[Endpoint Adı]** kısmından gelen özelliklerden **BehaviorConfiguration**' a **ValidationBehavior** değerinin atanması yeterlidir.



Sonuç olarak servis tarafına ait konfigurasyon dosyası içeriği aşağıdaki gibi olacaktır.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <extensions>
      <behaviorExtensions>
        <add
name="ValidationExtension"type="Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WCF.ValidationElement,Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WCF, Version=5.0.414.0,Culture=neutral,
PublicKeyToken=31bf3856ad364e35" />
      </behaviorExtensions>
    </extensions>
    <services>
      <service behaviorConfiguration="PlayerServiceBehavior"
name="MyCompanyServiceApp.PlayerService">
```

```
            <endpoint address="http://localhost:50511/PlayerService.svc"
              behaviorConfiguration="ValidationExtensionBehavior"binding="basicHttpBinding"
              name="PlayerServiceHttpEndpoint"
contract="MyCompanyServiceApp.IPlayerService" />
        </service>
      </services>
      <behaviors>
        <endpointBehaviors>
          <behavior name="ValidationExtensionBehavior">
            <ValidationExtension />
          </behavior>
        </endpointBehaviors>
        <serviceBehaviors>
          <behavior name="PlayerServiceBehavior">
            <serviceMetadata httpGetEnabled="true" />
            <serviceDebug includeExceptionDetailInFaults="false" />
          </behavior>
        </serviceBehaviors>
      </behaviors>
  </system.serviceModel>
</configuration>
```

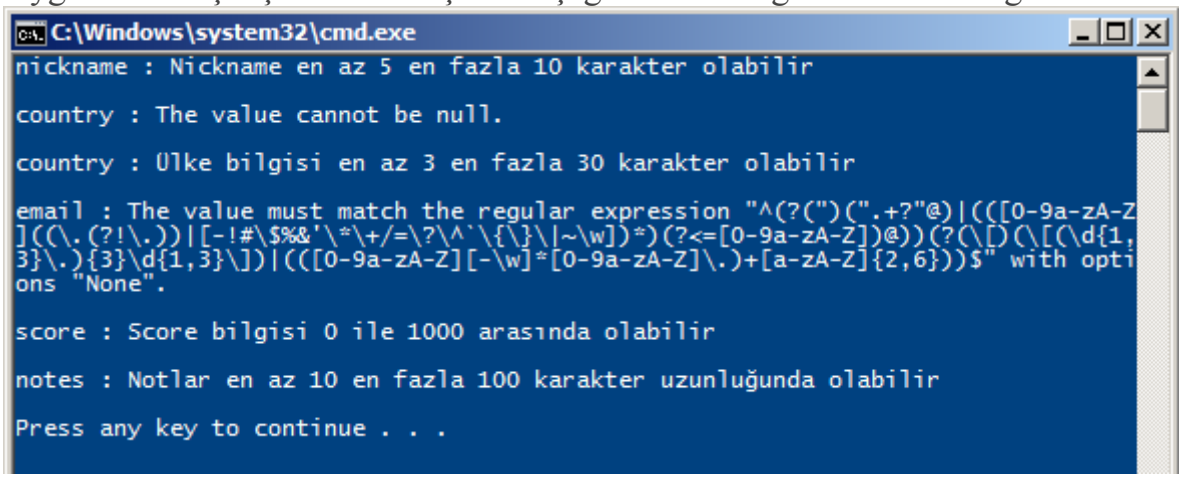
Şimdi örnek bir istemci uygulama geliştirerek senaryomuzu teste çıkabiliriz. Basit bir **Console**uygulaması pekala işimizi görecektir **Console** uygulamasına servis referansını ekledikten sonra, aşağıdaki kodları geliştirdiğimizi düşünelim.

İstemci uygulamada ValidationFault tipinin kullanılabilmesi için, Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WCF.dll assemby'ının da projeye referans edilmesi gerekmektedir. Bu Assembly ne yazık ki Add Service Reference seçeneği sonrası otomatik olarak eklenmemektedir.

```
using ClientApp.Company;
using Microsoft.Practices.EnterpriseLibrary.Validation.Integration.WCF;
using System;
using System.ServiceModel;
namespace ClientApp
{
  class Program
  {
    static void Main(string[] args)
    {
      PlayerServiceClient proxy = new
PlayerServiceClient("PlayerServiceHttpEndpoint");
```

```
            try
            {
               Player newPlayer = proxy.AddPlayer(
                  nickname: "kısa"
                  , password: "şifre"
                  , country: null
                 , email: "deneme"
                  , score: -10
                  , notes: "kısa not"
                 );
               //Player newPlayer = proxy.AddPlayer(
               //       nickname: "burkicik"
               //       , password: "şifre"
               //       , country: "Birleşik Krallık"
               //       , email: "selim@buraksenyurt.com"
               //       , score: 128
               //       , notes: "oyuna yeni katılmış bir oyuncudur ve ilk seferinde turnayı
gözünde vurmuştur"
               //       );
            }
            catch (FaultException<ValidationFault> excp)
            {
               foreach (ValidationDetail validationDetail in excp.Detail.Details)
               {
                  Console.WriteLine("{0} : {1}\n", validationDetail.Tag,
validationDetail.Message);
               }
            }
            catch (Exception excp)
            {
               Console.WriteLine(excp.Message);
            }
            finally
            {
               if (proxy.State == CommunicationState.Opened)
                  proxy.Close();
            }
         }
      }
   }
}
```

Dikkat edileceği üzere doğrulama kurallarına takılacak şekilde test verileri girilmeye çalışılmıştır. Oluşması muhtemel ne kadar doğrulama hatası varsa, bunların tamamı**FaultException<ValidationFault>** ile gelen **exception** nesne örneğinin **Detail** özelliğinin**Details** koleksiyonunda toplanacaktır. Dolayısıyla bu koleksiyon içeriği dolaşılarak ilgili doğrulama ihlallerine ait bir takım bilgilere ulaşılabilinir.
Uygulamanın çalışma zamanı çıktısı aşağıdaki ekran görüntüsündeki gibi olacaktır.

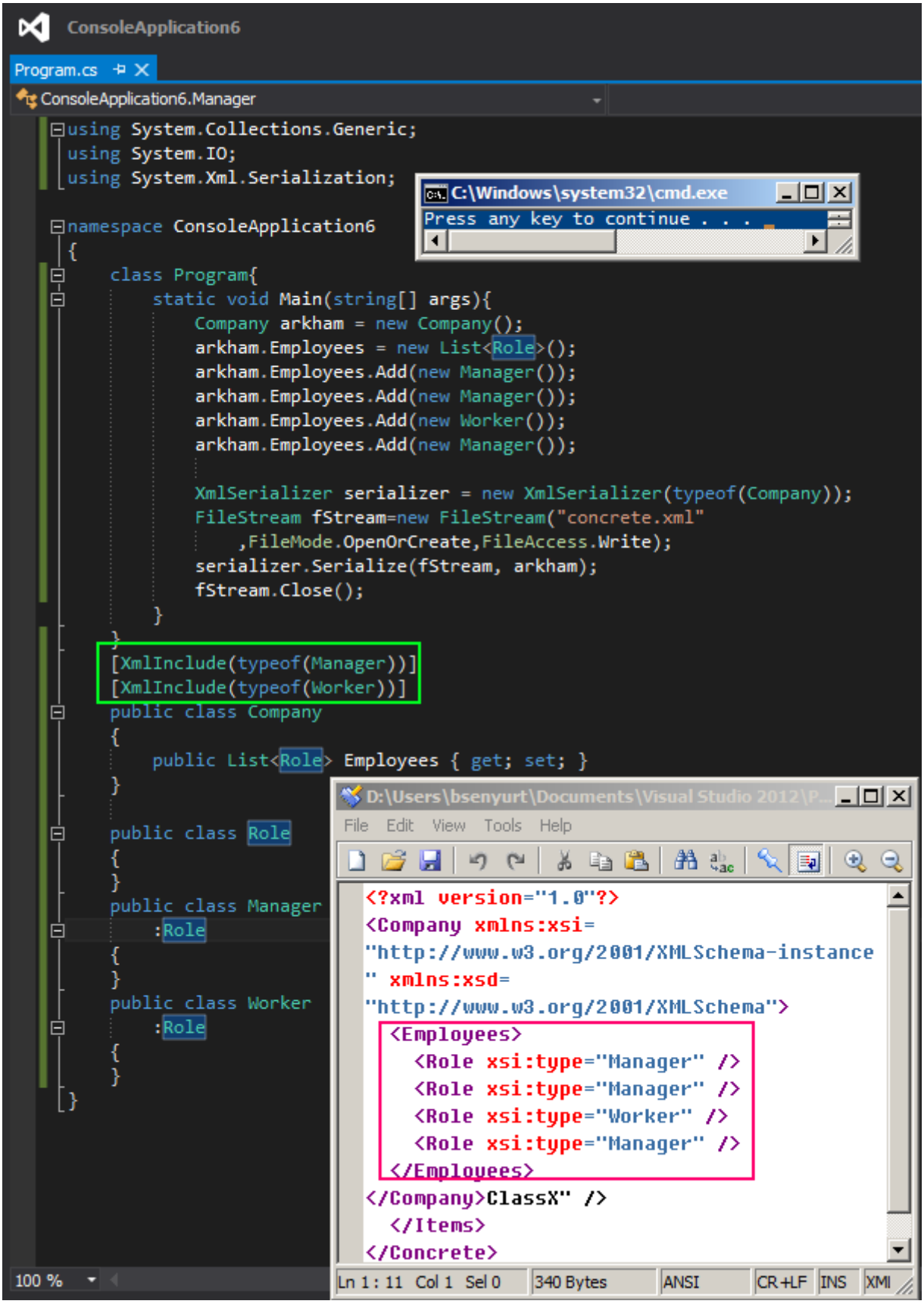


nickname, country, email, score ve **notes** metod parametrelerinde tanımlanan doğrulama ihlalleri istemci tarafına bu şekilde yansımıştır. Tabi yorum satırları altına alınmış Player' ın oluşturulmasını denersek, bu durumda herhangibir doğrulama kriterine takılmadan ekleme işleminin yapılabildiğine şahit oluruz.
Görüldüğü gibi **Enterprise Library Validation Application Block**' u kullanarak, **WCF** tarafında operasyon bazında doğrulama kriterleri uygulanabilmektedir. Burada işin önemli yanlarından birisi ise, nitelik bazlı yapılan bu tanımlamaların servis operasyonları üzerinde gerçekleştirilmesidir. Dolayısıyla bu nitelikler değiştirilse bile, istemci tarafı için yeniden **Proxy** tipinin üretilmesine gerek yoktur Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.
WCFandValidationBlock.zip (1,92 mb)
[Orjinal Yazım Tarihi 10.09.2012]

Tek Fotoluk İpucu 102–Ne Zaman XmlInclude Gerekir?

Pazartesi, 24 Haziran 2013 11:00 by bsenyurt

Merhaba Arkadaşlar,
Diyelim ki elinizde **Role** isimli bir sınıf var. Hatta bu sınıftan türemiş **Manager** ve **Worker** isimli iki ayrı sınıf daha var. Hatta **Role** tipinden **Employees** isimli bir listeyi **özellik*(Property)*** olarak içeren**Company** isimli başka bir sınıf daha var…Derken **Company** sınıfına ait bir nesne örneğini çalışma zamanında **XML** serileştirmek istediniz. Klasik olarak **XmlSerializer** tipini işin içerisine kattınız. Peki ya sonra? Aldınız **InvalidOperationException** hatasını oturdunuz aşağıya Nasıl çözersiniz? Aşağıdaki gibi olabilir mi?



Aslında yapılan işlem gayet basittir. **Role** tipinin **Manager** ve **Worker** isimli sınıflara ait nesne örneklerini içerebileceği **XmlInclude niteliği*(Attribute)*** yardımıyla ifade edilmiştir. Hepsi bu. Bir başka ipucunda görüşmek dileğiyle 😉

Asp.Net Routing – Hatırlamak

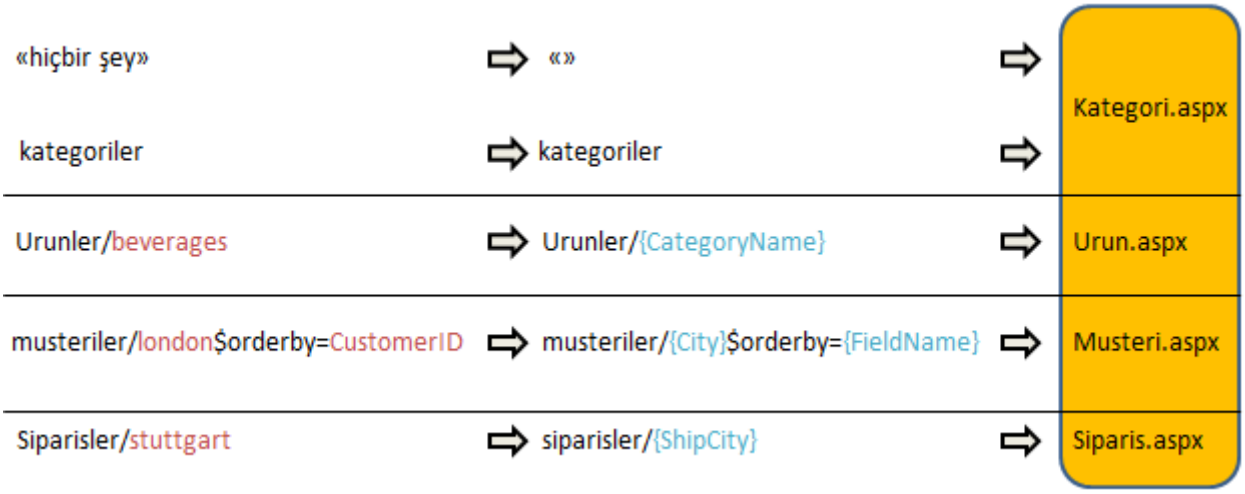
Cuma, 21 Haziran 2013 20:07 by bsenyurt

Merhaba Arkadaşlar,
Geçtiğimiz günlerde şirkette çok küçük bir web uygulamasına ihtiyaç duyuldu. Neredeyse bir günlük geliştirme maliyeti olan, küçük bir departmanın önemli bir gereksinimi karşılayacaktı. Tabi insan uzun zaman kodlama yapmayınca veya kodlamaya ara verince bazı temel bilgileri de unutabiliyor. Ben de kafayı **Team Foundation Server**entegrasyonu, **SOA** mimarisi ve **Scrum** gibi metodolojiler ile bozunca, zihinsel diskimdeki ana **partition**' a yeni bilgilerin yazıldığına ve eskilerinin yerinde yeller estiğine şahit oldum. Ama malum, günümüz teknolojilerinde bilginin tamamını ezberlemeye çalışmak yerine, en doğrusuna en hızlı şekilde nasıl ulaşabileceğimizi bilmek daha önemli. İşte bu felsefeden yola çıkıp dedim ki, şu**Asp.Net Routing** konusunu bir hatırlayayım ve hatta kayıt altına alayım. İşte hikayemiz böyle başladı

Asp.Net MVC' nin en cazip yanlarından birisi sanırım sağladığı **URL** eşleştirme*(Routing)* sistemidir. Özellikle **Search Engine Optimization*(SEO)*** kriterleri göz önüne alındığında, **orders.aspx?categoryName=Beverages&shipCity=istanbul&orderNumber=12903** gibi bir ifade yerine,**orders/beverages/istanbul/12903** şeklinde bir **URL** çok daha değerlidir.
Bilindiği üzere **Asp.Net 4.0** sürümü ile birlikte, **URL** ve asıl kaynak*(aslında yönlendirme sonucu gidilmesi gereken bir aspx sayfa kodu düşünebiliriz)* eşleştirmelerinde kullanılan yönlendirme işlemleri oldukça kolaylaştırılmıştır. İşte bu yazımızda, biraz temelleri hatırlamaya çalışacak ve**SEO**'cu arama motorlarının olmassa olmaz isterlerinden birisi olan **Routing** konusunu dört basit örnek üzerinden inceleyeceğiz. İlk olarak senaryomuza bir göz atalım.

Senaryo

Senaryomuzda veri kaynağı olarak emektar **Northwind** veritabanını kullanacağız. Örneklerimizde hem **Entity Framework** kullanacağımız hem de doğrudan **SQL** sorgusu çalıştıracağımız bir vakamız yer alacak. Temel olarak amacımız aşağıdaki ekran görüntüsünde yer alan **URL** eşleştirmelerini web uygulaması üzerinden işlettirmek.



Dikkat edileceği üzere **URL** satırından girilecek olan anlamlı ifadeler, aslında arka planda bir eşleştirme tablosuna uygun olacak şekilde ilgili kaynaklara yönlendirilmekteler. Örneğin **beverages**isimli kategoride yer alan ürünlerin listelenmesi için yazılan **urunler/beverages** sorgusu, sisteme daha önceden öğretilen **Urunler/{CategoryName}** üzerinden geçerek **urun.aspx** sayfasına yönlendiriliyor. Çok doğal olarak ilgili sayfa içerisinde, **CategoryName** değerine bakılarak bir sonuç kümesinin sunulması gerekiyor.

Route Eşleştirmelerinin Ayarlanması

Bu işlem için **global.asax.cs** dosyasında aşağıdaki kodlamaları yapmamız gerekmektedir.

```
using System;
using System.Web;
using System.Web.Routing;
namespace HowTo_EasyRouting
{
   public class Global
      : HttpApplication
   {
      private void SetRouteMaps()
      {
         RouteTable.Routes.MapPageRoute("Varsayilan", "", "~/kategori.aspx");
         RouteTable.Routes.MapPageRoute("Kategoriler", "kategoriler",
"~/kategori.aspx");
        RouteTable.Routes.MapPageRoute("KategoriBazliUrunler","urunler/{Catego
ryName}","~/urun.aspx");
         RouteTable.Routes.MapPageRoute("SehirBazliSiraliMusteriler",
"musteriler/{City}$orderby={FieldName}", "~/musteri.aspx");
         RouteTable.Routes.MapPageRoute("SehirBazliSiparisler",
```

```
"siparisler/{ShipCity}", "~/siparis.aspx");
        }
        protected void Application_Start(object sender, EventArgs e)
        {
            SetRouteMaps();
        }
    }
}
```

Application_Start olay metodu bilindiği üzere, Web uygulaması ayağa kalktığında devreye girmektedir. Dolayısıyla uygulamanın başladığı bir yerde, **URL** eşleştirme tanımlamalarını yapmak son derece mantıklıdır. Olayın ana kahramanı **RouteTable** sınıfıdır. Söz konusu tipin **static** olarak erişilebilen **Routes** özelliği bir **RouteCollection** referansını işaret etmektedir. Bu koleksiyon tahmin edileceği üzere **URL** ile asıl kaynak eşleştirmelerini taşımaktadır. Bu nedenle **MapPageRoute**metodundan da yararlanılarak gerekli eşleştirme bilgileri koleksiyona eklenir.

İlk satır ile **Root URL** adresine gelen bir talebin doğrudan **kategori.aspx** sayfasına yönlendirilmesi gerektiği ifade edilmektedir. İkinci satırda ise web kök adresini takiben kategoriler şeklinde gelen bir ifadenin gelmesi halinde yine, kategori.aspx sayfasına gidilmesi gerektiği belirtilmektedir.

KategoriBazliUrunler ismi ile tanımlanmış eşleştirmeye göre, **urunler/{CategoryName}** şeklinde gelen talepler **urun.aspx** sayfasına yönlendirilmektedir. İlginç kullanımlardan birisi de**SehirBazliSiraliMusteriler** isimli eşleştirmedir. Burada **City** ve **FieldName** isimli iki **Route**parametresi söz konusudur. İfade ise size sanıyorum tanıdık gelecektir. Neredeyse bir **REST** servis sorgusuna*(örneğin **OData** sorgusuna)* oldukça yakın değil mi? 😎

Şimdi bu durumları kod tarafında nasıl karşılayacağımızı örnek bir Asp.Net uygulaması üzerinden incelemeye çalışalım.

Birinci Durum

İlk olarak **kategori.aspx** sayfasına doğru yapılacak yönlendirmeleri ele almaya çalışacağız. Bunun için web uygulamamıza **kategori.aspx** isimli bir sayfa ekleyip içeriği ile kod tarafını aşağıdaki gibi geliştirelim.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Kategori.aspx.cs"
Inherits="HowTo_EasyRouting.Kategori" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
```

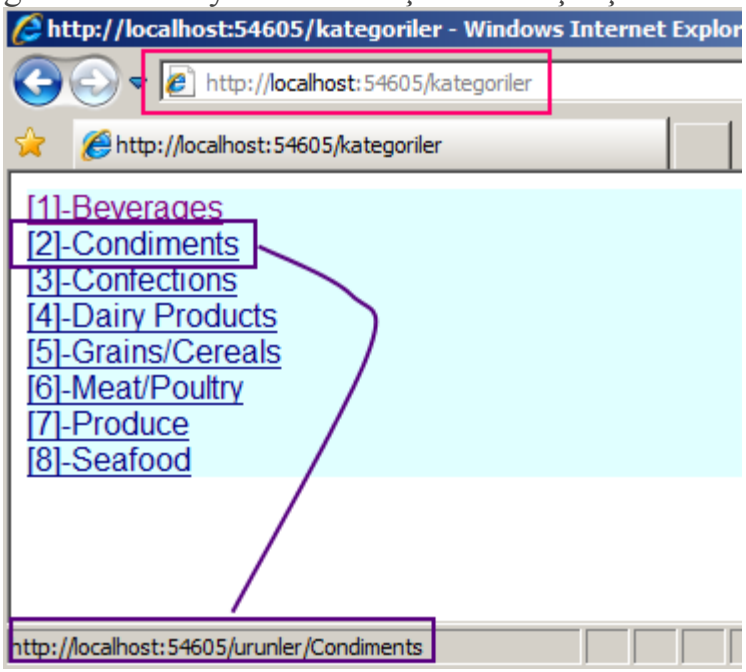
```
    <form id="form1" runat="server">
    <div id="divCategories" runat="server" style="background-color:lightcyan">

    </div>
    </form>
</body>
</html>
kod tarafı
using System;
using System.Linq;
using System.Web.UI;
using System.Web.UI.WebControls;
namespace HowTo_EasyRouting
{
    public partial class Kategori
        : Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!Page.IsPostBack)
            {
                using (NorthwindEntities context = new NorthwindEntities())
                {
                    var categories = from c in context.Categories
                                     orderby c.CategoryName
                                     select new
                                     {
                                         c.CategoryID,
                                         c.CategoryName
                                     };
                    foreach (var category in categories)
                    {
                        HyperLink categoryLink = new HyperLink();
                        categoryLink.NavigateUrl = GetRouteUrl("KategoriBazliUrunler",
new { CategoryName = category.CategoryName });
                        categoryLink.Text = string.Format("[{0}]-{1}<br/>",
category.CategoryID.ToString(), category.CategoryName);
                        divCategories.Controls.Add(categoryLink);
                    }
                }
            }
```

```
        }
    }
}
```

Aslında **kategori.aspx** sayfasında tipik olarak **Entity Framework** odaklı bir sorgulama gerçekleştirilmekte ve kategori adları birer **HyperLink** bileşeni olarak **div** içerisine eklenmektedir. Konu itibariyle işin önemli olan kısmı ise **HyperLink** bileşeninin **NavigateUrl** özelliğine**GetRouteUrl** metodu sonucunun atanmasıdır.

GetRouteUrl metodu dikkat edileceği üzere iki parametre alır. İlk parametre **route** adıdır. Yazdığımız değere göre **urunler/{CategoryName}** şeklindeki atama değerlendirilir. İkinci parametre ise bu **Route** içerisinden kullanılmak istenen değişken adı ve değerini içeren nesnenin örneklendiği kısımdır. **CategoryName** tahmin edileceği üzere **Route** tanımı içerisindeki parametre adıdır. Değeri ise zaten **LINQ*(Language INtegrated Query)*** sorgusu içerisinden elde edilmektedir. İkinci parametre **object** tipinden olduğundan **bir isimsiz tip*(anonymous type)*** ataması yapılabilmiştir. Bu nedenle **Route** içerisinde birden fazla parametre olması halinde, isimsiz tipin de birden fazla özellik içermesi gerektiğini ifade edebiliriz.

İlk durumda herhangibir sayfa talep edilmediğinde veya kök web adresi ardından **/kategoriler**şeklinde bir **URL** ifadesi kullanıldığında, aşağıdaki ekran görüntüsünde yer alan sonuçlar ile karşılaşırız.



Dikkat edilmesi gereken en önemli nokta, her hangi bir bağlantı üstüne gelindiğinde oluşan sorgu adresidir.

Örneğin **Condiments** için **http://localhost:54605/urunler/condiments** şeklinde bir**URL** tanımı oluşmuştur. Peki bu bağlantıya tıklanırsak ne olur? 🤓

İkinci Durum

kategori.aspx sayfasında bir bağlantıya tıklandığında, **HyperLink** bileşeninin **NavigateUrl**özelliğinin sahip olduğu değerin **Route** tablosundaki eşleniğine bakılmalıdır. Yaptığımız tanımlamalara göre **urun.aspx** sayfasına gidilmesi beklenmelidir*(KategoriBazliUrunler isimli Route tanımına dikkat edin)* Buna göre **urun.aspx** sayfasının içeriğini aşağıdaki gibi düzenleyebiliriz.

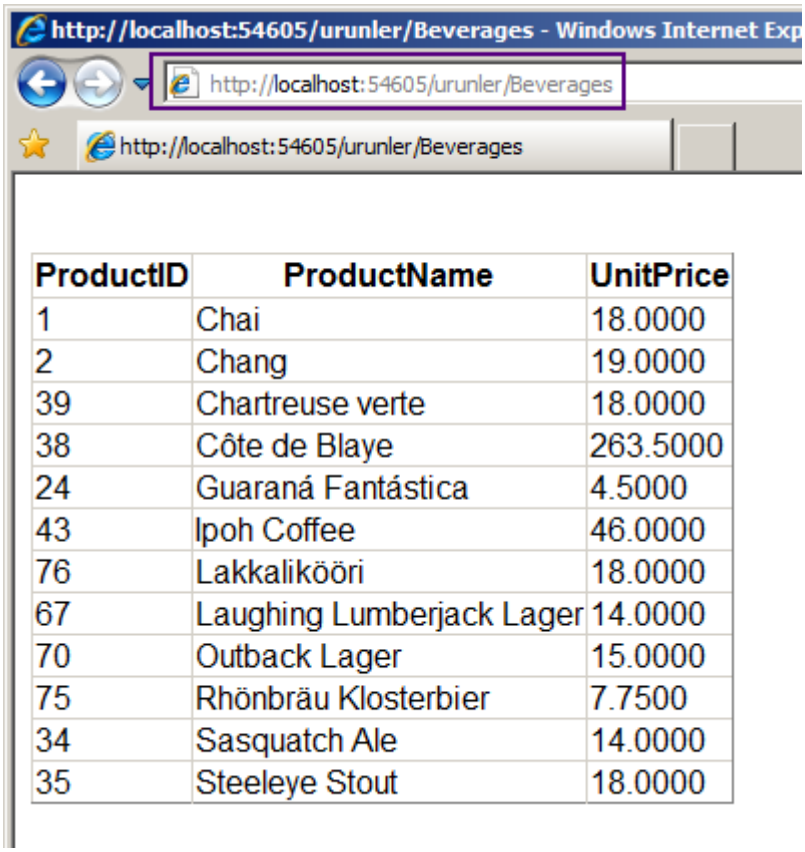
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Urun.aspx.cs"
Inherits="HowTo_EasyRouting.Urun" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1 style="color:purple">
       <asp:Label ID="lblCategoryName" runat="server" /></h1>
      <br />
      <asp:GridView ID="grdUrunler" runat="server" />
   </div>
   </form>
</body>
</html>
```

kod tarafı

```
using System;
using System.Linq;
using System.Web.UI;
namespace HowTo_EasyRouting
{
   public partial class Urun
      : Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         if (RouteData.Values["CategoryName"] != null)
         {
            string categoryName = RouteData.Values["CategoryName"].ToString();
            using (NorthwindEntities context = new NorthwindEntities())
            {
               var products = from p in context.Products.Include("Product")
```

```
                    where p.Category.CategoryName == categoryName
                    orderby p.ProductName
                    select new
                    {
                       p.ProductID,
                       p.ProductName,
                       p.UnitPrice
                    };
            grdUrunler.DataSource = products.ToList();
            grdUrunler.DataBind();
          }
        }
        else
          Response.Redirect(GetRouteUrl("Kategoriler", null));
      }
    }
}
```

Tabi bu sayfaya gelindiğinde aslında **Route** tanımlaması içerisinde yer alan parametre değerinin okunması gerekmektedir. Bu sebepten sayfanın **RouteData** özelliğinden hareket edilerek**RouteValueDictionary** tipinden olan **Values** özelliğine gidilir ve indeksleyiciye verilen**CategoryName** alanının var olup olmadığına bakılır. Malum urun.aspx sayfasına farklı bir şekilde erişilmek istenebilir ve **CategoryName** değeri null olarak gelebilir. Bu nedenle bir **null** değer kontrolü ardından **Entity** sorgulama işlemi yapılmıştır. **RouteData.Values[“CategoryName”]** ile**URL** satırındaki kategori adı bilgisi alındıktan sonra standart olarak bir **Entity** sorgusu icra edilmektedir. Eğer kategori adı **null** olarak gelirse bu durumda varsayılan **URL** eşleştirilmesi nedeniyle kategorilerin gösterildiği sayfaya gidilir.



ProductID	ProductName	UnitPrice
1	Chai	18.0000
2	Chang	19.0000
39	Chartreuse verte	18.0000
38	Côte de Blaye	263.5000
24	Guaraná Fantástica	4.5000
43	Ipoh Coffee	46.0000
76	Lakkalikööri	18.0000
67	Laughing Lumberjack Lager	14.0000
70	Outback Lager	15.0000
75	Rhönbräu Klosterbier	7.7500
34	Sasquatch Ale	14.0000
35	Steeleye Stout	18.0000

Üçüncü Durum

URL eşleştirmelerinden **SehirBazliSiraliMusteriler** isimli olanı, iki adet **route** parametresi içermektedir. Burada başta da belirttiğimiz üzere **OData** sorgularına benzer bir ifade tanımlanmıştır. Eşleştirme bilgisine göre **musteri.aspx** sayfasına doğru bir yönlendirme söz konusudur.**musteri.aspx** içeriğini aşağıdaki gibi geliştirdiğimizi düşünelim.

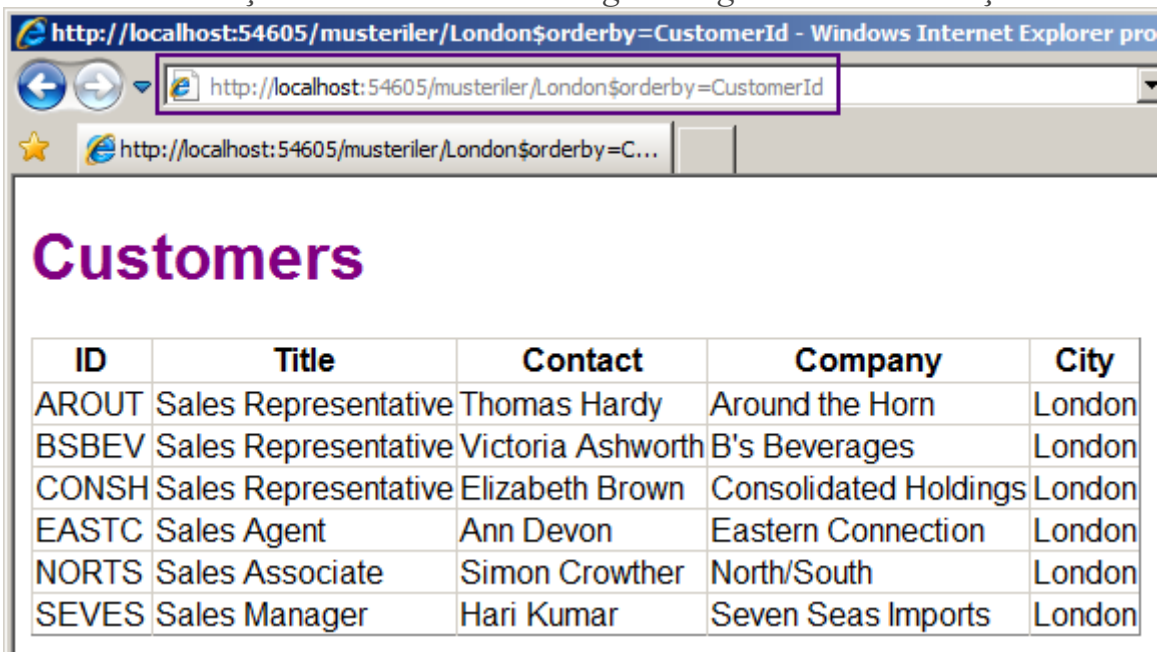
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Musteri.aspx.cs"
Inherits="HowTo_EasyRouting.Musteri" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        <h1 style="color:purple">Customers</h1>
        <asp:GridView ID="grdCustomer" runat="server" />
    </div>
    </form>
```

```
</body>
</html>
kod tarafı
using System;
using System.Linq;
using System.Reflection;
using System.Web.Routing;
namespace HowTo_EasyRouting
{
   public partial class Musteri : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
        if(RouteData.Values["City"]!=null
            || RouteData.Values["FieldName"]!=null)
         {
            string cityName=RouteData.Values["City"].ToString();
            string fieldName=RouteData.Values["FieldName"].ToString();

            using(NorthwindEntities context=new NorthwindEntities())
            {
               var customers = context
                  .Customers
                  .Where(c => c.City == cityName)
                  .OrderBy(GetField<Customer>(fieldName))
                  .Select(c => new
                  {
                     ID=c.CustomerID,
                     Title=c.ContactTitle,
                     Contact=c.ContactName,
                     Company=c.CompanyName,
                     c.City
                  });
               grdCustomer.DataSource = customers.ToList();
               grdCustomer.DataBind();
            }
         }
      }
      public static Func<T, string> GetField<T>(string fieldName)
      {
         PropertyInfo pInfo=typeof(T).GetProperty(fieldName);
```

```
            if (pInfo == null)
                pInfo = typeof(T).GetProperty("CustomerID");
            return o => Convert.ToString(pInfo.GetValue(o, null));
        }
    }
}
```

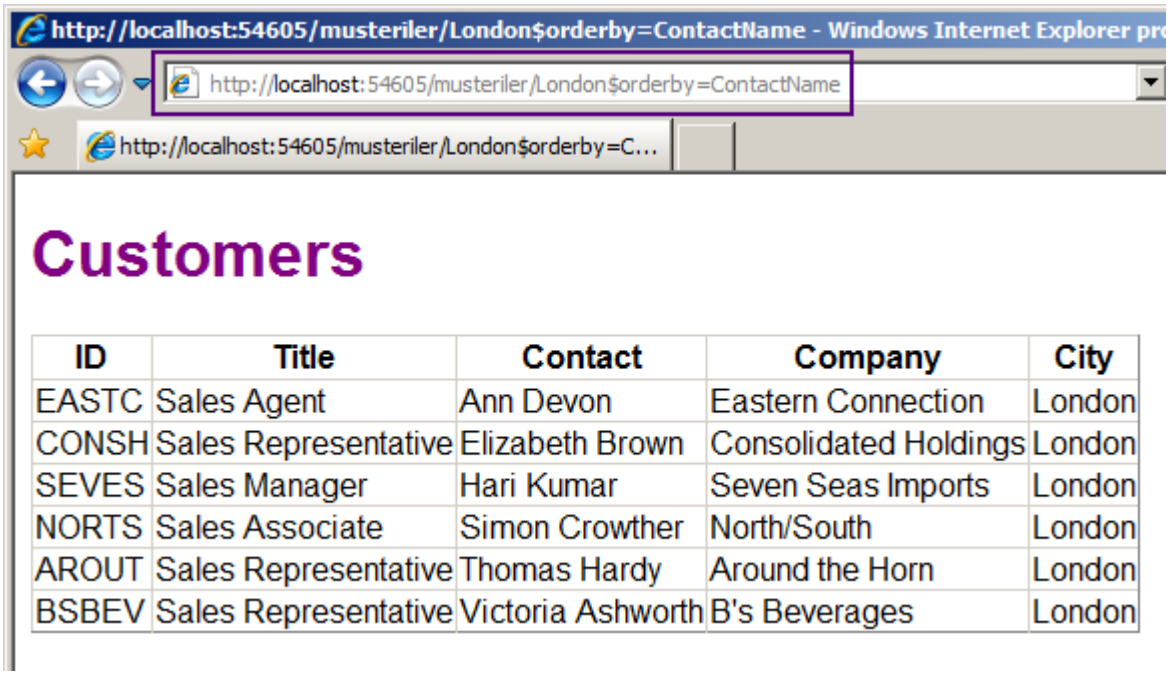
RouteData.Values özelliğinden yararlanılarak **CityName** ve **FieldName** değerlerinin **null** olup olmamasına göre bir kod parçası çalıştırılmaktadır. Bir önceki örnekten farklı bir durum olmasa da**Entity** sorgusunda **OrderBy extension** metodunu nasıl kullandığımıza dikkat etmenizi rica ederim. İşte bu vakaya ait örnek ekran çıktıları.

Londra' daki müşterilerin CustomerId bilgilerini göre sıralı olarak çekilmesi



ID	Title	Contact	Company	City
AROUT	Sales Representative	Thomas Hardy	Around the Horn	London
BSBEV	Sales Representative	Victoria Ashworth	B's Beverages	London
CONSH	Sales Representative	Elizabeth Brown	Consolidated Holdings	London
EASTC	Sales Agent	Ann Devon	Eastern Connection	London
NORTS	Sales Associate	Simon Crowther	North/South	London
SEVES	Sales Manager	Hari Kumar	Seven Seas Imports	London

Londra' daki müşterilerin ContactName bilgisine göre sıralı olarak çekilmesi



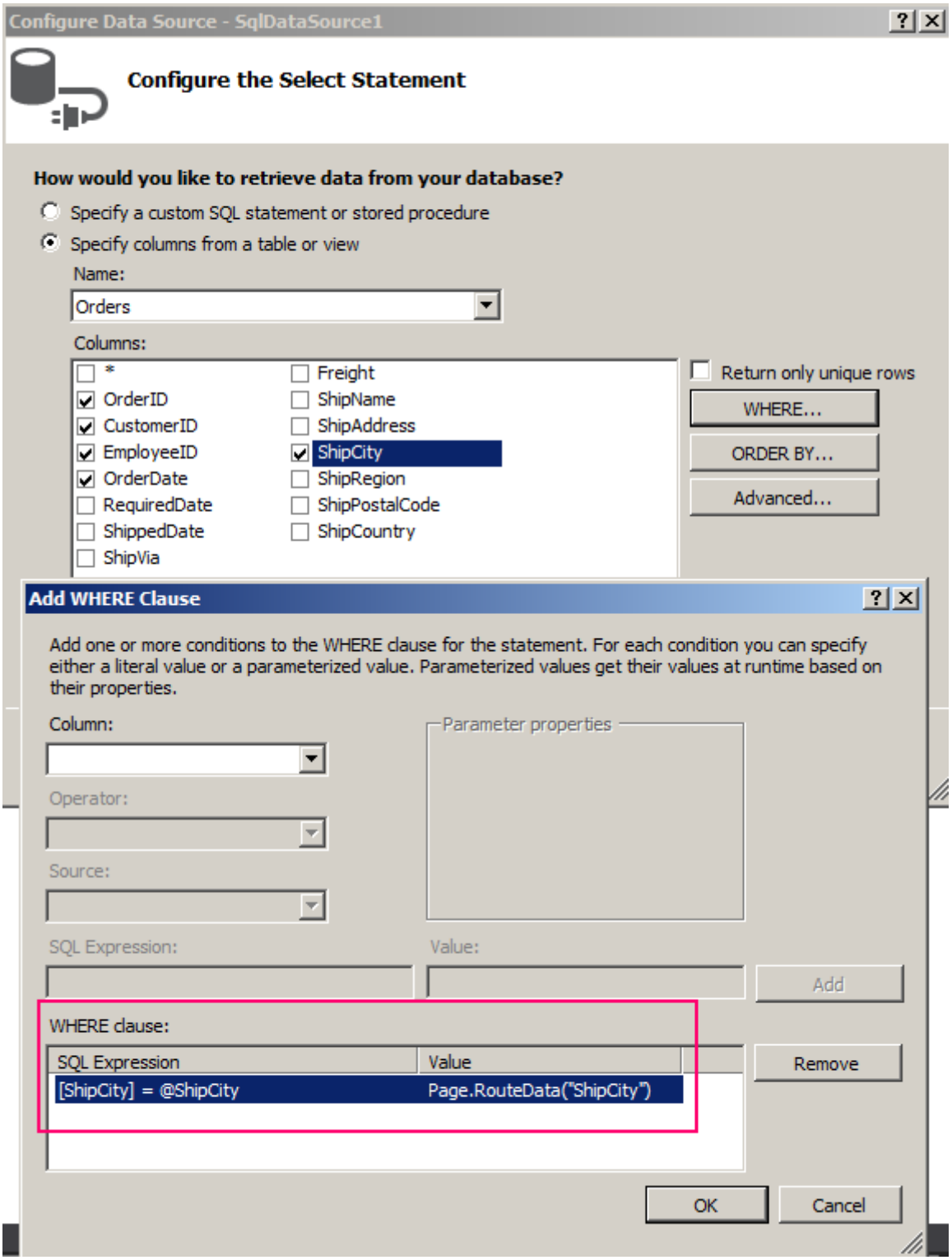
Dördüncü Durum

Son vakada bir **Route** parametrenin her hangi bir veri bağlı kontrol ile nasıl ilişkilendirilebileceğini görmeye çalışacağız. Örneğin bir **SqlDataSource** bileşenindeki **Select** sorgusuna ait **Where**koşullarını **Route** parametreler ile ilişkilendirebiliriz. Bu durumu **siparis.aspx** sayfası içerisinde ele almaya çalışalım. **Siparis** sayfasına gelinebilmesi için **Route** tablo tanımlamalarına göre**/siparisler/{ShipCity}** şeklinde bir **URL** talebinin gönderilmesi gerekmektedir.

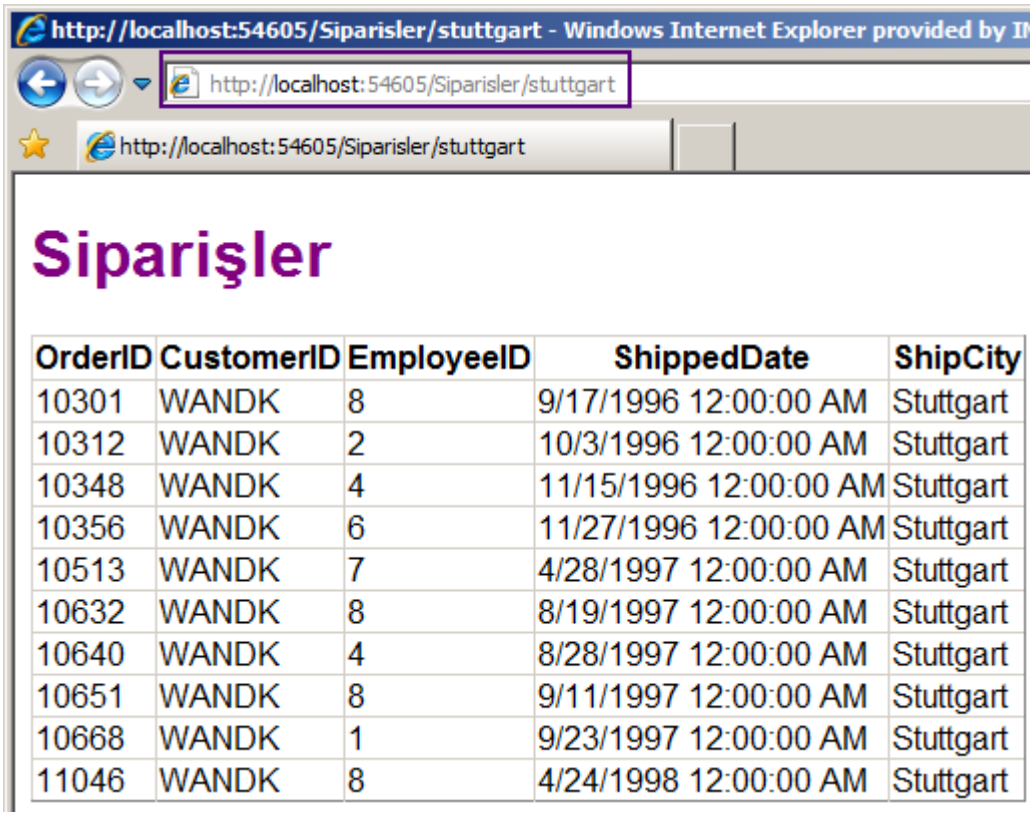
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Siparis.aspx.cs"
Inherits="HowTo_EasyRouting.Siparis" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1 style="color:purple">Siparişler</h1>
      <asp:GridView ID="grdOrders" runat="server" AllowPaging="True"
AutoGenerateColumns="False" DataKeyNames="OrderID"
DataSourceID="SqlDataSource1" >
         <Columns>
            <asp:BoundField DataField="OrderID" HeaderText="OrderID"
InsertVisible="False" ReadOnly="True" SortExpression="OrderID" />
```

```
            <asp:BoundField DataField="CustomerID" HeaderText="CustomerID"
SortExpression="CustomerID" />
            <asp:BoundField DataField="EmployeeID" HeaderText="EmployeeID"
SortExpression="EmployeeID" />
            <asp:BoundField DataField="ShippedDate" HeaderText="ShippedDate"
SortExpression="ShippedDate" />
            <asp:BoundField DataField="ShipCity" HeaderText="ShipCity"
SortExpression="ShipCity" />
        </Columns>
      </asp:GridView>
      <asp:SqlDataSource ID="SqlDataSource1" runat="server"
        ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
        SelectCommand="SELECT [OrderID], [CustomerID], [EmployeeID],
[ShippedDate], [ShipCity] FROM [Orders] WHERE ([ShipCity] = @ShipCity)">
        <SelectParameters>
          <asp:RouteParameter DefaultValue="" Name="ShipCity"
RouteKey="ShipCity" Type="String" />
        </SelectParameters>
      </asp:SqlDataSource>
    </div>
    </form>
</body>
</html>
```

Önemli olan, **SqlDataSource** bileşenine ait **SelectCommand** ifadesindeki **where** koşulunda yer alan **ShipCity** isimli parametrenin bir **RouteParameter** ile ilişkilendirilmiş olmasıdır.**RouteParameter** bileşenine ait **RouteKey** özelliği, **Route Table**' daki ile aynı olmalıdır. Çok doğal olarak **aspx** kaynak tarafında yapılabilen bu eşleştirme, **Wizard** üzerinden de kolayca belirlenebilir. Aynen aşağıdaki ekran görüntüsünde olduğu gibi



Buna göre örneğin Stuttgart' a yapılan sevkiyatları aşağıdaki gibi elde edebiliriz.



Sonuç

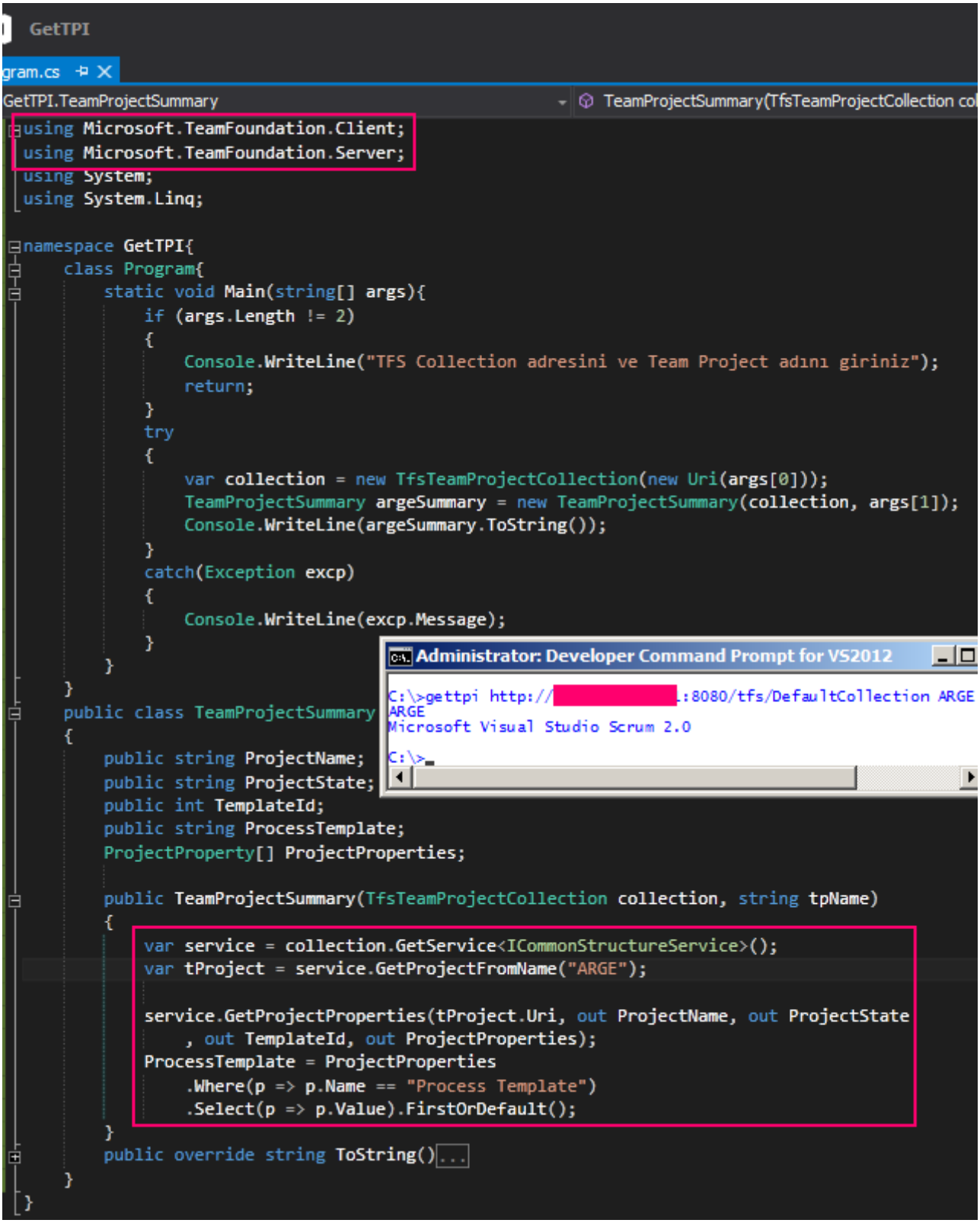
Görüldüğü üzere **URL** eşleştirme işlemleri klasik sunucu tabanlı **Asp.Net** uygulamalarında da etkili bir şekilde kullanılabilir. Hatta bu felsefeden yola çıkarak **OData** sorgularının daha fazla gelişmişlerini destekleyecek web uygulamaları yazılması da pekala olasıdır. Yazımıza konu olan basit örneklerimizde ki anahtar noktalar, **RouteTable** sınıfı, **RouteData.Values** özelliği ve **GetRouteUrl**metodudur. Örneği geliştirmek tamamen sizin elinizde. İşe **/siparisler/stuttgart/10301/**şeklinde bir sorguyu ele alıp, **10301** numaralı siparişe ait detay bilgileri göstereceğiniz bir sayfayı üretmeyi deneyerek başlayabilirsiniz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HowTo_EasyRouting.zip (605,23 kb)

Tek Fotoluk İpucu 101–Team Project Process Template

Salı, 18 Haziran 2013 09:30 by bsenyurt

Merhaba Arkadaşlar,
Bazı komut satırı araçları oldukça işlevseldir ve çoğunlukla tercih edilir.
Örneğin **TFS** üzerinde bir**Team Project** silinmek istendiğinde, **tfsdeleteproject** komut satırı aracına başvrulur. Peki şirketinizde kurulu
olan **TFS** üzerindeki **Team Project** örneklerinin kullandıkları **ProcessTemplate'** leri yine bir komut satırı aracı ile öğrenmek isteseydiniz, nasıl bir yol izlerdiniz? Aşağıdaki fotoğraf size bir ipucu verebilir belki de

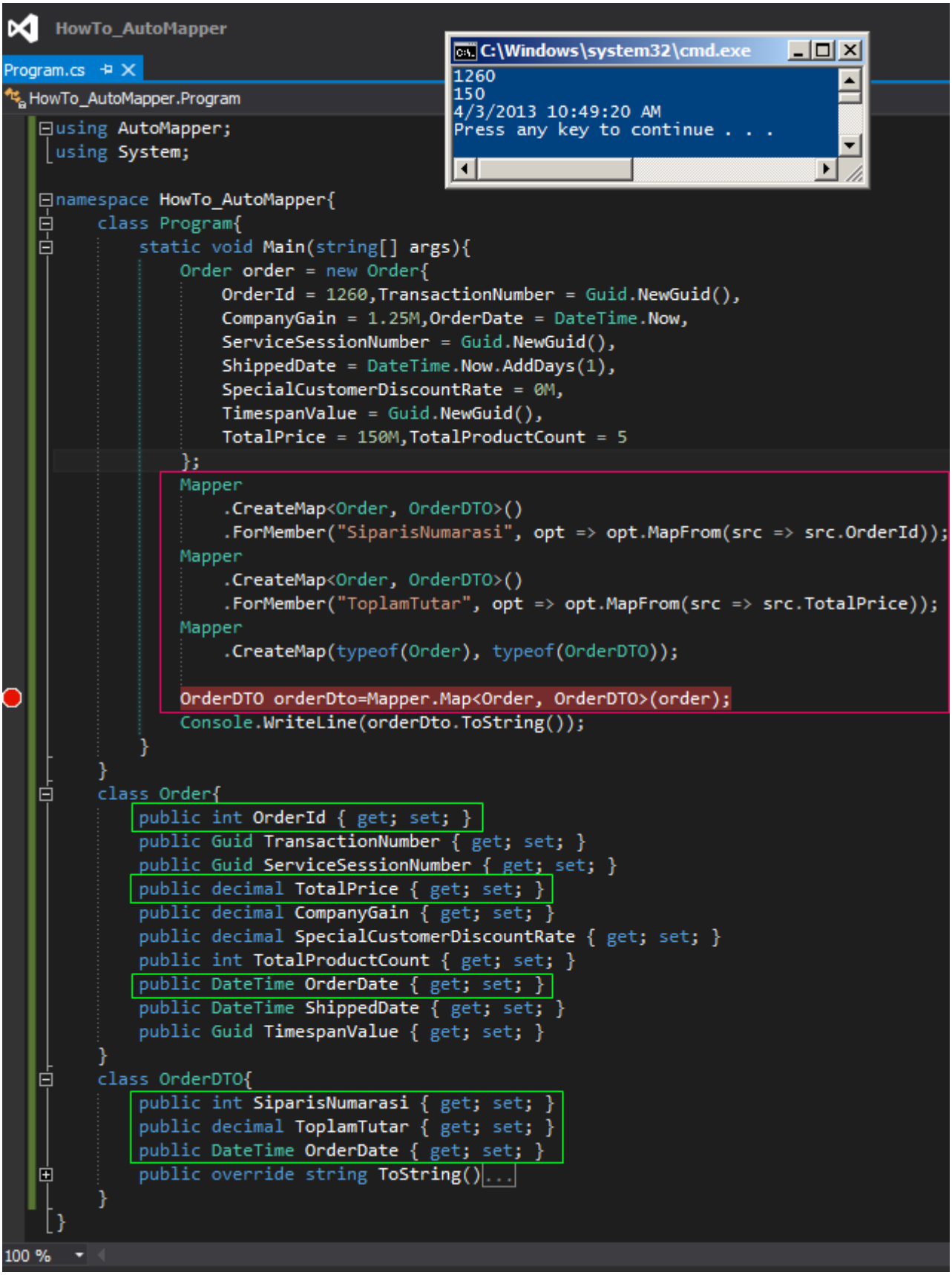


Bu örnek farklı fikirlerinde doğmasına yol açabilir. Örneğin **TFS** sunucusunda ne kadar **TeamProject** varsa, her birinin **Process Template** bilgisini gösteren bir **Windows** uygulaması*(hatta Web' de olur)* geliştirmeyi deneyebilirsiniz 😉

Tek Fotoluk İpucu 100–AutoMapper Kullanımı

Pazartesi, 17 Haziran 2013 09:45 by bsenyurt

Merhaba Arkadaşlar,
Bildiğiniz üzere şu yazımızda nesneler arası **özellik*(Property)*** eşleştirmelerinin nasıl yapılabileceğini incelemeye çalışmıştık. Ancak işin çok daha profesyonel bir boyutu var. Örneğin tipler arası özellik adları birbirlerinden farklı olabilir ve bu nedenle bir haritayı önceden söylemeniz gerekebilir. Neyseki **NuGet** üzerinden yayınlanan **AutoMapper** kütüphanesi çok gelişmiş özellikleri ile buna imkan vermektedir. Söz gelimi aşağıdaki fotoğraf özellik adlarının farklı olması halinde bile**AutoMapper** ile başarılı bir şekilde eşleştirme yapılabileceğini göstermektedir.



Denemeden önce en azından **install-package AutoMapper** komutunu **Package Manager Console**' dan çalıştırıp ilgili kütüphaneyi yüklemeyi unutmayınız.

Asp.Net 4.5 için Yeni Nesil Doğrulama(Validation)

Çarşamba, 12 Haziran 2013 14:09 by bsenyurt

Merhaba Arkadaşlar,
Bilişim sektöründe yer alan ve özellikle 70li yıllarda doğanların neredeyse tamamı bu efsane cümleyi bilir.
A long tim ago in a galaxy far, far away…
Hikaye hep yazılı bir anlatım ile başlar ve daha sonra ekran yukarıdaki yıldızlardan aşağıya doğru inerek devam eder. Sornasında ya bir uzay mekiğinin kaçış sahnesi ya da imparatorluk güçleri ile isyancılar arasındaki savaşla karşı karşıya kalırız.
İşte geçen gün özlediğim **Star Wars** serilerinden birisini izlerken bir den kendimi bilgisayarımın başında ve başka bir hikayenin giriş noktasında buldum. Karşımdaki ekran da karalara bağlamış, alacalı bulacalı bir geliştirme penceresi duruyordu. İçerisinde ise **HTML** ve**Asp.Net** karışımı bir şeyler…

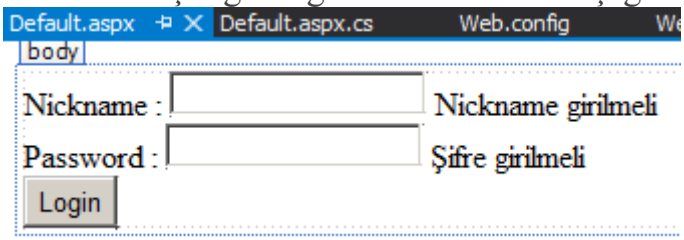
Hikayenin Başı

Her şey uzun bir zaman önce değil ama kısa bir süre önce **Asp.Net 4.5** tabanlı bir **Empty Web Application** açmamla başlamış ve sonrasında olanlar olmuştu Aslında senaryo gereği çok basit olarak bir web form üzerinde doğrulama kontrollerini kullanacaktım. Bunun için **Visual Studio 2012** ortamında **Asp.Net Empty Web Application** tipinden bir proje oluşturdum ve aşağıdaki**Web Form** içeriğini tasarladım.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="WebApplication2.Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>

      Nickname :
      <asp:TextBox ID="txtNickname" runat="server"></asp:TextBox>
 <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
runat="server"ControlToValidate="txtNickname"
ErrorMessage="Nickname girilmeli"></asp:RequiredFieldValidator>
```

```
        <br />
        Password :
        <asp:TextBox ID="txtPassword" runat="server"></asp:TextBox>
 <asp:RequiredFieldValidator ID="RequiredFieldValidator2"
runat="server"ControlToValidate="txtPassword" ErrorMessage="Şifre
girilmeli"></asp:RequiredFieldValidator>
        <br />
        <asp:Button ID="btnLogin" runat="server" OnClick="btnLogin_Click" Text="Login"
/>
    </div>
    </form>
</body>
</html>
```

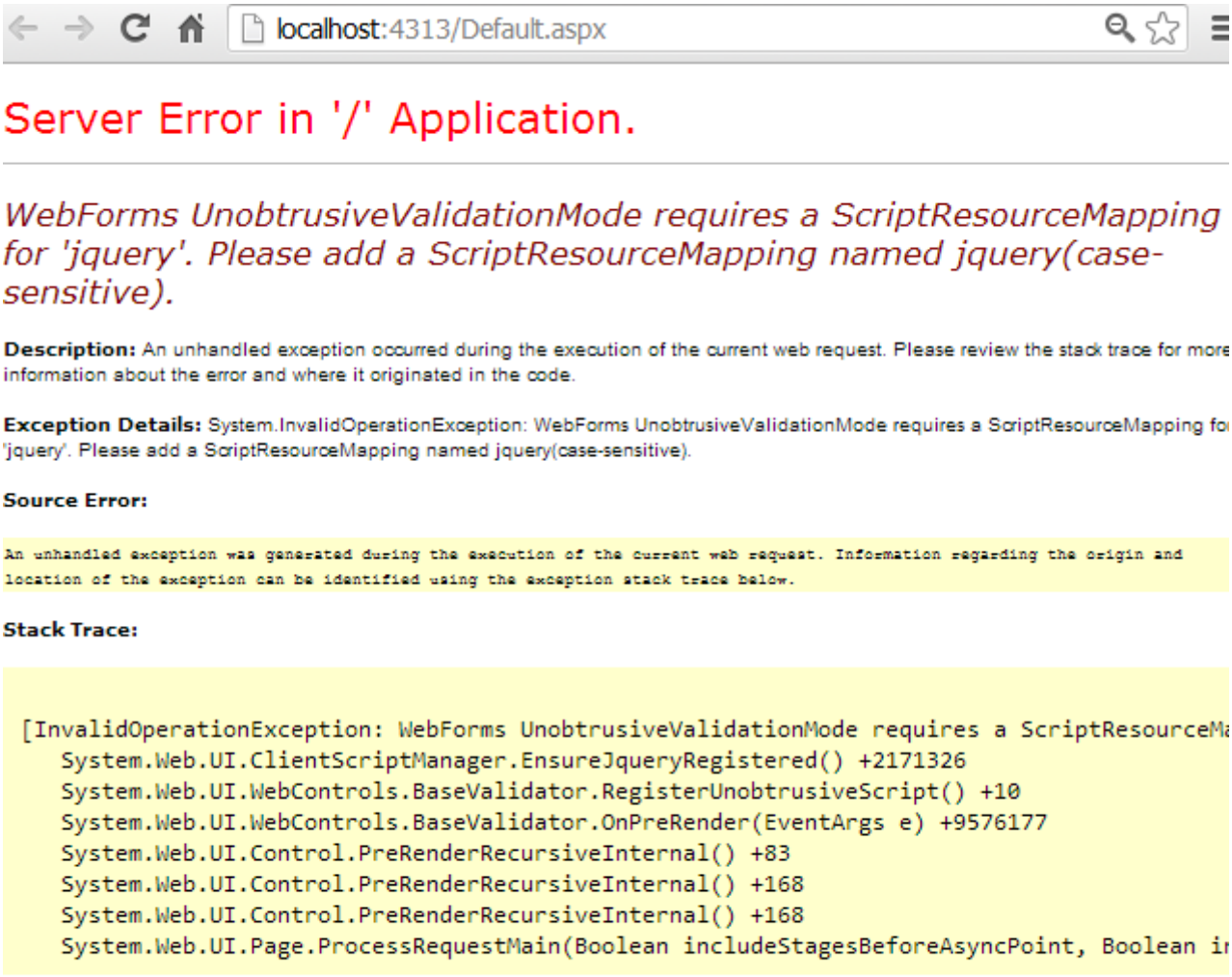
Web form içeriğinin görsel tasarımı ise aşağıdaki ekran görüntüsündekine benzemişti.



Ekranın görevi oldukça basitti. Kullanıcıdan **Nickname** ve **Password** bilgisi ile giriş yapması isteniyordu. Eğer bu **TextBox** kontrollerinin içeriği boş bırakılırsa da **RequiredFiledValidator**kontrolleri devreye girerek kullanıcıyı uyarmaktaydı.

İlk Çalışma

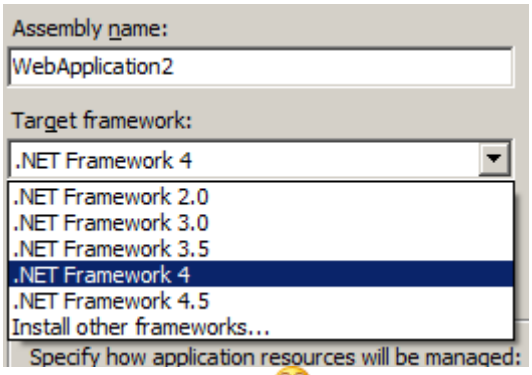
Herşey bana göre son derece normaldi ancak çalışma zamanı böyle demiyordu. Sonuç aşağıdaki ekran görüntüsünde ki gibi olmuştu



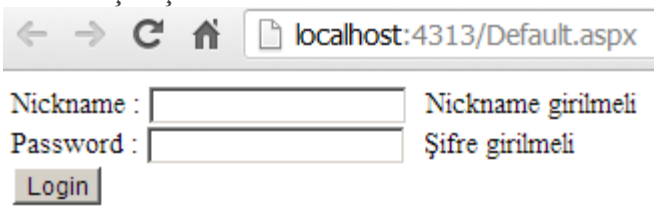
Tabi ilk dikkatimi çeken nokta **Unobtrusive** olarak yazılan ve telafüz etmesini halen daha başaramadığım kelime idi. **"Dikkati çekmeyen"**, **"mütevazi"**, **"kendi halinde"**, **"fark edilmeyen"**, **"kolay görülmeyen"** gibi Türkçe karşılıkları olan kelimenin **Asp.Net** açısından önemini araştırırken bakın neler buldum

Peki neden böyle oldu?

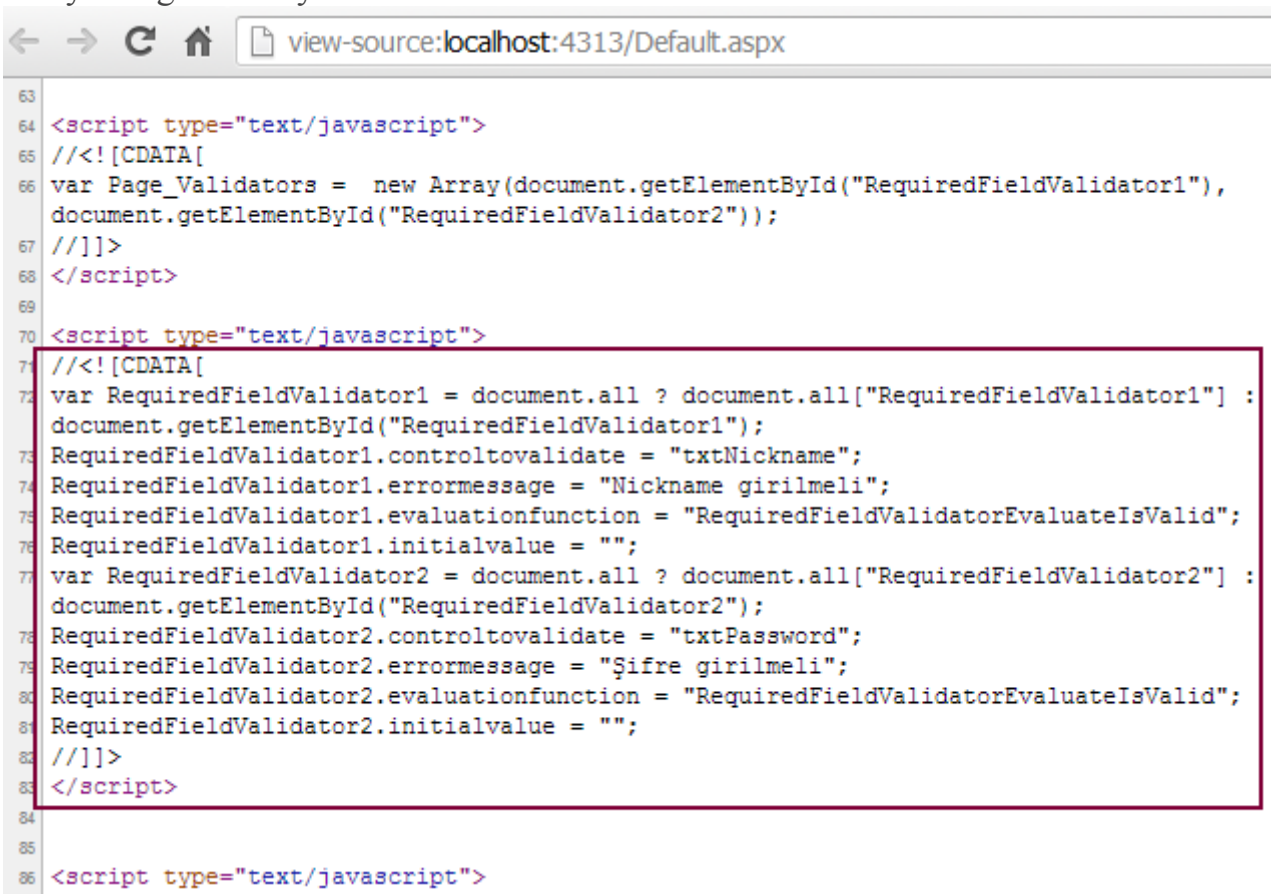
İlk olarak zamanı geriye alıp **Asp.Net 4.5** öncesine bakmamız gerekiyor*(Zamanı geriye almak keşke bir **Framework** değişikliği kadar kolay olsa değil mi?)* Bu nedenle projeyi **.Net Framework 4.0**odaklı hale getirip yeniden derledim.



ve tabi çalıştırdım



Çalışma zamanında hiç bir sorun yoktu. Her şey yolunda görünmekteydi. **Doğrulama*(Validation)***kontrolleri de devreye girmiş durumdaydı. Sayfanın istemci tarafına gönderilen kaynak içeriğine bakıldığında doğrulama işlemleri için üretilmiş bir takım **Javascript** kod parçaları olduğu ve **CDATA**olarak entegre edildiği kolaylıkla görülebiliyordu.

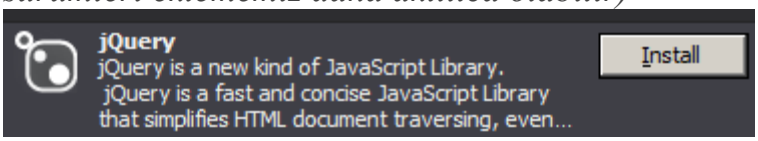


Bilindiği üzere bu sayede istemci tarafı sunucuya gönderilmeden de doğrulama işlemleri yerine getirilebilmektedir.

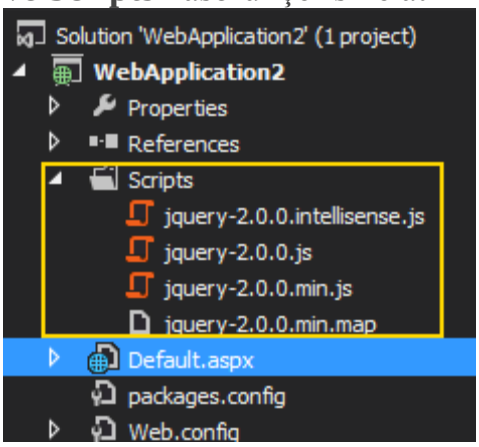
Web tarafında doğrulama işlemleri istemci tarafında başlamakta ama sunucu tarafında da bir kontrol yapılmaktadır. Bunun en büyük sebebi istemci tarafının Javascript yürütme gibi bir desteği olmaması halinde karşı tedbir alınmak istenmesidir.
Asp.Net 4.5 ile birlikte ise daha önceden kullanılan **javascript** odaklı sistem yerine, varsayılan olarak **HTML 5**' in **data-val-controltovalidate**, **data-val-errormessage**, **data-val**, **data-val-evaluationfunction**, **data-val-initialvalue** gibi **nitelikleri*(attribute)*** ve **jQuery** kütüphanesi ele alınmaktadır.
Dolayısıla **Asp.Net 4.5** tipinden bir **Empty Web Application** söz konusu olduğunda ve doğrulama işlemlerini uygulamak istediğimizde, bazı ayarlamaları yapmamız söz konusudur.*(Bu işlemlere **Web Forms** tipinden bir **Asp.Net** uygulaması açtığınızda ihtiyaç duymayabilirsiniz)*

Adımlar

Tekrar **Target Framework**' ü **.Net Framework 4.5**' e çekelim. Sonrasında ise **UnobtrusiveValidation** akışı için ihtiyacımız olan **jQuery** kütüphanelerini **NuGet** paket yönetim aracı ile projemize dahil edelim. *(Son sürümleri eklememiz daha akıllıca olabilir)*



Bu **install** işlemi sonrası ihtiyacımız olan **jQuery** kütüphaneleri projeye ilave edilmiş ve **Scripts**klasörü içerisine atılmış olacaktır.



ScriptResourceDefinition Bildirimleri

Sonrasında tek yapılması gereken uygulamanın başlatıldığı bir yerde doğrulama işlemleri için gerekli**path** tanımlamalarının **ScriptResourceDefinition** sınıfı için yapılmasıdır. En uygun yer**global.asax.cs** içerisindeki **Application_Start** olay metodudur*(**global.asax** varsayılan olarak bu proje şablonunda yer almamaktadır. Bir*

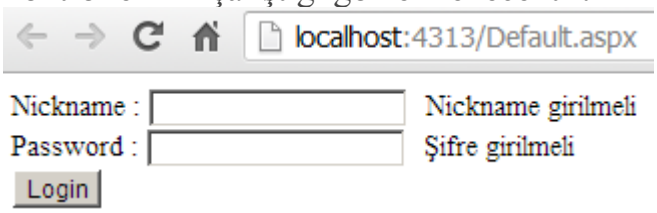
başka deyişle ilave etmeniz gerekmektedir) Bu metod içeriğini aşağıdaki gibi düzenleyerek devam edelim.

```
using System;
using System.Web.UI;
namespace WebApplication2
{
   public class Global : System.Web.HttpApplication
   {
      protected void Application_Start(object sender, EventArgs e)
      {
         ScriptResourceDefinition jQuery = new ScriptResourceDefinition();
        jQuery.Path = "~/scripts/jquery-2.0.0.min.js";
         jQuery.DebugPath = "~/scripts/jquery-2.0.0.js";
         jQuery.CdnPath = "http://ajax.microsoft.com/ajax/jQuery/jquery-2.0.0.min.js";
         jQuery.CdnDebugPath = "http://ajax.microsoft.com/ajax/jQuery/jquery-2.0.0.js";
         ScriptManager.ScriptResourceMapping.AddDefinition("jquery", jQuery);
      }
…
}
```

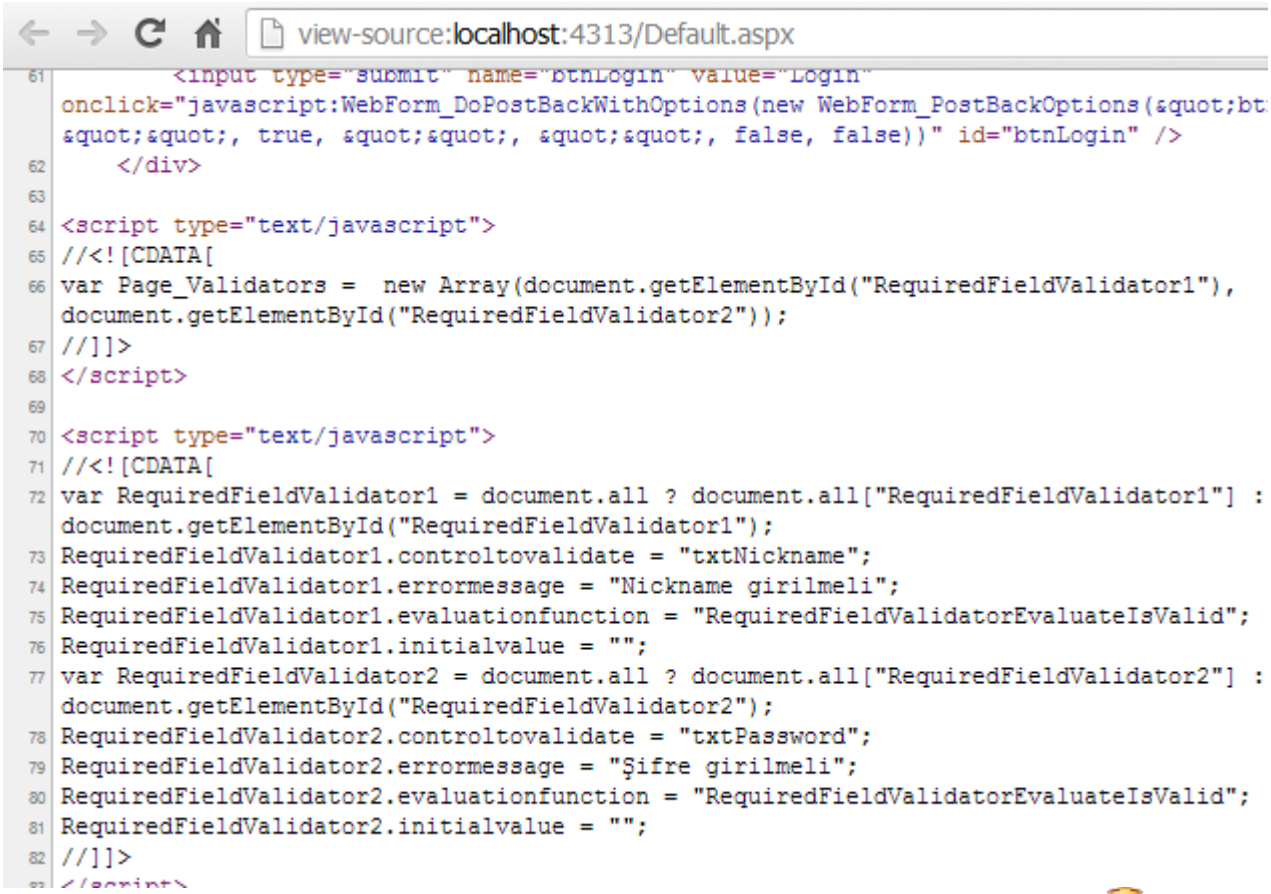
Bir takım path tanımlamaları yapıldığı görülmektedir. **jQuery** için yapılan tanımlamalar haricinde**Content Delivery Network*(CDN)*** için de bazı **path** bildirimleri belirtilmiştir. Senaryo da,**Microsoft AJAX CDN'** leri kullanılmaktadır. İlgili path tanımlamaları **ScriptResourceDefinition**sınıf örneği için yapıldıktan sonra, ilgili nesne örneğinin **ScriptResourceMapping** özelliğine eklenmesi yeterlidir.

Yeni Bir Test

Uygulamamızı tekrar çalıştırıp test edelim. Bir hata oluşmayacak ve doğrulama kontrollerinin çalıştığı gözlemlenecektir.



Tabi ki bizim için daha önemli olan istemci tarafına giden kod içeriğidir. Eğer kaynak koda bakarsak aşağıdaki sonuçlarla karşılaşırız.



Hımmm bir terslik var gibi. Sanki buralar da **HTML 5**' den hiç bir eser yok
Bu son derece doğal çünkü **Web Form**' lar için bu yeni stil doğrulama işleminin yapılacağını bir yerler de belirtmemiz gerekiyor. **web.config** dosyası tahmin edileceği üzere en uygun yer ve içeriğini aşağıdaki gibi değiştirmemiz bu senaryo için yeterli.

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.5"/>
    <httpRuntime/>
    <pages controlRenderingCompatibilityVersion="4.0"/>
  </system.web>
  <appSettings>
    <add key="ValidationSettings:UnobtrusiveValidationMode" value="WebForms"
/>
  </appSettings>
</configuration>
```

appSettings sekmesinde yer alan **ValidationSettings:UnobtrusiveValidationMode key** değeri,**Asp.Net** çalışma zamanı için anlamlıdır. **Value** niteliğine **WebForms** dışında **None** değeri de verilebilmektedir. **None** değerinin verilmesi halinde tahmin edileceği üzere eski stil

doğrulama sürecine geçilmektedir. Şu anki haliyle de yeni stilin kullanılacağı belirtilmektedir.

Bir Test Daha

Öyleyse uygulama tekrardan çalıştırılır ve istemci tarafına giden kaynak kod içeriğine bakılır.

```
    <div>

        Nickname :
        <input name="txtNickname" type="text" id="txtNickname" />
 <span data-val-controltovalidate="txtNickname" data-val-errormessage="Nickname girilmeli"
id="RequiredFieldValidator1" data-val="true" data-val-
evaluationfunction="RequiredFieldValidatorEvaluateIsValid" data-val-initialvalue=""
style="visibility:hidden;">Nickname girilmeli</span>
        <br />
        Password :
        <input name="txtPassword" type="text" id="txtPassword" />
 <span data-val-controltovalidate="txtPassword" data-val-errormessage="Şifre girilmeli"
id="RequiredFieldValidator2" data-val="true" data-val-
evaluationfunction="RequiredFieldValidatorEvaluateIsValid" data-val-initialvalue=""
style="visibility:hidden;">Şifre girilmeli</span>
        <br />
        <input type="submit" name="btnLogin" value="Login"
onclick="javascript:WebForm_DoPostBackWithOptions(new WebForm_PostBackOptions(&quot;btnLogin&quot;,
&quot;&quot;, true, &quot;&quot;, &quot;&quot;, false, false))" id="btnLogin" />
    </div>
    </form>
</body>
</html>
```

İstediğimiz olmuştur ve istemci tarafındaki doğrulama işlemleri için **HTML 5** nitelikleri devreye girmiştir. Kontrolün boş geçilmemesi için **data-val-evaluationfunction** niteliğinin değeri ele alınmaktadır. Hangi kontrolün denetleneceği bilgisi için **data-val-controltovalidate** niteliği kullanılmaktadır. Hata mesajı ise **data-val-errormessage** ile belirtilir vb…

Size tavsiyem diğer doğrulama kontrollerini de işin içine katarak senaryoyu genişletmeniz ve özellikle data-val-evaluationfunction değerlerinin nasıl üretildiğine bakmanız yönünde olacaktır.

Kıssadan Hisse

Yeni nesil doğrulama stratejisi için son teknoloji ürünüdür diyebilir miyiz acaba? Bence evet  Hem**HTML 5**' e hem de **jQuery**' ye yatırım yapmış bir doğrulama süreci söz konusu. Üstelik yeni nesil çıktılar da, doğrulama operasyonları adına **Javascript** kullanımı*(CDATA içerisindeki kısımlar)* mevcut değil. Doğrulama bilgisi tamamen **HTML 5** içerisine, **nitelik-değer*(key-value)*** bazlı olarak yıkılmış durumda. Elbette bunun en büyük artısı **sayfa cevap boyutunun*(Page Response Size)***küçülmüş olması. *(Elbette tarayıcı desteği de önem arz eden bir konu)*

Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

C#' ın Enteresan Yanları

Pazartesi, 10 Haziran 2013 08:44 by bsenyurt

[İlk yazım tarihi 31 ekim 2012]

Merhaba Arkadaşlar,

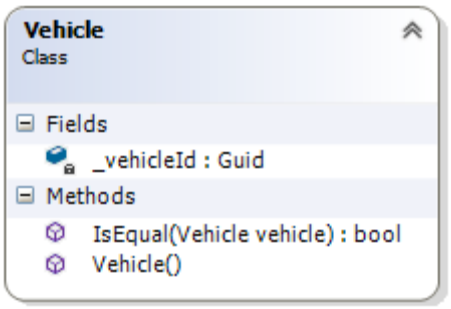
Yazılım sektöründe yer alan bizler, mutlak suretle en az bir programlama dilini çok iyi seviyede öğrenmeye çalışır ve bunun için epey yoğun çaba sarf ederiz*(Hatta değerli bir büyüğümüzün sözüne göre, hayatımızın her hangibir noktasında C veya C++ gibi bir dili öğrenmeye çalışmış ama hiç bir zaman iyi bir C/C++ geliştiricisi olamamışızdır)* Ne varki bazen dilin kullanılmayan pek çok özelliğini, zamanında öğrenmiş olsak dahi unutabiliriz. Hatta bazı ilginç olan yanlarını bugüne kadar hiç görmemiş, denememiş ya da duymamış olabiliriz. İşte size C# dili ile ilişkili olarak pek çoğumuzun hatırından giden bir kaç enteresan vaka…

Bu arada C# diline ait geniş bir dökümantasyonu C:\Program Files\Microsoft Visual Studio 11.0\VC#\Specifications\1033 klasörü altında bulabilirsiniz 527 sayfalık CSharp Language Specification isimli bu döküman, elinizin altındaydı her zaman. Başka bir kitaba ihtiyacınız yok. En azından başlangıç seviyesinde. Visual Studio 2012 kurulumu sonrası gelen bu dökümantasyon aslında C# 3.0 sürümünden beri de mevcut.

Bu yazımızda örnek 5 vaka çalışması üzerinde duruyor olacağız.

Vaka 1 – private olarak tanımlanmış bir alana(field), tanımlandığı sınıf dışındanerişilemez.

Hep bu şekilde öğrendik. Genel olarak cümle kalıbı böyleydi. C# tarafından baktığımızda temel olarak 5 **erişim belirleyicisi***(Access Modifier)* olduğunu biliyoruz. **private, public, internal, protected** ve son olarak da **protected internal**. **private** olarak tanımlanış üyelerin de*(members)*, tanımlı oldukları yer dışından erişilemez olduklarını biliyoruz. Tabi istisnai durumlarda yok değil. Söz gelimi aşağıdaki kod parçasını göz önüne alalım.



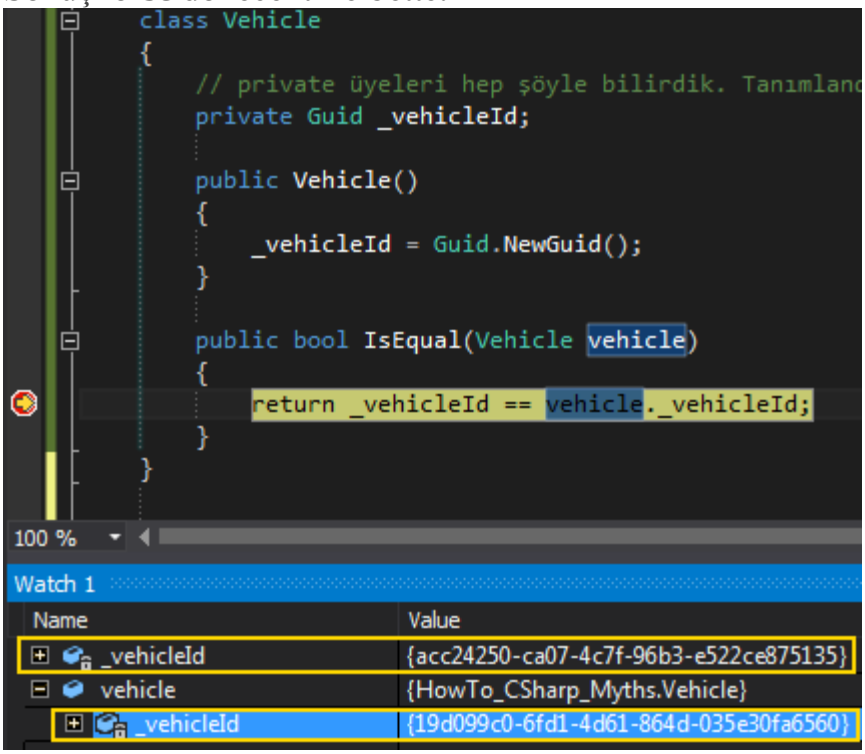

```
class Vehicle
{
  private Guid _vehicleId;
   public Vehicle()
   {
```

```
        _vehicleId = Guid.NewGuid();
    }
    public bool IsEqual(Vehicle vehicle)
    {
        return _vehicleId == vehicle._vehicleId;
    }
}
```

Vehicle sınıfı içerisinde tanımlanmış olan **_vehicleId** alanı **private** erişim belirleyicisi ile işaretlenmiştir. Ancak dikkat edilmesi gereken bir nokta vardır. Sınıf içerisinde tanımlanan **IsEqual**metodu parametre olarak başka bir **Vehicle** nesne örneğini almakta ve içerisinde bu örneğe ait **_vehicleId** alanını kullanmaktadır. Kullanabilmektedir. Derleyici buna kızmamaktadır. Peki uygulama tarafına geçelim ve aşağıdaki test kodlarını değerlendirelim.

```
static void Main(string[] args)
{
    #region Case 1 Test (Private Fields)
    Vehicle v1 = new Vehicle();
    Vehicle v2 = new Vehicle();
    Console.WriteLine(v1.IsEqual(v2));
    #endregion
}
```

Sonuç **false** dönecektir elbette.

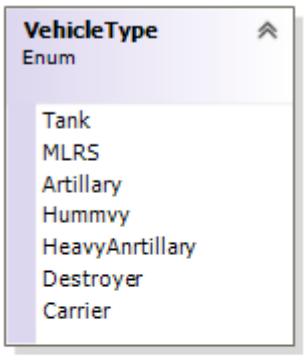


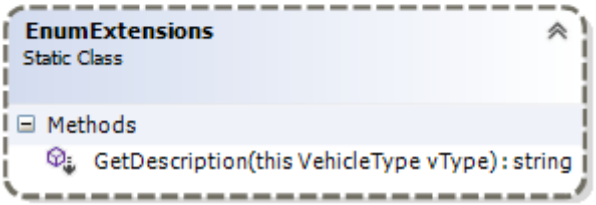
Dikkat edileceği üzere parametre olarak gelen **vehicle** değişkeni üzerinden, **private** olarak tanımlanmış **_vehicleId** alanına erişilebilmiştir. Tabi **_vehicleId'** nin değeri, **v2** isimli değişkene ait olarak üretilen **Guid** değeridir.

Dolayısıyla private alan kullanımları ile ilişkili olarak şunu da ifade edebiliriz. private tanımlanmış bir üyeye tanımlandığı sınıfa ait başka nesne örnekleri(Instance) üzerinden erişilebilinir.

Vaka 2 – Çok yerde faydasını gördüğümüz genişletme metodları(Extension Methods), Enum sabitlerine de uygulanabilir.

Genişletme metodları*(Extension Methods)* özellikle elimize kodları kapalı olarak gelen **assembly**dosyaları düşünüldüğünde, bunları ek fonksiyonellikler ile genişletmede kullanılan önemli kavramlardan birisidir. Çoğunlukla türetilemeyen veya az önce de bahsettiğimiz gibi kodları kapalı gelen sınıflar için kullanıldığına sıklıkla şahit oluru. *(Extension Method' lar ile ilişkili bir internet sitesi dahi vardır*  *)* Ancak bu özelliğin **Enum** sabitleri için de kullanılabildiğini fark etmiş miydiniz? Örneğin,






```
enum VehicleType
  {
     Tank,
     MLRS,
     Artillary,
     Hummvy,
     HeavyAnrtillary,
     Destroyer,
     Carrier
  }
  static class EnumExtensions
  {
     public static string GetDescription(this VehicleType vType)
     {
        string result = String.Empty;
        switch (vType)
        {
```

```
            case VehicleType.Tank:
                result= "Paletli zırhlı tank. 135mm top";
                break;
            case VehicleType.MLRS:
                result = "Kundağı motorlu çoklu roket atar sistemi";
                break;
            case VehicleType.Artillary:
                result = "75mm - 205mm arası hafif topçu";
                break;
            case VehicleType.Hummvy:
                result = "Hummer jeep";
                break;
            case VehicleType.HeavyAnrtillary:
                result = "205mm üstü ağır kara topçusu";
                break;
            case VehicleType.Destroyer:
                result = "Güneş sınıfı yeni nesil zırhlı destroyer";
                break;
            case VehicleType.Carrier:
                result = "Nimitz sınıfı nükleer Uçak gemisi";
                break;
        }
        return result;
    }
}
```

VehicleType enum sabiti için **EnumExtensions** sınıfı içerisinde **GetDescription** isimli bir genişletme metodu tanımlanmıştır. Bu metod, **enum** sabiti ile ilişkili bir açıklamayı geri döndürecek şekilde tasarlanmıştır. *(İlk parametre pek tabi this anahtar kelimesi ile başlamalıdır)*

Enum sabitinin kodlarının kapalı bir **Assembly** içerisinde olduğunu farz edeceğimiz bir senaryoda, bu tip bir yaklaşım önemli esneklikler sağlayacaktır. Örneğin uygulanmasında da, bildiğimiz sınıf bazlı extension metodların kullanımından farklı bir yaklaşım söz konusu değildir.

```
#region Case 2 Test (Enum Extensions)

VehicleType vType = VehicleType.Artillary;
Console.WriteLine(vType.GetDescription());
vType = VehicleType.HeavyAnrtillary;
Console.WriteLine(vType.GetDescription());

                        (extension) string VehicleType.GetDescription()
#endregion
```

```
C:\Windows\system32\cmd.exe
75mm - 205mm arası hafif topçu
205mm üstü ağır kara topçusu
Press any key to continue . . .
```

Vaka 3 – static alanların tanımlanma sıraları önemli midir?

Bunu yazdığımızda göre önemli olsa gerek  Haydi gelin aşağıdaki kod parçasını göz önüne alalım.

```
using System;
using System.Linq;
using System.Collections.Generic;
namespace HowTo_CSharp_Myths
{
   class Program
   {
      #region Case 3 : static alanların sırası önemli midir?
      static double r1 = Math.PI;
      static double r2 = r1;
      #endregion
      static void Main(string[] args)
      {
         #region Case 3 Test (static Field order)
         Console.WriteLine(r2.ToString());
         #endregion
      }
   }
}
```

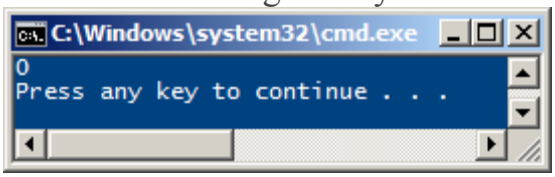
Program sınıfı içerisinde tanımlanmış olan **r1** ve **r2 static** alanlarının şu an pek dikkati çekmeyen sıraları aslında önemlidir. Yukarıdaki kod çalıştırıldığında, tahmin edileceği üzere ekrana **pi** değeri yazılacaktır. Nitekim **r2** tanımlanırken, **r1'** in kendisine atandığı görülmektedir. **r1'** de zaten **Pi**değerini taşımaktadır. İşte çalışma zamanı çıktısı.

```
C:\Windows\system32\cmd.exe
3.14159265358979
Press any key to continue . . .
```

Peki **static** alanların sırasını değiştirirsek?

```
using System;
using System.Linq;
using System.Collections.Generic;
namespace HowTo_CSharp_Myths
{
   class Program
   {
      #region Case 3 : static alanların sırası önemli midir?
      // static double r1 = Math.PI;
      // static double r2 = r1;

      // Sırayı değiştirdik
      static double r2 = r1;
      static double r1 = Math.PI;
      #endregion
      static void Main(string[] args)
      {
         #region Case 3 Test (static Field order)
         Console.WriteLine(r2.ToString());
         #endregion
      }
   }
}
```

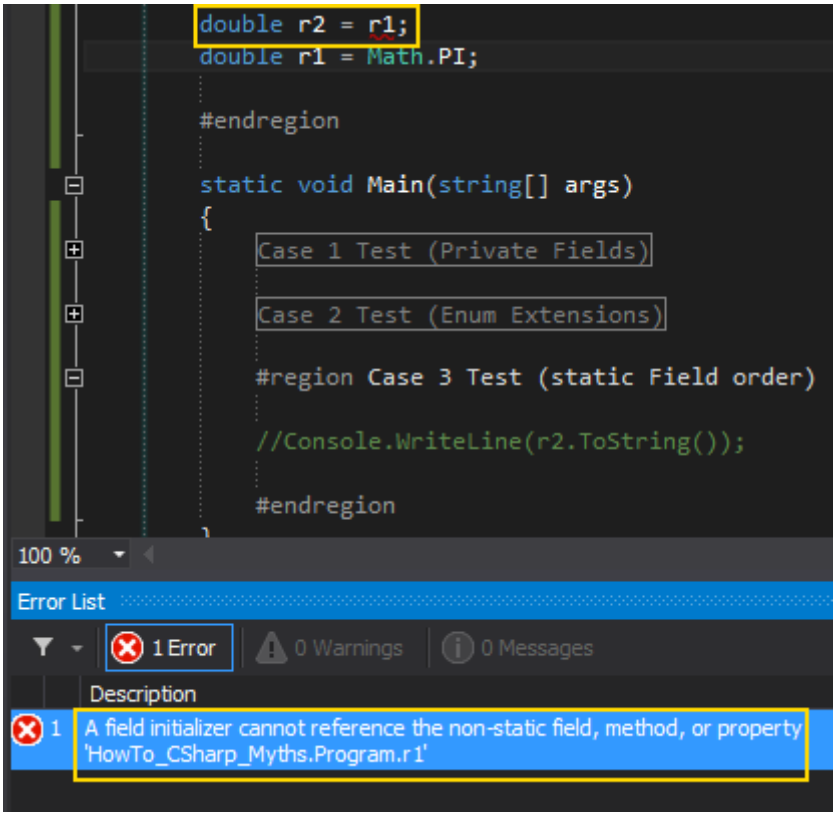
Yine ekrana **Pi** değerinin yazılmasını bekleyebiliriz öyle değil mi? Ama,



sıfır yazmıştır.

Görüldüğü gibi sıralama static alanların kullanıldığı durumda önemlidir. Nitekim **r2** ilk tanımlandığında **r1** değerini alırken, **r1'** in o anki değeri varsayılan **int** için **0**' dır. Dolayısıyla, sonraki sırada yapılan **r1** tanımlanması ve atamasında verilen **Pi** değeri sadece **r1** için söz konusudur.

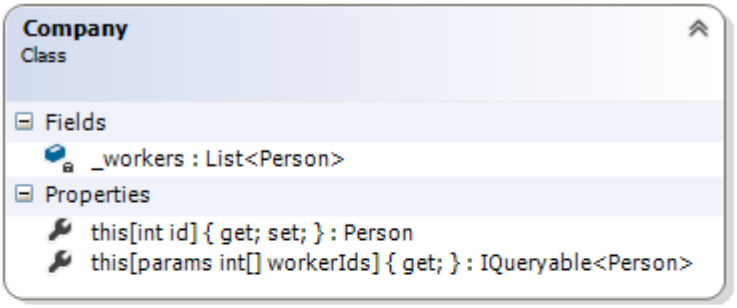
Peki bu durum static olmayan alanlar için de geçerli midir acaba? Bunu denediğimizde sizce ne olur?

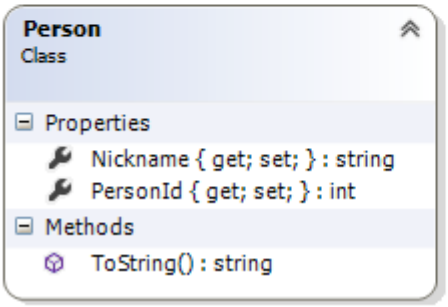


Böyle bir atamaya zaten bu Console uygulaması açısından bakıldığında, derleme zamanı izin vermeyecektir. Sıralamayı değiştirip double r2=r1; ifadesini bir alt satıra geçirseniz dahi durum değişmez.

Vaka 4 – Indeksleyicilerde params anahtar kelimesi de kullanılabilir

Indeksleyiciler*(Indexers)* bildiğiniz üzere sınıf örnekleri üzerinden köşeli parantez operatörünü kullanarak iç üyelere erişimde yardımcı olmaktadırlar. **Özelliklere*(Properties)*** benzer şekilde **get**ve **set** blokları vardır ve çoğunlukla içsel dizi bazlı sınıf üyelerini ele alırlar. Fakat istenirse indeksleyicilerde **params** anahtar sözcüğü de kullanılabilir. Aşağıdaki örneği göz önüne alalım.





ve kod parçamız

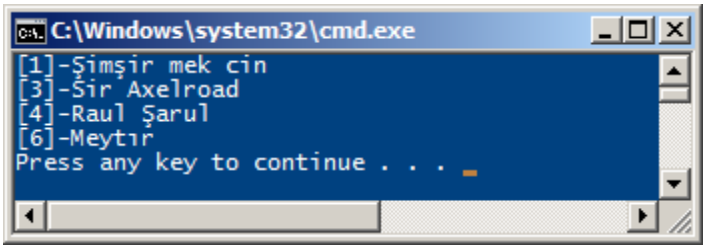
```
class Company
{
   private List<Person> _workers = new List<Person>(8);
   public Person this[int id]
  {
      get {
         return
            _workers
           .Where(p=>p.PersonId==id)
            .FirstOrDefault();
      }
      set {
         _workers.Add(value);
      }
   }
  public IQueryable<Person> this[params int[] workerIds]
   {
      get
     {
        return workerIds
           .Select(id => _workers.Where(p=>p.PersonId==id).FirstOrDefault())
            .AsQueryable();
      }
   }
}
```

```
class Person
{
   public int PersonId { get; set; }
   public string Nickname { get; set; }
   public override string ToString()
   {
      return string.Format("[{0}]-{1}", PersonId.ToString(), Nickname);
   }
}
```

Bu örnekte **Person** tipinden **generic** bir **List** alanının indeksleyiciler ile kullanımına yer verilmiştir. Sadece tek bir **int** parametre alan versiyon, standart bir indeksleyici kullanımını göstermektedir. Diğer yandan **params** anahtar kelimesinin kullanıldığı ikinci versiyon, asıl odaklanacağımız yerdir.**Company** sınıfının örnek kullanımını göz önüne alırsak konuyu daha iyi irdeleyebiliriz.

```
   static void Main(string[] args)
   {
      #region Case 4 (Indeksleyicilerde params)
      Company company = new Company();
      company[0] = new Person { PersonId = 1, Nickname = "Şimşir mek cin" };
      company[1] = new Person { PersonId = 3, Nickname = "Sir Axelroad" };
      company[2] = new Person { PersonId = 4, Nickname = "Raul Şarul" };
      company[3] = new Person { PersonId = 2, Nickname = "William" };
      company[4] = new Person { PersonId = 6, Nickname = "Meytır" };
      var subSet = company[1, 3, 4];
      foreach (var person in subSet)
      {
         Console.WriteLine(person.ToString());
      }
      var singlePerson = company[6]; // params kullanılmayan indeksleyici çağırılır
      Console.WriteLine(singlePerson.ToString());

      #endregion
   }
}
```

Dikkat edileceği üzere **0, 1, 2, 3** ve **4** numaralı indislere farklı **Person** nesne örnekleri atanmıştır.**subSet** değişkeninin elde ediliş şekline dikkat ettiniz mi İşte burada **params** anahtar kelimesinin etkisi görülmektedir. Senaryomuza göre burada **PersoneId** değeleri gönderilmiş ve ona uygun olacak bir sonuç alınmıştır. **singlePerson** değişkeninin elde edilmesi sırasında ise, **params** anahtar kelimesinin kullanılmadığıindeksleyici versiyonu çalışacaktır. İşte uygulamanın çalışma zamanı sonuçları.



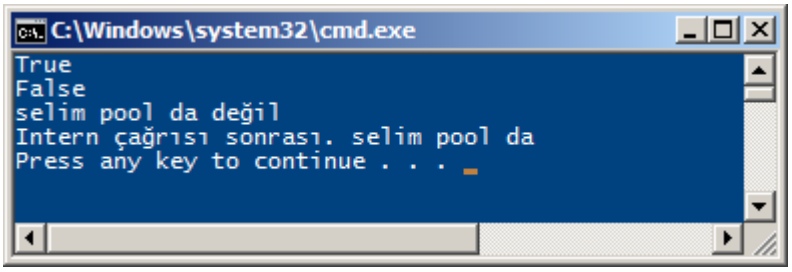
Vaka 5 – Hiç String sınıfının Intern veya IsInterned metodlarını kullandınız mı/duydunuz mu? *(.Net odaklı bir fark ama olsun)*

string değişkenler bilindiği üzere **System.String** sınıfı ile temsil edilirler. **Referans** tipi olan**string** değişkenler esasında maliyetleri zaman zaman yüksek olabilecek örneklerdir. Bazı durumlarda*(ve hatta çok nadiren de olsa)* programlarımızın içerisinde aynı veriyi tutan **string**değişkenlerin **n** sayıda örneğine ihtiyaç duyabiliriz. Milyonlarca string değişkeniniz olduğunu ve her birinin aslında aynı veri içeriğini tuttuğunu düşünün. Bu çok doğal olarak **CLR***(Common Language Runtime)* tarafında önemli bir yük anlamına gelmektedir.

Aslında **string** tipteki değişkenler için **CLR** bu performans handikapını ortadan kaldırmak adına akıllı bir yol izler ve söz konusu referansları bir havuz da ***(Intern pool)*** tutar. Çalışma zamanında yeni bir **string** değişken gündeme geldiğinde içeriği bu havuzdan kontrol edilir. Kısacası **CLR** aynı içeriğe sahip olan **string** değişkenler için **unique** bir referans tutar. Peki ya elimizdeki bir **string'** in eğer var ise **unique** olan referansını buradan nasıl alabiliriz? İşte bu noktada devreye **String.Intern** metodu girer.

Eğer söz konusu içerik **Intern Pool** içerisinde var ise, onun referansı ile bir **string** değişken elde edilir. Yoksa da bu içerik, **Intern Pool**' a atılır ve **unique** bir referans olarak tutulmaya devam edilir.**IsInterned** metodu ise söz konusu içerik eğer **Intern Pool** içerisinde yer alıyorsa yine referansı döndürecek ama yoksa **null** değerini verecektir. İşte örnek bir kod parçası ve çalışma zamanında elde edilen sonuçlar.

```
string name1 = "burak"; // name1 Intern Pool içerisinde set edildi
string name2 = String.Intern("burak");
if(name1==name2)
   Console.WriteLine(true);
else
   Console.WriteLine(false);
string name3 = String.Intern("burak s.");
if(name2==name3)
   Console.WriteLine(true);
else
   Console.WriteLine(false);
string name4 = new string(new char[]{'s', 'e', 'l', 'i', 'm'});
Console.WriteLine("selim pool {0}",String.IsInterned(name4)==null?"da değil":"da");
String.Intern(name4);
Console.WriteLine("Intern çağrısı sonrası. selim pool {0}", String.IsInterned(name4) ==
null ? "da değil" : "da");
```



Bu yazımızda şöyle kıyıda köşede kalmış olabilecek bir kaç dil kabiliyetine yer vermeye çalıştık. Kimbilir gözümüzden kaçan, dikkat etmediğimiz veya kullanmadığımız için zamanla unuttuğumuz neler var. Biraz ilham vermiş olabilirim. Siz de araştırın bakalım. Bir başka yazımızda görüşmek dileğiyle hepinize mutlu günler dilerim

HowTo_CSharp_Myths.zip (42,82 kb)

Asp.Net Web API için Sayfalama Tekniği

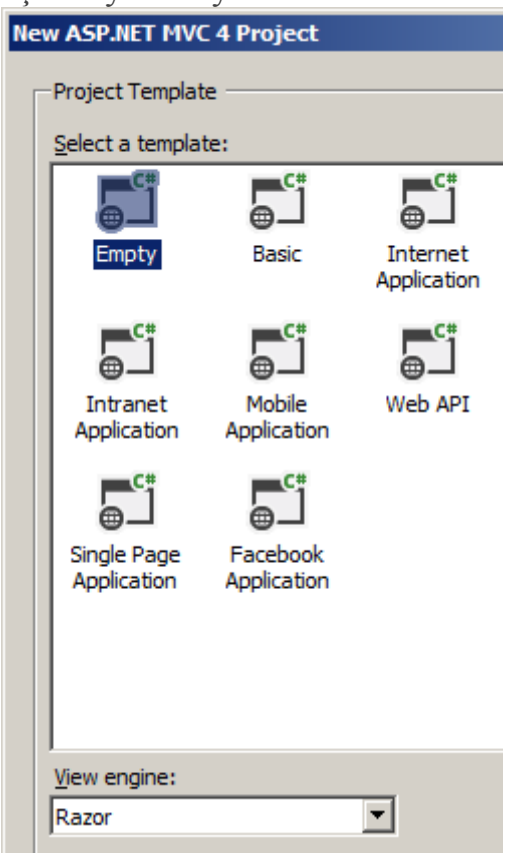
Salı, 21 Mayıs 2013 12:11 by bsenyurt

Merhaba Arkadaşlar,
Bu aralar şirkette işler oldukça kesat. En azından benim bulunduğum departman itibariyle böyle bir durum söz konusu. Sanırım kurumsal kimlik kazanmış firmaların genel sorunu da bu olsa gerek. Kaynak planlaması ve dağıtımının bir türlü istenen şekilde yapılamayışı. Hal böyle olunca aynı firmada hatta aynı departman içerisinde, çok yoğun çalışan insanlara ve beraberinde her hangi bir işi olmayanlara*(benim gibi)* rastlamak mümkün.
Böyle bir durumda keyif sürmek ve tembel tembel internette gezmek*(Video paylaşmak, onun bunun ciklemesine yetişmeye çalışmak vb)* yapılabilecek en cazip işlerden birisi gibi gözükse de, hızla ilerleyen teknoloji ne yazık ki buna müsade etmemekte. Neredeyse her hafta yeni konuların ele alındığı bilmem kaç katlık yottabyte' lık bilgi denizinde sürekli bir şeyler öğrenmek zorunda olan biz köle geliştiricilerin iş olmasa da kendisine iş yaratması şart. **Eğitim şart** da diyebiliriz Öyleyse tembel tembel oturmayalım ve gelin birlikte yeni bir mevzuya dalalım.
Asp.Net Web API alt yapısının popüler olmasının ardında yatan en büyük sebeplerden birisi, **HTTP**tabanlı servis yayılımına izin vermesidir. Hemen her fonksiyonel birimin veya bütünlüğün servis odaklı teknolojiler ile ele alındığı ve istemcilere sunulduğu bir dünyada, bu ihtiyacı eskiden beri var olan **HTTP** protokolünün **Post, Put, Get, Delete** gibi standart metodlarına göre karşılamak elbette önemlidir. Bu sayede **Microsoft** tabanlı olarak geliştirilen Web API servislerinin, dış dünyadaki herhangibir **Client** tarafından tüketilmesi de oldukça kolaydır. Üstelik **OData*(Open Data Protocol)*** desteği sayesinde, veri odaklı servislerin standart **URL** bazlı parametreler ile sorgulanabilmesi mümkün hale gelmektedir.
Bir önceki cümlede belirttiğimiz **veri odaklı servis*(Data-Centric Service)*** aslında bu yazımızdaki senaryomuzun da ana konularındandır. Veri odaklı servisler tahmin edileceği üzere büyük boyutlu verileri de ele alabilirler. Çok eskilerden de bildiğimiz üzere **Asp.Net** ile web programlamanın ilk yıllarında yaşanan en büyük sıkıntılardan birisi, verinin sayfalanarak getirilmesiydi. **Stored Procedure**' ler de yapılan bazı hamleler ile bu işi çözebiliyor olsak da, SP desteği olmayan bir veri kaynağının kullanılma olasılığını da göz ardı etmemek gerekiyor.
Hatırlayacağınız gibi web tarafındaki ilk veri bağlı kontrollerde sayfalama sistemi şöyle çalışmaktaydı:
Sorgulama sırasında tüm veri çekilir ve içinden örneğin Xnci sıradaki 10 luk eleman kümesi getirilirdi. Çok doğal olarak her seferinde tüm veri kümesinin çekilmesi çok

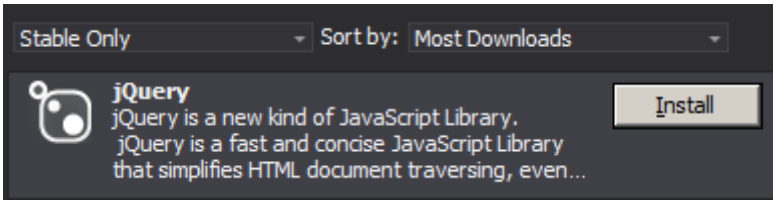
da istenen bir yaklaşım değildi. Örneğin 10 milyon satır içinden 3ncü 50lik kümeyi getirmek istediğimizde
Ancak ilerleyen sürümler de bu durum değişti ve özellikle SQL tarafında row_number kullanımı ile doğru sayfalama işlemlerinin yapılabilmesinin yolu açıldı. Buna bir de LINQ tarafındaki anahtar kelime desteği eklenince, Entity Framework gibi alanlarda doğru sayfalama stratejilerini kullanabilir olduk.
Peki **Asp.Net Web API** tarafında sayfalama işlevselliği nasıl karşılanabilir? İşte bu yazımızda cevap bulmaya çalışacağımız soru bu. Şimdi basit bebek adımları ile ilerleyerek senaryomuzu hayata geçirmeye çalışalım.

Proje için Ön Hazırlıklar

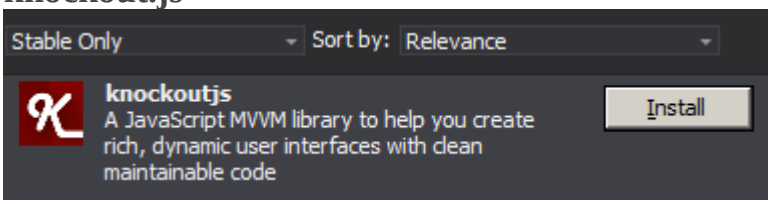
İlk olarak **Visual Studio 2012** ortamında **Empty MVC 4** şablonunda bir web uygulaması açarak yola koyulabiliriz.



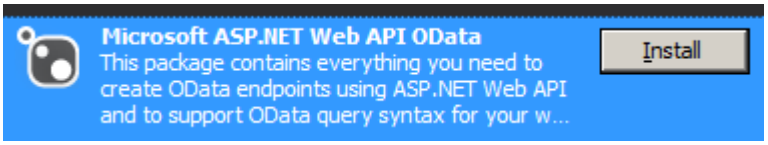
Söz konusu senaryomuzda istemci tarafında yazacağımız kodlar oldukça önemlidir. Bu yüzden**jQuery** ve **Knockout.js**' in son sürümlerinin kullanılmasında yarar vardır.
Ayrıca **OData** sorgularını kullanacağımız için **Microsoft ASP.NET Web API OData** paketini de eklememiz gerekmektedir. Bu referansları **NuGet** paket yönetim aracı ile projeye kolayca dahil edebiliriz.
jQuery



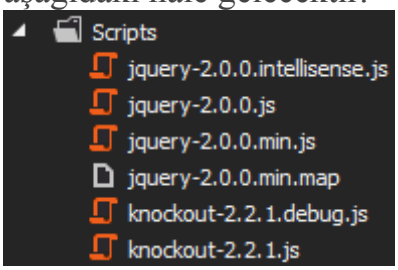
knockout.js



ve **Microsoft ASP.NET Web API OData**



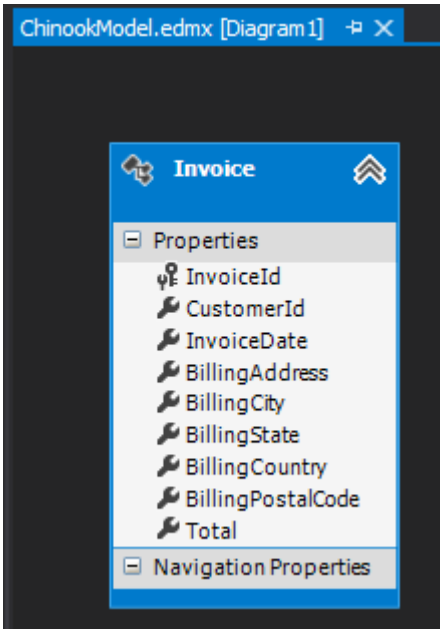
Bu eklentilerin yüklenmesi sonrasında **jQuery** ve **knockout.js** için **scripts** klasörü aşağıdaki hale gelecektir.



Buradaki **Javascript** kütüphaneleri **cshtml** tarafında kullanılacaktır.

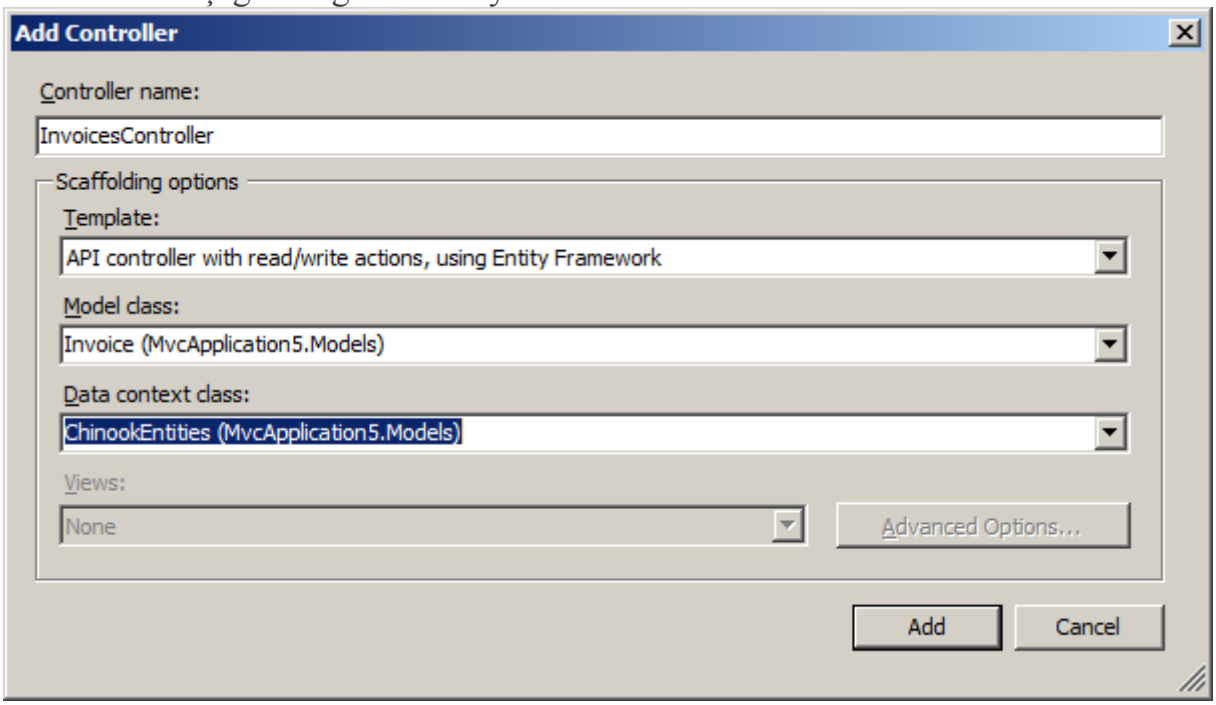
Veri Modelinin Eklenmesi

Çok doğal olarak senaryomuzda veri odaklı bir uygulama öngörülmektedir. Örneğimizde **Entity Framework**' den yararlanılabilir. Bu nedenle **Model** klasörüne yeni bir **Ado.Net Entity Data Model**öğesi ekleyelim. Örneğimizde kobay veritabanlarımızdan birisi olan **Chinook**' a bağlanıyor olacağız. Bu veritabanın yer alan **Invoice** tablosu **400** satırdan fazla veri içermekte ve senaryomuz için ideal bir veri kümesi sunmaktadır. Bu sebpeten sadece ilgili tabloyu kullansak yeterli olacaktır.



Controller Eklenmesi

Pek tabi veri modelinin oluşturulmasının ardından bir de **Controller**' ın eklenmesi gerekmektedir.**Model** ile **View** arasındaki köprüyü kuracak olan **Controller** sınıfının temel özelliklerini aşağıdaki gibi belirleyebiliriz.



Sınıf içeriği bizim için otomatik olarak üretilecektir. Ancak senaryomuz için gerekli olmayan detayları çıkartabiliriz. Buna göre içeriği aşağıdaki kod parçasında görüldüğü gibi değiştirmemiz yeterli olacaktır.

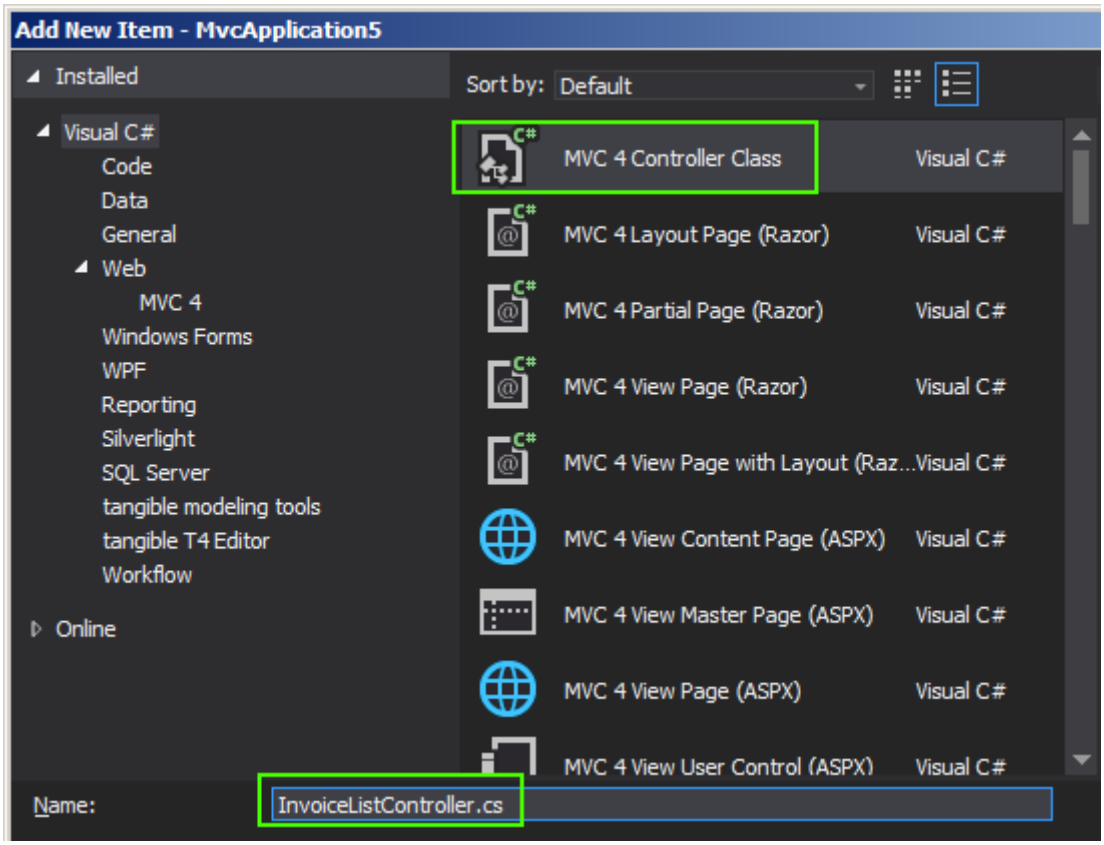
```
using System.Web.Http;
using MvcApplication5.Models;
```

```
namespace MvcApplication5.Controllers
{
   public class InvoicesController : ApiController
   {
      private ChinookEntities db = new ChinookEntities();
      // GET api/Invoices
      [Queryable]
      public IQueryable<Invoice> GetInvoices()
      {
         return db.Invoices.AsQueryable();
      }

      protected override void Dispose(bool disposing)
      {
         db.Dispose();
         base.Dispose(disposing);
      }
   }
}
```

ApiController türevli **InvoicesController** sınıfı içerisinde yer alan **GetInvoices** metodu,**Queryable** tipi ile nitelendirilmiştir. Bu **nitelik*(Attribute)*** sayesinde **OData** sorgu desteği sağlanmış olmaktadır ki bu, sayfalama için kullanılacak olan **top,skip** ve **orderby** komutları için gereklidir.

Görsel Taraf(İstemci) için Controller Eklenmesi

ApiController bilindiği üzere **Web API** servis desteği için gereklidir. Ancak istemci tarafını düşündüğümüzde standart bir **MVC Controller**' ının kullanılması gerekecektir. Nitekim **cshtml**içeriğini kullanarak **Web API** servisi üzerinden **OData** anahtar kelimeleri ile sayfalama talebinin gönderilip, sonuçların gösterileceği bir **View** öğesi gerekmektedir. Bunun için projeye yeni bir **MVC 4 Controller** ekleyerek ilerleyebiliriz.



Bu işlem sonucunda aşağıdaki sınıf içeriği üretilmiş olacaktır.

```
using System.Web.Mvc;
namespace MvcApplication5.Controllers
{
   public class InvoiceListController
      : Controller
   {
      //
      // GET: /InvoiceList/
      public ActionResult Index()
      {
         return View();
      }
   }
}
```

View Öğesinin Eklenmesi (index.cshtml)

Tahmin edileceği üzere bir de **View** öğesine ihtiyacımız bulunmakta. Bu amaçla **Views** klasörü içerisinde **InvoicesList** isimli bir alt klasör açarak içerisine yeni bir **View** ilave edip devam edebiliriz. *(**View**' un adını **index** olarak belirleyip **Razor Engine'** i kullanacak şekilde tesis edelim)*
İçeriği ise aşağıdaki şekilde düzenleyelim.

```
@{
   ViewBag.Title = "Index";
}
<h2>Invoice List</h2>
<script src="~/Scripts/jquery-2.0.0.min.js"></script>
<script src="~/Scripts/knockout-2.2.1.js"></script>
<div>
   Gösterilmek istenen satır sayısı<br />
   <input type="text" id="txtRowSize" />
   <br />
   Başlangıç Noktası<br />
   <input type="text" id="txtRowIndex" />
   <br />
  <input type="button" id="btnGetInvoices" value="Get Inovices" data-
bind="click:InvoiceModel.GetInvoices"/>
</div>
<table border="1">
<thead>
  <tr>
   <th>InvoiceId</th>
   <th>CustomerId</th>
   <th>InvoiceDate</th>
   <th>BillingAddress</th>
   <th>BillingCity</th>
   <th>BillingState</th>
   <th>BillingCountry</th>
   <th>BillingPostalCode</th>
   <th>Total</th>
  </tr>
</thead>
<tbody data-bind="template: { name: 'InvoiceDataModel', foreach:
InvoiceModel.Invoices }">
</tbody>
</table>
<script type="text/html" id="InvoiceDataModel">
<tr>
  <td>
   <span style="width:100px;"  data-bind="text: $data.InvoiceId" />
  </td>
  <td>
   <span style="width:100px;"  data-bind="text: $data.CustomerId" />
```

```
  </td>
  <td>
   <span style="width:100px;"  data-bind="text: $data.InvoiceDate" />
  </td>
  <td>
   <span style="width:100px;" data-bind="text: $data.BillingAddress"  />
  </td>
  <td>
   <span style="width:100px;"  data-bind="text: $data.BillingCity" />
  </td>
     <td>
   <span style="width:100px;"  data-bind="text: $data.BillingState" />
  </td>
    <td>
   <span style="width:100px;"  data-bind="text: $data.BillingCountry" />
  </td>
    <td>
   <span style="width:100px;"  data-bind="text: $data.BillingPostalCode" />
  </td>
    <td>
   <span style="width:100px;"  data-bind="text: $data.Total" />
  </td>
</tr>
</script>
<script type="text/javascript">
var InvoiceModel = {
  Invoices:ko.observableArray([])
};
InvoiceModel.GetInvoices= function ()
{
    InvoiceModel.Invoices([]);
  var rowSize = $("#txtRowSize").val();
  var rowIndex = $("#txtRowIndex").val();

  var url = "/api/Invoices?$top=" + rowSize + '&$skip=' + (rowIndex * rowSize) +
'&$orderby=InvoiceId';

  $.ajax({
   type: "GET",
   url: url,
   success: function (data)
```

```
  {
   InvoiceModel.Invoices(data);
  },
  error: function (err)
  {
   alert(err.status + "," + err.statusCode);
  }
 });
};
ko.applyBindings(InvoiceModel);
</script>
```

Kısa da olsa neler yaptığımıza bir bakalım. En önemli kısım tabi ki **btnGetInvoices** isimli düğmeye basıldıktan sonra çalışan kod içeriğidir. Burada kilit nokra **url** değişkenine atanan ifadedir. Dikkat edileceği üzere burada bir **OData** sorgusu oluşturulmakta ve **top** ile **skip** komutlarından yararlanılarak bir veri çekme işlemi gerçekleştirilmektedir. Elde edilen sonuç kümesinin ilgili veri kontrolüne bağlanması noktasında ise bir **Ajax** çağrısı söz konusudur. **success** bloğunda sorgu sonuçlarının ilgili veri kontrolüne doldurulması işlemi icra edilmektedir.

Route Ayarları

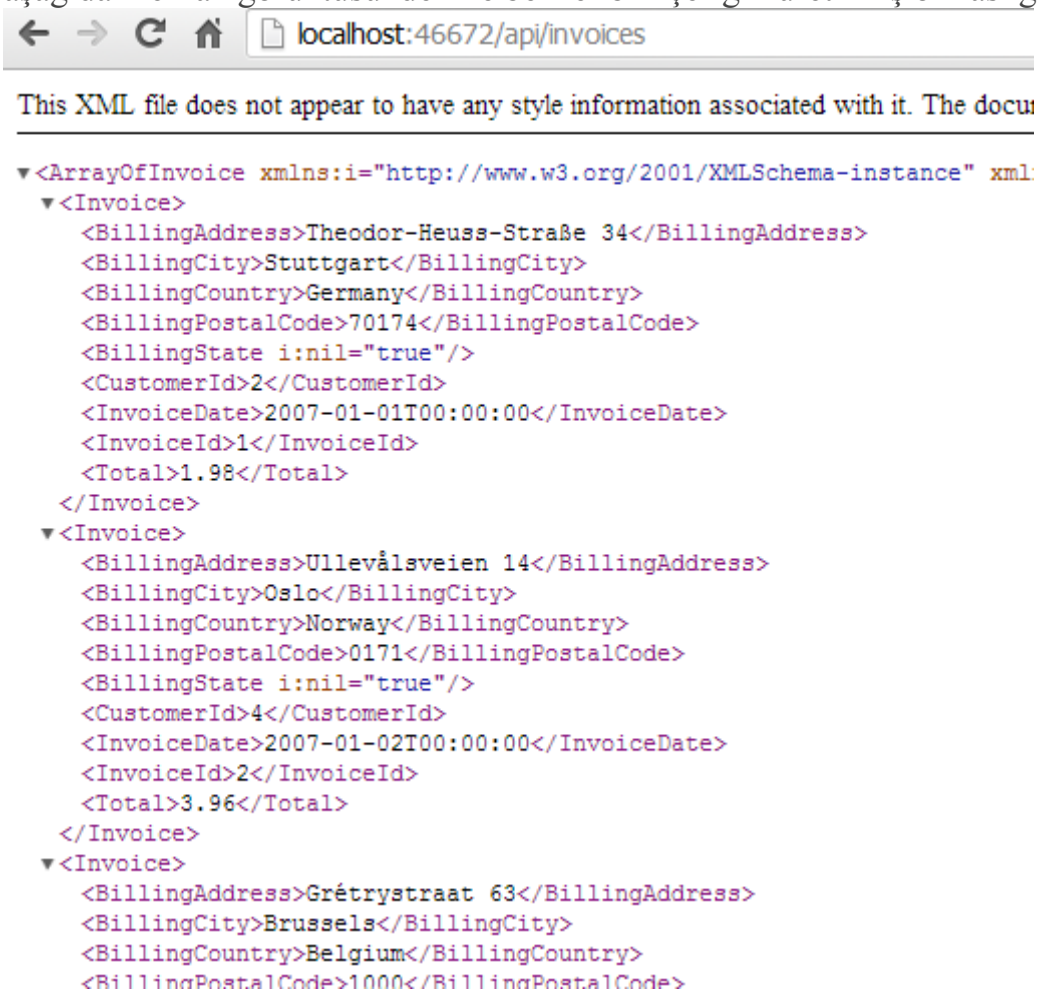
Testlere başlamadan önce **InvoiceList** görünümü için **Route** ayarlarını güncellememizde yarar vardır. Bunun için **App**_Start klasöründe yer alan **RouteConfig.cs** içeriğini aşağıdaki gibi değiştirelim.

```
using System.Web.Mvc;
using System.Web.Routing;
namespace MvcApplication5
{
   public class RouteConfig
   {
      public static void RegisterRoutes(RouteCollection routes)
      {
         routes.IgnoreRoute("{resource}.axd/{*pathInfo}");
         routes.MapRoute(
            name: "Default",
            url: "{controller}/{action}/{id}",
            defaults: new { controller = "InvoiceList", action = "Index", id =
UrlParameter.Optional }
         );
      }
   }
}
```

Buna göre uygulamamızı çalıştırdığımızda varsayılan olarak **InoviceList** ile ilişkili **View**' a gidilecektir.

Test

İlk olarak **Web API** servisinin çalıştığından emin olmalıyız. Bu amaçla **URL** satırınahttp://localhost:46672/api/Invoices benzer bir ifade girildiğinde, aşağıdaki ekran görüntüsündekine benzer bir içeriğin üretilmiş olması gerekmektedir.



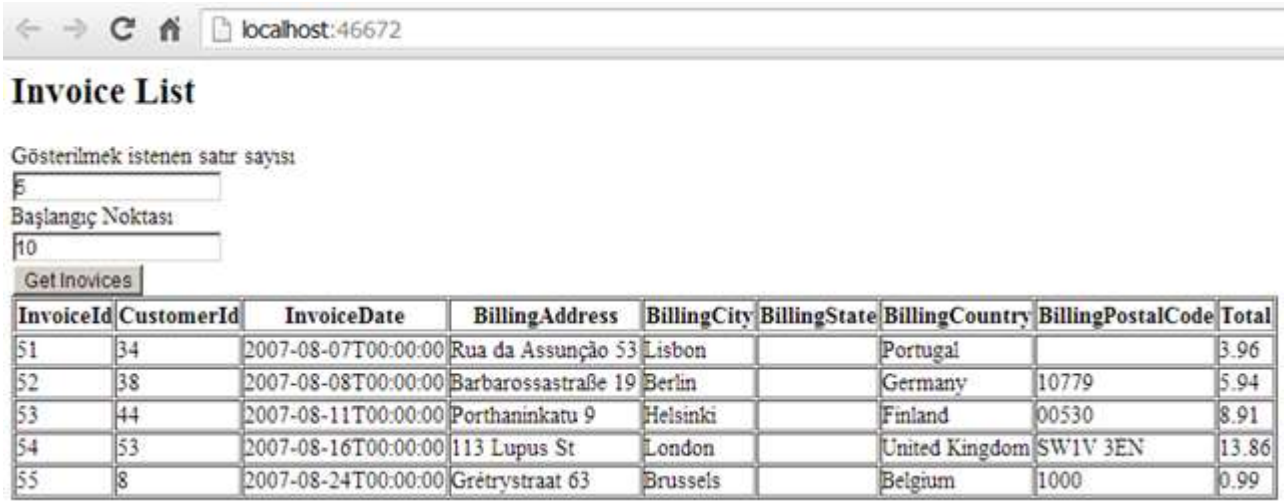
Eğer aşağıya doğru inerseniz tüm Invoice içeriğinin çekildiğini görebilirsiniz.
Ne var ki, **Web API** servisimiz için test noktasında önem arz eden bir mevzuda **top, skip** ve**orderby** komutlarına cevap verebiliyor olmasıdır. Örneğin http://localhost:46672/api/invoices?$top=3&$skip=10&$orderby=InvoiceId şeklin de bir talep girdiğimizi düşünelim. Aslında bu talep ile servis tarafına şu mesajı iletmiş oluyoruz;
Önce Invoice satırlarını InvoiceId değerine göre bir diz bakalım. Sonra da10ncu indisten itibaren bana ilk 3 sıradakini getir.
Talebin işlenmesi sonrası tarayıcı üzerinde aşağıdakine benzer bir sonuç elde etmemiz gerekmektedir.

This XML file does not appear to have any style information associated with it. The document tr


```
▼<ArrayOfInvoice xmlns:i="http://www.w3.org/2001/XMLSchema-instance" xml
  ▼<Invoice>
     <BillingAddress>202 Hoxton Street</BillingAddress>
     <BillingCity>London</BillingCity>
     <BillingCountry>United Kingdom</BillingCountry>
     <BillingPostalCode>N1 5LH</BillingPostalCode>
     <BillingState i:nil="true"/>
     <CustomerId>52</CustomerId>
     <InvoiceDate>2007-02-06T00:00:00</InvoiceDate>
     <InvoiceId>11</InvoiceId>
     <Total>8.91</Total>
   </Invoice>
  ▼<Invoice>
     <BillingAddress>Theodor-Heuss-Straße 34</BillingAddress>
     <BillingCity>Stuttgart</BillingCity>
     <BillingCountry>Germany</BillingCountry>
     <BillingPostalCode>70174</BillingPostalCode>
     <BillingState i:nil="true"/>
     <CustomerId>2</CustomerId>
     <InvoiceDate>2007-02-11T00:00:00</InvoiceDate>
     <InvoiceId>12</InvoiceId>
     <Total>13.86</Total>
   </Invoice>
  ▼<Invoice>
     <BillingAddress>1600 Amphitheatre Parkway</BillingAddress>
     <BillingCity>Mountain View</BillingCity>
     <BillingCountry>USA</BillingCountry>
     <BillingPostalCode>94043-1351</BillingPostalCode>
     <BillingState>CA</BillingState>
     <CustomerId>16</CustomerId>
     <InvoiceDate>2007-02-19T00:00:00</InvoiceDate>
     <InvoiceId>13</InvoiceId>
     <Total>0.99</Total>
   </Invoice>
 </ArrayOfInvoice>
```

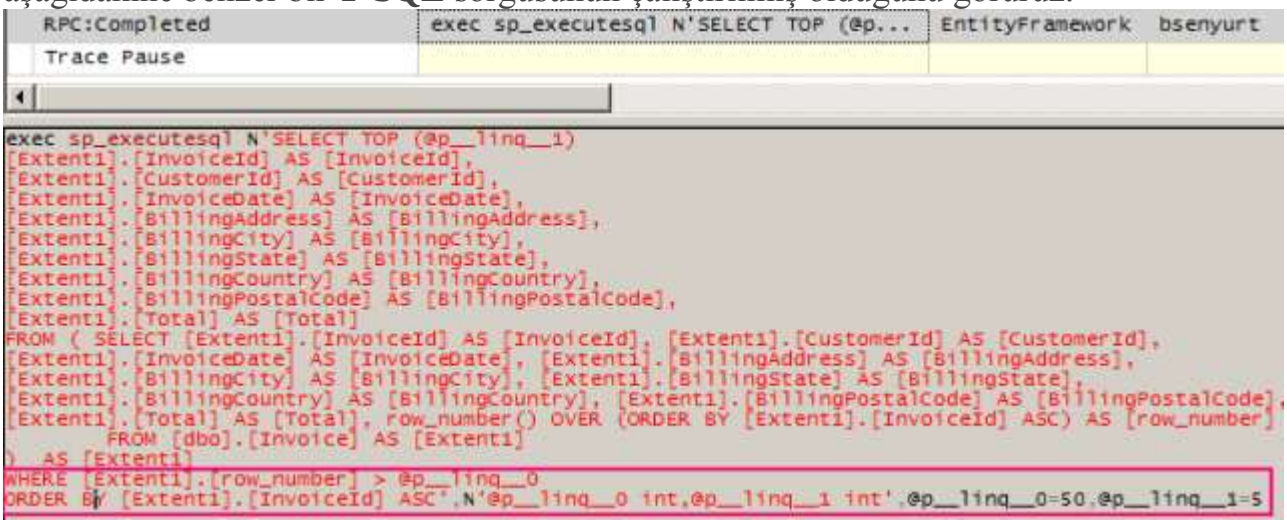

Şimdi asıl **View** içeriğini test ederek asıl senaryomuzu yürütebiliriz. Uygulamamızı varsayılan olarak çalıştırdığımızda **Route** tanımlaması nedeniyle doğrudan **index.cshtml** içeriğini görüyor oluruz.

Şimdi bazı veriler girerek örneğimizi test edelim.



InvoiceId	CustomerId	InvoiceDate	BillingAddress	BillingCity	BillingState	BillingCountry	BillingPostalCode	Total
51	34	2007-08-07T00:00:00	Rua da Assunção 53	Lisbon		Portugal		3.96
52	38	2007-08-08T00:00:00	Barbarossastraße 19	Berlin		Germany	10779	5.94
53	44	2007-08-11T00:00:00	Porthaninkatu 9	Helsinki		Finland	00530	8.91
54	53	2007-08-16T00:00:00	113 Lupus St	London		United Kingdom	SW1V 3EN	13.86
55	8	2007-08-24T00:00:00	Grétrystraat 63	Brussels		Belgium	1000	0.99

Dikkat edileceği üzere **51 numaralı InvoiceId değerinden itibaren 5 adet satır** getirilmesi istenmiş ve buna göre bir sonuç kümesi elde edilmiştir. Tabi burada önemli olan bir diğer nokta da fonksiyonun icra edilmesi sırasında **SQL** tarafında çalıştırılan sorgu ifadesidir. Özellikle bu sorgu ifadesinde doğru bir sayfalama yapılması da çok önemlidir. Bu amaçla **SQL Server Profiler**aracından yararlanabiliriz. Sonuç itibariyle aşağıdakine benzer bir **T-SQL** sorgusunun çalıştırılmış olduğunu görürüz.



Dikkat edileceği üzere **row_number** komutundan yararlanılarak gerçek anlamda sayfalama işlemi uygulanmıştır.

Sonuç

Görüldüğü üzere **OData** sorgu desteği sunan **Asp.Net Web API** servislerini kullanarak sayfalama işlemlerini gerçekleştirmek oldukça kolaydır. Bu iş de başrol oyuncu olan **top, skip** ve **orderby**anahtar kelimeleri bir **OData** standardı olduğundan, istemci tarafı **Microsoft** dışı bir platform da olabilir. Tabi burada tek bağlayıcı nokta **SQL** veritabanı ve **Entity Framework** kullanımıdır. Farklı veri kaynaklarında **row_number** gibi bir kullanım şekli söz konusu olmayabilir. Böyle bir vaka da tahmin edileceği üzere **Web API Controller** içerisindeki ilgili operasyon noktalarında müdahale de bulunmak gerekebilir*(Araştırmadım benim yerime siz bu işi*

yapın *)* Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Eclipse Üzerinden Java ile TFS Client Object Model Konuşuyor

Pazartesi, 13 Mayıs 2013 07:10 by bsenyurt

Merhaba Arkadaşlar,
Çok değil daha bir kaç sene öncesine kadar*(Özellikle .Net' in duyurulduğu yıllarda ve izleyen bir kaç senede)* yazılım dünyasında neredeyse yandaki resimdekine benzer bir kavga vardı*(Benzetmeyi biraz abartmış olabilirim)*
Java' cılar, **C#**' çıları pek sevmez iken tam tersi durum da pekala geçerliydi. Ben hiç bir zaman birisinin fanatiği olmadım. Hatta Java ile ufak çaplı bir kaç deneyimim bile oldu.
Peki gerçek dünya böyle mi? Özellikle kalabalık yazılım ekiplerinin olduğu, çok fazla sayıda ürünün koştuğu dünyalarda, sadece **Java**' cıları, **C#**' çıları değil, daha pek çok programlama dili geliştiricilerini bir arada görmekteyiz. Öneğin ben bulunduğum konum itibariyle **C**' cilerin, **Assembler**' cıların, **PowerBuilder**' cıların, **.Net**' çilerin,**Java**' cıların ve hatta **Cobol**' cuların arasında yaşamaktayım.
Hepsi kendi dünyalarını kullanarak ürünler geliştiriyor olsalar da, zaman içerisinde birbirleriyle konuşması gereken uygulamalar bütününün de bir parçası olmaktan kaçamıyorlar. Özellikle işin içerisine bir **ALM*(Application LifeCycle Management)*** aracı girdiğinde. İşte bu günkü konumuzda buna itaf edilecek Haydi gelin başlayalım.
Bildiğiniz üzere bir süredir **Team Foundation Server'** ın çevre dünya ile olan etkileşimini incelemeye çalışıyorum. Açıkçası **TFS'** in gerek servis yapısı gerek **Client Object Model** gibi kütüphaneleri sayesinde, dış dünya ile olan entegrasyonu son derece kolay. Bu gün buna bir kere daha inandım. Çünkü bir **Java** uygulaması içerisinde **TFS Client Object Model**' i kullanarak, bir**Team Project**' in **Work Item** listesini sorguladım Nasıl yaptığımı merak ediyorsanız okumaya devam edin. Tabi bu işte de çok önemli bir yardımcım vardı. O da **Microsoft** tarafından geliştirilen ve ücretsiz olarak sunulan **Client Object Model SDK**'sı. Ama **Java** için olan sürümü.
Microsoft Team Foundation Server 2012 Software Development Kit for Java içeriğini bu adresten indirebilirsiniz*(Makaleyi hazırladığım tarih itibariyle 15 Şubat 2013' te yayınlanmış güncel bir sürümü bulunmaktaydı)*

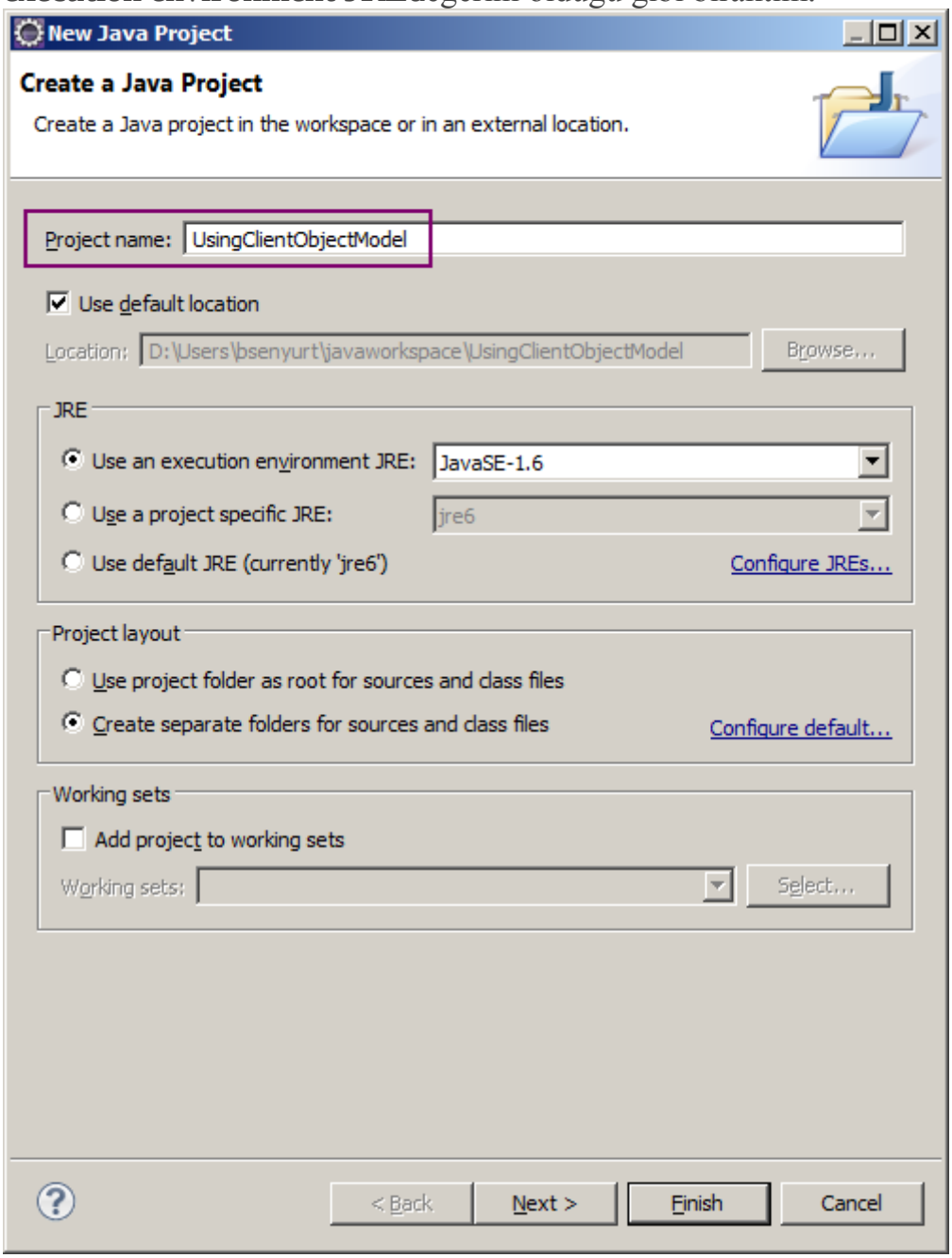
Senaryo

Senaryomuz esas itibariyle yine bir **Hello World** uygulaması olacak. **Console** ekranına belirli bir**Team Project** içerisinde yer alan **Work Item** bilgilerini*(Product Backlog Item, Bug, Task gibi)*yazdırmaya çalışacağız. Örneğin **Work Item**' ın başlığını*(Title)*, tipini*(WorkItem Type)*, numarasını*(ID)*düzgün bir sırada çekmeyi deneyebiliriz. Java kodlaması yapacağımız için **Eclipse** gibi bir **IDE** tercih edilebilir ki ben öyle yaptım

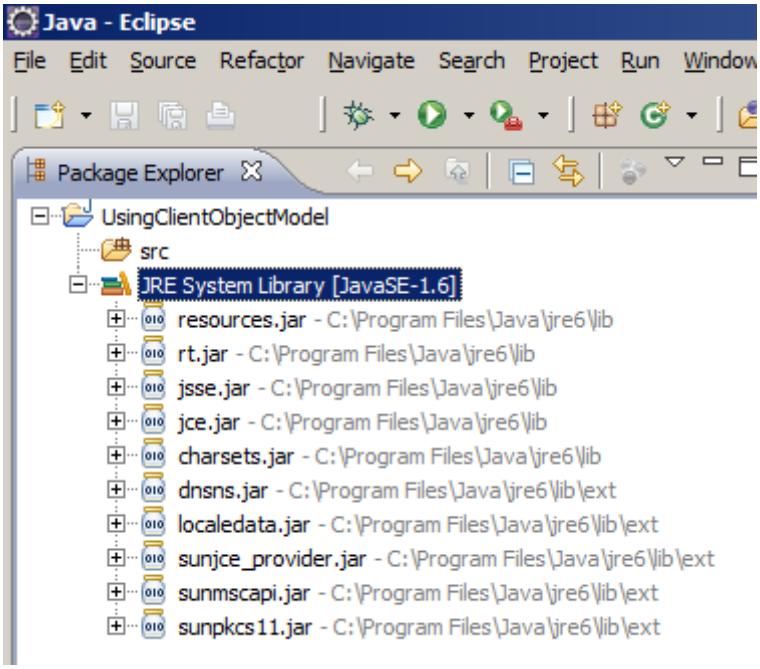
Bebek Adımları

Adım 0

İlk olarak **Eclipse** üzerinde yeni bir **Java Projesi** oluşturarak işe başlayalım*(File->New->Java Project)*. Ben **UsingClientObjectModel** olarak adlandırdığım projeyi, sistemimde kurulu olan**Workspace** içerisinde oluşturdum. **Runtime** olarak **1.6** sürümünü kullanmayı tercih ediyorum. Bu nedenle daha önceki projelerden varsayılan olarak kalan **Use an execution environment JRE**değerini olduğu gibi bıraktım.

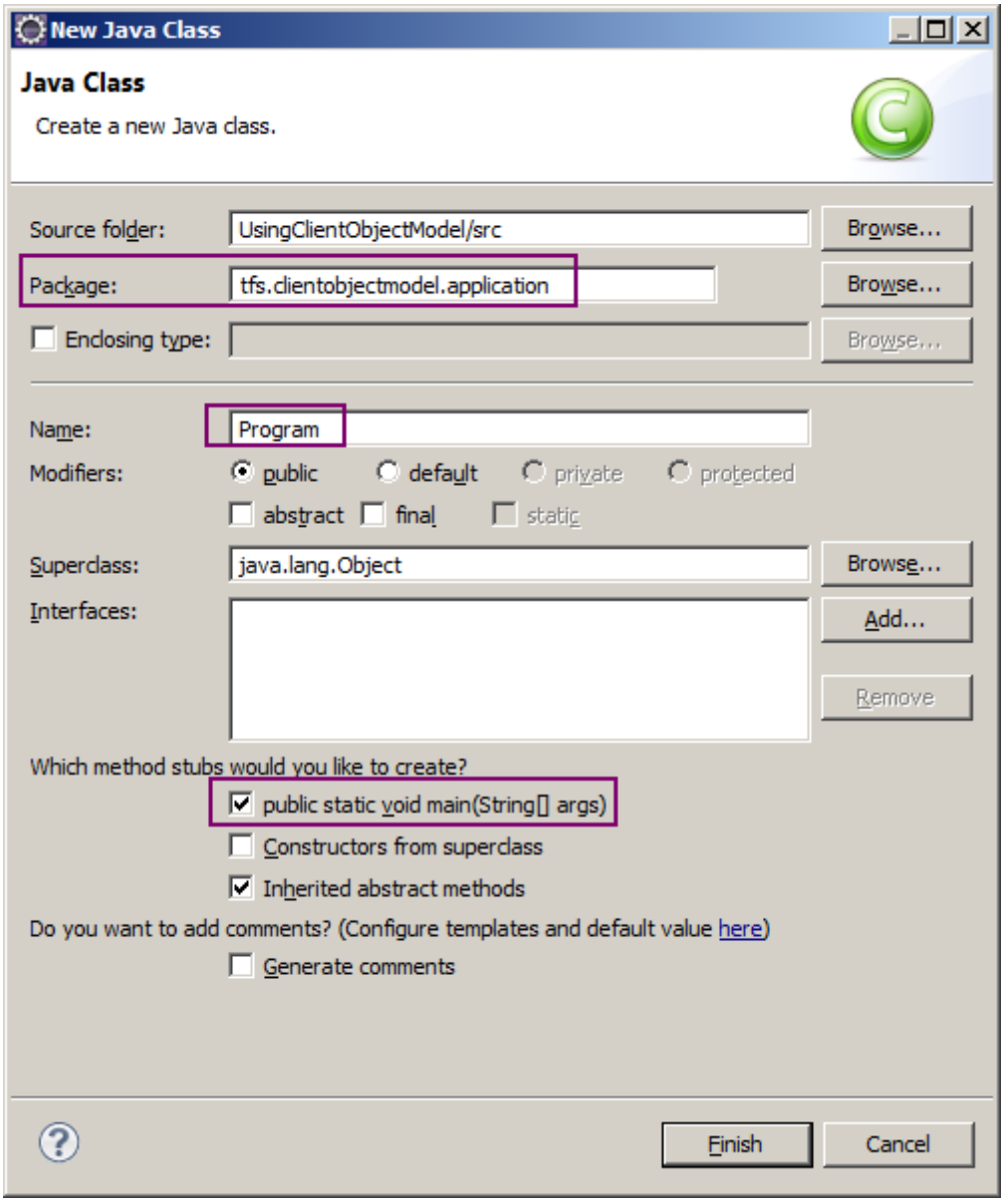


Bu üretim işlemi sonrasında **Eclipse IDE**' sinde aşağıdakine benzer bir içerik oluştuğunu görmeliyiz.*(Elbette sisteminizde **Java Runtime Environement'** in yüklü olduğu fiziki adresler daha farklı olabilir.)*

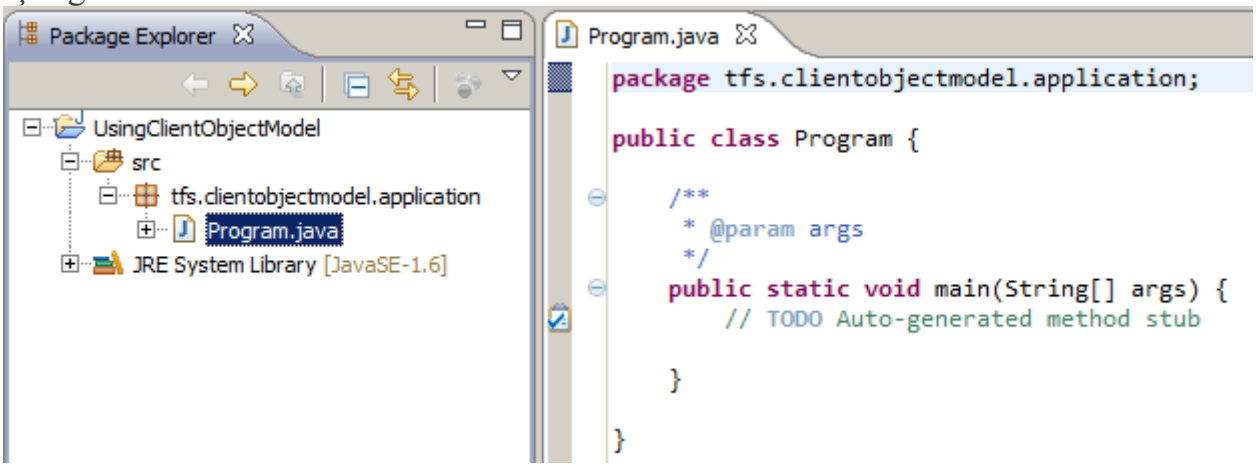


Adım 1

İkinci adım ise main metodunu içerecek olan ve asıl kodlarımızı yazacağımız sınıfı eklemek olacaktır.**Visual Studio** tarafından kalma bir alışkanlık nedeniyle **Program** olarak isimlendirdiğim sınıf,**tfs.clientobjectmodel.application** isimli paket içerisinde konuşlandırılacak*(Siz kendi istediğiniz paket tanımlamasını yapabilirsiniz)* Krtik noktalardan birisi de, **public static void main(String[] args)** seçeneğinin işaretli olmasıdır. Bu, tahmin edeceğiniz üzere programın giriş noktası olan **main**metodudur.
C# tarafından da bildiğiniz üzere static Main metodu, exe tipindeki uygulamaların giriş noktasıdır. Aynı durum Java uygulamaları için de geçerlidir. Tek fark Java tarafında isimlendirme standartı gereği küçük m harfi ile başlanmasıdır. Dikkat edileceği üzere her iki dilde metoda parametre olarak string tipinden bir dizi alır. Programa dış ortamdan parametre aktarabilmek amacıyla

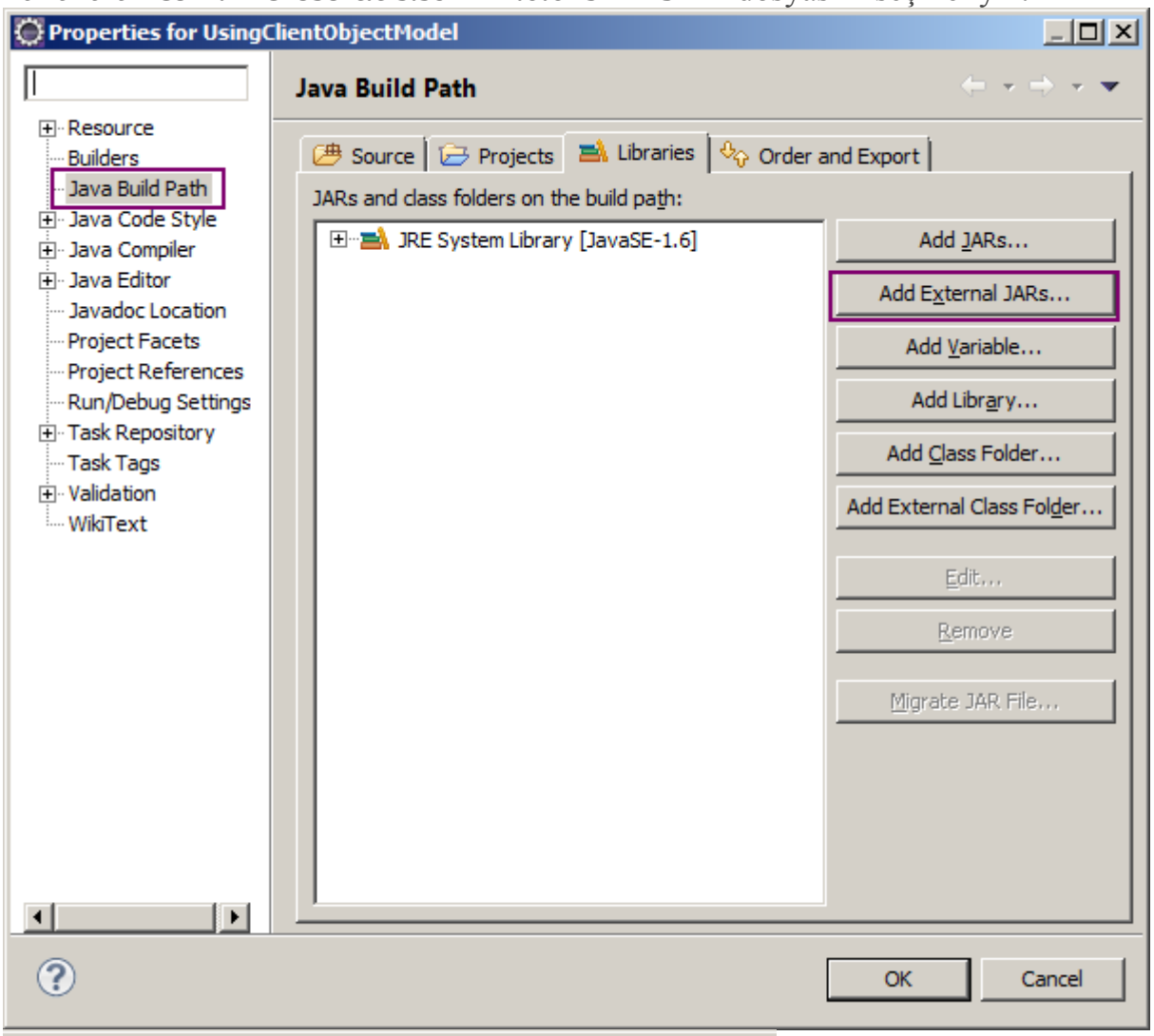


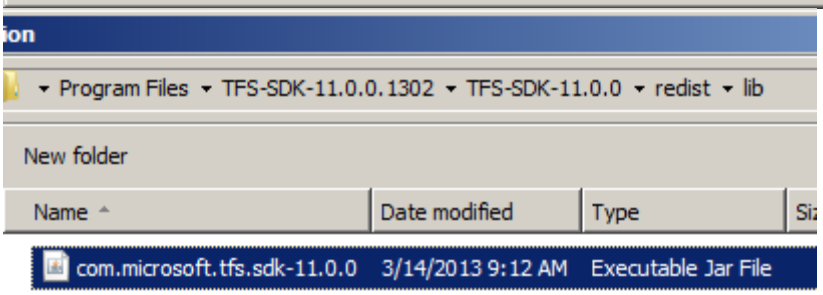
Bu işlemin sonucunda **Eclipse IDE**' si bizim için aşağıdaki ekran görüntüsünde yer alan içeriği üretmelidir.



Adım 2

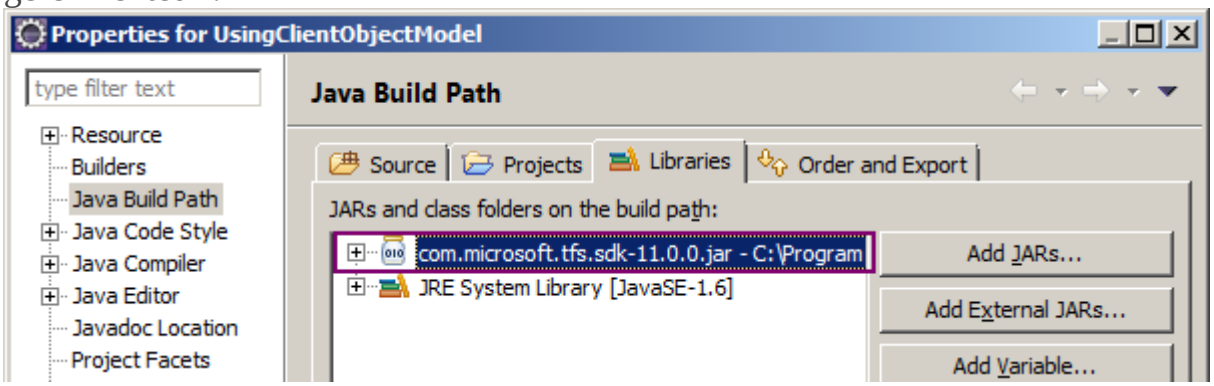
Üçüncü adımda, bilgisayarımıza indirip açtığımız **TFS SDK**' sını referans ediyor olacağız. Bunun için ilgili **JAR** dosyasının uygulamaya bildirilmesi*(bir başka deyişle referans edilmesi)* gerekmektedir. Söz konusu bildirim için proje özelliklerinden **Java Build Path** kısmına gelmeli ve **Add External JARs**düğmesini kullanarak **com.microsoft.tfs.sdk-11.0.0** isimli **JAR** dosyasını seçmeliyiz.



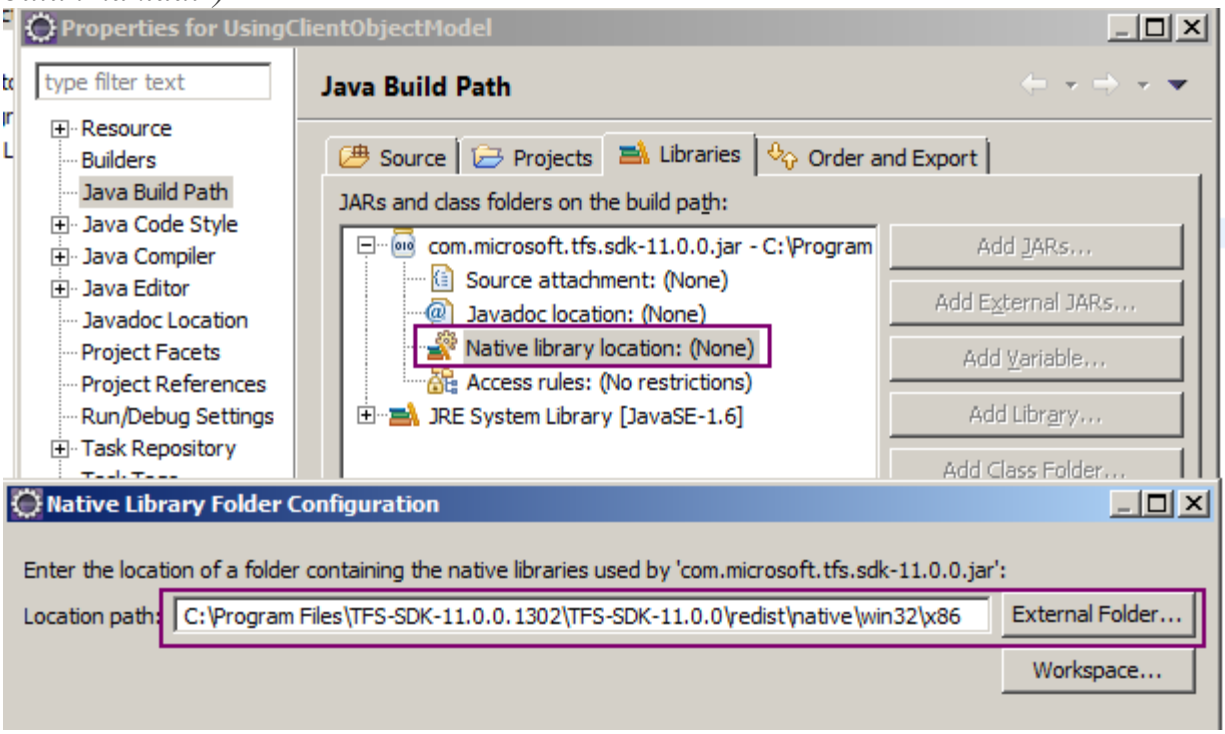


Konu ile ilişkili olarak yaptığım araştırmalarda şöyle bir kullanıma da rastladım. Hazırlanan Java uygulamasının dağıtılabileceği de düşünülürse SDK' nın Redistributable olan parçalarının proje klasörü altına kopyalanması ve ilgili path bildirimlerinin bu fiziki adresleri gösterecek şekilde yapılması yolu da tercih edilebilir. Böyle bir durumda Add JARs düğmesinden hareket edilir.

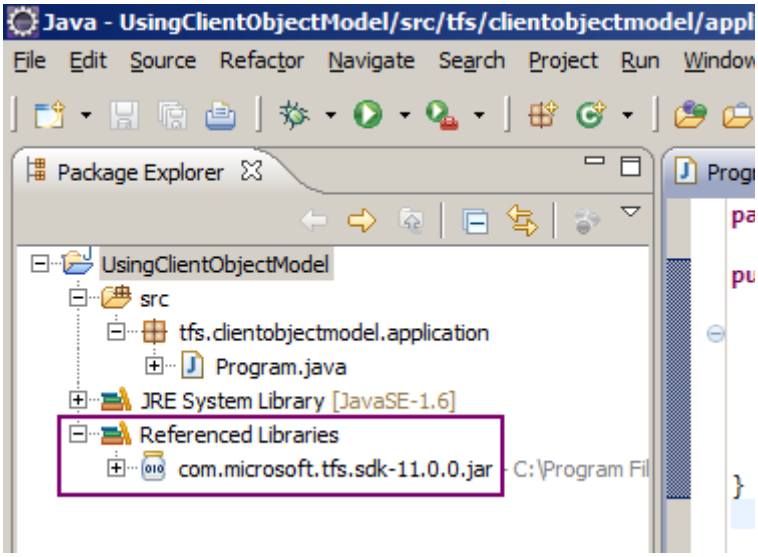
Sonuç olarak **Java Build Path** kısmının aşağıdaki ekran görüntüsündeki gibi oluşması gerekmektedir.



Ne varki bu yeterli değildir. Bir de eklenen **JAR** dosyası için **Native Libary Location** değerinin **set**edilmesi gerekmektedir. Bunun için aşağıdaki ekran görüntüsünde görüldüğü üzere, **SDK**' nın açılması sonucu oluşan **redist\native** klasörüne kadar inilmeli ve platforma uygun olan klasör seçilmelidir. Ben **x86** işlemcili **win32** tabanlı bir sistem de çalıştığımdan buna uygun olan klasörü seçtim*(Native klasörü altında **Linux**, **MacOSX**, **Solaris** gibi pek çok sistem için gerekli kütüphaneler bulunmaktadır)*



Buraya kadar ki işlemlerimiz sonrasında **Eclipse IDE**' sinde projemize ait olan son görünüm de aşağıdaki gibi olacaktır. Dikkat edileceği üzere **TFS SDK**' sı, **Referenced Libraries** kısmında görülmektedir.



Örnek kodlar

Şimdi kod tarafını geliştirmeye başlayabiliriz. Aslında **.Net** tarafında **Client Object Model'** in kullanımından çok da farklı bir işlem yapmamıza gerek yoktur. **TFS Client Object Model**' in nesne yapısının hemen hemen aynısı burada da inşa edilmiştir. Şimdi Program sınıfına ait kodlarımızı aşağıdaki gibi geliştirelim.

```
package tfs.clientobjectmodel.application;
import java.net.URI;
import java.net.URISyntaxException;
import com.microsoft.tfs.core.TFSTeamProjectCollection;
import com.microsoft.tfs.core.clients.workitem.WorkItem;
import com.microsoft.tfs.core.clients.workitem.WorkItemClient;
import com.microsoft.tfs.core.clients.workitem.query.WorkItemCollection;
import com.microsoft.tfs.core.httpclient.Credentials;
import com.microsoft.tfs.core.httpclient.HttpException;
public class Program {
   public static void main(String[] args)
         throws HttpException, URISyntaxException {

      System.setProperty("com.microsoft.tfs.jni.native.base-directory"
           , "C:\\Program Files\\TFS-SDK-11.0.0.1302\\TFS-SDK-
11.0.0\\redist\\native");

      URI uri=new URI("http://tfsserver:8080/tfs/defaultcollection");
      Credentials user=new com.microsoft.tfs.core.httpclient.DefaultNTCredentials();

      TFSTeamProjectCollection collection=new
TFSTeamProjectCollection(uri,user);
```

```
        WorkItemClient wiClient=collection.getWorkItemClient();

        WorkItemCollection workItems=wiClient
              .query("Select ID,Title from WorkItems " +
                    "where ([Team Project]='ARGE') order by [Work Item Type]");
        System.out.println("Active statusunde olan toplam "+
              workItems.size()+" work item bulunmustur");

        for(int i=0;i<workItems.size();i++)
        {
           WorkItem workItem=workItems.getWorkItem(i);
           System.out.println(workItem.getTitle()
                 +" "+workItem.getType().getName()
                 +" "+workItem.getID()
                 +" "+workItem.getProject().getName()
                 );
        }
     }
}
```

İlk dikkat edilmesi gereken nokta **System** özelliklerine bir **key-value** eklenmiş olmasıdır.**com.microsoft.tfs.jni.native.base-directory** isimli **key** için **C:\\Program Files\\TFS-SDK-11.0.0.1302\\TFS-SDK-11.0.0\\redist\\native** adresi **value** olarak verilmiştir. Normal şartlarda bu bildirim komut satırından **java.exe** uygulaması kullanılarak da yapılabilir. Söz konusu sistem özelliğinin bir kere set edilmesi yeterlidir. Bu bildirim ile aslında **native loader**' ın sisteme tanıtılması işlemi gerçekleştirilir. Eğer ilgili bildirim yapılmazsa, çalışma zamanında aşağıdaki hata mesajı ile karşılaşılması muhtemeldir.

```
Exception in thread "main"
java.lang.UnsatisfiedLinkError:com.microsoft.tfs.jni.internal.platformmisc.NativePlatformMisc.nativeGetEnvironmentVariable (Ljava/lang/String;)Ljava/lang/String;
    at
com.microsoft.tfs.jni.internal.platformmisc.NativePlatformMisc.nativeGetEnvironmentVariable (Native Method)
    at
com.microsoft.tfs.jni.internal.platformmisc.NativePlatformMisc.getEnvironmentVariable (NativePlatformMisc.java:134)
    at com.microsoft.tfs.jni.PlatformMiscUtils.getEnvironmentVariable
(PlatformMiscUtils.java:52)
    at
com.microsoft.tfs.core.config.httpclient.DefaultHTTPClientFactory.shouldAcceptUntrustedCertificates (DefaultHTTPClientFactory.java:288)
```

```
    at
com.microsoft.tfs.core.config.httpclient.DefaultHTTPClientFactory.configureClientParams
(DefaultHTTPClientFactory.java:324)
    at com.microsoft.tfs.core.config.httpclient.DefaultHTTPClientFactory.newHTTPClient
(DefaultHTTPClientFactory.java:137)
    at com.microsoft.tfs.core.TFSConnection.getHTTPClient (TFSConnection.java:1041)
    at com.microsoft.tfs.core.TFSConnection.getWebService (TFSConnection.java:874)
    at com.microsoft.tfs.core.config.client.DefaultClientFactory$9.newClient
(DefaultClientFactory.java:265)
    at com.microsoft.tfs.core.config.client.DefaultClientFactory.newClient
(DefaultClientFactory.java:90)
    at com.microsoft.tfs.core.TFSConnection.getClient (TFSConnection.java:1470)
    at com.microsoft.tfs.core.TFSTeamProjectCollection.getWorkItemClient
(TFSTeamProjectCollection.java:370)
    at tfs.clientobjectmodel.application.Program.main (Program.java:26)
```

Bundan sonraki kısımda ilk olarak **TFSTeamProjectCollection** tipinden bir nesne örneklendiği görülmektedir. Örnekleme sırasında ilk parametre olarak **DefaultCollection'** a ait **url** adresi verilmiştir*(Programda TFS sunucusunun kurulu olduğu makinedeki varsayılan **Team Project Collection** kullanılmaktadır)*. **URI** sınıfından yapılan bu bildirimi **Credentials** tipinden bir parametre takip etmektedir. Makineyi açan kullanıcının **Credential** bilgisi ile **TFS**' e bağlanılmak istendiğinden **DefaultNTCredentials** tipinden bir nesne örneği ele alınmıştır. Ancak bir kullanıcı adı ve şifre ile gidilmek istenirse, **UsernamePasswordCredentials** sınıfından da yararlanılabilir*(Elbette ilgili kullanıcıların söz konusu TFS sunucusuna ve koleksiyona erişebildiğini, bir başka deyişle gerekli yetkilere sahip olduğunu var sayıyoruz)*

Kod parçasında **ARGE** isimli projeye ait **Work Item**' lar çekilmeye çalışılmaktadır. Bu sebepten**WorkItemStore** servisine erişilmesi gerekmektedir. İlgili servisi kullanabilmek için**getWorkItemClient** fonksiyonu çağırılmaktadır. Bu metod geriye **WorkItemClient** tipinden bir referans döndürmektedir. Bu referans üzerinden çağırılan **query** fonksiyonuna girilen**WIQL*(WorkItem Query Language)*** sorgusu ile de **WorkItemCollection** elde edilir. Bu koleksiyon sorguya uygun bir **Work Item** içeriğini taşıyacaktır.

Pek tabi **Java** tarafında, **.Net**' te olduğu gibi **Property** isimli bir tip üyesi*(Type Member)*bulunmamaktadır. Ancak **.Net**' ten aşina olduğumuz **WorkItem** özelliklerine **get** ön ekli metodlar yardımıyla ulaşabiliriz. Örnekte **Work Item**' ın **Title** değeri için **getTitle()**, Work Item tipinin adı için**getType().getName()**, sistem de kayıtlı olan **ID** bilgisi için de **getID()** metodu kullanılmıştır.

Sonuç olarak **ARGE** isimli projedeki **Work Item**' ların **ID,Title** değerlerinin, **Work Item** tipine göre sıralanarak elde edilmesi işlemi icra edilmektedir. Uygulamanın çalışma

zamanı sonuçları aşağıda görüldüğü gibidir*(Farklı WIQL sorguları ile örneği zenginleştirmeyi denemenizi öneririm)*

Program.java

```
TFSTeamProjectCollection collection=new TFSTeamProjectCollection(uri,user);

WorkItemClient wiClient=collection.getWorkItemClient();

WorkItemCollection workItems=wiClient
        .query("Select ID,Title from WorkItems " +
                "where ([Team Project]='ARGE') order by [Work Item Type]");
System.out.println("Active statusunde olan toplam "+
        workItems.size()+" work item bulunmustur");

for(int i=0;i<workItems.size();i++)
{
    WorkItem workItem=workItems.getWorkItem(i);
    System.out.println(workItem.getTitle()
            +" "+workItem.getType().getName()
```

Problems | Javadoc | Declaration | Console

```
<terminated> Program (1) [Java Application] C:\Program Files\Java\jre6\bin\javaw.exe (Mar 14, 2013 5:44:26 PM)
Active statusunde olan toplam 21 work item bulunmustur
Bug Sample 1 Bug 4031 ARGE
Some bug Bug 4177 ARGE
On Yüz Ekran?n?n Tan?mlanmas? Product Backlog Item 2027 ARGE
Tüm harici Library' lerin referans edilmesi Product Backlog Item 2154 ARGE
ALM Dokumanlar?n?n Eklenmesi Product Backlog Item 2356 ARGE
Product Backlog Item 1 Product Backlog Item 4013 ARGE
Product Backlog 2 Product Backlog Item 4015 ARGE
Mü?teri karakteristi?i olu?turma Product Backlog Item 6538 ARGE
Model ekranlar?n tasar?m? Task 2155 ARGE
View kodlar?n?n yaz?m? Task 2156 ARGE
Controller kodlar?n?n yaz?m? Task 2157 ARGE
3rd Party Kutuphanelerin Referans Edilmesi Task 2158 ARGE
WCF Kodlama Standartlar?n?n Eklenmesi Task 2358 ARGE
Task A Task 4014 ARGE
Task X Task 4016 ARGE
Task Y Task 4017 ARGE
Deneme Task 4175 ARGE
Mü?teri Login Kontrolü Task 6518 ARGE
Mü?teri önceki bakt??? ürünlerin çekilmesi Task 6519 ARGE
Mü?teri Sistem de geziniyor mu? Task 6539 ARGE
Mü?terinin son 10 hareketini almak. Task 6540 ARGE
```

Senaryoyu işlettiğim sistemde test amaçlı olarak kullandığım **ARGE** isimli **Team Project**, **Scrum 2.0** şablonunu kullanmaktaydı. Bu nedenle **Product Backlog Item, Task** ve **Bug** gibi **Work Item**öğelerini barındırmaktadır.

Biraz Daha

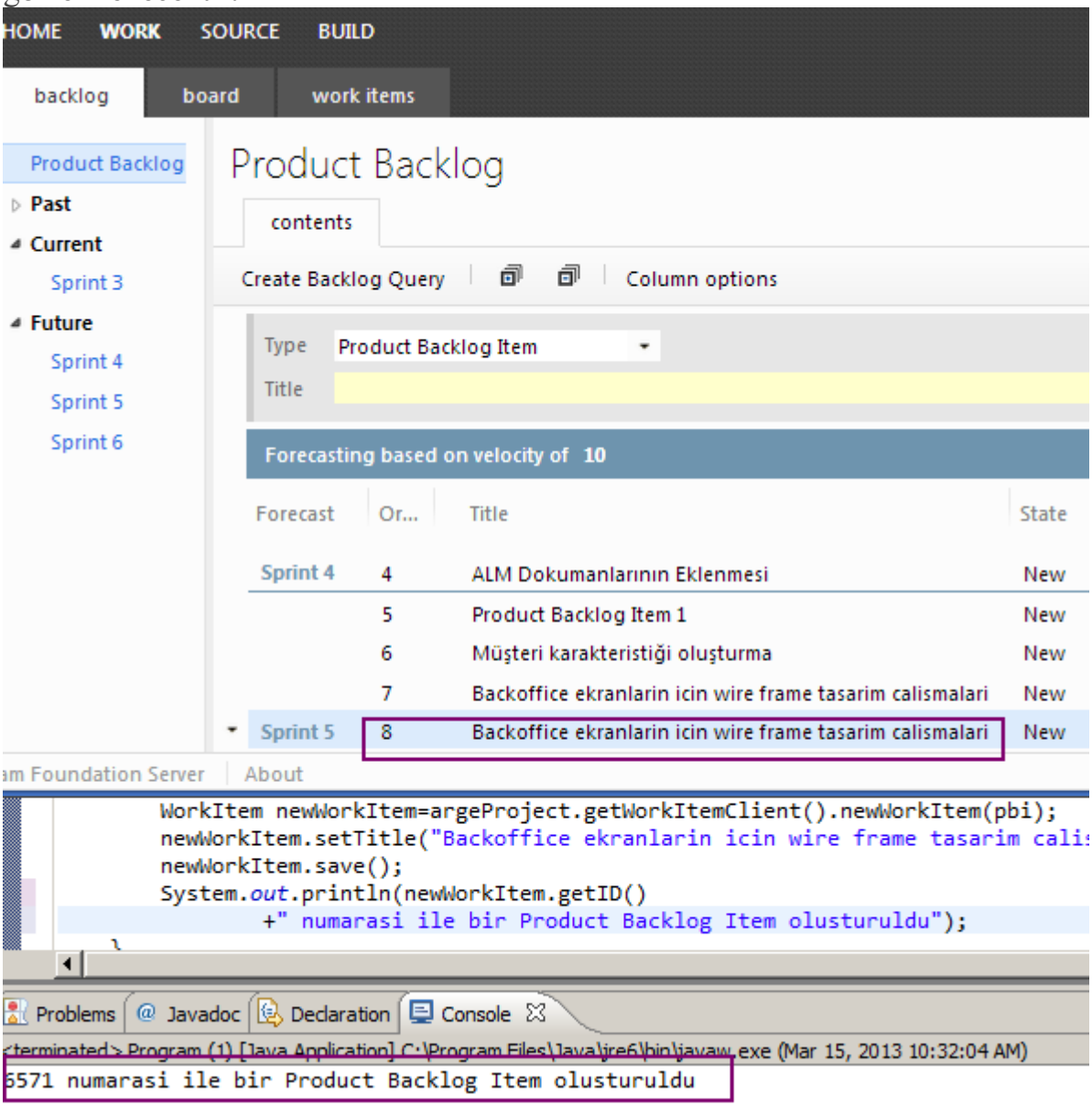
Dilerseniz örnek kod parçasını biraz daha geliştirmeye çalışalım. Örneğin yeni bir **Work Item** nasıl eklenir ona bakalım. **.Net** tarafından biraz farklı olarak yeni bir **WorkItem** nesnesinin örneklenmesi için **Project** sınıfı

üzerinden **WorkItemClient** referansına ulaşışması ve **newWorkItem** metodunun çağırılması gerekmektedir*(Yani **WorkItem** sınıfını doğrudan bir **yapıcı metod-Constructor** ile örnekleyemiyoruz)* Aynen aşağıdaki kod parçasında olduğu gibi.

```
package tfs.clientobjectmodel.application;
import java.net.URI;
import java.net.URISyntaxException;
import com.microsoft.tfs.core.TFSTeamProjectCollection;
import com.microsoft.tfs.core.clients.workitem.WorkItem;
import com.microsoft.tfs.core.clients.workitem.WorkItemClient;
import com.microsoft.tfs.core.clients.workitem.project.Project;
import com.microsoft.tfs.core.clients.workitem.wittype.WorkItemType;
import com.microsoft.tfs.core.httpclient.Credentials;
import com.microsoft.tfs.core.httpclient.HttpException;
public class Program {
   public static void main(String[] args)
         throws HttpException, URISyntaxException {

      URI uri=new URI("http://tfsserver:8080/tfs/defaultcollection");
      Credentials user=new com.microsoft.tfs.core.httpclient.DefaultNTCredentials();

      TFSTeamProjectCollection collection=new TFSTeamProjectCollection(uri,user);

      WorkItemClient wiClient=collection.getWorkItemClient();

      Project argeProject=wiClient.getProjects().get("ARGE");
      WorkItemType pbi=argeProject.getWorkItemTypes().get("Product Backlog Item");

      WorkItem newWorkItem=argeProject.getWorkItemClient().newWorkItem(pbi);
      newWorkItem.setTitle("Backoffice ekranlarin icin wire frame tasarim calismalari");
      newWorkItem.save();
      System.out.println(newWorkItem.getID()+" numarasi ile bir Product Backlog Item olusturuldu");
   }
}
```

Dikkat edilmesi gereken noktalardan birisi de **newWorkItem** metoduna parametre olarak üretilmek istenen **work item** tipinin verilmesidir. Bu bildirim için **WorkItemType** sınıfına ait bir nesne örneği kullanılmaktadır. **WorkItemType** üretimi için projeye ait nesne örneği üzerinden önce var olan**WorkItem** tiplerinin elde edilmesi işlemi gerçekleştirilmiş,

ardından ise **Product Backlog Item** tipi çekilmiştir. Bu son derece mantıklıdır, nitekim ilgili projenin şablonu*(ki örneğimizde Scrum 2.0 söz konusudur)* tarafından kullanılan **Work Item** tipleri ne ise, onlara ait **Work Item** nesneleri örneklenebilir. Üretilen **Product Backclog Item** için bir **Title** değeri verilmiş ve sonrasında **Save**metodu kullanılarak kayıt işlemi gerçekleştirilmiştir. Uygulama çalıştırıldığında hem **Console**penceresinden hem de ilgili projeye ait **Backlog**' da bir öğenin oluşturulduğu gözlemlenecektir.



Görüldüğü üzere **Team Foundation Server Client Object Model**' in, **Java** tarafında kullanılması da son derece kolaydır. Örnek, tahmin edeceğiniz üzere **Hello World** formatındadır. Ancak daha önceki yazılarımızı baz alarak, .**Net** tarafındaki **Client Object Model** kodlarını, **Java** tarafına taşımayı deneyebilirsiniz. **Kim demiş Java, Microsoft' u, Microsoft' ta Java' yı umursamıyor diye** Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

TFS Client Object Model ile Word Entegrasyonu

Cuma, 12 Nisan 2013 10:15 by bsenyurt

Merhaba Arkadaşlar,
Geçtiğimiz gün **National Geographic**kanalında **Mega Fabrikalar'** ı seyretme fırsatı buldum.
Amerikalı **Dodge** firması efsane**Challanger**' ı yeniden üretmekteydi. Konu bu üretimin gerçekleştirildiği mega fabrikaydı.
Robotların, gelişmiş endüstürinin ve insan gücünün bir araya geldiği fabrika, sadece 24 saat içerisinde üretim hattından mükemmel spor arabalar çıkmaktaydı. Üstelik motor bloğu da kıtanın bir diğer ucundan geliyordu. Her ne kadar mükemmele yakın bir üretim bandı da olsa, akıllı bilgisayarlar üretim sürecindeki her adımı gözlemliyor ve bir istisna olması halinde bandı durduruyordu. O da yetmiyor pek çok noktada usta insanlar devreye giriyor ve gerekirse üretim bandını kendi insiyatifleri ile durduruyorlardı. Tabi burada yazarak anlatmak çok zor o yüzden mutlaka seyredin derim. Aslında belgeseli izlerken en çok da şunu düşündüm **"Böyle muazzam yapılar nasıl oluyor da inşa ediliyor? İnsan aklı ne kadar muazzam ki her ayrıntıyı düşünüyor, düşünmeye çalışıyor"**
Derken bilgisayarımın başına döndüm ve kendi kendime şöyle dedim **"E illa ki bir Hello World uygulamaları vardır yahu"**
Bu yazımız ile birlikte **Team Foundation Server** maceralarımıza devam etmeye çalışıyor olacağız. Yeni bölümüzde **Client Object Model**' i bir **Word** uygulaması içerisinde kullanmaya çalışacağız ve çok basit olarak **Work Item** öğelerini nasıl kayıt edebileceğimizi göreceğiz.
Team Foundation Server dünyası ile ilişkili önceki yazılarıma aşağıdaki adreslerden ulaşabilirsiniz.

- **TFS Client Object Model için Hello World**
- **TFS Web Services ve Kullanımları**
- **Heryerden TFS Kullanabilmek**

Çalışmakta olduğumuz **Team Project**' in **süreç şablonu*(Process Template)*** ne olursa olsun***(Scrum, MSF, CMMI)*** giriş yapılan öğeler **Work Item** olarak düşünülmektedir. Örneğin **Scrum**felsefesi göz önüne alındığında **Product Backlog Item**, **Task**, **Bug**, **Test Case** ve **Impediment**birer **Work Item**' dır. **CMMI** şablonuna bakıldığında
ise **Requirement**, **Task**, **Bug**, **ChangeRequest**, **Issue**, **Review**, **Risk** ve **Test Case** birer **Work Item** olarak düşünülmektedir.
Pek tabi bu **Work Item** öğeleri **Client Object Model** tarafında da birer tip olarak ele alınabilirler.**Client Object Model** bilindiği üzere, **Team Foundation Server** ın dış dünya ile olan iletişiminde ve özellikle çevre araçlar ile olan entegrasyonunda önemli bir yere

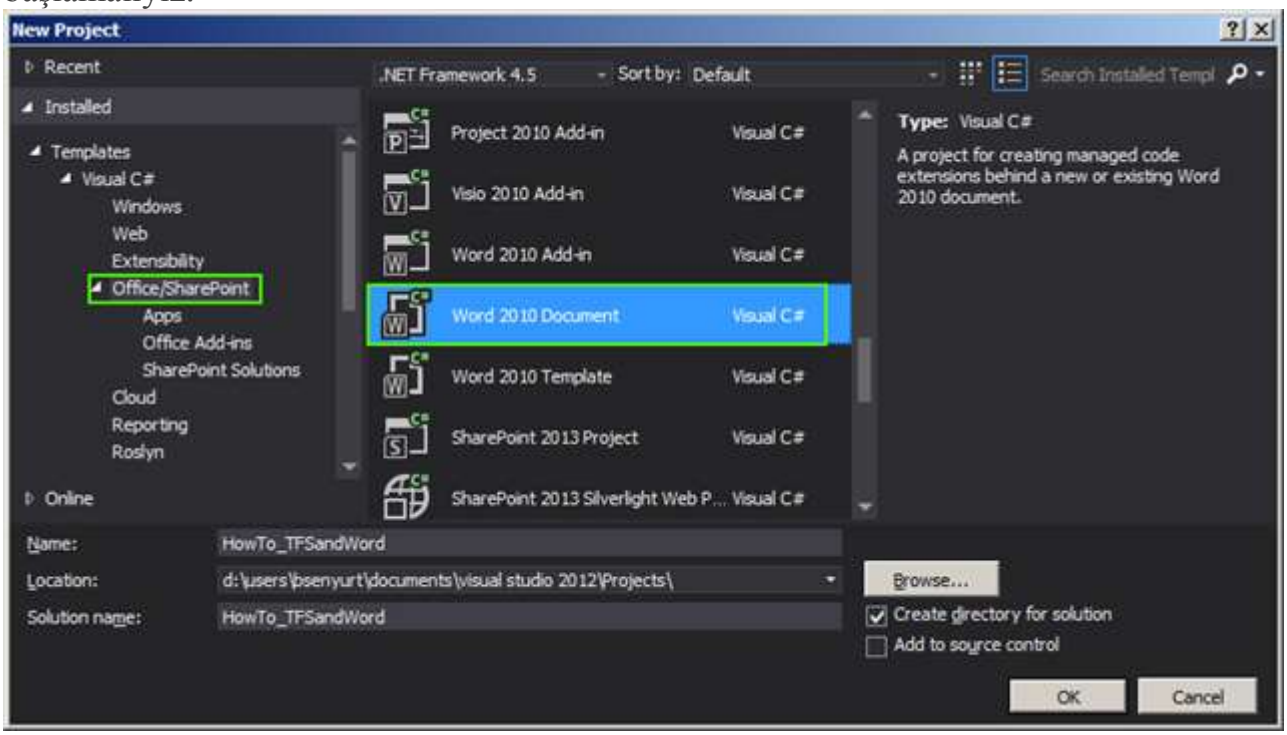
sahiptir. Dilerseniz örneğimize geçelim ve adım adım ilerleyerek konuyu anlamaya çalışalım.

Senaryo

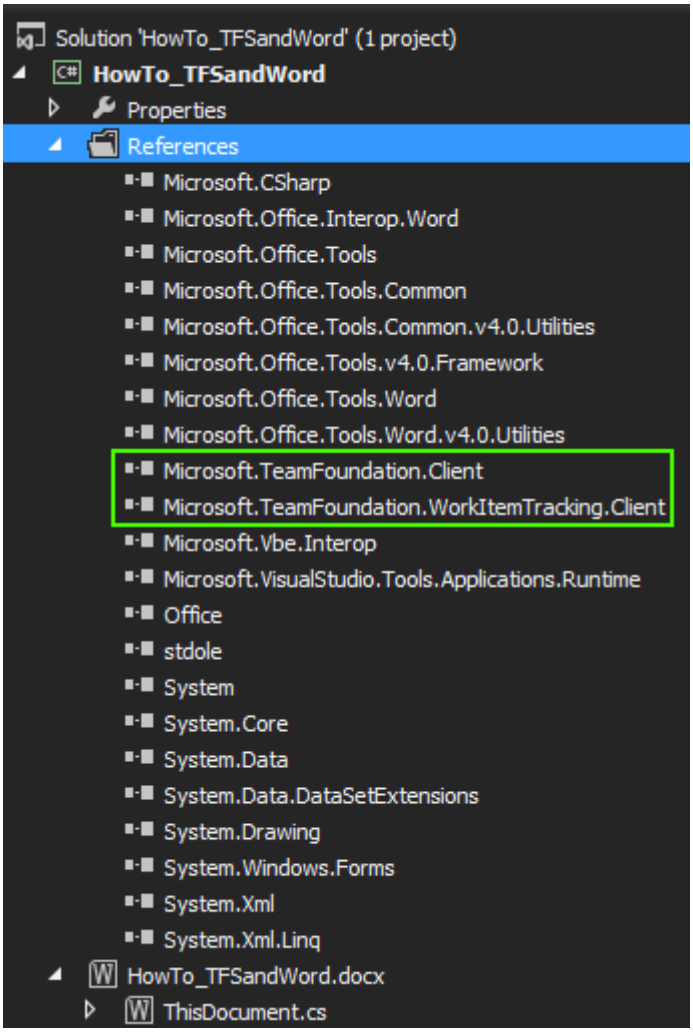
Elimizde **Scrum 2.0** formatında oluşturulmuş bir **Team Project** var.
Amacımız **Word** belgesi içerisinden, bir **Product Backlog Item** ve buna bağlı iki **Task** öğesinin kayıt edilmesini sağlamak. Olayı son derece basit bir biçimde ele alacağımızdan **Product Backlog Item** ve **Task** öğeleri için sadece **Title** ve **Description** içeriklerin yer veriyor olacağız. Kritik noktalardan birisi de, **Task**öğeleri ile **Product Backlog Item** arasında **Parent-Child** ilişkinin kurulmasıdır. Yani **Task** öğeleri ilgili **Product Backlog**' a bağlı olacaklardır.

Ön Hazırlıklar ve Doküman Tasarımı

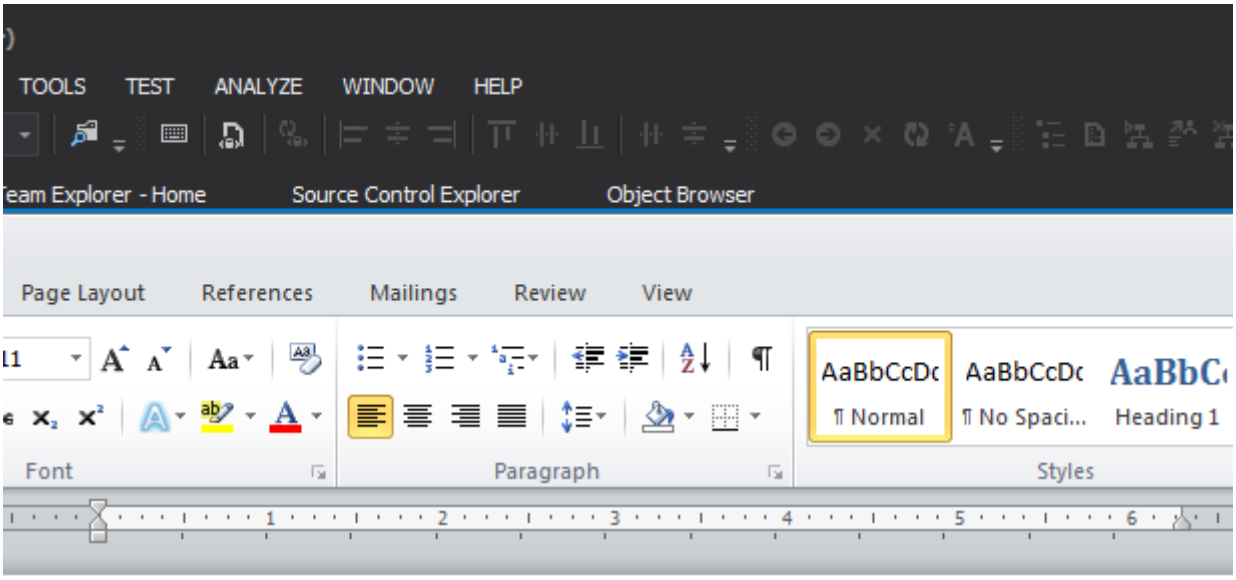
İşe ilk olarak **Visual Studio 2012** ortamında bir **Word 2010 Document** projesi oluşturarak başlamalıyız.



Bunun için yukarıdaki ekran görüntüsünde olduğu gibi **Office/Sharepoint** sekmesinde yer alan**Word 2010 Document** şablonunun seçilmesi yeterlidir. **Word** uygulaması içerisinde **Team Foundation Server Client Object Model** kullanılacağından ilgili **Assembly** referanslarının projeye eklenmesi de gerekmektedir. *(Bu örnek için **Microsoft.TeamFoundation.Client** ve**Microsoft.TeamFoundation.WorkItemTracking.Client** dll' lerini eklemeliyiz)*



Word dokümanının tasarımını ise aşağıdaki ekran görüntüsünde yer aldığı gibi yapabiliriz.



Product Backlog Item	
Title	Description
Click here to enter text.	Click here to enter text.
Tasks	
Task Title	Task Description
Click here to enter text.	Click here to enter text.
Click here to enter text.	Click here to enter text.

Oldukça sade bir tasarımımız var. Örneği mümkün olduğunca basit seviyede tutmamız önemli.**Product Backlog Item** ile **Task** öğelerine ait **Title** ve **Description** girişleri için**PlainTextContentControl** bileşeninden yararlanılmaktadır. Ve,

Kod

Kullanıcı **Save** işlemini icra ettiğinde*(Ribbon kontrolüne basabilir, Ctrl+Save yapabilir vb)*, girdiği veri içeriğinin **TFS** tarafındaki ilgili projenin **backlog**' una yazılması gerekmektedir. Bu nedenle odak noktası dokümanın **BeforeSave** olay metodudur.
Lakin **Client Object Model** nesne referanslarının nasıl kullanıldığına da dikkat edilmelidir.

```
using Microsoft.Office.Tools.Word;
using Microsoft.TeamFoundation.Client;
using Microsoft.TeamFoundation.WorkItemTracking.Client;
using System;
namespace HowTo_TFSandWord
{
    public partial class ThisDocument
```

```
{
    #region Global değişkenler
    Uri tfsAddress = new Uri("http://tfsserver:8080/tfs/defaultcollection");
    TfsTeamProjectCollection collection = null;
    WorkItemStore store = null;
    Project argeProject = null;
    WorkItemType witBacklogItem = null;
    WorkItemType witTask = null;
    WorkItemLinkTypeEnd linkTypeEnd = null;
    #endregion Global değişkenler
    private void InitializeTFSComponents()
    {
        collection = new TfsTeamProjectCollection(tfsAddress);
        store = collection.GetService<WorkItemStore>();
        argeProject = store.Projects["ARGE"];
        witBacklogItem = argeProject.WorkItemTypes["Product Backlog Item"];
        witTask = argeProject.WorkItemTypes["Task"];
        linkTypeEnd = store.WorkItemLinkTypes.LinkTypeEnds["Parent"];
    }
    void ThisDocument_BeforeSave(object sender, SaveEventArgs e)
    {
        #region Yeni Bir Product Backlog Item Oluşturmak
        int createdBacklogItemId=CreateBacklogItem(textBacklogTitle.Text,
textBacklogDescription.Text);
        CreateTask(createdBacklogItemId, textTask1Title.Text,
textTask1Description.Text);
        CreateTask(createdBacklogItemId, textTask2Title.Text,
textTask2Description.Text);
        #endregion Yeni Bir Product Backlog Item Oluşturmak
    }
    private int CreateBacklogItem(string title,string description)
    {
        WorkItem newBacklogItem = new WorkItem(witBacklogItem);
        newBacklogItem.Title = title;
        newBacklogItem.Description = description;
        newBacklogItem.Save();
        return newBacklogItem.Id;
    }
    private void CreateTask(int createdBacklogItemId,string title,string description)
    {
        WorkItem newTask = new WorkItem(witTask);
```

```
            newTask.Title = title;
            newTask.Description = description;
            newTask.WorkItemLinks.Add(new
WorkItemLink(linkTypeEnd,  createdBacklogItemId));
            newTask.Save();
        }
        #region VSTO Designer generated code

        private void InternalStartup()
        {
            this.Startup += new System.EventHandler(ThisDocument_Startup);
            this.Shutdown += new System.EventHandler(ThisDocument_Shutdown);
            this.BeforeSave += ThisDocument_BeforeSave;
            InitializeTFSComponents();
        }
        #endregion
        private void ThisDocument_Startup(object sender, System.EventArgs e)
        {
        }
        private void ThisDocument_Shutdown(object sender, System.EventArgs e)
        {
        }
    }
}
```

Öncelikli olarak **TFS** sunucusuna ve ilgili koleksiyona bağlanılması gerekir. Bu amaçla**TfsTeamProjectCollection** sınıfından yararlanılmaktadır. Dikkat edileceği üzere **yapıcı metoda*(Constructor)*** parametre olarak **Default Collection** için kullanılabilecek adres verilmiştir.**Work Item'** lar üzerinde **CRUD** işlemlerini icra edebilmek için **WorkItemStore** servisine ulaşılması gerekmektedir. Bu sebepten **collection** değişkeni üzerinden **GetService** fonksiyonuna başvurulmuştur. Çok doğal olarak **ARGE** isimli örnek **Team Project** içerisine **Work Item** eklenmesi istenmektedir. Bu nedenle ilgili proje referansı, **store** değişkeni üzerinden **Projects** koleksiyonuna erişilerek elde edilmektedir.

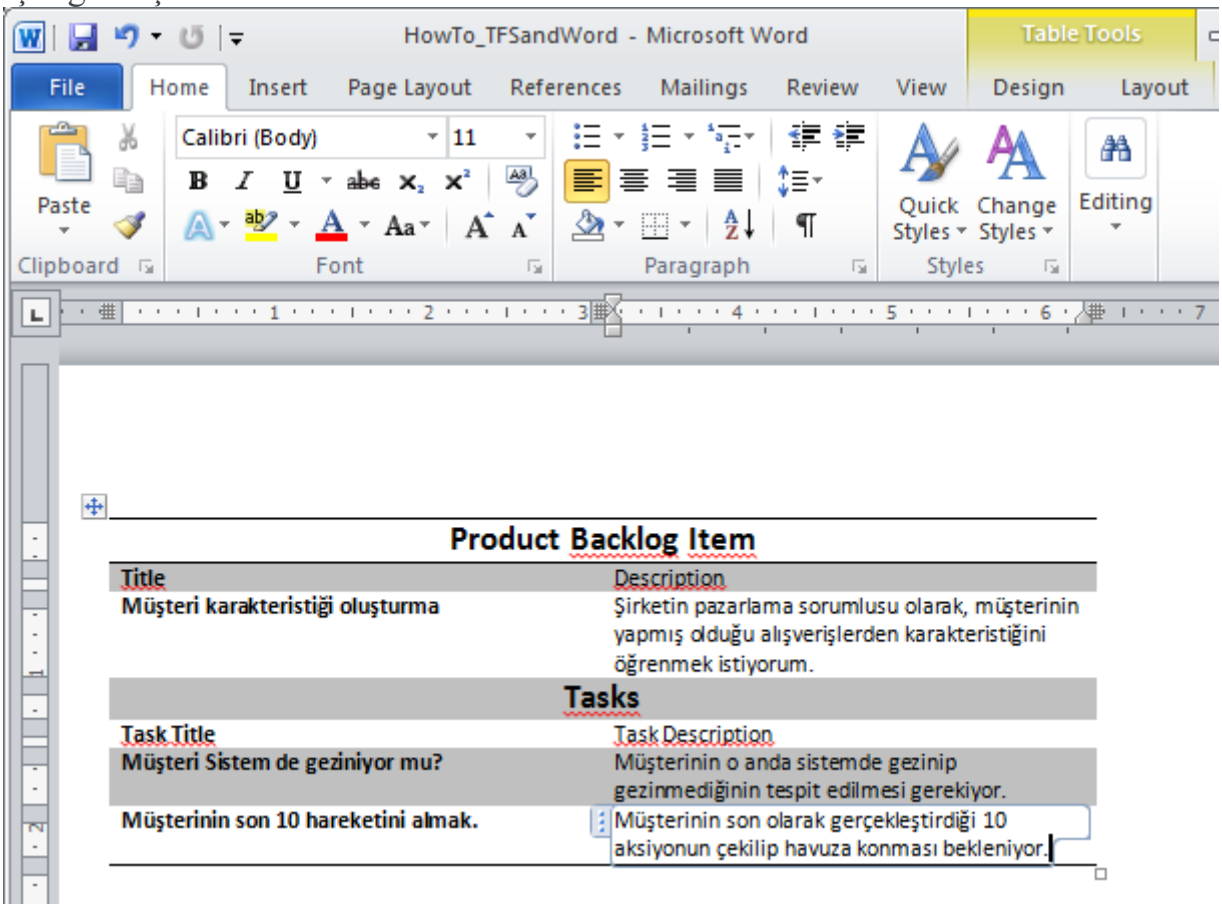
Gelelim **Work Item** oluşturma kısımlarına.

Product Backlog Item için **CreateBacklogItem**, **Task** içinse **CreateTask** isimli metodlar kullanılmaktadır. Aslında her ikisi de içerisinde bir **WorkItem** nesnesini örneklemekte ve ilgili özelliklerini **set** etmektedir. Ancak önemli bir fark vardır. Bir **WorkItem** nesne örneği üretilirken,**WorkItemType** tipinden bir parametre verilir. Bu parametre, ilgili **Work Item** öğesinin **ne** olacağını belirtir. Bir **Task** mı, bir **Product Backlog Item** mı, bir **Bug** mı vs. Bundan sonraki kısımlarda yer alan özellikler temel anlamda ortak sayılabilir. **Title** ve **Description** özelliklerine ilgili

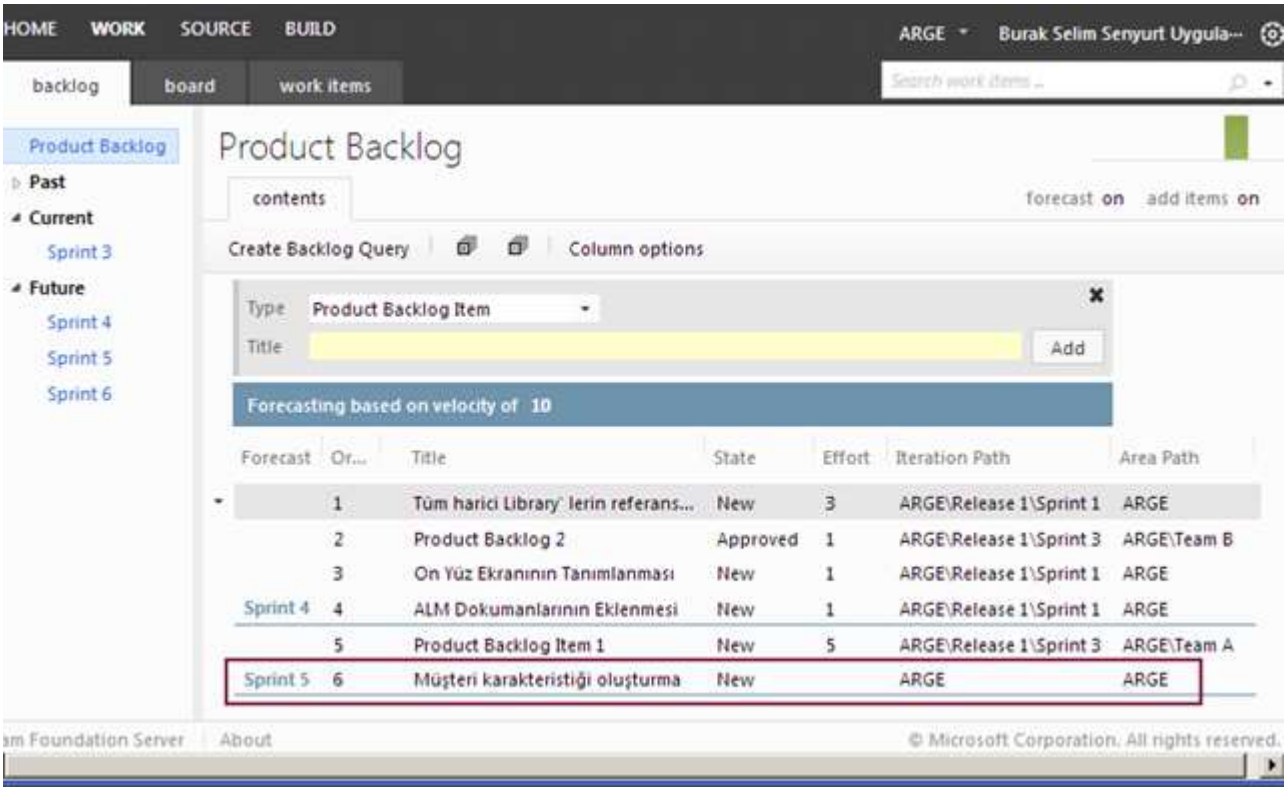
değerler **set**edildikten sonra, **WorkItem** nesne örneği üzerinden **Save** işleminin icra edilmesi yeterlidir.
WorkItem nesneleri kayıt edildiğinde **Id** özellikleri de, sunucu tarafından verilen değer ile otomatik olarak doldurulur. Nitekim **Parent-Child** ilişkinin oluşturulması noktasında, **Parent Work Item**' ın**Id** değerinin bilinmesi önemlidir. Bu sebepten **CreateBacklogItem** metodu geriyer üretilen **WorkItem Id** değerini döndürmektedir. Bu değer **CreateTask** fonksiyonu için bir girdidir. **CreateTask**metodunun en kritik noktası ise **WorkItemLinks** koleksiyonuna **Parent** tipte bir bağlantı tanımının eklenmesidir. Bunun için **WorkItemLinkTypeEnd** ve **WorkItemLink** tiplerinden yararlanılmıştır. Parametre olarak gelen integer değer bu senaryo da **Parent** olacak **Product Backlog Item'** ı işaret etmektedir.

Testler

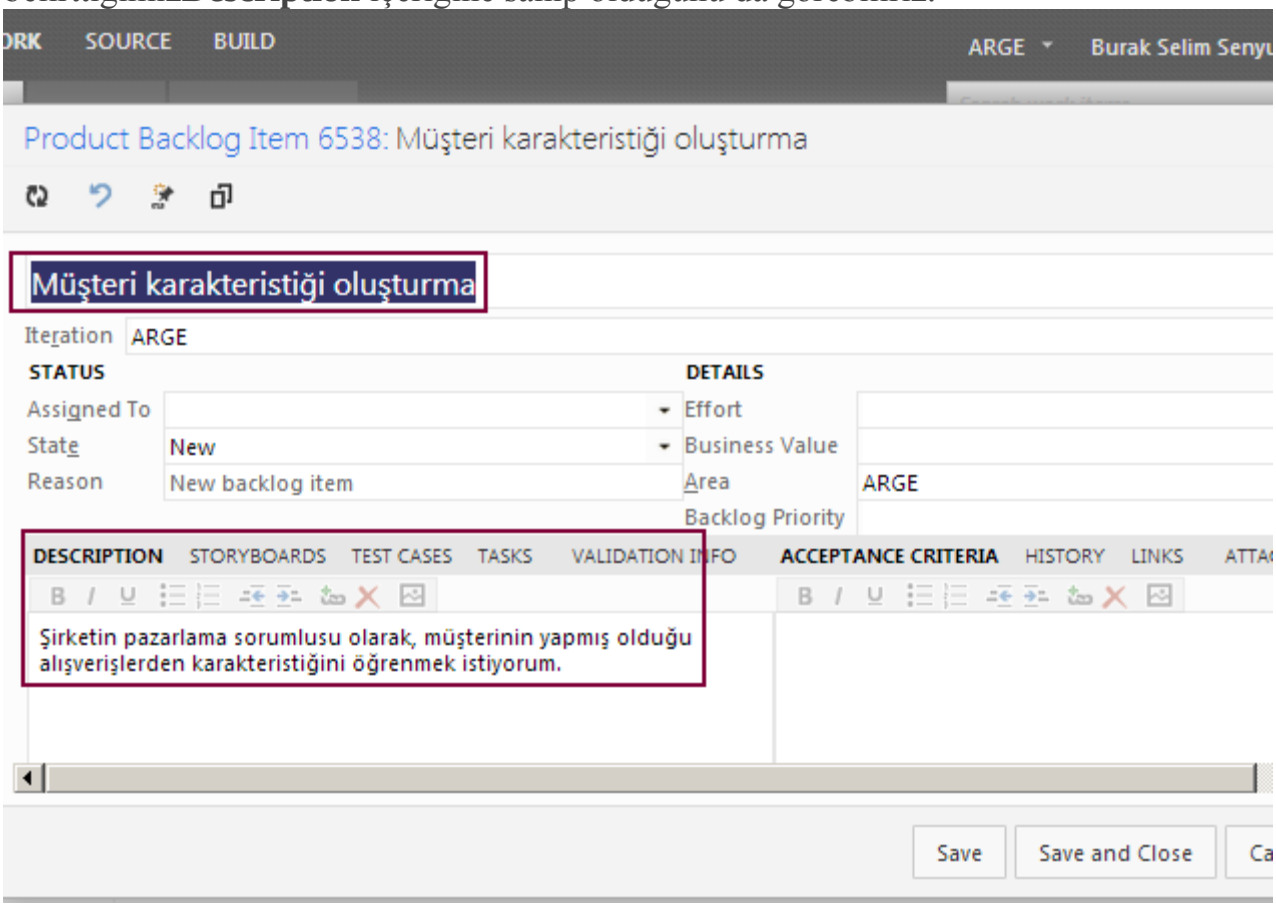
Artık uygulamayı test edebiliriz. Örnek olarak ben aşağıdaki ekran görüntüsünde yer alan içeriği oluşturdum.



Şu aşamda **Save** işlemini icra ettiğimizde **TFS** tarafında aşağıdaki içeriklerin oluştuğuna şahit olabiliriz. Dikkat edileceği üzere **"Müşteri karakteristiği oluşturma"** isimli bir **Product BacklogItem** oluşturulmuş ve varsayılan olarak o anki güncel **Sprint**' e ilave edilmiştir.

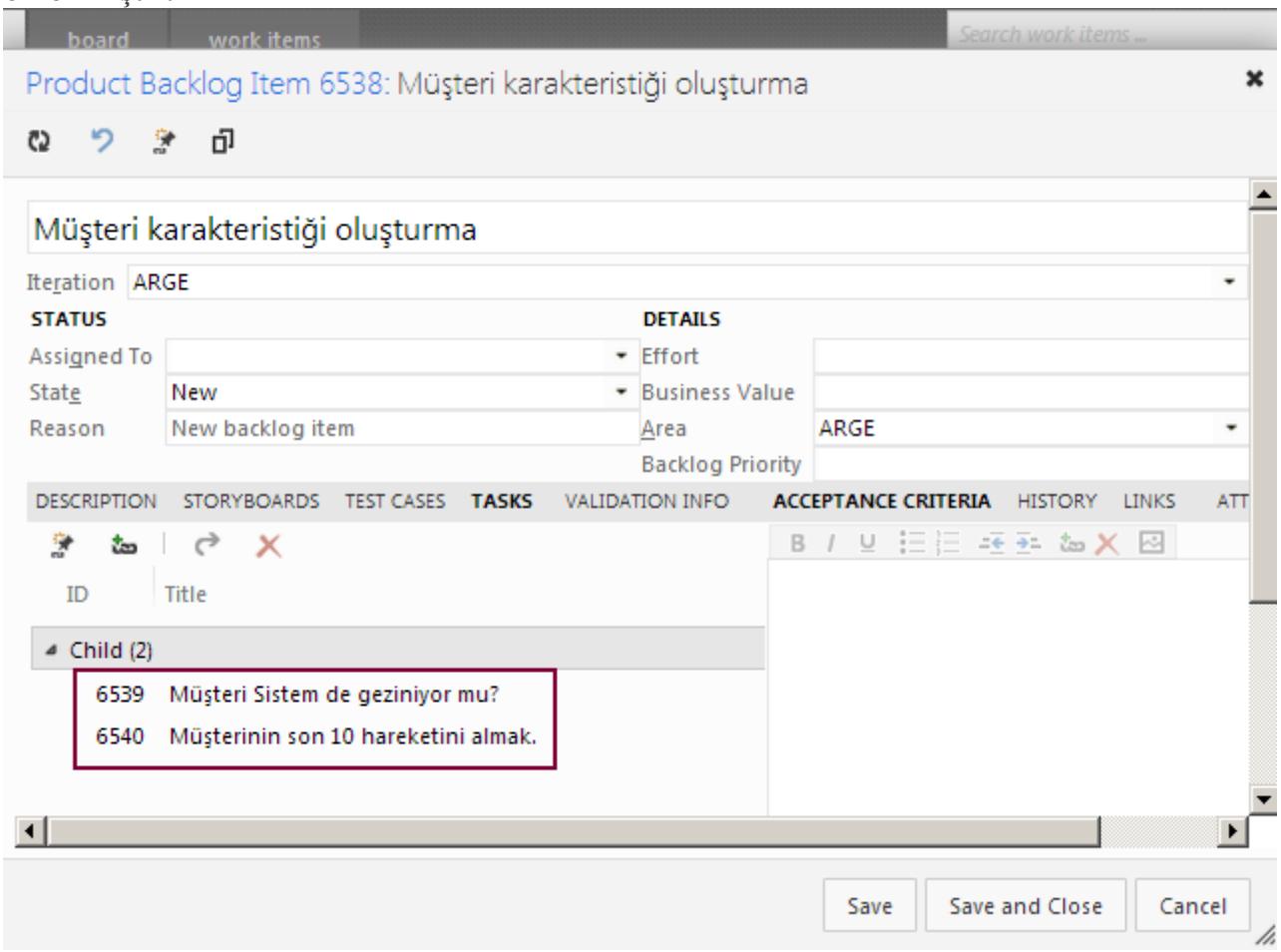


Product Backlog Item açıldığında bir **Id** değeri aldığını ve **Word** dosyasında belirttiğimiz**Description** içeriğine sahip olduğunu da görebiliriz.

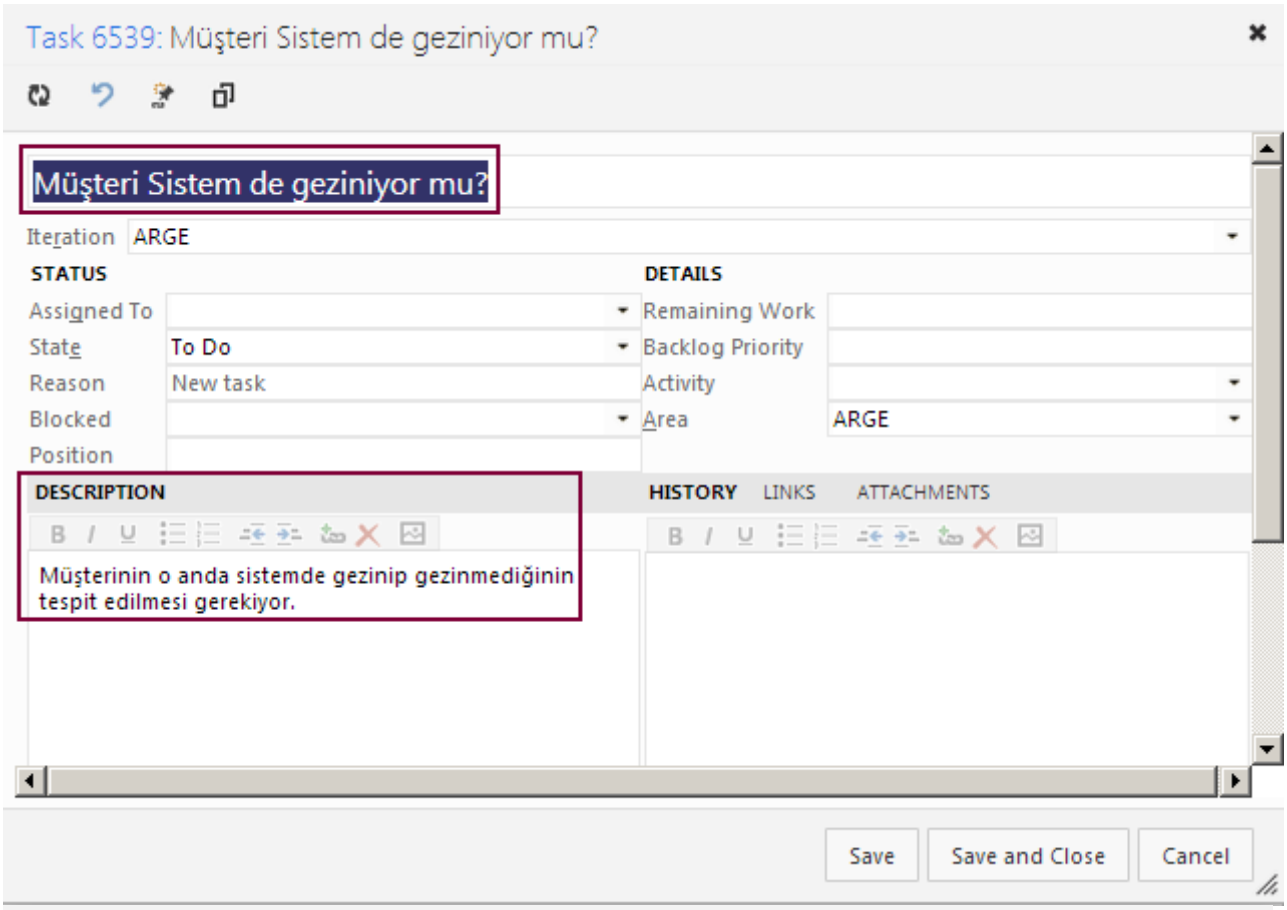


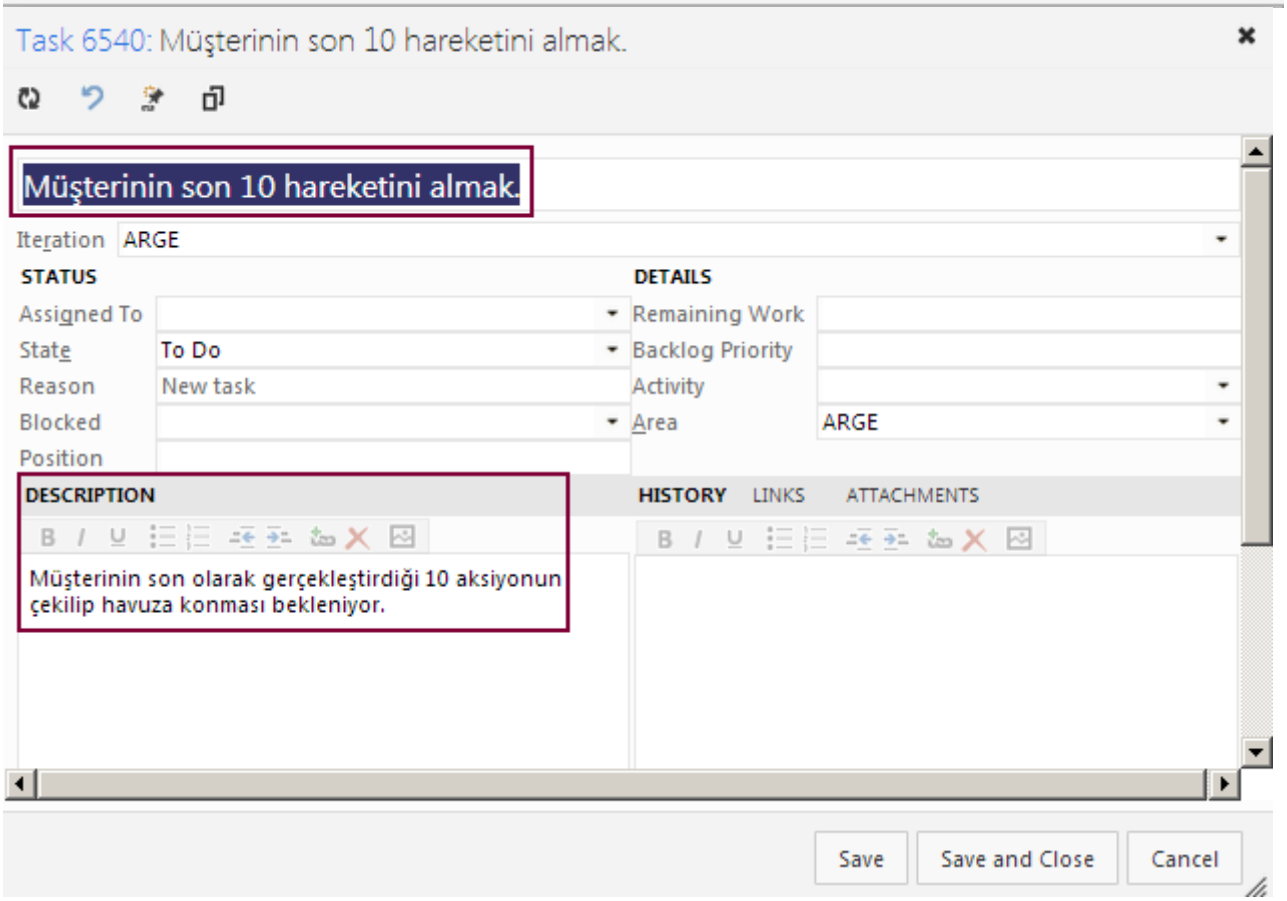
Hatta **Tasks** kısmına geçtiğimizde, **Child** olarak bağladığımız **Work Item** öğelerini de görebiliriz. Dikkat edileceği

üzere **6539** ve **6540** numaralı **Task** öğeleri **Child** olarak **6538** numaralı **ProductBacklog**' a eklenmiştir.



Pek tabi ki bu **Task** örneklerine çift tıklandığında, **Word** dosyasında belirttiğimiz **Title** ve**Description** bilgilerine sahip olduklarını görebiliriz.


Task 6539: Müşteri Sistem de geziniyor mu?
Müşteri Sistem de geziniyor mu?
Iteration ARGE
STATUS
Assigned To
State To Do
Reason New task
Blocked
Position
DETAILS
Remaining Work
Backlog Priority
Activity
Area ARGE
DESCRIPTION
HISTORY LINKS ATTACHMENTS
Müşterinin o anda sistemde gezinip gezinmediğinin tespit edilmesi gerekiyor.
Save
Save and Close
Cancel


Task 6540: Müşterinin son 10 hareketini almak.
Müşterinin son 10 hareketini almak.
Iteration ARGE
STATUS
Assigned To
State To Do
Reason New task
Blocked
Position
DETAILS
Remaining Work
Backlog Priority
Activity
Area ARGE
DESCRIPTION
HISTORY LINKS ATTACHMENTS
Müşterinin son olarak gerçekleştirdiği 10 aksiyonun çekilip havuza konması bekleniyor.
Save
Save and Close
Cancel

Her şey çok kolay görünüyor değil mi? Ama pek çok eksik ve tamamlanması gereken iş var. Bu işlerin tamamlanması da önemli bir development eforunu gerektirmekte.

Eksikler

Bu nokta da örneğimizin aslında sadece bir **Hello World** olduğunu ifade etmem gerekiyor. Sizin de fark edeceğiniz gibi bazı eksik noktalar var. Örneğin,

- Kullanıcı doküman içeriğini doldurmadan da göndermeyi deneyebilir. Bu durumda bir tedbir almak faydalı olacaktır. Hatta **TFS**' in böyle bir durumda vereceği olası **Exception** tepkisine karşı bir geliştirme yapılmalıdır.
- Akıllı bir **Save** mekanizması gerekebilir. Aynı **Title'** a sahip bir **Work Item** içeriğinin oluşturulmasının önüne kolayca geçilebilir*(WorkItem listesini çek, Title' larını karşılaştır)* Ancak bire bir eşleşmeyen fakat aynı anlama gelebilen **Title'** lar var ise, belki kullanıcıya bir pencere ile bildirimde bulunulup aralarında seçim yapması istenebilir.
- Örnekte sadece **Title** ve **Description** alanları doldurulmuştur. Oysaki bir **Product Backlog Item** veya**Task** için set edilmesi gereken daha pek çok özellik vardır. Söz gelimi kapasite planlaması ve **Velocity'**nin çıkmasında önem arz eden **Product Backlog Item'** ın **Effort** ve **Business Value** değerleri doldurulmamıştır. Benzer şekilde **Task'** lar için de bir atama işlemi yapılmamıştır. Yani ilgili **Task** kime atanmıştır bilgisi eksiktir. *(Bilindiği üzere **Sprint** planlama toplantılarında bu tip öğeler son derece dikkatli bir şekilde tespit edilmekte ve doldurulmaktadır)*
- Öğeler varsayılan olarak kök **Area** altına atanmıştır. Ancak farklı bir **Area** içerisinde tutulması da istenebilir. Dolayısıyla bu bilginin dosya daha açılmadan çekilmesi ve hatta dokümanda belki de bir**ComboBox** ile gösterilerek seçtirilmesi düşünülmelidir*(**Area**' ya benzer şekilde **Sprint** ve hatta **Iteration**seçimler de yaptırılabilir)*
- Örnekte sadece bir **Product Backlog Item** ve buna bağlı iki **Task** girilmesi senaryosuna yer verilmiştir. Oysa n adet giriş yapılabilir. Dolayısıyla daha dinamik bir içerik giriş alt yapısı hazırlanmalıdır*(Bu noktada**Excel**' in **TFS** ile olan varsayılan entegrasyonuna bakmanızı öneririm)*
- Senaryomuzda **ARGE** isimli bir **Team Project** kullanılmaktadır. Pek ala **Word** dosyası açılırken kullanıcıdan bir proje seçmesi istenebilir. Bu doğal olarak bir arabirimi gerektirmektedir*(Hatta bu arabirimde hangi **Team Project'** in hangi **Sprint'** ine içerik girileceği bilgisi dahi sorulabilir)*

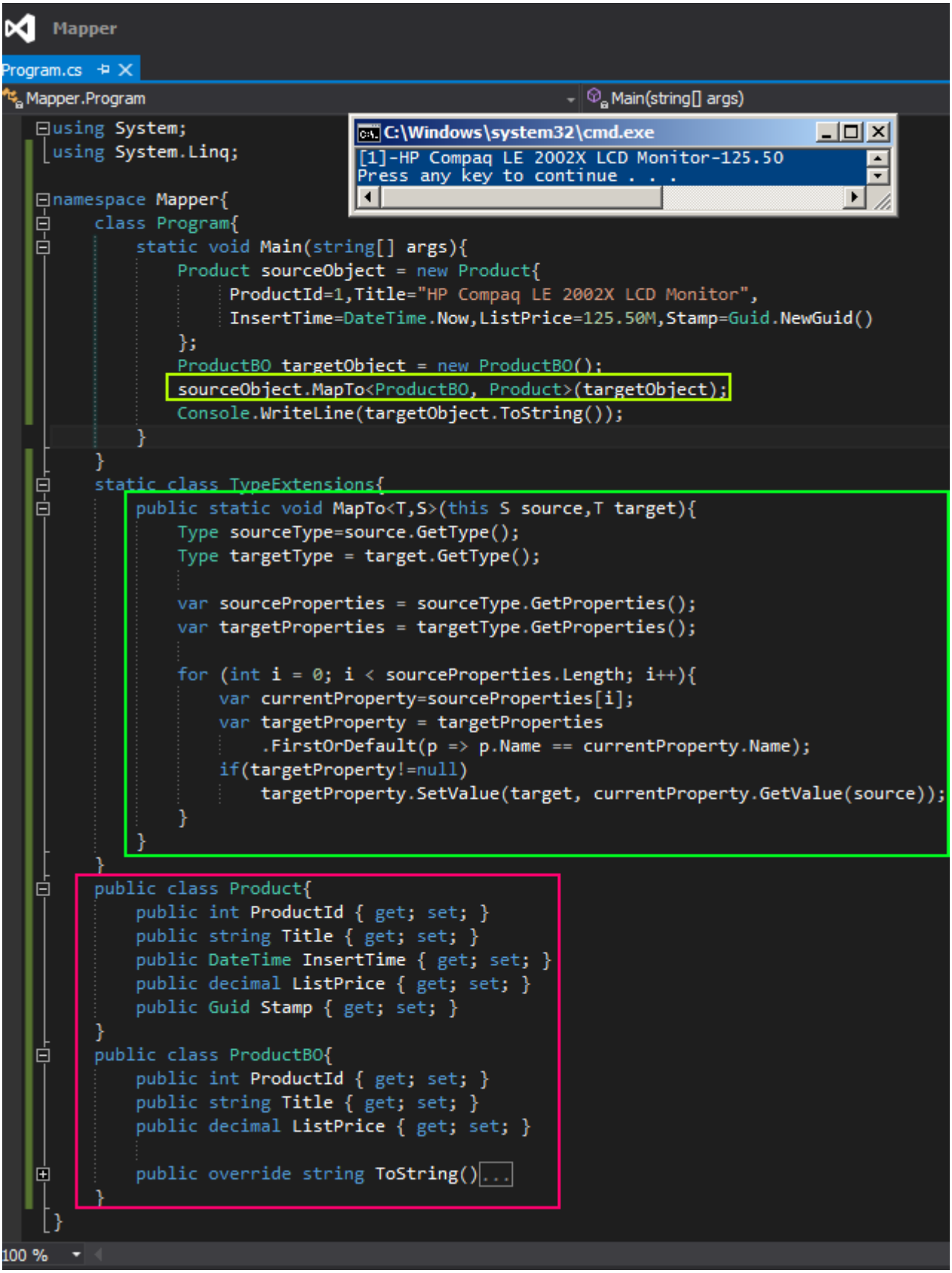
ve benzeri pek çok eksik bulunabilir. Ancak amacımıza ulaştığımızı ve her zaman ki gibi kapıyı sadece araladığımızı, ardına kadar açmak için sizin çaba sarf etmeniz gerektiğini hatırlatmak isterim. Böylece geldik bir yazımızın daha sonuna. **Team Foundation Server** ile ilişkili araştırmalarıma fırsat buldukça devam ediyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HowTo_TFSandWord.zip (208,21 kb)

Tek Fotoluk İpucu 99–Tipler Arası Property Eşleştirme

Salı, 26 Mart 2013 12:35 by bsenyurt

Merhaba Arkadaşlar,
Özellikle **ORM** araçlarının ele alındığı uygulamalarda bazen **Entity** içeriklerini sistem içerisinde dolaştıran ve nispeten daha az sayıda özellik*(asıl ihtiyaç olunanları)* taşıyan tiplere ihtiyaç duyarız.**Business Object** veya **Data Transfer Object** gibi isimler de verirler bunlara.
Diyelim ki buna benzer bir senaryo da bir A tipini, nispeten daha az sayıda özelliği olan Business Object karşılığına çevrimek istiyorsunuz. Hatta bu senaryo birden fazla tip için de söz konusu olabilir. Bİr sürü **Entity** tipiniz ve karşılığı olan **Business Object** sınıfınız olduğunu düşünün. Ne yaparsınız? Aşağıdaki ip ucu bir fikir verebilir mi?

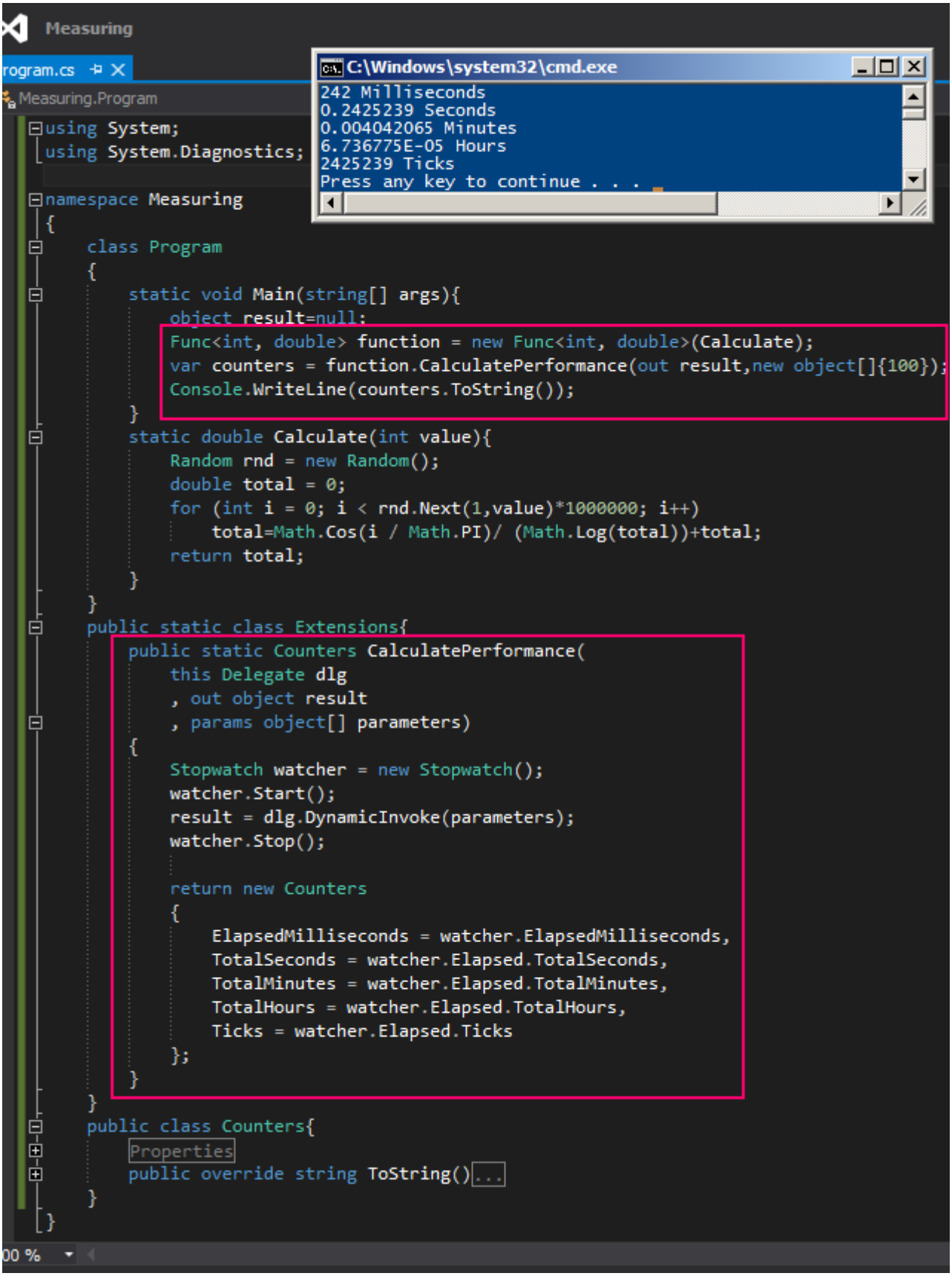


Bir başka ip ucunda görüşmek dileğiyle.

Tek Fotoluk İpucu 98–Stopwatch ile Performans Ölçümü

Salı, 26 Mart 2013 12:32 by bsenyurt

Merhaba Arkadaşlar,
Diyelim ki elinizde çeşitli tipte ve sayıda fonksiyon var ve bunların çalışma zamanındaki işleyiş sürelerini hesaplamak istiyorsunuz. Normal şartlarda her metoda gidip **DateTime** tipini ele alarak süre ölçümleri yapabiliriz. Ya da **Delegate** sınıfına bir genişletme fonksiyonu yazarak sorunu halletmeye çalışırız. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.

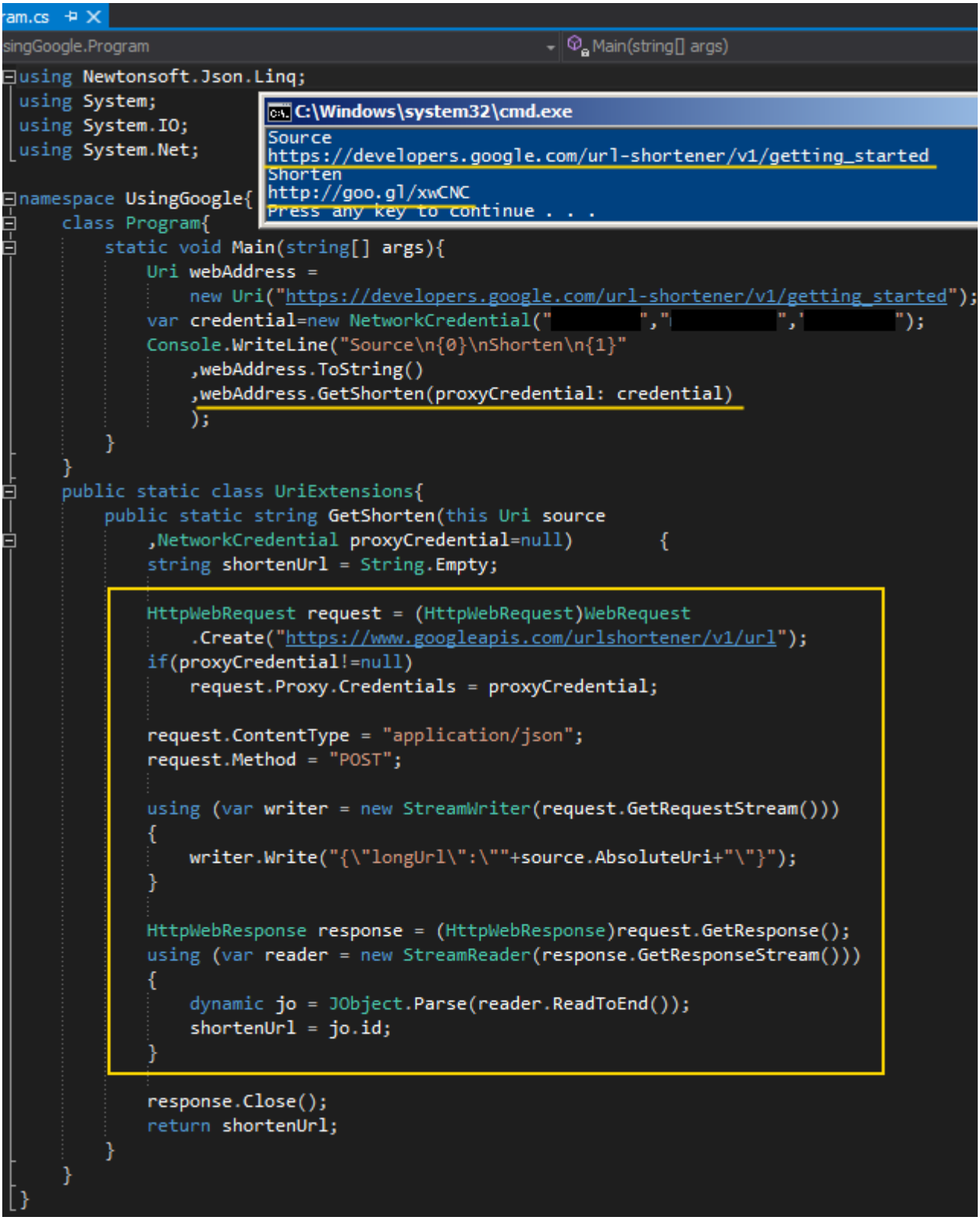


Bir başka ipucunda görüşmek dileğiyle hepinize mutlu günler dilerim.

Tek Fotoluk İpucu 97–Google Shortener URL Hizmetini C# ile Kullanmak

Salı, 26 Mart 2013 12:30 by bsenyurt

Merhaba Arkadaşlar,
Malumunuz bazen Web adreslerine ait **URL** satırları epeyce uzun olabiliyorlar ve bunları saklamak gibi amaçlarla kullanmak istediğimizde, genellikle kısaltma yoluna gitmeyi tercih ediyoruz*(Sanırım kimse 20 haneye sığdırılabilecek 200 karakterlik bir URL bilgisi ile uğraşmak istemez)* Bir URL adresini kısaltmak için kullanılabilecek pek çok global hizmet bulunmakta. Bunlardan birisi de**Google**' ın **Shortener** servisi*(ki bu adresten de görebileceğiniz gibi kendisi de epeyce kısa)*. Peki bir tarayıcı ile bu söz konusu servise kolayca gönderebildiğimiz bir talebi kod tarafında C# ile gerçekleştirmek isteseydiniz nasıl bir yol izlersiniz? Aşağıdaki gibi olabilir mi?




```
using Newtonsoft.Json.Linq;
using System;
using System.IO;
using System.Net;

namespace UsingGoogle{
    class Program{
        static void Main(string[] args){
            Uri webAddress =
                new Uri("https://developers.google.com/url-shortener/v1/getting_started");
            var credential=new NetworkCredential("[redacted]","[redacted]",'[redacted]");
            Console.WriteLine("Source\n{0}\nShorten\n{1}"
                ,webAddress.ToString()
                ,webAddress.GetShorten(proxyCredential: credential)
                );
        }
    }
    public static class UriExtensions{
        public static string GetShorten(this Uri source
            ,NetworkCredential proxyCredential=null)        {
            string shortenUrl = String.Empty;

            HttpWebRequest request = (HttpWebRequest)WebRequest
                .Create("https://www.googleapis.com/urlshortener/v1/url");
            if(proxyCredential!=null)
                request.Proxy.Credentials = proxyCredential;

            request.ContentType = "application/json";
            request.Method = "POST";

            using (var writer = new StreamWriter(request.GetRequestStream()))
            {
                writer.Write("{\"longUrl\":\""+source.AbsoluteUri+"\"}");
            }

            HttpWebResponse response = (HttpWebResponse)request.GetResponse();
            using (var reader = new StreamReader(response.GetResponseStream()))
            {
                dynamic jo = JObject.Parse(reader.ReadToEnd());
                shortenUrl = jo.id;
            }

            response.Close();
            return shortenUrl;
        }
    }
}
```

Örnekten de görüleceği üzere https://www.googleapis.com/urlshortener/v1/url adresine **JSON**formatında bir talep gönderilmekte olup, gelen cevap içerisinden **id** niteliğinin değeri yakalanmaktadır. Üstelik bu işlem sırasında **NewtonSoft'** un ilgili **NuGet** paketinden yararlanılmış olup söz konusu fonksiyonellik, **Uri** sınıfı için bir **Extension Method** olarak tanımlanmıştır.*(NewtonSoft ile*

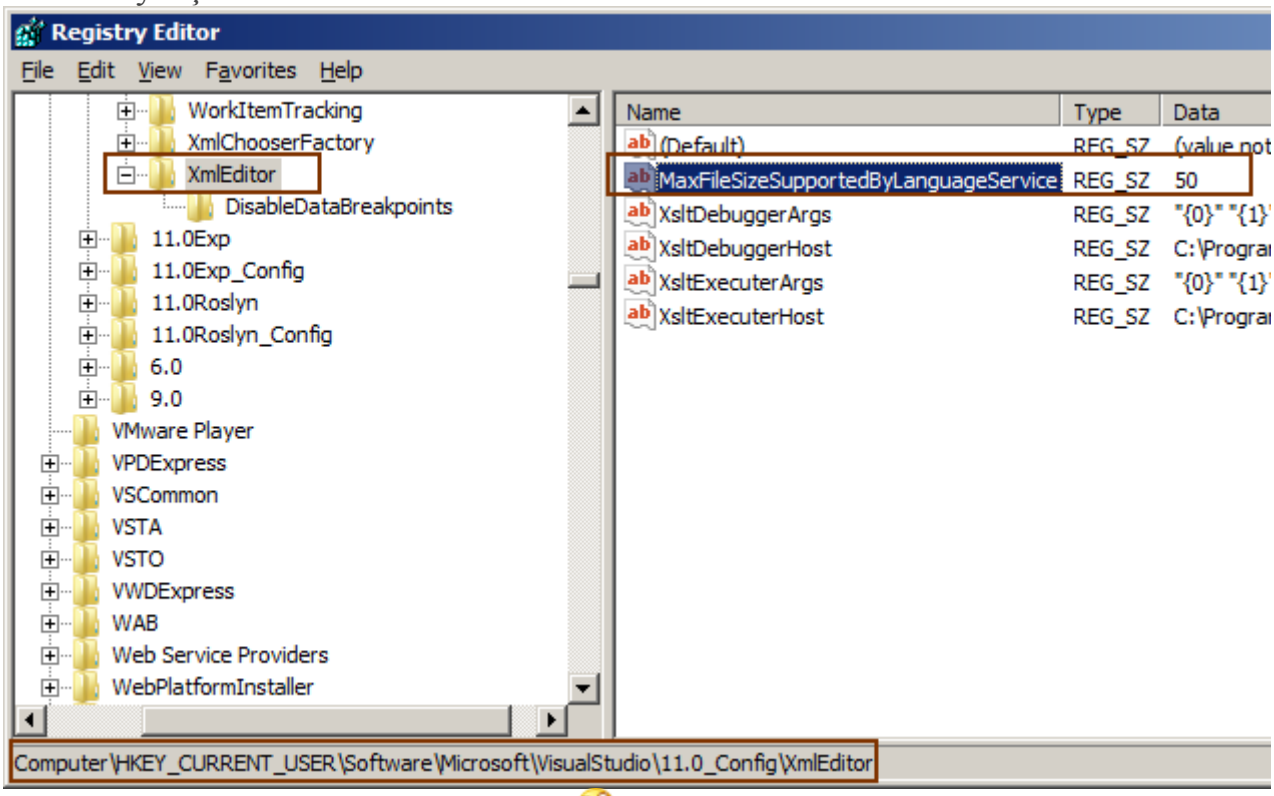
ilişkili olarak şuradaki ve buradaki ipuçlarına bakabilirsiniz) Bir başka ip ucunda görüşmek dileğiyle

[Örneği denerken Google' ın Shortener servisinin web adresini kontrol etmenizi öneririm. Yazının yazıldığı tarihe rağmen, yayınlandığı tarihte değişmiş, hatta kaldırılmış dahi olabilir]

Tek Fotoluk İpucu 96–10Mb Üstü XML Dosyaları

Salı, 26 Mart 2013 12:28 by bsenyurt

Merhaba Arkadaşlar,
Geçtiğimiz günlerde bloğumdaki içeriği yedeklemek için dışarı aktardım. Yaklaşık olarak **20Mb**büyüklüğündeki **XML** içeriğini, sonrasında **Visual Studio 2012** ile açıp incelemek istedim*(Daha önceden de yaptığım bir işti. Genellikle Almanac' ı üretmek için kullanıyorum)* Ancak bulunduğu ortamdaki **Visual Studio 2012, 10 Mb üstü XML** içeriğini açamayacağını, ille de açmak istiyorsam **Registery**' de bu konu ile ilişkili değeri değiştirme gerektiğini söyledi. Sonuç olarak aşağıdaki düzenlemeyi yaparak bu kısıtlamayı aşabilirsiniz.

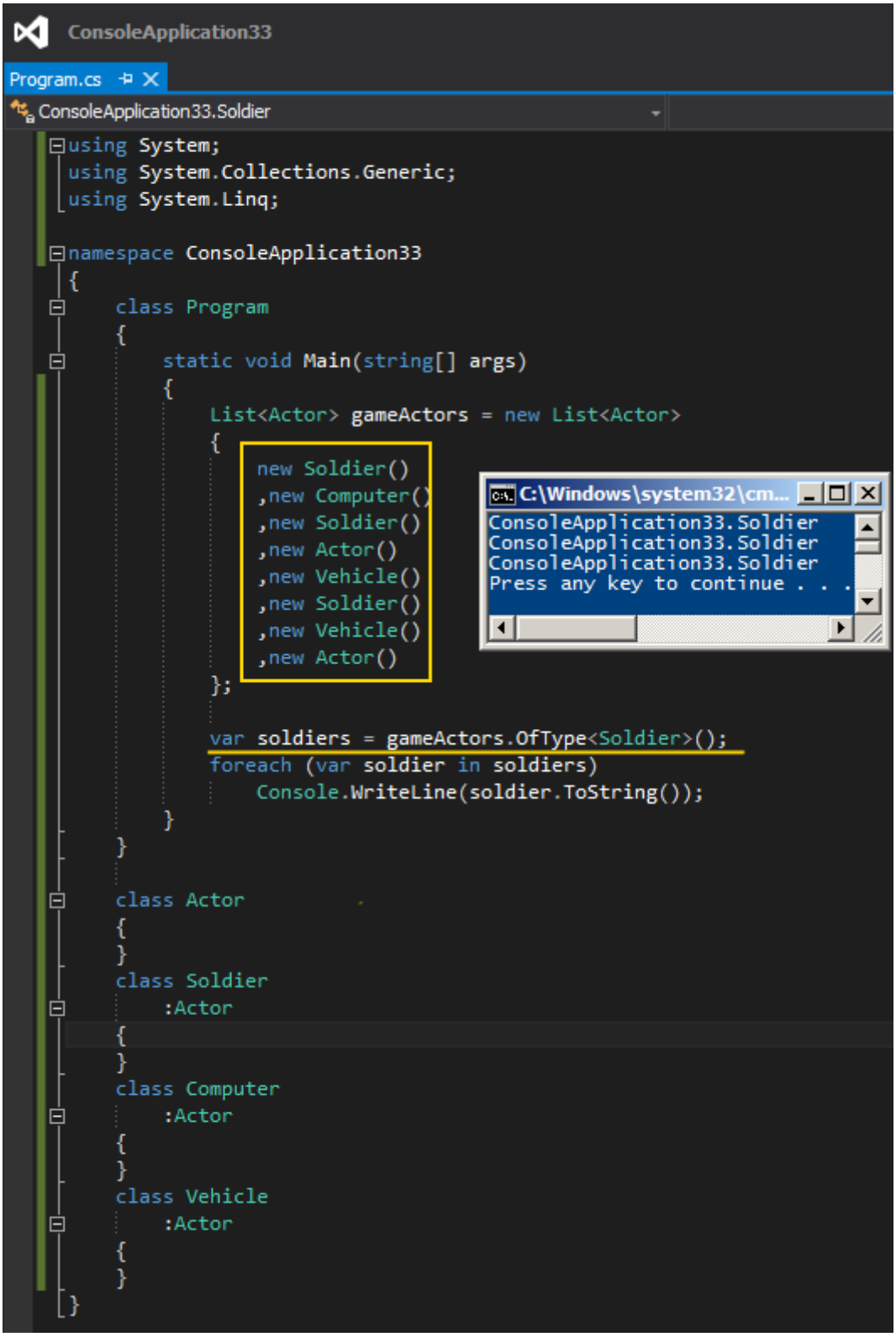


Bir başka ipucunda görüşmek dileğiyle

Tek Fotoluk İpucu 95–OfType

Salı, 26 Mart 2013 12:25 by bsenyurt

Merhaba Arkadaşlar,
LINQ*(Language INtegrated Query)* tarafını her ne kadar yoğun kullanıyor olsak da gözümüzden kaçırdığımız, dikkat etmediğimiz, yerine yeni geliştirmeler yaptığımız ama aslında bizim kullanmamızı bir köşede bekleyen fonksiyonlar vardır.
Örneğin kalıtımsal ilişki içerisinde olan **Actor, Soldier, Computer, Vehicle** tiplerini düşünün. Hatta bu tipler arasında, **Soldier** bir **Actor**' dür, **Computer**' **de** bir **Actor**' dür ve hatta **Vehicle**' da bir **Actor**' dür şeklinde **is-a** ilişkisi olduğunu düşünelim.
Peki elinizde **Actor** tipinden bir liste varsa ve bir an gelipte içinden sadece **Soldier** olanları çekmek isterseniz, ne yaparsınız? Aşağıdaki gibi bir kullanım işinize yarayabilir mi acaba?

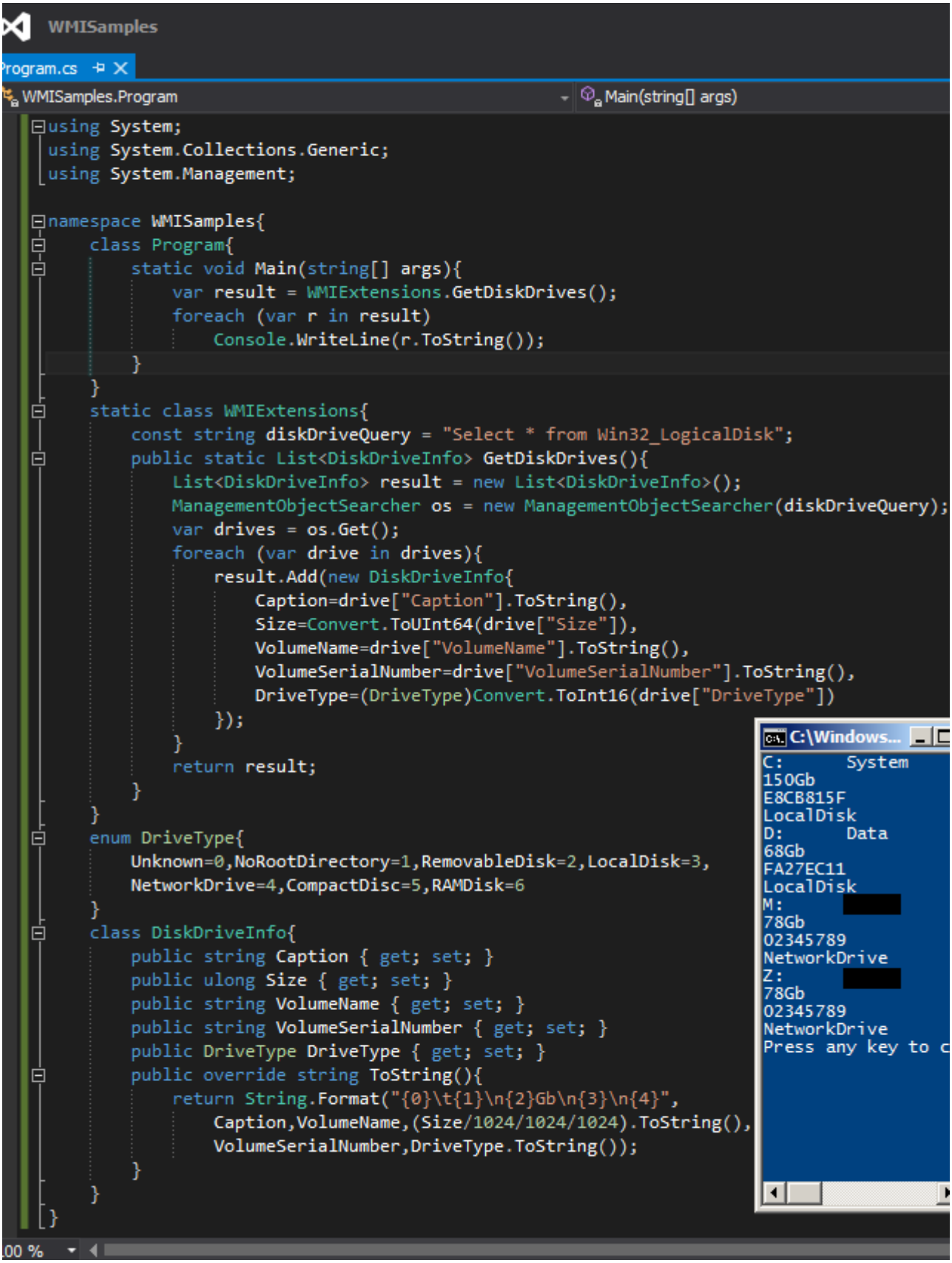


Bir başka ipucunda görüşmek dileğiyle.

Tek Fotoluk İpucu 94–WMI ile Disk Bilgilerini Okumak

Salı, 26 Mart 2013 12:22 by bsenyurt

Merhaba Arkadaşlar,
WMI*(Windows Management Instrumentation)* maceralarımıza devam etmeye ne dersiniz? Eğer biraz daha kasarsak, geniş bir **WMI** kütüphanesi bile oluşturabiliriz. Bu fotoğrafımıza konu olan güncel senaryomuz ise şöyle; İşletim sistemi tarafından **Map** edilmiş **Disk** bilgilerine nasıl ulaşabiliriz? Sadece Hard Disk' ler değil. Bağlı olduğumuz Network Driver' ları da öğrenmek istediğimizi varsalayım Dilerseniz önce **WMI** tarafında önceki ipuçlarımızdan da yararlanarak senaryoyu gerçekleştirmeye çalışın. Sonrasında fotoğrafımıza bakarsınız.

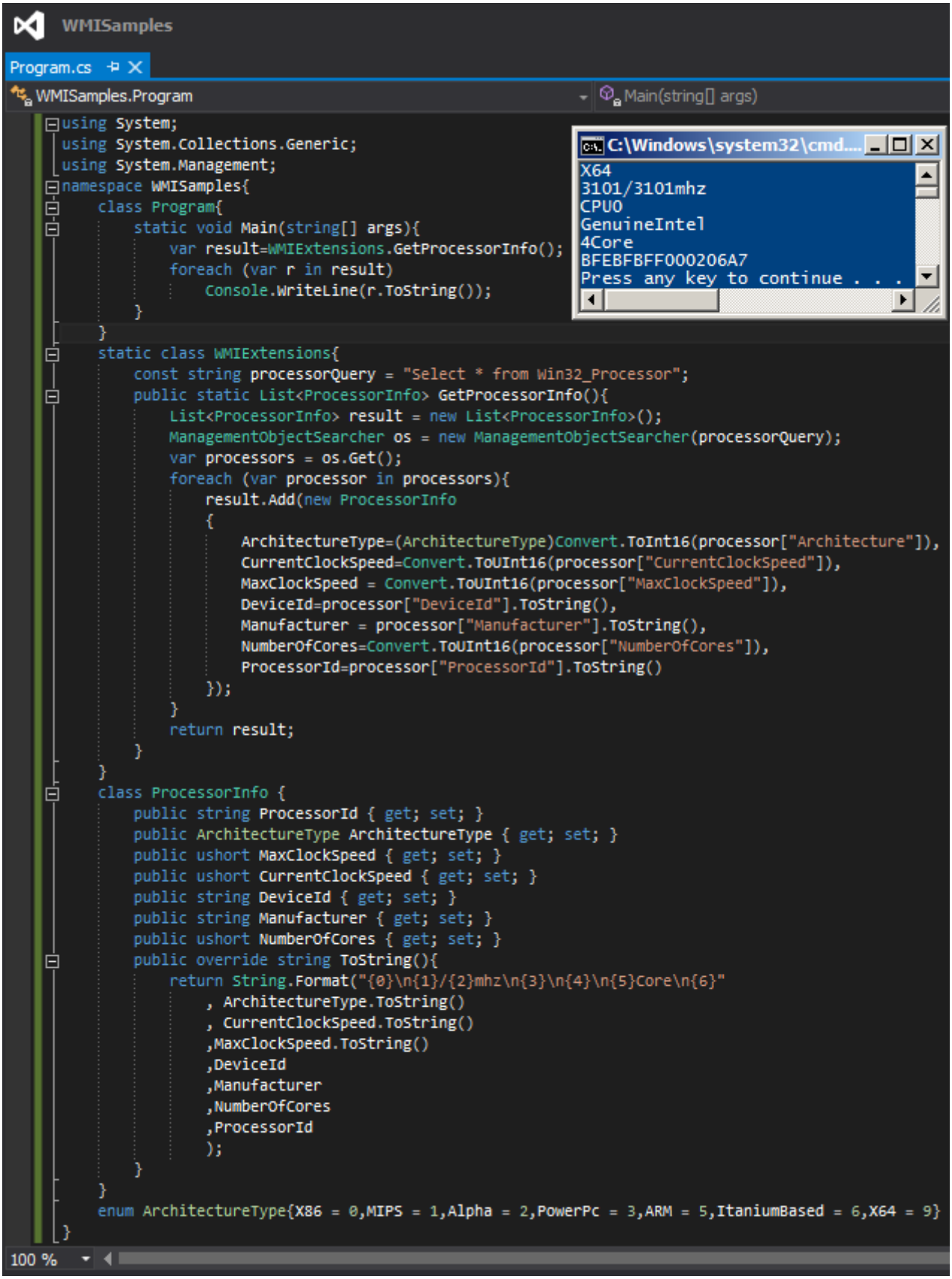


Tabi **Win32_LogicalDisk** tipinin kullanılabilecek farklı özellikleri de mevcut. Bu özelliklere de şu adresten bakabilirsiniz. Bir başka ipucundan görüşmek dileğiyle 😉

Tek Fotoluk İpucu 93–WMI ile Processor Bilgisini Okumak

Salı, 26 Mart 2013 12:20 by bsenyurt

Merhaba Arkadaşlar,
Hatırlayacağınız üzere bir önceki Tek Fotoluk İpucunda, **Win32_PhysicalMemory** isimli**WMI*(Windows Management Instrumentation)*** tipinden yararlanarak, makinede takılı olan RAM' ler hakkında temel bilgileri nasıl alabileceğimizi incelemiştik. Bu seferki ipucumuzda ise işlemci bilgilerini okumaya çalışıyor olacağız. Aşağıdaki fotoğrafta görüldüğü gibi.



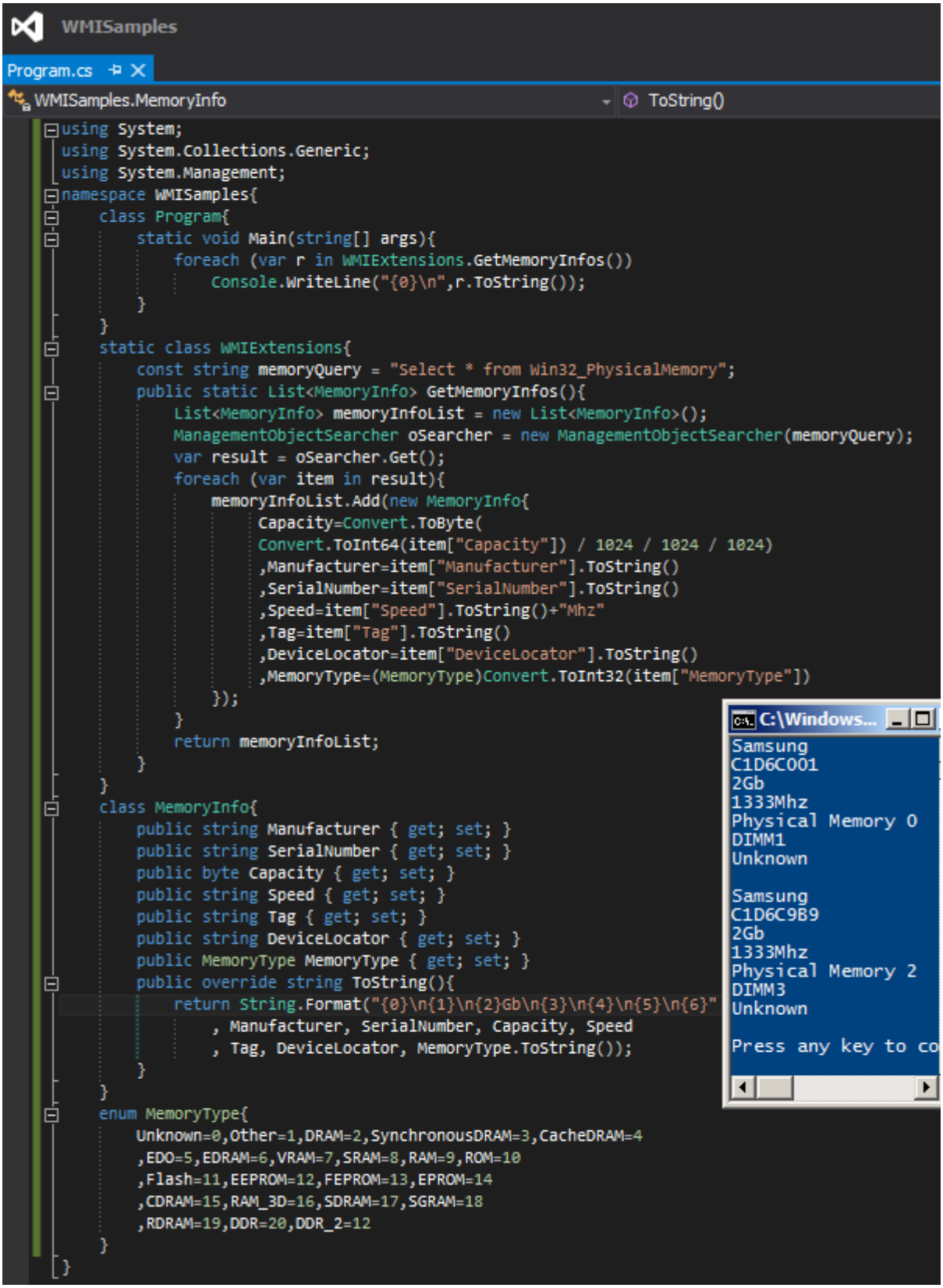

```
using System;
using System.Collections.Generic;
using System.Management;
namespace WMISamples{
    class Program{
        static void Main(string[] args){
            var result=WMIExtensions.GetProcessorInfo();
            foreach (var r in result)
                Console.WriteLine(r.ToString());
        }
    }
    static class WMIExtensions{
        const string processorQuery = "Select * from Win32_Processor";
        public static List<ProcessorInfo> GetProcessorInfo(){
            List<ProcessorInfo> result = new List<ProcessorInfo>();
            ManagementObjectSearcher os = new ManagementObjectSearcher(processorQuery);
            var processors = os.Get();
            foreach (var processor in processors){
                result.Add(new ProcessorInfo
                {
                    ArchitectureType=(ArchitectureType)Convert.ToInt16(processor["Architecture"]),
                    CurrentClockSpeed=Convert.ToUInt16(processor["CurrentClockSpeed"]),
                    MaxClockSpeed = Convert.ToUInt16(processor["MaxClockSpeed"]),
                    DeviceId=processor["DeviceId"].ToString(),
                    Manufacturer = processor["Manufacturer"].ToString(),
                    NumberOfCores=Convert.ToUInt16(processor["NumberOfCores"]),
                    ProcessorId=processor["ProcessorId"].ToString()
                });
            }
            return result;
        }
    }
    class ProcessorInfo {
        public string ProcessorId { get; set; }
        public ArchitectureType ArchitectureType { get; set; }
        public ushort MaxClockSpeed { get; set; }
        public ushort CurrentClockSpeed { get; set; }
        public string DeviceId { get; set; }
        public string Manufacturer { get; set; }
        public ushort NumberOfCores { get; set; }
        public override string ToString(){
            return String.Format("{0}\n{1}/{2}mhz\n{3}\n{4}\n{5}Core\n{6}"
                , ArchitectureType.ToString()
                , CurrentClockSpeed.ToString()
                ,MaxClockSpeed.ToString()
                ,DeviceId
                ,Manufacturer
                ,NumberOfCores
                ,ProcessorId
                );
        }
    }
    enum ArchitectureType{X86 = 0,MIPS = 1,Alpha = 2,PowerPc = 3,ARM = 5,ItaniumBased = 6,X64 = 9}
}
```

Bu arada Win32_Processor tipi için kullanabileceğiniz diğer özellikleri de bu adresten bulabilir ve deneyebilirsiniz. Bir başka ipucunda görüşmek dileğiyle

Tek Fotoluk İpucu 92–WMI ile RAM Bilgilerini Almak

Salı, 26 Mart 2013 12:17 by bsenyurt

Merhaba Arkadaşlar,
Diyelim ki uygulamanız içerisinden, çalışmakta olduğu **Windows** işletim sistemi tabanlı makinenize ait fiziki **RAM** bilgilerini almak istiyorsunuz. Örneğin markasını, hangi slota takılı olduğunu, boyutunu, tipini vs…
Bu amaçla kullanabileceğiniz etkili yöntemlerden birisi de bildiğiniz üzere **WMI*(Windows Management Instrumentation)*** alt yapısından yararlanmaktır. Aslında tek yapmanız gereken**ANSI-SQL** standartlarının bir alt kümesi olan basit bir **WQL*(WMI Query Language)*** ifadesi kullanmaktır. Nasıl mı? Aynen aşağıdaki fotoğrafta görüldüğü gibi

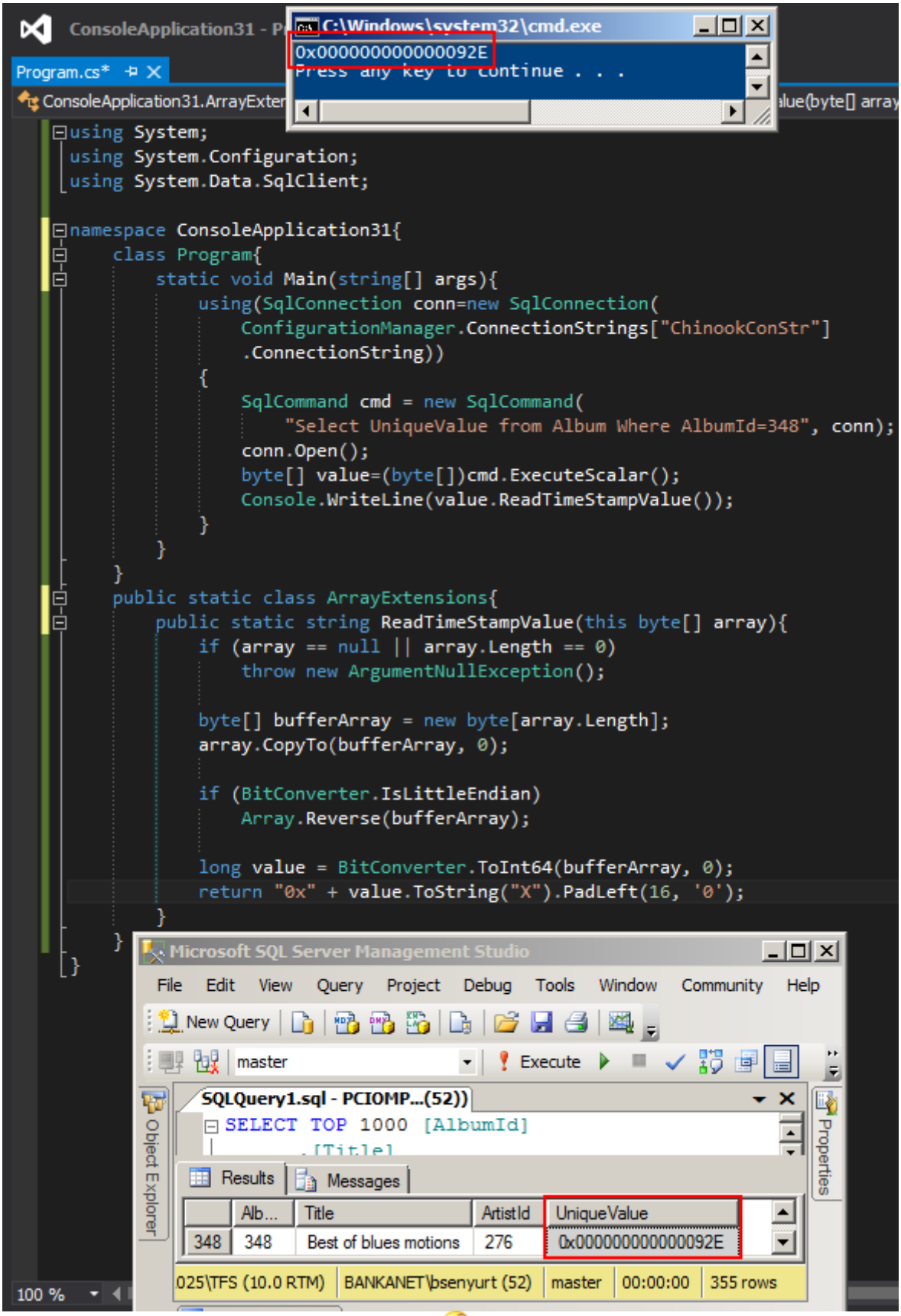


Örneğin benim sistemimde 2 adet Samsung marka 2Gb RAM varmış. Tipleri Unknown gelse de biraz fikir sahibi oldum diyebilirim. Bir başka ipucundan görüşmek dileğiyle

Tek Fotoluk İpucu 91–Timestamp Veriyi String Olarak Okumak

Salı, 26 Mart 2013 12:15 by bsenyurt

Merhaba Arkadaşlar,
Diyelim ki **SQL Server** üzerinde duran tablolarda **timestamp** veri tipinden alanlar bulunmakta ve siz bu alanları belki bir **Backoffice** uygulamasında belki bir **admin** panelde, kullanıcalara göstermek istiyorsunuz. Normal şartlarda bilindiği üzere bu alan bir **byte[]** array olarak elde edilmektedir. Dolayısıyla **timestamp** içeriği taşıyan bu **byte[] array'** in anlamlı bir **string** tipine dönüştürülmesi okunurluğu açısından şarttır. Ne yaparsınız? Belki basit bir **extension method**' u bu amaçla projeye dahil edebilirsiniz. Aynen aşağıda görüldüğü gibi.

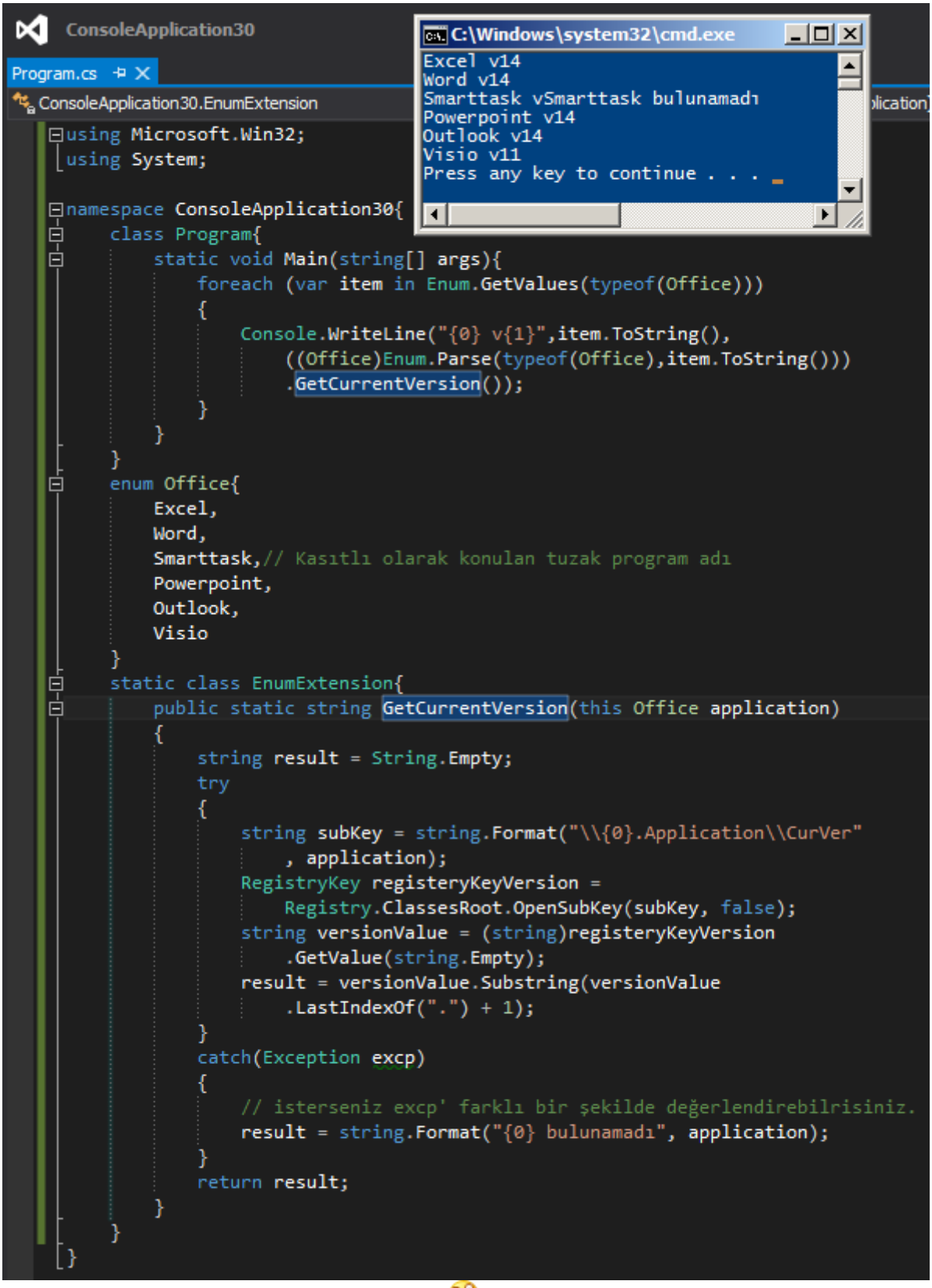


Bir başka ipucunda görüşmek dileğiyle

Tek Fotoluk İpucu 90–Office Ailesinin Versiyonlarını Öğrenmek

Salı, 26 Mart 2013 12:10 by bsenyurt

Merhabalar,
Diyelim ki yazmış olduğunuz ürünü kuracağınız/kurduğunuz **Windows** platformunda yüklü olan**Office** ürünleri var ise, bunların versiyonlarını öğrenmeniz gerekiyor. Ne yaparsınız? Bilinen bir kaç yol var*(önreğin bunlardan birisi Late Binding nedeniyle yavaş olan dynamic kelimesinin de kullanıldığı Activator.CreateInstance metodu)* ama en hızlılarından birisi, **Registery** ayarlarına bakmak. Hatta bir de işin içerisine **Enum** sabiti kattığımızı ve hatta ona bir **Extension** metod olarak bu işlevselliği dahil ettiğimizi düşünelim. Aynen aşağıdaki fotoğrafta görüldüğü gibi

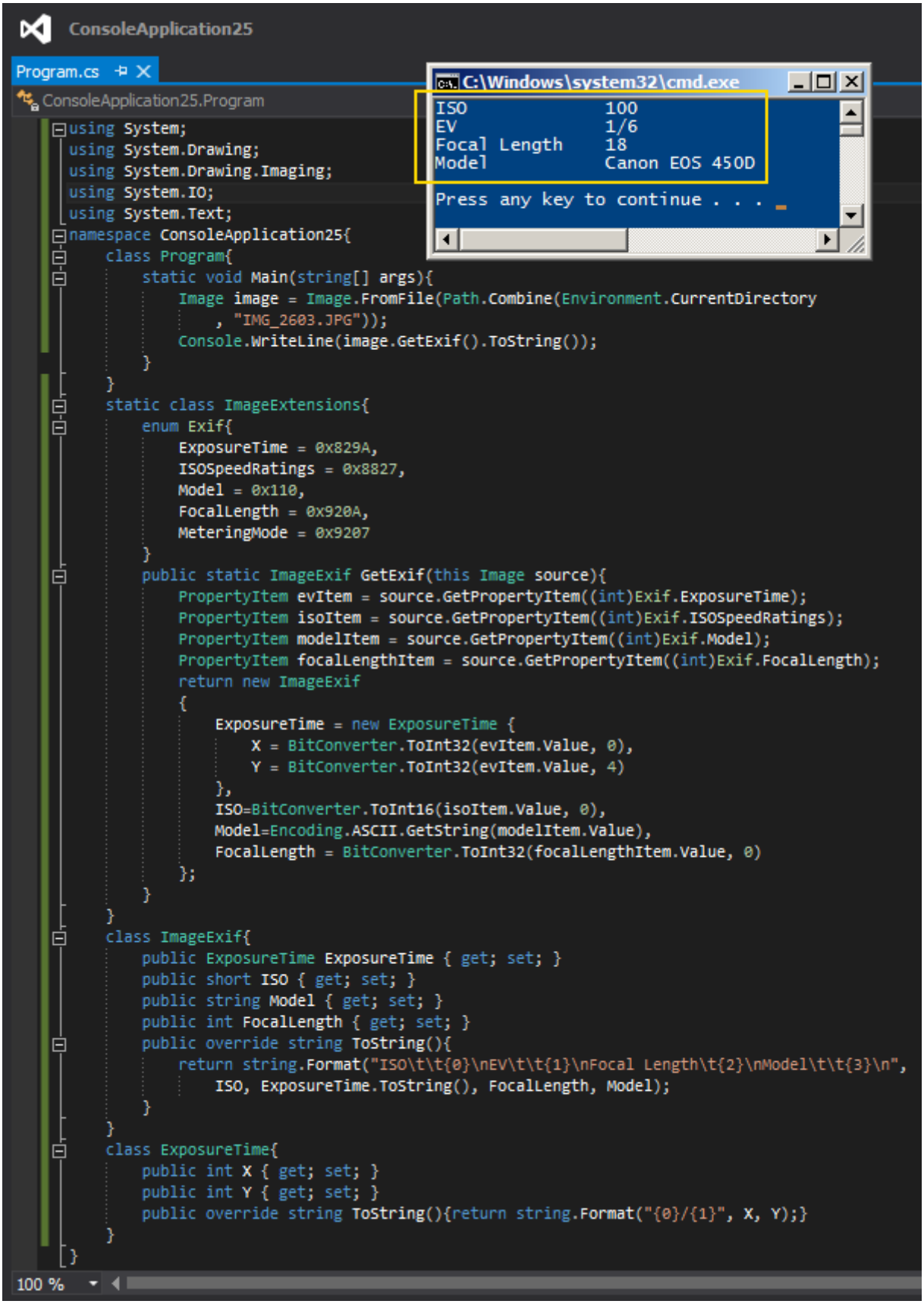


Bir başka ipucunda görüşmek dileğiyle

Tek Fotoluk İpucu 89–Exif Bilgilerini Okumak

Salı, 26 Mart 2013 12:05 by bsenyurt

Merhaba Arkadaşlar,
Takip edenler amatör düzeye yaklaşmaya çalışan/çabalayan fotoğrafçılık tutkunu birisi olduğumu bilirler. Hatta okullarda **"Fotoğrafçı ve Hataları…"** konulu bir ders konusu olabilecek kadar iddialı bir foto bloğumda da bulunmaktadır
Bu bloğa fotoğraf yüklerken her seferinde yapmak zorunda olduğum ama bundan çok fazla derecede sıkıldığım bir işlemde fotoğrafların **exif** bilgilerini öğrenip yazmaktır.
Oysaki **Image** tipine eklenebilecek bir **Extension** metod ve fotoğrafa ait bazı özellik bilgilerinden yararlanarak **ISO, EV, Focal Length** gibi istediğim temel bilgilere ulaşabilirim. Nasıl mı? Aynen aşağıdaki fotoğrafta görüldüğü gibi

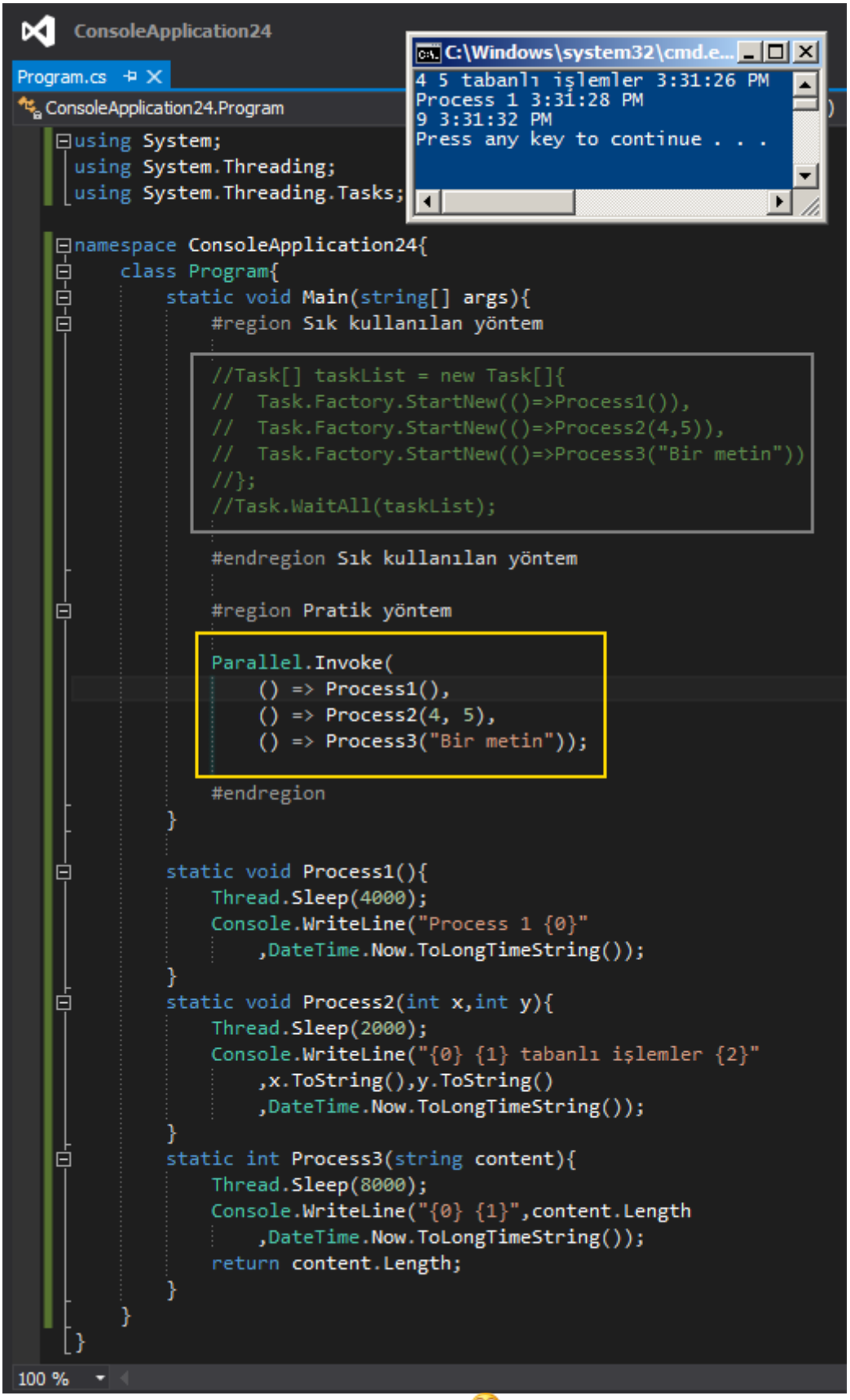


Bir başka ipucunda görüşmek dileğiyle.

Tek Fotolok İpucu 88–Task.WaitAll out, Parallel.Invoke in

Salı, 26 Mart 2013 12:00 by bsenyurt

Merhaba Arkadaşlar,
Bildiğiniz üzere paralel çalışmasını istediğimiz görevler olduğunda genellikle bunları birer **Task**halinde üretir ve bir dizi içerisinde toplarız*(En azından TPL-Task Parallel Library geldikten sonra böyle yapmakta olduğumuzu ifade edebiliriz)* Görevleri Task tipinden bir dizi içerisinde toplamamızın sebebi ise genellikle **WaitAll** gibi bir çağrıya ihtiyaç duyabilecek olmamızdır.
Ancak bunun daha pratik olan bir yolu da vardır. O da **Parallel** sınıfı üzerinden erişilebilen **Invoke**metodudur. Bu metod birden fazla **Action** örneğini parametre olarak alabilir, çalıştırabilir ve hatta tamamı sonlanıncaya kadar kod satırını duraksatabilir. Nasıl mı? Aynen aşağıda görüldüğü gibi

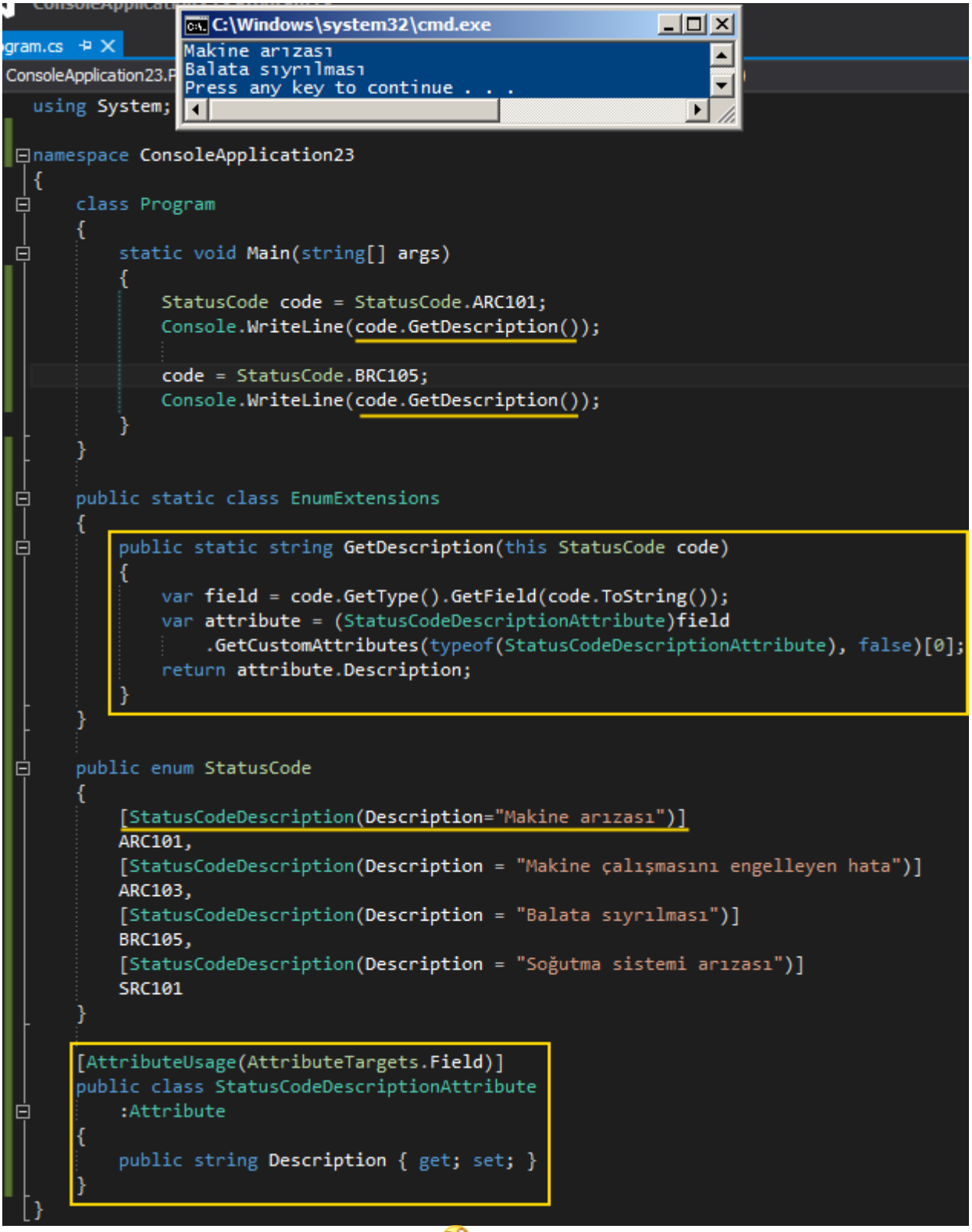


Bir başka ipucunda görüşmek dileğiyle

Tek Fotoluk İpucu 87–Enum Sabitleri ile Attribute Kullanımı

Salı, 26 Mart 2013 11:55 by bsenyurt

Merhaba Arkadaşlar,
Bazen uygulamalarımızda kullandığımız **Enum** sabitlerinin içerikleri anlaşılabilir kelimelerden oluşmayabilir. Örneğin **AR101,BR103** isimli **Enum** değerleri çalışma zamanında nasıl yorumlanabilirler. Mantıklı isimler verebilecek olsak da bazen bu değerler sistemin kendisi için anlamlı olabilirler. Ancak dilerseniz enum sabiti değerlerini, kendi geliştireceğiniz nitelikler*(Attribute)*ile donatabilir ve çalışma zamanına ek bilgiler sunabilirsiniz. Hem de basit bir **extension** metod yardımıyla bu fonksiyonelliğin kolayca ulaşılabilir olmasını sağlayabilirsiniz. Nasıl mı? İşte size bir örnek



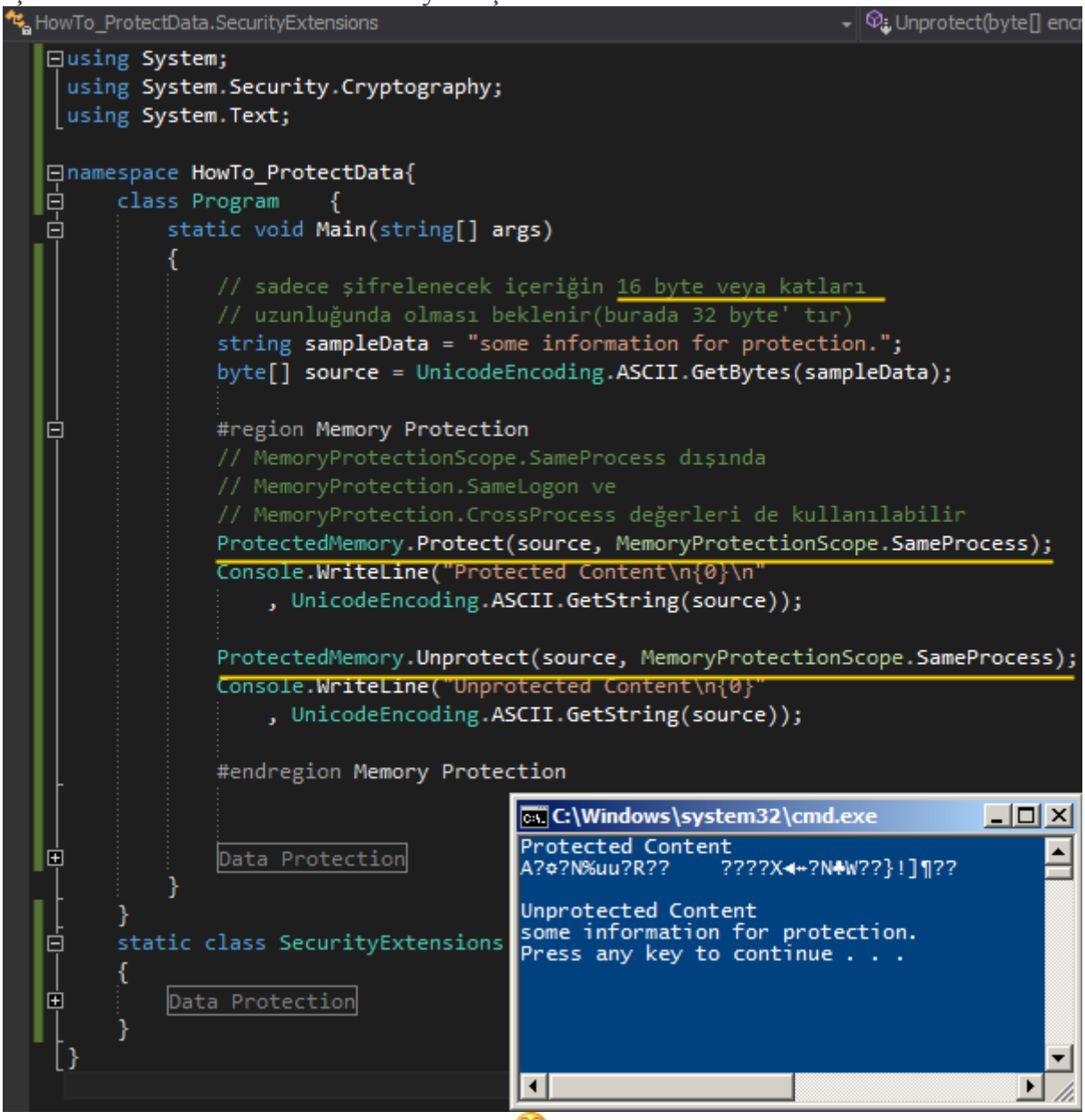
Bir başka ipucunda görüşmek dileğiyle

Tek Fotoluk İpucu 86–Zahmetsizce Encryption (ProtectedMemory)

Salı, 26 Mart 2013 11:50 by bsenyurt

Merhaba Arkadaşlar,

Bir önceki tek fotoluk ipucunda ProtectedData sınıfından yararlanmış ve basitçe bir **byte** dizisinin nasıl şifrelenebileceğini/çözümlenebileceğini görmüştük. Hatırlarsanız veriyi **Current User** ve**Local Machine** seviyelerinde ele alabiliyorduk. **DPAPI**' nin kullanıldığı veri odaklı bu tekniğin yanında, bellek üzerinde yer alan bir içeriğin **Process** bazında şifrelenmesi/çözümlenmesi de mümkündür. Aynı Process*(SameProcess)*, farklı Process*(CrossProcess)* veya aynı giriş*(SameLogon)*bilgisi için…Üstelik son derece de kolay bir şekilde. Nasıl mı?

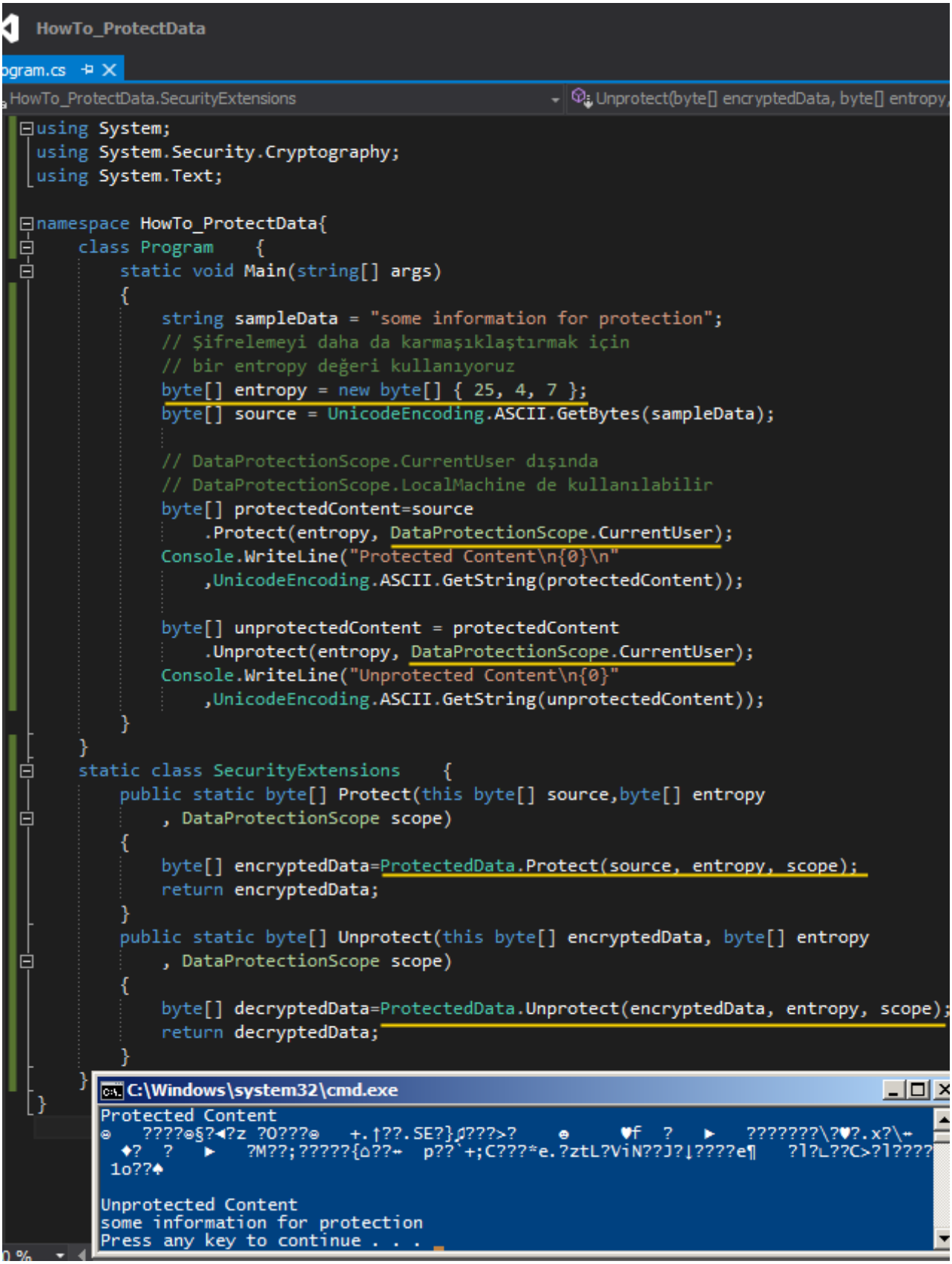


Bir başka ipucunda görüşmek dileğiyle

Tek Fotoluk İpucu 85–Zahmetsizce Encryption(ProtectedData)

Salı, 26 Mart 2013 11:46 by bsenyurt

Merhaba Arkadaşlar,
Cryptography denilince **.Net Framework** tarafında epey bir çözüm bulunmakta. Bazıları oldukça karmaşıktır ve **simetrik** yada **a-simetrik** olmalarına bağlı olaraktan, ortak noktalarından birisi de, tekniğe göre kullanılan **Vector-Key** değerlerinin tutulması/bilinmesi gibi zorunluluklardır.
Aslında **Windows** tarafında, **XP** işletim sisteminden beri var olan*(hatta Windows 8 de bunun Cloud destekli bir versiyonu da vardır-> DPAPI-NG)* bir **API** mevcut. **DPAPI**. Nam-ı diğer **Data Protection API**.
Dilersek bu API' yi kullanarak verilerimizi şifreleyebilir ve güvenliklerini sağlayabiliriz. Bunu yaparken**CurrentUser** veya **LocalMachine** seçeneklerini belirterek şifreyi kimin açabileceğini kolayca ifade edebiliriz.
Oldukça kolay bir biçimde; herhangibir algoritmaya bağımlı kalmadan, vector-key saklamadan, üstelikte karmaşıklığı basitçe arttırabileceğimiz*(entropy değerlerini değiştirerek deneyin)* bir formatta…Nasıl mı? Buyrun

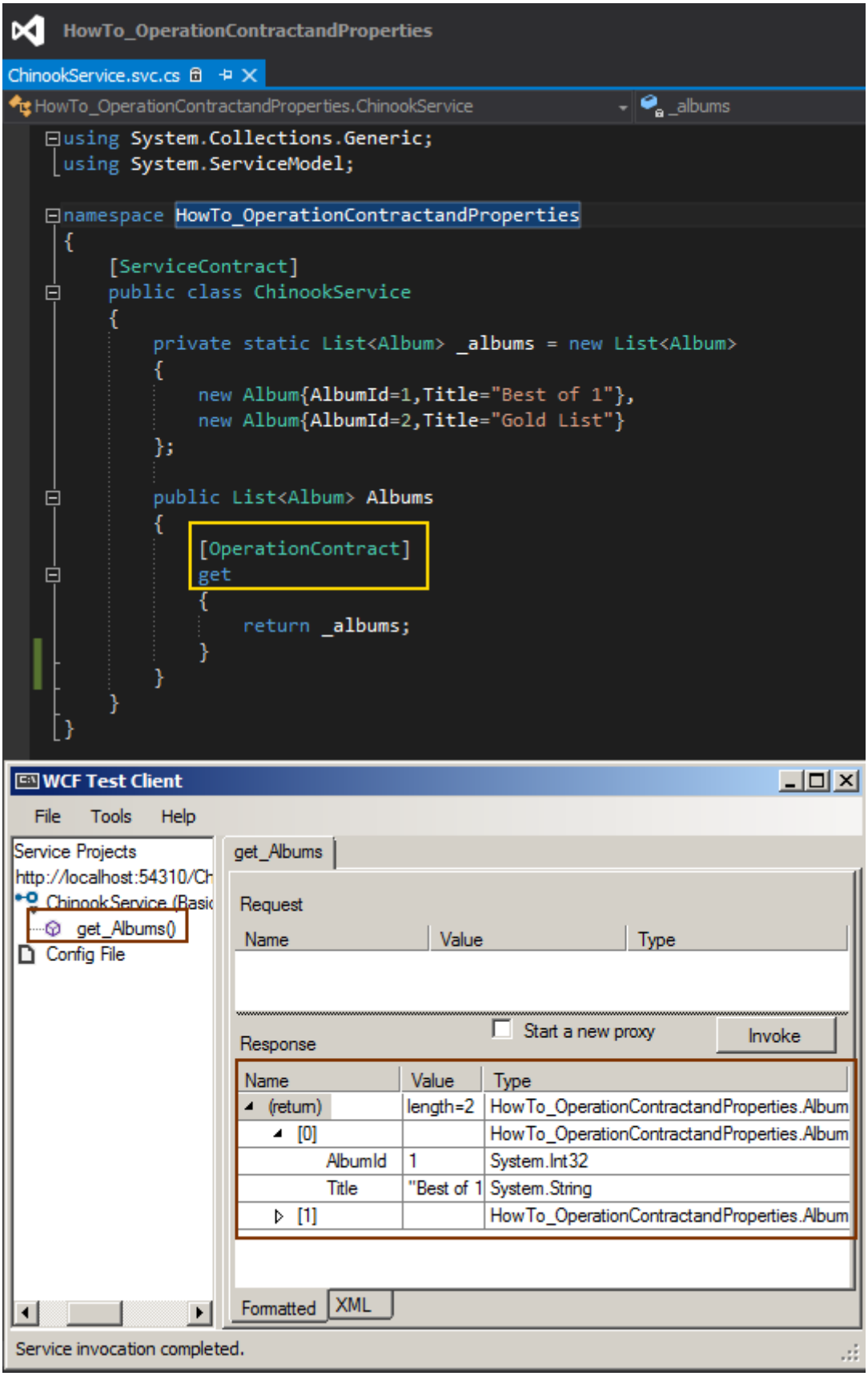


Burada ki ipucundan **Data Protection** kullanımı söz konusudur. Aslında aynı **API**' yi kullanarak bir de **Memory Protection** yapabiliriz. Buna da bir sonraki ipucumuzda bakalım

Tek Fotoluk İpucu 84–WCF içerisinde Property Kullanımı

Salı, 26 Mart 2013 11:40 by bsenyurt

Merhaba Arkadaşlar,
Malum bildiğiniz üzere **get** ve **set** bloklarından oluşan özellikler*(Properties)* aslına bakarsanız arka planda*(IL-Intermediate Language)* birer metod olarak ifade edilirler. Bu teoriden yola çıkarsak bir servis içerisine özellik*(Property)* yazıp get,set metoldarını operasyon olarak dış dünyaya sunabiliriz Nasıl mı? Aynen aşağıdaki fotoğrafta görüldüğü gibi.



Gördüğünüz gibi **ReadOnly** olarak tanımlanmış bir **Property**, **OperationContract** niteliği ile işaretlenen **get** metodunu dışarıya operasyon olarak sunabilmekte. Bir başka ipucundan görüşmek dileğiyle

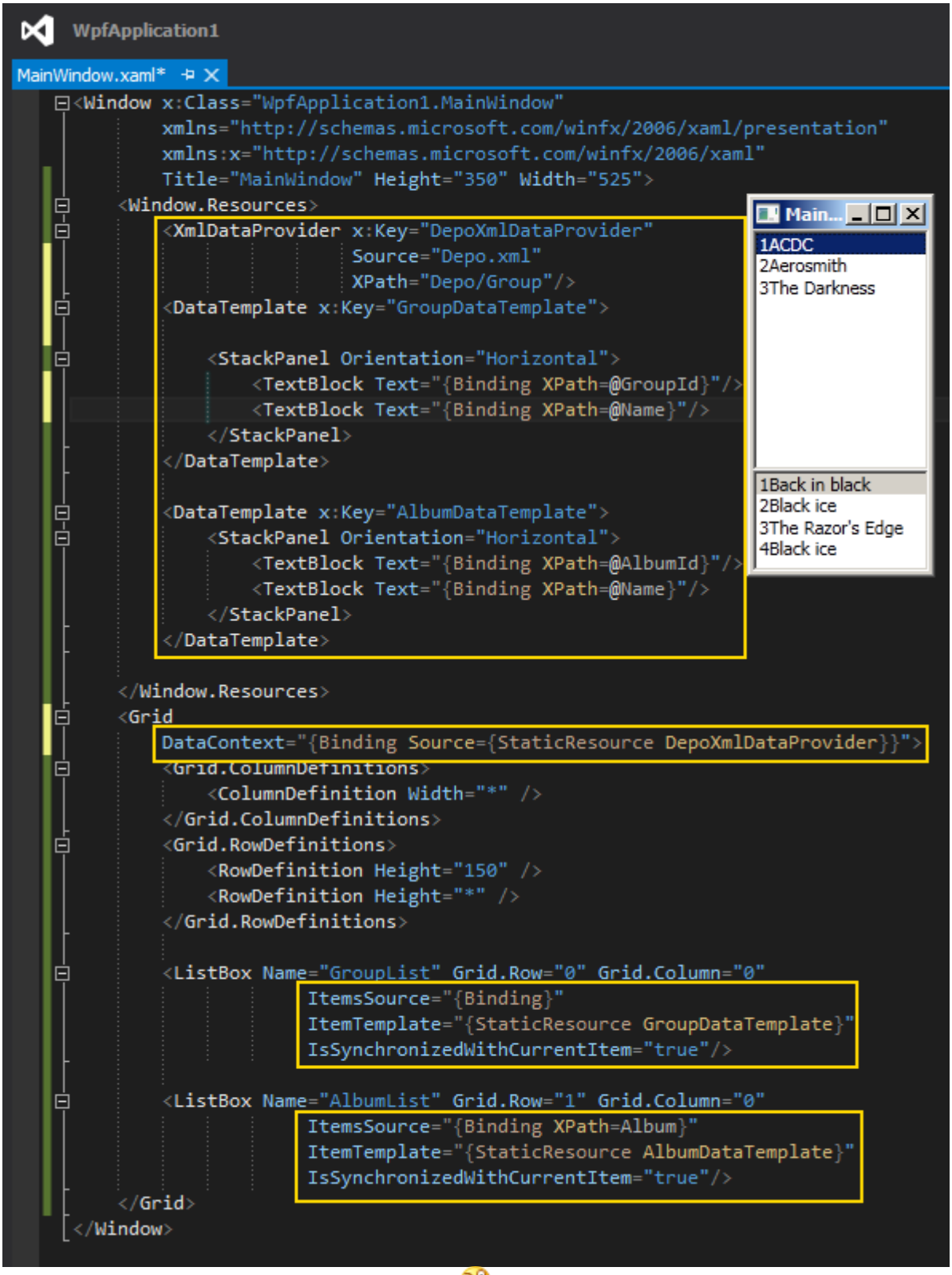
Tek Fotoluk İpucu 83–XML, XAML, XmlDataProvider ve Master Child Binding

Salı, 26 Mart 2013 11:35 by bsenyurt

Merhaba Arkadaşlar,
Diyelim ki elinizde aşağıdaki gibi **Master-Child** veri ilişkisi içeren*(1 gruba bağlı birden fazla albüm)*bir **XML** dosyası var.

```
<?xml version="1.0" encoding="utf-8"?>
<Depo>
  <Group GroupId="1" Name="ACDC">
    <Album AlbumId="1" Name="Back in black"/>
    <Album AlbumId="2" Name="Black ice"/>
    <Album AlbumId="3" Name="The Razor's Edge"/>
    <Album AlbumId="4" Name="Black ice"/>
  </Group>
  <Group GroupId="2" Name="Aerosmith">
    <Album AlbumId="5" Name="O Yeah! Ultimate Aerosmith"/>
  </Group>
  <Group GroupId="3" Name="The Darkness">
    <Album AlbumId="6" Name="One way ticket to hell"/>
    <Album AlbumId="7" Name="Permission to land"/>
  </Group>
</Depo>
```

ve sizde örneğin **WPF-XAML** tarafında buradaki **Master-Detail** ilişkiyi kullanmak ve hatta iki veri bağlı kontrol üzerinden sembolize etmek istiyorsunuz. Ne yaparsınız? Belki de aşağıdaki fotoğrafta görülen tekniği kullanırsınız



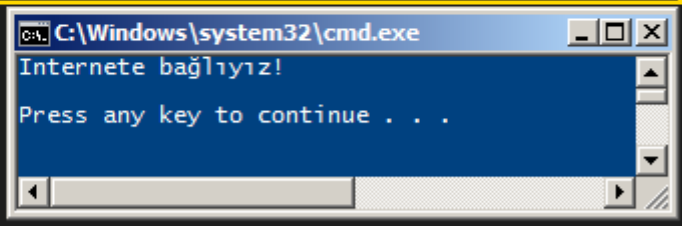
Bir başka ipucunda görüşmek dileğiyle

Tek Fotoluk İpucu 82–Internete Bağlı mıyız? (Round II)

Salı, 26 Mart 2013 11:30 by bsenyurt

Merhaba Arkadaşlar,
Bir önceki ipucumuzda **wininet.dll WinAPI** kütüphanesinden yararlanarak, internet' e bağlı olup olmadığımızı nasıl öğrenebileceğimizin fotoğrafını çekmiştik. Tahmin edeceğiniz üzere söz konusu senaryo için tek yol bu değil. Örneğin aşağıdaki gibi bir kullanımı da tercih edebilirsiniz.

```
using System;
using System.Net;

namespace ConsoleApplication21{
    class Program{
        static void Main(string[] args){
            Exception exception;
            bool connection = IsConnected("http://www.bing.com", out exception
                // Eğer bir proxy üzerinden çıkıyorsak Credential oluşturmalıyız
                ,credential:new NetworkCredential
                        {
                            Domain = [redacted]
                            ,UserName = [redacted],
                            Password = [redacted]
                        }
                );
            Console.WriteLine("Internete {0}\n{1}",
                connection?"bağlıyız!":"bağlı değiliz!",
                exception!=null?exception.Message:String.Empty
                );
        }
        static bool IsConnected(string webAddress,out Exception exception
            ,NetworkCredential credential=null)
 {
            bool result=false;
            exception = null;

            try{
                HttpWebRequest request = (HttpWebRequest)WebRequest.Create(webAddress);
                // Eğer bir proxy üzerinden çıkıyorsak Credential oluşturmalıyız
                if(credential!=null)
                    request.Proxy.Credentials = credential;

                request.Method = "HEAD";
                HttpWebResponse response = (HttpWebResponse)request.GetResponse();
                response.Close();

                result = true;
            }
            catch(Exception excp){
                exception = excp;
            }

            return result;
        }
    }
}
```



Aslında bu yol belki de ilk akla gelen yoldur. Eğer bir internet sitesine ulaşabiliyorsak zaten internete bağlıyız diye düşünebiliriz Lakin bir önceki ipucunda kullandığımız ve daha çok donanımsal talep ile çalışan tekniğe nazaran daha yavaş ve hatta daha az bilgi veren bir çözümdür.

Bir başka ipucunda görüşmek dileğiyle.

WCFden, XML Web Servisine TransactionScope Activity Bileşeni Üzerinden Transaction Aktarmak

Pazartesi, 25 Mart 2013 21:21 by bsenyurt

[Örnek Visual Studio 2010, .Net Framework 4.0 tabanlıdır]

Merhaba Arkadaşlar,

Bir süre öncesine kadar **Composition** adı verilen bir katmanda yer alacak çeşitli servisler ile yoğun şekilde güreşmekteydim. Çok fazla faktör, çok fazla farklı sistem ve tabiri yerinde ise oyun ve oyuncu söz konusuydu.

WCF servisleri, **XML Web Servisleri**, **Java**t abanlı olanları ve belki de yarın gelecek olan çeşitli **COM** bileşenleri, 3ncü parti uygulamalar, koduna müdahale edemeyeceğimiz programlar vs.

Sadece bunlar olsa iyi. Bir de bunlar içerisine **Oracle** üzerinde koşan **Transactional** veritabanı işlemleri de mevcut olunca, işler ister istemez karışıyor ve künde pozisyonuna geliyorsunuz. Nitekim bu servisler n sayıda kombinasyon ile birbirleriyle etkileşimde bulunabilirler ve bu tip senaryolarda bir şekilde **Distributed Transaction** terminolojisinin uygulanması ve servisler arasında başarılı bir şekilde akıtılarak, **Two Phase Commit** ilkesinin gerçekleştirilebiliyor olması gereklidir.

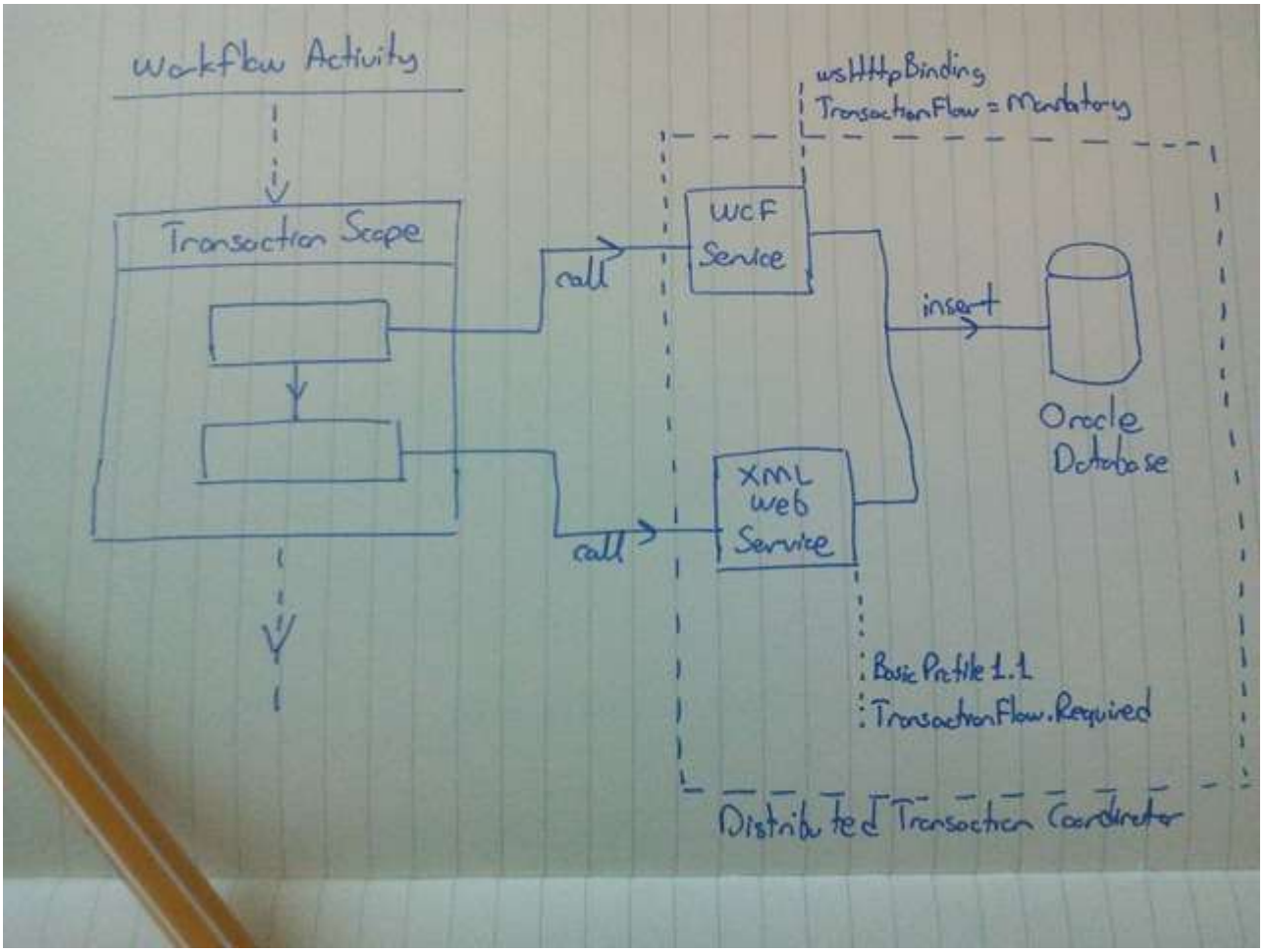
Daha önceden ele aldığımız bir yazıda, WCF servislerinin Workflow tarafındaki TransactionScope kontrolüne dahil edilme durumlarını incelemeye çalışmıştık hatırlayalım.

Workflow Foundation, Oracle, WCF ve TransactionScope

Senaryo

Bu seferki senaryomuz biraz daha farklı ve karışık Elimizde iki adet servis bulunmakta. Her ikisi de kendi içerisindeki operasyonlarında, **Oracle** tabanlı bir veritabanı üzerine **insert** işlemi gerçekleştirmekte*(Senaryo gereği insert, ama tabiki her tür CRUD operasyonu söz konusu olabilir)*

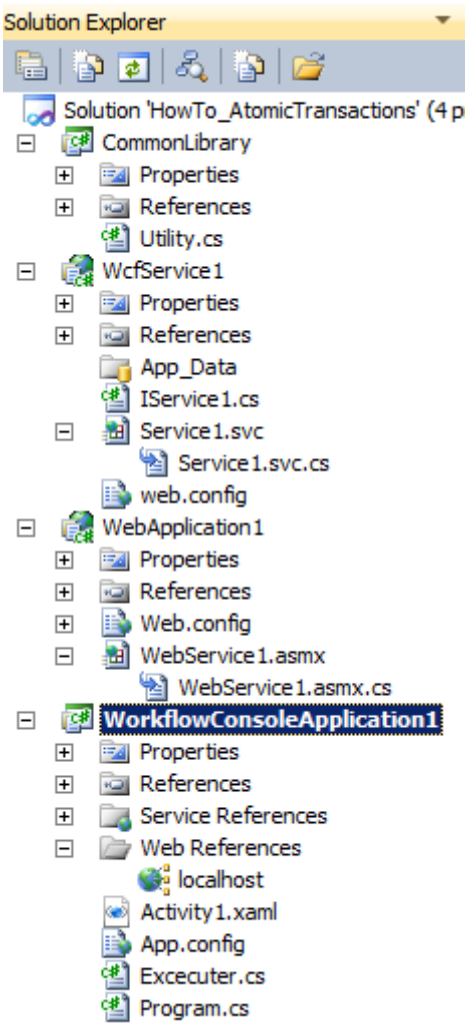
Ne varki bu servislerden birisi **Windows Communication Foundation** yapısında iken diğeri eski**XML Web Service** formatında tasarlanmış durumdalar. Senaryomuzda bu iki servisi kendi bünyesinde birleştiren bir de **Workflow Activity** bulunuyor. Çok doğal olarak bu bileşenin içerisinde ele alacağımız bir **TransactionScope** kontrolü de yer almakta. Hal böyle olunca**TransactionScope** içerisinde üretilen **Transaction'** ın servisler arasında nasıl akacağı, büyük soru işareti olarak karşımıza çıkıyor. Aslında senaryoyu aşağıdaki şekle bakarak kafamızda daha iyi canlandırabileceğimizi düşünüyorum.



WCF servislerinin kullanıldığı senaryolarda **atomic transaction'** ların servis içerisine nasıl akıtılacağını bu yazımızda incelemiştik hatırlayacağınız üzere. Ancak işin içerisine eski stilde yazılmış bir **XML Web Service** girince, durum biraz farklılaşıyor 😜
Ne yazık ki, istemci tarafında başlatılan **Transaction'** ın kod yardımıyla **XML Web Service**içerisindeki ilgili **Web Method'** a aktarılması gerekmektedir. Aksi durumda bir çalışma zamanı hatası alınmıyor olmasına karşın, bir dağıtık **Transaction'** ın ilgili **Web Method** içerisindeki**CRUD*(Create Retrieve Update Delete)*** işlemini ele alamadığı görülür. Bu görünmez hata fark edilmediği takdirde, kötü sonuçlara neden olabilir tahmin edeceğiniz üzere.

Örnek

Öyleyse gelin yola koyulalım ve adım adım **Solution** içeriğimizi inşa ederek teste çıkalım. Çözümümüz içerisinde bir **ortak fonksiyonellik kütüphanesi**, bir **WCF Servis** uygulaması, bir**XML Web Servis** içeren **Asp.Net Web uygulaması** ve son olarak da bir adet **Workflow Console**projesi bulunmakta.



İlk olarak **WCF servis** tarafını geliştiriyor olacağız.

Servis sözleşmesi

```
using System.ServiceModel;
namespace WcfService1
{
   [ServiceContract]
   public interface IService1
   {
      [OperationContract]
      [TransactionFlow( TransactionFlowOption.Mandatory)]
      int InsertAccount(int accountId, string name, string surname);
   }
}
```

Servis kodları

```
using System.Configuration;
using System.ServiceModel;
using System.Transactions;
```

```
using CommonLibrary;
using Oracle.DataAccess.Client;
namespace WcfService1
{
   public class Service1
      : IService1
   {
      [OperationBehavior(TransactionScopeRequired = true,
TransactionAutoComplete = true)]
      public int InsertAccount(int accountId,string name,string surname)
      {
         int result = 0;
         Utility.Log(Transaction.Current);
         using (OracleConnection conn = new
OracleConnection(ConfigurationManager.ConnectionStrings["ConStr"].ConnectionString)
)
         {
            using (OracleCommand command = new OracleCommand("INSERT INTO
ACCOUNT (ACCOUNTID,NAME,SURNAME) VALUES
(:pACCOUNTID,:pNAME,:pSURNAME)", conn))
            {
               command.Parameters.Add(":pACCOUNTID", accountId);
               command.Parameters.Add(":pNAME", name);
               command.Parameters.Add(":pSURNAME", surname);
               conn.Open();
               result = command.ExecuteNonQuery();
            }
         }
         return result;
      }
   }
}
```

Servis Konfigurasyon içeriği

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <connectionStrings>
      <add name="ConStr" connectionString="User Id=...;Password=...;Data Source=..."
providerName="Oracle.DataAccess.Client"/>
   </connectionStrings>
   <system.serviceModel>
      <services>
```

```
        <service name="WcfService1.Service1">
          <endpoint address="http://localhost:12809/Service1.svc"
           binding="wsHttpBinding" bindingConfiguration="wsB"
contract="WcfService1.IService1" />
        </service>
      </services>
      <bindings>
        <wsHttpBinding>
          <binding transactionFlow="true" name="wsB"/>
        </wsHttpBinding>
      </bindings>
      <behaviors>
        <serviceBehaviors>
          <behavior name="">
            <serviceMetadata httpGetEnabled="true"/>
            <serviceDebug includeExceptionDetailInFaults="true"/>
          </behavior>
        </serviceBehaviors>
      </behaviors>
      <serviceHostingEnvironment multipleSiteBindingsEnabled="false"/>
    </system.serviceModel>
</configuration>
```

Servis metodu **Account** isimli bir tabloya veri girişi işlemini icra edecek şekilde tasarlanmıştır.**wsHttpBinding** bağlayıcı tipini kullanmaktadır. **Oracle** üzerinde bir veri işlemi söz konusu olması nedeniyle **ODP.Net'** in ilgili versiyonu ele alınmıştır.

Bu servis uygulamasında ve takip eden kod satırlarında belirteceğimiz XML Web Service uygulamasında, güncel Transaction bilgilerini fiziki bir dosyaya loglamak amacıyla bir kütüphane fonksiyonundan yararlanılmaktadır. Utility sınıfına ait static Log metodunun içeriği aşağıdaki gibidir.

Log bilgisinin en önemli parçalarından birisi de DistributedIdentifier özelliğinin değeridir. Bu değer ilk başlatıldığı noktada ne ise, diğer servis çağrıları içerisinde de aynı olmalıdır ki ortak bir dağıtık transaction yapısından söz edilebilsin.

```
using System.IO;
using System.Transactions;
namespace CommonLibrary
{
  public class Utility
  {
    public static void Log(Transaction current)
    {
      if (current != null)
```

```
        {
            var information = Transaction.Current.TransactionInformation;
            string report = string.Format("Time:{0},Isolation Level:{1},Distributed
ID:{2},Local ID:{3},Status:{4}\n"
                , information.CreationTime
                , Transaction.Current.IsolationLevel.ToString()
                , information.DistributedIdentifier.ToString()
                , information.LocalIdentifier
                , information.Status
                );
            File.AppendAllLines("c:\\Log.txt", new string[] { report });
        }
    }
  }
}
```

Gelelim **XML Web Service** tarafına. Eminim ki uzun süredir bir **XML Web Servis** yazmıyor veya tüketmiyorsunuzdur  Ama çalıştığınız kurum da eski sistemler söz konusu ise bundan kaçış yolunuz olmadığını ifade edebilirim.

XML Web Servis kodları

```
using System.Configuration;
using System.EnterpriseServices;
using System.Transactions;
using System.Web.Services;
using CommonLibrary;
using Oracle.DataAccess.Client;
namespace WebApplication1
{
  [WebService(Namespace = "http://tempuri.org/")]
  [WebServiceBinding(ConformsTo = WsiProfiles.BasicProfile1_1)]
  [System.ComponentModel.ToolboxItem(false)]
  public class WebService1 : System.Web.Services.WebService
  {
    [WebMethod(TransactionOption= TransactionOption.Required)]
    public int InsertBranch(int branchId, string title, int code,byte[] propToken)
    {
      int result = 0;
      Transaction tx =
TransactionInterop.GetTransactionFromTransmitterPropagationToken(propToken);

      using (TransactionScope scope = new  TransactionScope(tx))
```

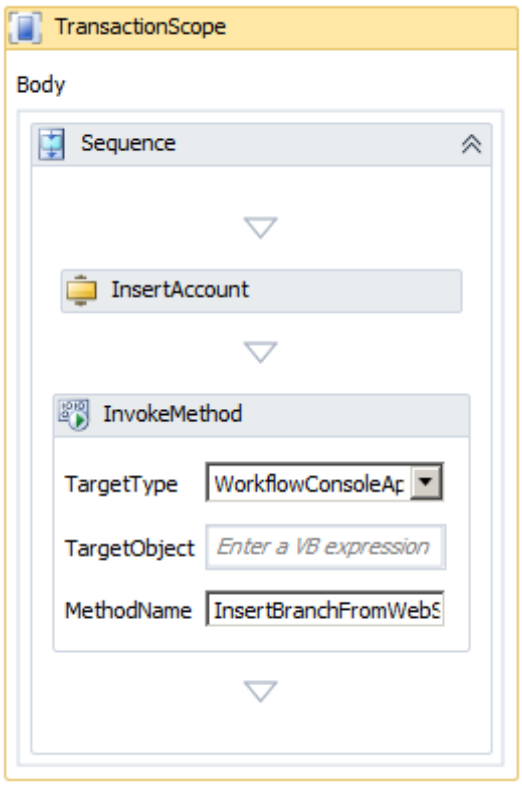
```
        {
            Utility.Log(Transaction.Current);
            using (OracleConnection conn = new
OracleConnection(ConfigurationManager.ConnectionStrings["ConStr"].ConnectionString)
)
            {
                using (OracleCommand command = new OracleCommand("INSERT INTO
BRANCH (BRANCHID,TITLE,CODE) VALUES (:pBRANCHD,:pTITLE,:pCODE)",
conn))
                {
                    command.Parameters.Add(":pBRANCHID", branchId);
                    command.Parameters.Add(":pTITLE", title);
                    command.Parameters.Add(":pCODE", code);
                    conn.Open();
                    result = command.ExecuteNonQuery();
                    //throw new Exception("Some error");
                }
            }
            scope.Complete();
        }
        return result;
    }
  }
}
```

Branch tablosuna insert işlemini icra eden **InsertBranch** web metodu içerisindeki en önemli kısım, parametre olarak gelen **byte[]** tipindeki **propToken** değişkeninin kullanılış şeklidir. Dikkat edileceği üzere bu değişken değerinden yararlanılarak istemci tarafından gelen **dağıtık transaction** tipi yakalanmakta ve onu baz alacak şekilde bir **TransactionScope** bloğu üretilmektedir. Bu sayede istemci tarafındaki **Transaction Scope'** un başlattığı **dağıtık transaction'** a dahil olunabilecektir. **Pek tabi kodumuz içerisinde daha sonradan yapacağımız testler için ele alacağımız bir yorum satırı vardır** **Amaç olarak ilgili satırı test sırasında açıp bu web servis ve öncesinden gelen WCF servisi içerisindeki veritabanı operasyonlarının rollback edildiğini görmek istiyoruz.**

İstemci Tarafı

Öyleyse gelelim istemci tarafına. İstemci uygulamamız bildiğiniz üzere bir **Workflow** çağrısı gerçekleştiriyor olacak. Bu sebepten bir **Workflow Console Application** projesinden yararlanabiliriz. Söz konusu projeye hem **WCF** hem de **XML Web servislerini** referans olarak eklememiz gerekiyor.

XML Web Service referansını eklemek için Add Service Reference->Advanced sekmesine geçip oradaki Add Web Reference düğmesini kullanmanız gerektiğini unutmayın. Yoksa Web Servisinizi de WCF servis gibi eklenmiş bulursunuz. Workflow uygulamamızda aşağıdaki **TransactionScope** bileşenini içeren bir **Flow Chart** söz konusudur.



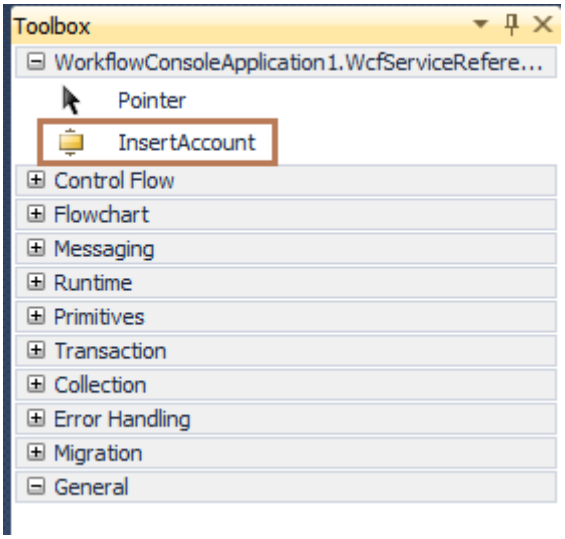
*Akışa ait **XAML** içeriğinde özellikle bold olan alanara dikkat edin. Örneği yazarken işinize yarayacaktır.*

```
<Activity mc:Ignorable="sads sap" x:Class="WorkflowConsoleApplication1.Activity1"
sap:VirtualizedContainerService.HintSize="654,676"
mva:VisualBasic.Settings="Assembly references and imported namespaces for internal
implementation"
xmlns="http://schemas.microsoft.com/netfx/2009/xaml/activities"
xmlns:av="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:local="clr-namespace:WorkflowConsoleApplication1"
xmlns:local1="clr-
namespace:WorkflowConsoleApplication1.WcfServiceReference1.Activities"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:mv="clr-namespace:Microsoft.VisualBasic;assembly=System"
xmlns:mva="clr-
namespace:Microsoft.VisualBasic.Activities;assembly=System.Activities"
xmlns:p="http://schemas.microsoft.com/netfx/2009/xaml/servicemodel"
xmlns:s="clr-namespace:System;assembly=mscorlib"
xmlns:s1="clr-namespace:System;assembly=System"
```

```
xmlns:s2="clr-namespace:System;assembly=System.Xml"
xmlns:s3="clr-namespace:System;assembly=System.Core"
xmlns:s4="clr-namespace:System;assembly=System.ServiceModel"
xmlns:sa="clr-namespace:System.Activities;assembly=System.Activities"
xmlns:sad="clr-namespace:System.Activities.Debugger;assembly=System.Activities"
xmlns:sads="http://schemas.microsoft.com/netfx/2010/xaml/activities/debugger"
xmlns:sap="http://schemas.microsoft.com/netfx/2009/xaml/activities/presentation"
xmlns:sc="clr-namespace:System.ComponentModel;assembly=System"
xmlns:scg="clr-namespace:System.Collections.Generic;assembly=System"
xmlns:scg1="clr-
namespace:System.Collections.Generic;assembly=System.ServiceModel"
xmlns:scg2="clr-namespace:System.Collections.Generic;assembly=System.Core"
xmlns:scg3="clr-namespace:System.Collections.Generic;assembly=mscorlib"
xmlns:sd="clr-namespace:System.Data;assembly=System.Data"
xmlns:sl="clr-namespace:System.Linq;assembly=System.Core"
xmlns:st="clr-namespace:System.Text;assembly=mscorlib"
xmlns:ww="clr-namespace:WorkflowConsoleApplication1.WcfServiceReference1"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Flowchart
sad:XamlDebuggerXmlReader.FileName="d:\users\bsenyurt\documents\visual studio
2012\Projects\HowTo_AtomicTransactions\WorkflowConsoleApplication1\Activity1.xaml
" sap:VirtualizedContainerService.HintSize="614,636">
    <Flowchart.Variables>
      <Variable x:TypeArguments="x:Int32" Name="InsertAccountExceuteResult" />
      <Variable x:TypeArguments="x:Int32" Name="InsertBranchExecuteResult" />
    </Flowchart.Variables>
    <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">False</x:Boolean>
        <av:Point x:Key="ShapeLocation">270,2.5</av:Point>
        <av:Size x:Key="ShapeSize">60,75</av:Size>
        <av:PointCollection x:Key="ConnectorLocation">300,77.5
300,160.5</av:PointCollection>
      </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <Flowchart.StartNode>
      <FlowStep x:Name="__ReferenceID0">
        <sap:WorkflowViewStateService.ViewState>
          <scg3:Dictionary x:TypeArguments="x:String, x:Object">
            <av:Point x:Key="ShapeLocation">195,160.5</av:Point>
            <av:Size x:Key="ShapeSize">210,59</av:Size>
```

```
        </scg3:Dictionary>
      </sap:WorkflowViewStateService.ViewState>
      <TransactionScope sap:VirtualizedContainerService.HintSize="266,396">
        <sap:WorkflowViewStateService.ViewState>
          <scg3:Dictionary x:TypeArguments="x:String, x:Object">
            <x:Boolean x:Key="IsExpanded">True</x:Boolean>
          </scg3:Dictionary>
        </sap:WorkflowViewStateService.ViewState>
        <Sequence sap:VirtualizedContainerService.HintSize="230,315">
          <sap:WorkflowViewStateService.ViewState>
            <scg3:Dictionary x:TypeArguments="x:String, x:Object">
              <x:Boolean x:Key="IsExpanded">True</x:Boolean>
            </scg3:Dictionary>
          </sap:WorkflowViewStateService.ViewState>
          <local1:InsertAccount sap:VirtualizedContainerService.HintSize="208,22"InsertAccountResult="[InsertAccountExceuteResult]" mva:VisualBasic.Settings="Assembly references and imported namespaces serialized as XML namespaces" accountId="9000" name="Burak" surname="Senyurt" />
          <InvokeMethod sap:VirtualizedContainerService.HintSize="208,129"MethodName="InsertBranchFromWebService" TargetType="local:Excecuter">
            <InvokeMethod.Result>
              <OutArgument x:TypeArguments="x:Int32">[InsertBranchExecuteResult]</OutArgument>
            </InvokeMethod.Result>
            <InArgument x:TypeArguments="x:Int32">8888</InArgument>
            <InArgument x:TypeArguments="x:String">Çınaraltı</InArgument>
            <InArgument x:TypeArguments="x:Int32">40</InArgument>
          </InvokeMethod>
        </Sequence>
      </TransactionScope>
    </FlowStep>
  </Flowchart.StartNode>
  <x:Reference>__ReferenceID0</x:Reference>
 </Flowchart>
</Activity>
```

Ne yazık ki **XML Web Service'** ler referans olarak bir **Workflow** projesine eklendiklerinde, **Toolbox**üzerinde aynı **WCF** Servislerinde olduğu gibi bir **component** olarak görülmemektedir. Tabi bunu Visual Studio 2012 için konuştuğumuzu tekrardan hatırlatalım. *(Eskiye olan support' un kalktığını bu noktada bariz bir şekilde görebiliriz aslında)*



Dolayısıyla ilgili **XML Web Servis'** ini çağırmak için belki bir **InvokeMethod** bileşeninden yararlanabiliriz ki o da içeride **static** bir tip metodu kullanacaktır. Bu metod **Executer** isimli sınıf içerisinde aşağıdaki gibi tanımlanmıştır.

```
using System.Transactions;
using WorkflowConsoleApplication1.localhost;
namespace WorkflowConsoleApplication1
{
   public class Excecuter
   {
      public static int InsertBranchFromWebService(int branchId,string title,int code)
      {
         WebService1 service = new WebService1();
         byte[] propToken =
TransactionInterop.GetTransmitterPropagationToken(Transaction.Current);
         return service.InsertBranch(branchId, title, code, propToken);
      }
   }
}
```

Metoda ait kod içeriği oldukça kritiktir. Görüldüğü üzere **TransactionInterop** tipinin **staticGetTransmitterPropagationToken** metoduna o anki **Transaction** nesne örneği gönderilerek bir**byte** dizi içeriğinin elde edilmesi söz konusudur. Bu şekilde, aslında **TransactionScope** bileşeni içerisine dahil olan ve **WCF Servis** çağrısı yoluyla başlatılan **Transaction'** ın **XML Web Servis**metodu içerisine parametre olarak bildirilebilmesi mümkün hale gelmektedir.

Workflow Console uygulamasının **Main** metoduna ait içerik ise son derece standarttır.

```
using System;
using System.Activities;
namespace WorkflowConsoleApplication1
{
```

```
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            Activity wf = new Activity1();
            WorkflowInvoker.Invoke(wf);
         }
         catch (Exception excp)
         {
            Console.WriteLine(excp.Message);
         }
      }
   }
}
```

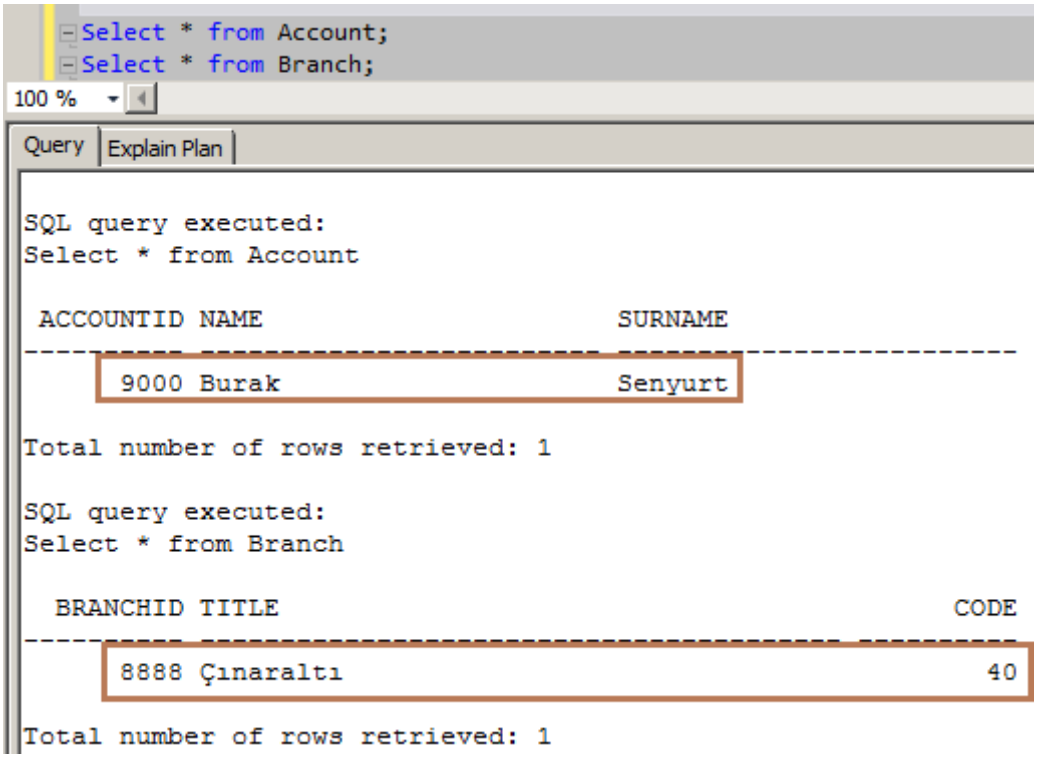
Testler

Artık testlerimize başlayabiliriz. Senaryoyu doğrudan bu şekilde işlettiğimizde, **C** dizini altında üretilen **Log.txt** dosyasında aşağıdakine benzer bir içeriğin oluşturulacağı gözlemlenecektir.

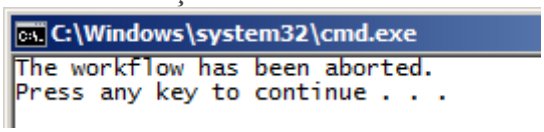
Time:09.08.2012 09:28:45,Isolation Level:Serializable,**Distributed ID:2cef4851-a7fe-41e3-a42a-cf4aab1aa9fe**,Local ID:632e8dc4-5446-4531-9426-05f092527d54:1,Status:Active

Time:09.08.2012 09:28:45,Isolation Level:Serializable,**Distributed ID:2cef4851-a7fe-41e3-a42a-cf4aab1aa9fe**,Local ID:16135af3-ce46-42c3-9acf-ec5dc5d43275:1,Status:Active

Bu çıktılardan ilki **WCF** servis metodu, ikincisi ise **XML Web Servis** metodu içerisinde gelmektedir. Dikkat edileceği üzere her iki çağrı içinde aynı **Distributed Transaction ID** değeri üretilmiştir. Eğer veritabanına gidilirse, icra edilen **insert** işlemlerinin her iki tablo içinde başarılı bir şekilde yapıldığı görülebilir.



Şimdi **XML Web servis** metodu içerisindeki sihirli yorum satırımızı açalım ve testimizi tekrardan yapalım. Çalışma ortamına düşen **exception** mesajı aşağıdaki gibi olacaktır. **TransactionScope**kontrolünün **AbortInstanceOnTransactionFlow** özelliğinin değeri varsayılan olarak **true**olduğundan, **Web Servis** içerisinden gelen **Fault Exception** nedeni ile, akışa ait nesne örneğinin çalışması otomatikman durdurulmuştur.



Log içeriğine bakıldığında ise **Distribute Transaction'** ın yine başarılı bir şekilde oluşturulduğu ve her iki servis çağrısında da, aynı **ID** değerlerini kullanıldığı görülebilir.
Time:09.08.2012 09:32:35,Isolation Level:Serializable,**Distributed ID:6dd28d89-aa10-4a97-8b32-e3439f0374ff**,Local ID:632e8dc4-5446-4531-9426-05f092527d54:2,Status:Active
Time:09.08.2012 09:32:35,Isolation Level:Serializable,**Distributed ID:6dd28d89-aa10-4a97-8b32-e3439f0374ff**,Local ID:384682c2-e4e1-47af-bc8e-f28c5abd7229:1,Status:Active
Ancak veritabanına gidilip ilgili tablolar sorgulandığında, iki **Insert** işleminin de yapılmadığı gözlemlenecektir. Bir başka deyişle istediğimiz durum oluşmuş ve **Transaction** işlemleri **Abort**edilerek o ana kadar yapılan ne kadar veritabanı işlemi var ise onaylanmamıştır
Görüldüğü üzere biraz kodlama yardımıyla **WCF** ve **XML Web Servislerini, TransactionScope**bileşeni içerisinde bir arada kullanabildik. Tabiki senaryonun genişletilmesi ve geliştirilmesi gerekiyor. Örneğin Savepoint kullanımları durumu var. Ya da **Long Running Process** söz konusu ise **Persistence** sisteminin bu tip vakalarda nasıl

tepki vereceği. Hatta daha da zor bir senaryo var. Ya bu **XML Web Service'** ler daha önceden yazılmışlar ve sizin müdahale alanınız dışındaysalar Şimdilik bu kötü kokan vakaları bir kenara bırakıp önümüze bakalım derim. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

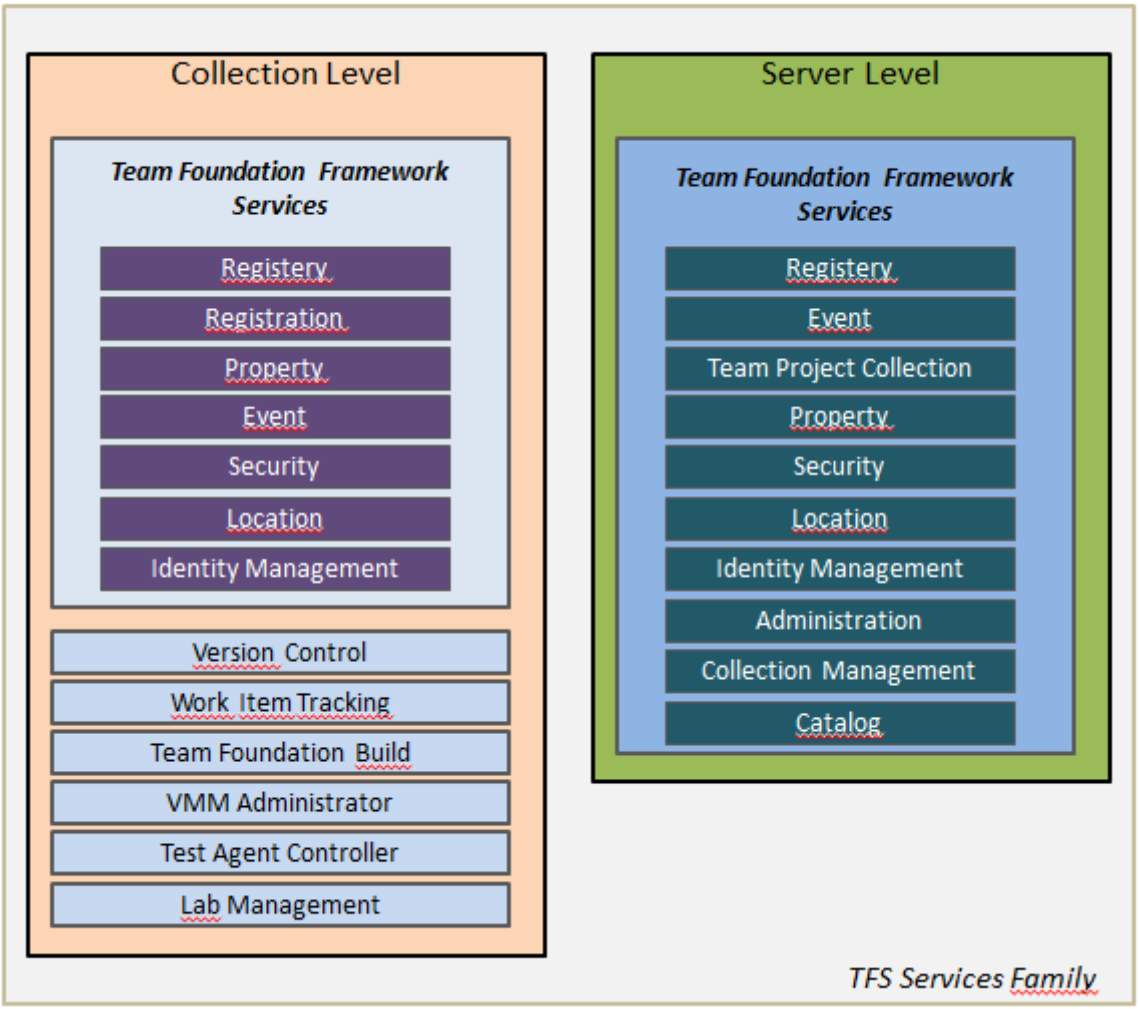
HowTo_AtomicTransactions.zip (117,25 kb)

[Örnek Visual Studio 2010, .Net Framework 4.0 tabanlıdır]

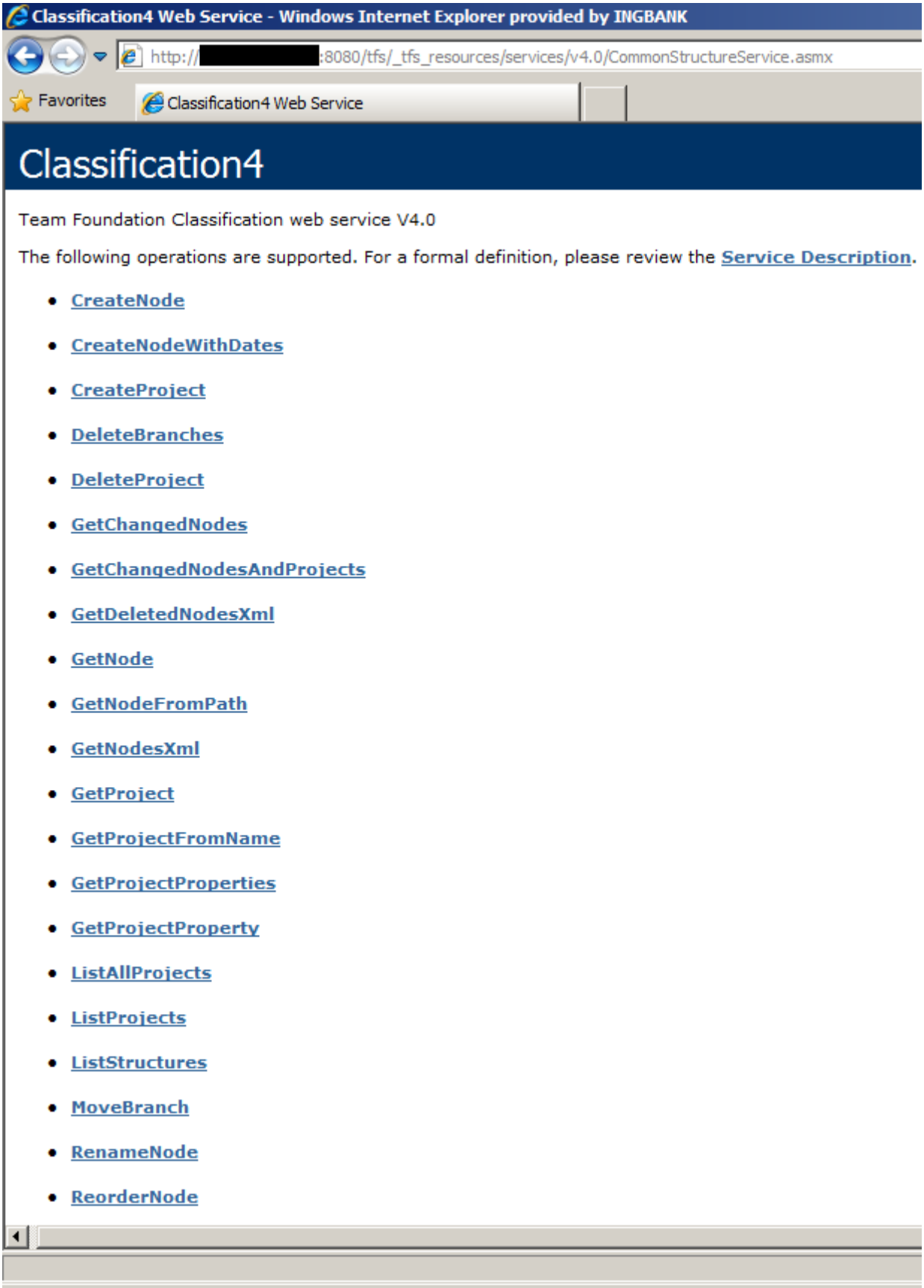
TFS Web Services ve Kullanımları

Pazartesi, 18 Mart 2013 11:30 by bsenyurt

Merhaba Arkadaşlar,
Yedek Subay olarak askerlik hizmetimi yerine getirdiğim yıllarda*(O zamanlar 16 ay idi)* Jandarma Genel Komutanlığı Personel Şube' de görev almıştım. Aslında temel işim**Powerpoint** ile sunum hazırlamaktı ama verilen emir her ne ise onu da yerine getirmek mesuliyetini taşımaktaydım.
Bir gün komutanım ile birlikte yine sivil hayat için anlamsız olan ama Askeri disiplin kuralları çerçevesinde gayet de makul görünen bir işe adanmıştık. Neredeyse tüm komutanlık personelinin iğneli Printer' dan çıkartılmış karınca yazısı ebatlarındaki bilgilerini, bir diğer koca liste ile karşılaştıracak ve bir filtreleme işlemi gerçekleştirecektik. *(İşin yaklaşık olarak kesintisiz çalışma ile 48 saate varabileceğini biliyorduk)*
Takdir edersiniz ki o yıllarda bilgisayar kullanılıyor olmasına rağmen, komutanlığın ihtiyaç duyduğu ve hayatı kolaylaştıracak işlevsellikler için genellikle **Microsoft Office** ürünleri ele alınmaktaydı ama elimizde şöyle zırt diye filtreleme yapabileceğimiz bir **SQL/Oracle** ortamı da yoktu.*(Yıl 2001-2003 arası diyeyim)*
Hal böyle olunca elimize aldık çıktıları başladık tek tek karşılaştırmaya. Bir ara ben durup bunu daha kolay nasıl yapabiliriz diye düşünmeye başladığımı hatırlıyorum. Hatta o anlarda arkamda beliren Yarbay' ımın da sert bir ses tonu ile beni rüyamdan uyandırdığını…"Asteğmenimmmm!!!…"
Doğruyu söylemek gerekirse ne ben ne de komutanım tüm listeyi dolaşmak istemiyorduk. Kafa kafaya vererek güzel bir yol bulduk elbette Yol derken elimizde cetvel, kalem vesaire vardı. Ne bilgisayar ne de başka bir akıllı cihaz. O zaman anladım ki bazı işlerde kişiye büyük sabır gerekebiliyor. Örneğin **TFS*(Team Foundation Server)*** üzerinde kullanılan **XML Web Service**örneklerinin teker teker bulunup çıkartılması gibi
TFS mimari alt yapısı ve çevre etkileşimini incelediğimiz şu yazımızda **Client Object Model**' i kısaca anlamaya çalışmıştık. O makalede yer alan mimari çizime dikkatlice bakarsak eğer, **Client Object Model**' in aslında **TFS Web Service**' ler ile haberleştiğini görebiliriz. Aslına bakarsanız**Team Foundation Server** tarafında epeyce fazla sayıda **XML Web Service** yer almaktadır. Bu servisleri ana hatları ile değerlendirdiğimizde ise sunucu ve koleksiyon seviyesinde olmak üzere iki ana dala bölündüklerini görürüz.
Aşağıdaki şekilde bu servisler genel isimlendirmeler halinde ifade edilmektedir.



Dikkat edilmesi gereken notkalardan birisi de bazı servislerin her iki seviyede de yer alıyor olmasıdır. Ne varki bu servislerin alanları farklıdır. Sadece bir koleksiyon ve içeriği için kullanılabilecek olan hizmetler **Collection Level** grubunda yer almaktadır. Diğer taraftan tüm **TFS** sunucusunu ilgilendiren servisler de **Server Level** grubuna dahildir.
Yine dikkat edileceği üzere koleksiyon seviyesinde farklılaşan*(Team Project Collection örneklerine özel olan)* hizmetler bulunmaktadır. Örneğin **Version Control** veya **Lab Management** gibi. Normal şartlarda bu servislere, kurulu olan **TFS** ortamına erişim yetkisi olan istemcilerden ulaşılabilinmektedir. Örneğin koleksiyon seviyesinde kullanılabilen ve proje listesinin çekilmesi, proje oluşturması, silinmesi, branch silinmesi vb operasyonları ele alabildiğimiz **Common Structure Service** hizmetinin 4ncü versiyonuna aşağıdaki şekilde görüldüğü gibi ulaşabiliriz.



Çok doğal olarak diğer servislere de ulaşmamız, hatta **WSDL*(Web Service Description Language)***çıktılarına bakmamız mümkündür. Ben çalışmakta olduğum **Team Foundation Server 2012** için, uygulama sunucusunun yüklü olduğu **IIS*(Internet Information***

Services) altında yaptığım araştırmalarda, aşağıdaki uzun listeye ulaştığımı rahatlıkla ifade edebilirim.

Administration
http://tfsserver:8080/tfs/TeamFoundation/Administration/v4.0/AccessControlService.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v4.0/FileHandlerService.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v4.0/IdentityManagementService2.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v3.0/AdministrationService.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v3.0/CatalogService.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v3.0/EventService.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v3.0/IdentityManagementService.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v3.0/JobService.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v3.0/LocationService.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v3.0/PropertyService.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v3.0/RegistryService.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v3.0/SecurityServices.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v3.0/TeamProjectCollectionService.asmx http://tfsserver:8080/tfs/TeamFoundation/Administration/v3.0/WarehouseControlService.asmx
Lab
http://tfsserver:8080/tfs/TeamFoundation/Lab/v3.0/LabFrameworkService.asmx
TFS Resources – Services
http://tfsserver:8080/tfs/_tfs_resources/services/v1.0/AuthorizationService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v1.0/CommonStructureService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v1.0/ConnectedServicesService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v1.0/EventService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v1.0/GroupSecurityService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v1.0/ProcessConfigurationService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v1.0/ProcessTemplate.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v1.0/ProjectMaintenance.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v1.0/registration.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v1.0/ServerStatus.asmx

http://tfsserver:8080/tfs/_tfs_resources/services/v1.0/StrongBoxService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v1.0/TeamConfigurationService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v2.0/GroupSecurityService2.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v2.0/ProcessConfigurationService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v2.0/ProcessConfigurationService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v3.0/AuthorizationService3.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v3.0/CommonStructureService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v3.0/IdentityManagementService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v3.0/JobService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v3.0/LocationService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v3.0/PropertyService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v3.0/RegistryService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v3.0/SecurityService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v3.0/SyncService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v4.0/AccessControlService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v4.0/AuthorizationService4.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v4.0/CommonStructureService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v4.0/FileHandlerService.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v4.0/IdentityManagementService2.asmx http://tfsserver:8080/tfs/_tfs_resources/services/v4.0/SyncService4.asmx
TFS Resources - Work Item Tracking
http://tfsserver:8080/tfs/_tfs_resources/workitemtracking/v5.0/clientservice.asmx http://tfsserver:8080/tfs/_tfs_resources/workitemtracking/v4.0/ClientService.asmx http://tfsserver:8080/tfs/_tfs_resources/workitemtracking/v3.0/ClientService.asmx http://tfsserver:8080/tfs/_tfs_resources/workitemtracking/v1.0/ClientService.asmx http://tfsserver:8080/tfs/_tfs_resources/workitemtracking/v1.0/ConfigurationSettingsService.asmx http://tfsserver:8080/tfs/_tfs_resources/workitemtracking/v1.0/ExternalServices.asmx http://tfsserver:8080/tfs/_tfs_resources/workitemtracking/v1.0/Integration.asmx
TFS Resources - Verison Control
http://tfsserver:8080/tfs/_tfs_resources/VersionControl/v4.0/Repository.asmx http://tfsserver:8080/tfs/_tfs_resources/VersionControl/v3.0/Repository.asmx http://tfsserver:8080/tfs/_tfs_resources/VersionControl/v1.0/Repository.asmx http://tfsserver:8080/tfs/_tfs_resources/VersionControl/v1.0/Administration.asmx http://tfsserver:8080/tfs/_tfs_resources/VersionControl/v1.0/Integration.asmx

http://tfsserver:8080/tfs/_tfs_resources/VersionControl/v1.0/ProxyStatistics.asmx
TFS Resources - Test Management
http://tfsserver:8080/tfs/_tfs_resources/TestManagement/v2.0/TestManagementWebService.asmx http://tfsserver:8080/tfs/_tfs_resources/TestManagement/v1.0/TestImpactService.asmx http://tfsserver:8080/tfs/_tfs_resources/TestManagement/v1.0/TestResults.asmx http://tfsserver:8080/tfs/_tfs_resources/TestManagement/v1.0/TestResultsEx.asmx
TFS Resources - Sync
http://tfsserver:8080/tfs/_tfs_resources/sync/v4.0/AdministrationService.asmx http://tfsserver:8080/tfs/_tfs_resources/sync/v3.0/AdministrationService.asmx
TFS Resources - Lab
http://tfsserver:8080/tfs/_tfs_resources/lab/v4.0/LabService.asmx http://tfsserver:8080/tfs/_tfs_resources/lab/v3.0/Integration.asmx http://tfsserver:8080/tfs/_tfs_resources/lab/v3.0/LabAdminService.asmx http://tfsserver:8080/tfs/_tfs_resources/lab/v3.0/LabService.asmx http://tfsserver:8080/tfs/_tfs_resources/lab/v3.0/TestIntegrationService.asmx http://tfsserver:8080/tfs/_tfs_resources/lab/v3.0/WorkflowIntegrationService.asmx
TFS Resources - Discussion
http://tfsserver:8080/tfs/_tfs_resources/discussion/v1.0/discussionwebservice.asmx
TFS Resources - Build
http://tfsserver:8080/tfs/_tfs_resources/build/v4.0/AdministrationService.asmx http://tfsserver:8080/tfs/_tfs_resources/build/v4.0/AgentReservationService.asmx http://tfsserver:8080/tfs/_tfs_resources/build/v4.0/BuildDeploymentService.asmx http://tfsserver:8080/tfs/_tfs_resources/build/v4.0/BuildQueueService.asmx http://tfsserver:8080/tfs/_tfs_resources/build/v4.0/BuildService.asmx http://tfsserver:8080/tfs/_tfs_resources/build/v4.0/SharedResourceService.asmx http://tfsserver:8080/tfs/_tfs_resources/build/v3.0/AdministrationService.asmx http://tfsserver:8080/tfs/_tfs_resources/build/v3.0/AgentReservationService.asmx http://tfsserver:8080/tfs/_tfs_resources/build/v3.0/BuildQueueService.asmx http://tfsserver:8080/tfs/_tfs_resources/build/v3.0/BuildService.asmx http://tfsserver:8080/tfs/_tfs_resources/build/v3.0/Integration.asmx http://tfsserver:8080/tfs/_tfs_resources/build/v3.0/SharedResourceService.asmx

http://tfsserver:8080/tfs/_tfs_resources/build/v2.0/BuildService.asmx http://tfsserver:8080/tfs/_tfs_resources/build/v2.0/Integration.asmx
Metadata Publishing' i kapalı olan WCF servisleri
http://tfsserver:8080/tfs/queue/_tfs_resources/services/v4.0/MessageQueueService.svc http://tfsserver:8080/tfs/queue/_tfs_resources/services/v4.0/MessageQueueService2.svc

Görüldüğü üzere oldukça uzun bir servis listesi söz konusu.
Aslına bakarsanız bu servislerin detaylı olarak ne iş yaptıklarına dair MSDN üzerinde çok fazla bilgi bulunmamaktadır. Standart bir teknik Help dokümanından ötesi değildir. Bu nedenle TFS' in çalışma yapısını bilip, biraz tahminler yürüterek ilerlemeye çalışmak, işinizi epeyce kolaylaştıracaktır.

#Region Off Topic

Bu arada dilerseniz bu servislerin tamamı izlemek için basit bir Web uygulaması geliştirebilir ve örneğin listedeki adresleri bir **Web User Control** içerisine gömerek, tıklama usulüyle ilgili **XML Web Service** adreslerine gidebilirsiniz. Örneğin aşağıdaki gibi bir **Web User Control** ve yardımcı sınıf işinizi görecektir.
***Text** dosyasında durmakta olan servis adreslerini okuyan yardımcı tip ve metod;*

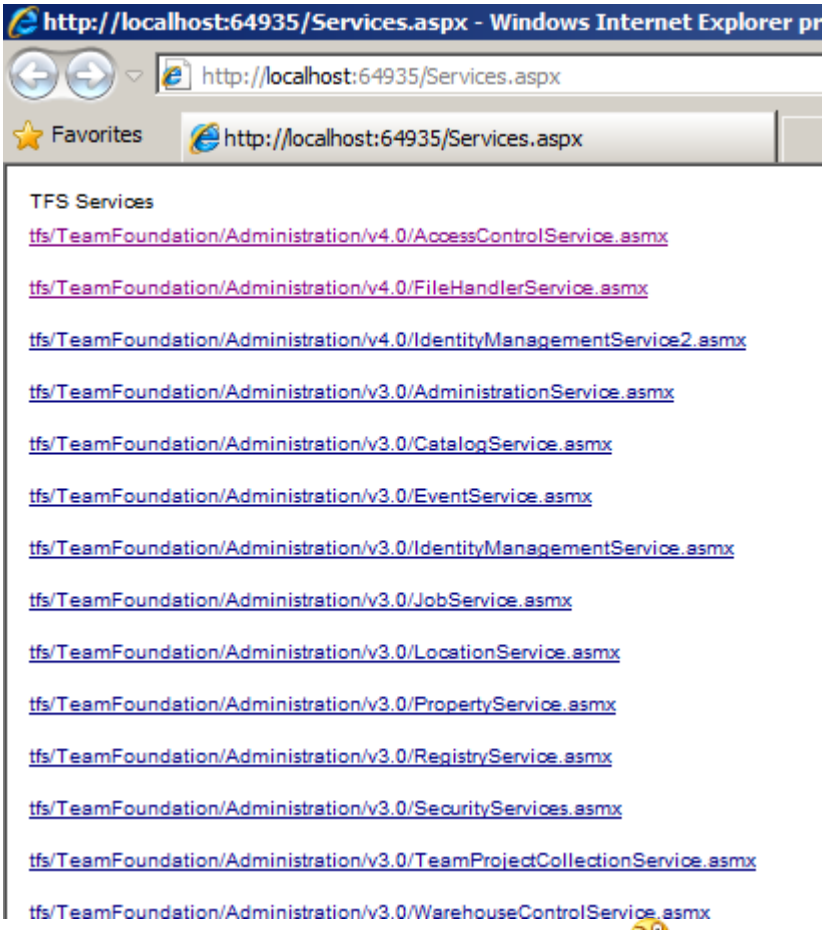
```
using System.Collections.Generic;
using System.IO;
using System.Web.UI.WebControls;
namespace AllServices
{
   public static class Utility
   {
      public static List<HyperLink> GetServiceLinks(string textFilePath)
      {
         List<HyperLink> links = new List<HyperLink>();
         string[] lines = File.ReadAllLines(textFilePath);
         foreach (string line in lines)
         {
            if (line.StartsWith("http://"))
            {
               HyperLink link = new HyperLink
               {
                  Text = line.Substring(line.LastIndexOf("tfs")),
                  NavigateUrl = line
               };
```

```
                links.Add(link);
            }
        }
        return links;
    }
  }
}
```

Web User Control kodu;

```
using System;
using System.Web.UI.WebControls;
namespace AllServices
{
  public partial class TfsServicesControl : System.Web.UI.UserControl
  {
    protected void Page_Load(object sender, EventArgs e)
    {
      var serviceLinks=Utility.GetServiceLinks(Server.MapPath("~/Services.txt"));
      foreach (var serviceLink in serviceLinks)
      {
        divLinks.Controls.Add(serviceLink);
        divLinks.Controls.Add(new Literal { Text = "<br/><br/>" });
      }
    }
  }
}
```

ve işte çalışma zamanına ait örnek ekran çıktısı 

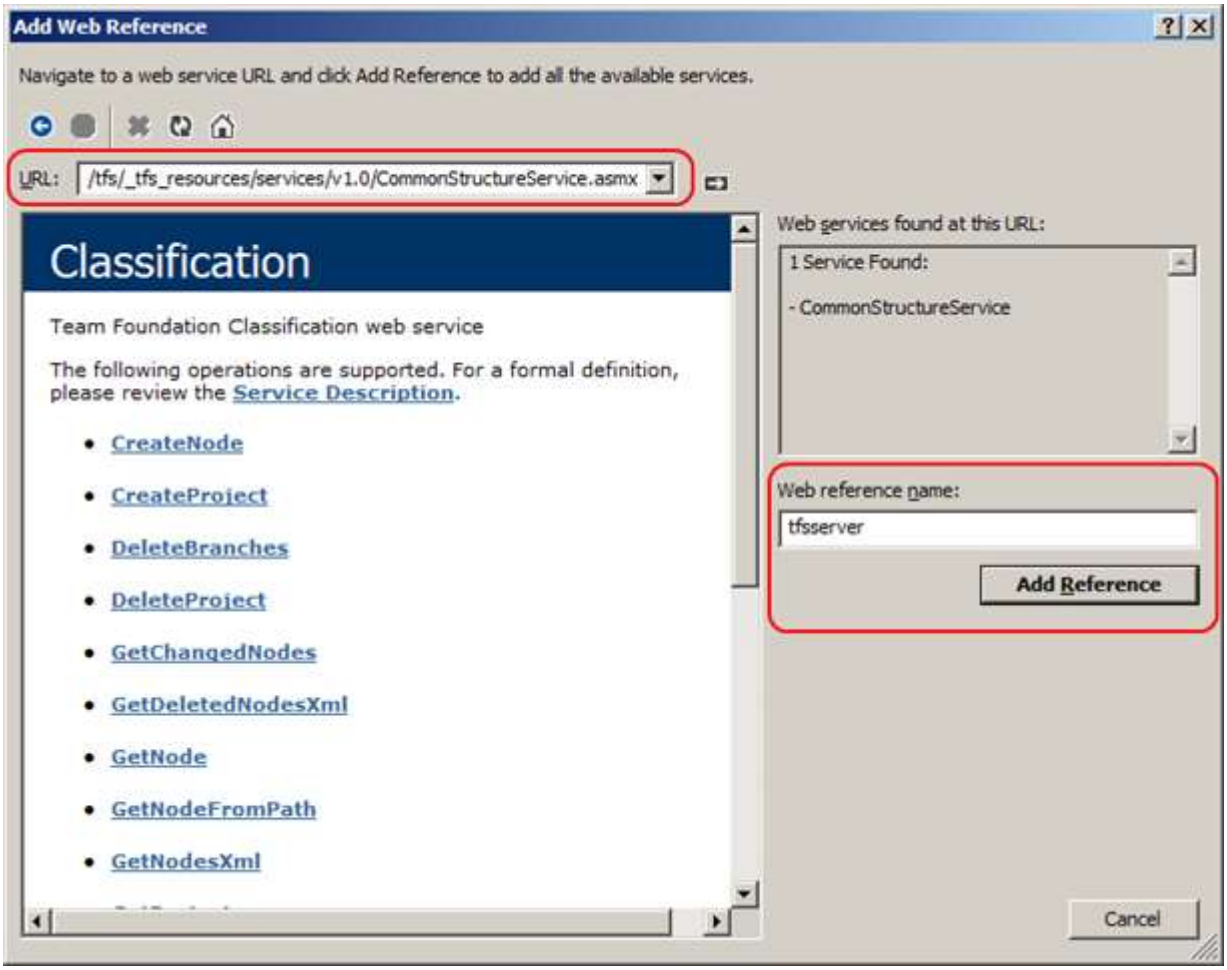


Bir servis adına tıklayın ve içeriğine ulaşın

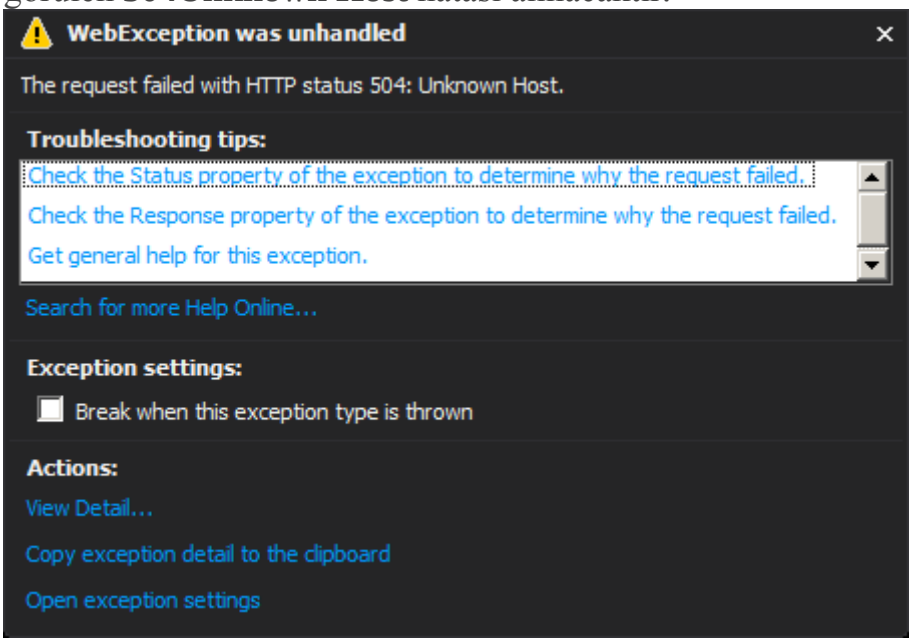
#endregion Off Topic

Referans Etmek

Team Foundation Server üzerinden sunulan **XML Web Service** örneklerini herhangibir **.Net**istemcisi tarafından tüketmek istediğimizde, **Client Object Model** üzerinden erişimde bulunmamız gerekmektedir. Normal şartlarda ilgili **Web Service'** leri, **Add Service Reference** sekmesinden hareket edilerek projeye ilave edilebilir ve üretilen **Proxy** tipinin metodlarına ulaşılabilir.

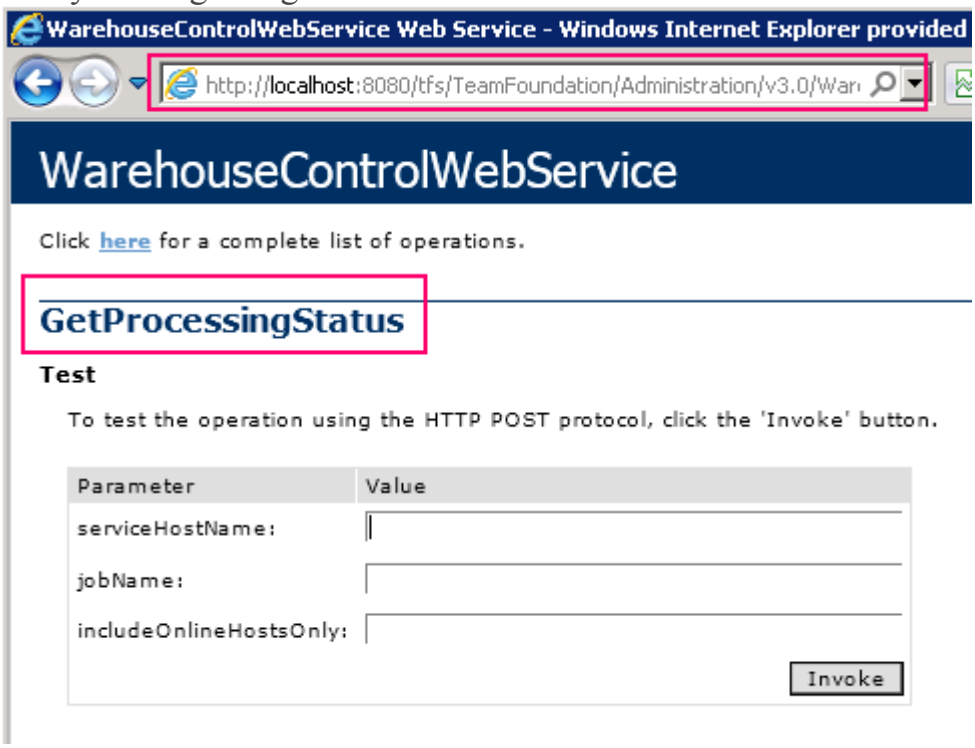


Ancak çalışma zamanında servis tarafında kuvvetle muhtemel aşağıdaki şekilde görülen **504Unknown Host** hatası alınacaktır.



Bunun nedeni aslında ilgili servislerin, **Client Object Model*(veya Server Object Model)***tafaından ele alınış ve üretiliş şekilleridir. Ancak unutulmaması gereken bir nokta da şudur. Bu servislerin çoğu, TFS'in kurulu makine üzerinden erişilmeye çalışıldığında(en azından bir tarayıcı ile) bildiğimiz **XML Web Service**' leri gibi tüketilebilirler.

Örneğin **TFS**' in kurulu olduğu makinede**http://tfsserver:8080/tfs/TeamFoundation/Administration/v3.0/WarehouseControlService.asmx**' e ulaşmayı deneyin. *(Yani ilgili servislere kurulu olduğu makineden yerel olarak ulaşmayı)* Bu durumda aşağıdaki gibi bir operasyonu deneyebileceğinizi görebilirsiniz.



Dilerseniz TFS' e uzaktan bağlanacak bir istemci açısından olaya bakmaya devam edelim ve basit bir kullanım şeklini değerlendirerek ilerlemeye çalışalım.

Hello World

Aşağıdaki kod parçasında **Collection Level** grubundan iki örnek servisin kullanımına yer verilmiştir. Bu servislerden birisi **ICommonStructureSerice4**, diğeri ise **IProcessTemplatesarayüzleri*(Interface)*** tarafından taşınmaktadır. Dikkat edileceği üzere anahtar nokta**TfsTeamProjectCollection** tipinin örnekleniş şeklidir. Burada**TfsTeamProjectCollectionFactory** sınıfının static **GetTeamProjectCollection** metodundan yararlanılmakta olup, fonksiyona parametre olarak **TFS** sunucusundaki **Team Project Collection**' ın **HTTP** tabanlı adresi geçilmektedir.

Bildiğiniz üzere TFS kurulumunda aksi belirtilmedikçe mutlaka Default Collection isimli bir Team Project Collection oluşturulmaktadır.

Tabi ki **Server Level** servis gruplarını kullanmak da isteyebiliriz. Bu durumda**TfsConfigurationServer** ve **TfsConfigurationServerFactory** tiplerinden yararlanmamız gerekmetedir.

Gelelim örnek uygulama kodlarımıza. Bunun için herzaman olduğu gibi basit bir **Console**uygulamasından yararlanıyor olacağız. Eğer makinemizde **TFS Client Object**

Model veya ilgili **SDK**yüklüyse en azından **Microsoft.TeamFoundation.Client** ve**Microsoft.TeamFoundation.Common assembly**' larının referans edilmesi gerekmektedir. Ancak bazı servisler farklı **Assembly**' ların referans edilmesini gerektirebilir. Söz gelimi **WorkItemStore**için, **Microsoft.TeamFoundation.WorkItemTracking.Client.dll** isimli **assembly** referans edilir.

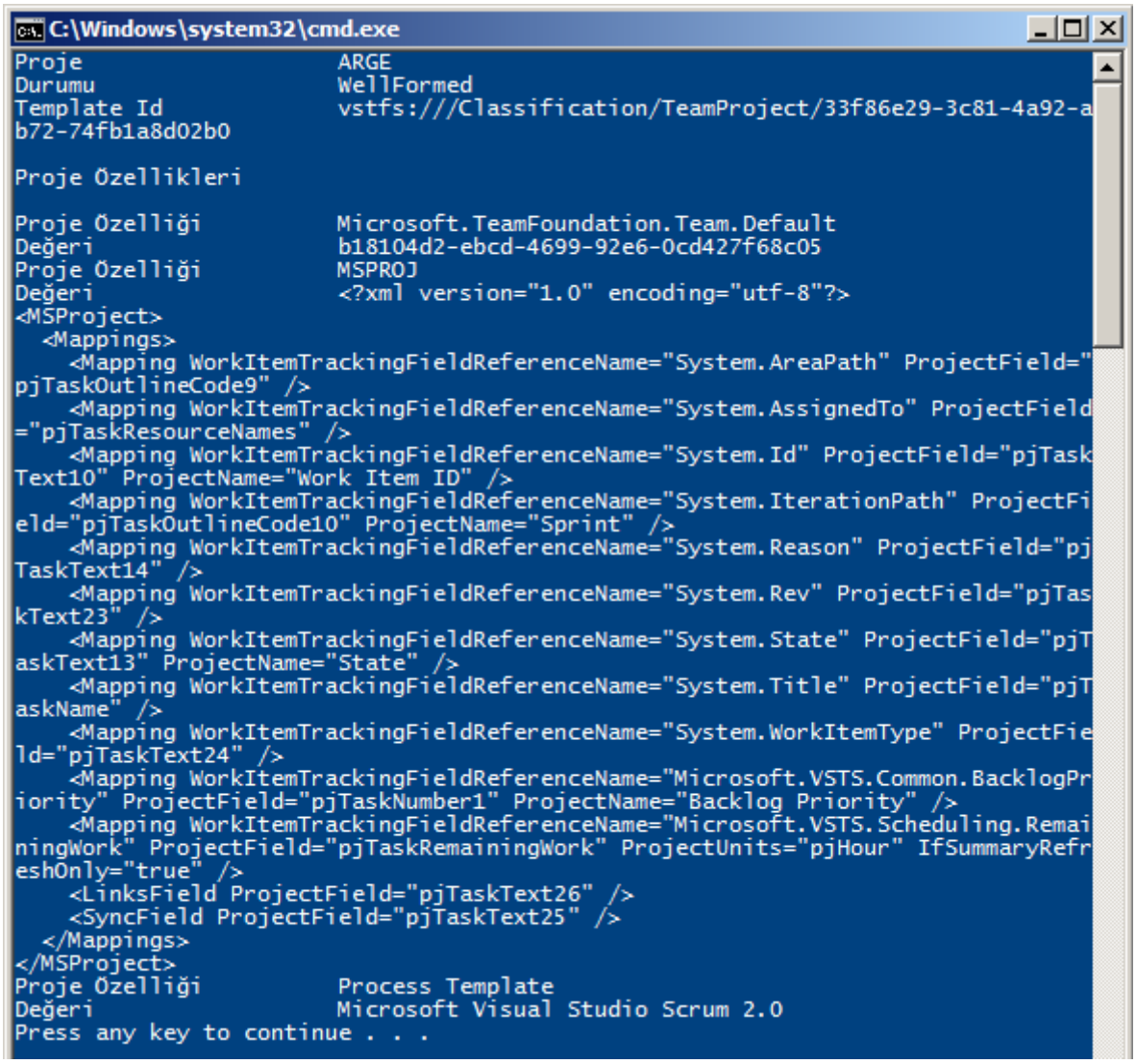
```
using Microsoft.TeamFoundation.Client;
using Microsoft.TeamFoundation.Server;
using System;
namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
        TfsTeamProjectCollection collection = TfsTeamProjectCollectionFactory
           .GetTeamProjectCollection(new
Uri("http://tfsserver:8080/tfs/defaultcollection"));
         #region Bazı servislerin çekilmesi
        ICommonStructureService4 commonStructureService =
(ICommonStructureService4)collection
           .GetService(typeof(ICommonStructureService4));
         IProcessTemplates processTemplateService =
(IProcessTemplates)collection.GetService<IProcessTemplates>();
         #endregion Bazı servislerin çekilmesi
         #region Bir Team Project adı üzerinden temel proje bilgisinin çekilmesi
         ProjectInfo argeInfo =
commonStructureService.GetProjectFromName("ARGE");
         Console.WriteLine("Proje\t\t\t{0}\nDurumu\t\t\t{1}\nTemplate Id\t\t{2}"
            , argeInfo.Name
            , argeInfo.Status
            , argeInfo.Uri.ToString()
            );
         string name;
         string state;
         int templateId;
         ProjectProperty[] projectProperties;
         // Tüm Proje özelliklerinin çekilmesi
        commonStructureService.GetProjectProperties(
            argeInfo.Uri.ToString()
            , out name
```

```
            , out state
            , out templateId
            , out projectProperties);
         Console.WriteLine("\nProje Özellikleri\n");
         foreach (var projectProperty in projectProperties)
            Console.WriteLine("Proje Özelliği\t\t{0}\nDeğeri\t\t\t{1}",
projectProperty.Name, projectProperty.Value);
         #endregion Bir Team Project adı üzerinden temel proje bilgisinin çekilmesi
         #region Sunucuda yüklü Process Template' lerin bilgilerinin elde edilmesi
         Console.WriteLine("\nProcess Templates\n");
         TemplateHeader[] templateHeaders =
processTemplateService.TemplateHeaders();
         foreach (TemplateHeader templateHeader in templateHeaders)
         {
            Console.WriteLine("Template
Id\t{0}\nAdı\t\t{1}\nRank\t\t{2}\nDurumu\t\t{3}\nMetadata\t{4}\nAçıklama\t{5}\n",
               templateHeader.TemplateId,
               templateHeader.Name,
               templateHeader.Rank,
               templateHeader.State,
               templateHeader.Metadata,
               templateHeader.Description
               );
         }
         #endregion Sunucuda yüklü Process Template' lerin bilgilerinin elde
edilmesi
      }
   }
}
```

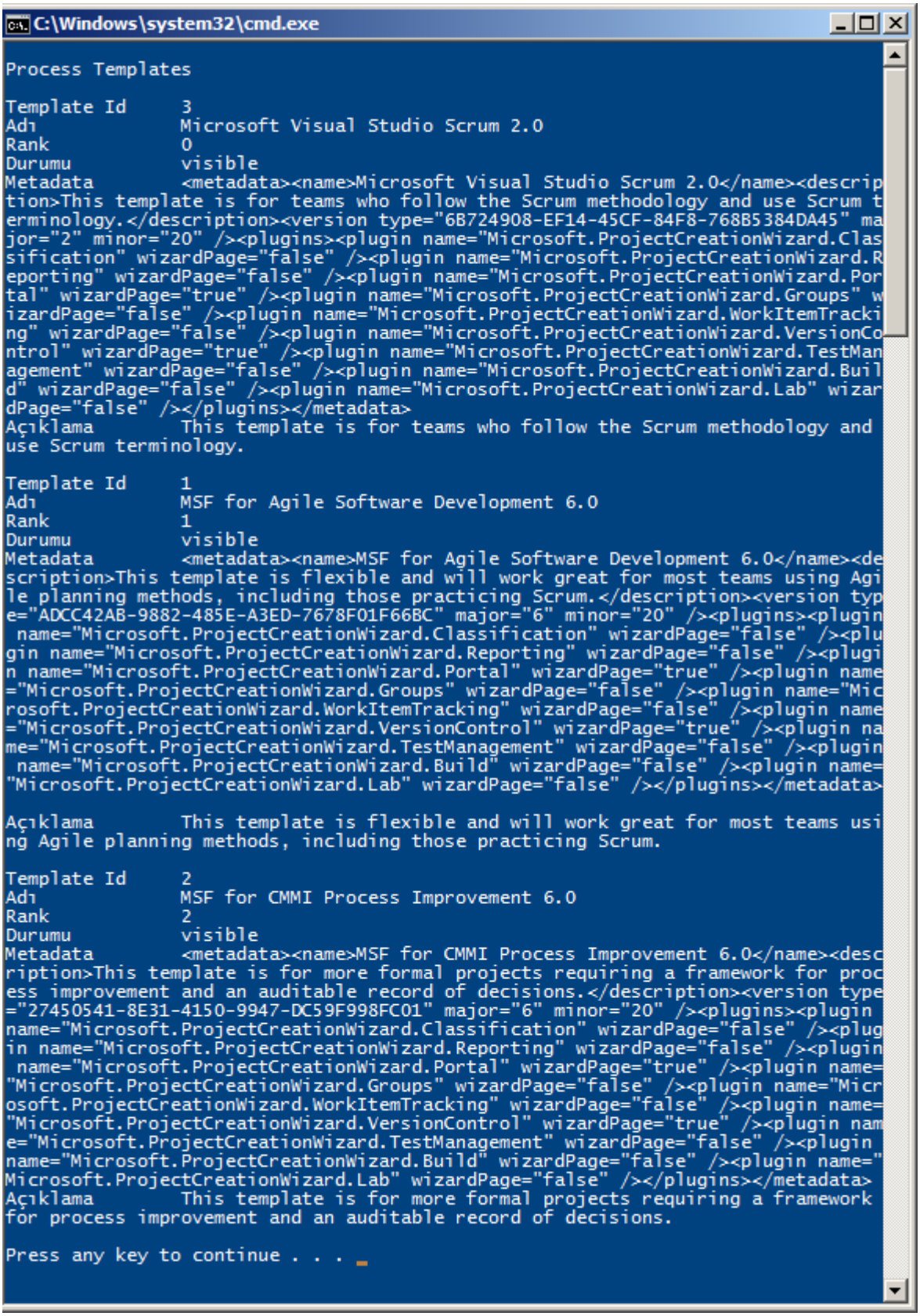
TfsTeamProjectCollection referansı elde edildikten sonra, alınmak istenen hizmete ait nesne örneğinin üretilmesi için **GetService** metodundan yararlanılmaktadır. Bu metodun **Type** parametresi ile çalışan versiyonu dışında **generic** olan bir versiyonu daha bulunmaktadır. Örneğin,

```
ICommonStructureService4 commonStructureService =
(ICommonStructureService4)collection
            .GetService(typeof(ICommonStructureService4));
```

kod satırı ile **Common Structure Service** örneği üretilmektedir. Bu adımdan sonra söz konusu servise ait referansın fonksiyonları kullanılabilir. Söz gelimi bir proje adı verilerek özelliklerinin elde edilmesi sağlanabilir. Bu özellikler arasında bir **Team Project**' in uyguladığı **Process Template**bilgisi ve hatta **XML** tabanlı şablon içeriği de yer almaktadır. Aşağıdaki örnek ekran çıktısında **ARGE**isimli projenin ulaşılan bilgileri gösterilmektedir.



Diğer yandan **IProcessTemplate** arayüzüne atanan servis referansının elde edilmesi için,**GetService** metodunun generic sürümünden yararlanılmıştır. Sonrasında ise **TFS** sunucusunda yüklü olan **Process Template** listesine gidilerek **Name**, **State**, **Metadata**, **Id**, **Description** gibi bilgileri elde edilmiştir. Aşağıdaki ekran çıktısına bakıldığında **Scrum 2.0, CMMI** ve **MSF** şablonlarının yüklenmiş olduğu bilgisine ulaşılabilinir.



Tabi burada akla takılan en önemli sorunlardan birisi kullanabileceğimiz **TFS Web Service**' lerinin kod tarafındaki **GetService** metodu tarafından kullanılabilecek karşılıklarının neler olduğudur? Bu konuda aşağıdaki listenin yardımcı olabileceğini düşünüyorum.

Servisin Adı	Collection Level	Server Level	Assembly	Namespace
ITeamFoundationRegistry	Var	Var	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Framework .Client
IIdentityManagementService	Var	Var	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Framework .Client
ITeamFoundationJobService	Var	Var	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Framework .Client
IPropertyService	Var	Var	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Framework .Client
IEventService	Var	Var	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Framework .Client
ISecurityService	Var	Var	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Framework .Client
ILocationService	Var	Var	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Framework .Client
TswaClientHyperlinkService	Var	Var	Microsoft .TeamFoundation	Microsoft .TeamFoundation

			.Client	.Framework .Client
ITeamProjectCollectionService	*Yok*	Var	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Framework .Client
IAdministrationService	Var	Var	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Framework .Client
ICatalogService	Yok	Var	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Framework .Client
VersionControlServer	Var	*Yok*	Microsoft .TeamFoundation .VersionControl .Client	Microsoft .TeamFoundation .VersionControl .Client
WorkItemStore	Var	*Yok*	Microsoft .TeamFoundation .WorkItemTracking .Client	Microsoft .TeamFoundation .WorkItemTrackin g .Client
IBuildServer	Var	*Yok*	Microsoft .TeamFoundation .Build .Client	Microsoft .TeamFoundation .Build .Client
ITestManagementService	Var	*Yok*	Microsoft .TeamFoundation .TestManagement .Client	Microsoft .TeamFoundation .TestManagement .Client
ILinking	Var	*Yok*	Microsoft .TeamFoundation	Microsoft .TeamFoundation

			.Common	
ICommonStructureService3	Var	*Yok*	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Server
IServerStatusService	Var	*Yok*	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Server
IProcessTemplates	Var	*Yok*	Microsoft .TeamFoundation .Client	Microsoft .TeamFoundation .Server

Yukarıdaki listede eksikliker olabilir. Gelen Update' ler, Service Pack' ler ve yeni sürümler sonrasın güncellenebilir. Lütfen MSDN üzerinden kontrol ediniz.
Demek ki, istemci tarafında çalışarak hayatımızı kolaylaştırmakta olan pek çok **.Net** tabanlı uygulama*(Örneğin **Power Tools** veya **Team Explorer** gibi)* söz konusu nesne modellerini ve ilgili**XML Web Service**' lerini kullanmaktadır. Elbette **Team Foundation Server** tarafında çok fazla sayıda **XML Web Service** metodu ve hatta versiyonu bulunmaktadır. **TFS** üzerindeki tecrübelerinizden yararlanarak bu servislerin efektif olarak kullanımlarını irdeleyebilirsiniz. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.
HowTo_TFSServices.zip (213,36 kb)

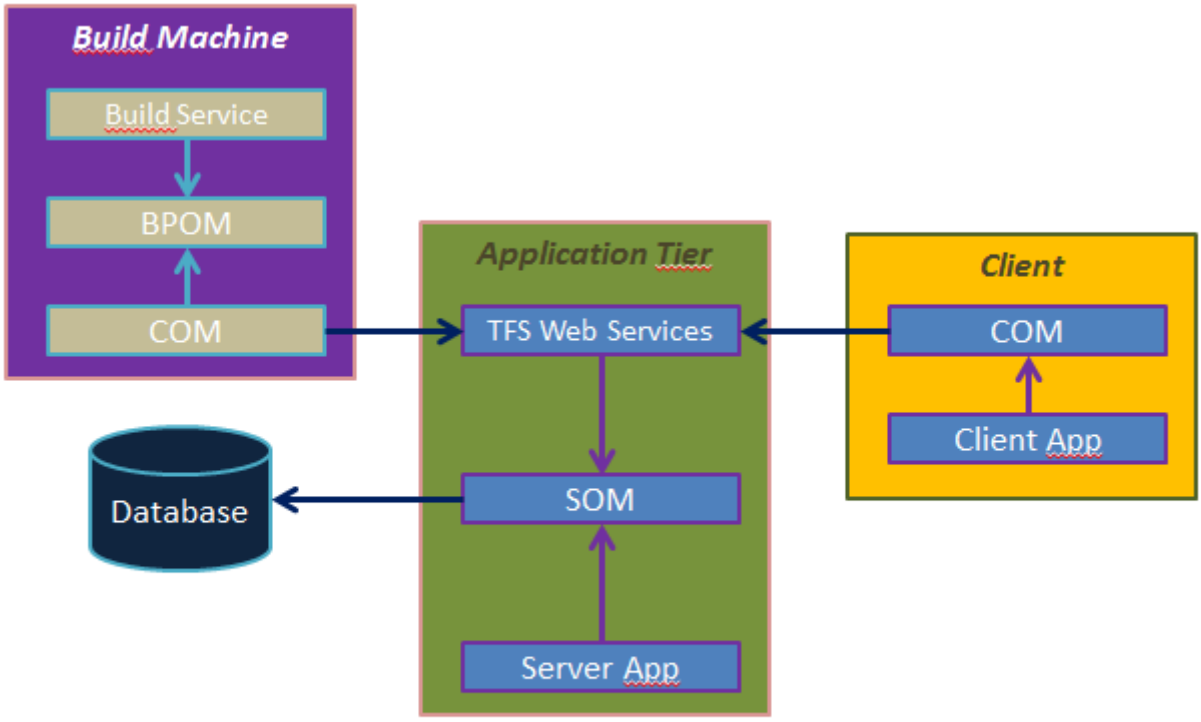
TFS–Client Object Model için Hello World

Cuma, 15 Mart 2013 15:30 by bsenyurt

Merhaba Arkadaşlar,
Çok eskidendi diyemeyeceğimiz kadar yakın bir zamanda, bilgisayar programcılarının ilah olduğu devirlerde, evimizin 37 ekran TV' lerine girmiş**Commodore 64K**, **Amiga** oyunlarına sabaha kadar vakit ayırdığımız yıllarda; ne **Source Code Control**denen bir kavram vardı, ne de 9 kişilik askeri manga misali çalışan **Scrum** ekipleri. Ancak teknoloji ve yazılım dünyası öylesine hızla ilerledi ki…Koşar adımlarla geldiğimiz günümüzde, özellikle **Enterprise**çapta yürütülen projelerde, ekip olmadan hareket etmek neredeyse imkansız hale geldi.
Yazılıma başladığım yıllarda **Microsoft Visual SourceSafe** kullanan birisi olarak olayın uzun bir süre önce kod kontrolü ve saklanmasının ötesine geçtiğini söylesem sanırım hepimiz bu noktada hem fikir oluruz Artık **ALM*(Application Lifecycle Management)*** olarak ifade edilen ve ürün geliştirmenin sadece koddan oluşmadığını ifade eden önemli bir kavram var hayatımızda. Hatta bu kavramın içerisine giren **MSF*(Microsoft Solution Frameworks)***, **SCRUM**, **CMMI*(Capability Maturity Model Integration)*** gibi pek çok süreçte mevcut. Müşteri ihtiyaçlarının daha hızlı daha çevik bir şekilde karşılanması esasına dayalı bu süreçler çok ama çok popüler.
Tabi işin önemli bir parçasını da **ALM** için kullanılan etkili araçlar üstlenmekte. **Microsoft** bu alanda 2000li yılların ortalarından itibaryen **Visual SourceSafe**' i terk edip **Team Foundation Server**' a geçiş yaptı. **Team Foundation Server** şu an geldiği **2012** sürümü ile, yazılım alanındaki önemli bir ihtiyacı da karşılamakta: **ALM' in dijital ortamda yönetimi, yürütümü ve kontrolü.**
Tabi **Team Foundation Server**' ın çok geniş bir kavram olduğunu ve bir makaleye sığdırılamayacak kadar çok özelliği bulunduğunu gönül rahatlığı ile ifade edebilirim Diğer yandan bugünkü yazımıza konu olan onun küçük görünen ama çok önemli işlerin altına imza eden bir parçası…

Client Object Model

Peki, Microsoft adresinden download edebileceğiniz bu nesne modeli bize ne sunuyor? Olaya aşağıdaki grafik ile başlayalım.



COM – Client Object Model
SOM – Server Object Model
BPOM – Build Process Object Model

Yukarıdaki grafik anlatımında **TFS**' in uygulama modeline ait örnek bir dağılıma yer verilmektedir. Hepimiz esas itibariyle **Team Foundation Server**' ın, n sayıda makine üzerine fiziki olarak dağıtılabilen bir uygulama sunucusu ve çevre programlar bütünü olduğunu biliyoruz. Bu anlamda **TFS**uygulama sunucusuna bağlanan pek çok istemci çeşidi de mevcut. Örneğin **Continous Integration**gibi modelleri destekleyen **Build** planlarının yönetildiği **Build Server** veya bir geliştirici makinesi üzerinde koşan **Visual Studio** gibi.

TFS' i bir dünyadan ziyade bir evren olarak nitelendirmek sanıyorum ki yanlış olmaz. Sadece kurulum sonrasında elimizin altında Application Server, Sharepoint ve SSRS*(Sql Server Reporting Services)*, SSAS*(Sql Server Analysis Services)*, Build Server gibi parçalar oluşmakta. Hatta sanallaştırma da işin içerisinde girdiğinde Lab Management için gerekli ek bir çok ortam türemekte. Tüm bunlara bir de Tool setlerini eklediğinizde Güneş Sisteminin üretmiş olabilirsiniz.

Evren demiştik…Çünkü Güneş Sistemi dışına çıkarak farklı sistemleri de bu çembere dahil edebilirsiniz(*LINUX' dan UNIX'e, MacOS X' den Eclipse' e, Oracle' dan TIBCO' ya…)*

Aslında **TFS** sistemine dahil olabilecek istemcileri düşündüğümüzde çoğumuzun aklına standart bir geliştirici makinesi ve üzerinde yüklü olan **Visual Studio** sürümü gelmektedir. Oysaki **TFS**' i kullanabilen istemcilerin böyle bir zorunluluğu yoktur. **TFS**' in çevre birimler ile olan entegrasyonu adına aşağıdakileri ifade edebiliriz.

- **Visual Studio IDE**' sinin bir parçası olan **Team Explorer** ücretsizdir. Geliştirici olmayan birisi tarafından rahatlıkla kullanılabilir.

- **MS Office** uygulamaları **TFS**' e entegre olabilir, dolayısıyla **Work Item**' lar*(Seçtiğiniz süreç şablonuna göre değişiklik gösterebilir)* örneğin **Excel**' e indirebilir, güncellenebilir*(Hatta Ms Project ile proje planınızı aktarabilirsiniz vs)*
- Takım üyeleri **Web** arayüzünü kullanıp **TFS** ortamına bağlanabilir, **ALM**' e dahil olabilir, **Source Code**' ları görebilir, hatta raporlama hizmetlerine*(SSRS-Sql Server Reporting Services)* ulaşıp güncel duruma bakabilir*(Elbette yetkilendirmelere bağlı olarak neleri görüp neleri göremeyeceklerini nerelere erişip nerelere erişemeyeceklerini belirleyebilirsiniz)*
- **Sharepoint** portalı ile doküman bazlı proje akışları **Team Project**' ler ile ilişkilendirebilir.
- Team Explorer Everywhere sayesinde herhangi bir **Eclipse IDE** ile de **Team Explorer**' ın avantajlarından yararlanılabilir. Dolayısıyla örneğin **Java** ekipleri **TFS**' e entegre olabilir. *****
- Son olarak MSSCCI Provider hizmetinden yararlanıp, yabancı araçlarında*(SQL Navigator, PowerBuilder, Solaris UNIX, LINUX, TIBCO vb)* **TFS**' e entegre olması mümkündür.*****

****** *Son iki madde saha da tecrübe edilmiştir* 

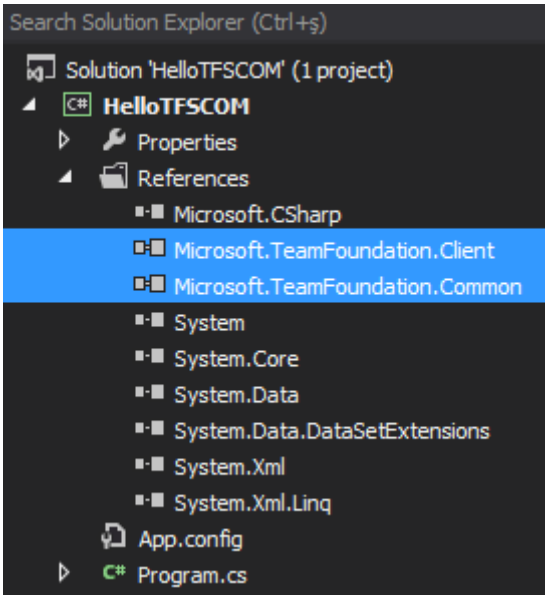
Esas itibariyle istemciler, uygulama sunucusu üzerinde yer alan **TFS Web servislerini** kullanırlar. Bu açıdan bakıldığında söz konusu servisleri tüketen istemci uygulamalar bu işi **Client Object Model**üzerinden icra etmektedirler. Bu şu anlama da geliyor; TFS' i kullanacak kendi istemci uygulamalarımızı geliştirebiliriz.

Online olarak da Cloud tabanlı hizmet veren TFS' in yakında zaman da OData protokolünü baz alan servisleri yayınlandı. Bu konuda Brian Keller' ın şu adresteki yazısını incelemenizi şiddetle tavsiye ederim.

TFS açısından olaya bakıldığında 3 farklı nesne modeli olduğunu ifade edebiliriz. Sunucu tarafı için**Server Object Model**, istemci tarafı için bu yazımızda ele alacağımız **Client Object Model** ve son olarak da **Build Service**' lerin, **COM** ile olan iletişiminde devreye giren **Build Process Object Model**. Biz bu yazımızda sadece **Client Object Model**' in nasıl kullanılabileceğini basit bir Hello World uygulaması ile irdelemeye çalışıyor olacağız.

Hello Client Object Model

Client Object Model' i yüklendikten sonra **.Net** projesine aşağıdaki ekran görüntüsünde yer alan**Microsoft.TeamFoundation.Client** ve **Microsoft.TeamFoundation.Common assembly**referanslarının eklenmesi yeterli olacaktır.



Olur da Reference penceresinde Extensions kısmında çıkmazlar, bu durumda C:\Program Files\Microsoft Visual Studio 11.0\Common7\IDE\ReferenceAssemblies\v2.0\ adresine bir uğrayın derim Şimdi dilerseniz ilk örnek kodlarımızı yazalım.

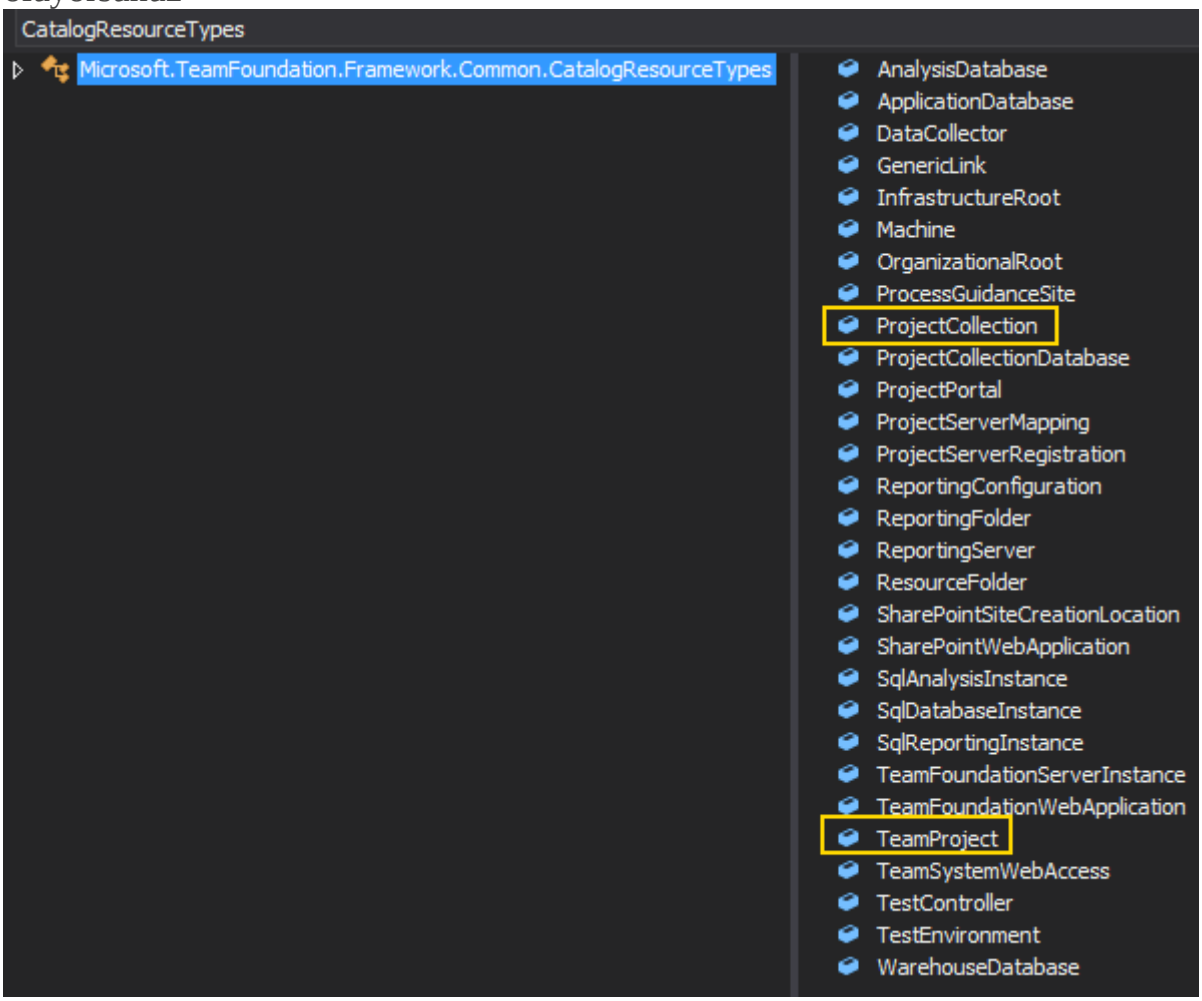
```
using Microsoft.TeamFoundation.Client;
using Microsoft.TeamFoundation.Framework.Client;
using Microsoft.TeamFoundation.Framework.Common;
using System;
using System.Configuration;
using System.Net;
namespace HelloTFSCOM
{
   class Program
   {
      static void Main(string[] args)
      {
         Uri tfsAddress = new Uri(ConfigurationManager.AppSettings["TfsAddress"]);
         TfsConfigurationServer tfsServer =
            TfsConfigurationServerFactory.GetConfigurationServer(tfsAddress);
         tfsServer.Credentials = new NetworkCredential(
            ConfigurationManager.AppSettings["Username"],
            ConfigurationManager.AppSettings["Password"],
            ConfigurationManager.AppSettings["Domain"]
            );
         tfsServer.Connect(ConnectOptions.IncludeServices);
         Console.WriteLine("TFS Server : {0}\n"
            ,tfsServer.Name
            );
```

```
            var teamCollections = tfsServer.CatalogNode.QueryChildren(
               new[] { CatalogResourceTypes.ProjectCollection },
               false, CatalogQueryOptions.None);
            foreach (var teamCollection in teamCollections)
            {
               Guid teamCollectionId = new
Guid(teamCollection.Resource.Properties["InstanceId"]);
               TfsTeamProjectCollection teamProjectCollection =
tfsServer.GetTeamProjectCollection(teamCollectionId);
               Console.WriteLine("Team Project Collection : {0}\n"
                  ,teamProjectCollection.Name
                  );
               var teamProjects = teamCollection.QueryChildren(
                  new[] { CatalogResourceTypes.TeamProject },
                  false, CatalogQueryOptions.None);
               foreach (CatalogNode teamProject in teamProjects)
               {
                  Console.WriteLine("Team Project {0}\n\tDescription {1}"
                     ,teamProject.Resource.DisplayName
                     ,teamProject.Resource.Description
                     );
               }
            }
         }
      }
   }
}
```

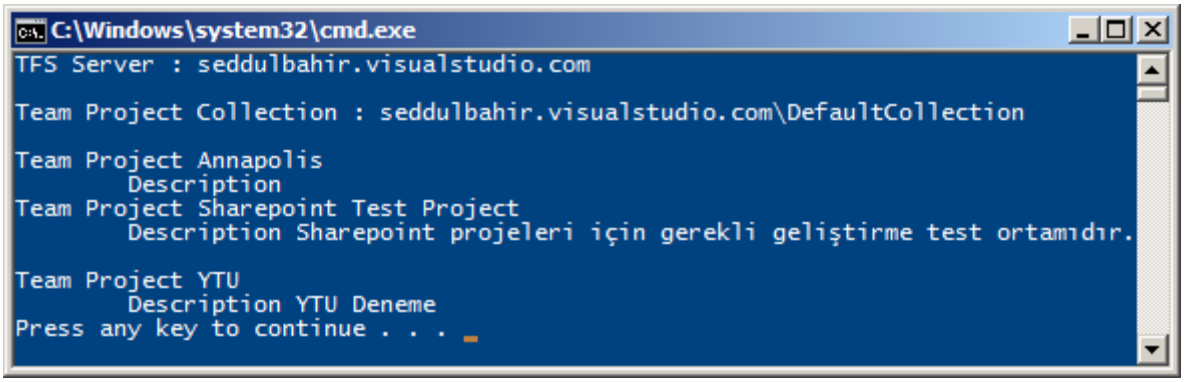
İlk olarak **TfsConfigurationServer** tipine ait bir nesne örneği oluşturuyoruz. Bunun için**TfsConfigurationServerFactory** fabrika sınıfından ve **GetConfigurationServer** metodundan yararlanılmaktadır. Operasyona gelen parametre **TFS** sunucu adresidir. Bu adresi**app.config/web.config** gibi bir dosyadan alabiliriz. Çok doğal olarak **TFS** sunucuları genellikle**domain** kontrolü altında kurulurlar. Bu sebepten **domain** içerisinde yetkisi olan kullanıcıların sunucuya erişebilmesi mümkündür. Bu sebepten şirket ortamlarında kullanıcı adı ve şifre haricinde bir de **domain** bilgisine ihtiyaç vardır. Kod parçasında bu 3 bilgi, **NetworkCredential** sınıfına ait bir nesne örneğinde toplanarak, **Credentials** özelliğine atanmıştır.

Sonraki adımlarda sırasıyla bir bağlantı işlemi ve sonrasında da **TFS** sunucusu üzerinde yer alan**Team Project Collection** içeriklerinin sorgulanması işlemi gerçekleştirilmektedir. Aslında sorgulamaların ortak noktası, dikkat edileceği üzere **QueryChildren** isimli bir metottan yararlanılması ve neyin sorgulanacağını belirtmek için **CatalogResourceTypes enum** sabitinden yararlanılmasıdır. Bu sabitin alacağı

değerleri gördüğünüzde aslında bu seviyede neleri sorgulayabileceğinizi de anlamış oluyorsunuz



Dolayısıyla **TFS** üzerinde oldukça fazla nesneyi sorgulayabiliriz. Örneğimizde sadece**Team Project Collection** ve içerisinde yer alan **Team Project** örnekleri değerlendirilmiştir. Oysaki test ortamından raporlara, Sharepoint Proje portallerinden SQL veri tabanı örneklerine kadar pek çok içerik sorgulanabilmektedir.
Örnek uygulamayı çalıştırdığımızda ben aşağıdaki ekran görüntüsünde yer alan sonuçları aldım.

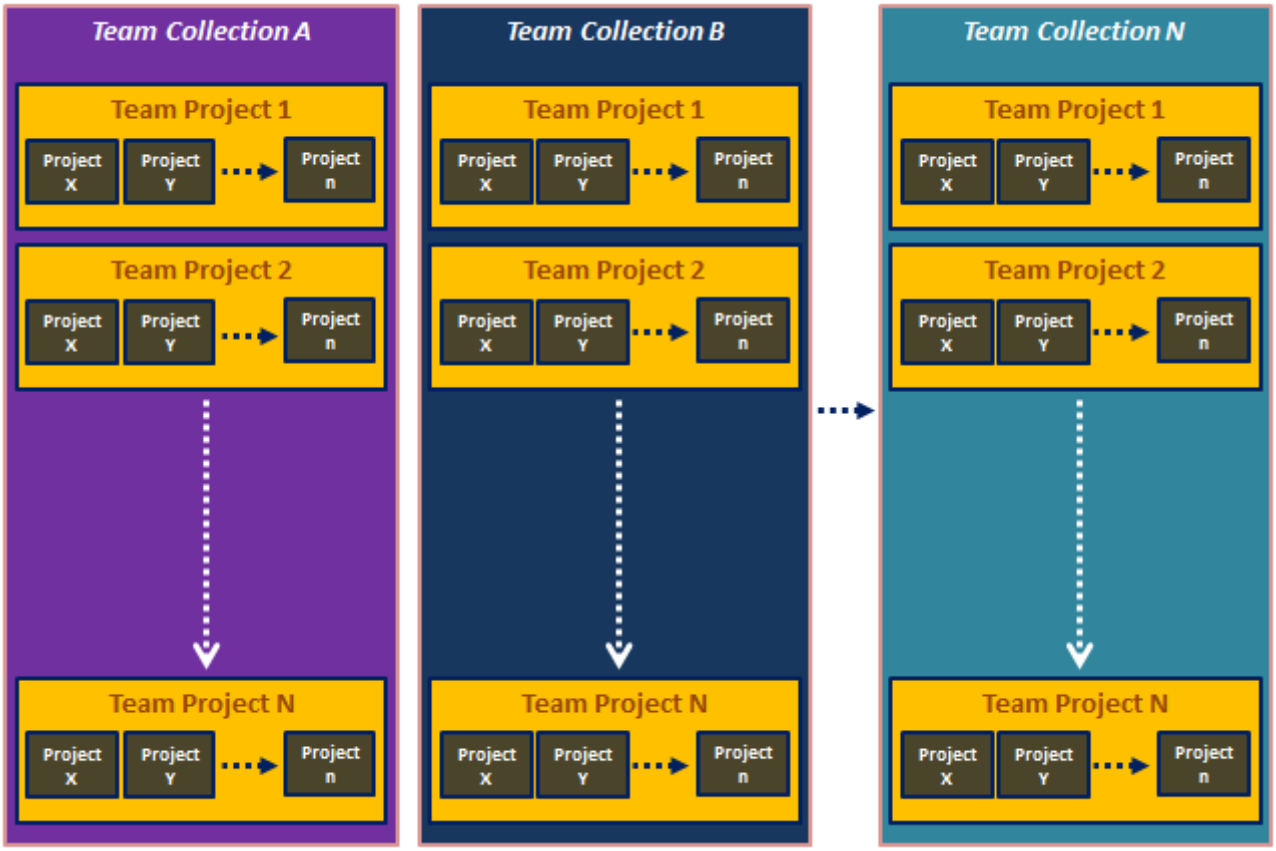


Bu örnek **tfs.visualstudio.com** üzerinde oluşturduğum bir koleksiyona aittir.**seddulbahir.visualstudio.com** adresinden ulaşabildiğim koleksiyon içerisinde bir kaç deneme projesi oluşturdum. Tahmin edeceğiniz üzere **Windows Live ID** ile burada yer

almanız ve eğer bir değişiklik olmadıysa 5 kişiye kadar ücretsiz olarak yararlanmanız mümkün. Yani 5 kişilik bir ekibiniz var ise hemen bir TFS hesabı açıp çalışmaya başlayabilirsiniz Bu arada Windows Live ID ile bağlandığımız bu sistemde Credential için bir Domain bilgisi vermenize gerek yoktur.

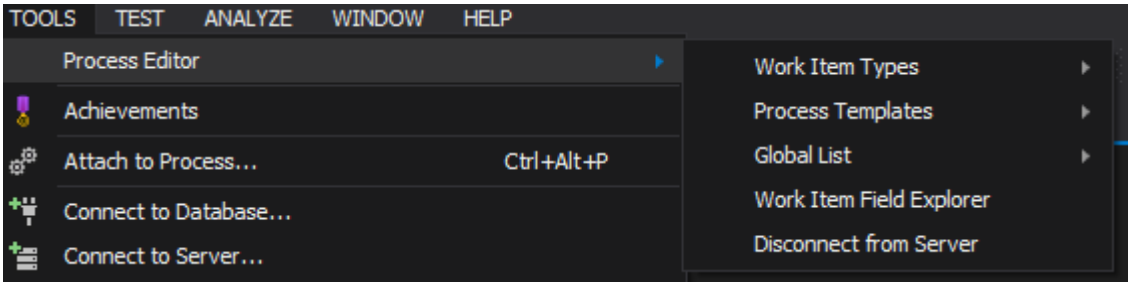
TFS Proje İskeleti

Aslında bu tip bir örneği işletmeden önce **TFS**' in genel olarak proje iskelet yapısını bilmekte de yarar vardır. Aşağıdaki şekilde bu durum kısaca özetlenmeye çalışılmaktadır.



Normal şartlarda **TFS** i kurduğumuzda*(ki install işlemi eğer farm üzerine kurulum yapıyorsanız biraz sıkıntılı olabilir)* hep **Default Collection** üzerinden çalışırız. **Default Collection** aslında **SQL**tarafında da bir veri tabanına karşılık gelmektedir. Oysaki 100lerce projeye sahip olup, bunların çoğunu **Enterprise** seviyede inşa eden firmalarda birden fazla **Team Project Collection**kullanıldığı da görülmektedir. Her **Team Project Collection** aslında bir **Team Project** ailesini işaret etmektedir. Bir başka deyişle bir **Team Project Collection içerisinde n sayıda Team Project** barındırabilirsiniz. Çok doğal olarak her **bir Team Project de kendi içerisinde birden fazla proje** barındırabilir.
Ne yazık ki **Team Project** ile **Project** kavramları zaman zaman birbirlerine karışabilmektedir. Aslında bu ayırım uygulanmak istenen süreç noktasında kendisini daha belirgin gösterir. Nitekim bir**Team Project** oluştururken **Scrum, MSF, CMMI** veya özelleştirilmiş bir **Process Template**seçilmelidir.

Var olan bir Process Template' i indirip, XML içerikleri ile oynayabilir ve şirket kültürünüze uygun farklı bir süreç şablonu oluşturabilirsiniz. Bu anlamda[Microsoft' un Power Tools] ürününü kullanmanızı öneririm. Visual Studio 2012' ye bir eklenti şeklinde gelip şablonları görsel olarak yönetebilmenize olanak tanımaktadır.



Yani **ALM** yoğurt yiyiş şekli belirlenmelidir. Sonrasında ise bu **Team Project** içerisine dahil olan ve aynı şekilde yoğurt yiyecek olan ekip elemanları, n sayıda ve n çeşitte proje üzerinde çalışabilir. Bu projeler .Net uygulamaları olabileceği gibi .Net dışı ortamlar da olabilir. Önemli olan tüm bu projelerin aynı **Team Project** içerisinde dahil olmalarıdır.

WorkItemStore ile Work Item Öğelerini Sorgulamak

Şimdi örneği biraz daha ilerletelim ve diğer **TFS Web Service**' lerinden nasıl yararlanabileceğimize bakalım. Söz gelimi bir **Team Project** için söz konusu **Work Item**' ları çekmeye çalışalım. Bu amaçla örnek kodlarımızı aşağıdaki gibi geliştirebiliriz.

```
using Microsoft.TeamFoundation.Client;
using Microsoft.TeamFoundation.Framework.Client;
using Microsoft.TeamFoundation.Framework.Common;
using Microsoft.TeamFoundation.WorkItemTracking.Client;
using System;
using System.Configuration;
using System.Net;
namespace HelloTFSCOM
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Bir Team Project Collection' daki Task' ların Çekilmesi
         TfsTeamProjectCollection tfsCollection = new TfsTeamProjectCollection(
            new Uri(ConfigurationManager.AppSettings["TfsAddress"] +
"/DefaultCollection"));
         tfsCollection.Credentials = new NetworkCredential(
            ConfigurationManager.AppSettings["Username"],
            ConfigurationManager.AppSettings["Password"]
```

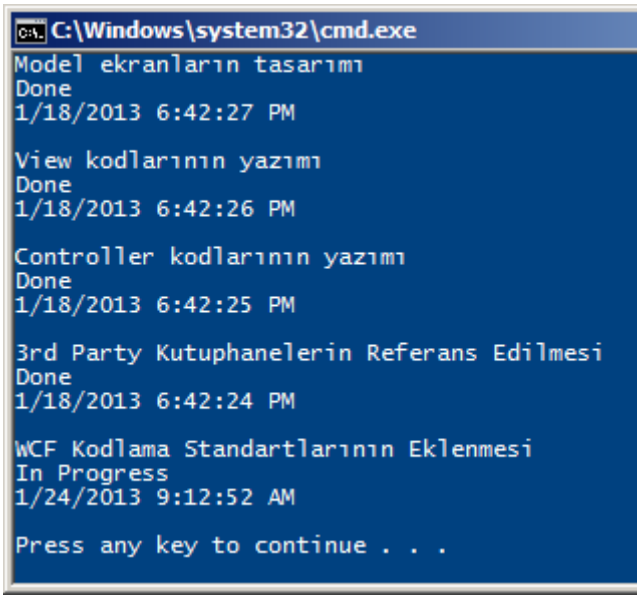
```
            , ConfigurationManager.AppSettings["Domain"]
            );
          WorkItemStore store =
(WorkItemStore)tfsCollection.GetService(typeof(WorkItemStore));
         WorkItemCollection queryResults = store.Query(@"
                    Select [State], [Title]
                    From WorkItems
                    Where [Work Item Type] = 'Task' and [Team Project]='ARGE'
                    Order By [State] Asc, [Changed Date] Desc");
         for (int i = 0; i < queryResults.Count; i++)
         {
            Console.WriteLine("{0}\n{1}\n{2}\n",queryResults[i].Title,
queryResults[i].State, queryResults[i].ChangedDate.ToString());
         }
         #endregion Bir Team Project Collection' daki Task' ların Çekilmesi
      }
   }
}
```

WorkItemStore tipi Microsoft.TeamFoundation.WorkItemTracking.Client.dll assembly' ı içerisinde yer aldığından söz konusu kütüphanenin projeye referans edilmesi gerekmektedir.

Work Item' ların sorgulanabilmesi için **WorkItemStore** servisinden yararlanılabilir. Bu servise ait bir nesne örneğini elde etmek içinse var olan bir **TFS Team Project Collection**' dan faydalanılabilir. Bu sebepten örnek kod parçasında **TfsTeamProjectCollection** tipinden bir örnek üretilmiştir. Adres kısmındaki **/DefaultCollection** ilavesine lütfen dikkat edelim. Bu şekilde**DefaultCollection**' ı işaret eden bir örnek oluşturmuş bulunuyoruz. Sonrasında **tfsCollection**değişkeni üzerinden **GetService** fonksiyonunu çağırmaktayız. Dikkat edileceği üzere fonksiyon parametre olarak **WorkItemStore** tipini almaktadır. Bu durumda **TFS** Sunucusuna şunu söylemiş oluyoruz.

Ey o adresteki DefaultCollection. Bana, tuttuğun Work Item örneklerini sorgulayabilmem için bir servis referansı verrrr!!!

Bu işlemin ardından artık **Work Item**' ların sorgulanmasına başlanıyor. Aslında tipik bir **SQL** sorgusu diyebileceğimiz ama literatürde **WIQL*(WorkItem Query Language)*** olarak geçen ifademizi **Query**metoduna parametre olarak veriyoruz. Sorgumuz son derece basit. **ARGE** isimli **Team Project**içerisinde yer alan **WorkItem**' lardan **Task** tipinden olanları, önce **State**' e göre **A…Z** sırasında, sonrasında da son değişiklik zamanına*(Changed Date)* göre **Z…A** sırasında talep ediyoruz. Sonuç olarak benim deneme amaçlı olarak kullandığım **ARGE** isimli **Team Project** için aşağıdaki ekran görüntüsünde yer alan sonuçları elde ettiğimi ifade edebilirim.



Görüldüğü üzere **ARGE** projesine ait **Task** tipinden **Work Item** öğelerinin başlıkları*(Title)*, güncel durumları*(State)* ve son değişiklik zamanları*(Changed Date)* elde edilebilmiştir. Bu noktada neler yapabileceğinizi ifade etmek istediğimizde sadece **Team Explorer** ile veya **TFS**' in Web arayüzü ile yapabildiklerinizi düşünmeniz kafi olacaktır.*(**WIQL**' in örnek kullanımları ve 5 parçadan oluşan iskelet yapısının teknik detayı için MSDN adresini ziyaret edebilirsiniz)*

WorkItemStore dışında kullanılabilen pek çok servis vardır. Örneğin ISecurityService, ILocationService, IBuildServer, IProcessTemplates, VersionControlServer vb…Söz konusu servislerden bazıları Team Project Collection, bazıları da TFS Server seyiyesinde kullanılabilmektedir. Detaylı bilgiye bu adresten ulaşabilirsiniz.

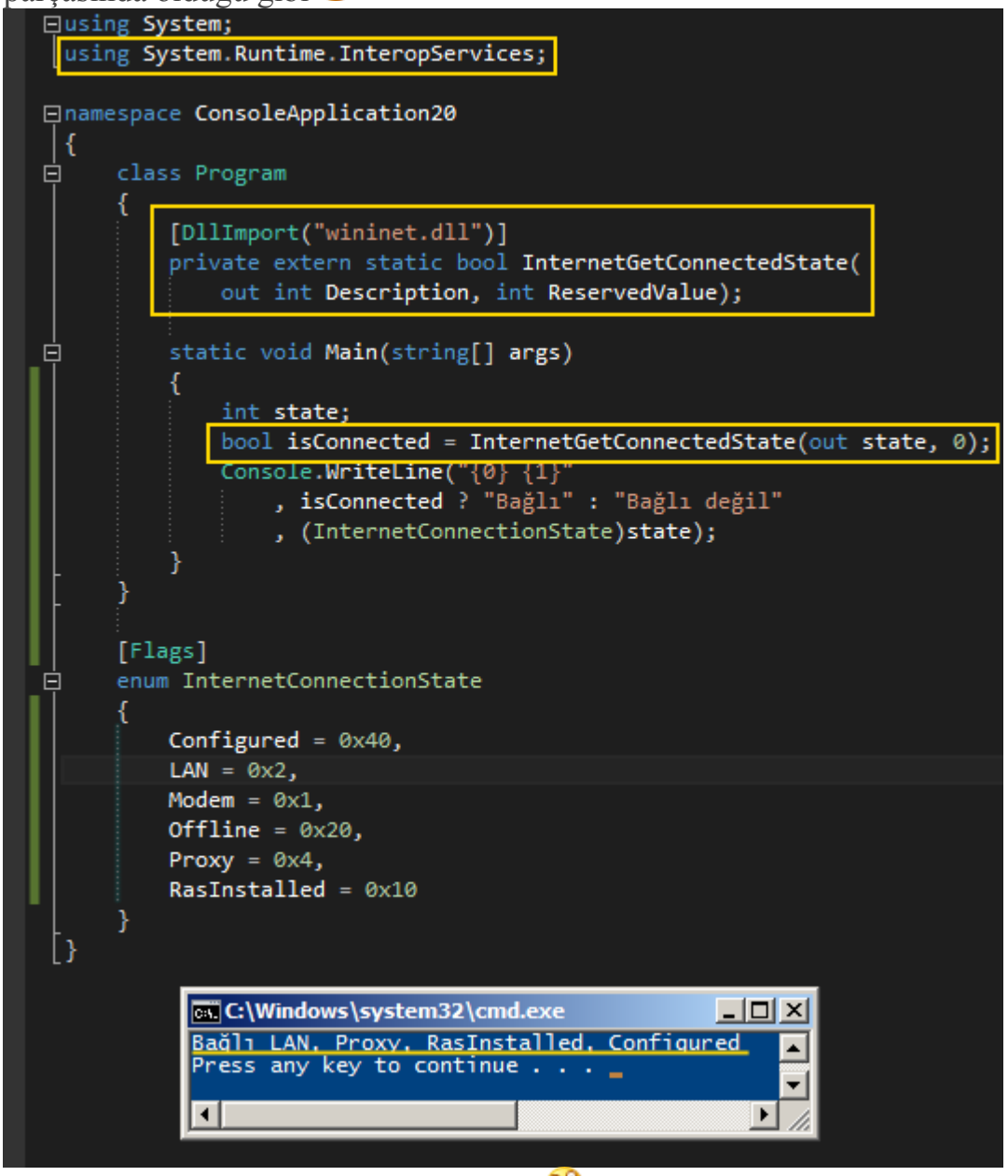
Bu yazımızda **TFS Client Object Model** ile basit de olsa el sıkışmaya çalıştık. Elbetteki örneği devam ettirip ilerletmek sizin elinizde. Söz gelimi iyi bir antrenman olarak **Visual Studio Team Explorer** penceresinin bir benzerini ve hatta **Windows Phone 8** veya **Windows 8** üzerinde çalışacak şekilde zengin kullanıcı deneyimi sunacak olan bir türevini geliştirebilirsiniz. Dokunmatikliği işin içerisine katın Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HelloTFSCOM.zip (181,68 kb)

Tek Fotoluk İpucu 81–Internete Bağlı mıyız?

Pazartesi, 11 Mart 2013 21:38 by bsenyurt

Merhaba Arkadaşlar,
Acaba çok basit ve hızlı bir şekilde internete bağlı olup olmadığımızı nasıl kontrol edebiliriz, hiç düşündünüz mü? Bunun pek çok yolu var aslında. Ancak bir tanesi oldukça hızlı. Basit bir **WinAPI**yardımıyla bu fonksiyonelliği sağlayabilir ve internete bağlı olup olunmadığını kontrol edebiliriz. Aynen aşağıdaki ekran görüntüsünde yer alan kod parçasında olduğu gibi



Bir başka ip ucunda görüşmek dileğiyle

WCF Service' lerinde Routing ile Versiyonlama

Pazartesi, 4 Mart 2013 11:50 by bsenyurt

[Orjinal yazım tarihi 2012 Eylül' dür]

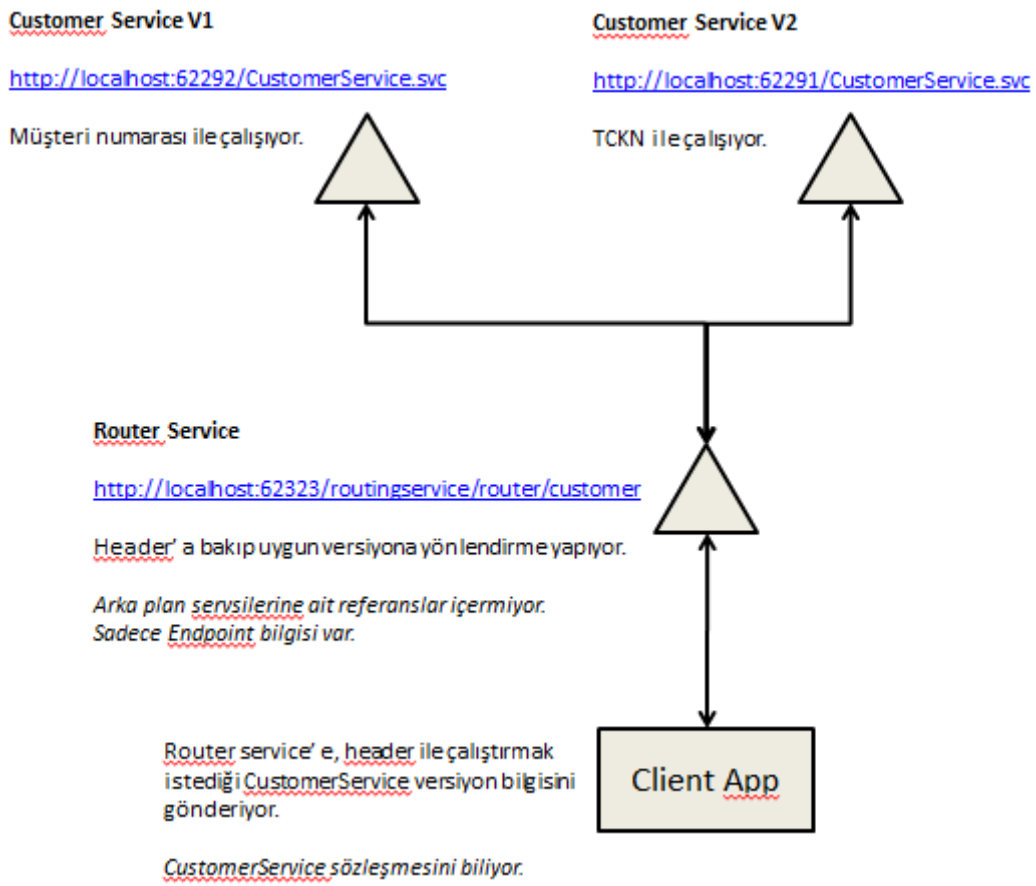
Merhaba Arkadaşlar,
Geçen gün şöyle eskiden yazmış olduğum makalelere bir bakayım dedim. Derken gözüm **WCF 4.0**'ın **Beta**zamanlarında yazdıklarıma takıldı. O zamanlar **.Net Framework 4.0'** ün **Beta** sürümü çıktığında, incelemeye çalıştığım önemli yeniliklerden birisi de yönlendirme servisleri**(Routing Service)** idi. İstemciden gelen talepleri analiz ederek, arka planda yer alan asıl servislere mesajların taşınması noktasında göz önüne alınabilecek önemli bir kabiliyet sunulmaktaydı. Aslında **RoutingService,** buradaki işi güçlü filtreleme özellikleri ile epeyce kolaylaştıran bir tip olarak karşımıza çıkmaktaydı. Hatta bu konuyu bir kaç makale ile de ele almaya çalışmıştım.

WCF 4.0 Yenilikleri - Routing Service Geliştirmek - Giriş [Beta 1]
WCF 4.0 Yenilikleri - Routing Service Geliştirmek - Hello World [Beta 1]
WCF 4.0 Yenilikleri - Routing Service - Hata Yönetimi [Beta 1]
WCF 4.0 Yenilikleri - Routing Service - MatchAll Filtresi [Beta 1]
WCF WebHttp Services - Routing

Tabi bunların hepsi **Beta** sürümüne ait incelemelerdi. Zaman ilerledikçe yönlendirme servislerinin önemli senaryolarda göz önüne alınabileceğini de gördük.
Örneğin **Load Balancing** gibi.
Service yönelimli mimarinin*(Service Oriented Architecture)* uygulandığı **WCF(Windows Communication Foundation)** gibi alt yapılarda karşılaşılan önemli sorunlardan birisi de, aynı servisin birden fazla farklı versiyona sahip sürümlerinin bulunması ve böyle bir durumda istemcilerin istedikleri versiyonu nasıl çağırabilecekleridir. Bu tip bir durum ile pek çok sebepten ötürü karşılaşılabilir. Tabi bir versiyonlamanın söz konusu olması için servis üzerinde bir takım değişikliklerin mevzu bahis olması da gerekmektedir  Bu değişiklikler aşağıda görülen 4 farklı kategori üzerinden ele alınabilirler.

- **Contract(Sözleşme) değişimleri :** Örneğin servise bir operasyon ilave edilmiş veya operasyona gelip giden parametre yapısında değişiklikler(ekleme, çıkartma vb) yapılmış olabilir.
- **Address(Adres) değişimleri :** Servis farklı bir lokasyona taşındığı için yeni bir **endpoint** adresine gereksinim duyulabilir.
- **Binding(Bağlayıcı tip) değişimleri :** Örneğin servis endpoint' i için **WS-I** ile belirtilen bir **security**değişikliği söz konusu olabilir. Bu durumda yeni bir **bağlayıcı tipin(Binding Type)** uygulanması gündeme gelebilir.
- **Implementation değişimleri :** Servis içerisinde yer alan operasyonel bir metodun içeriği değişmiş olabilir.

Tüm bu değişim çeşitleri mümkün olabilir ve bu son derece de olağandır. Ancak uzun vadede bakıldığında, yaşamakta olan bazı sistemlerin yenilenmesi sırasında veya kademeli olarak güncellenmeleri noktalarında, istemcilerin söz konusu değişime uğraşmış servislerin hangi sürümlerini/versiyonlarını kullanması da bir sorun olarak karşımıza çıkmaktadır. Örneğin yenilenecek olan bir ürünün modüllerinin bir kısmı A servisinin 1.0 versiyonu ile çalışmaya devam edecek iken, diğer bir kısmı da yine aynı A servisinin 2.0 versiyonu ile çalışmak durumunda kalabilir. İşte bu noktada versiyonlanan servislerden hangisinin, istemciler tarafından nasıl ele alınabileceğinin bir çözümünün olması gerekmektedir.
Biz bu yazımızda söz konusu versiyonalama problemini **WCF** tarafındaki **Routing** servis aracılığı ile çözümlemeye çalışıyor olacağız. Bildiğiniz üzere **Routing** servisler sayesinde, istemciden gelen taleplerin karşılanarak arka planda duran asıl servislere yönlendirilmesi mümkün olabilir. Bu, **LoadBalancing** gibi çözümlerde ele alınabilecek bir servis çeşidi olmakla birlikte, biraz sonra göreceğimiz gibi versiyonlama probleminde de değerlendirilebilmektedir. Tabi biz örneğimizde aşağıdaki senaryoyu göz önüne alıyor olacağız.
Müşteri bilgisini çekmek için kullanılan bir servisin standart olarak kullandığı operasyonların bazılarının*(ki örneğimizde basitlik açısından tek bir operasyon olacak)* parametrik yapısında değişimler olmuştur. Var olan sürüm müşterinin hesap numarasına göre çalışmaktayken, yeni sürümde söz konusu servis operasyonu TCKN ile çalışıyor olacaktır. Buna göre var olan uygulamalar eski versiyonu kullanmaya devam edecek iken, yeni geliştirilecek olan uygulamalar son sürümü ele alacaktır. Ama ilerleyen zamanlarda eski sürümlerin içerisine müdahalelerde bulunarak hangi versiyonu kullanacaklarına çalışma zamanında karar verilmesi yeteneğine sahip olmaları da gündemdedir. İşte burada servis sürümlerinin uygun olan versiyonlarına çalışma zamanında karar verilmesi yeteneği, Routing Service' lerin önemini biraz daha arttıracaktır.
Görüldüğü üzere servis sözleşmesinde bir değişim söz konusudur. **Routing** servisin buradaki görevi, istemciden gelecek olan versiyon talebine göre arka planda istenen servislere yönlendirme yapmaktır.
İlk sıkıntılardan birisi şudur. Aslında servisin iki farklı versiyonu olmasına rağmen operasyon adları aynıdır. Bu yüzden Routing servisi yönlendirmeye karar vermek için **Action** adlarından yararlanamaz*(Bir başka deyişle **Action Filters** kullanılamaz bu senaryoda)* Bunun yerine, istemciden gelen mesaj içerisinde özel bir **başlık bilgisi(Custom Header)** kullanılabilir ve bu bilgi **Routing**servis tarafında filtrelenebilir. Kafalar karıştı değil mi? Benim de 
Gelin basit adımlar ile ilerleyerek bir **Solution** üzerinden ilgili senaryoyu ele almaya çalışalım. Senaryomuzda aşağıdaki şekilde görülen uygulamalar söz konusudur.



Şekilden de görüleceği üzere versiyonları farklı olan iki servisimiz ve bunlara istemciden gelen talep doğrultusunda yönlendirme yapan bir **Router Service** uygulamamız bulunmaktadır.

Senaryomuzda işlemleri olabildiğince basite indirgedik ve tüm servis noktalarında BasicHttpBinding bağlayıcı tipinden yararlandık. Çok doğal olarak arka planda yer alan servislerin sayısı artabilir ve her biri farklı Binding tipleri ile de bezenmiş olabilir ki bu durumda Binding bazlı bir versiyonlama farkı da oluşacaktır.

Söz gelimi servislerden birisin In-Proc modda erişilebilecek şekilde tasarlanmışken, diğer biri WS Federation 2007 standartlarında kullanılabilecek şekilde geliştirilmiş olabilir.

CustomerServiceV1 ve **CustomerServiceV2** temel olarak iki farklı **WCF** servis uygulaması içerisinde yer almaktadır*(Bu tabiki zorunlu değildir. Aynı WCF Service uygulaması içerisinde de birden fazla svc bulunabilir ve bu şekilde bir versiyonlama yaptırılabilir)* Senaryoda bu servisler farklı **WCFService Application** projeleri içerisine serpiştirilmişlerdir. Her ikisi de aynı servis sözleşmesini*(ICustomerService)* dışarıya sunmaktadır ve **GetCustomer** isimli bir operasyon içermektedirler. Lakin ilk versiyon **Müşteri numarası** ile çalışacak şekilde tasarlanmışken, ikinci versiyon **TC Kimlik Numarasını** kullanmaktadır. Söz konusu uygulamaların kod ve konfigurasyon içerikleri*(web.config)* aşağıdaki gibidir.

CustomerServiceV1 için ;

using System.ServiceModel;

```
namespace CustomerServiceV1
{
   [ServiceContract]
   public interface ICustomerService
   {
      [OperationContract]
      string GetCustomer(string customerNumber);
   }
}

namespace CustomerServiceV1
{
   public class CustomerService
      : ICustomerService
   {
      public string GetCustomer(string customerNumber)
      {
         return "Müşteri numarası ile müşteri bilgisi talep edildi";
      }
   }
}
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <behaviors>
         <serviceBehaviors>
            <behavior name="">
               <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true" />
               <serviceDebug includeExceptionDetailInFaults="false" />
            </behavior>
         </serviceBehaviors>
      </behaviors>
      <serviceHostingEnvironment aspNetCompatibilityEnabled="true"
         multipleSiteBindingsEnabled="true" />
   </system.serviceModel>
</configuration>
```

CustomerServiceV2 için;

```
using System.ServiceModel;
namespace CustomerServiceV2
{
   [ServiceContract]
```

```
  public interface ICustomerService
   {
      [OperationContract]
      string GetCustomer(string tckn);
   }
}

namespace CustomerServiceV2
{
   public class CustomerService
      : ICustomerService
   {
      public string GetCustomer(string tckn)
      {
         return "TCKN ile müşteri bilgisi çekilir";
      }
   }
}
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <behaviors>
         <serviceBehaviors>
            <behavior name="">
               <serviceMetadata httpGetEnabled="true" httpsGetEnabled="true" />
               <serviceDebug includeExceptionDetailInFaults="false" />
            </behavior>
         </serviceBehaviors>
      </behaviors>
      <serviceHostingEnvironment aspNetCompatibilityEnabled="true"
         multipleSiteBindingsEnabled="true" />
   </system.serviceModel>
</configuration>
```

Buraya kadar ki kısımda görüldüğü üzere standart **WCF** servislerinin geliştirilmesi söz konusudur. Asıl mühim olan nokta ise **Routing Service** uygulamasının içeriğidir. Bu servis uygulamasını basit bir **Console Application** şeklinde tasarlıyor olacağız. Kendisi bir **Service Host** olarak görev yapıyor olacak. Dolayısıyla açık kaldığı sürece istemcilere hizmet verebilecek. En önemli nokta konfigurasyon içeriğidir. Örnek senaryomuzda **app.config** içeriği aşağıdaki gibidir.

Console uygulamasına, System.ServiceModel.Routing ve System.ServiceModel assembly' larının referans edilmesi gerektiğini hatırlatmak isterim.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service behaviorConfiguration="rtngCnfg"
               name="System.ServiceModel.Routing.RoutingService">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:62323/routingservice/router/" />
          </baseAddresses>
        </host>
        <endpoint address="customer"
                  binding="basicHttpBinding"
                  name="routerEndpoint"
                  contract="System.ServiceModel.Routing.IRequestReplyRouter" />
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="rtngCnfg">
          <routing filterTableName="versioningFilterTable" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <client>
      <endpoint name="customerServiceV1"
                address="http://localhost:62292/CustomerService.svc"
                binding="basicHttpBinding"
                contract="*" />
      <endpoint name="customerServiceV2"
                address="http://localhost:62291/CustomerService.svc"
                binding="basicHttpBinding"
                contract="*" />
    </client>
    <routing>
      <namespaceTable>
        <add prefix="customer" namespace="http://core.services.customer/"/>
      </namespaceTable>
      <filters>
        <filter name="XPathFilterForVersion1" filterType="XPath"
                filterData="sm:header()/customer:SrvVersion = '1'"/>
```

```
      <filter name="XPathFilterForVersion2" filterType="XPath"
           filterData="sm:header()/customer:SrvVersion = '2'"/>
    </filters>
    <filterTables>
     <filterTable name="versioningFilterTable">
      <add filterName="XPathFilterForVersion1"
endpointName="customerServiceV1"/>
       <add filterName="XPathFilterForVersion2"
endpointName="customerServiceV2"/>
     </filterTable>
    </filterTables>
  </routing>
 </system.serviceModel>
</configuration>
```

Pek tabi bu uygulama bir **Routing** servis olduğundan, arka planda kullanacağı servisler için de bir**Client** gibi düşünülmelidir. Bu sebepten **client** elementi içerisinde, **CustomerServiceV1** ve**CustomerServiceV2** için gerekli **End Point** tanımlamaları bulunmaktadır.

Router servis uygulaması herhangibir şekilde CustomerService' lerin servis referanslarını içermemektedir.No Add Service Reference yani 😊

Servisin kendisi **System.ServiceModel.Routing.IRequestReplyRouter** sözleşme tipini kullanmaktadır. **Config** dosyası içerisindeki en önemli parça ise **routing** elementinin içeriği ve tanımlanan filtreleme opsiyonlarıdır.

XPath tekniğine göre, gelen mesajın **Header** kısmı analiz edilmekte ve **SrvVersion** kelimesinin karşılığı olan değere bakılmaktadır. Bu **değerin 1 olması halinde CustomerService' in 1nci versiyonu, 2 olması halinde ise 2nci versiyonu** için bir yönlendirme yapılacağı planlanmıştır. Filtreleme değeri ile, arka plandaki servis uç noktasının eşleştirilmesi ise **filterTables** elementi içerisinde gerçekleştirilmektedir. İstemci uygulamanın **Main** metodunun içeriği ise son derece basit ve sadedir.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Routing;
namespace RouterServer
{
  class Program
  {
    static void Main(string[] args)
    {
      using (ServiceHost serviceHost =new ServiceHost(typeof(RoutingService)))
      {
        serviceHost.Open();
```

```
            Console.WriteLine("Yönlendirme servisi etkin. Kapatmak için bir tuşa
basınız.");
            Console.ReadLine();
        }
    }
  }
}
```

Dikkat edileceği üzere, **RoutingService** tipinden bir **ServiceHost** örneği oluşturulmaktadır. Bu,**.Net Framework**' ün **built-in routing** alt yapısının yüklenmesi için yeterlidir.
Gelelim istemci tarafına

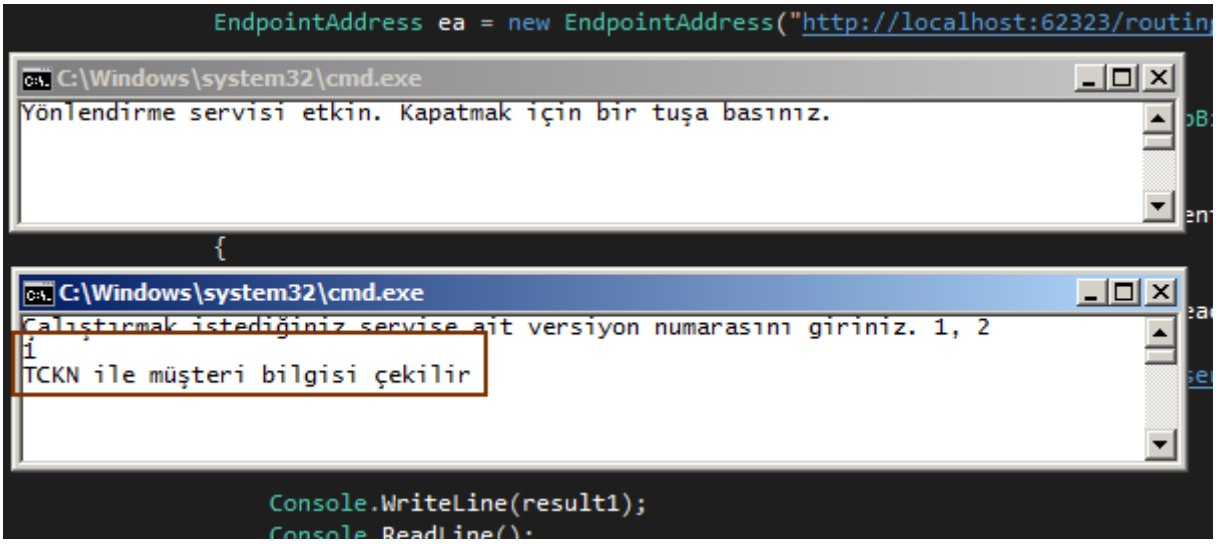
İstemci, müşteri servislerinden hangi versiyonu kullanırsa kullansın, bir şekilde sözleşmeden haberdar olmalı ve **GetCustomer** metoduna erişebilmelidir. Nitekim bu operasyonun çağırılabilmesi için bir de **Proxy** tipine ihtiyacı vardır.
Burada **CustomerServiceV1** veya **CustomerServiceV2**isimli servislerin herhangibigirisinden yararlanılıp bir servis referansının istemci uygulamaya eklenmesi mümkündür. Şahsen ben bu şekilde bir yolu tercih ettim

Unutmayın! İstemcinin bilmesi gereken sözleşme Router servise ait değildir. Zaten Router servis için Metadata Publishing opsiyonu da kapalıdır ve hatta ortada bir Contract' da yoktur.
İstemci, GetCustomer operasyonunu kullanacağı CustomerService sözleşmesine gereksinim duymaktadır. Sadece uygun versiyonu çağırmak için Router servisden yararlanacak ama aslında CustomerService' ine ait GetCustomer metodunu çağırıyor olacaktır.
İstemci tarafına **Add Service Reference** ile ilgili sözleşme eklendikten sonra*(ki **svcutil** komut satırı aracılığı ile de ortak bir isim adı altında üretim yaptırılıp eklenmesi tercih edilebilirdi)* **Main**metodu içerisine aşağıdaki örnek kodları eklememiz yeterli olacaktır.

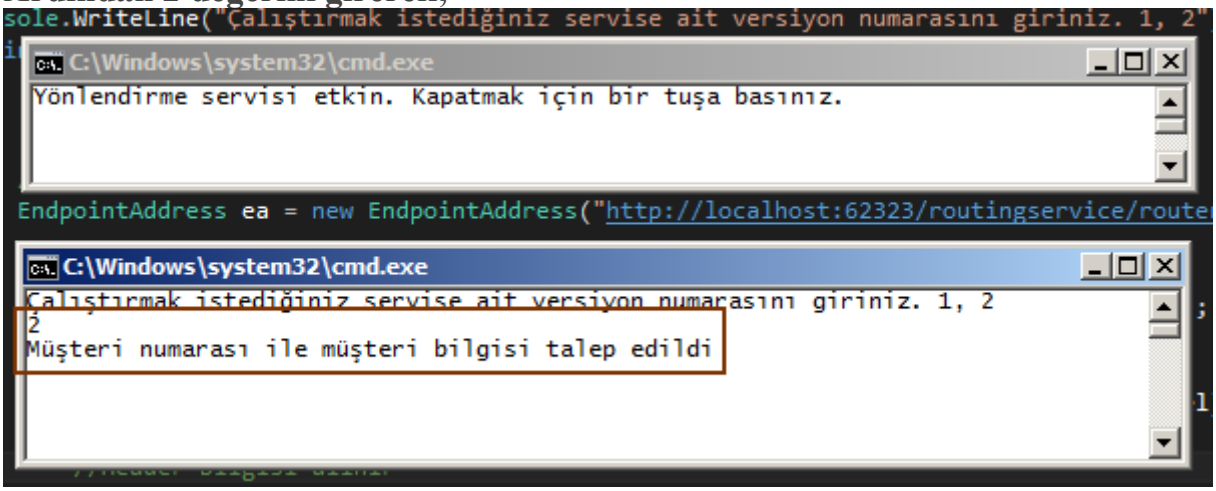
```
using ClientApp.CustomerServiceReference;
using System;
using System.ServiceModel;
using System.ServiceModel.Channels;
namespace ClientApp
{
  class Program
  {
    static void Main(string[] args)
    {
      Console.WriteLine("Çalıştırmak istediğiniz servise ait versiyon numarasını giriniz.
1, 2");
      string versionNumber=Console.ReadLine();
```

```
        try
        {
            //Routing servis için bir Endpoint bildirimi yapılır
            EndpointAddress ea = new
EndpointAddress("http://localhost:62323/routingservice/router/customer");
            // Proxy nesnesi örneklenir
            CustomerServiceClient client = new CustomerServiceClient(new
BasicHttpBinding(), ea);
            // Scope örneklenir
            using (OperationContextScope context = new
OperationContextScope((client.InnerChannel)))
            {
                //Header bilgisi alınır
                MessageHeaders header =
OperationContext.Current.OutgoingMessageHeaders;
                //Header bilgisine kullanılmak istenen versiyon numarası verilir
                header.Add(MessageHeader.CreateHeader("SrvVersion",
"http://core.services.customer/", versionNumber));
                // Operasyon çalıştırılır
                string result1 = client.GetCustomer("1122");
                Console.WriteLine(result1);
                Console.ReadLine();
            }
        }
        catch (Exception excp)
        {
            Console.WriteLine(excp.Message);
        }
    }
  }
}
```

İstemci uygulamada dikkat edileceği üzere kod üzerinden bir **EndPoint** tanımlaması ile işe başlanmakta ve bu uç noktayı **BasicHttpBinding** ile kullanan bir **proxy**örneği**(CustomerServiceClient)** üretilmektedir.
Kodun önemli olan kısmı **MessageHeader** içerisine istenen versiyon bilgisinin eklenmesidir. Bu, o andaki operasyonel **Context'** in elde edilmesinin ardından **MessageHeaders** tipi kullanılarak gerçekleştirilmektedir.
Artık senaryomuzu test edebiliriz. İşte benim uygulamaları test ederken aldığım sonuçlara ait ekran çıktıları.
İlk olarak 1 değerini girerek bir çağrıda bulunuyoruz;



Ardından 2 değerini girerek;



Pek tabi geçerli olmayan bir değerin girilmesi halinde istemci tarafında **Exception** alınması da son derece doğaldır

Görüldüğü gibi bir servisin farklı versiyonlarını host ettiğimiz senaryolarda, istemcilerin istedikleri servislere gitmelerini sağlamak için araya konuşlandıracağımız bir **Routing** servis uygulamasından kolayca yararlanabiliriz. Senaryoda teknik olarak istemcinin **Message Header** içerisine koyduğu versiyon bilgisinin, yönlendirme servisi üzerinde **XPath** ile sorgulanması ile yakalanmaya çalışılması değerlendirilmiştir.

Örnek senaryo istenirse daha da zorlaştırılabilir. Söz gelimi servislerin sözleşmelerinde yapılacak kritik değişiklikler sonucu istemcinin bilmesi gereken contract bilgisinin de güncellenmesi şarttır. Bu konuyu bir düşünmenizi ve böyle bir senaryo oluşması halinde istemcinin istediği versiyonlara nasıl gidebileceğini araştırmanızı ve bir çözüm yolu bulmaya çalışmanızı öneririm. Bu iyi bir ev ödevi gibi geldi bana Versiyonlama her zaman için zor ve karmaşık bir konudur. Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

RoutingAndVersioning.zip (117,79 kb)

Heryerden TFS Kullanabilmek

Cumartesi, 2 Mart 2013 21:50 by bsenyurt

Merhaba Arkadaşlar,
Yandaki fotoğrafta bir duvar prizi içinden USB bağlantısı yapıldığını ve cep telefonunun şarj edildiğini görmektesiniz. Bir süredir hayatımızda olan ilginç buluşlardan birisi de **USB Priz**' ler. Bu aslında USB' nin pek çok farklı ortama entegre edilebilmesi anlamına da geliyor. Söz gelimi bir süredir pek çok araç modelin USB çıkışları neredeyse standart. Telefonlarımızı yol boyunca şarj edebiliyoruz. Hatta USB olmayan araçlarda, çakmaktan gelen enerjiyi **USB** bağlantısı ile aktaran ara dönüştürücüler bile var.
Dolayısıyla çeşitli ve pek çoğu standart hale gelen cihazlar ile USB çıkışları verebilmek mümkün. Nerden geldik şimdi bu USB Priz konusuna. Hem bir Plug-In gibi görülebildiği hem de entegrasyon anlamında sağladığı yetenekleri göz önüna alalım.
Bazen kullandığımız yazılım ürünlerinin de bu tip kolay takılabilir ve entegre olabilir şekilde üretilmelerini bekleriz. Örneğin **TFS**' in sadece **Visual Studio, MS Office, Sharepoint** vb ürünler ile değil başka başka ürünler ile de çalışmasını isteriz.
Microsoft' un **ALM*(Application Lifecycle Management)*** tarafındaki en önemli aracı bilindiği üzere **Team Foundation Server** ürünüdür. Genellikle **Microsoft**' un yazılım geliştirme ürünleri ile haşırneşir olan firmalar **TFS**' i ve uygun bir süreç geliştirme metodolojisini seçerek yaşamlarına devam ederler. Bu tip firmalar için karşılaşılabilecek sorunlar daha çok **ALM**' in layıkıyla uygulanamayışıdır ki bu aslında hepimizin en büyük sorunudur ve **tool**' dan bağımsız bir konudur. Yine de **TFS** kullanımı ile ilişkili olarak çok daha büyük bir sıkıntı vardır. Entegrasyon
Özellikle **Enterprise** çözümler üretmeye çalışan, bünyesinde 100lerce proje barındırabilen, zaman zaman hantallaşan firmaların, duruma göre tercih ettikleri pek çok geliştirme aracı/ortamı söz konusudur. Bazı firmalarda, **Oracle**' cıları, **Java**' cıları, **Linux** üzerinde **C** kodları yazanları, yeni nesil uygulamaları **.Net** ile geliştirenleri sıklıkla görebilirsiniz.
Hatta daha başka 3ncü parti araçlar bile söz konusu olabilir. Bu yerlerde pek tabi irili ufaklı sayısız ekip de söz konusudur. Bu ekipler, kendi içlerinde olduğu gibi şirket bazında da bir uygulama yönetim sürecine dahil olmak durumunda kalırlar. Kimi zaman **CMMI** gibi sıkıcı süreçler, kimi zaman da **Scrum** gibi eğlenceli süreçler, sistemin bir parçasıdır. İşte böyle bir senaryoda firmanın topyekün bir karar alarak **TFS**' e geçeceğini hayal ediniz*(ve tabi gelen tepkileri, direnci de haya ediniz)*. Aslında hayal etmenize gerek yok. Yapanlar var Bu durumda entegrasyon son derece önemli bir hale gelmektedir.
İşte bu yazımızda **TFS**' in bize yabancı gelebilecek bazı geliştirme ortamları ile olan entegrasyonunu incelemeye çalışıyor olacağız.

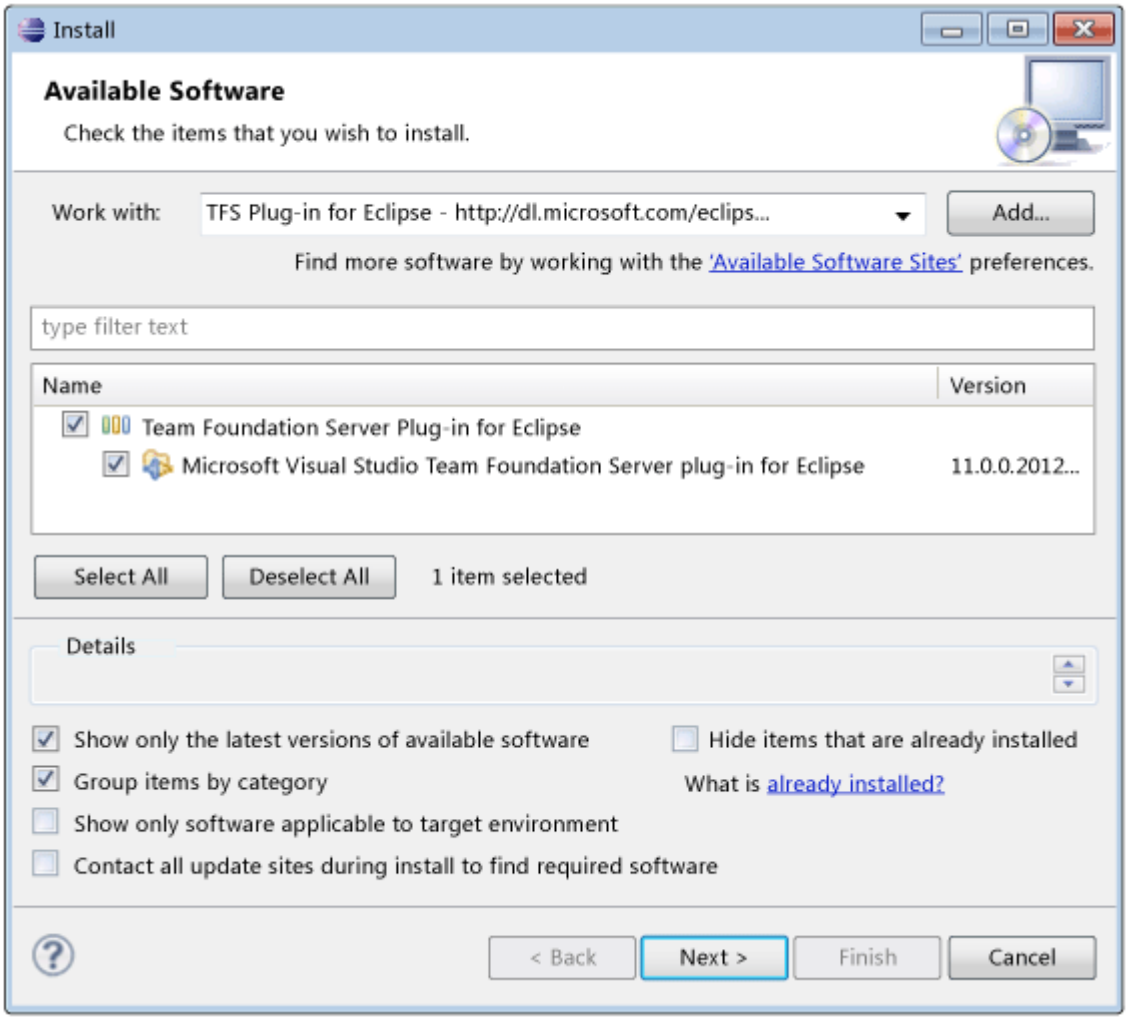
Microsoft, **TFS**' in çevre ürünler*(Visual Studio harici diyebiliriz)* ile olan entegrasyonunda için 3 önemli çözüm sunmaktadır. Temel prensip **Team Explorer**' a ait arabirimlerin bir şekilde diğer araçlara takılabilmesidir*(Plug-In, Add-In vb olacak şekilde)* Yazının bundan sonraki kısımlarında ilgili araçları ve özellikle çeşitli arabirimler ile olan entegrasyonlarını mercek altına alıyor olacağız. Sloganımsı ada sahip olan ürün ile işe başlayalım.

Team Explorer Everywhere

Özellikle **Eclipse** gibi **IDE**' lerin, **Team Explorer** arabirimine sahip olması ve **TFS** ile entegre çalışabilmesi için kullanılmaktadır. Bu adresten indirilebilen ürünün ayrıca diğer platformlar için komut satırından çalışabilen bir versiyonu da bulunmaktadır. Peki, örneğin **Eclipse Juno** ile bu entegrasyonu nasıl gerçekleştirebiliriz? Gelin adım adım ilerleyelim.
Bazı geliştirme ortamları, Eclipse IDE' sini kabuk olarak kullanır. Örneğin Business Process Management araçlarından birisi olan TIBCO Business Studio örnek olarak verilebilir. Bu tip araçlar da Eclipse IDE' sini baz aldıklarından Team Explorer Everywhere ile TFS' e ve doğal olarak ALM süreçlerine dahil olabilirler. En azından TIBCO tarafı için bunu test ettiğimi rahatlıkla ifade edebilirim

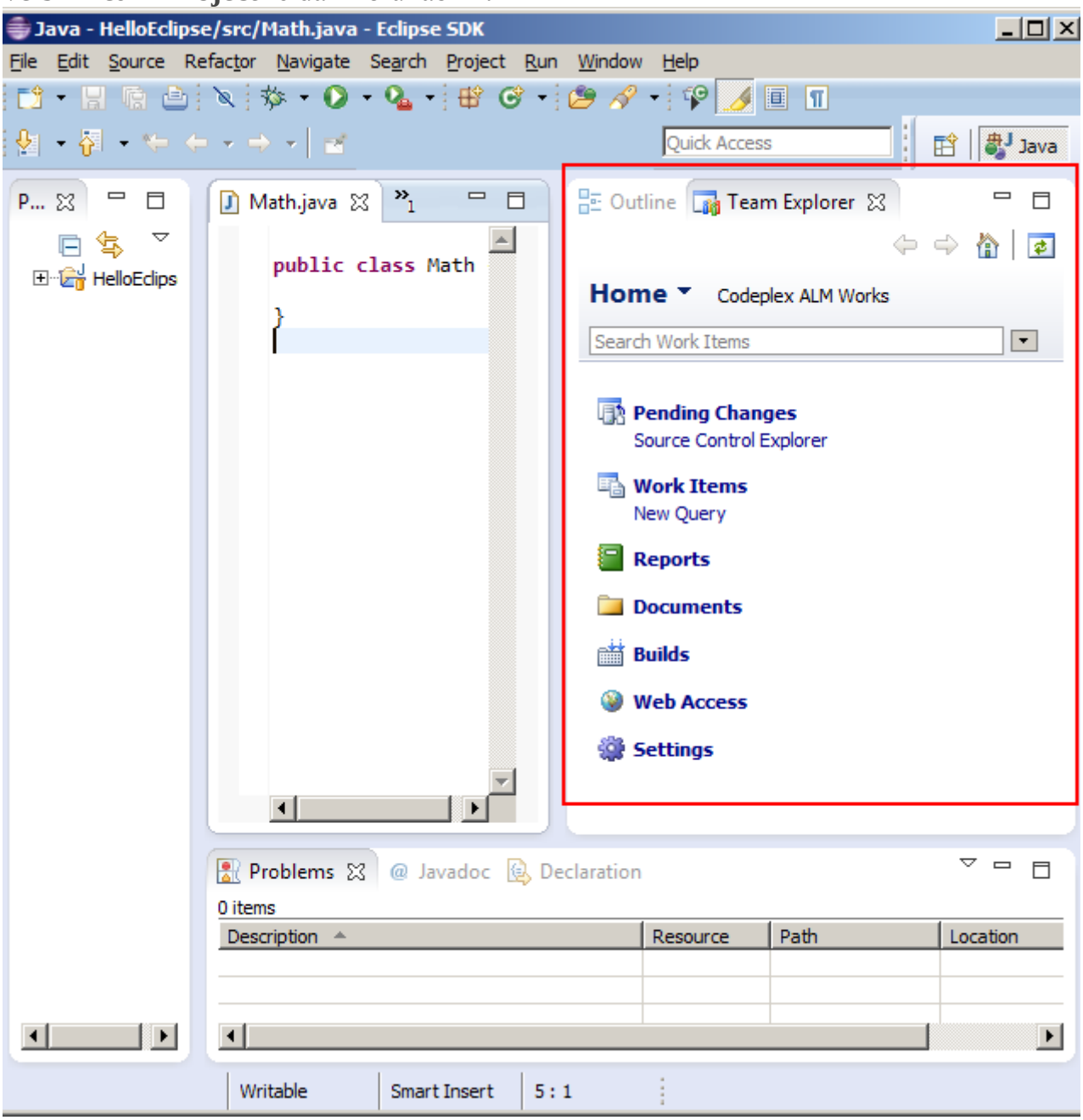
Eclipse Juno

İlk olarak **Eclipse Juno IDE**' si üzerinden **Help** menüsüne girip buradan **Install New Software**seçeneğini işaretlemeliyiz. Gelen arabirim de **Work With** kutucuğunahttp://dl.microsoft.com/eclipse/tfs/ adresine gitmemiz yeterlidir.

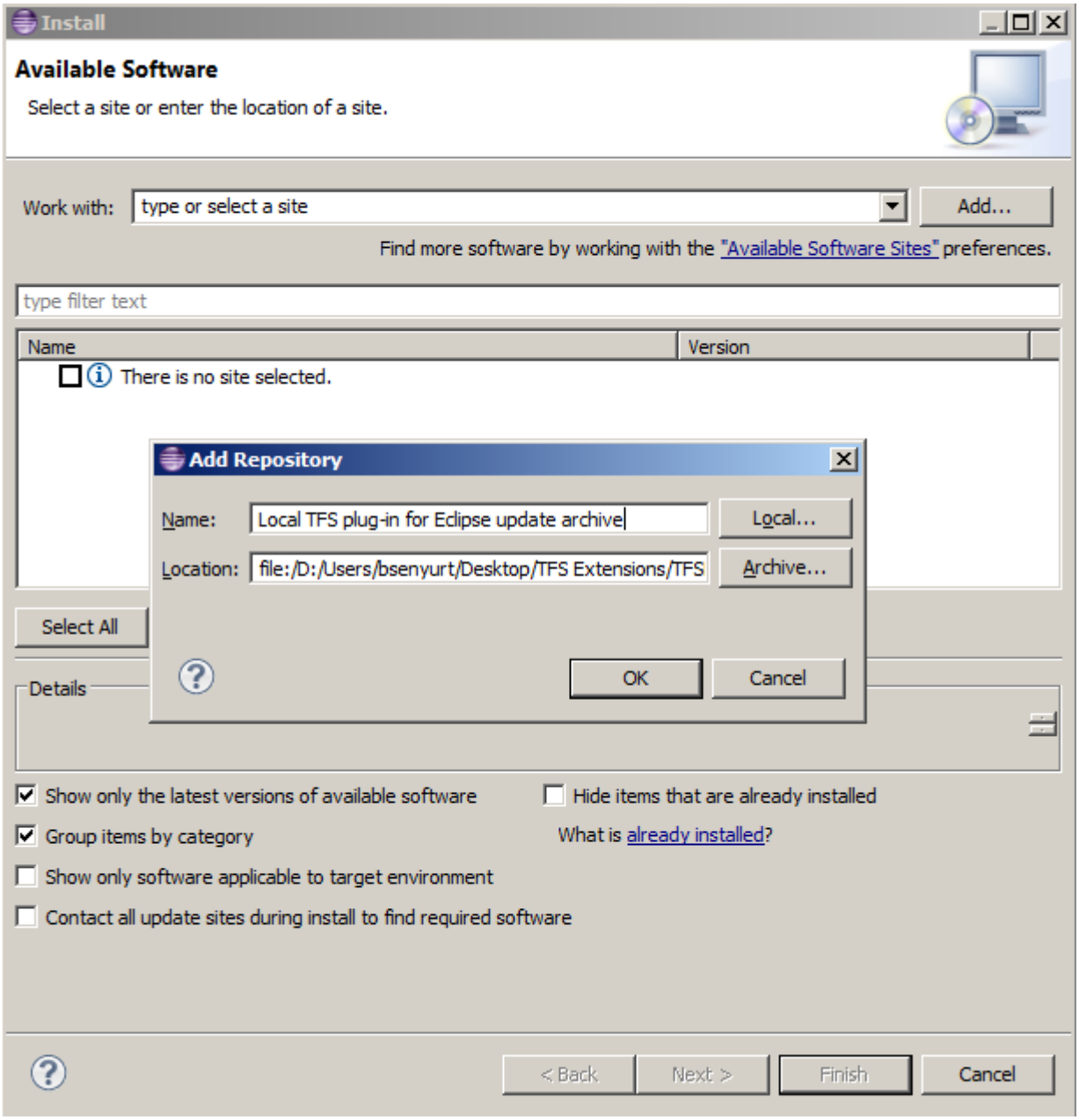


Microsoft sunucularına gidilecek ve **Team Explorer**' ın kullanmakta olduğumuz **Eclipse IDE**' sine uygun olan versiyonu çekilecektir. **Team Foundation Server Plug-in for Eclipse** seçimi yapıldıktan sonra **Next** ile ilerlenip kurulum işlemi tamamlanır.
Sonuç olarak **Eclipse IDE**' sinde **Visual Studio**' dan aşina olduğumuz **Team Explorer** penceresi oluşacaktır. Buradan çok doğal olarak bir **TFS** sunucuna bağlanılabilir

ve bir **Team Project**' e dahil olunabilir.



Eclipse kurulumunda yaşanabilecek sıkıntılardan biriside internet bağlantısı olmayan bir ortamda bu işin nasıl gerçekleştirilebileceğidir. Bu durumda bir şekilde yine **download** sayfasından**TFSEclipsePlugin-UpdateSiteArchive-11.0.0.1212.zip** dosyasının indirilmesi ve bu kez**Install->Add** kısmında aşağıdaki gibi ilgili dosyanın seçilerek eklenmesi yeterli olacaktır.



Eclipse IDE' leri arasında Install adımları için farklılıklar olabilir. Test ettiğim IDE' lerden birisinde Install Software penceresinde iki farklı tab bulunmaktaydı. Önce Available Software kısmından ilgili Team Explorer Everywhere ürününü bulmak*(local file veya url ile)* sonrasında Installed Software tabına geçerek ilgili versiyonu seçtikten sonra Install işlemini uygulamak mecburiyetinde kalmıştım. Son test ettiğim güncel Eclipse Juno ürününde ise bu işlem tek adıma indirilmişti.

Peki ya **Team Explorer IDE**' sine sahip olamayacak farklı geliştirme ortamları söz konusu olursa ne yapacağız?

MSSCCI Provider

Özellike **Team Explorer** desteği bulunmayan*(ya da entegre edilemeyen)* **IDE** ve ortamlar için geliştirilmiş olan bu sağlayıcı, esasında bir **Source Code Control API**' si olarak

düşünülebilir. Bu adresten indirebileceğiniz **Provider**' ın benim incelediğim sürümünün güncel olarak desteklediği**IDE**' ler aşağıdaki gibidiydi.

- Visual Studio .NET 2003
- Visual C++ 6 SP6
- Visual Visual Basic 6 SP6
- Visual FoxPro 9 SP2
- Microsoft Access 2007
- SQL Server Management Studio
- Enterprise Architect 7.5
- PowerBuilder 11.5
- Microsoft eMbedded VC++ 4.0

Hımmm güzel bir listeye benziyor. Sanırım ilk dikkati çekenlerden birisi de **PowerBuilder 11.5**

Peki ya elinizde çok daha eski bir sürüm var ise. Ya bu sürüm üzerinde geliştirilmiş onlarca uygulama bulunmaktaysa ve daha uzun bir süre hayatta olacaklar ise. Ya **.Net** tabanlı geliştirme yapanların bu ortamda geliştirilen arabirimleri de kullanması söz konusu ise. Ya her iki ortamda **ALM** üzerinden takip edilmek zorunda ise. Örneğin elinizde **PowerBuilder**' ın **9.0.3** sürümü olduğunu düşünün.

MSSCCI Provider' ın yüklenmesi sırasında makinede en azından Visual Studio Team Explorer' ın yüklü olması gerekmektedir. Ayrıca ürünün versiyonu da önemlidir. Örneğin, TFS 2012 ile entegre olunacak ise MSSCCI Provider' ın 2012 sürümü kullanılmalıdır.

Bir diğer ön gereklilik de şudur. Diyelim ki developer makinesinde Visual Studio 2010 da bulunmakta ve yine aynı makine de yer alan SQL Navigator ile TFS 2012' ye bağlanılmak isteniyor. MSSCCI Provider' ın 2012 sürümü yüklenirken uyarı mesajı alınacak ve Team Explorer' ın yüklenmesi istenecektir. Dolayısıyla 2012 tabanlı bir TFS ortamında, MSSCCI Provider 2012 ve Team Explorer 2012 olmalıdır.

PowerBuilder

İlk olarak **MSSCCI Provider'** ın **PowerBuilder'** ın bulunduğu ortama yüklenmesi gerekmektedir.

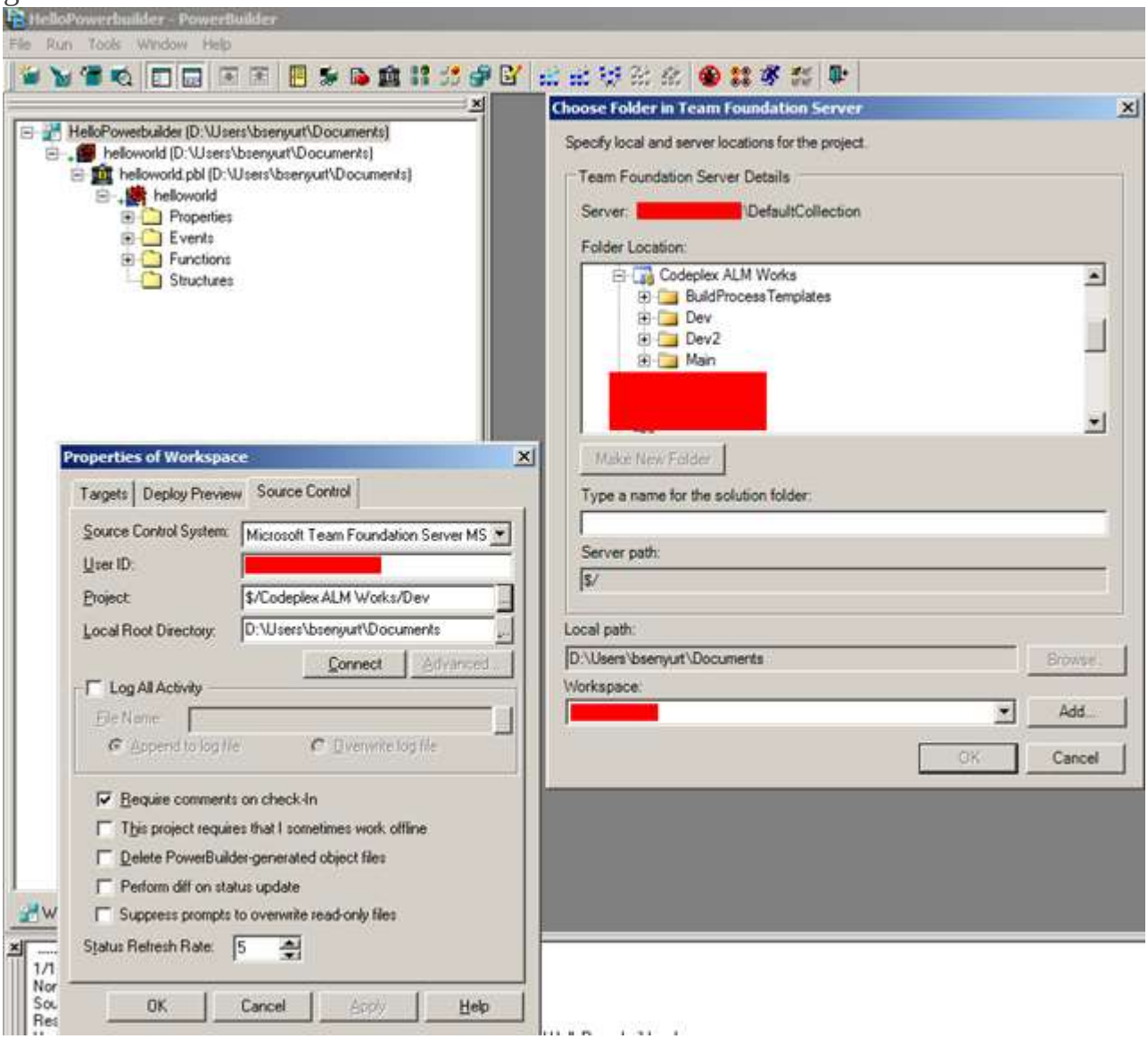
Bazen kalabalık ekiplerin yer aldığı firmalarda, geliştiricilerin bir program install etmeye yetkisi olmaz. Böyle bir durumda gerekli tüm makinelere sistem ekibi tarafından Update geçilir. Ya da geliştirici makinelerinde uzaktan bağlanılarak sessiz modda bir kurulum*(Slient Install)* işlemi yapılır. Bu noktada msi tipinden bir setup dosyasının sessiz modda nasıl çalıştırılacağı da önemlidir. İşte MSSCCI Provider için örnek bir kullanım şekli.

```
msiexec /i"Visual Studio Team Foundation Server 2012 MSSCCI Provider (32-bit).msi" /quiet /norestart
```

Buna göre MSSCCI provider bir arayüz göstermeden arka planda otomatik olarak varsayılan ayarları ile kurulacaktır. /quiet parametresi bunun için kullanılmaktadır.

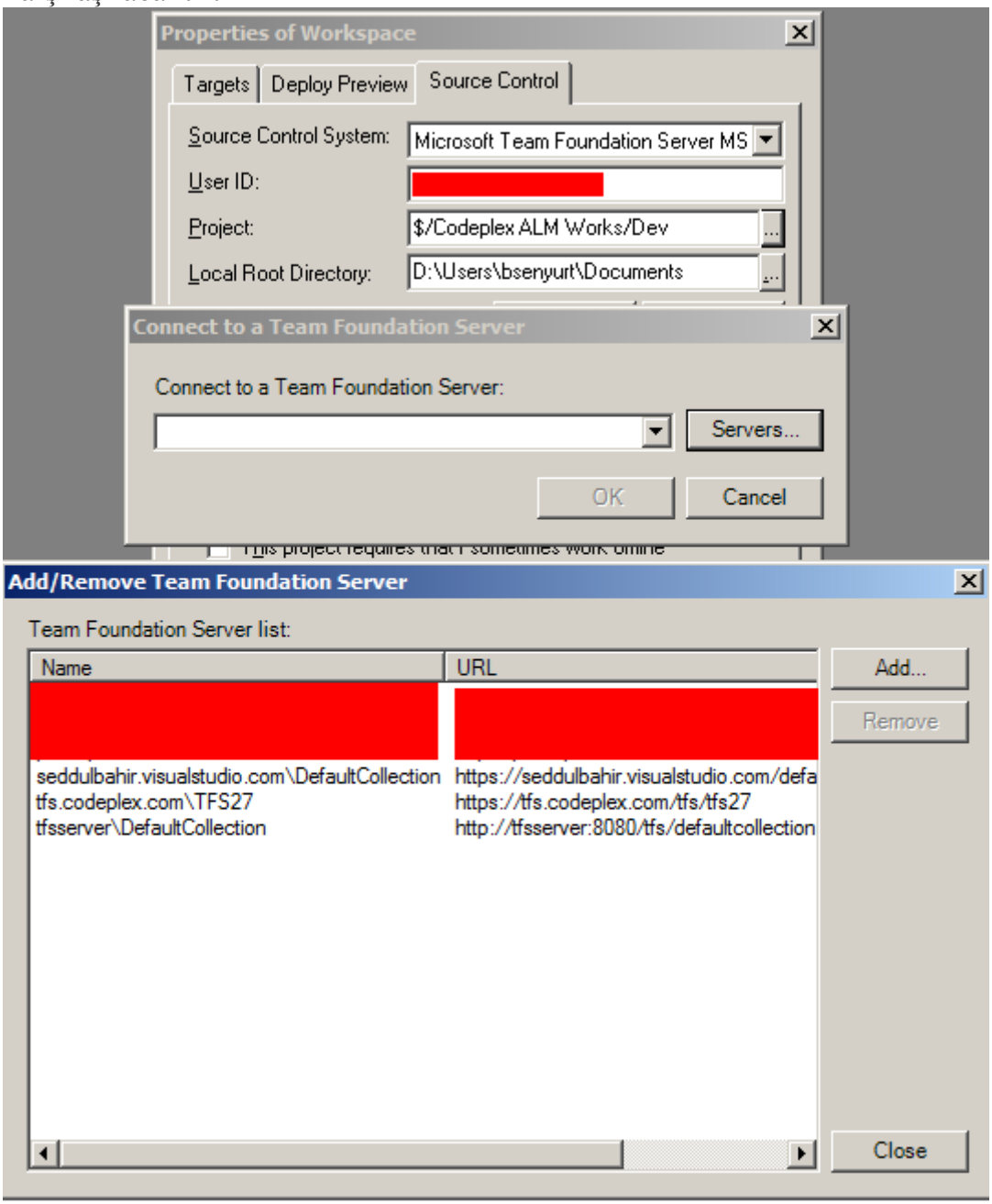
Tahmin edileceği üzere /norestart parametresi sayesinde de, ilgili kurulum işlemi sonrası makinede bir restart yaptırılmaması sağlanmaktadır. Elbetteki başrol oyuncusu msiexec aracıdır.

PowerBuilder gibi ortamların **TFS** ile ortak çalışması **Eclipse**' de olduğu gibi değildir. Aslında**MSSCCI** sağlaycısını kullanan ürünlerin **TFS** ile çalışmaları oldukça farklı olabilmektedir. Örneğin**Powerbuilder'** da **Workspace**' ler ile çalışılmaktadır*(Powerbuilder uzmanı değilim onu belirteyim)*Bu sebepten bir **Workspace**' in **TFS** üzerindeki bir **Team Project'** e hatta bir **Branch**' a dahil edilmesi için **Properties** kısmına gidilmesi ve **Source Control** tabında gerekli ayarların yapılması gerekir.



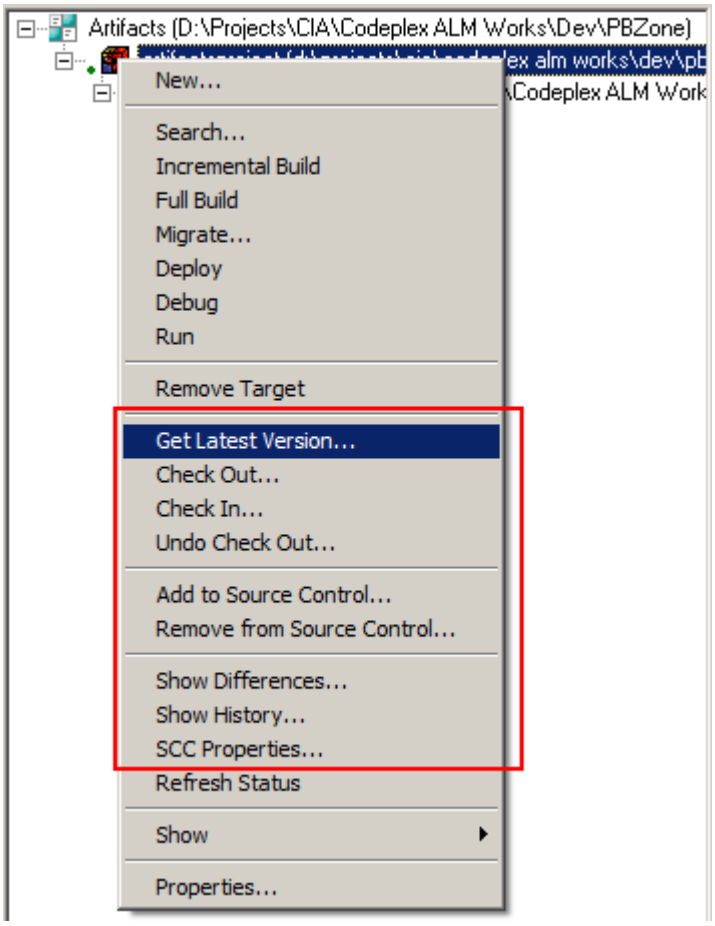
Eğer provider yüklendiyse **Source Control System** listesinde de çıkacaktır. Çok doğal olarak**PowerBuilder** geliştiricisi, **TFS** üzerindeki projeye giderken bir kullanıcı adı ile hareket etmelidir. Bu sepebten projede uygun rolde olması şarttır*(Contributor olur, Project Administrator olur vs*) Dolayısıyla **User ID** kısmında **domainı/kullanıcıadı** gibi bir bildirim yapılmalıdır. **Project** kısmı ise en sevdiğim taraflardan birisidir. Burada 3 nokta

butonuna basıldıktan sonra **TFS** sunucunun adresinin sorulduğu bir pencere ile karşılaşılacaktır.



Eğer sistemde yüklü bir **Visual Studio** ürünü varsa ve daha önceden çeşitli **TFS** sunucularına bağlanıldıysa, bu durumda ilgili **Server** listesinin bu pencerelerde de çıktığı görülecektir. İstenirse**Add** düğmesi ile yeni bir **TFS** sunucu adresinin bildirimi de pekala yapılabilir.

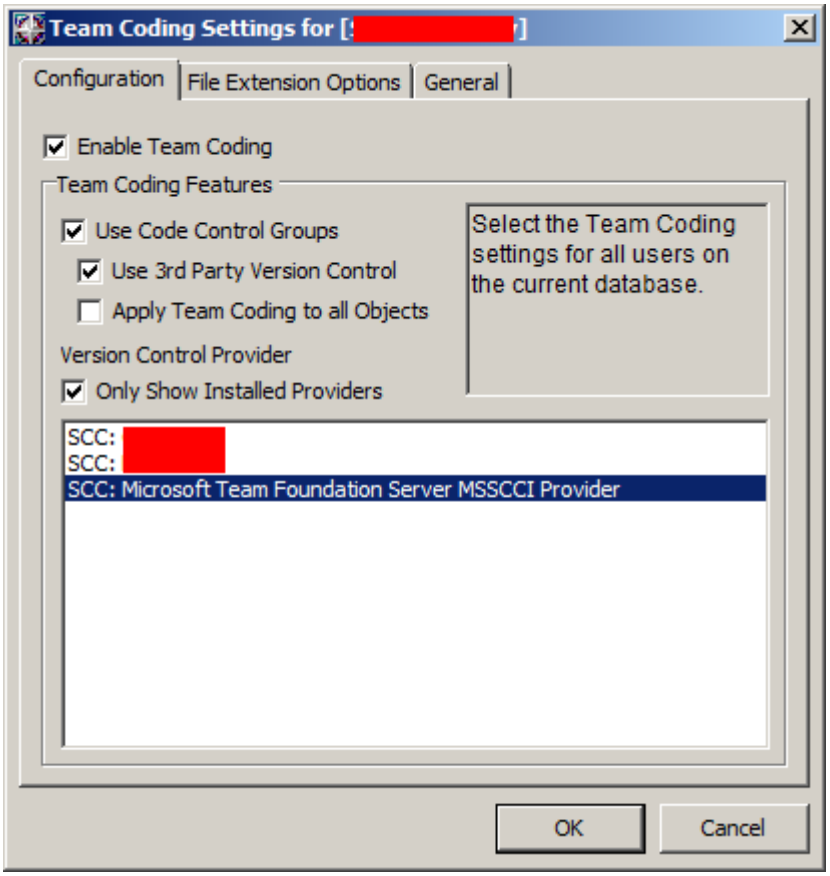
Sunucu seçimi yapıldıktan sonra ise **Choose Folder in Team Foundation Server** isimli bir diğer pencere açılacaktır. Burada ise **Team Project**' in ve ilgili klasörün*(hatta çoğunlukla Branch' in)*seçilmesi yeterli olacaktır. Tüm bu işlemler sonrasında **Workspace**' in belirtilen **TFS Branch'** ine dahil edilmesi işlemi de tamamlanmış olmaktadır. Eğer herşey yolunda gittiyse ve doğru ayarlamaları yaptıysanız bir **Workspace**' in öğelerinde aşağıdaki şekilde görülen özelliklere ulaşılabildiğini fark edebilirsiniz.



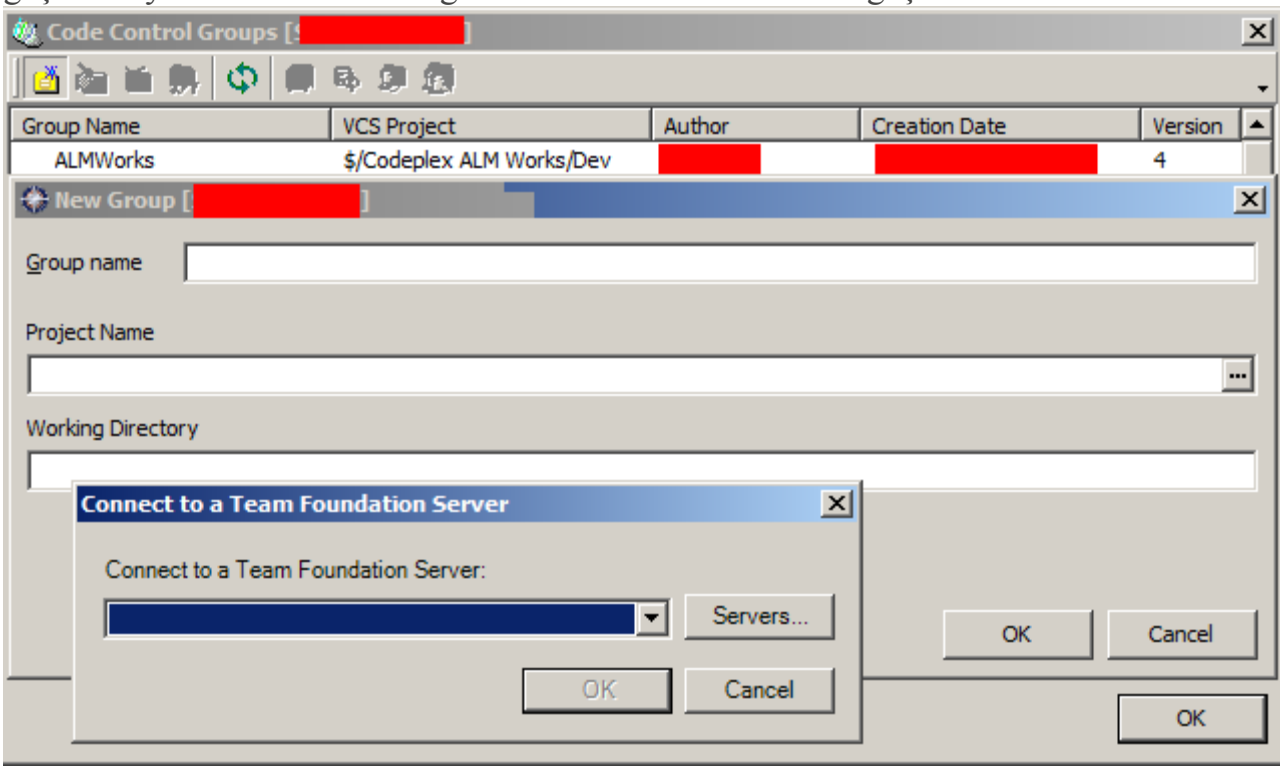
Şimdi bakış açımızı yine eskilerden birisine çevirelim.

SQL Navigator for Oracle 6.5.

Öncelikli olarak **Team Coding** menüsünden **Connection Settings** kısmına girilir. Eğer **MSSCCI**yüklüyse burada **SCC: Microsoft Team Foundation Server MSSCCO Provider** ismi ile listelenecektir.

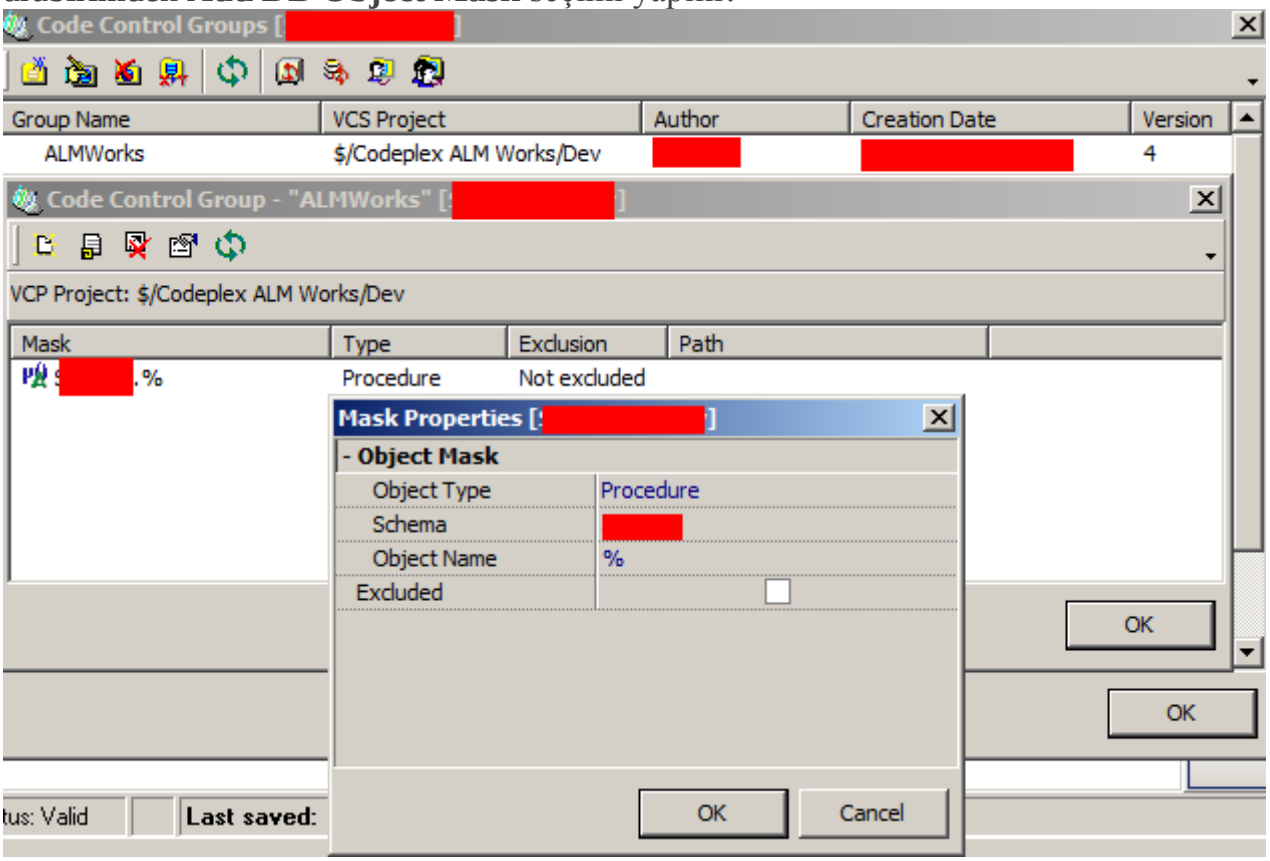


Bu seçim yapıldıktan sonra ise yine **Team Coding** menüsünden **Code Control Groups**' a geçilerek yeni bir kod kontrol grubunun eklenmesi adımına geçilir.

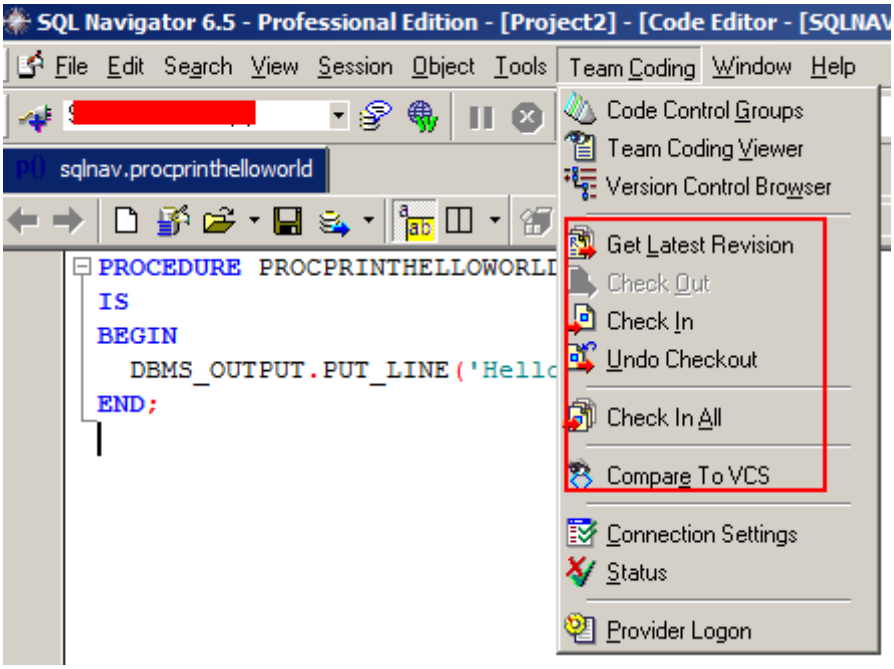


Grup adının girilmesi haricinde en önemli kısım tahmin edileceği üzere **Project Name** seçimidir. Burada yine **PowerBuilder** implementasyonunda olduğu gibi bir **TFS** sunucusunun seçimi ile başlayan adımlar yer almaktadır. Sunucu seçiminin

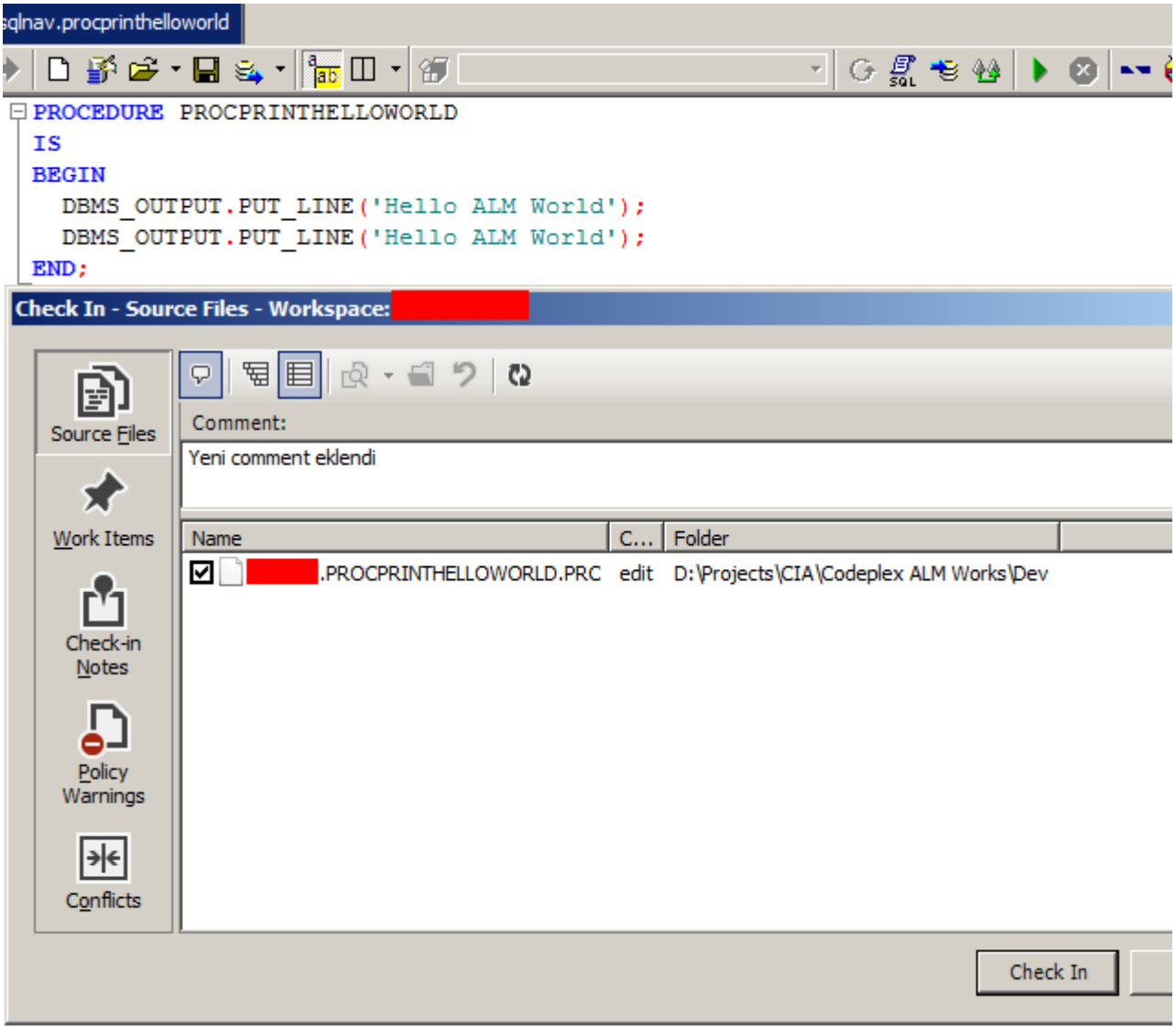
ardından yine **Team Project**' in ve **Branch**' in işaretlenmesi gerekmektedir. Sonrasında ise bir **Working Directory** belirtilmelidir.
Buraya kadarki işlemler tamamlansa da yeterli değildir Bir de **Mask** belirtilmesi ve **Source CodeControl** operasyonlarının ilgili **Oracle** şemasındaki hangi nesneler için yapılacağının belirtilmesi gerekir. Bunun için oluşturulan grup çift tıklanır ve açılan arabirimden **Add DB Object Mask** seçimi yapılır.



Örnekte ilgili şemada yer alan herhangibir **Procedure**' ün ele alınacağı ifade edilmektedir. Bu işlemi de tamamladıktan sonra grub adı seçilerek istenirse **Export to VCS** ile **Mask'** a uygun olarak çıkan listeden nesne seçimleri yapılabilir. Bu kalabalık bir **Procedure** topluluğunda sadece işaretli olanların **Source Code Control** ile ele alınması açısından işe yarar bir özellik olabilir*(Denemedim deneyiniz)* Artık Navigator **IDE**' sinde çalışırken **Procedure**' lerin **Check-In/Check-Out** edilmesi mümkündür. Hatta **Get Latest Version** özellikleri dahi çalışacaktır. Zaten projeyi ilişkilendirirken**Branch** seçimini de yapmıştık ve doğal olarak yapılanların hangi **branch** üzerinde olacağı da bellidir.



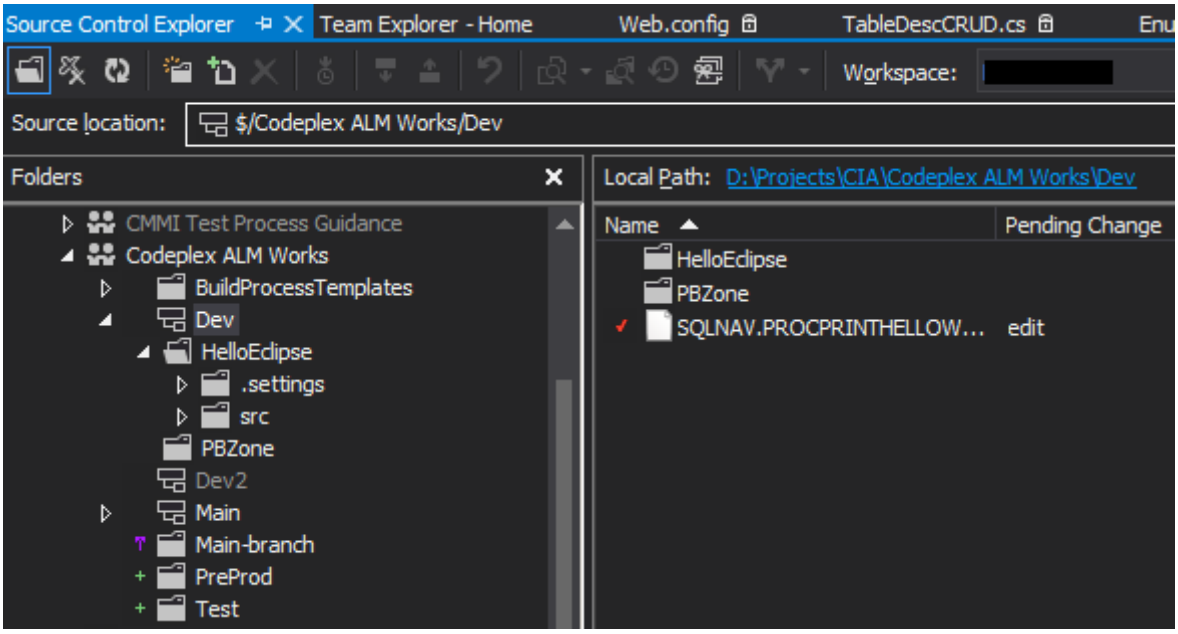
Kurulum aşamalarında değinmedik ancak her hangibir **IDE**' deki bir öğenin **Check-In**' lenmesi sırasında çok tanıdık bir arabirim ile karşılaşılmaktadır.



Edindiğim bu araştırma tecrübeleri sonucunda aşağıdaki ürünleri **TFS 2012**' ye entegre edebildiğimi ve **ALM** içerisinde çalışabildiğimi gördüm. Üstelik çoğu oldukça eski sürümler olmasına rağmen.

- SQL Navigator for Oracle 6.5
- Powerbuilder 9.0.3
- TIBCO Business Studio 3.4
- Eclipse Juno 4.2.1

Tabi kafa da bazı sorular mutlaka oluşacaktır. Özellikle Branch' lerin Merge edilmeleri noktasında. Şunu açıkça ifade edelim ki aslında projenin ana yöneticisi Visual Studio arabirimine ait olan Team Explorer bölümü Source Code Explorer' dır Dolayısıyla Merging gibi bazı yönetimsel operasyonlar zaten bu IDE üzerinden yapılır/yapılmalıdır.



Anlattıklarımız **TFS**' in, **Microsoft** dışı veya eski **Microsoft** ürünlerinin tamamı ile entegre olabileceğini ifade etmese de ucundan da olsa bir tüme varım ispatını gerçekleştirdiğimizi ifade edebiliriz.
Yazının bu kısmına geldiyseniz eğer kafanızda bir soru da oluşmuş olabilir. **3ncü çözüm nedir?**

LINUX/UNIX/MAC OS X Tarafı

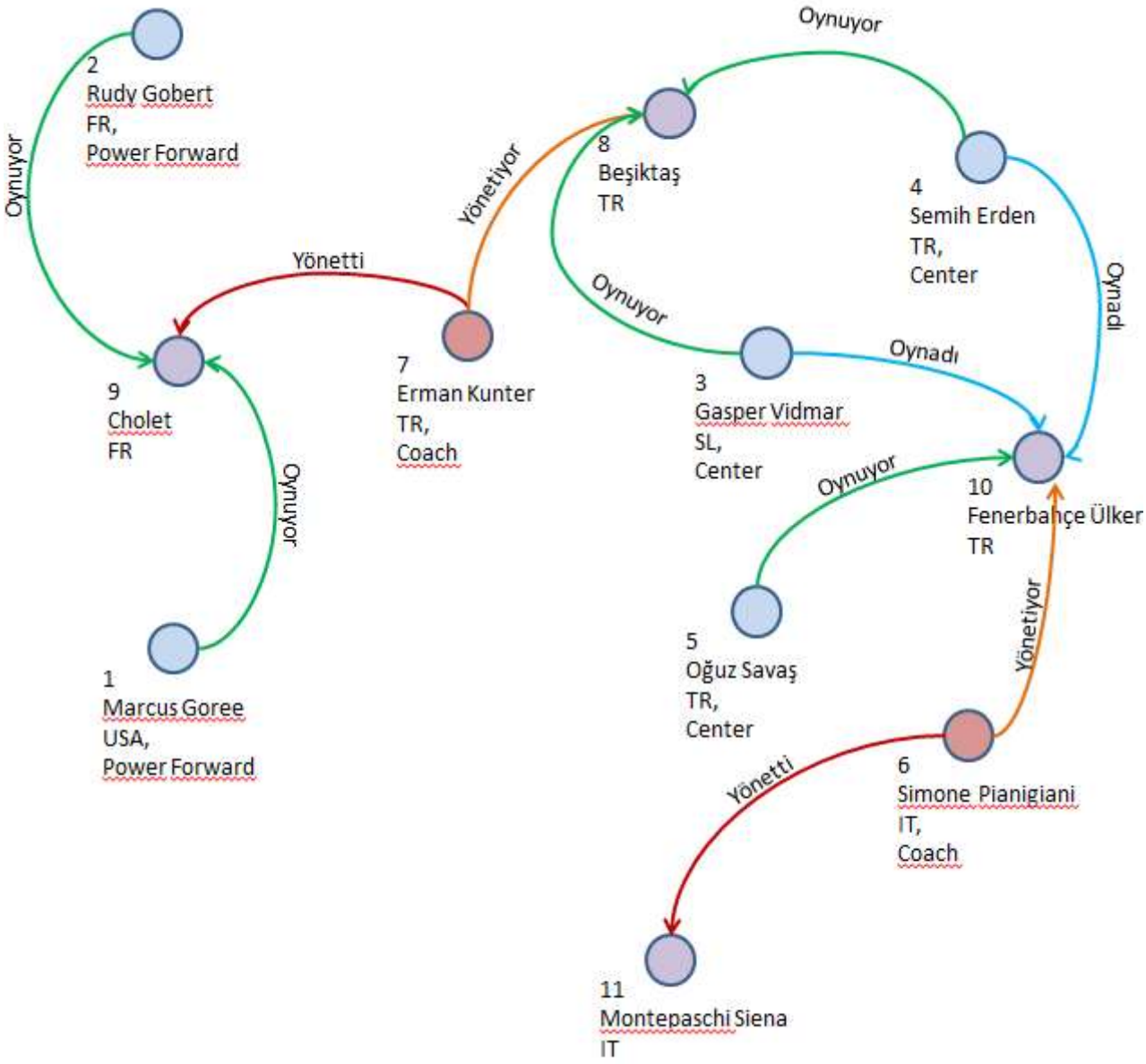
Aslında 3ncü çözüm **Windows** dışı işletim sistemlerini daha fazla ilgilendirmektedir. Örneğin**SOLARIS SPARC** sistemi veya **RED HAT** yüklü bir **LINUX** sistemi söz konusu olabilir. Pek tabi bu platformlar üzerinde yapılan geliştirmelerde komut satırı yaygın olarak kullanılmakta ve hatta ağırlıklı olarak **C** kodları geliştirilmektedir. Hal böyle olunca **TFS** ile olan entegrasyon kocaman bir soru işerati olarak görünmektedir. İşte 3ncü çözüm bu konu ile ilintilidir. **Git-tf**
Sanırım adı siz de bir çağırışım yapmıştır. **Git** ile **Team Foundation Server**' ın arasında bir köprü görevi gören bu araç, **Codeplex** üzerinden sunulan bir projedir. Bu adresten ulaşabilen proje, **27 Ağustos 2012**' de ilk stable sürümünü de çıkartmıştır. Henüz testlerini yapamadım ama ana sayfada da bahsedildiği üzere **Linux** ve **Mac OS X** platformlarında kurulabilen*(nitekim Java Runtime üzerinde çalışmakta olan)* bir ürün. Üstelik komut satırından çalıştırılabildiği için **Linux/Unix**tarafında çalışan geliştiriciler için de oldukça kullanışlı. Bu konuda ilk testleri yaptığımda sanıyorum ki burayı güncelliyor olacağım. *(Şimdilik kurulum ve diğer detaylarla ilişkili olarak Microsoft' un yayınladığı şu dökümana bir göz atabilirsiniz)*
Red Hat Linux' testi gelecek[Henüz yazılmadı]
Bu makalemizde **Team Foundation Server**' ın çevre **IDE**' ler ile olan entegrasyonunu incelemeye çalıştık ve bu konuda başarılı olduğunu gördük. Ancak pek tabiki proje geliştirmek ve özellikle **ALM**akışlarının doğru ve sorunsuz bir şekilde yürütülmesini sağlamak, sadece kullanılan araçların entegre edilmesi ile olacak bir iş değildir. Dolayısıyla

olayın farklı bir boyutu olduğunu da gözden kaçırmamak gerekir. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Graph Database DEX

Çarşamba, 20 Şubat 2013 15:00 by bsenyurt

Merhaba Arkadaşlar,
Eminim pek çoğunuzun hastası/fanatiği olduğu yerli veya yabancı diziler vardır. Küçük bir çocukken çizgi filmlere olan düşkünlüğümüz kadar olmasa da, hemen her bölümünü heyecanla beklediklerimiz mutlaka vardır*(Hatta ülkemizde geç yayınlanıyor diye ilgili dizileri internetten indirenlerimizde vardır)*
Bilişim alanında görev alanların ağırlıkla **CNBC-E** gibi kanallarda yer alan dizilere olan bağımlılığı da aslında su götürmez bir gerçektir. Örneğin benim fanatiği olduğum dizilerden birisi **Dexter** ve ne tesadüftür ki bu gün yazımızda ele alacağımız ürünün adı da onun lakabı ile eş: **DEX**
Daha önceden hatırlayacağınız üzere şuradaki makalede Apache Cassandra' yı, oradaki makalede iseRavedDB' yi incelemeye çalışmıştık. Bu yazımızda ise yine **NoSQL** veritabanı çeşitlerinden birisi olup**Graph** teorisini baz alan **DEX** isimli ürünü incelemeye çalışıyor olacağız.
Sparsity firmasının bir ürünü olan DEX, **Community** kullanımında 1milyon nesneye*(Objects)* kadar ücretsiz olarak yararlanılabilen bir veritabanı sunmaktadır. Veritabanının en önemli özelliği ise içeriği nesnel olarak **Graph** teorisine göre tutuyor olmasıdır. *(Graph teorisi hakkında Wikipediabağlantısından özet bir bilgi alabilirsiniz)*
Kısaca özetlemek gerekirse **Graph** veritabanlarında **Node, Attribute** ve **Edge** adı verilen nesneler söz konusudur. Her bir **Node** ve **Edge** nesnesinin **attribute**' lar ile tanımlanabilen özellikleri mevcuttur**. Graph** veritabanlarında, **node'** lar arası ilişkiler **Edge** örnekleri ile tanımlanmaktadır.**Facebook, Twitter, Linkedin** gibi popüler sosyal ağların veri ambarlarının tasarlanması noktasında son derece isabetli bir seçimdir. Nitekim **node**' lar arası en kısa yolu bulmak veya ilişkileri ortaya çıkarmak, **Graph** teorisi nedeniyle oldukça kolay, tutarlı ve hızlıdır. Bu sebepten sadece sosyal ağlar da değil **IMDB, Wikipedia** tarzı oluşumlarda, **Lojistik, Telekom ağları** gibi daha endüstüriyel çözümlerde de değerlendirilebilmektedir. *(Aslına bakarsanız Graph teorisini uygulayabileceğiniz ne kadar veri bazlı çözüm var ise DEX gibi sistemleri göz önüne alabilrsiniz)*
DEX veritabanı **C++** ile yazılmıştır. **Java, .Net, C++, Blueprints Interface API** desteği bulunmaktadır. Dolayısıyla pek çok farklı platform tarafından da kullanılabilir bir üründür. Şimdi dilerseniz fazla vakit kaybetmeden basit bir **Hello World** uygulaması geliştirmeye çalışalım. Tabi ilk olarak bir senaryoyu göz önüne almamız gerekiyor. Senaryomuza ait basit **Graph** çizimimiz aşağıdaki gibidir.



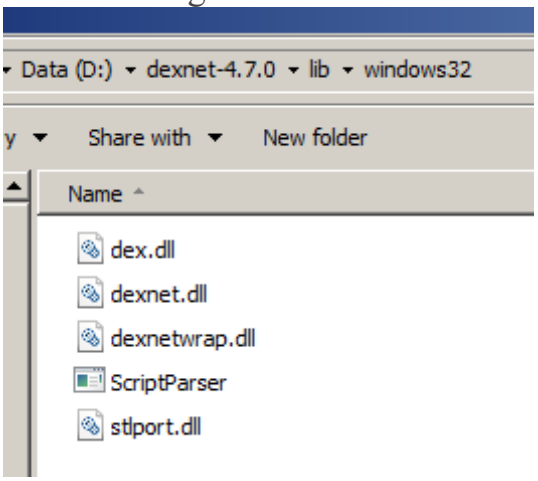
Bu şekli biraz inceleyelim

Basketbol oyuncuları, takım koçları ve takımların yer aldığı bir şema görmekteyiz. Ayrıca bu karakterlerin bazı özellikleri de bulunmaktadır. Örneğin **isimler**, **ülkeler** ve benzersiz olmalarını sağlayan sayısal numaralar gibi. Ayrıca bu karakterler arasında belirli bir yöne doğru çizilmiş ilişkiler olduğu görülmektedir. Tüm bunları birleştirdiğimizde şekle bakarak aşağıdaki cümleleri ve benzerlerini sarf edebilmekteyiz.

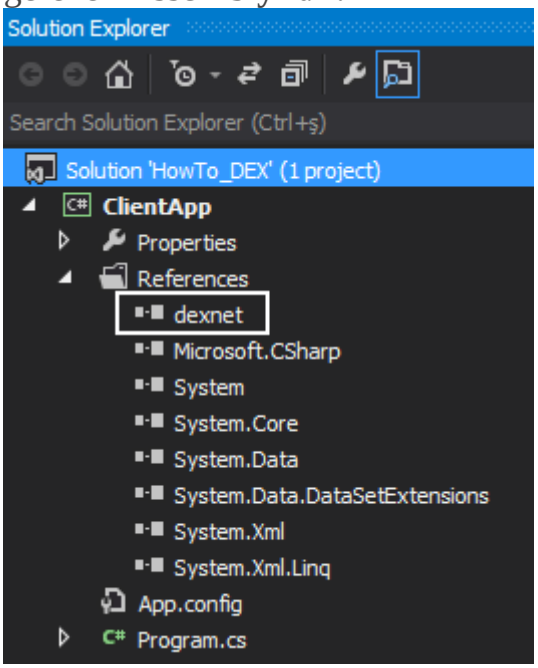
- Simone Pianigiani koç olarak Fenerbahçe Ülker' i yönetmekte olup Montepaschi Siena' yı da önceki bir dönemde çalıştırmıştır.
- Gasper Vidmar şu anda Beşiktaş takımında oynamakta olup daha önceden Fenerbahçe Ülker' de de oynamıştır.
- Semih Erden şu anda Beşiktaş takımında oynamakta olup, daha önceden de Fenerbahçe Ülker formasını giymiştir.
- Amerikalı Power Forward Marcus Goree, daha önceden Erman Kunter' in çalıştırdığı Fransız Cholet basketbol takımının formasını giymektedir.

Örnekler çoğaltılabilir. Aslında Facebook' de yer alan arkadaşlarınızı, dahil olduğunuz grupları, bu teori ışığında kağıda dökmeyi deneyerek kendi örneğiniz üzerinden de ilerleyebilirsiniz.

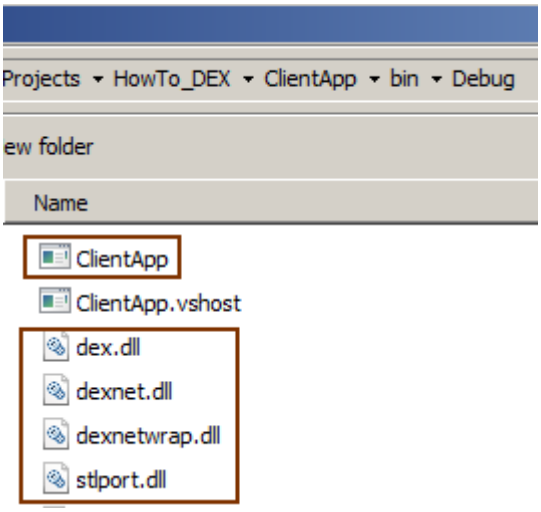
Peki bu cümleleri, bir başka deyişle şekilde görülen **Graph** unsurlarını bilgisayar ortamında nasıl saklayabiliriz? Bu amaçla indirdiğimiz **DEX** ürününü kullanıyor olacağız. Ağız alışkanlığı nedeniyle bir veritabanı olarak tanımladığımız **DEX** aslında aşağıda şekilde görülen bir **kaç DLL** ile birlikte gelmektedir. Yani daha önceden incelediğimiz **RavenDb** gibi bir **Server** uygulamasına veya arayüze sahip değildir. Yine de kavramsal olarak tuttuğu içerik bir veri kümesini ifade etmektedir. Daha çok bir **API** olduğunu ifade edebiliriz. Veriyi disk üzerinde bir dosya şeklinde tutmaktadır.



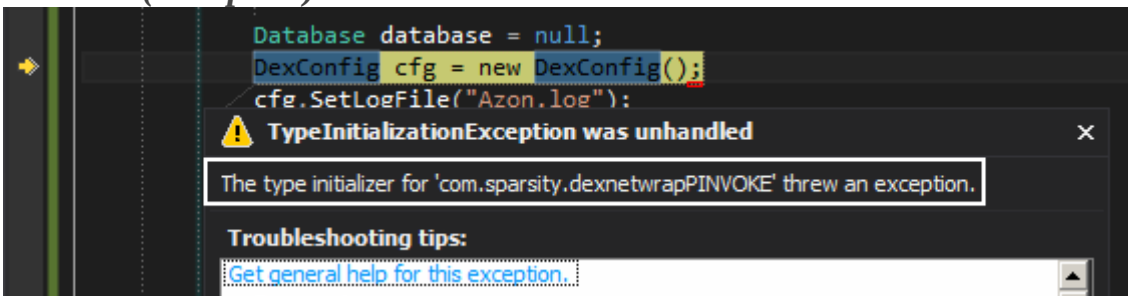
Dexnet.dll bizim kullanacağımız **Wrapper'** dır. Yani projeye referans etmemiz gereken **Assembly**' dır.



Ancak bu yeterli değildir. Diğer **dll** dosyalarının, **Dexnet.dll** ve uygulamaya ait **exe** dosyası ile aynı klasör altında bulundurulmaları gerekmektedir. Yani **dex.dll, dexnetwrap.dll** ve **stlport.dll**dosyalarının da **exe** çıktısının olduğu klasöre kopyalanması gerekmektedir.



Bu işlem yapılmadığı takdirde çalışma zamanında **Platform Invoke** ile ilişkili bir**istisna*(Exception)*** alınacaktır.



Bu hazırlıkların ardından örnek kodlarımızı yazmaya başlayabiliriz. Ben uygulamayı bir **Console**projesi olarak geliştireceğim ve sadece ilk kullanımlarını göstermeye çalışacağım. Bir gerçek hayat senaryosunda **şema*(Schema)*** oluşturulması gibi adımların tek seferde yapılmasını garanti etmeye çalışmalısınız. Hatta şemaların kolayca yapılmasını sağlamak amacıyla ayrı bir arabirim dahi geliştirilebilir*(SQL Server Management Studio tarzı bir şey olmasa da işe yarar bir arayüz pekala çok isabetli bir tercih olabilir)*
İlk olarak veritabanını ve **Graph** nesnelerine ait şemaları tanımlayacağımız kodları yazarak işe başlayabiliriz.

Database database = null;
DexConfig cfg = new DexConfig();
cfg.**SetLogFile**("Azon.log");
cfg.**SetCacheMaxSize**(1024);
//cfg.SetLicense("lisans numarası");
Dex dexter = new Dex(cfg);
database = dexter.Create("AzonGraphDb.dex", "Azon");

Yukarıdaki kod parçasında bir veritabanının oluşturulma adımları örneklenmektedir. Önemli olan noktalardan birisi **Dex** tipini örneklerken bazı konfigurasyon ayarları için **DexConfig** sınıfından yararlanılmasıdır. Örneğin ürünün lisanslı olan bir sürümü alınırsa bunun **SetLicence** metodu ile bildirilmesi gerekecektir. Veritabanı nesne örneği oluşturulduktan sonra **Node, Attribute** ve **Edge**tanımlalarını yaparak şemayı da oluşturabiliriz.

Session session= **database.NewSession();**
Graph g = session.GetGraph();
#endregion
#region Schema' nın Oluşturulması
int teamType = **g.NewNodeType("Takim");**
int teamIdType = **g.NewAttribute(teamType, "TakimId", DataType.Long, AttributeKind.Unique);**
int teamNameType = g.NewAttribute(teamType, "Ad", DataType.String, AttributeKind.Indexed);
int teamCountryType = g.NewAttribute(teamType, "Ulke", **DataType.String**, AttributeKind.Indexed);
int staffType = **g.NewNodeType("Eleman");**
int staffIdType = **g.NewAttribute(staffType, "ElemanId", DataType.Long, AttributeKind.Unique);**
int staffNameType = g.NewAttribute(staffType, "Ad", DataType.String, **AttributeKind.Indexed**);
int staffTitleType = g.NewAttribute(staffType, "Unvan", DataType.String, AttributeKind.Indexed);
int staffCountryType = g.NewAttribute(staffType, "Ulke", DataType.String, AttributeKind.Indexed);
#endregion Schema' nın Oluşturulması
#region Edge Tanımlamaları
// Bu senaryoda sadece Directed Edge kullanılmıştır. Normalde hem Tail hem de Head rolü üstlenen bir node söz konusu ise UnDirected Edge kullanılması gerekir.
// Directed Edge' ler de bir Tail' den Head' e doğru giden bir ilişki ifade edilir
int roleType = **g.NewEdgeType("Rol", false, false);**
int roleTitleType = **g.NewAttribute(roleType, "Title", DataType.String, AttributeKind.Basic);**
#endregion Edge Tanımlamaları

Şema oluşturma işlemlerinde olayın kahramanı **Graph** tipinden olan nesne örneğidir. Bir **Graph**nesnesini örneklemek için veritabanına ait oturumdan yararlanılmalıdır. Bu sebepten bir **Session**örneği oluşturulmuştur. **Session** açık olduğu süre zarfında **Graph** örneği üzerinden yapılan tüm işlemler*(şema oluşturma, veri ekleme, güncelleme vb)* bu oturuma ait olacak şekilde gerçekleşecektir.

Dikkat edileceği üzere takımlar ve oyuncular ile antrenörleri ifade eden elemanlar için birer **Node**üretilmiştir. Her **Node'** un kendine has bir takım nitelikleri vardır. Örneğin her elemanın bir adı, uyruğu ve ünvanı gibi. Bir **Node** üretilirken bir veri tipinin belirtildiğine ve **AttributeKind** ile şekillendirildiğine de dikkat edelim. **Indexed** olarak işaretlenenler sorgulanabilir olduklarını göstermektedir. **Unique**, tahmin edileceği üzere ilgili niteliğin değerinin benzersiz olmasını sağlamaktadır. Hatta **ElemanId** ve **TakimId** bu

anlamda **Primary Key** olmuşlardır. **AttributeKind**ile belirtilebilecek bir diğer değer de **Basic**' tir. **Basic** tipindeki nitelikler sorgulamalarda*(Query)*kullanılamazlar.
Node tanımlalarının ardından dikkat edileceği üzere bir **Edge** örneği de üretilmiş ve bu kendisine bir nitelik de eklenmiştir. Bir tane **Edge** şu andaki senaryomuz için yeterlidir. Bu **Edge'** in ilgili niteliğinden yararlanarak **Oynuyor, Oynadı, Yönetti, Yönetiyor** şeklindeki ilişkileri tesis edebiliriz. Pek tabiki bir gerçek hayat senaryosunda bu tip sabit değerleri bir **Enum** tipi içerisinde toplamak daha doğru bir yaklaşım olabilir.
DEX, **Edge** tanımlamalarını iki şekilde değerlendirmektedir. **Directed** ve **Undirected**. **Directed**ilişkisinde bir kaynak **Node***(Tail olarak adlandırılmakta)* ve bir de hedef **Node** vardır*(Head olarak adlandırılmaktadır)*. **Undirected** ilişkilerde de ise **Node**' lar hem **Tail** hem de **Head** rolündedir. Örneğimizde **Undirected** ilişki senaryosu ele alınmamıştır. Ancak **DEX**' e ait teknik dökümantasyonda örnek bir kullanımı mevcuttur.
Artık şema tanımlamalarımızı yaptığımıza göre örnek verilerin eklenmesi işlemini gerçekleştirebiliriz. Yapacağımız veri eklemeleri ile temel hedefimiz **Graph** görselindeki ilişkileri ve değerleri üretmektir. İşte kodlarımız.

```
#region Örnek Veri Eklenmesi
Value value = new Value();
long marcusGoree = g.NewNode(staffType);
g.SetAttribute(marcusGoree, staffIdType, value.SetLong(1));
g.SetAttribute(marcusGoree, staffNameType, value.SetString("Marcus Goree"));
g.SetAttribute(marcusGoree, staffTitleType, value.SetString("Power Forward"));
g.SetAttribute(marcusGoree, staffCountryType, value.SetString("USA"));
long rudyGobert = g.NewNode(staffType);
g.SetAttribute(rudyGobert, staffIdType, value.SetLong(2));
g.SetAttribute(rudyGobert, staffNameType, value.SetString("Rudy Gobert"));
g.SetAttribute(rudyGobert, staffTitleType, value.SetString("Power Forward"));
g.SetAttribute(rudyGobert, staffCountryType, value.SetString("FR"));
long gasperVidmar = g.NewNode(staffType);
g.SetAttribute(gasperVidmar, staffIdType, value.SetLong(3));
g.SetAttribute(gasperVidmar, staffNameType, value.SetString("Gasper Vidmar"));
g.SetAttribute(gasperVidmar, staffTitleType, value.SetString("Center"));
g.SetAttribute(gasperVidmar, staffCountryType, value.SetString("SL"));
long semihErden = g.NewNode(staffType);
g.SetAttribute(semihErden, staffIdType, value.SetLong(4));
g.SetAttribute(semihErden, staffNameType, value.SetString("Semih Erden"));
g.SetAttribute(semihErden, staffTitleType, value.SetString("Center"));
g.SetAttribute(semihErden, staffCountryType, value.SetString("TR"));
long oguzSavas = g.NewNode(staffType);
g.SetAttribute(oguzSavas, staffIdType, value.SetLong(5));
g.SetAttribute(oguzSavas, staffNameType, value.SetString("Oğuz Savaş"));
```

```
g.SetAttribute(oguzSavas, staffTitleType, value.SetString("Center"));
g.SetAttribute(oguzSavas, staffCountryType, value.SetString("TR"));
long simonePianigiani = g.NewNode(staffType);
g.SetAttribute(simonePianigiani, staffIdType, value.SetLong(6));
g.SetAttribute(simonePianigiani, staffNameType, value.SetString("Simone Pianigiani"));
g.SetAttribute(simonePianigiani, staffTitleType, value.SetString("Coach"));
g.SetAttribute(simonePianigiani, staffCountryType, value.SetString("IT"));
long ermanKunter = g.NewNode(staffType);
g.SetAttribute(simonePianigiani, staffIdType, value.SetLong(7));
g.SetAttribute(simonePianigiani, staffNameType, value.SetString("Erman Kunter"));
g.SetAttribute(simonePianigiani, staffTitleType, value.SetString("Coach"));
g.SetAttribute(simonePianigiani, staffCountryType, value.SetString("IT"));
long besiktas = g.NewNode(teamType);
g.SetAttribute(besiktas, teamIdType, value.SetLong(8));
g.SetAttribute(besiktas, teamNameType, value.SetString("Beşiktaş"));
g.SetAttribute(besiktas, teamCountryType, value.SetString("TR"));
long cholet = g.NewNode(teamType);
g.SetAttribute(cholet, teamIdType, value.SetLong(9));
g.SetAttribute(cholet, teamNameType, value.SetString("Cholet"));
g.SetAttribute(cholet, teamCountryType, value.SetString("FR"));
long fenerbahceUlker = g.NewNode(teamType);
g.SetAttribute(fenerbahceUlker, teamIdType, value.SetLong(10));
g.SetAttribute(fenerbahceUlker, teamNameType, value.SetString("Fenerbahçe Ülker"));
g.SetAttribute(fenerbahceUlker, teamCountryType, value.SetString("TR"));
long montepaschiSiena = g.NewNode(teamType);
g.SetAttribute(montepaschiSiena, teamIdType, value.SetLong(11));
g.SetAttribute(montepaschiSiena, teamNameType, value.SetString("Montepaschi Siena"));
g.SetAttribute(montepaschiSiena, teamCountryType, value.SetString("IT"));
long edge;
edge = g.NewEdge(roleType,marcusGoree,cholet);
g.SetAttribute(edge, roleTitleType, value.SetString("Oynuyor"));
edge = g.NewEdge(roleType, rudyGobert, cholet);
g.SetAttribute(edge, roleTitleType, value.SetString("Oynuyor"));
edge = g.NewEdge(roleType, ermanKunter, cholet);
g.SetAttribute(edge, roleTitleType, value.SetString("Yönetti"));
edge = g.NewEdge(roleType, ermanKunter, besiktas);
g.SetAttribute(edge, roleTitleType, value.SetString("Yönetiyor"));
edge = g.NewEdge(roleType, gasperVidmar, besiktas);
g.SetAttribute(edge, roleTitleType, value.SetString("Oynuyor"));
edge = g.NewEdge(roleType, gasperVidmar, fenerbahceUlker);
g.SetAttribute(edge, roleTitleType, value.SetString("Oynadı"));
```

```
edge = g.NewEdge(roleType, semihErden, besiktas);
g.SetAttribute(edge, roleTitleType, value.SetString("Oynuyor"));
edge = g.NewEdge(roleType, semihErden, fenerbahceUlker);
g.SetAttribute(edge, roleTitleType, value.SetString("Oynadı"));
edge = g.NewEdge(roleType, oguzSavas, fenerbahceUlker);
g.SetAttribute(edge, roleTitleType, value.SetString("Oynuyor"));
edge = g.NewEdge(roleType, simonePianigiani, fenerbahceUlker);
g.SetAttribute(edge, roleTitleType, value.SetString("Yönetiyor"));
edge = g.NewEdge(roleType, simonePianigiani, montepaschiSiena);
g.SetAttribute(edge, roleTitleType, value.SetString("Yönetti"));
#endregion Örnek Veri Eklenmesi
```

Kod satırlarının uzun görünmesine aldırış etmeyin. Temel olarak icar ettirdiğimiz iki fonksiyonel akış söz konusudur. **Node** ve **Edge** oluşturmak. Bir **Node** üretilirken hangi tipten olduğu belirtilir. Ardından elde edilen nesne örneği için söz konusu tipin içerisinde tanımlanan **niteliklere***(Attribute)*değer atamaları gerçekleştirilir.

Edge örnekleri oluşturulurken de ilk olarak **Edge** tipi belirtilmektedir. Tip belirtildikten sonra ise yine **Node** oluşturulmasına benzer olacak şekilde nitelik değerlerinin verilmesi söz konusudur. Her iki kullanımda da değerlerin atanması için **Value** tipinden ve ilgili **Set** fonksiyonundan yararlanılmaktadır. Örneğin **long** tipinden olan **ElemanId** için **value** nesne örneğinin **SetLong**metodundan yararlanılırken, **string** tipte olan takım adları için **SetString** fonksiyonu kullanılmaktadır. Tanımlanan her **Edge** ile **Graph** görselinde yer alan ilişkilerin tanımlandığına dikkat edilmelidir.

Siz tabiki makaleyi okuyup kullanım tekniklerini öğrendikten sonra şöyle güzel janjanlı WPF/Asp.Net ekranları hazırlayarak bu işi daha zevkli hale getirebilirsiniz 

Veri ekleme işlemlerini tamamladığımıza göre basit bir arama işlemi ile devam edebiliriz. Örneğin**Semih Erden**' in bağlı olduğu boğumları bulalım.

```
Objects trace = g.Neighbors(semihErden, roleType, EdgesDirection.Outgoing);
ObjectsIterator iterator = trace.Iterator();
Value nameValue = new Value();
Value countryValue = new Value();
Console.WriteLine("Semih Erden bağlantıları\n");
while (iterator.HasNext())
{
   long objectId = iterator.Next();
  g.GetAttribute(objectId, teamNameType, nameValue);
   g.GetAttribute(objectId, teamCountryType, countryValue);
   Console.WriteLine("Takım {0}, Ülke {1}",
      nameValue.GetString(),
```

```
        countryValue.GetString());
}
```

Yine **Graph** nesne örneğinden yararlanılmaktadır. İlk olarak **Neighbors** metodu ile **semihErden**örneğinin **roleType**' a göre dışarıya doğru olan komşularına gidilmektedir. **roleType** bildiğiniz üzere bir **Edge** örneğidir. Tabi **n sayıda sonuç dönebileceğinden** ileri yönlü bir iterasyona ihtiyaç vardır. Bu sebepten **ObjectsIterator** tipinden bir nesne örneklenmiş ve **while** döngüsüne başvurulmuştur. **HasNext'** in **true** döndürdüğü sürece devam eden döngü içerisinde ise**GetAttribute** metodundan yararlanılarak elde edilen **Node**' un bazı değerleri okunmaktadır. Takım adı ve bulunduğu ülke.

Peki iki **Node** arasındaki **Edge** örneğini nasıl yakalayabiliriz? Bunun için örnek bir kullanım aşağıdaki kod parçasında görüldüğü gibidir.

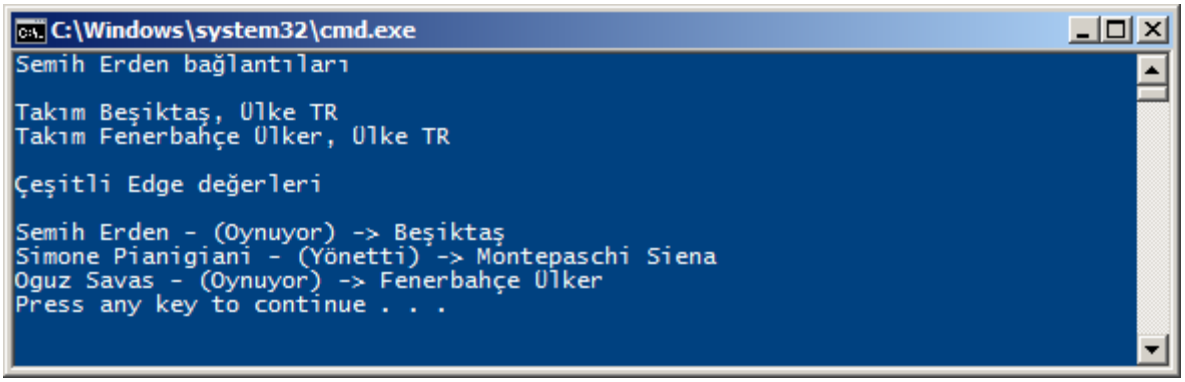
```
#region Edge değeri okumak
Value roleTitleValue = new Value();
Console.WriteLine("\nÇeşitli Edge değerleri\n");
long edgeId=g.FindEdge(roleType, semihErden, besiktas);
g.GetAttribute(edgeId, roleTitleType, roleTitleValue);
Console.WriteLine("Semih Erden - ({0}) -> Beşiktaş",roleTitleValue );
edgeId = g.FindEdge(roleType, simonePianigiani, montepaschiSiena);
g.GetAttribute(edgeId, roleTitleType, roleTitleValue);
Console.WriteLine("Simone Pianigiani - ({0}) -> Montepaschi Siena", roleTitleValue);
edgeId = g.FindEdge(roleType, oguzSavas, fenerbahceUlker);
g.GetAttribute(edgeId, roleTitleType, roleTitleValue);
Console.WriteLine("Oguz Savas - ({0}) -> Fenerbahçe Ülker", roleTitleValue);
#endregion Edge değeri okumak
```

İlk sorguda, **FindEdge** metodu ile **semihErden** ve **besiktas** isimli **Node** örnekleri arasında yer alan**roleType** tipinden olan **Edge** örneğinin nesne numarası alınmaktadır. Bu nesne numarası**GetAttribute** metodunda kullanılır ve **roleTitleType** tipinden olan niteliğin taşıdığı değer yakalanır. Bu senaryo takip eden sorgulamalarda **simonePianigiani** ve **montepaschiSiena** ile**oguzSavas** ve **fenerbahceUlker** için de yapılmıştır.

Yapılan tüm bu işlemler sonrasında ise **Database, Graph, Objects** ve **ObjectsIterator** gibi nesne örneklerinin kapatılması gerekmektedir. Yani bu nesne örneklerine ait **Close** metodlarına çağrıda bulunulmalıdır.

```
iterator.Close();
trace.Close();
session.Close();
database.Close();
```

Uygulamanın çalışma zamanı çıktısına baktığımızda aşağıdaki ekran görüntüsünde yer alan sonuçlar ile karşılaşırız.



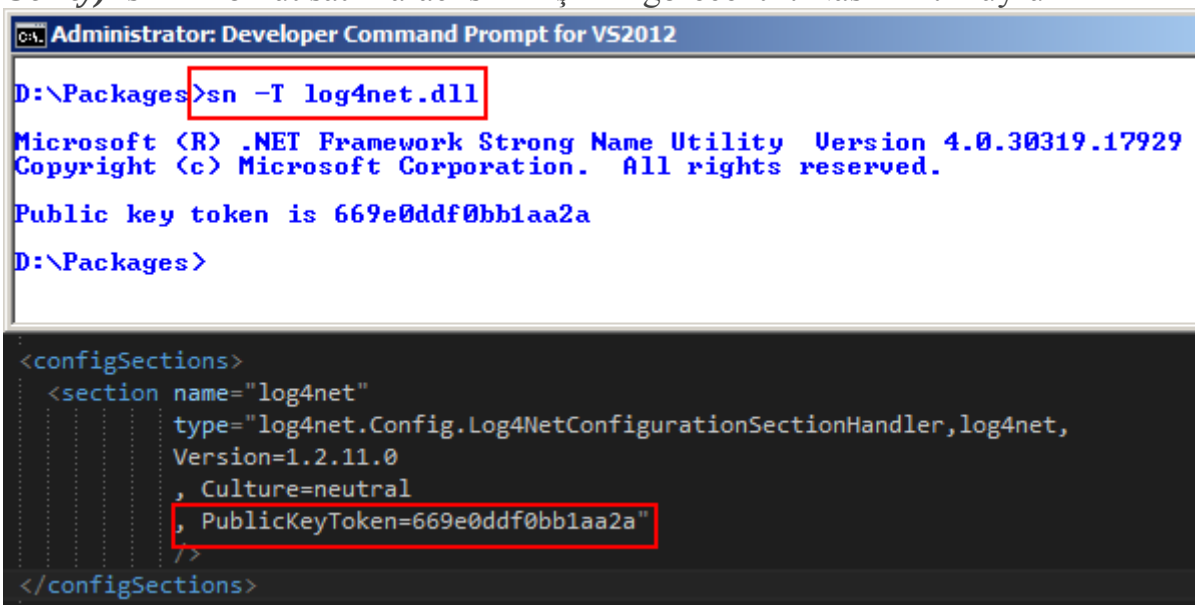
Görüldüğü üzere **Graph** teorisine bağlı kalaraktan, **DEX API**' sinden de yararlanarak tüm**Euroelague** takımları ve oyuncuları için*(hatta bunların içerisine başka nesneleri de katabiliriz)*kocaman bir veri içeriğini oluşturmamız mümkündür. Tabi böyle bir içerik kuvvetle muhtemel**1milyon** nesneyi aşabilir ve dolayısıyla lisans satın alınması gerekebilir. DEX gibi başka pek çok**Graph** veritabanı mevcuttur. Örneğin **Trinity, BigData** vb…Bunları da fırsatım olursa incelemeye çalışıyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim

HowTo_DEX.zip (785,18 kb)

Tek Fotoluk İpucu 80–Bir Assembly' ın Public Key Token Değerini Bulmak

Pazartesi, 18 Şubat 2013 15:00 by bsenyurt

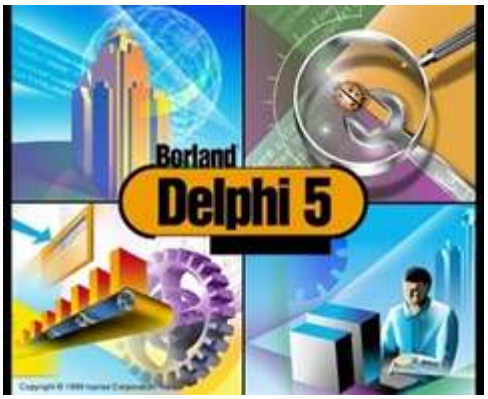
Merhaba Arkadaşlar,
Diyelim ki elinizde projeye referans ettiğiniz bir **.Net assembly** dosyası bulunmakta. Örneğin**Log4Net** ve bununla birlikte konfigurasyon dosyası içerisinde de ilgili **assembly'** ın versiyon numarasını ve daha da önemlisi **Public Key Token** değerini girmeniz gereken bir bölüm yer almakta. Söz konusu **Assembly'** ın **Public Key Token** değerini öğrenmek için pratik olarak nasıl bir yol izlersiniz acaba?
Kodla elde etmek veya **ILDASM** aracı ile **Metadata** kısmında yer alan **hexadecimal** içeriğe bakmak, bir çözüm olabilir mi? Belki de **SN*(Strong Name Utility)*** isimli komut satırı aracı sizin işinizi görecektir. Nasıl mı? Buyrun



Strong Name Utility' yi hep **assembly'** ları işaretlemek için*(özellikle GAC' a atacaklarımızı)*kullanırdık değil mi? Bakın başka pratik kullanımları da varmış. Bir başka ip ucundan görüşmek dileğiyle

Visual Studio 2012 için Entity Framework Yenilikleri

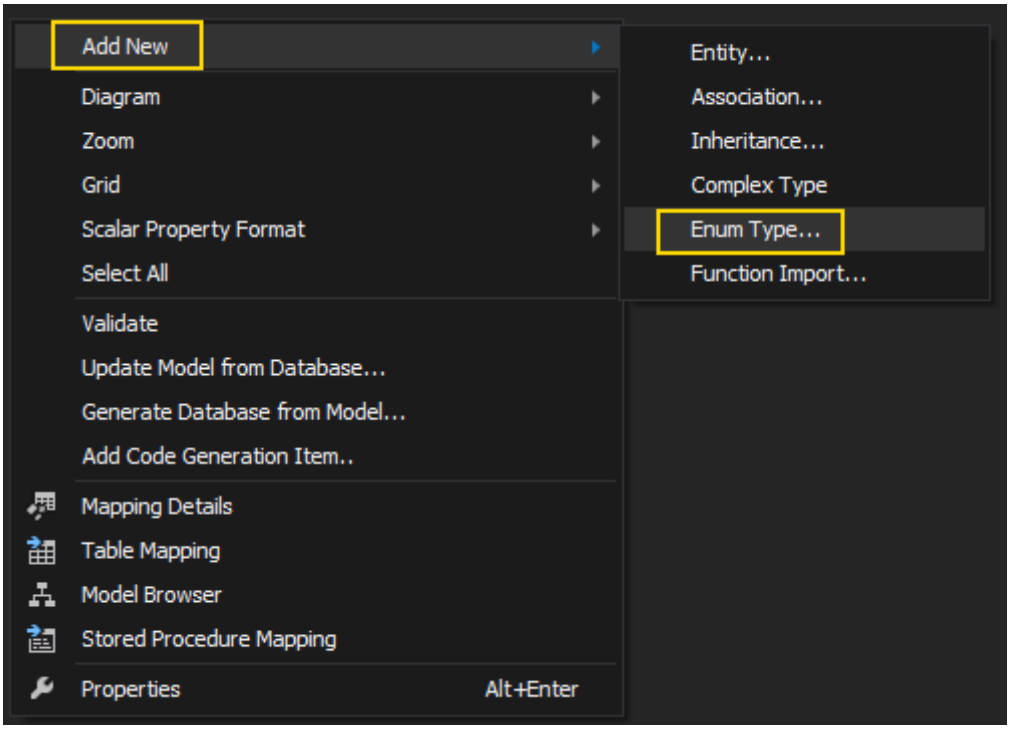
Çarşamba, 13 Şubat 2013 10:18 by bsenyurt



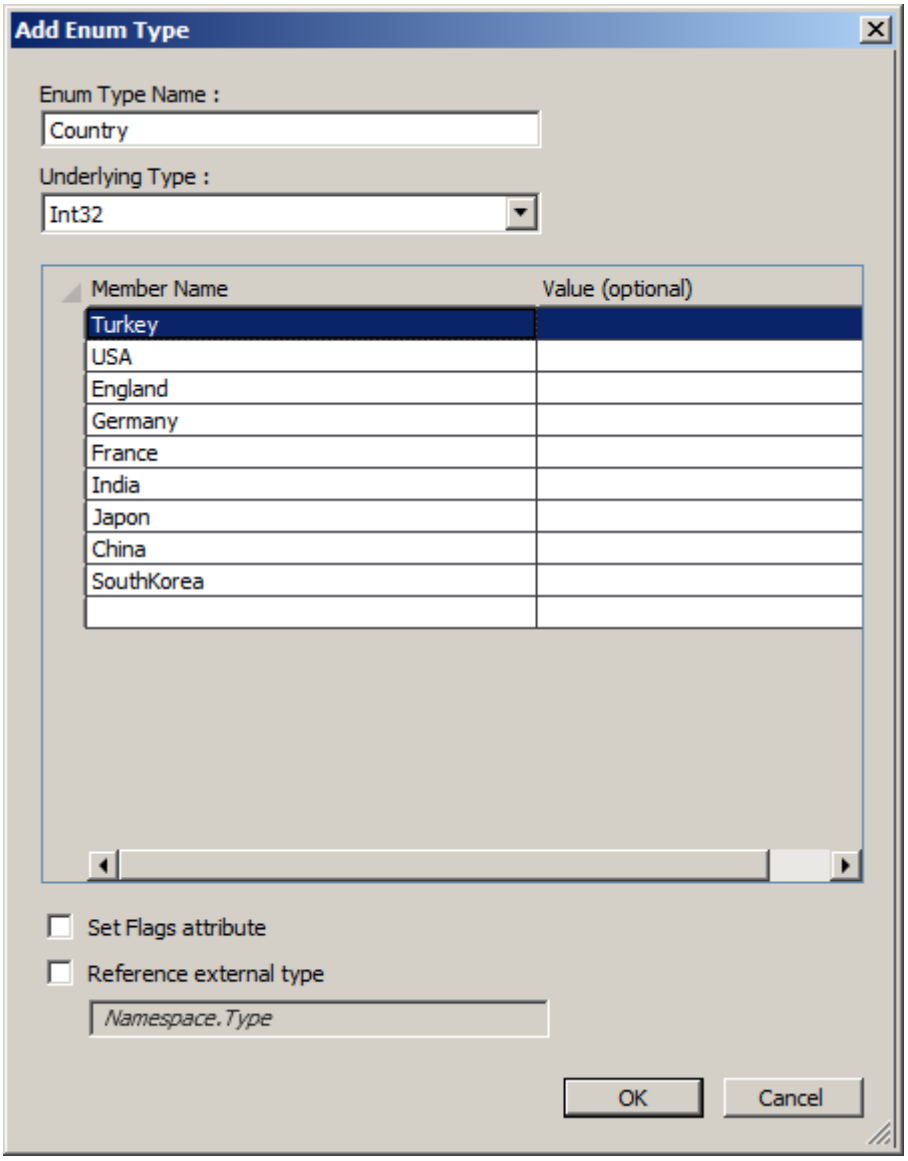
Merhaba Arkadaşlar,
Çok eskiden kullanılan programlama dilleri ve platformları düşünüldüğünde çok ilkel **IDE'** ler ile çalışmış olduğumuzu görmekteyiz. Hatta bazı programlama dilleri ile yapılan geliştirmelerde değil **IDE,** komut satırına mahkum olmuşuzdur(*Gerçi komut satırında script yazarak geliştirme yapmak özellikle fonksiyonel programlama dilleri göz önüne alınırsa oldukça popüler ve isabetlidir)*
Gelişen sistemler, kullanıcı deneyimleri ve görselliklerin artması ile de yazılımcıların daha profesyonel olan **IDE**' ler üzerinde çalışması şart olmuştur.
Bu anlamda tarihin belki en başarılı geliştirme arayüzlerinden birisi, herkesin bildiği üzere **Delphi**ortamıdır. Ne varki o zaman ki **Borland** firmasının sahip olduğu bu özellik, **Anders Heijslberg**' in**Microsoft** takımına geçmesi ile birlikte yerini **Visual Studio** ailesine bırakmıştır. **Anders** sadece**.Net Framework** plaformu ve **C#** diline babalık etmemiş önemli bir **User Experience** tecrübesini de **Microsoft** şirketine taşımıştır. **Visual Studio** özellikle **2008** sürümünden itibaren inanılmaz derecede gelişti ve gelişmeye de devam ediyor.
Bu IDE ile çalışmak hem çok keyifli, hem de tek bir ortam üzerinden çok geniş bir yelpazeye ulaşılabilmekte. Açıkçası **IDE** dışına çıkmadan herşeyin elinizin altında olduğu bir geliştirme ortamında çalışmanın değeri paha biçilemez…Gerisi için **Master Card**
Entity Framework bilindiği üzere son sürümlerinden itibaren çok daha fazla etkili olmaya başladı. Bu noktada **Ado.Net** geliştirici takımının müşteri ihtiyaçlarını da çeşitli anketler yardımıyla dinliyor olmasının önemi büyüktür. Örneğin uzun bir zaman anketin en üst sıralarında yer alan Enum desteğinin getirilmesi gibi. *(Bu amaçla açılan Entity Framework Feature Suggestions forumunu takip etmenizi öneririm)*
Biz bu yazımızda **Entity Framework**' ün **Visual Studio 2012** tarafındaki bazı yeniliklerine değinmeye çalışıyor olacağız*(Bazı diyorum çünkü yazıyı yazdığım sırada yeni bir update çıkmış da olabilir. Lütfen kontrol ediniz ve takipte kalınız)*Haydi gelin hiç vakit kaybetmeden işe başlayalım.

Enum Desteği

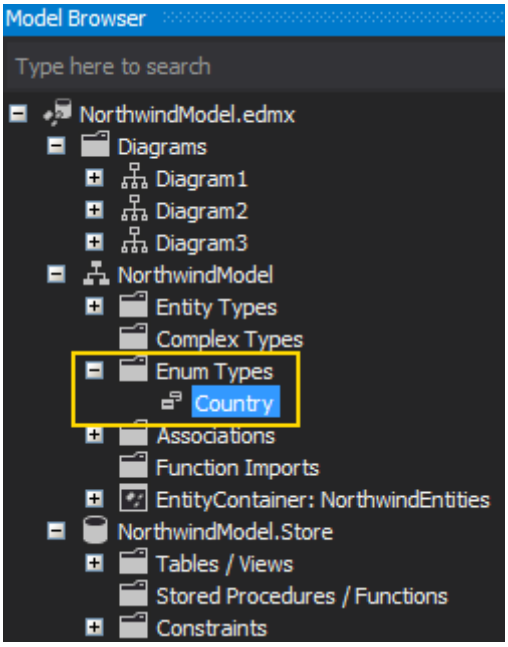
Enum tipi için **Entity Framework** tarafındaki desteği epeyce bekledik aslında. Bu desteğin gelmesi ile birlikte elbetteki **Visual Studio 2012 IDE'** si de gerekli kolaylığı göstermekte. **Model** içerisine yeni bir **Enum** tipi eklenmek istendiğinde tek yapılması gereken, diagram üzerinden **Add New –> Enum Type** seçeneğini işaretlemek.



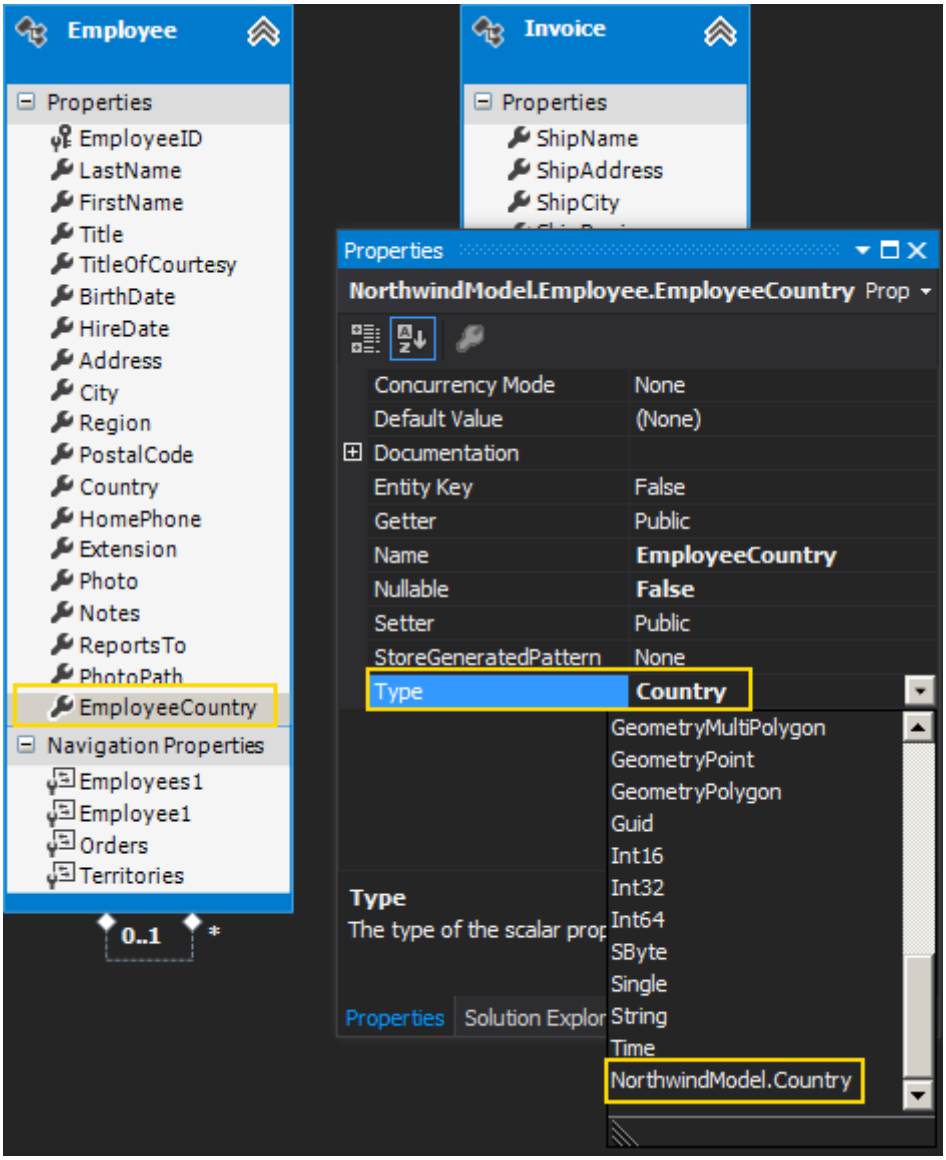
Bu işlemin ardından çıkan iletişim penceresinden, **Enum** tipine ait sabit değerleri girilir. Örneğin aşağıdaki şekilde görüldüğü üzere ülkeleri kullanabiliriz.



İstenirse harici bir **assembly/isim alanı** içerisinde yer alan **Enum** tiplerinin kullanılması da mümkündür. Bunun için **Reference external type** seçeneğinin etkinleştirilmesi ve **[Namespace].[EnumTypeName]** formatında ilgili tip adının girilmesi yeterlidir. **Enum** tipi oluşturulduktan sonra**Model Browser** penceresinde yer alan **Enum Types** kısmında görülecektir.



Enum tipleri için **Designer** tarafında bir destek yoktur*(Şimdilik)* ama bu pek de gerekli olmayabilir.**Enum** tipleri sonuç itibariyle sabitlerini sayısal olarak ifade etmektedir. Bu açıdan bakıldığında, bir**Entity** tip özelliğinin sayısal değerinin karşılığı olarak ilgili **Enum** sabitlerinin kullanılabilmesi de mümkündür. Örneğin **Employee** isimli bir **Entity**' nin **EmployeeCountry** isimli **int** tipinden bir özelliğinin*(Veritabanı tablosunda da bunun karşılığı alanın olduğunu varsayıyoruz)* bulunduğunu düşünelim. Bu özelliği **Country** isimli **Enum** sabiti ile aşağıdaki şekilde görüldüğü üzere eşleştirebilir ve artık kod içerisinde yer **LINQ** sorgularında sayısal değer yerine **Enum** sabitlerini kullanabiliriz.



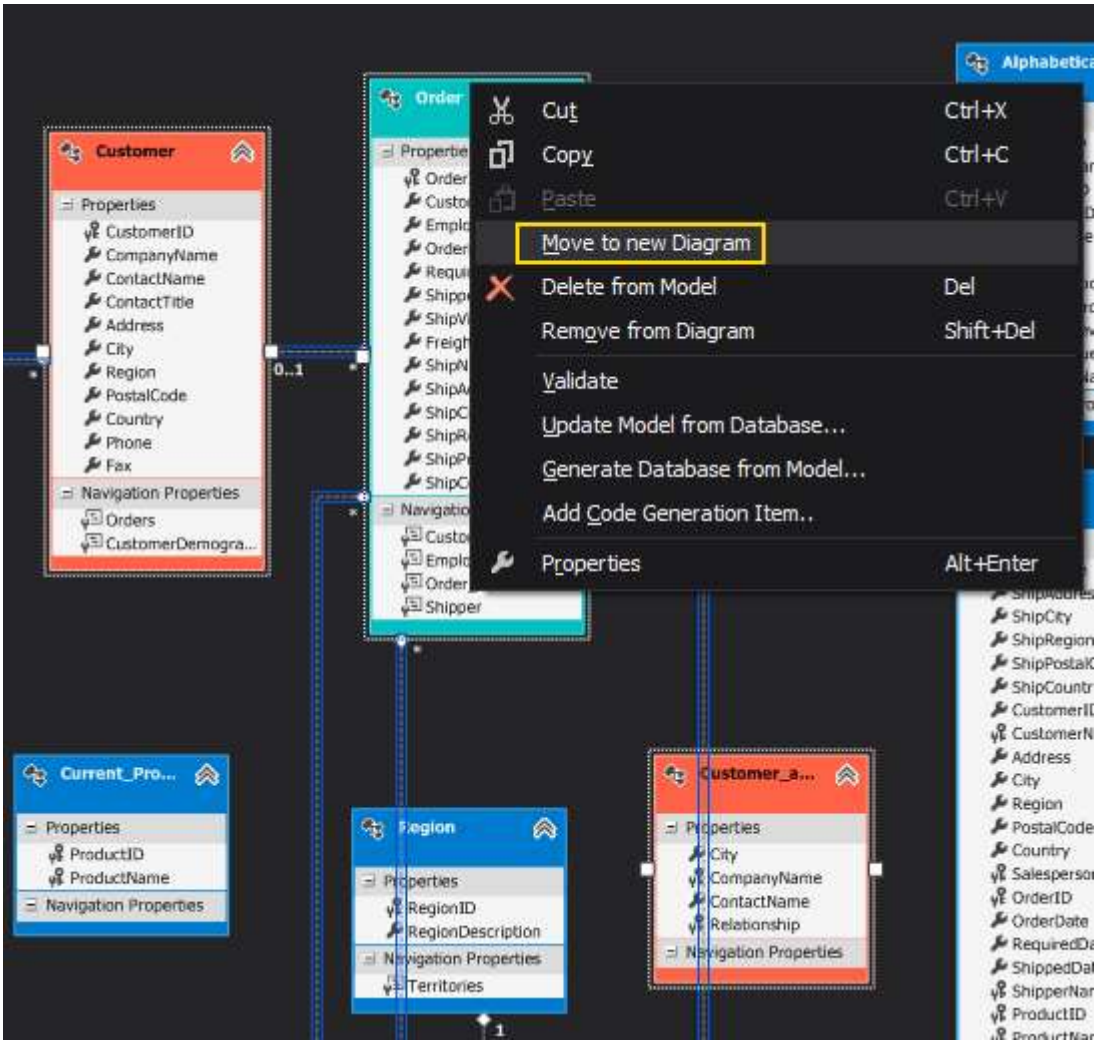
Kalabalık Diagramlardan Kurtulduk

Entity sayısının model üzerinde artması sonucu oluşan görsel kullanım güçlüğünü engellemenin yollarından birisi de, diagramları bölümlemektir. Bunun şu an için kullanışlı olan iki farklı tekniği mevcuttur.

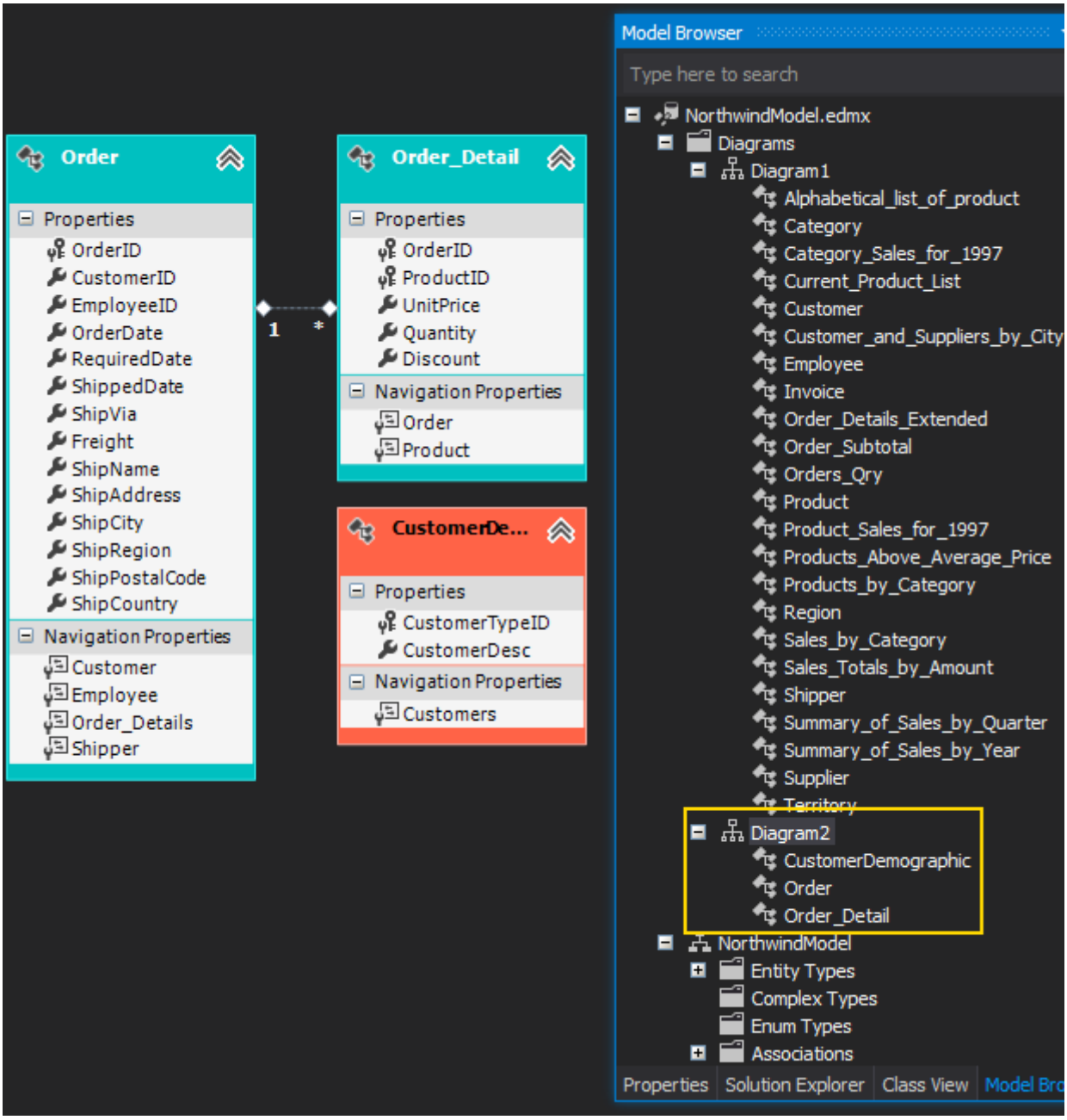
Model Diagramın bölümlenmesinin sebeplerininin başında, kalabalık ve takibi zor olan, çok sayıda Entity içeren modeller yer almaktadır. Ancak tek sebep bu değildir. N sayıda Entity içeren bir model' de sadece bir kaç Entity ile çalışıldığı durumlarda, bu senaryoya konu olan objeleri kolayca takip edebilme ihtiyacı da nedenler arasında gösterilebilir.

Diagram Bölümleme (Move Metodu ile)

Bu teknikte **Model** üzerinden başka bir diagrama alınmak istenen **Entity'** ler seçilir ve **Move to new diagram** ile taşınması sağlanır. Örneğin **Northwind** veritabanı için üretilen model' de yer alan bir kaç **Entity** objesini seçip, sağ tıklama sonrası açılan menüden **Move to new diagram** seçeneğini işaretleyerek söz konusu bölümlemeyi gerçekleştirebiliriz.



Bu işlem sonucunda seçilen **Entity** objeleri yeni bir diagrama alınacak ve bu değişiklik **ModelBrowser** penceresinden de izlenebilecektir. Aşağıdaki çıktıda görüldüğü gibi.



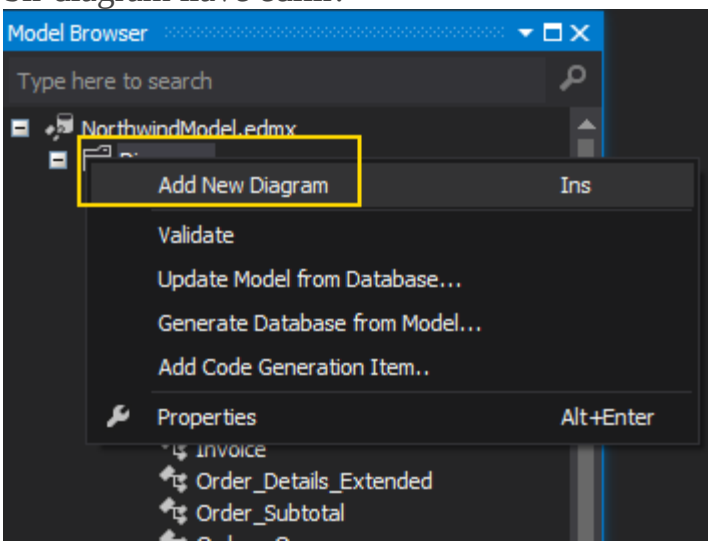
Bu teknikte dikkat edilmesi gereken en önemli nokta gerçek anlamda bir **Move** işlemi yapılmasıdır. Yani, taşınan **Entity** objeleri aslında kaynak diagram üzerinden silinirler. Eğer bu istenmiyorsa takip eden teknikten yararlanılabilir.

Farklı sayıda diagram kullanıldığı durumlarda bunları anlamlı şekilde isimlendirmekte önemlidir. AdventureWorks gibi veritabanlarını düşündüğümüzde, tabloların şema(schema) bazlı olarak tutulduklarını ve domain olarak ayrıştırıldıklarını görürüz.

Buna göre her domain için ayrı bir diagram oluşturmak, görsellik ve takip edilebilirlik açısından oldukça yararlı olabilir. Böyle bir vakada, parçalanan ana modele ait diagramları yine veritabanında olduğu gibi şema adları ile tutmak mantıklı olacaktır.

Diagram Bölümleme (Model Browser üzerinden)

Model Browser, **Entity** diagramlarının yönetilebildiği etkili dialog pencerelerinden birisidir. İstenirse bir modelin bölümlenerek farklı diagramlara taşınması/kopyalanması işlemi buradan da gerçekleştirilebilir. Bunun için öncelikli olarak **Model Browser'** a yeni bir diagram ilave edilir.



Yeni bir diagram oluşturulduktan sonra yine **Model Browser** içerisinde yer alan **Entity'** leri boş alana sürükleyerek söz konusu işlemi gerçekleştirebiliriz. Burada ne yazık ki bir diagramın içindeki öğeyi seçip başka bir diagrama sürükleme işlemi söz konusu değildir*(En azından kullandığım Visual Studio 2012 sürümü için bu geçerli değildi)* Bunun yerine örneğin **NorthwindModel** ismiyle duran öğe altından sürükleme işlemlerinin yapılması gerekmektedir.

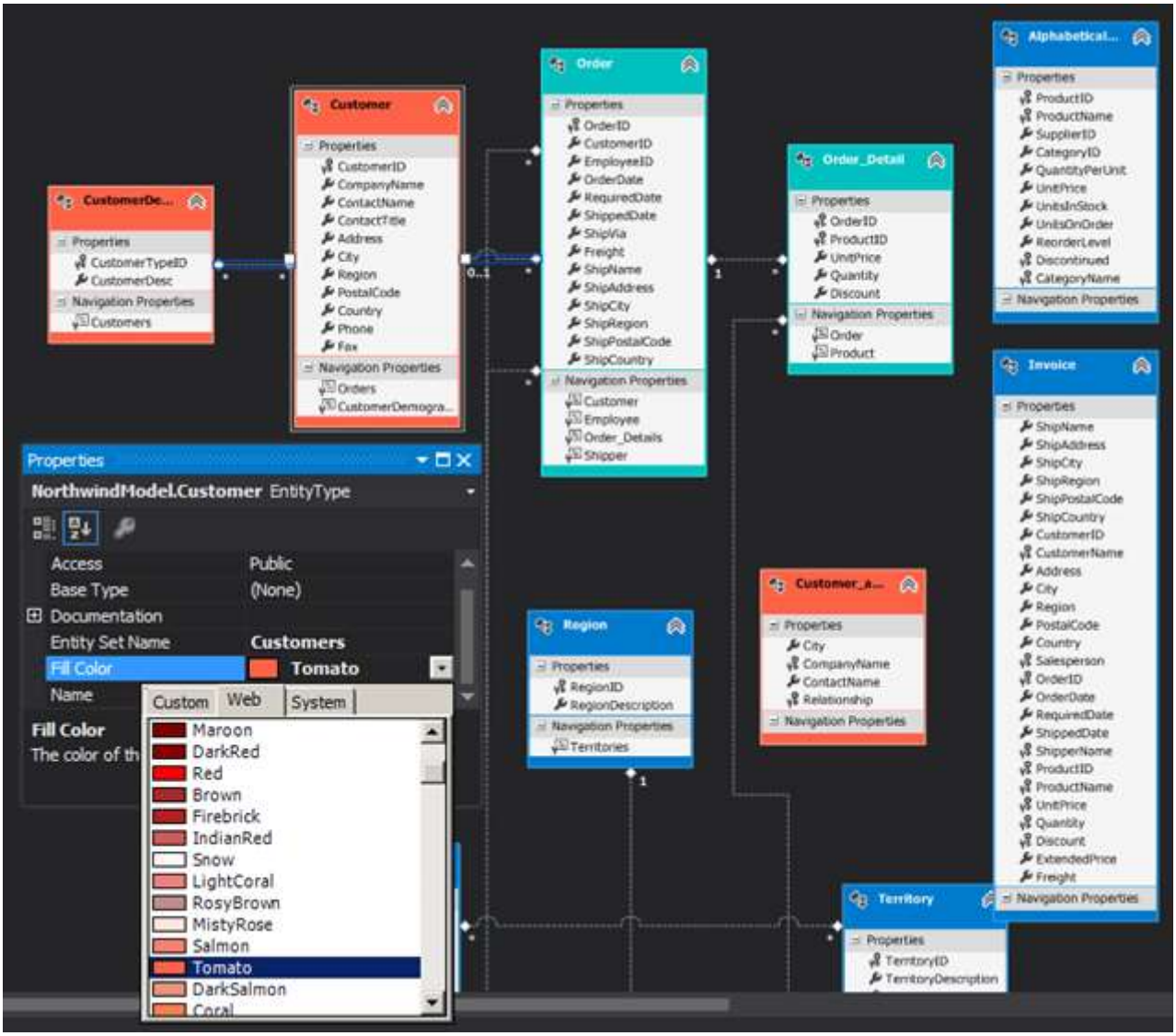
Diagram bölümleme aslında görsel bir ayrıştırma anlamına gelmektedir. Bir başka deyişle diagram bazlı yapılan taşıma, kopyalama gibi işlemler modelin tip yapısını bozmaz.

Daha Renkli Bir Diagram

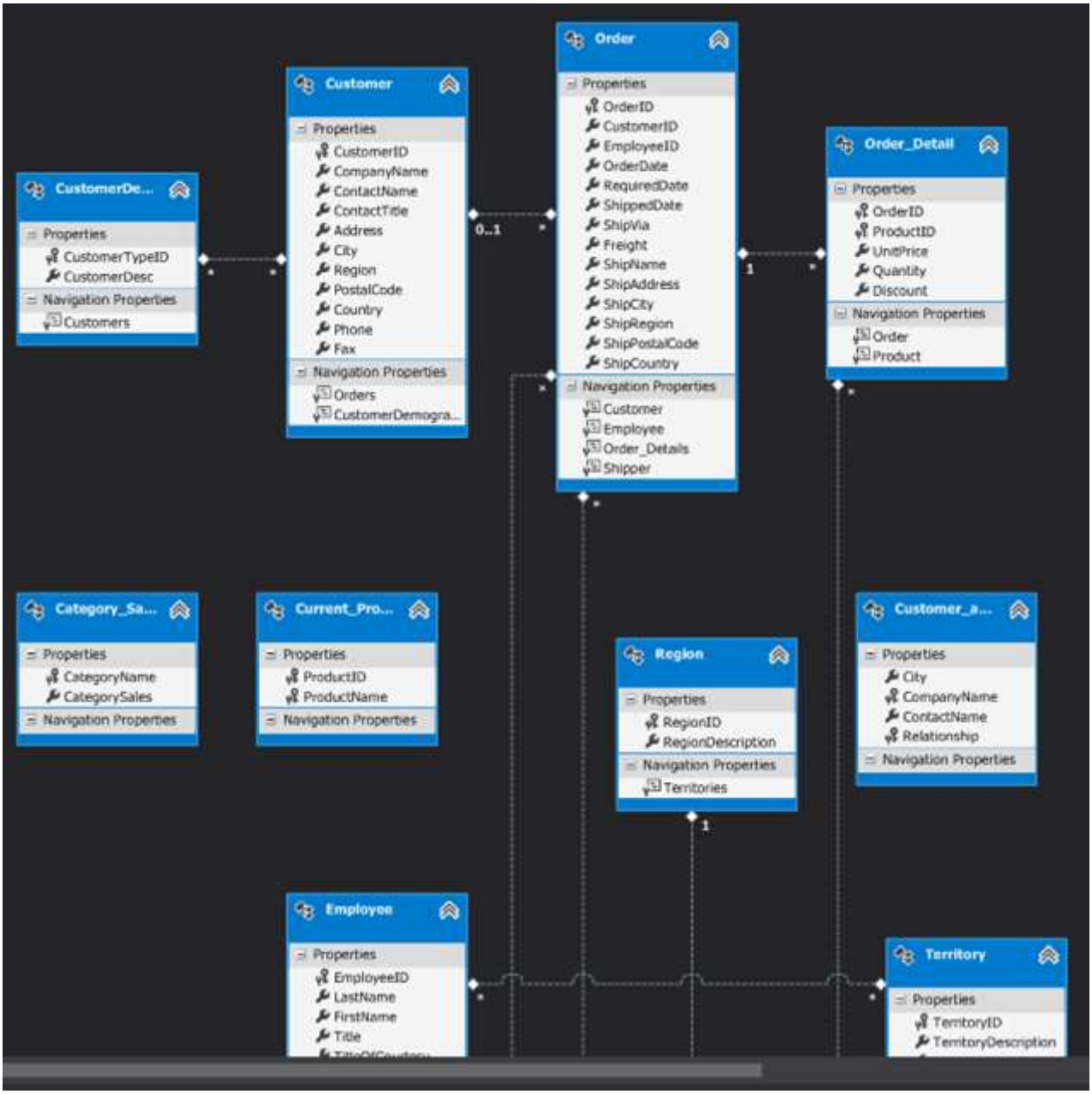
Malum pek çok veritabanı sistemi*(özellikle Relational olanlar ve Enterprise çözümlerde kullanılanlar)*oldukça fazla sayıda **tablo, view** ve **Stored Procedure** içermekte. Hal böyle olunca Entity modellerini içeren diagramların, **Visual Studio IDE** ortamında takip edilmesi de çok zor olabilmekte.

Diagramların bölünebilmesi özelliği haricinde **Visual Studio 2012** ile gelen önemli özelliklerden birisi de, **Entity** objelerinin renklendirilebiliyor olması. Bu özellik sayesinde örneğin aynı isim alanına ait*(veya aynı domain içerisinde yer alması gereken)* **Entity**' leri farklı şekillerde renklendirerek, diagramın hem göze daha hoş görünmesi sağlanabilmekte hem de algının daha da güçlendirilmesi mümkün olmakta. Bunun için bir kaç **Entity** objesini aynı anda seçip, özellikler penceresinden **Fill Color**' ı set etmemiz yeterli olacaktır. Aşağıdaki ekran çıktısında bu kullanıma örnek bir görüntü yer almaktadır

Renkli TV Versiyonu

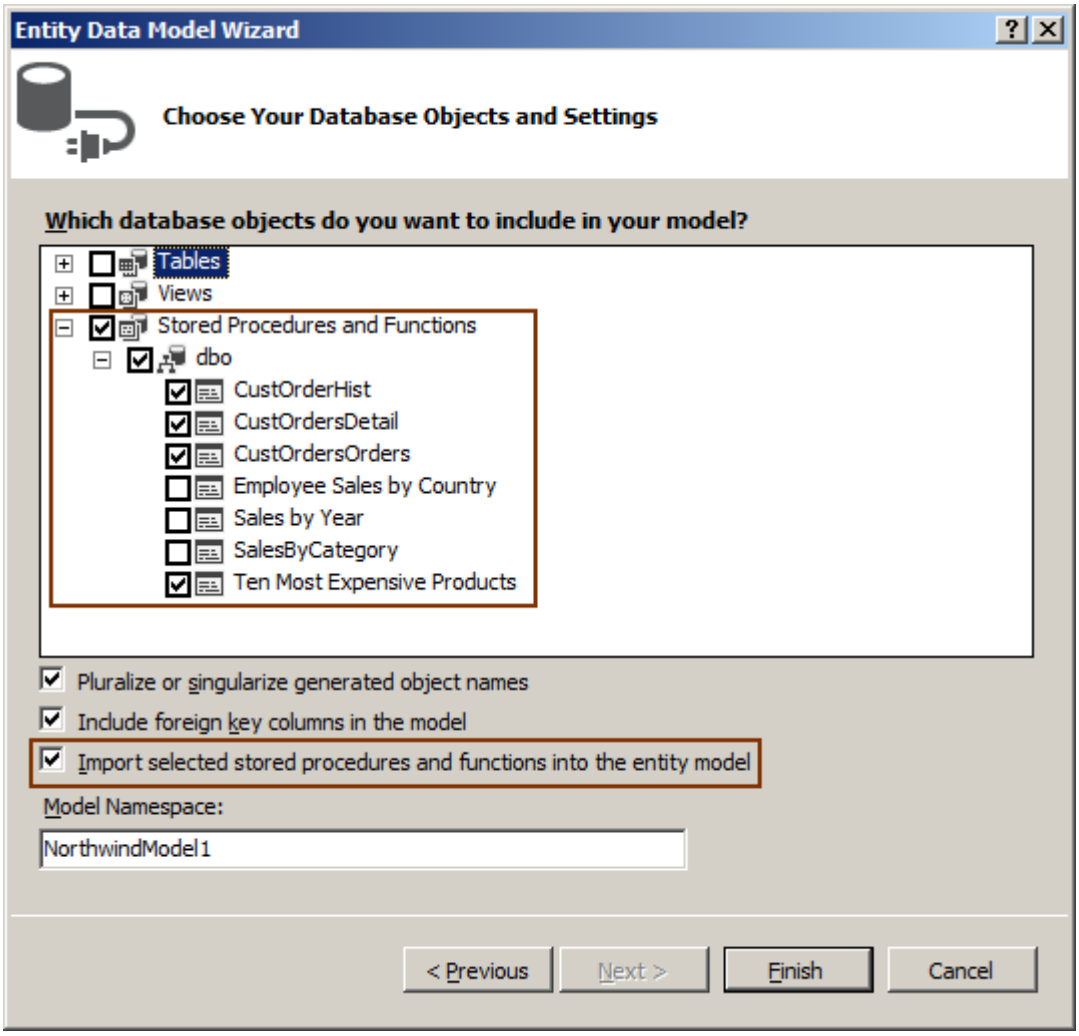


Sıkıcı Olan Siyah Beyaz TV Versiyonu



Birden Fazla Stored Procedure Seçerek Eklemek (Batch Insert)

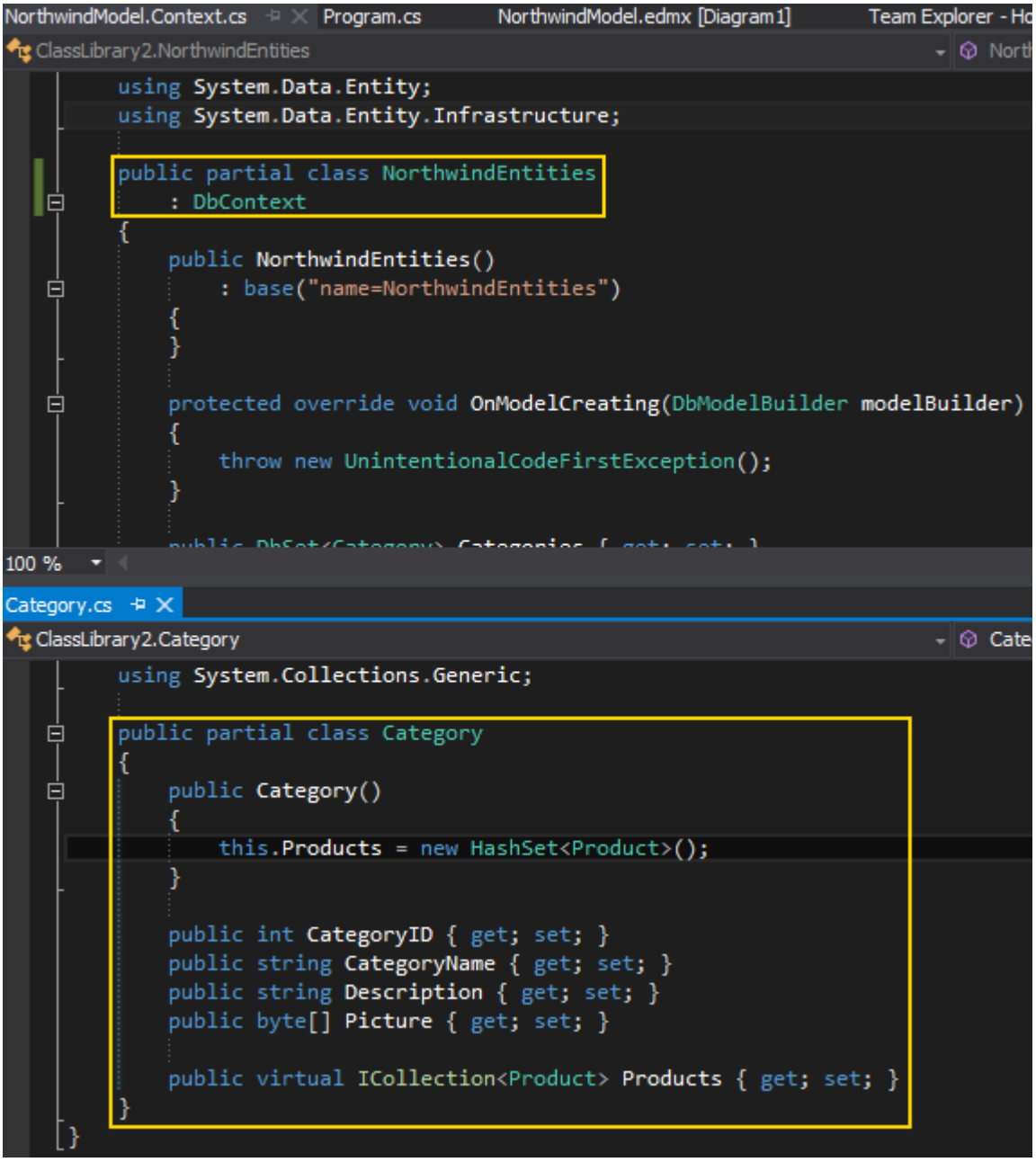
Entity Framework bilindiği üzere veritabanı tarafındaki Stored Procedure' leri birer fonksiyon olarak model tarafına almaktadır. Pek tabi Database First modelin kullanıldığı senaryolarda, Model üretilirken Stored Procedure' lerden istenilenlerinin seçilerek de ilave edilmesi istenebilir. Visual Studio 2012' de bunun için ek bir seçenek getirilmiştir.



Entity Data Model Wizard üzerinde ilerlerken çıkan **Import selected stored procedures and functions into the entity model** seçeneği işaretlendiği takdirde, **Stored Procedures andFunctions** kısmında seçilen yordamlar için gerekli fonksiyonların topluca üretildiği ve modele dahil edildiği görülecektir. Bu, **Visual Studio 2010** da yapılan t anında tek bir **Stored Procedure'** ü fonksiyonelleştirme aksiyonuna göre çok daha iyidir*(Add->Function Import)*

Varsayılan Olarak DbContext Türevli Entity Model

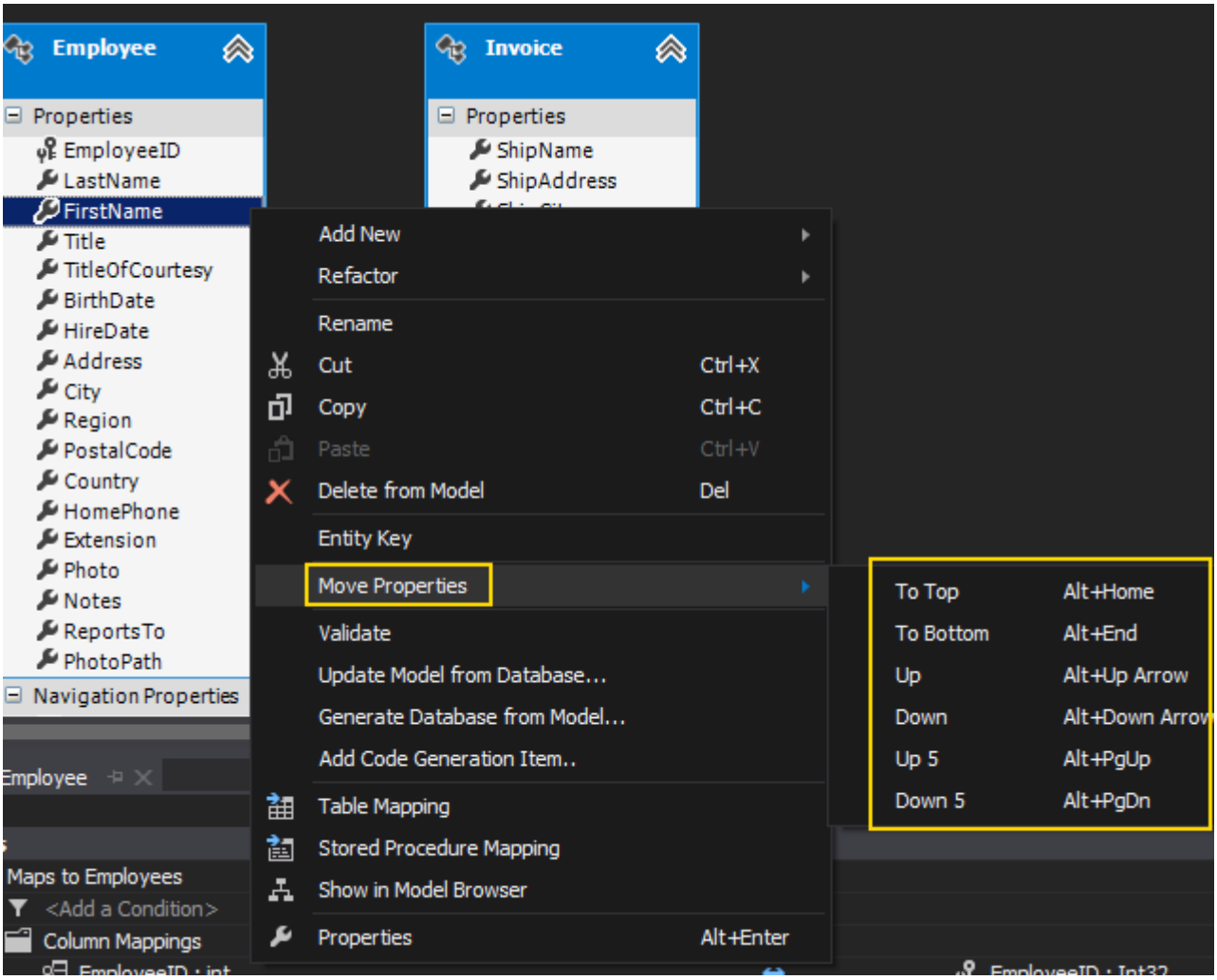
Code First Development yaklaşımının en önemli noktalarından birisi de **POCO***(Plain Old CLR Objects)* dışından **DbContext** türevli bir **Context** tipinin kullanılmasıdır. **Visual Studio 2012**arabirimi **Entity Model**' lerin üretilmesinde artık varsayılan olarak **Entity Framework 5 DbContext T4 Template**' ini kullanmakta ve buna göre de üretilen **entity context** sınıfı **DbContext** türevli oluşturulmakta. Hatta, **Entity** sınıflarının da birer **POCO** tipi olarak üretildiğini görmekteyiz.



Bilindiği üzere **Visual Studio 2010** bu konuda **ObjectContext** türevli bir yaklaşımı benimsemektedir. Ancak **Entity Framework** takımın uygulama geliştiricilere önerisi **DbContext**türevli içerik tipini kullanmaları ve **POCO** sınıfları ile ilerlemeleri yönündedir.

Property Sıralaması

Model diagramda yer alan **Entity** özellikleri varsayılan olarak veritabanındaki kolon sıralamasına göre gelmektedir. Ama istenirse bu özelliklerin sırası değiştirilebilir. Bunun için **Alt+Yön Tuşu**kombinasyonu kullanılabilir*(Örneğin Alt+Up ile özellik bir üste, Alt+Home ile en başa, Alt+End ile en sona geçer)* veya özellikler penceresinden aşağıdaki şekilde görüldüğü gibi ilerlenebilir. *(Visual Studio 2010 IDE' sin de bu işlem için XML tarafına geçmek gerekirdi)*

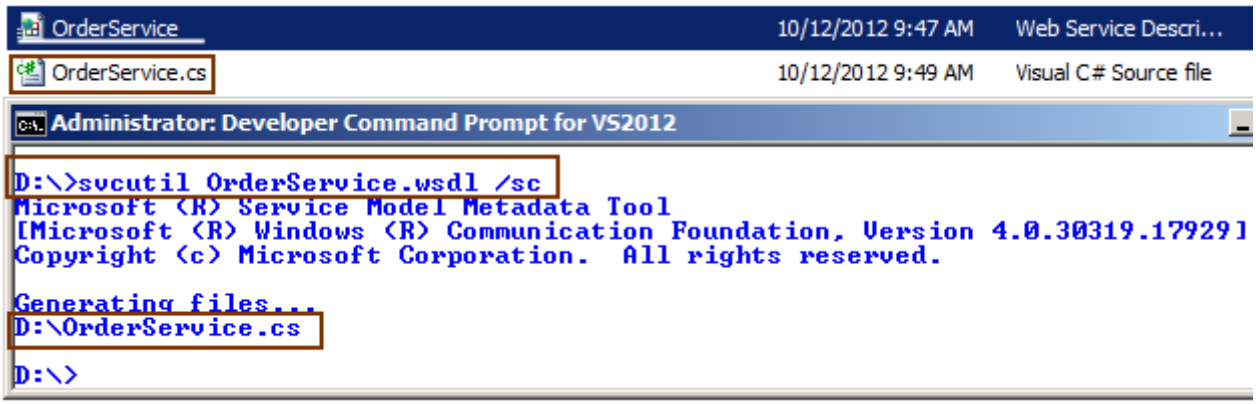


Böylece geldik bir yazımızın daha sonuna. Bu yazımızda özellikle **Visual Studio 2012** ile **Entity Framework** tarafına gelen yenilikleri incelemeye çalıştık. Çok doğal olarak çeşitli **Extension'** lar yardımıyla **Visual Studio 2012** ortamındaki **Entity Framework** niteliklerini arttırmak mümkündür. Bu yazımızda, varsayılan olarak gelen kabiliyetlere bakmaya çalıştık. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

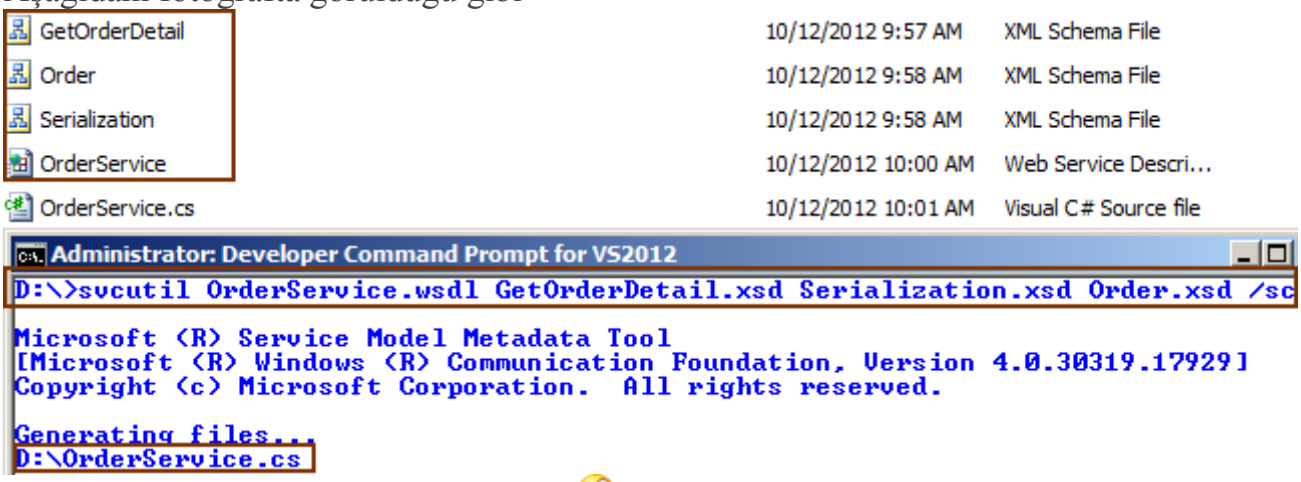
Tek Fotoluk İpucu 79– svcutil ile Contract-First Development

Pazartesi, 11 Şubat 2013 11:56 by bsenyurt

Merhaba Arkadaşlar,
WCF 4.5 tarafında gelen yeniliklerden birisi de **svcutil** komut satırına eklenen **servicecontract***(ya da kısa haliyle sc)* parametresidir. Bu parametre sayesinde bir **WSDL** dokümanından*(ve beraberinde kullandığı XSD' ler var ise onlardan)* **servis sözleşmesinin***(Service Contract)* elde edilebilmesi mümkündür. Tek yapmanız gereken aşağıdakine benzer şekilde **sc** parametresini kullanmanız olacaktır.



Bu örnekte **WSDL** dökümanı **XSD'** leri de bünyesinde barındırmaktadır. Eğer **XSD'** ler harici dosyalarda tutulmaktaysalar onları da komut satırında belirtmeniz gerekecektir. Aşağıdaki fotoğrafta görüldüğü gibi

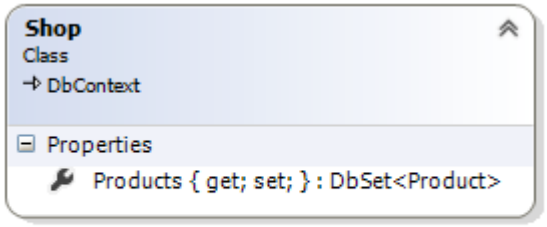


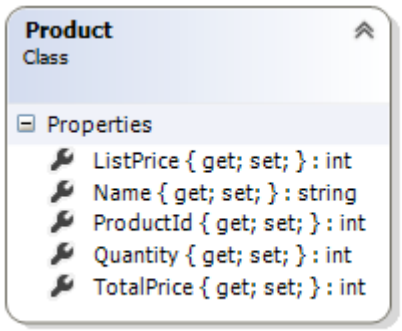
Başka bir ipucunda görüşmek dileğiyle

Entity Framework Code-First için Calculated Fields Kullanımı

Cuma, 8 Şubat 2013 16:15 by bsenyurt

Merhaba Arkadaşlar,
Genellikle göç etmek gibi anlamlarda kullanılan **Migrate**kelimesinin yazılım dünyasındaki karşılığını düşündüğümüzde, elbetteki yandaki fotoğrafta yer alan ve bir birlerinin akvaryumuna atlayan balıklar gelmeyecektir/gelmemelidir.
Ancak **Entity Framework Code-First** yaklaşımı ve**Calculated Fields** kavramını göz önüne getirdiğimizde,**Migration** kelimesini ciddi manada düşünmemiz gerekebilir. Nasıl mı? Haydi okumaya devam
Hesaplanmış alanlar*(Calculated Fields/Columns)* veritabanı programcılığında sık kullanılan özelliklerden birisidir. Bu alanların içeriği genellikle tablonun diğer alanları kullanılarak bir hesaplama sonucu üretilir. Söz gelimi personel verilerinin tutulduğu bir tablodaki FirstName ve LastName alanlarının değerleri birleştirilerek, bir **Calculated Field** oluşturulması mümkündür. Peki bu desteği **Entity Framework Code-First** yaklaşımında nasıl değerlendirebiliriz?
Bildiğiniz üzere **Entity Framework Code-First** yaklaşımında, veritabanı nesnelerinin tasarımları**POCO(Plain Old CRL Object)** tipleri üzerinden gerçekleştirilmektedir. Dolayısıyla **Calculated Field** şeklinde düşünülmesi gereken bir özelliğin veritabanı tarafına nasıl yansıtılacağı kafalarda bir soru işareti oluşturmaktadır. Pek tabi bunun için de bir **nitelik(attribute)** desteği sunulmuş olabilir ki öyledir. **DatabaseGenerated** niteliğinde **DatabaseGeneratedOption.Computed enum** sa biti değerini kullanarak, istenilen hesaplanabilir alan bildirimlerini yaptırabiliriz. Acaba durum gerçekten böyle midir?
Dilerseniz basit bir örnek üzerinden hareket ederek konuyu incelemeye çalışalım. İlk etapta aşağıdaki sınıf çizelgesinde*(Class Diagram)* yer alan tipleri geliştirdiğimizi düşünelim. Senaryomuzdaki başrol oyuncuları, **Shop** isimli **Context** tipi ve **Product** sınıfının **TotalPrice** özelliğidir.





Product isimli örnek **POCO(Plain Old CLR Object)** tipi;

```
using System.ComponentModel.DataAnnotations.Schema;
namespace HowTo_CalculatedFields
{
   public class Product
   {
      [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
      public int ProductId { get; set; }
      public string Name { get; set; }
      public int ListPrice { get; set; }
      public int Quantity { get; set; }
      [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
      public int TotalPrice {
         get;
         private set;
      }
   }
}
```

DbContext türevli Context tipi;

```
using System.Data.Entity;
namespace HowTo_CalculatedFields
{
   public class Shop
      :DbContext
   {
      public DbSet<Product> Products{ get; set; }
```

```
    }
}
```

Product tipi içerisinde yer alan **TotalPrice** özelliğine dikkat edelim. Bu özellik içerisinde ürünün fiyatı ve miktarından yararlanılarak gerçekleştirilen bir hesaplama işlemi söz konusudur. Bunun veritabanı tarafına da yansıtılması için **DatabaseGenerated** niteliğinden yararlanılmaktadır. Peki çalışma zamanı bu durumu anlayabilecek midir?

Örneğimizde Code-First yaklaşımına istinaden config dosyasında aşağıdaki bağlantı bilgisini kullanmayı tercih ettim. Herhangibir bilgi ifade etmediğimizde SQL Express sürümü üzerinde bir veritabanı oluşturulmaya çalışıldığını hatırlatmak isterim. Diğer önemli bir nokta da DbContext türevli sınıf adı ile ConnectionString elementinin name niteliğinin değerlerinin aynı olmasıdır. Bu sayede çalışma zamanı Shop veritabanı için gerekli bağlantı bilgisini bulabilir.

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="entityFramework"
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=4.4.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false"/>
  </configSections>
  <entityFramework>
    <defaultConnectionFactory
type="System.Data.Entity.Infrastructure.SqlConnectionFactory,
EntityFramework"/>
  </entityFramework>
  <connectionStrings>
    <add
      name="Shop"
      connectionString="data source=localhost;database=Shop;integrated
security=true"
      providerName="System.Data.SqlClient"/>
  </connectionStrings>
<startup><supportedRuntime version="v4.0"
sku=".NETFramework,Version=v4.5"/></startup></configuration>
```

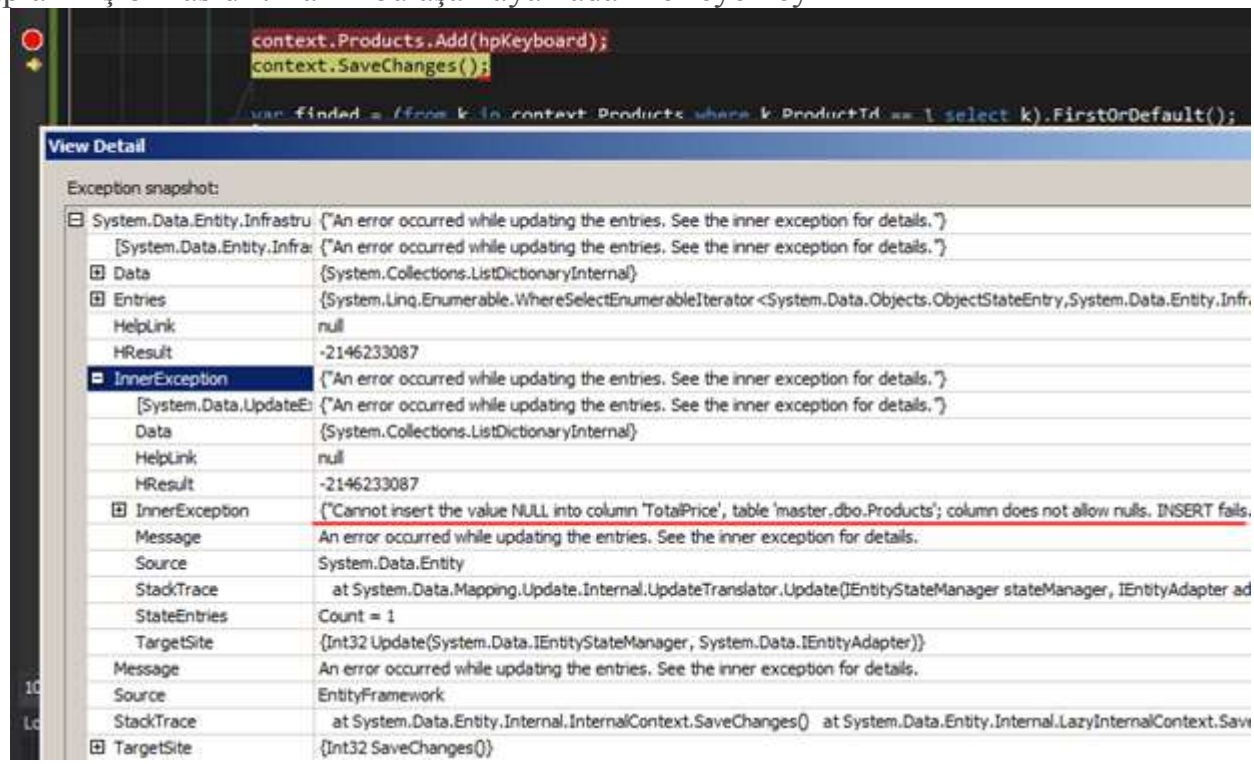
Program.cs içeriğini aşağıdaki şekilde kodlayarak senaryomuza devam edelim.

```
using System;
using System.Linq;
namespace HowTo_CalculatedFields
{
    class Program
    {
```

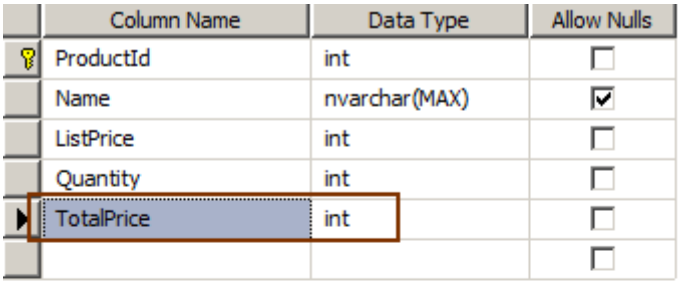
```
        static void Main(string[] args)
        {
            using (Shop context = new Shop())
            {
                Product hpKeyboard = new Product
                {
                    Name="HP 102 Tuş Kablosuz Klavye",
                    ListPrice=35,
                    Quantity=125
                };
                context.Products.Add(hpKeyboard);
                context.SaveChanges();
                var finded = (from k in context.Products
                    where k.Name == "HP 102 Tuş Kablosuz Klavye"
                    select k)
                   .FirstOrDefault();
                Console.WriteLine(finded.TotalPrice);
            }
        }
    }
}
```

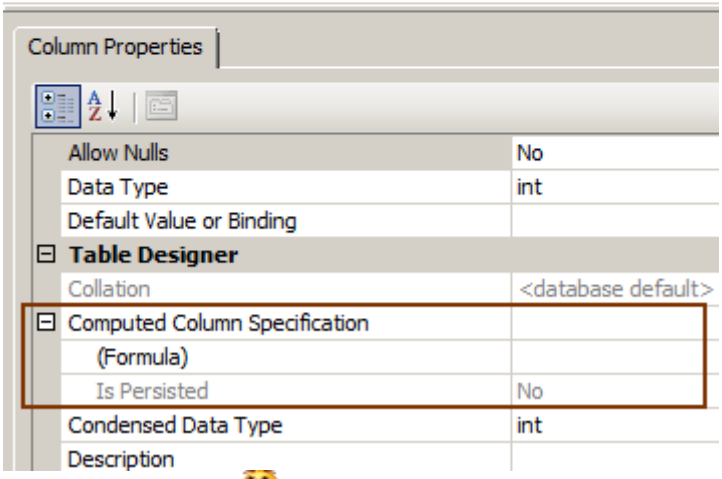
Shop context tipinin örneklenmesinin ardından bir Product nesnesi üretilmektedir. Dikkat edileceği üzere **identity** alan olarak **set** edilen **ProductId** ve **Calculated Field** olması planlanan **TotalPrice**için bir atama işlemi söz konusu değildir. Beklentimiz yeni **Product**, **context** üzerine eklendiğinde**TotalPrice** alanınında otomatik olarak

hesaplanmış olmasıdır. Lakin bu aşamaya kadar ilerleyemeyiz bile.



Dikkat edileceğiz üzere **SaveChanges** metoduna yapılan çağrı sonrasında bir çalışma zamanı istisnası*(Runtime Exception)* oluşmuştur. Söylenene göre **TotalPrice** alanı **null** değer içeremez. Aslında bu mesajı doğrudan Calculated Field ile ilişkili değildir. Yine de veritabanı tarafına baktığımızda şöyle bir durum oluştuğunu gözlemleyebiliriz; **Shop** isimli veritabanı üretilmiş,**Products** isimli tablo oluşturulmuş ve hatta içerisine **TotalPrice** isimli alan da dahil edilmiştir. Hımmm...Ne var ki **TotalPrice** kolonu **Calculate Field** haline gelmemiştir.

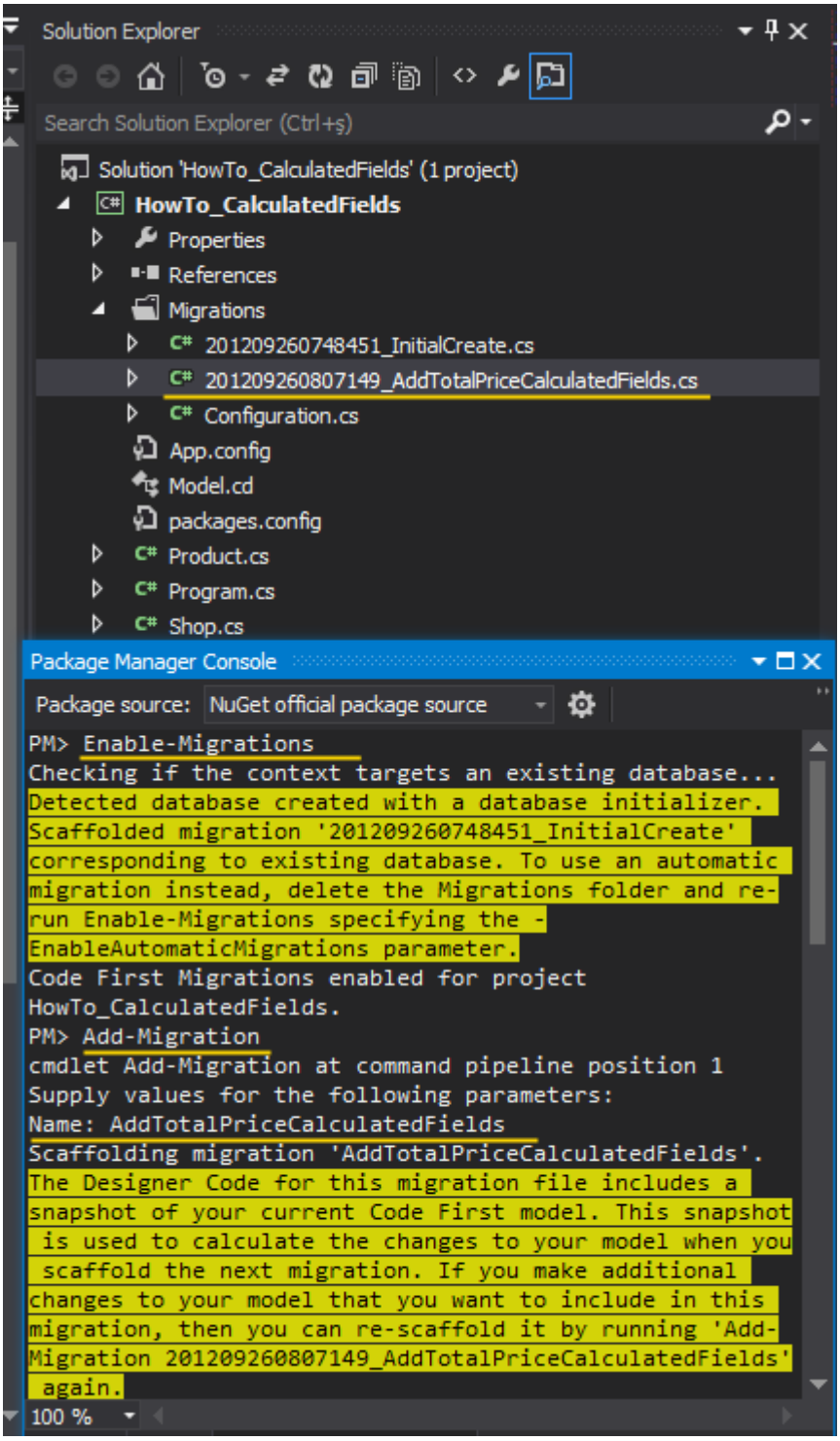




Peki ya çözüm?

Neyseki elimizin altında **migration** diye bir kabiliyet bulunmakta. Şu anda var olan veritabanı yapısını biraz değiştirip, **TotalPrice** alanı için de bir müdahalede bulunmamız gerekecek. *(Hatta**Name** alanının boyutuna bir dokunuş yaparsak hiç de fena olmaz)*

Şimdi **Migration** özelliğini etkinleştirip yeni bir **Migration** setini projeye dahil ediyor olacağız. Bunun için **Package Manager Console** penceresinden sırasıyla **Enable-Migrations** ve **Add-Migration** komutlarını çağıralım. Aşağıdaki gibi.



AddTotalPriceCalculateFields olarak adlandırdığımız **Migration** sınıfının içeriğinde yer alan **Up**ve **Down** metodlarını ise şu şekilde düzenleyebiliriz.

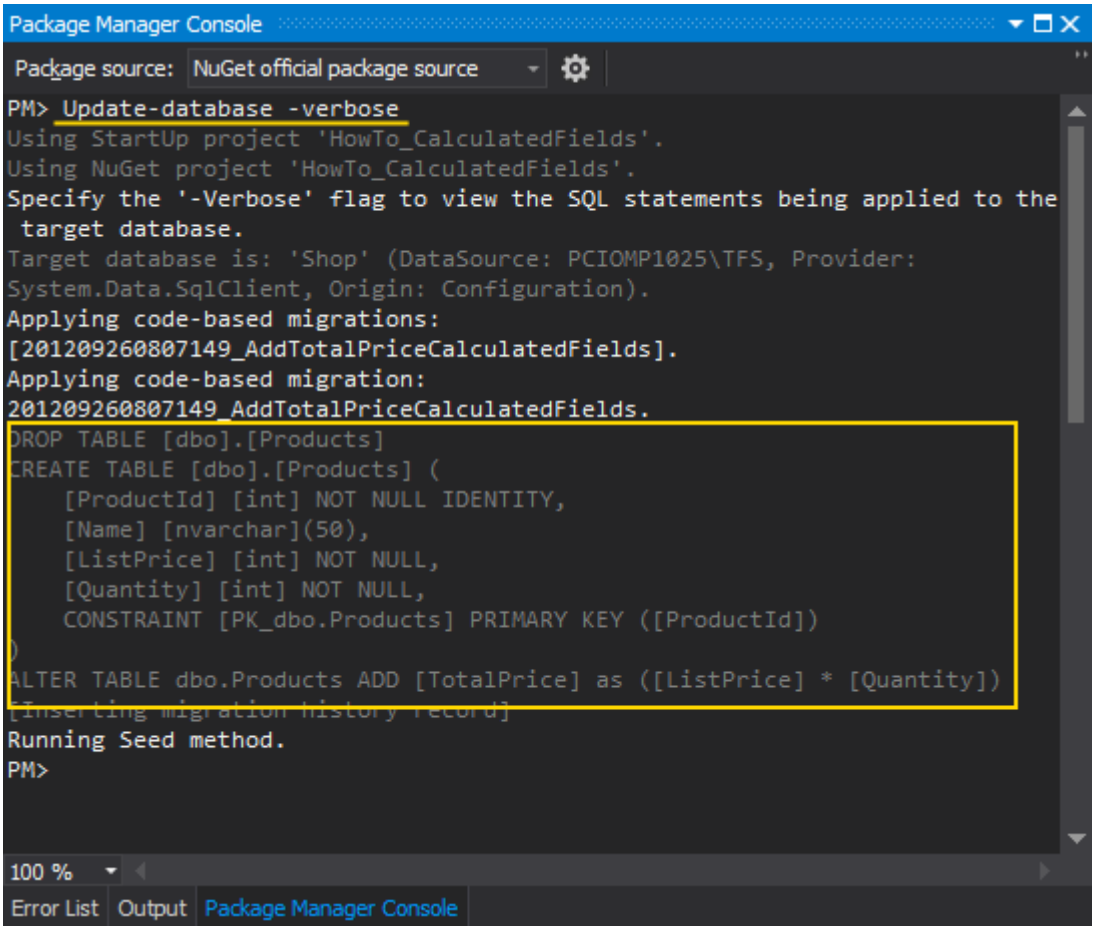
```
namespace HowTo_CalculatedFields.Migrations
{
    using System;
    using System.Data.Entity.Migrations;

    public partial class AddTotalPriceCalculatedFields
```

```
    : DbMigration
  {
    public override void Up()
    {
      DropTable("dbo.Products");
      CreateTable(
        "dbo.Products",
        c => new
        {
          ProductId = c.Int(nullable: false, identity: true),
          Name = c.String(maxLength:50),
          ListPrice = c.Int(nullable: false),
          Quantity = c.Int(nullable: false)
        })
        .PrimaryKey(t => t.ProductId);
      Sql("ALTER TABLE dbo.Products ADD [TotalPrice] as ([ListPrice] *
[Quantity])");
    }
    public override void Down()
    {
      DropTable("dbo.Products");
    }
  }
}
```

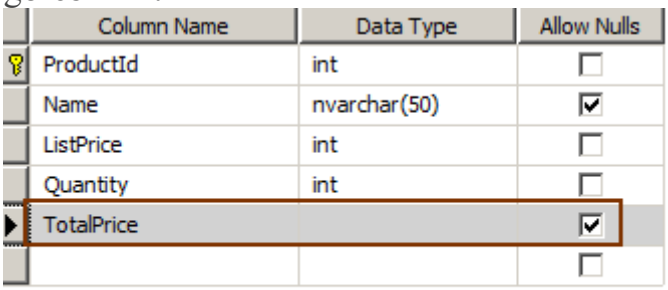
Aslında iki noktaya dokunduk. İlk olarak **TotalPrice** alaının eklenmesi için herhangibir işlem yapmadığımızı görüyoruz. İkinci olarak da bir **T-SQL** ifadesinin çalıştırılması için gerekli metod çağrısında bulunduk. **Sql** Metod çağrısına dikkat edilecek olursa **Calculated Field** için gerekli olan T-SQL ifadesini içerdiğini görebiliriz. Kısacası tablo **Create** edildikten sonra bir **Alter** işlemini bilinçli olarak uygulatıyor ve hesaplanabilir alanın bildirilmesini sağlıyoruz.

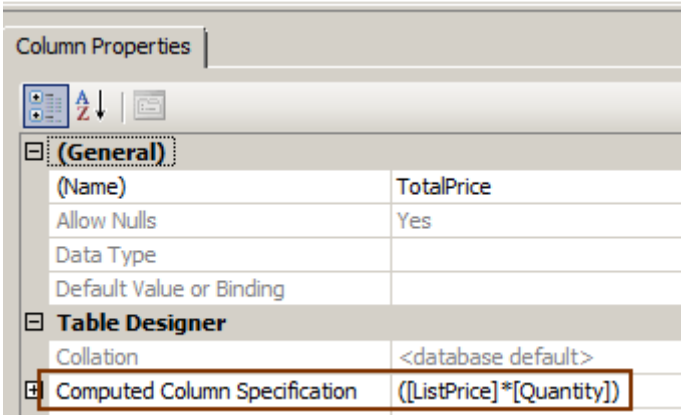
Artık veritabanını manuel olarak güncelletebiliriz. Bu güncelleme işlemi için **Package Manager Console** üzerinden Update-Database komutunu göndermemiz yeterli olacaktır 

Verbose anahtarını kullanmamızın tek sebebi, veritabanına doğru giden T-SQL ifadelerini görmektir.



Bu adımdan sonra veritabanına gidip **Products** tablosuna baktığımızda, gerçektende **TotalPrice** için bir **Calculated T-SQL** ifadesinin yazılmış olduğunu görebiliriz.





Üstelik çalışma zamanında bir ürünü çektiğimizde, miktar ve birim fiyata göre **TotalPrice** özelliğinin de veritabanından hesaplanarak getirildiğini görebiliriz.

```
context.Products.Add(hpKeyboard);
context.SaveChanges();

var finded = (from k in context.Products where
    .FirstOrDefault();
Console.WriteLine(finded.TotalPrice);
```

```
C:\Windows\system32\cmd.exe
4375
Press any key to continue . . .
```

Herşey buraya kadar iyi gitti diyebiliriz. Lakin ufak bir sorunumuz daha var. Eğer **Quantity** veya**ListPrice** değerlerinde, nesne örneği üzerinden değişiklik yaparsak, bu durumda **Calculated Field**beklediğimiz gibi bir davranış göstermeyecektir. Aşağıdaki ekran görüntüsünde yer alan kod parçasını dikkate alalım.

```
context.Products.Add(hpKeyboard);
context.SaveChanges();

var finded = (from k in context.Products
              where k.Name == "HP 102 Tuş Kablosuz Klavye"
              select k)
    .FirstOrDefault();

Console.WriteLine(finded.TotalPrice);

finded.Quantity += 10;

Console.WriteLine(finded.TotalPrice);
```

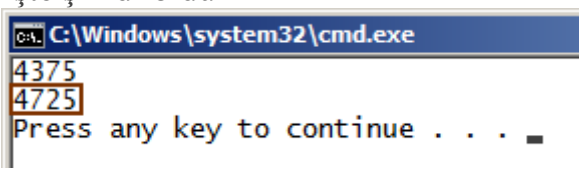
```
C:\Windows\system32\cmd.exe
4375
4375
Press any key to continue . . .
```

Senaryoda ürün eklendikten sonraki durumda **Calculated Field** alanının hesaplanarak geldiği görülmektedir. Yani ilk eklemeden sonra gerçekleştirilen **LINQ** ifadesine göre TotalPrice için SQL tarafındaki hesaplama devreye girmiştir. Ancak bellek üzerinde kalan **Product** nesne örneğinin**Quantity** veya **ListPrice** alanlarında bir değişiklik yapıldığında, bu çok doğal olarak **TotalPrice'** a yansımayacaktır.
Bu durum çok doğal olarak veritabanına gidilmeden yapılan nesne örneği bazlı özellik güncellemelerinde doğru verinin gösterilemeyeceği anlamına gelir ki bu da pek istemediğimiz bir durumdur. Soruna **Product** tipi içerisindeki **TotalPrice** özelliği üzerinden müdahalede bulunarak çözüm getirebiliriz. Aynen aşağıdaki kod parçasında görüldüğü gibi;

```
using System.ComponentModel.DataAnnotations.Schema;
namespace HowTo_CalculatedFields
{
   public class Product
   {
      [DatabaseGenerated(DatabaseGeneratedOption.Identity)]
      public int ProductId { get; set; }
```

```
        public string Name { get; set; }
        public int ListPrice { get; set; }
        public int Quantity { get; set; }
        [DatabaseGenerated(DatabaseGeneratedOption.Computed)]
        public int TotalPrice
        {
            get
            {
                return ListPrice * Quantity;
            }
            private set // get bloğunu açtığımız için aşağıdaki bloğu boş olsa bile açmak mecburiyetindeyiz.
            {
            }
        }
    }
}
```

İşte şimdi oldu



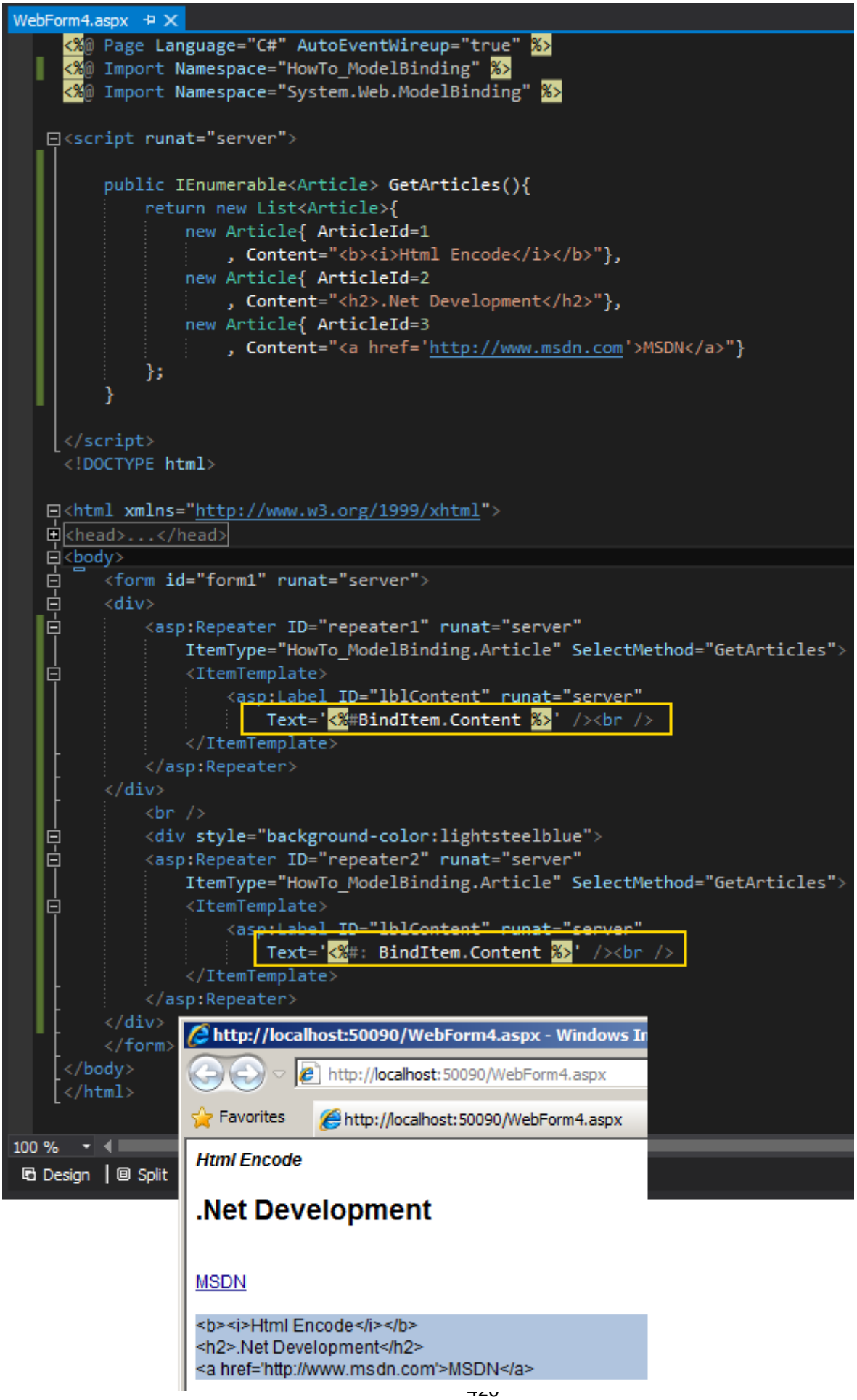
Böylece geldik bir yazımızın daha sonuna. Bu makalemizde **Code-First** yaklaşımının kullanıldığı senaryolarda, biraz da veritabanı tarafına özgü olan **Calculated Field'** ların nasıl etkin hale getirilebileceğini bir kaç küçük hile ile incelemeye çalıştık. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HowTo_CalculatedFields.zip (2,60 mb)

Tek Fotoluk İpucu 78 - Asp.Net 4.5 ile HtmlEncode

Pazartesi, 4 Şubat 2013 02:07 by bsenyurt

Merhaba Arkadaşlar,
Bazı durumlarda **Asp.Net** sayfasının çıktısına basacağımız içeriğin **HTML** formatlı elementlerinin**Text** tabanlı görünümleri olmasını isteriz. Örneğin **** takısının, uygulandığı metni **bold** olarak göstermesini istemeyiz. Bunun yerine **yazı** şeklinde düz metin olarak gösterilmesini arzu ederiz*(Hatta bazı blogların yorum kısımlarında, yorumda kullanılabilecek HTML Tag' leri ifade edilir. Ama metin olarak basılmışlardır)* Bunun için **Asp.Net 4.5** tarafında işimizi oldukça kolaylaştıracak bir özellik yer almakta. **İki nokta üst üste işaretini** kullanmamız **HTML** içeriğinin metinsel olarak kullanılmasında yeterli oluyor. Nerede mi? Özellikle **Veri bağlama*(Data Binding)*** noktalarında Örneğin,


WebForm4.aspx
<%@ Page Language="C#" AutoEventWireup="true" %>
<%@ Import Namespace="HowTo_ModelBinding" %>
<%@ Import Namespace="System.Web.ModelBinding" %>
<script runat="server">
public IEnumerable<Article> GetArticles(){
return new List<Article>{
new Article{ ArticleId=1
, Content="Html Encode"},
new Article{ ArticleId=2
, Content="<h2>.Net Development</h2>"},
new Article{ ArticleId=3
, Content="MSDN"}
};
}
</script>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head>...</head>
<body>
<form id="form1" runat="server">

<asp:Repeater ID="repeater1" runat="server"
ItemType="HowTo_ModelBinding.Article" SelectMethod="GetArticles">
<ItemTemplate>
<asp:Label ID="lblContent" runat="server"
Text='<%#BindItem.Content %>' />
</ItemTemplate>
</asp:Repeater>

<asp:Repeater ID="repeater2" runat="server"
ItemType="HowTo_ModelBinding.Article" SelectMethod="GetArticles">
<ItemTemplate>
<asp:Label ID="lblContent" runat="server"
Text='<%#: BindItem.Content %>' />
</ItemTemplate>
</asp:Repeater>

</form>
</body>
</html>
100 %
Design
Split
http://localhost:50090/WebForm4.aspx - Windows In
http://localhost:50090/WebForm4.aspx
Favorites
http://localhost:50090/WebForm4.aspx
Html Encode
.Net Development
MSDN
Html Encode
<h2>.Net Development</h2>
MSDN

Bir başka ip ucunda görüşmek dileğiyle 

Workflow Foundation, Oracle, WCF ve TransactionScope

Cuma, 1 Şubat 2013 07:36 by bsenyurt

[Örnek Visual Studio 2010, .Net Framework 4.0 tabanlıdır]

Merhaba Arkadaşlar,

Yandaki fotoğrafta görülen buluşa baktığınızda aslında gerçekten bu pilotun o koca pervaneler ile uçup uçamayacağına pek kanaat getiremiyoruz öyle değil mi? Sonuçta en azından kağıt üstünde ve teorik olarak da bu tip bir uçuş aracının çalışacağının ispat edilmesi ve sonrasında pratikteki kullanımı için teste çıkılması beklenir*(Tabi buna cesaret edecek de bir pilotun olması gerekir)*

Bir dostumuzun söylediği üzere **"tasarlanan her uçak uçmuş ama her yazılım çalışmamıştır"** 😜

Bir başka deyişle yazılım tarafında bir şeylerin ispatını yaparken bir uçağı uçuracakmış gibi düşünerek hareket etmeyiz genelde. İstesek de edemiyoruz sanırım. Yine de elimizden geldiğince titiz çalışmamız da yarar var. Öyleyse gelelim bu günün konusuna.

Geçtiğimiz günlerde **Workflow Foundation** tabanlı bir uygulama içerisinde **Transaction Scope**kullanımına ihtiyacım oldu. **Transaction'** a dahil olan işlemler **Oracle** tabloları üzerinde gerçekleştirilecekti. Senaryoyu zorlaştıran noktalardan birisi ise, akış içerisinde harici bir **WCF** servis çağrısının yapılacak olmasıydı. Nitekim söz konusu **WCF** servisi içerisindeki operasyonda da, yine**Oracle** veritabanı üzerinde yapılması planlanan **Transactional** bir işlem söz konusuydu.

Dolayısıyla **Oracle'** ın ve **WCF** servisinin işin içerisinde yer aldığı bir **Workflow** senaryosunda,**TransactionScope** bileşeninin işe yarayıp yaramadığının araştırılması ve gerekli ispatların yapılması gerekmekteydi. Bir başka deyişle bir **POC(Proof Of Concept)** çalışması ile karşı karşıyaydık. Ben de**Visual Studio 2010** un başına oturdum ve yola koyuldum.

Senaryoda kendi oluşturduğum iki **Oracle** tablosu söz konusu idi. Bu tabloların içerdiği alanların çok fazla önemi yoktu aslında. **Account** ve **Branch** olarak adlandırdığım tablolar üzerinde iki basit**Insert** işlemi gerçekleştirecektim. Buna ek olarak bir de **WCF** servisi söz konusu olmalıydı. İlgili**WCF** servisi de yine benzer bir **Insert** işlemi gerçekleştirilmek üzere tasarlanmalıydı.

Teorik olarak **Workflow Activity'** si içerisinde kullanılacak **TransactionScope Component'** i servis tarafına **Transaction'** ı aktarabilmeliydi. Tam bu noktada **Distributed TransactionCoordinator** servisinin de işlevselliği söz konusuydu. Elbette servis tarafının da, **Transaction**akışına izin verecek şekilde tesis edilmesi şarttı.

Bu noktada özellikle Binding tipinin Transaction Flow' a destek veriyor olması önemlidir. Nitekim varsayılan bağlayıcı tip olan BasicHttpBinding, istemciden gelen Transaction akışına izin vermez. Ama örneğin wsHttpBinding buna olanak tanır.
Dilerseniz işe ilk olarak bu **WCF** servisini geliştirerek başlayalım. Servisimize ait basit kod içeriği aşağıdaki gibidir.

```
using System.ServiceModel;
namespace CRUDer
{
   [ServiceContract]
   public interface IAccountService
   {
      [OperationContract]
      [TransactionFlow(TransactionFlowOption.Mandatory)]
      int DoWork(int AccountId,string Name,string Surname);
   }
}
```

Sözleşme tipi içerisindeki en önemli nokta **TransactionFlow** niteliğinin kullanılması ve **Mandatory**değerinin verilmesidir. Bu nitelik **Allowed**, **NotAllowed** şeklinde iki farklı değer daha alabilir.**Mandatory**, istemci tarafında bir **Transaction Scope** başlatılma zorunluluğunu ifade etmektedir. Nitekim operasyon içerisindeki işlemin bir **Transaction Scope'** a dahil olması şarttır.

```
using System.Configuration;
using System.ServiceModel;
using System.Transactions;
using Oracle.DataAccess.Client;
namespace CRUDer
{
   //[ServiceBehavior(TransactionIsolationLevel=IsolationLevel.Serializable)]
   public class AccountService
      : IAccountService
   {
     [OperationBehavior(TransactionScopeRequired=true,
TransactionAutoComplete=true)]
      public int DoWork(int AccountId, string Name, string Surname)
      {
         int result = -1;
         var currentTransaction=Transaction.Current;
         using (OracleConnection conn = new
OracleConnection(ConfigurationManager.ConnectionStrings["ConStr"].ConnectionString)
)
         {
```

```
            using (OracleCommand command = new OracleCommand("INSERT INTO
ACCOUNT (ACCOUNTID,NAME,SURNAME) VALUES
(:pACCOUNTID,:pNAME,:pSURNAME)", conn))
            {
               command.Parameters.Add(":pACCOUNTID",AccountId);
               command.Parameters.Add(":pNAME", Name);
               command.Parameters.Add(":pSURNAME", Surname);
               conn.Open();
               result = command.ExecuteNonQuery();
            }
         }
         return result;
      }
   }
}
```

Servisin asıl iş yapan kodlarında yine dikkat çekici ve önemli olan nokta **OperationBehavior** niteliği ile atanan **özellik(Property)** değerleridir. Operasyonun bir **TransactionScope** gerektirdiği ve ayrıca **unhandled exception** oluşması halinde operasyona ait **transaction** örneğinin otomatik olarak tamamlanıp tamamlanmayacağı belirtilmektedir.

Servis tarafında System.Transactions assembly' ının referans edilmesiunutulmamalıdır. Ayrıca hem servis hem de istemci tarafı Oracle fonksiyonellikleri için ODP.Net' e(Oracle DataProvider for .Net) ait assembly' ları kullanmaktadır. Dolayısıyla Oracle.DataAccess assembly' ının uygun olan versiyonunun referans edilmelidir.

Servis tarafındaki **config** dosyasının içeriği de oldukça önemlidir. Daha önceden de belirttiğimiz üzere **Binding** tipi, **Transaction** akışına izin verir nitelikte olmalıdır. Ancak buna ek olarak**Transaction** akışına izin verileceğinin de konfigurasyon dosyası içerisinde belirtilmesi şarttır. Çünkü varsayılan olarak **Transaction** akışı etkin değildir.

```
<?xml version="1.0"?>
<configuration>
   <connectionStrings>
      <add name="ConStr" connectionString="User Id=bir kullanıcı adı;Password=bir
şifre;Data Source=oracle veri kaynağı" providerName="Oracle.DataAccess.Client"/>
   </connectionStrings>
 <system.serviceModel>
   <services>
     <service name="CRUDer.AccountService">
       <endpoint address="http://localhost:35662/AccountService.svc"
        binding="wsHttpBinding" bindingConfiguration="wsB" contract="CRUDer.IAcc
ountService" />
```

```
    </service>
  </services>
    <bindings>
      <wsHttpBinding>
        <binding transactionFlow="true" name="wsB"/>
      </wsHttpBinding>
    </bindings>
  <behaviors>
    <serviceBehaviors>
      <behavior name="">
        <serviceMetadata httpGetEnabled="true"/>
        <serviceDebug includeExceptionDetailInFaults="true"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <serviceHostingEnvironment multipleSiteBindingsEnabled="false"/>
 </system.serviceModel>
 <system.web>
  <compilation debug="true"/>
 </system.web>
</configuration>
```

Dikkat edileceği üzere **wsHttpBinding** elementi içerisinde yer alan **binding** alt elementinin**transactionFlow** niteliğinin değeri **true** olarak set edilmiştir. Servis tarafında yapılması gerekenler aslında bu kadardır. Öyleyse **Workflow** tasarımına başlayabiliriz. **Workflow Console Application**olarak tasarlanan uygulamamızda çalışma zamanını izleyebilmek için basit de bir **Tracker** tipine yer verilmektedir. Bu tipin içeriği aşağıdaki gibidir.

```
using System;
using System.Activities.Tracking;
using System.Collections.Generic;
using System.Text;
namespace HowToTransaction
{
  public class CustomTracker
    : TrackingParticipant
  {
    protected override void Track(TrackingRecord record, TimeSpan timeout)
    {
      WorkflowInstanceRecord instanceRecord = record as
WorkflowInstanceRecord;
```

```
            if (instanceRecord != null)
               Console.WriteLine(instanceRecord.ToString());
            ActivityStateRecord activity = record as ActivityStateRecord;
            if (activity != null)
            {
               var variables = activity.Variables;
               StringBuilder builder = new StringBuilder();
               if (variables.Count > 0)
               {
                  builder.AppendLine(" Variables:");
                  foreach (KeyValuePair<string, object> variable in variables)
                  {
                     builder.AppendLine(String.Format(" {0} Value: [{1}]", variable.Key,
variable.Value));
                  }
               }
               Console.WriteLine(
                  String.Format(" Activity: {0} State: {1} {2}",
                  activity.Activity.Name
                  , activity.State
                  , builder.ToString())
                  );
            }
         }
      }
   }
}
```

Tracker sınıfı içerisinde hem **Workflow Actvitiy'** si hem de içeride yürümekte olan **Activity**örneklerinin çalışmalarının izlenmesi işlemi gerçekleştirilmektedir. Bu, **TrackingRecord** tipinden olan parametrenin **WorkflowInstanceRecord** ve **ActivityStateRecord** nesne örneklerine dönüştürülmesi ile gerçekleştirilmektedir. **Tracker** tipi dışında **Account** ve **Branch** tablolarına**insert** işlemi yapan iki **Code Activiy** bileşeni de bulunmaktadır. Söz konusu Activity bileşenlerinin kodları aşağıdaki gibidir.

Account insert işlemini üstlenen Activity,

```
namespace HowToTransaction
{
   using System.Activities;
   using System.Configuration;
   using Oracle.DataAccess.Client;
   public class InsertAccountActivity
      :CodeActivity
```

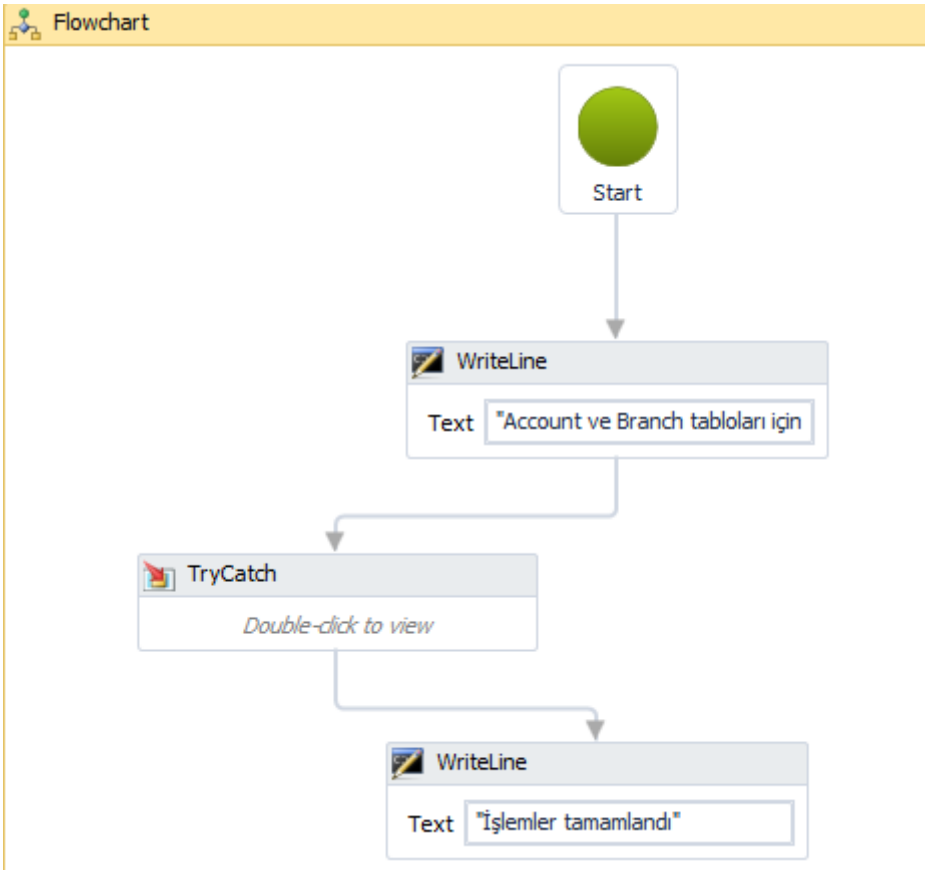
```
   {
      public InArgument<int> AccountId { get; set; }
      public InArgument<string> Name { get; set; }
      public InArgument<string> Surname { get; set; }
      public OutArgument<int> ExecuteNonQueryResult { get; set; }
      protected override void Execute(CodeActivityContext context)
      {
         using (OracleConnection conn = new
OracleConnection(ConfigurationManager.ConnectionStrings["ConStr"].ConnectionString)
)
         {
            using (OracleCommand command = new OracleCommand("INSERT INTO
ACCOUNT (ACCOUNTID,NAME,SURNAME) VALUES
(:pACCOUNTID,:pNAME,:pSURNAME)", conn))
            {
               command.Parameters.Add(":pACCOUNTID",context.GetValue(AccountId));
               command.Parameters.Add(":pNAME",context.GetValue(Name));
               command.Parameters.Add(":pSURNAME",context.GetValue(Surname));
               conn.Open();
               int result=command.ExecuteNonQuery();
               context.SetValue(ExecuteNonQueryResult, result);
            }
         }
      }
   }
}
```

ve Branch insert işlemini üstlenen Code Activity

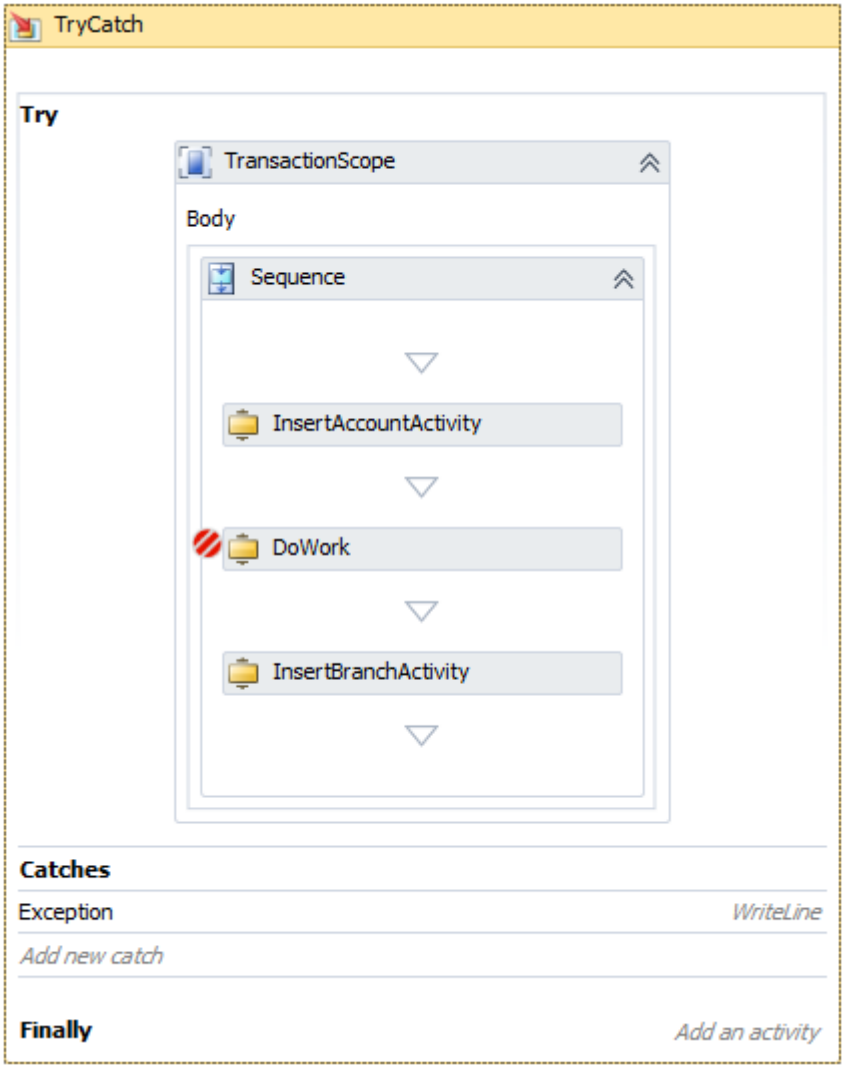
```
namespace HowToTransaction
{
   using System.Activities;
   using System.Configuration;
   using Oracle.DataAccess.Client;
   using System;
   public class InsertBranchActivity
      :CodeActivity
   {
      public InArgument<int> BranchId { get; set; }
      public InArgument<string> Title { get; set; }
      public InArgument<int> Code { get; set; }
      public OutArgument<int> ExecuteNonQueryResult { get; set; }
```

```
        protected override void Execute(CodeActivityContext context)
        {
            using (OracleConnection conn = new
OracleConnection(ConfigurationManager.ConnectionStrings["ConStr"].ConnectionString)
)
            {
                using (OracleCommand command = new OracleCommand("INSERT INTO
BRANCH (BRANCHID,TITLE,CODE) VALUES (:pBRANCHD,:pTITLE,:pCODE)",
conn))
                {
                    command.Parameters.Add(":pBRANCHID",context.GetValue(BranchId));
                    command.Parameters.Add(":pTITLE",context.GetValue(Title));
                    command.Parameters.Add(":pCODE",context.GetValue(Code));
                    conn.Open();
                    int result = command.ExecuteNonQuery();
                    context.SetValue(ExecuteNonQueryResult, result);
                   //throw new Exception("Some error");
                }
            }
        }
    }
}
```

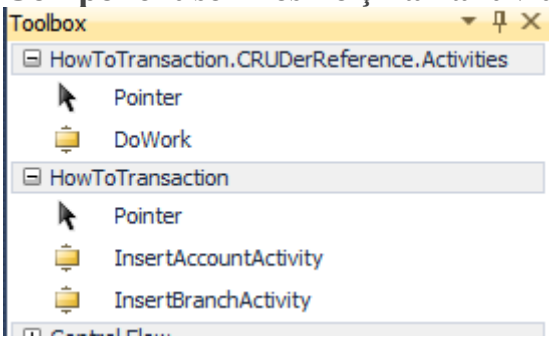
Burada yorum satırı olarak bırakılmış kısım daha sonradan yapılacak testler sırasında açılacak ve**Transaction Scope'** un çalışması izlenecektir.
Gelelim test amaçlı kullanacağımız **Workflow Activity** içeriğine.



FlowChart şeklinde tasarladığımız akışın içerisindeki en kritik yer **TryCatch** bileşenin içerisidir.



Try bloğundan **TransactionScope** bileşeni altında
sırasıyla **Account Insert** işlemi, **DoWork** ile WCF servis çağrısı ve
tekrar **Branch Insert** işlemi gerçekleştirilmektedir.
WCF servisinin Workflow uygulamasına Add Service Reference ile eklenmesi sonrası Component sekmesine çıkan aktivite bileşeni kullanılmaktadır*(DoWork bileşeni)*



Workflow un **XAML(eXtensibleApplicationMarkupLanguage)** içeriği aşağıdaki gibidir. Burada, kullanılan **variable'** lar daha net bir şekilde görülebilmektedir.
<Activity mc:Ignorable="sads sap" x:Class="HowToTransaction.Workflow1"
sap:VirtualizedContainerService.HintSize="654,676"

```
mva:VisualBasic.Settings="Assembly references and imported namespaces for internal
implementation"
xmlns="http://schemas.microsoft.com/netfx/2009/xaml/activities"
xmlns:av="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:hc="clr-namespace:HowToTransaction.CRUDerReference"
xmlns:local="clr-namespace:HowToTransaction"
xmlns:local1="clr-namespace:HowToTransaction.CRUDerReference.Activities"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:mv="clr-namespace:Microsoft.VisualBasic;assembly=System"
xmlns:mva="clr-
namespace:Microsoft.VisualBasic.Activities;assembly=System.Activities"
xmlns:p="http://schemas.microsoft.com/netfx/2009/xaml/servicemodel"
xmlns:s="clr-namespace:System;assembly=mscorlib"
xmlns:s1="clr-namespace:System;assembly=System"
xmlns:s2="clr-namespace:System;assembly=System.Xml"
xmlns:s3="clr-namespace:System;assembly=System.Core"
xmlns:s4="clr-namespace:System;assembly=System.ServiceModel"
xmlns:sa="clr-namespace:System.Activities;assembly=System.Activities"
xmlns:sad="clr-namespace:System.Activities.Debugger;assembly=System.Activities"
xmlns:sads="http://schemas.microsoft.com/netfx/2010/xaml/activities/debugger"
xmlns:sap="http://schemas.microsoft.com/netfx/2009/xaml/activities/presentation"
xmlns:scg="clr-namespace:System.Collections.Generic;assembly=System"
xmlns:scg1="clr-
namespace:System.Collections.Generic;assembly=System.ServiceModel"
xmlns:scg2="clr-namespace:System.Collections.Generic;assembly=System.Core"
xmlns:scg3="clr-namespace:System.Collections.Generic;assembly=mscorlib"
xmlns:sd="clr-namespace:System.Data;assembly=System.Data"
xmlns:sl="clr-namespace:System.Linq;assembly=System.Core"
xmlns:st="clr-namespace:System.Text;assembly=mscorlib"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Flowchart
sad:XamlDebuggerXmlReader.FileName="d:\users\bsenyurt\documents\visual studio
2010\Projects\Windows Phone\HowToTransaction\HowToTransaction\Workflow1.xaml"
sap:VirtualizedContainerService.HintSize="614,636">
    <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">False</x:Boolean>
        <av:Point x:Key="ShapeLocation">270,2.5</av:Point>
        <av:Size x:Key="ShapeSize">60,75</av:Size>
        <av:PointCollection x:Key="ConnectorLocation">300,77.5
300,139.5</av:PointCollection>
```

```
    </scg3:Dictionary>
   </sap:WorkflowViewStateService.ViewState>
   <Flowchart.StartNode>
    <FlowStep x:Name="__ReferenceID0">
     <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">
       <av:Point x:Key="ShapeLocation">194.5,139.5</av:Point>
       <av:Size x:Key="ShapeSize">211,59</av:Size>
       <av:PointCollection x:Key="ConnectorLocation">300,198.5 300,228.5 160,228.5
160,245.5</av:PointCollection>
      </scg3:Dictionary>
     </sap:WorkflowViewStateService.ViewState>
     <WriteLine sap:VirtualizedContainerService.HintSize="211,59" Text="Account ve
Branch tabloları için Insert işlemleri başlıyor">
      <sap:WorkflowViewStateService.ViewState>
       <scg3:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">True</x:Boolean>
       </scg3:Dictionary>
      </sap:WorkflowViewStateService.ViewState>
     </WriteLine>
     <FlowStep.Next>
      <FlowStep x:Name="__ReferenceID2">
       <sap:WorkflowViewStateService.ViewState>
        <scg3:Dictionary x:TypeArguments="x:String, x:Object">
         <av:Point x:Key="ShapeLocation">60,245.5</av:Point>
         <av:Size x:Key="ShapeSize">200,49</av:Size>
         <av:PointCollection x:Key="ConnectorLocation">160,294.5 160,324.5
290,324.5 290,340.5</av:PointCollection>
        </scg3:Dictionary>
       </sap:WorkflowViewStateService.ViewState>
       <TryCatch sap:VirtualizedContainerService.HintSize="418,529">
        <TryCatch.Variables>
         <Variable x:TypeArguments="x:Int32" Name="ServiceCallResult" />
        </TryCatch.Variables>
        <sap:WorkflowViewStateService.ViewState>
         <scg3:Dictionary x:TypeArguments="x:String, x:Object">
          <x:Boolean x:Key="IsExpanded">True</x:Boolean>
         </scg3:Dictionary>
        </sap:WorkflowViewStateService.ViewState>
        <TryCatch.Try>
         <TransactionScope AbortInstanceOnTransactionFailure="False"
```

```
sap:VirtualizedContainerService.HintSize="258,351">
            <sap:WorkflowViewStateService.ViewState>
              <scg3:Dictionary x:TypeArguments="x:String, x:Object">
                <x:Boolean x:Key="IsExpanded">True</x:Boolean>
              </scg3:Dictionary>
            </sap:WorkflowViewStateService.ViewState>
            <Sequence sap:VirtualizedContainerService.HintSize="222,270">
              <sap:WorkflowViewStateService.ViewState>
                <scg3:Dictionary x:TypeArguments="x:String, x:Object">
                  <x:Boolean x:Key="IsExpanded">True</x:Boolean>
                </scg3:Dictionary>
              </sap:WorkflowViewStateService.ViewState>
              <local:InsertAccountActivity ExecuteNonQueryResult="{x:Null}"
AccountId="2" sap:VirtualizedContainerService.HintSize="200,22"
Name="Delinin" Surname="Biri">
                <sap:WorkflowViewStateService.ViewState>
                  <scg3:Dictionary x:TypeArguments="x:String, x:Object">
                    <x:Boolean x:Key="IsExpanded">True</x:Boolean>
                  </scg3:Dictionary>
                </sap:WorkflowViewStateService.ViewState>
              </local:InsertAccountActivity>
              <local1:DoWork AccountId="99" DoWorkResult="[ServiceCallResult]"
EndpointConfigurationName="WSHttpBinding_IAccountService"
sap:VirtualizedContainerService.HintSize="200,22" Name="Maykıl"
mva:VisualBasic.Settings="Assembly references and imported namespaces serialized
as XML namespaces" Surname="Cordın" />
              <local:InsertBranchActivity ExecuteNonQueryResult="{x:Null}"
BranchId="34" Code="4" sap:VirtualizedContainerService.HintSize="200,22"
Title="germany">
                <sap:WorkflowViewStateService.ViewState>
                  <scg3:Dictionary x:TypeArguments="x:String, x:Object">
                    <x:Boolean x:Key="IsExpanded">True</x:Boolean>
                  </scg3:Dictionary>
                </sap:WorkflowViewStateService.ViewState>
              </local:InsertBranchActivity>
            </Sequence>
          </TransactionScope>
        </TryCatch.Try>
        <TryCatch.Catches>
          <Catch x:TypeArguments="s:Exception"
sap:VirtualizedContainerService.HintSize="404,20">
```

```
            <sap:WorkflowViewStateService.ViewState>
              <scg3:Dictionary x:TypeArguments="x:String, x:Object">
                <x:Boolean x:Key="IsExpanded">False</x:Boolean>
                <x:Boolean x:Key="IsPinned">False</x:Boolean>
              </scg3:Dictionary>
            </sap:WorkflowViewStateService.ViewState>
            <ActivityAction x:TypeArguments="s:Exception">
              <ActivityAction.Argument>
                <DelegateInArgument x:TypeArguments="s:Exception" Name="exception"
/>
              </ActivityAction.Argument>
              <WriteLine sap:VirtualizedContainerService.HintSize="211,59"
Text="[exception.Message]" />
            </ActivityAction>
          </Catch>
        </TryCatch.Catches>
      </TryCatch>
      <FlowStep.Next>
        <FlowStep x:Name="__ReferenceID1">
          <sap:WorkflowViewStateService.ViewState>
            <scg3:Dictionary x:TypeArguments="x:String, x:Object">
              <av:Point x:Key="ShapeLocation">184.5,340.5</av:Point>
              <av:Size x:Key="ShapeSize">211,59</av:Size>
              <av:PointCollection x:Key="ConnectorLocation">310,210.5
310,309.5</av:PointCollection>
            </scg3:Dictionary>
          </sap:WorkflowViewStateService.ViewState>
          <WriteLine sap:VirtualizedContainerService.HintSize="211,59" Text="İşlemler
tamamlandı">
            <sap:WorkflowViewStateService.ViewState>
              <scg3:Dictionary x:TypeArguments="x:String, x:Object">
                <x:Boolean x:Key="IsExpanded">True</x:Boolean>
              </scg3:Dictionary>
            </sap:WorkflowViewStateService.ViewState>
          </WriteLine>
        </FlowStep>
      </FlowStep.Next>
    </FlowStep>
  </FlowStep.Next>
</FlowStep>
</Flowchart.StartNode>
```

```
    <x:Reference>__ReferenceID0</x:Reference>
    <x:Reference>__ReferenceID1</x:Reference>
    <x:Reference>__ReferenceID2</x:Reference>
  </Flowchart>
</Activity>
```

Workflow Application tarafındaki en önemli ayarlardan birisi de **config** dosyasında yer almaktadır.

```
<?xml version="1.0"?>
<configuration>
    <startup>
      <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
    </startup>
    <connectionStrings>
       <add name="ConStr" connectionString="User Id=bir kullanıcı adı;Password=bir
şifre;Data Source=bir veri kaynağı" providerName="Oracle.DataAccess.Client"/>
    </connectionStrings>
    <system.serviceModel>
       <bindings>
          <wsHttpBinding>
             <binding name="WSHttpBinding_IAccountService"
transactionFlow="true" />
          </wsHttpBinding>
       </bindings>
       <client>
          <endpoint address="http://localhost:35662/AccountService.svc"
             binding="wsHttpBinding" bindingConfiguration="WSHttpBinding_IAccount
Service"
             contract="IAccountService" name="WSHttpBinding_IAccountService">
             <identity>
                <userPrincipalName value="bsenyurt@domain.bankanet.com.tr" />
             </identity>
          </endpoint>
       </client>
    </system.serviceModel>
</configuration>
```

İstemci tarafındaki **Binding** ayarlarında da **transactionFlow** niteliğinin mutlak suretle **true** olması gerekmektedir. Gelelim **Main** metodu içerisindeki kodlarımıza.

```
using System.Activities;
using System.Threading;
namespace HowToTransaction
{
```

```
class Program
{
   static void Main(string[] args)
   {
      var activity = new Workflow1();
      AutoResetEvent arEvent = new AutoResetEvent(false);
      WorkflowApplication wfApp = new WorkflowApplication(activity);
      wfApp.Extensions.Add(new CustomTracker());
      wfApp.Completed = (o) =>
      {
         arEvent.Set();
      };
       wfApp.Run();
      arEvent.WaitOne();
   }
}
}
```

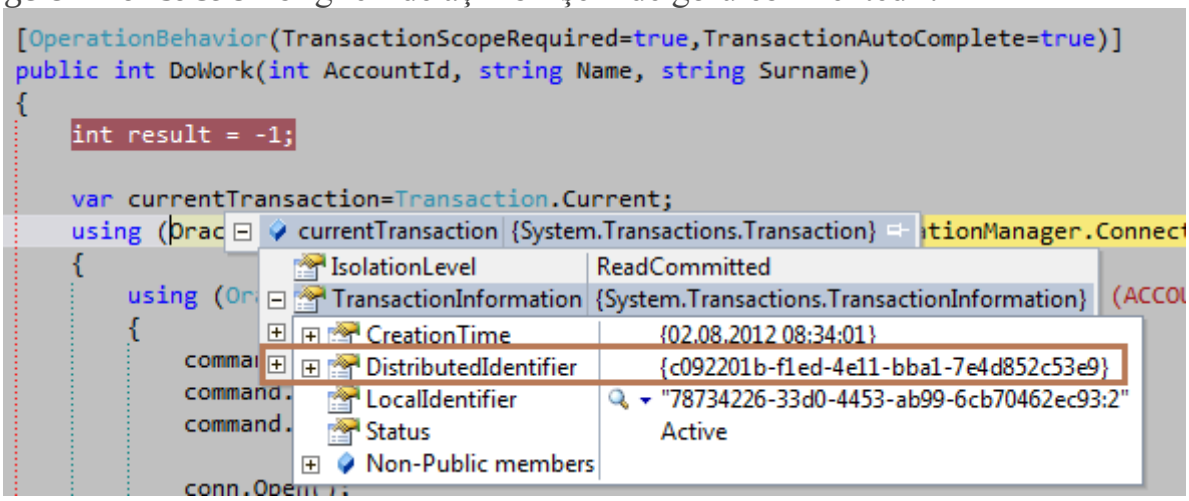
WorkflowApplication tipinden yararlanılarak **Workflow1** örneğinin başlatılması işlemi gerçekleştirilmektedir. **Tracking** işlemi için yazılmış olan **CustomTracker** sınıfının da, bir**Extension** olarak **WorkflowApplication** örneğine bildirilmesi gerekir.
Örneği bu hali ile çalıştırdığımda aşağıdaki sonuçları elde ettiğimi gördüm.


C:\Windows\system32\cmd.exe
WorkflowInstanceRecord { InstanceId = 8f82d24a-4bb5-4a3b-8e52-f0efcfc80950, Reco
rdNumber = 0, EventTime = 02.08.2012 08:27:35, ActivityDefinitionId = Workflow1,
State = Started }
Activity: Workflow1 State: Executing
Activity: Flowchart State: Executing
Account ve Branch tabloları için Insert işlemleri başlıyor
Activity: WriteLine State: Executing
Activity: WriteLine State: Closed
Activity: TryCatch State: Executing Variables:
ServiceCallResult Value: [0]
Activity: TransactionScope State: Executing
Activity: Sequence State: Executing
Activity: InsertAccountActivity State: Executing
Activity: InsertAccountActivity State: Closed
Activity: DoWork State: Executing
Activity: Sequence State: Executing Variables:
tempResult Value: []
_577ebd57_7f71_4c52_b09a_bb88cb213538_1 Value: [System.ServiceModel.Activities.
CorrelationHandle]
Activity: Send State: Executing
Activity: MessagingNoPersistScope State: Executing
Activity: Sequence State: Executing Variables:
RequestMessage Value: []
Activity: ToRequest State: Executing
Activity: ToRequest State: Closed
Activity: InternalSendMessage State: Executing
Activity: OpenChannelFactory State: Executing
WorkflowInstanceRecord { InstanceId = 8f82d24a-4bb5-4a3b-8e52-f0efcfc80950, Reco
rdNumber = 34, EventTime = 02.08.2012 08:27:51, ActivityDefinitionId = Workflow1
, State = Idle }
Activity: OpenChannelFactory State: Closed
Activity: OpenChannelAndSendMessage State: Executing
WorkflowInstanceRecord { InstanceId = 8f82d24a-4bb5-4a3b-8e52-f0efcfc80950, Reco
rdNumber = 38, EventTime = 02.08.2012 08:27:52, ActivityDefinitionId = Workflow1
, State = Idle }
Activity: OpenChannelAndSendMessage State: Closed
Activity: InternalSendMessage State: Closed
Activity: Sequence State: Closed Variables:
RequestMessage Value: [<s:Envelope xmlns:a="http://www.w3.org/2005/08/addressin
g" xmlns:s="http://www.w3.org/2003/05/soap-envelope">
<s:Header>
<a:Action s:mustUnderstand="1">http://tempuri.org/IAccountService/DoWork</a:
Action>
<a:MessageID>urn:uuid:e068d7b4-bf6d-4488-8b72-83eaf7bfa589</a:MessageID>
<a:ReplyTo>
<a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
</a:ReplyTo>
<CoordinationContext s:mustUnderstand="1" xmlns:mstx="http://schemas.microso
ft.com/ws/2006/02/transactions" xmlns="http://schemas.xmlsoap.org/ws/2004/10/wsc
oor">
<wscoor:Identifier xmlns:wscoor="http://schemas.xmlsoap.org/ws/2004/10/wsc
oor">urn:uuid:98959e1d-ddf3-49b1-aabb-650941d7a460</wscoor:Identifier>
<Expires>300000</Expires>
<CoordinationType>http://schemas.xmlsoap.org/ws/2004/10/wsat</Coordination
Type>
<RegistrationService>
<Address xmlns="http://schemas.xmlsoap.org/ws/2004/08/addressing">https:
//pciomp1025/WsatService/Registration/Coordinator/Disabled/</Address>
<ReferenceParameters xmlns="http://schemas.xmlsoap.org/ws/2004/08/addres
sing">
<mstx:RegisterInfo>
<mstx:LocalTransactionId>98959e1d-ddf3-49b1-aabb-650941d7a460</mstx:
LocalTransactionId>
</mstx:RegisterInfo>
</ReferenceParameters>
</RegistrationService>


C:\Windows\system32\cmd.exe
</RegistrationService>
<mstx:IsolationLevel>0</mstx:IsolationLevel>
<mstx:LocalTransactionId>98959e1d-ddf3-49b1-aabb-650941d7a460</mstx:LocalTransactionId>
<PropagationToken xmlns="http://schemas.microsoft.com/ws/2006/02/tx/oletx">AQAAAAMAAAAdnpaY892xSaq7ZQ1B16RgABAAAAAAAACUAAAAAAA

Ekran çıktısını okumak biraz zahmetli olabilir*(Özellikle WCF servis çağrısının yapıldığı aktivite mesajlaşma içeriğini de bastığından…)* ama işlemlerin başarılı bir şekilde yapıldığı görülmektedir ve doğal olarak veritabanı tarafındaki **insert** işlemleri de başarılı olmuştur. Özellikle **activity** tipleri için yapılan **State** bildirimlerine dikkatinizi çekerim.

Hatta **Debug** işlemi yapıldığında servis tarafındaki operasyon içerisinde, istemci tarafından gelen**Transaction** bilgileri de açık bir şekilde görülebilmektedir.



Dikkat edileceği üzere **DistributedIdentifier** özelliğinin **GUID** tipinden bir değeri mevcuttur. Bir başka deyişle **DTC** devreye girmiş ve şu andaki servis operasyonu içerisinde yapılacak işlemler,**Workflow Application** tarafında açılan **TransactionScope'** a dahil edilmiştir.

Eğer **InsertBranch Code Activity** bileşeni içerisindeki **Exception** fırlatılan satır etkinleştirilirse, **Transaction** işlemlerinin **Commit** edilmediği görülecektir. Bu kısım oldukça önemlidir. Çünkü**Workflow1** akışında önce akış içi bir **Transaction** işlemi, sonrasında servis tarafında bir**Transaction** işlemi ve son olarak da yine akış içerisinde bir **Transaction** işlemi söz konusudur.

Son işlemde oluşan **Exception** nedeni ile bir adım önceki servis **Transaction** işleminin iptal edilmesi ve edildiğinin görülmesi*(tabi o ana kadarki tüm Insert' lerin de iptal edildiğinin görülmesi)*son derece önemlidir. Bu, **TransactionScope'** un başarılı çalıştığının bir ispatı olarak düşünülebilir.

Görüldüğü üzere bir **Workflow** içerisinden başlatılan **Transaction'** ın, bir **WCF** servis operasyonuna aktarılabilmesi ve söz konusu operasyonun ilgili **Transaction Scope'** a dahil hale gelerek **Two Phase Commit** metodolojisine uygun biçimde sisteme dahil edilmesi mümkündür. Bu, tipik anlamda bir **Atomic Transaction** senaryosudur. Sadece bir kaç

küçük detaya ve ayarlamaya dikkat etmek gerekmektedir. Böylece geldik bir yazımızın daha sonuna. Tekrarda görüşünceye dek hepinize mutlu günler dilerim

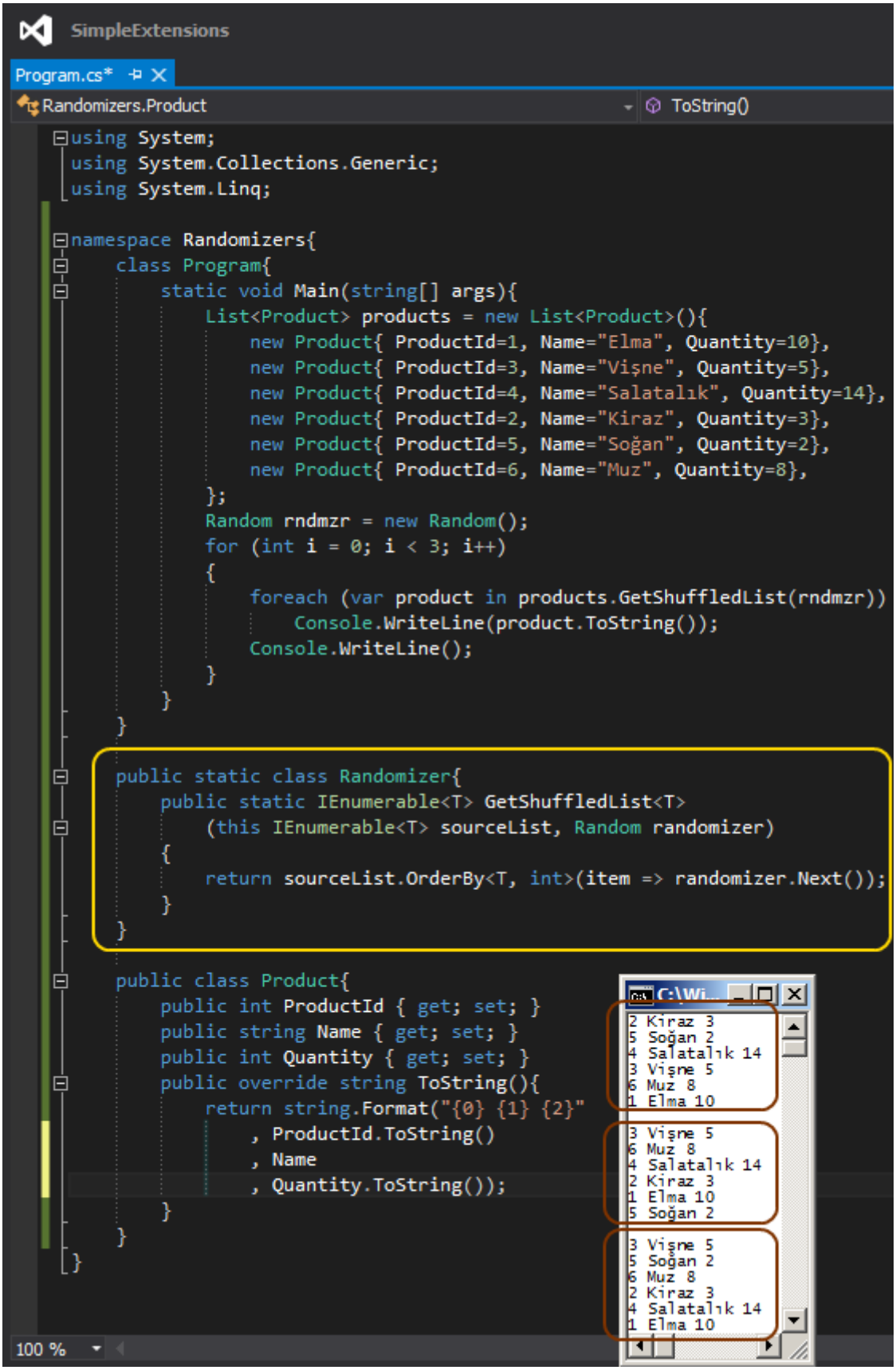
HowToTransaction.zip (457,18 kb)

[Örnek Visual Studio 2010, .Net Framework 4.0 tabanlıdır]

Tek Fotoluk İpucu 76–Bir Listeyi Shuffle' lamak

Perşembe, 31 Ocak 2013 08:50 by bsenyurt

Merhaba Arkadaşlar,
Malum hepimizin devasaaa/kocaman boyutlarda MP3 arşivleri var ve genelde müzik dinlerken de uygulamaların **shuffle** özelliklerini açarak, karışık sırada dinlemeyi tercih ediyoruz. Peki kendi tiplerinize ait **generic** bir listeyi **Shuffle**' layarak kullanmak isteseydiniz, nasıl bir yol izlersiniz? Aşağıdaki gibi olabilir mi?



Bir başka ipucunda görüşmek dileğiyle.

.Net Framework 4.5 Asenkron IO İşlemleri

Cuma, 25 Ocak 2013 13:27 by bsenyurt

Merhaba Arkadaşlar,
Geçtiğimiz gün özlem duyduğum bilgisayar oyunlarından birisi olan **Warcraft II**' nin ses efektlerini arar halde buldum kendimi. Olay tabi ses efektlerinin mükemmelliğinden çıktı oyuncu karakterlerine kadar geldi. Gerek Orc' lar da gerek Human' lar da süper kahramanlar vardı. Büyücüler, okçular, işçiler ve daha niceleri. Pek çoğumuzun bu oyun başında saatler harcadığından ve sabah ezanına kadar kaldığından eminim.
Nedense söz konusu karakterleri bir araştırma konusunda kullanma ihtiyacı da hissettim. Tesadüfe bakın ki aynı anda .Net tarafında da bir konuyu araştırmaktaydım. Sonuç olarak aşağıdaki konu için onları kendi bakış açımdan değerlendirmeye karar verdim. Bakalım bu günkü yazımızda nasıl bir maceraya dalıyor olacağız Haydi buyrun öyleyse başlayalım.
Asenkron çalışma çok uzun zamandır hayatımızda. Ancak yazılım geliştiriciler için halen daha tam anlamıyla mükemmel değil. Özellikle **.Net Framework** açısından olaya baktığımızda. Yine de her**.Net Framework** sürümünde, **User Experience**' ın yoğun ve cevap verebilirliği yüksek ekranların tasarlanmaya çalışıldığı senaryolarda, geliştiricilerin ellerinde daha güçlü kozlar oluşmakta. Burada en büyük yardımcı tabiki **Framework** içerisine dahil edilen **Built-In** operasyonlar/fonksiyonlar.
Bu alanlardan birisi de tahmin edileceği üzere **dosya giriş çıkış işlemleri(IO operasyonları).**Dosyalama işlemleri özellikle **.Net Framework 4.0** sürümüne kadar **senkron(Synchronous)**olarak işletilebilen operasyonlardan ibaretti. Diğer yandan **.Net Framework 4.0** sürümü ile birlikte**StreamReader** ve **StreamWriter** gibi sınıflara **BeginRead, BeginWrite** gibi temel asenkron çalışabilme metodları ilave edildi. **Bilindiği üzere en basit mana da bir işlevi başlatıp anında koda dönebilmek için IAsyncResult arayüzünün(Interface) kullanıldığın BeginX, EndX gibi dahili metodlardan yararlanılır. Bu, olay tabanlı asenkron programlamaya(Event Based Asynchronous Programming) da olanak tanımaktadır.**
Bu yeni metodlar sayesinden dosyalara asenkron olarak yazmak veya kaynakta yine asenkron olarak okuma yapmak mümkün. Yine de ortada geliştiriciler için bir sıkıntı var. O da kodun karmaşıklığı ve yazım zorluğu.

IO işlemlerinde neden asenkron çalışmaya ihtiyaç duyarız?

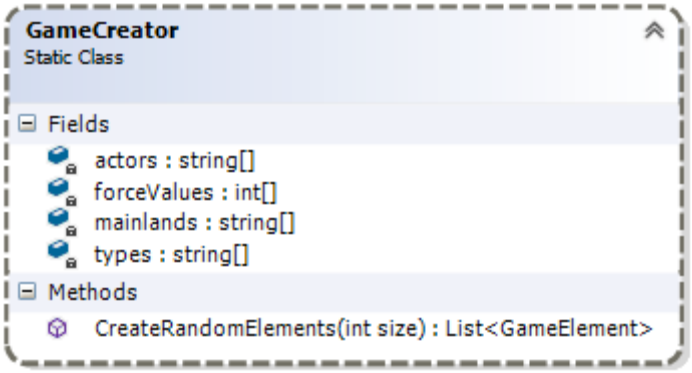
Bu soruyu da cevaplamak önemli. İlk etapta kullanıcıların ele aldıkları ekranların cevap verebilirliği(Responsivable) yer almakta. Kullanıcılar sabırsız kişilikte insanlar. O nedenle özellikle bir**IO** işlemi sırasına ekranın kitlenmesini*(kitlenmiş gibi gözükmesini)* istemezler.

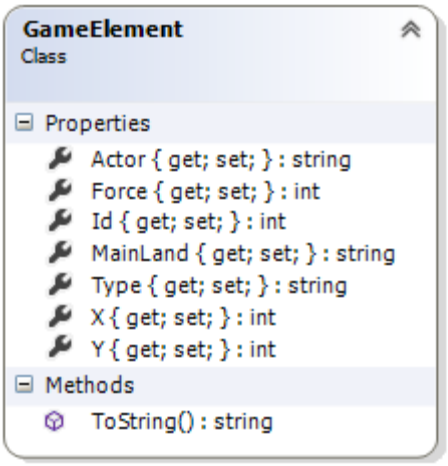
Diğer yandan ikinci bir sebep te tamamen performans ile alakalıdır. Bazı hallerde **IO** işlemi gerçekten çok zaman alan ve sistem kaynaklarını önemli ölçüde tüketen hareketliliklere gebedir. Dolayısıya bu yükü asenkron olarak dağıtabilmek önemlidir.
Pek tabi **.Net Framework 4.5** ile birlikte özellikle **async** ve **await** anahtar kelimelerinin de devreye girmesi ile **IO** tarafı için de bazı iyileştirmeler yapıldı. Bu iyileştirmeler sayesinde **IO** işlemleri sırasında cevap verebilirliği yüksek olan ekranların tasarlanması daha kolay hale gelmekte. Buna ek olarak **CLR(Common Language Runtime)** takımı bu yeni fonksiyonellikler için eş zamanlı*(Concurrent)* çalışmanın performansını da arttırıcı iyileştirmeler yapmış. Ancak bana göre belki de en önemli avatajımız, geliştirici açısından bakıldığında, ihtiyaçların daha kolay yazılabiliyor olması.
Gerçek hayattan;
daha önceden çalıştığım bir banka da, sistemler arasında veri taşınması veya dışarıdan gelen bazı verilerin database ortamlarına alınması, büyük boyutlu ve genellikle text tabanlı olan dosyalar üzerinden yapılmaktaydı. Çoğunlukla otomatik olarak devreye giren ve bir SQL Job ile ilişkilendirilmiş olan SSIS*(Sql Server Integration Services)* paketleri söz konusuydu.
Ancak SSIS' de ayrı bir uzmanlık alanı gerektirmekte ve bazı durumlarda söz konusu dosya aktarım işlemleri, Form tabanlı uygulamalar içerisinden yapılmak zorundaydı. Hal böyle olunca bazı vakalarda, boyutu 650 megabyte' ın üzerinde olan dosyalar için uygulama ekranının kilitlenmesi de son derece doğaldı. Hele de bu bir banka olunca süreci asenkron olarak çalıştırmak şarttır. İşte size Asenkron IO işlemleri için kocaman bir sebep 😊

Yeni Fonksiyonlar

Biz bu makalemizde özellikle **StreamReader** ve **StreamWriter** sınıflarına gelen ve asenkron **IO**işlemlerini yapmamızı sağlayan yeni metodları incelemeye çalışıyor olacağız. Konsept olarak sadece kullanım şekillerini incelemek niyetindeyiz. Bu sebepten senaryolarımızı bir **Console** uygulaması üzerinden ele alacağız. **StreamReader** ve **StreamWriter** gibi IO sınfılarına ilave edilen asenkron metodları bulmamız aslında son derece kolay. İsimlendirme standardı olarak sonu **async** kelimesi ile biten metodlar aradığımız fonksiyonlar olacaktır 😉
Dilerseniz işe ilk olarak asenkron yazma işlemleri ile başlayalım ki elimizde büyükçe bir test dosyası da oluşsun 😊
Bu amaçla basit ama işi zevkli hale getirecek de bir ön hazırlık yapacağız. Senaryomuzda en az**10milyon** elementten oluşan bir oyun sahası bulunacak. Bu saha içerisinde okçu, şovalye, mancınık, kale gibi karekterlere yer vereceğiz. Bu karakterlere ait başka bilgileri de tutuabiliriz. Örneğin bulundukları kıta, harita üzerindeki koordinat bilgisi, güç seviyleri, hangi taraftan oldukları*(insan, bilgisayar, hayalet* 😉 *)* gibi.

E tabi bu karakterlerden oluşan kümeyi üretecek bir de yardımcı tipimizin söz konusu olduğunu düşünebiliriz. Lafı fazla uzatmadan ön hazırlık için kullanacağımız kod parçalarına bir göz atalım dilerseniz.





GameElement.cs;

```
using System;
namespace NewIOFunctions
{
   public class GameElement
   {
      public int Id { get; set; }
      public string Actor { get; set; }
      public int Force { get; set; }
      public string Type { get; set; }
      public string MainLand { get; set; }
      public int X { get; set; }
      public int Y { get; set; }
      public override string ToString()
      {
         return String.Format("{0}|{1}|{2}|{3}|{4}|({5};{6})", Id.ToString(), Actor, Force,
Type, MainLand,X.ToString(),Y.ToString());
```

```
        }
    }
}
```

GameCreator.cs;

```
using System;
using System.Collections.Generic;
namespace NewIOFunctions
{
    public static class GameCreator
    {
        // Test verileri için örnek bir küme
        // Savaş alanına yerleştireceğimiz aktörler
        static string[] actors = { "Archer", "Footman", "Knight", "Castle", "Rider", "Catapult"
};
        // Bu aktörlerin güçlerini temsil eden değerler.
        static int[] forceValues = { 100, 200, 300, 400 };
        // Bu yerleştirilen aktörlerin hangi tipten olduğunu belirtecek taraf bilgileri(Ghost
enteresan bir karakterdir. Yeri gelince Human yeri gelince Computer tarafında oluyor :)
        static string[] types = { "Human", "Computer", "Ghost" };
        static string[] mainlands = { "Mordor", "Middle Earth", "Other Side", "Wrong Side",
"Red Zone", "Deadly Land", "Kolburg" };
        public static List<GameElement> CreateRandomElements(int size)
        {
            List<GameElement> elements = new List<GameElement>(size);
            Random randomizer = new Random();
            for (int i = 0; i < size-1; i++)
            {
                GameElement element = new GameElement();
                element.Id = i;
                element.Actor = actors[randomizer.Next(0, actors.Length - 1)];
                element.Force=forceValues[randomizer.Next(0, forceValues.Length - 1)];
                element.Type = types[randomizer.Next(0, types.Length - 1)];
                element.MainLand = mainlands[randomizer.Next(0, mainlands.Length - 1)];
                element.X = randomizer.Next(-90, 90);
                element.Y = randomizer.Next(-90, 90);
                elements.Add(element);
            }
            return elements;
        }
    }
}
```

static GameCreator sınıfının **CreateRandomElements** isimli metodu istenen boyuta göre rastgele veriler ile üretilen **GameElement** tipinden bir koleksiyon döndürecek şekilde tasarlanmıştır.

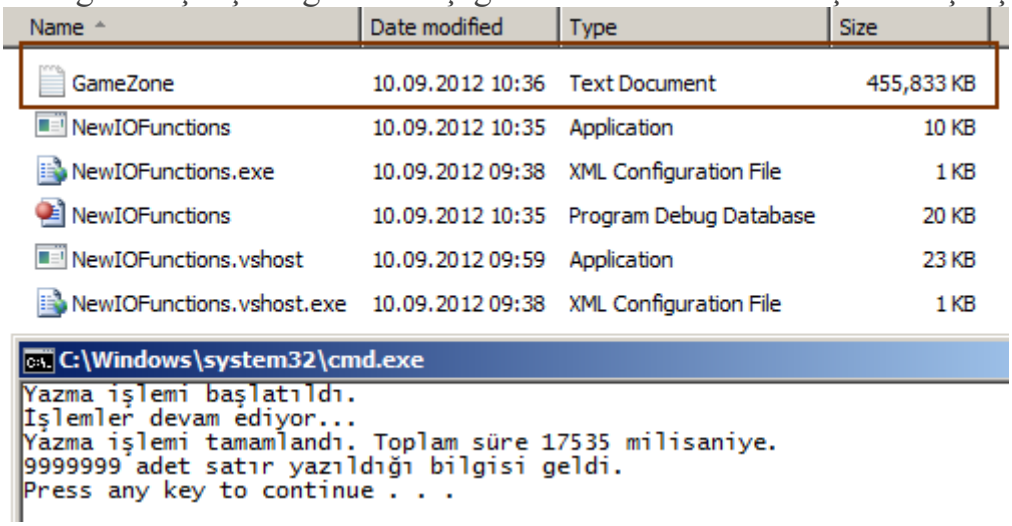
Dilerseniz buradaki üretim işini Task Parallel Library' yi ve Concurrent koleksiyonları da işin içerisine katarak asenkron doldurabilirsiniz. Ama dikkatli olun. Her üretiminizde 10milyonluk bir set elde edemessiniz. Tabi Thread' leri en azından gereken yerlerde kilitleyip senkronize etmedikçe 

Şimdi asıl senaryolarımıza geçebiliriz. İlk etapta üretilen bu rastgele veri içeriğini fiziki bir dosyaya asenkron moda da nasıl yazdırabileceğimizi incelemeye çalışacağız. Bu nedenle **Program.cs**içeriğinde aşağıdaki kodlar ile işe başlayabiliriz.

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Threading.Tasks;
namespace NewIOFunctions
{
   class Program
   {
      static void Main(string[] args)
      {
         //10 milyonluk bir test kümesi üretiyoruz. Bunu asenkron geliştirmediğimizden biraz zaman alacaktır.
         var gameZone = GameCreator.CreateRandomElements(10000000);
         var taskResult=WriteFileAsync(gameZone, Path.Combine(Environment.CurrentDirectory, "GameZone.txt"));
         Console.WriteLine("İşlemler devam ediyor...");
         // Dosya yazma ile ilişkili işlemlerin tamamlanması için uygulamayı bekletiyoruz.
         taskResult.Wait();
         Console.WriteLine("{0} adet satır yazıldığı bilgisi geldi.", taskResult.Result.ToString());
      }
      // WriteFileAsync metodundan geriye bilinçli olarak bir Task tipi döndürdük. Böylece burayı çağırdığımız Main metodu sonuna gelindiğinde ilgili Task' ın tamamlanıp tamamlanmadığını kontrol edebiliriz.
      private static async Task<int> WriteFileAsync(List<GameElement> elements, string filePath)
      {
         Console.WriteLine("Yazma işlemi başlatıldı.");
         Stopwatch watcher = new Stopwatch();
         watcher.Start();
```

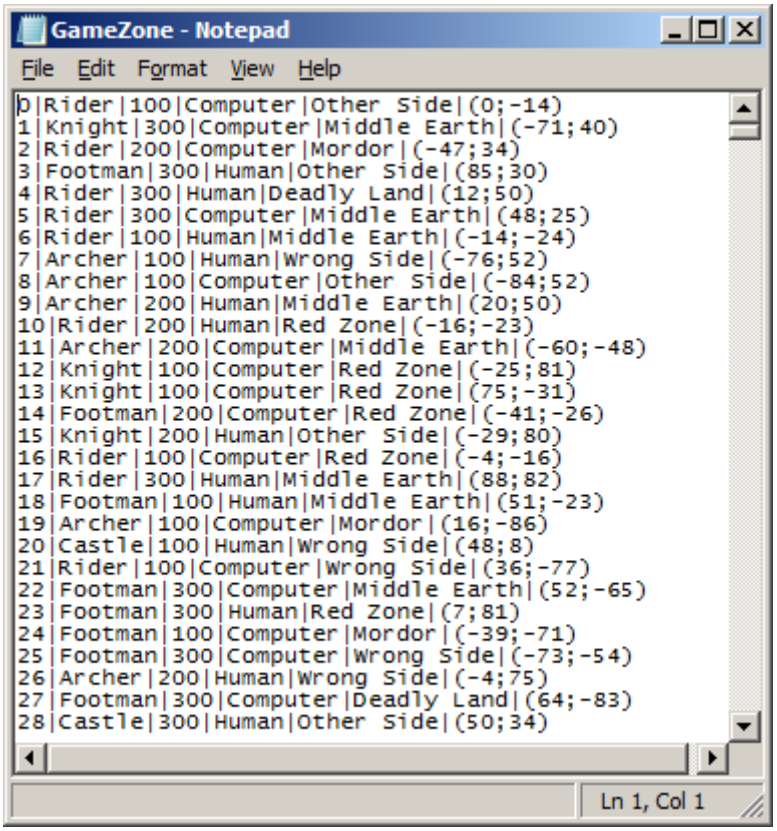
```
        // StreamWriter tipini üretiyoruz ve parametre olarak veriyi yazacağımız dosya adresini belirtiyoruz
        using (StreamWriter writer = new StreamWriter(filePath))
        {
           foreach (var element in elements)
           {
              // Awaitable olan WriteLineAsync metodunu çağırıyor ve içeriği yazdırıyoruz.
              await writer.WriteLineAsync(element.ToString());
           }
        }

        watcher.Stop();
        Console.WriteLine("Yazma işlemi tamamlandı. Toplam süre {0} milisaniye.",watcher.ElapsedMilliseconds.ToString());
        return elements.Count;
     }
  }
}
```

Örneğimizi çalıştırdığımızda aşağıdakine benzer bir sonuç ile karşılaşırız.



Üretilen dosya içeriğinin bir kısmı da aşağıdaki gibidir.

Tabi bu dosyayı Notepad açmayı nasıl başardı anlayabilmiş değilim  **Yine de siz kendi sistemlerinizde zorlanmamak adına, Notepad2' yi veya Notepad++' ı kullanmayı düşünebilirsiniz.**



Dikkat edileceği üzere **WriteFileAsync'** e yapılan çağrı içerisinde **StreamWriter** tipinin **await**edilebilir **WriteLineAsync** metodu kullanılmaktadır. **WriteFileAsync** metodu aslında geriye bir**Task<int>** tipi döndürmek zorunda değildir. Bunu daha çok, **Wait** metodu kullanılarak başlatılan**Task'** ın işlemlerini bitirene kadar uygulamanın beklemesi ve işlenen satır sayısının bir değerinin ana**Thread** tarafından elde edilebilmesi için kullandık. Nitekim uygulamayı akışına bırakıp **Wait**çağrısını gerçekleştirmessek, tüm içerik yerine program sonlanana kadar yazılabilen kısım fiziki olarak aktarılacaktır.

Örnek senaryonun işletilişi sırasında gözden kaçırılmaması gereken bir nokta da, aslında dosyaya yazma işleminin asenkron olarak gerçekleştirilmemesidir. Yani, GameElement tipindeki koleksiyon içeriğinin eş zamanlı(Concurrent) olarak yazılması gibi bir durum ortada yoktur. Dosya içeriğine baktığımızda, içeriğin sıralı olarak yazılmış olmasının nedeni de budur. Yanlış bir asenkron algı oluşmaması için bu hususu belirtmek istedim. Nitekim dosya içeriğini asenkron olarak doldurmak tamemen farklı bir vakadır.

Dilerseniz bir de dosyadan okuma işlemini basit bir senaryo ile ele almaya çalışalım. Yine satır satır okuma adımını gerçekleştirebiliriz. İşte kod içeriğimiz.

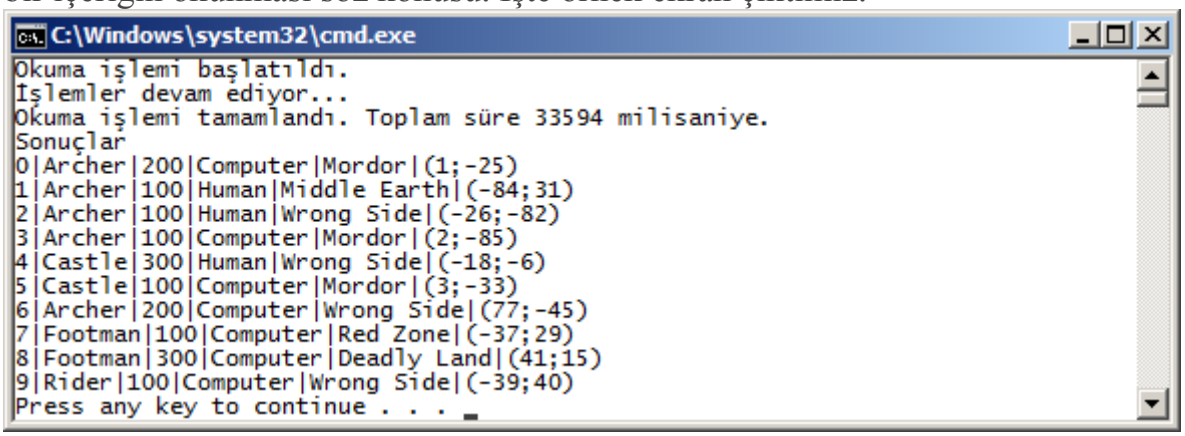
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Threading.Tasks;
```

```
namespace NewIOFunctions
{
   class Program
   {
      static void Main(string[] args)
      {
         var readTaskResult =
ReadFileLineByLineAsync(Path.Combine(Environment.CurrentDirectory,
"GameZone.txt"));
         Console.WriteLine("İşlemler devam ediyor...");
         readTaskResult.Wait();
         Console.WriteLine("Sonuçlar");
         for (int i = 0; i < 10; i++)
         {
            Console.WriteLine(readTaskResult.Result[i].ToString());
         }
      }
      private static async Task<List<GameElement>>
ReadFileLineByLineAsync(string file)
      {
         Console.WriteLine("Okuma işlemi başlatıldı.");
         List<GameElement> elements = new List<GameElement>();
         Stopwatch watcher = new Stopwatch();
         watcher.Start();
         using (StreamReader reader = new StreamReader(file))
         {
            string line;
            while (!String.IsNullOrEmpty(line = await reader.ReadLineAsync()))
            {
               string[] columns=line.Split('|');
               elements.Add(new GameElement
               {
                  Id=Convert.ToInt32(columns[0]),
                  Actor=columns[1],
                  Force=Convert.ToInt32(columns[2]),
                  Type=columns[3],
                  MainLand=columns[4],
                  X=Convert.ToInt32(columns[5].Split(';')[0].TrimStart('(')),
                  Y = Convert.ToInt32(columns[5].Split(';')[1].TrimEnd(')'))
               }
               );
```

```
            }
        }
        watcher.Stop();
        Console.WriteLine("Okuma işlemi tamamlandı. Toplam süre {0} milisaniye.",
watcher.ElapsedMilliseconds.ToString());
        return elements;
    }
  }
}
```

Okuma işleminin tamamlanmasını takiben ,üretilen **GameElement** içerikli **generic List**koleksiyonunun metoddan neden bu şekilde bir **Task** tipi ile döndürüldüğü noktasında kafalarda bir soru işaret olabilir. Neden normal bir liste döndürmedik? Ya da neden **out** veya **ref** işaretli metod parametrelerine başvurmadık? Cevap **async** kullanmış olmamızdır. Bu şekilde işaretlenmiş metodlar**void**, **Task** ya da **Task<T>** dönüşüne sahip olabilir. Ayrıca **ref** ve **out** kullanımına izin verilmez.

Örneğimizin çalışması tabiki biraz uzun sürebilir. Kolay değil yaklaşık **450 megabyte'** lık bir içeriğin okunması söz konusu. İşte örnek ekran çıktımız.



Her iki örnekte de önemli olan, dosyadan satır bazında okuma ve dosyaya satır bazında yazma işlemlerini içeren fonksiyonelliklerin ana uygulamadan bağımsız olarak asenkron çalışabiliyor olmalarıdır. Bir başka deyişle eş zamanlı*(Concurrent)* olarak bir dosya içerisine yazma veya okuma işlemi söz konusu değildir.

Async olarak kullanılabilecek başka **IO** metodları da mevcuttur. Bunları aşağıdaki şekilde sıralayabiliriz.

- ReadAsync
- ReadToEndAsync
- ReadBlockAsync
- WriteAsync
- WriteLineAsync
- FlushAsync

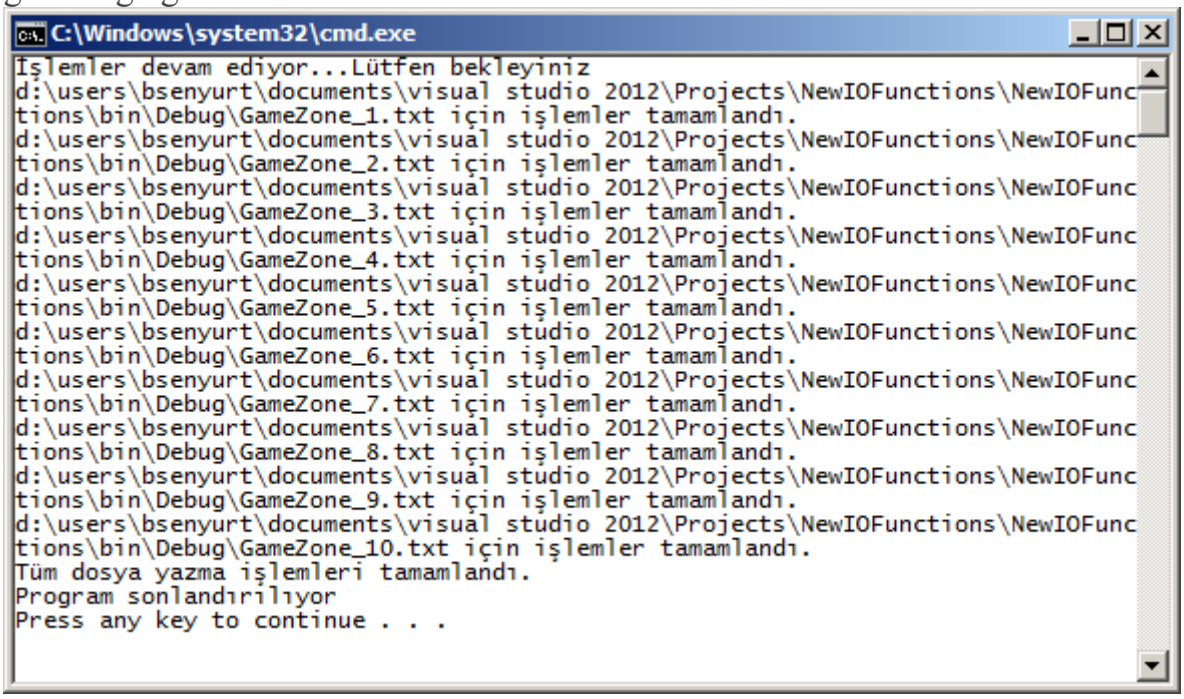
Parçalanmış Asenkronluk

Son bir senaryo ile yazımıza devam edelim dilerseniz
Bu kez **10milyon** element' ten oluşan sahımızı 10 parçaya bölüp her bir parça içerisinde ayrı bir**Text** dosyaya yazma işlemini, asenkron manada simüle etmeye çalışıyor olacağız. İşte senaryoya istinaden ele alabileceğimiz kod parçaları.

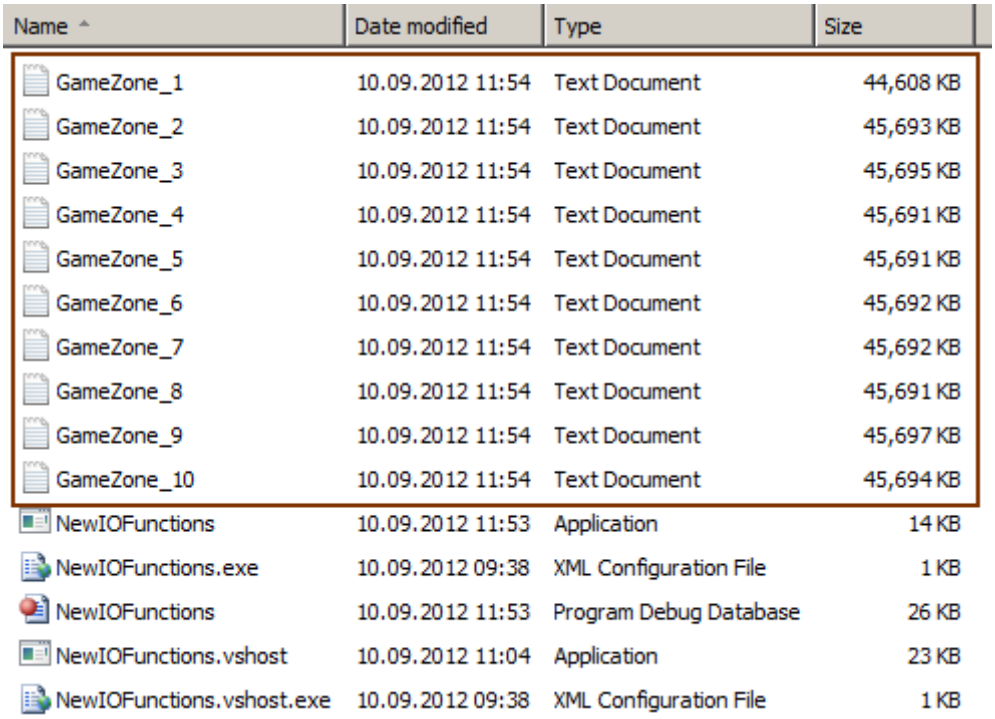
```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
namespace NewIOFunctions
{
   class Program
   {
      static void Main(string[] args)
      {
         //10 milyonluk bir test kümesi üretiyoruz. Bunu asenkron geliştirmediğimizden biraz zaman alacaktır.
         var gameZone = GameCreator.CreateRandomElements(10000000);
         var r=WriteTo10File(gameZone);
         Console.WriteLine("İşlemler devam ediyor...Lütfen bekleyiniz");
         r.Wait();
         Console.WriteLine("Program sonlandırılıyor");
      }
      private static async Task WriteTo10File(List<GameElement> elements)
      {
         for (int i = 1; i <= 10; i++)
         {
            string fileName = Path.Combine(Environment.CurrentDirectory, "GameZone_" + i.ToString() + ".txt");
            await WriteFileAsyncV2(elements.Skip((i-1)*1000000).Take(1000000).ToList(), fileName);
         }
         Console.WriteLine("Tüm dosya yazma işlemleri tamamlandı.");
      }
      private static async Task WriteFileAsyncV2(List<GameElement> elements, string filePath)
      {
         using (StreamWriter writer = new StreamWriter(filePath))
         {
            foreach (var element in elements)
```

```
            {
               // Awaitable olan WriteLineAsync metodunu çağırıyor ve içeriği yazdırıyoruz.
               await writer.WriteLineAsync(element.ToString());
            }
            Console.WriteLine("{0} için işlemler tamamlandı.",filePath);
         }
      }
   }
}
```

Hem **WriteTo10File** hem de **WriteFileAsyncV2** metodları **async** olarak işaretlenmiş olup kendi içlerinde **await** edilebilir operasyon çağrılarına yer vermektedirler. İlk olarak 10milyonluk küme 10 eşit parça şeklinde ele alınmakta ve her bir alt kümenin asenkron bir metod ile çalışacak şekilde ilgili dosyalara yazdırılması işlemi gerçekleştirilmektedir. Dosyaya satır bazlı yazma işlemi için yine **WriteLineAsync** fonksiyonunundan yararlanılmaktadır.**WriteTo10File** metoduda kendi içerisinde 1milyonluk kümelere ayırdığı element listelerini ayrı dosyalara yazmak
için **WriteFileAsyncV2** metodunu **await** anahtar kelimesi ile çağırmaktadır ki bu da gözden kaçırılmaması gereken bir noktadır. Kodun çalışma zamanı çıktısı aşağıda görüldüğü gibidir.



Tabi klasör yapısına bakıldığında GameZone_1' den GameZone_10' a kadar 10 farklı dosyanın üretildiği ve içerisine de 10milyon kümenin 1milyon parçalarının yazıldığı görülebilir.



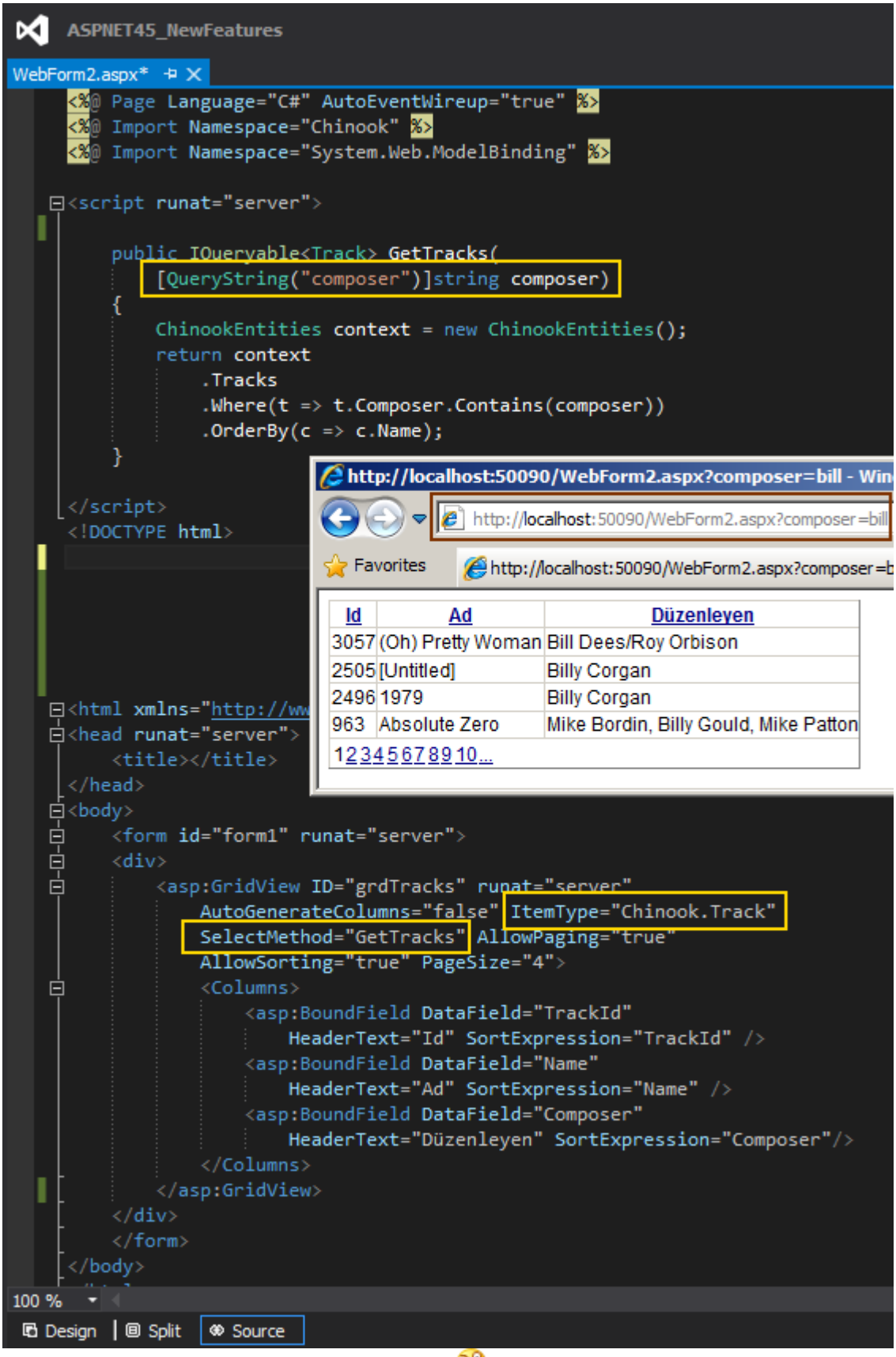
Name	Date modified	Type	Size
GameZone_1	10.09.2012 11:54	Text Document	44,608 KB
GameZone_2	10.09.2012 11:54	Text Document	45,693 KB
GameZone_3	10.09.2012 11:54	Text Document	45,695 KB
GameZone_4	10.09.2012 11:54	Text Document	45,691 KB
GameZone_5	10.09.2012 11:54	Text Document	45,691 KB
GameZone_6	10.09.2012 11:54	Text Document	45,692 KB
GameZone_7	10.09.2012 11:54	Text Document	45,692 KB
GameZone_8	10.09.2012 11:54	Text Document	45,691 KB
GameZone_9	10.09.2012 11:54	Text Document	45,697 KB
GameZone_10	10.09.2012 11:54	Text Document	45,694 KB
NewIOFunctions	10.09.2012 11:53	Application	14 KB
NewIOFunctions.exe	10.09.2012 09:38	XML Configuration File	1 KB
NewIOFunctions	10.09.2012 11:53	Program Debug Database	26 KB
NewIOFunctions.vshost	10.09.2012 11:04	Application	23 KB
NewIOFunctions.vshost.exe	10.09.2012 09:38	XML Configuration File	1 KB

Senaryolarımız ile **Console** üzerinden basit pratiklerimizi yapmış olduk. Şimdi bu pratikleri gerçek hayat senaryoları ile değerlendirmeye çalışmalıyız. Dolayısıyla işlemlerimizi artık görsel arabirimi olan ve gerçekten de cevap verilebilirliğe ihtiyaç duyan **Windows Forms**, **WPF(Windows Presentation Foundation)** ve belki de **Asp.Net** gibi uygulama çeşitlerinde ele almalıyız. Her zaman olduğu gibi bu kutsal görevi siz değerli meslektaşlarıma bırakıyorum  Böylece geldik bir yazmızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.
NewIOFunctions.zip (48,16 kb)

Tek Fotoluk İpucu 77–Asp.Net 4.5 QueryStringAttribute

Çarşamba, 23 Ocak 2013 08:55 by bsenyurt

Merhaba Arkadaşlar,
Asp.Net 4.5 tarafında gelen yeniliklerden birisi de **System.Web.ModelBinding** isim alanı altında yer alan ve metod parametrelerine uygulanan **QueryString** niteliğidir*(Attribute)*.
Bu nitelik ile bir metodun parametre değerinin, **URL Querystring** üzerinden okunabileceği ifade edilmektedir.
Özellikle veri bağlı kontrollerin*(GridView gibi)*, **IQueryable<T>** benzeri referanslar döndüren metodlar ile ilişkilendirildikleri senaryolarda, query string yardımıyla filtreleme yapılmasında kullanılır. Aynen aşağıdaki fotoğrafta görüldüğü gibi



Bir başka ipucunda görüşmek dileğiyle

RavenDB ile Hello World

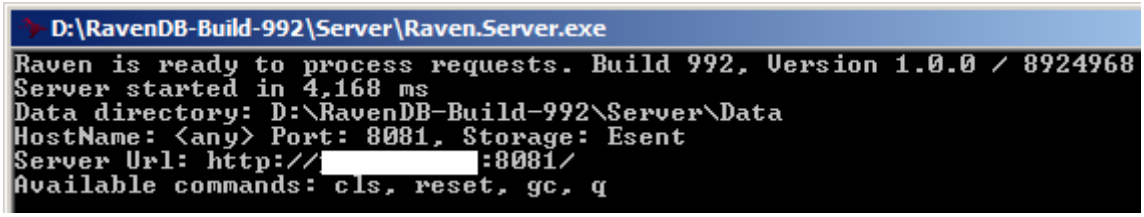
Cumartesi, 12 Ocak 2013 21:05 by bsenyurt

Merhaba Arkadaşlar,
Kuzgun' lar Kargagiller ailesinden gelen bir kuş çeşididir. Diğer karga cinslerine göre daha iridirler ve özellikle çok daha zeki oldukları söylenir. Yapılan araştırma ve deneyler sonrasında bu cins kargaların, sorunları çözmek için çevresel materyalleri kullanabilme*(kullanmak için de öğrenebilme)*becerisine sahip oldukları öne sürülmüştür. Hatta parlak, beyaz ve mavi renkli metallere karşı özel bir ilgileri olduğundan, hırsız olarak da ifade edilmektedirler. Kanat açıklığı 1.5 metreyi bulan bu kuşlar, aynı zamanda deli cesaretine sahiptir Niye mi? Çünkü, kendilerinden daha yırtıcı olan kuşlara hiç düşünmeden saldırabilirler. Tabi buradaki avantajları hep bir filo halinde hareket etmeleridir. Yani ekip olgusuna inanırlar. Bir diğer önemli özellikleri de ingilizce de **Raven** olarak adlandırılmalarıdır. *(Daha fazla detay istiyorsanız wikipedia bağlantısına bakabilirsiniz)*
Gelelim **Raven** ile ne işimiz olduğuna Açık kaynaklı **NoSQL** veritabanlarını incelemeye çalıştığımız ilk yazımızda, hatırlayacağınız üzere Apache Cassandra' ya kısaca bir göz atmış ve basit bir **Hello World** uygulaması geliştirmiştik. Tabi **NoSQL** veritabanı sistemleri denilince pek çok ürün olduğunu görmekteyiz. İşte bu yazımızda bu ürünlerden dikkate değer bir tanesini daha inceleyeceğiz; **RavenDB**.
RavenDb, açık kaynak **NoSQL** veritabanlarındandır. **.Net** ile yazılmıştır ve **şemasız*(Schema-less)*JSON*(Java Script Object Notation)*** veri tipini kullanmaktadır. Doküman tabanlı***(Document Based)*** çalışmaktadır. **JSON** formatınının kullanılması ve **.Net** ile yazılmış olması, erişilebilirlik ve ölçeklenebilirlik anlamında da bazı avantajlar sunmaktadır. Bunların arasında **LINQ*(Language INtegrated Query)*** ile sorgulanabilme ve **RESTful API** ile ulaşılabilme sayılabilir. Bu veritabanı ayrıca **Transactional**' dır. Bir başka deyişle **ACID*(Atomicity, Consistency, Isolation, Durability)*** prensiplerine destek vermektedir.
RavenDB' nin daha çok **Windows** tabanlı **.Net** uygulamaları için geliştirildiği düşüncesi hakimdir. Ancak **RESTful** desteği olması sebebiyle farklı platformlara da açılabilir. Hatta sunduğu **API**sayesinde **.Net, Silverlight, Javascript** ve **HTTP** tabanlı **REST** istemcileri ile çalışabilir. **JSON**formatını kullanması verinin az yer tutması için de bir avantajdır.
Document tipindeki NoSQL yapılarında genel olarak veri, bir dosya içerisinde saklanmakta ve bir anahtar*(key)* ile ilişkilendirilmektedir. Saklanacak veri için herhangibir şemaya*(Schema)* ihtiyaç yoktur.
Bu nedenle SQL, Oracle gibi ilişkisel veritabanı sistemlerinde*(Relational Database Management System)* yapılması gereken, tablo tanımlama ve benzeri işlemler mevcut değildir.
Çünkü buna gerekte yoktur

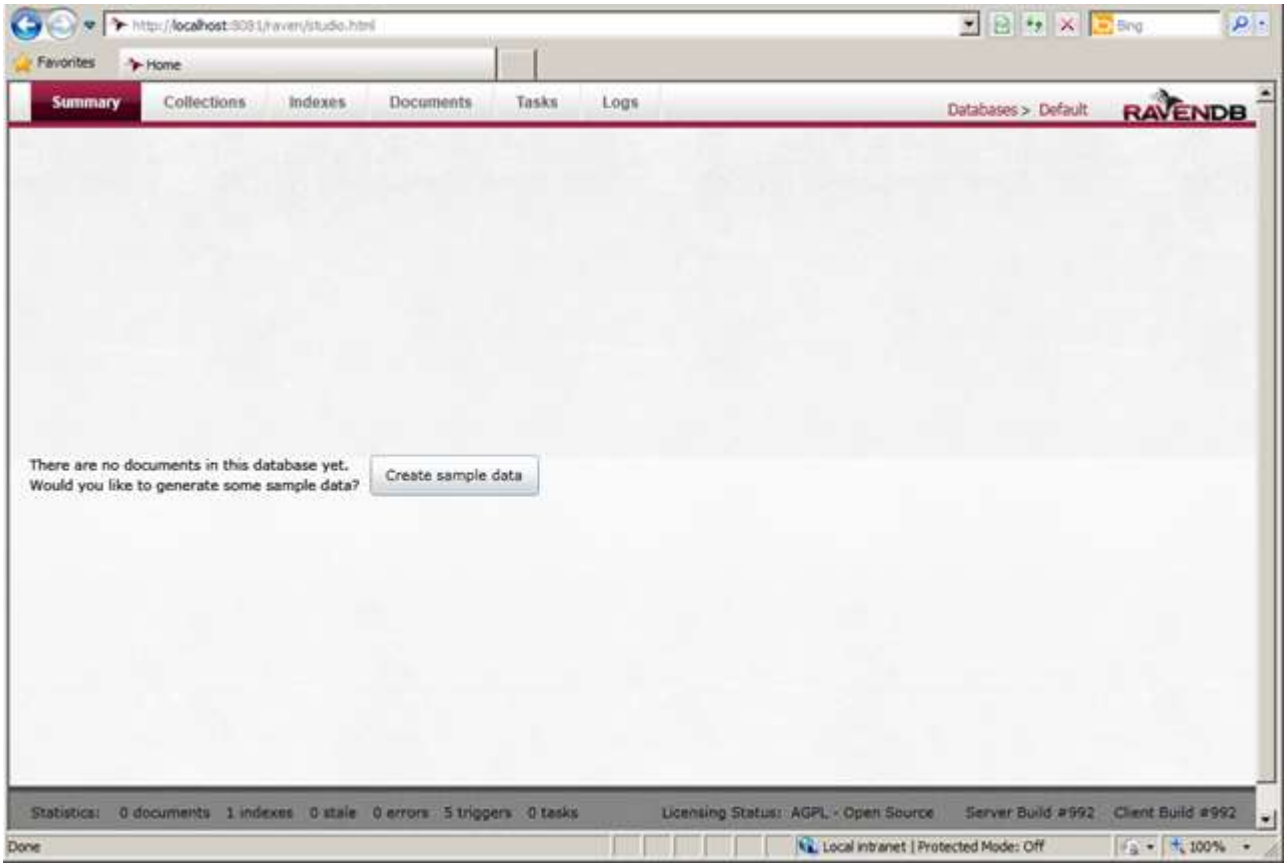
Düşünce son derece basit olmaktan yanadır. "Veriyi doğrudan diske yaz, okumak istediğinde anahtarı ile ulaş"
RavenDB ile ilişkili bir ürün tanıtımı ve kuzgun' ların kanat açıklığının 1.5 metre olması bilgilerinden sonra dilerseniz biraz da kod yazalım Öncelikli olarak **RavenDB**' yi kurmamız gerekiyor. Yazıyı yazdığım günlerde bunun için şu adrese bir uğramanız gerekmekteydi. Bilgilerden de anlaşılacağı üzere ürünü, **NuGet** paket yönetim aracı yardımıyla tedarik edebilirsiniz de.
NoSQL tipindeki veritabanı ürünlerinin bir diğer avantajıda kurulumlarının son derece kolay olmasıdır. Özellikle SQL, Oracle gibi ürünlerin kurulumları düşünüldüğünde Pek çok NoSQL ürünü açık kaynak olarak indirilebilir ve doğrudan çalıştırılıp kullanılabilir. Bir install işleminden sürecinden geçilmesine çoğu zaman gerek duyulmamaktadır.

Başlatma

RavenDB içeriğini indirdikten sonra, **Server** klasörü altında yer alan **Raven.Server.exe** isimli uygulamanın çalıştırılması yeterlidir. Komut satırından yürütülen uygulama, sunucunun çalışmasını sağlayacaktır. Tahmin edileceği üzere ürün, **client/server** modeline göre çalışmaktadır. Sunucu açık olduğu sürece, istemcilerin **RavenDB** sistemini kullanması mümkündür.



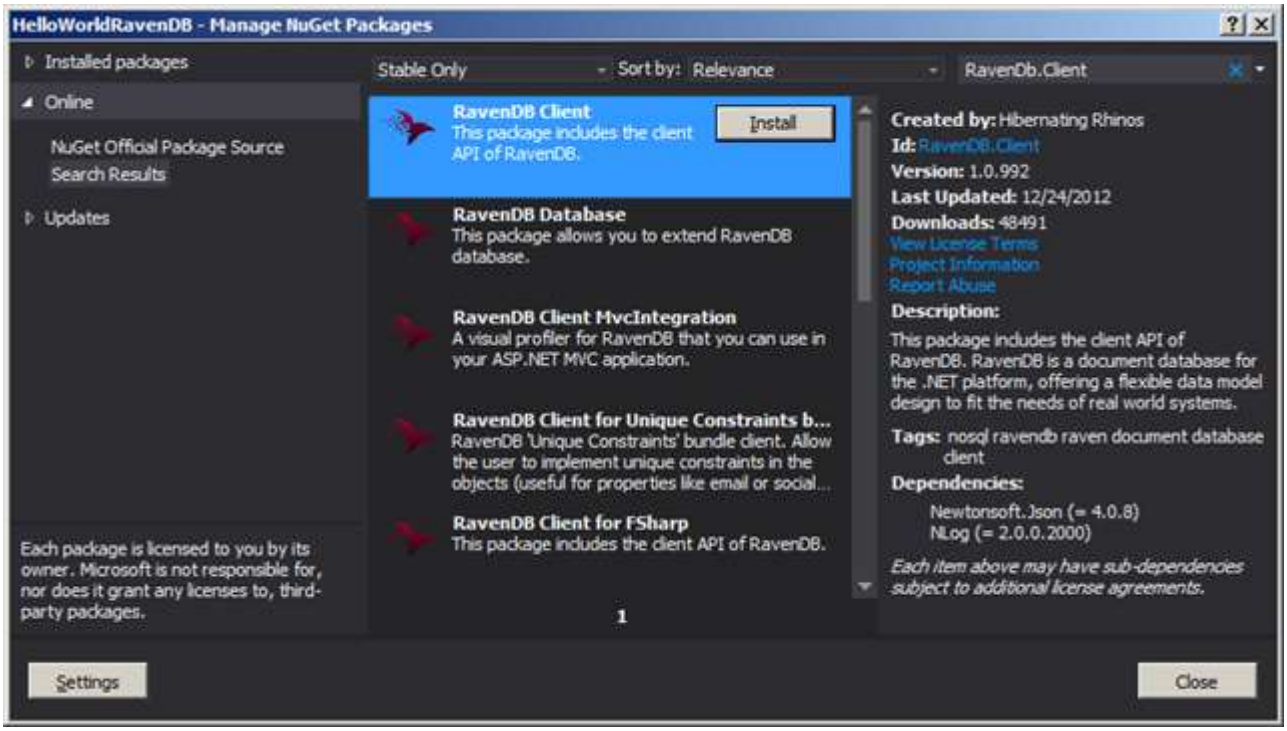
İşin güzel yanı, ürünün bir de web arayüzünün bulunmasıdır. Eğer makinenizde **8080 port'** u üzerinden yayın yapan bir başka uygulama var ise*(ki benim sistemimde vardı)* **RavenDB, 8081**numaralı **port** üzerinden hizmet vermeye çalışacaktır*(Eğer 8081 de doluysa tahminlerime göre bir sonraki boş portu bulana kadar deneyecektir)*
Buna göre **http://localhost:8081/** adresine gidildiğinde**http://localhost:8081/raven/studio.html** adresine yönlendirilip, web arayüzüne ulaşıldığı gözlemlenecektir.



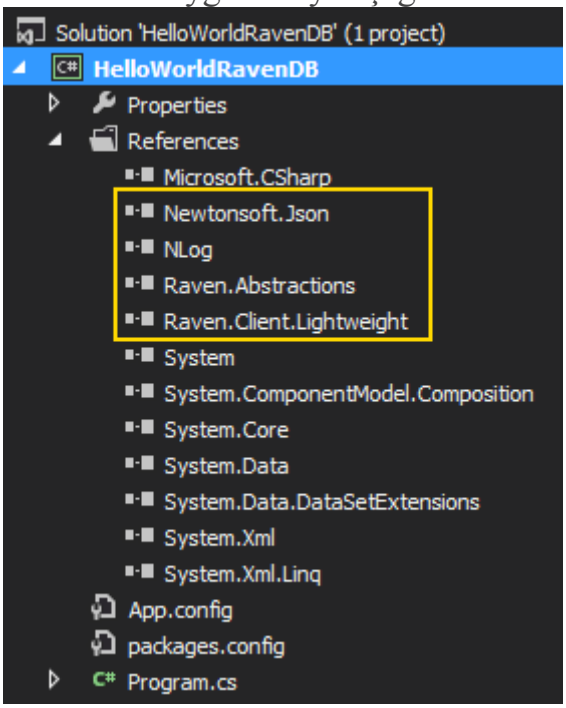
Bu arayüzden yararlanılarak verilerin eklenmesi, silinmesi, değiştirilmesi veya sorgulanması sağlanabilir. Elbette biz bunu kod üzerinden nasıl yapabileceğimizi incelemeye çalışacağız. Bu amaçla basit bir **Console** uygulaması oluşturarak işe başlayabiliriz.

İstemci için Hazırlık

RavenDB' yi istemci tarafında ele alırken yardımcı kütüphane olan **RavenDB.Client assembly**' ından yararlanılmaktadır. **RavenDB**' yi indirdiğimiz zaman **Client** klasörü içerisinden bu kütüphanenin farklı versiyonlarına da erişilebilinir. *(Hatta burada yakından tanıdığımız bir dost da vardır.* ***Newtonsoft.json.dll*** *Kendisi ile buradaki ve şuradaki tek fotoluk ipuçlarında haşırneşir olmuştuk)*
RavenDB istemci kütüphanesi, **NuGet** paket yönetim aracı ile de uygulamaya eklenebilir. Hatta bu şekilde ilerlenmesi, en güncel sürümün alınması ve yardımcı kütüphanelerin de indirilmesi açısından kolaylık sağlayan bir fonksiyonellik olarak görülmelidir.



Ben örnekte **NuGet** aracından yararlanarak ilgili kurulum işlemini gerçekleştirdim. Bunun sonucunda uygulamaya aşağıdaki **.Net** kütüphanelerinin eklendiğine şahit oldum.



Dikkat edileceği üzere 3ncü parti kütüphanelerden **NLog** ve **Newtonsoft.Json assembly**'larıda, referans edilmiş durumdadır.

Referans işlemlerinin ardından, istemcinin hangi sunucuya bağlanacağını da belirtmemiz gerekiyor. Aslında bu, **RavenDB** ile olan bağlantıda kullanılacak **ConnectionString** bilgisidir. Dolayısıyla söz konusu bağlantı bilgisi **app.config** dosyası içerisinde aşağıdaki gibi tanımlanabilir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
```

```
  <connectionStrings>
    <add name="RavenDBConnection"
connectionString="url=http://localhost:8081"/>
  </connectionStrings>
</configuration>
```

Çok doğal olarak ilk kuruluma göre **RavenDB**, **localhost'** daki **8081*(benim makinem de böyle varsayılan olarak 8080 portu)*** portu üzerinden yayın yapmaktadır.

İlk Kodlar

Aşağıdaki örnek kodları geliştirdiğimizi düşünelim.

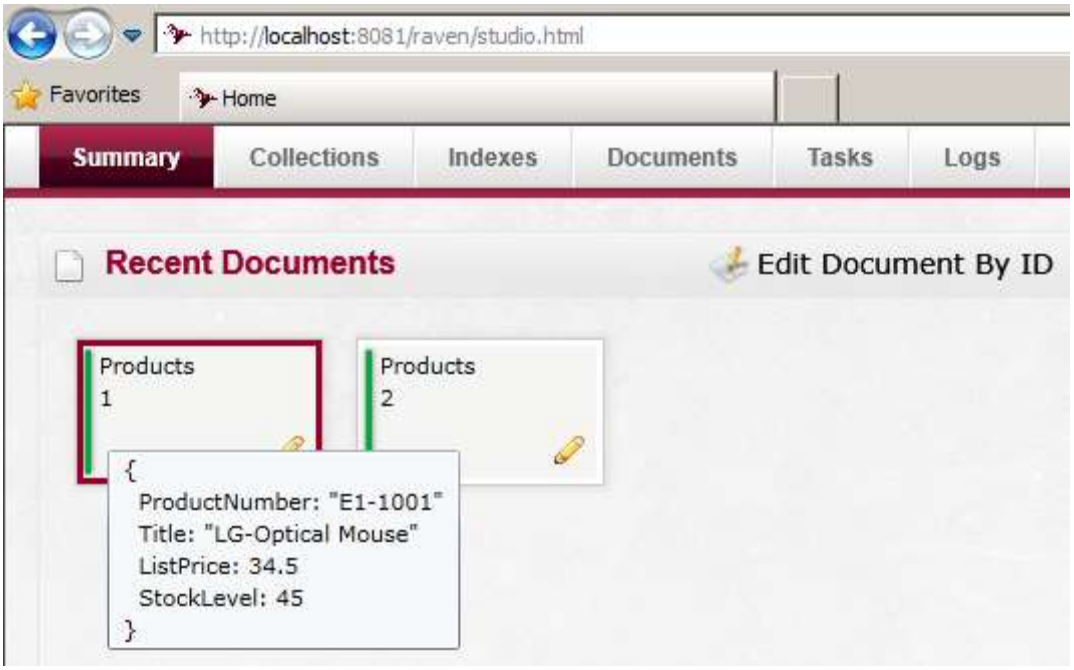
```
using Raven.Client.Document;
using System;
using System.Linq;
namespace HelloWorldRavenDB
{
   class Program
   {
      static DocumentStore docStore = null;
      static void Main(string[] args)
      {
         docStore=new DocumentStore { ConnectionStringName =
"RavenDBConnection" };
         docStore.Initialize();
         #region Örnek ürünlerin eklenmesi
         AddProduct(new Product
         {
            Id="1",
            ProductNumber = "E1-1001",
            Title = "LG-Optical Mouse",
            ListPrice = 34.50,
            StockLevel = 100
         }
         );
         AddProduct(new Product
         {
            Id="2",
            ProductNumber = "E1-1002",
            Title = "LG-Optical Keyboard",
            ListPrice = 44.25,
            StockLevel = 85
```

```
    }
    );
    AddProduct(new Product
    {
       Id="3",
       ProductNumber = "B1-1005",
       Title = "SOA Design Patterns",
       ListPrice = 49.95,
       StockLevel = 12
    }
    );
    #endregion
    ListAllProducts();
    ChangeStockLevel("1",45);
    ListAllProducts();
    DeleteProductByNumber("B1-1005");

    Product product=FindProductByNumber("E1-1002");
    Console.WriteLine(product.Title);
}
private static void AddProduct(Product newProduct)
{
    using (var session = docStore.OpenSession())
   {
       session.Store(newProduct);
       session.SaveChanges();
    }
}
private static void ChangeStockLevel(string id,int newStockLevel)
{
    using (var session = docStore.OpenSession())
    {
       Product product = session.Load<Product>(id);
       if(product!=null)
       {
          product.StockLevel = newStockLevel;
          session.SaveChanges();
       }
    }
}
```

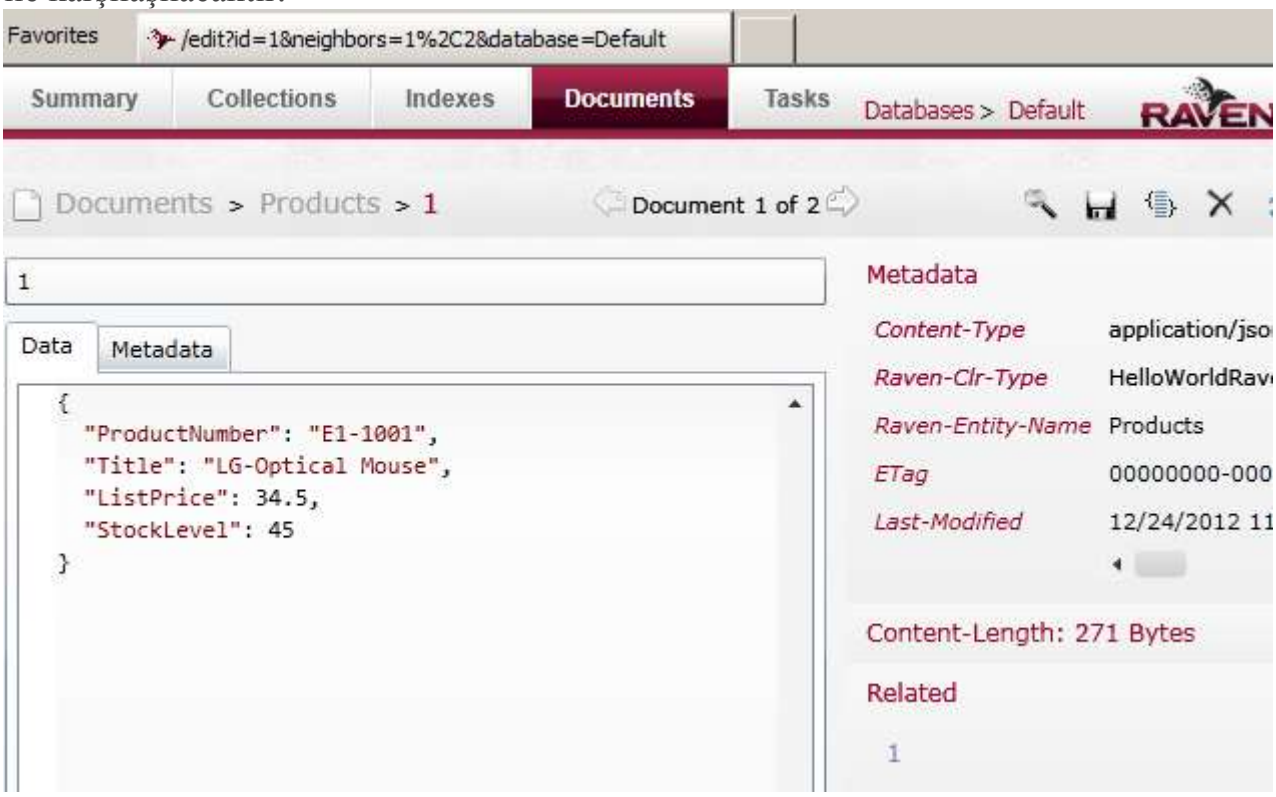
```
private static void DeleteProductByNumber(string productNumber)
{
   using (var session = docStore.OpenSession())
   {
      Product product = session
         .Query<Product>()
         .Where(p => p.ProductNumber == productNumber)
        .SingleOrDefault();
      if (product != null)
     {
         session.Delete(product);
         session.SaveChanges();
      }
   }
}
private static void ListAllProducts()
{
   using (var session = docStore.OpenSession())
   {
      var products = session.Query<Product>()
         .OrderBy(p => p.Title)
        .ToList();
      foreach (var product in products)
      {
         Console.WriteLine("{0}-{1},{2},{3}"
            ,product.ProductNumber
            ,product.Title
            ,product.ListPrice.ToString("C2")
            ,product.StockLevel.ToString());
      }
      Console.WriteLine("");
   }
}
private static Product FindProductByNumber(string productNumber)
{
   using (var session = docStore.OpenSession())
   {
      return session
         .Query<Product>()
         .Where(p => p.ProductNumber == productNumber)
         .SingleOrDefault();
```

```
        }
      }
   }
   class Product
   {
      public string Id { get; set; }
      public string ProductNumber { get; set; }
      public string Title { get; set; }
      public double ListPrice { get; set; }
      public int StockLevel { get; set; }
   }
}
```

Daha önceden **Entity Framework** veya **LINQ to SQL** ile çalıştıysanız eğer, kodlardaki yaklaşım oldukçta tanıdık gelecektir. İlk olarak genel bir **Context** tipine ihtiyacımız bulunmakta. Bunun için**DocumentStore** tipinden yararlanılmaktadır.
DocumentStore örneklenirken **ConnectionStringName** özelliğine **app.config** dosyasında yer alan **key** değeri verilmiştir. Sonrasında gerçekleştirilen veri çekme, ekleme, silme ve güncelleştirme işlemlerinin tamamında ise **OpenSession** metoduna yapılan çağrı ile elde edilen**IDocumentSession** arayüzü türevli referans tipi kullanılmaktadır.
Veri çekme işlemlerinden de dikkat edileceği üzere **LINQ** metodlarından yararlanılmaktadır. Sorgulamalar için başlangıç noktası **Query<T>** metodudur. Bunun dışında bir veriyi **Key** değeri üzerinden elde etmek istersek(*ki Product sınıfındaki string türünden Id özelliği bunun için eklenmiştir)* **Load<T>** metodundan yararlanılabilinir. Veri ekleme için **Store**, silme işlemi içinse**Delete** fonksiyonları kullanılmıştır. Elbette yapılan tüm veri ekleme, silme ve güncelleştirme işlemlerinin, döküman içerisine yazılması **SaveChanges** metoduna yapılacak çağrı ile mümkün olmaktadır.
Uygulamayı çalıştırdığımzda, 3 adet **Product** örneğinin eklendiğini, bir tanesinin güncelleştirildiğini ve bir diğerinin de silindiğini analiz edebiliriz. Ayrıca tüm bu işlemler Web arayüzü üzerinden de anlık olarak takip edilebilirler. Örneği çalıştırdıktan sonra **http://localhost:8081/raven/studio.html**adresine gidersek aşağıdaki ekran görüntüsü ile karşılaşırız.



1 numaralı ürün üzerinde durulduğunda ise verinin **JSON** formatındaki karşılığı da rahat bir şekilde gözlemlenebilir. Ayrıca herhangibir ürün açıldığında aşağıdaki ekran görüntüsü ile karşılaşılacaktır.



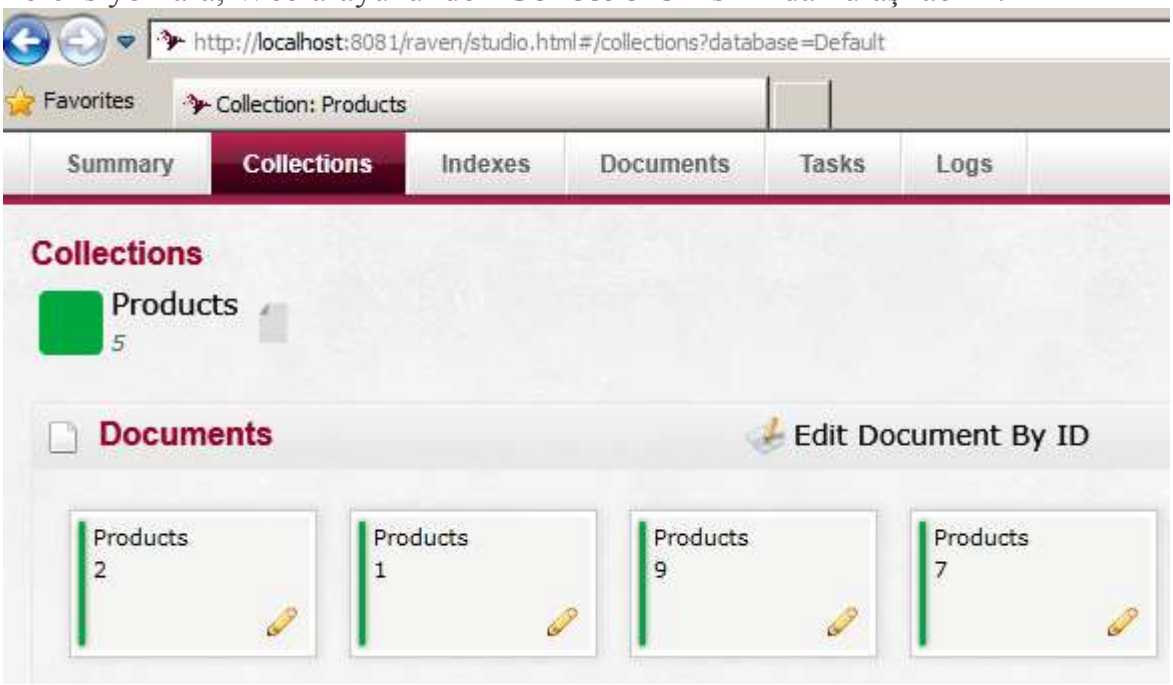
Mutlaka sağ tarafta yer alan **ETag** değeri de dikkatinizi çekmiştir. Dilerseniz verilerinizi etag ile ilişkilendirebilirsiniz. **Store** metodunun aşırı yüklenmiş versiyonlarında **GUID** tipinden **etag**değerlerinin girilebilmesine de izin verilmektedir.

```
using (var session = docStore.OpenSession())
{
    session.Store(newProduct,);
    ▲ 4 of 4 ▼ void IDocumentSession.Store(object entity, Guid etag, string id)
}
```

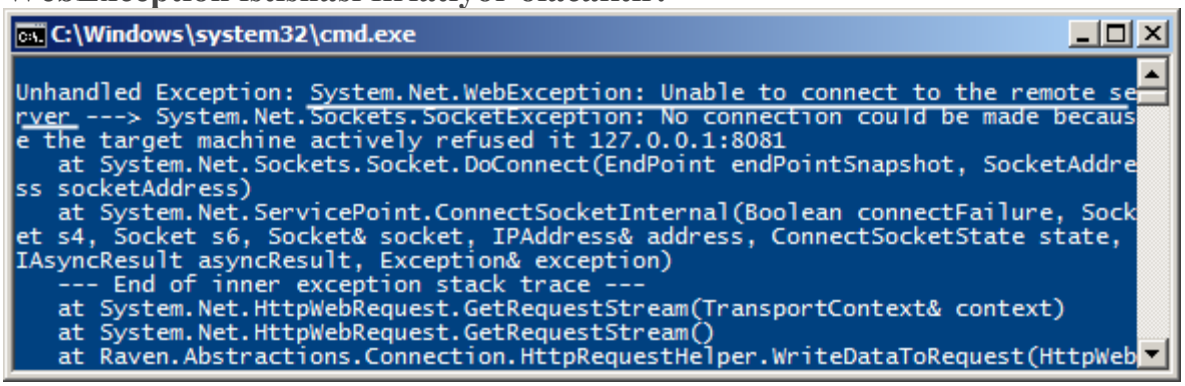
veya

```
using (var session = docStore.OpenSession())
{
    session.Store(newProduct,);
    ▲ 3 of 4 ▼ void IDocumentSession.Store(object entity, Guid etag)
}
```

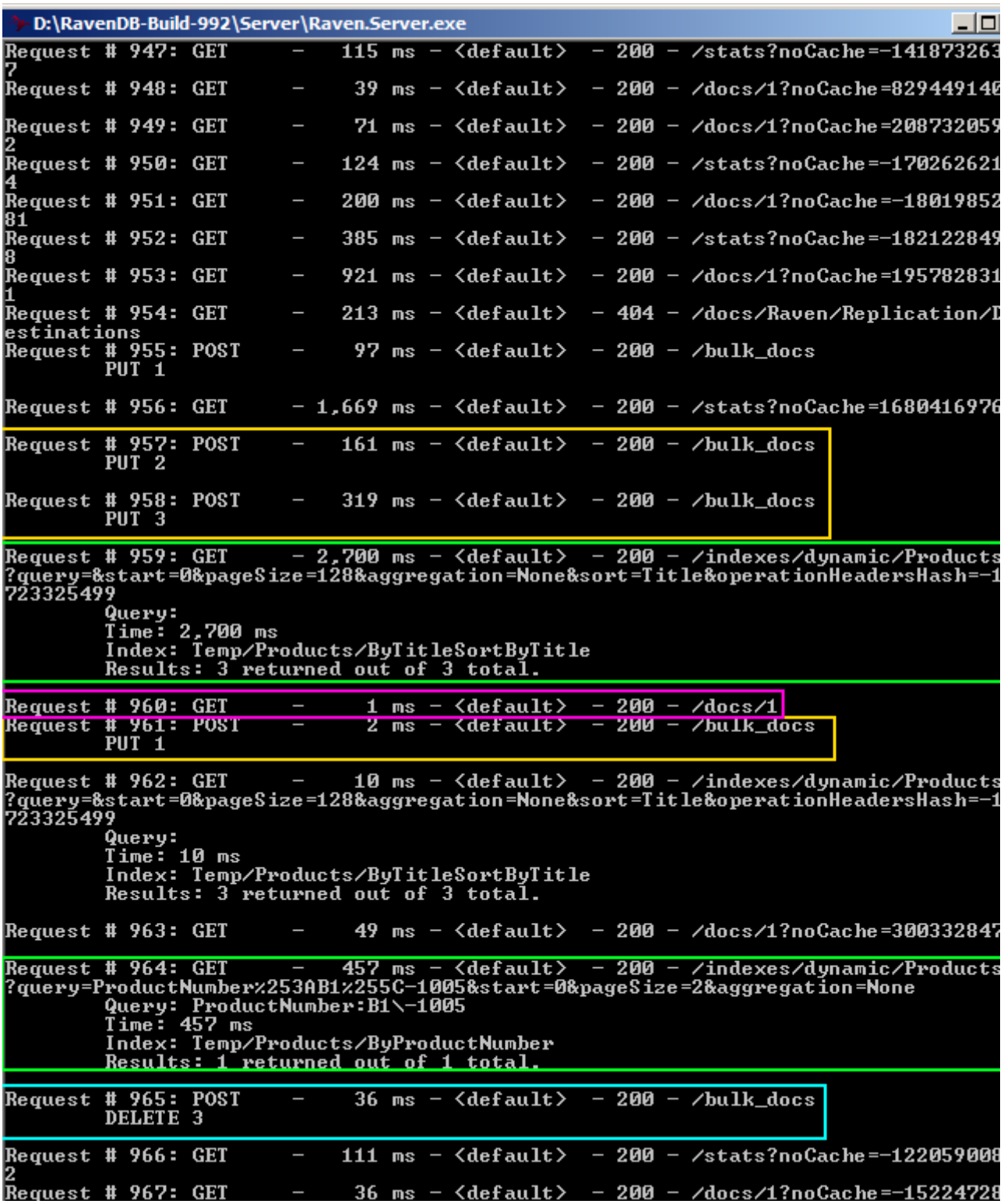
Eklenen bütün ürünler **Product** isimlidir ve **RavenDb** tarafından isim çoğullama yapılarak **Products**adındaki koleksiyon içerisine dahil edilmişlerdir. Bu ve varsa diğer koleksiyonlara, Web arayüzündeki**Collections** kısmından ulaşılabilir.



Çok doğal olarak server etkin değilse istemci tarafı, çalışma zamanına bir WebException istisnası fırlatıyor olacaktır.



Diğer yandan uygulama çalıştırılmadan önce, çalıştığı süre zarfı içinde ve sonrasında,**RavenDb.Server.exe** programının komut satırına bazı loglar attığına şahit oluruz. Aynen aşağıdaki ekran görüntüsünde yer aldığı gibi.



Dikkat edileceği üzere çeşitli **HTTP** metodları söz konusu olmuştur. Veri çekme işlemlerinde **GET**, ekleme işlemlerinde **POST** ve silme işlemlerinde de **DELETE** metodlarına ilişkin talepler*(Request)*oluşmuştur.

Bu arada **960** numaralı talebi dilerseniz **URL** den manuel olarak girmeyi deneyebilirsiniz.**http://localhost:8081/docs/1** şeklinde bir talep gönderdiğimizde, indeks değeri **1** olan içeriğin**JSON** formatlı çıktısına ulaşırız.

{"ProductNumber":"E1-1001","Title":"LG-Optical Mouse","ListPrice":34.5,"StockLevel":45}

Bu çok doğal olarak RavenDB' nin **RESTful** servis desteği sunmasından kaynaklanmaktadır.

Görüldüğü üzere **RavenDB**' yi kullanmak oldukça basittir. Özellikle **.Net** geliştiricilerin aşina olduğu teknikler söz konusudur*(LINQ metodlarının kullanılması gibi)*. Üstelik doğrudan **POCO*(Plain Old Clr Object)*** tipleri ile çalışılabilinir. Dolayısıyla kullanışlı bir **NoSQL** ürünü olduğunu ifade edebiliriz

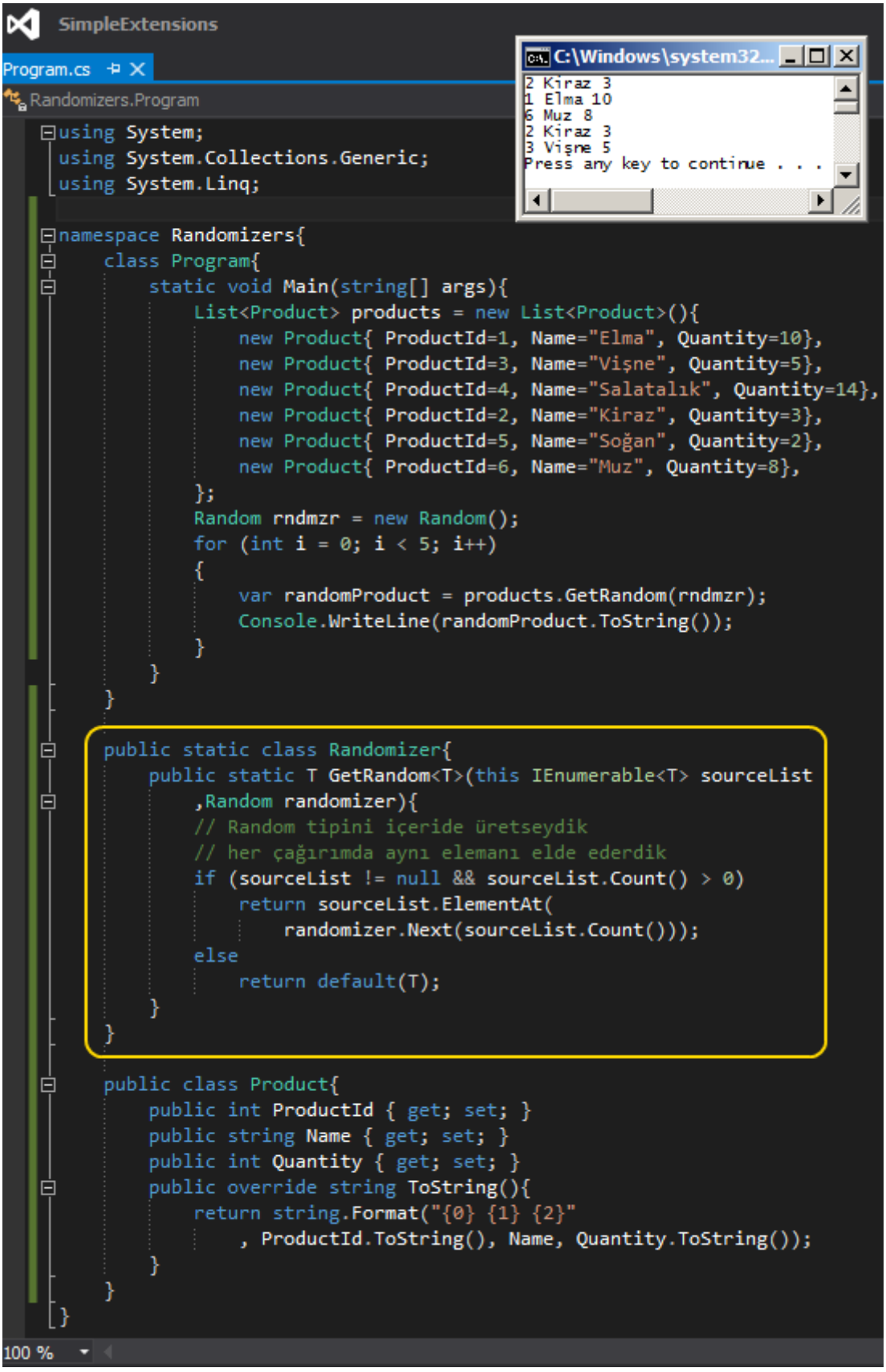
Size tavsiyem söz konusu örnekte yer alan kodlardan yola çıkarak, örneği **Web/Windows**platformuna taşımanız olacaktır. Hatta hali hazırda kullanmakta olduğunuz minik ve veritabanı odaklı çalışan bir uygulamanız var ise, bunu **RavenDB** ile çalışacak şekilde yeniden kurgulamayı deneyebilirsiniz. Farklı **NoSQL** veritabanlarını inceledikçe kullanımlarını sizlerle paylaşmaya çalışıyor olacağım. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HelloWorldRavenDB.zip (1,08 mb)

Tek Fotoluk İpucu 75–LINQ ile Rastgele Eleman Çekmek

Cuma, 4 Ocak 2013 11:15 by bsenyurt

Merhaba Arkadaşlar,
Pek çoğumuz **Random** tipini kullanır ve bir listeden rastgele elemenalar üretmeye veya elde etmeye çalışırız. Peki **T** tipinden bir listeden herhangibir anda rastgele eleman almak isteseniz ve bunu bir**Extension** metod olarak tasarlamayı planlasanız...Nasıl bir yol izlerdiniz? Aşağıdaki gibi olabilir mi mesela?



Bir başka ip ucunda görüşmek dileğiyle.

TFS 32Bit Uygulama Hatası (Bir Garip Geliştiricinin Haykırışı)

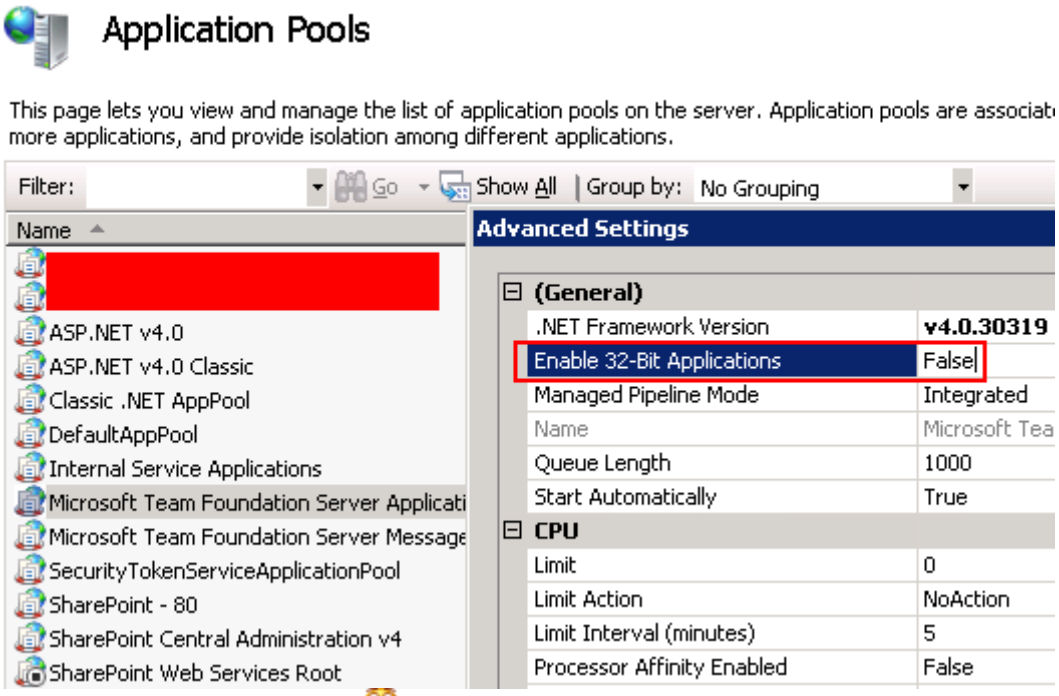
Çarşamba, 2 Ocak 2013 00:52 by bsenyurt

Merhaba Arkadaşlar,
Genelde bu kadar kısa yazılar pek yazmıyorum. En fazla**Tek Fotolok İpucu** serisi altında paylaşım yapmaktayım. Ancak karşılaştığım ilginç bir durumu da sizinle paylaşmak istedim. Tabi olayın başrol oyuncusu olarak en büyük kabahat bende Öyleyse haydi buyrun bakalım hiyayemize…
Biliyorsunuz **TFS** kurduğunuzda **IIS** alına bir **Team Foundation Server** isimli bir **Web Site** oluşturulmakta*(Web Access arayüzü buradaki **tfs** klasörü altında duruyor ve hatta **TFS servisleri** de yine buradaki **TeamProjectServices** uygulaması içerisinde yer almakta)*
Web Site' ın en belirgin özelliği ise **Microsoft Team Foundation Server Application Pool** isimli bir havuzu kullanıyor olması. Bu havuzun özelliklerine genellikle pek dokunmuyoruz ama ben bir test sırasında dokundum ve bakın neler oldu. Lafı fazla uzatmadan hemen senaryoya geçeyim dilerseniz
O gün elimde geliştirmelerini yeni bitirdiğim ve yerel makinede test ettiğim bir **WCF Servis**uygulaması vardı ve ağda ilk bulabildiğim sunucu üzerinde de test etmek istiyordum. Erişim hakkım olan ve üzerinde **TFS** yüklü makinemi gözüme kestirdim. Bu arada söz konusu **WCF servis**uygulaması içerisinde, sadece **32bit uyumlu olan bir Assembly** da kullanmaktaydım*(Oracle.DataAccess.dll)*.
Hemen bir deploy paketi oluşturdum ve servis uygulamasını **IIS** üzerinde bir **Web Site**' a atmak istedim. Ancak **Default Web Site**, **Sharepoint**' in **80** numaralı portu hakimiyeti altına alması nedeniyle kaput durumdaydı ve hiç hayat belirtisi vermiyordu. En uygun yer **Team Foundation Server** isimli **Web Site** idi. Ben de bunun üzerine paketi alıp **Team Foundation Server** sitesi altında bir **Application** haline dönüştürdüm.
Ancak servise erişmek istediğimde **Oracle.DataAcesss.dll** ile ilişkili bir hata aldım.

Server Error in '/' Application.

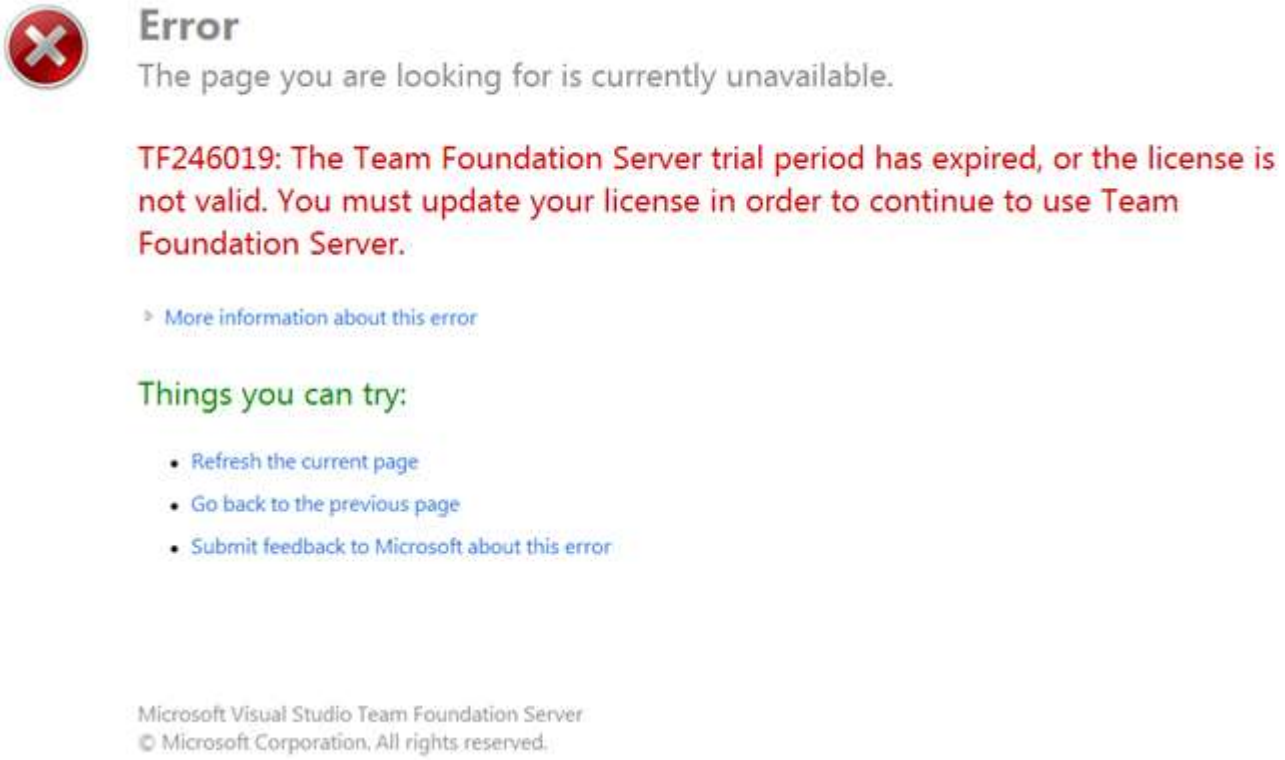
Could not load file or assembly 'Oracle.DataAccess' or one of its dependencies. An attempt was made to load a program with an incorrect format.

Hatanın sebebi belirgindi. Makine üzerindeki işletim sistemi **64bit** olarak yüklenmişti ve **IIS**' de bu şekilde yürütülmekteydi*(Zaten TFS 2012' de 64bit işletim sistemi üzerine kurulmaktadır. Detaylar için bu adresteki yazıya bakabilirsiniz)* Dolayısıyla **Microsoft Team Foundation Server Application Pool**' un **32bit** yazılmış **assembly**' ları yükleyebilmesi bu senaryo için gerekiyordu. Ben de ilgili **Pool**' un **Advanced Settings** kısmına giderek **Enable 32-Bit Applications** değerini**true** olarak değiştirdim. Bu masumane davranışın nasıl kötü bir sonucu olabilirdi ki



Hay değiştirmez olaydım

Artık servis çalışıyordu bunu görebiliyordum ama…Kendi kendimi bir sonraki adıma geçmeyi planladığım sırada bir telefon sesi duydum. Takım arkadaşlarımdan birisi **TFS** bağlantısı sırasında bir hata aldığını söylüyordu. Hemen TFS adresine girdim ve ben de hatayı gördüm



Bir anda ortalık karıştı tabi. Telefonlar ardı ardına geliyor, ter damlaları heryerden boşalıyordu. **TFS**ile çalışan çok fazla ekip vardı. Ürün lisanslıydı. Bunu biliyorduk. Emin

olmak için **IT** departmanımız ile görüştük. Doğruladılar. Lisans numaralarını kontrol ettik vs…
Sonunda oluşan hatanın sebebinin **Enable 32-Bit Applications** değerinin **true** olması olduğunu anladık. Nitekim **64bit** işletim sistemi üzerinde kurulmuş olan TFS, her nedense bu değişikliği lisans ihlali gibi algılamıştı*(Öyle tahmin ediyorum)*
O yüzden siz siz olun, mutlaka servislerinizi test etmek için ayrı bir IIS sunucusunun tahsis edilmesini isteyin
Başıma gelen başka bir garip olayda görüşmek dileğiyle hepinize mutlu günler dilerim.

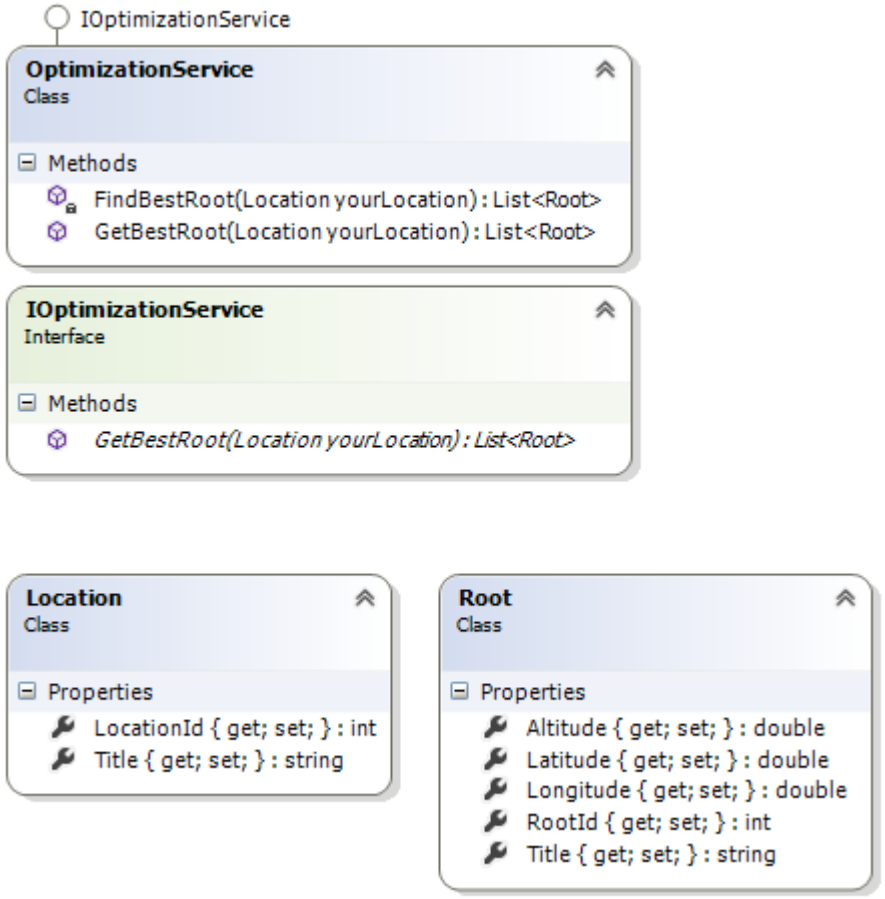
WCF 4.5–Task Based Asynchronous Operasyonlar

Salı, 1 Ocak 2013 22:51 by bsenyurt

Merhaba Arkadaşlar,
Yaklaşık olarak **4 dakika 38 saniye**…İzleyen yazıyı benim okuma hızım bu oldu. Aslında bu süre şu demek; Öğle arasına çıkmadan bir 4 dakika 38 saniye demek bu...Ya da geldikten sonra bir 4 dakika 38 saniye demek…Ya da sabah işe erken geldiğimizde ayırabileceğimiz bir 4 dakika 38 saniye demek...Ya da servisi/otobüsü/minibüsü beklerken ayırabileceğimiz 4 dakika 38 saniye demek. Hatta Facebook' a, Twitter' a, Linkedin' e, Youtube'a bakmadan geçireceğimiz bir 4 dakika 38 saniye demek… E o halde ne duruyorsunuz?Ayırın işte o zamanı

Bilindiği üzere **.Net Framework 4.5** ile birlikte altyapıya entegre olan **async** ve **await** anahtar kelimelerini kullanarak**, task bazlı asenkron** programlama teknikleri uygulanabilmektedir. Çok doğal olarak **WCF 4.5** tarafında da bunun bir yansımasını görmekteyiz. **Visual Studio** arabirimi üzerinden herhangibir **WCF** servis referansını istemci uygulamaya eklemeye çalıştığımızda **Task**bazlı operasyon desteği varsayılan olarak etkinleştirilmekte ve **proxy** tipi içeriğinde buna uygun metodlara yer verilmektedir. Dolayısıyla **WCF*(Windows Communication Foundation)*** servislerini kullanan istemciler, operasyon çağrılarında **async** ve **await** anahtar kelimelerinden de yararlanabilirler.

Servislerin asenkron çağrılar ile yürütülmesi, özellikle **User Experience**' ın önemli olduğu uygulamalarda, ön planda yer alan konular arasındadır. Gelin bu konuyu oldukça basit bir örnek üzerinden ele almaya çalışalım. İlk olarak **.Net Framework 4.5** versiyonunda bir **WCF Service Application** oluşturup içerisine aşağıdaki sınıf diagramında*(Class Diagram)* görülen tipleri ilave edelim.



OptimizationService sembolik olarak uzun süren bir optimizasyon işlemini üstelenecek şekilde planlanmıştır. Senaryo gereği istemciden bir lokasyon bilgisi almakta olan servis, bu lokasyon için en ideal yolu çıkartmaktadır. Sadece hayal ediyoruz tabi Amacımız uzun süren bir işlem ile servis tarafının istemciye geç cevap dönmesini sağlamak ve asenkronluğu devreye almaktır.

OptimizationService içerisinde yer alan **GetBestRoot** operasyonu **Location** tipinden bir parametre alırken, geriye de **Root** tipinden generic bir **List** koleksiyonu döndürmektedir. Servis uygulamasındaki tiplerin içerikleri ise aşağıdaki gibidir.

Servis sözleşmesi(Service Contract);

```
using System.Collections.Generic;
using System.ServiceModel;
namespace AzonServiceApp
{
   [ServiceContract]
   public interface IOptimizationService
   {
     [OperationContract]
      List<Root> GetBestRoot(Location yourLocation);
   }
}
```

Servis sınıfı;

```
using System.Collections.Generic;
using System.Threading;
namespace AzonServiceApp
{
   public class OptimizationService
      : IOptimizationService
   {
      public List<Root> GetBestRoot(Location yourLocation)
      {
         List<Root> roots=FindBestRoot(yourLocation);
         return roots;
      }
      private List<Root> FindBestRoot(Location yourLocation)
      {
         Thread.Sleep(10000);
         return new List<Root>{
            new Root{RootId=1,Latitude=34.5,Longitude=43.2,Altitude=500.50,Title="4ncü
cadde batı köşesi"},
            new Root{RootId=2,Latitude=34.85,Longitude=43.2,Altitude=450,Title="moda
sahil yolu"},
            new Root{RootId=3,Latitude=22.5,Longitude=43.2,Altitude=100,Title="iskele
caddesi durağı"},
            new
Root{RootId=4,Latitude=44.90,Longitude=12.90,Altitude=0,Title="iskelenin kendisi"}
         };
      }
   }
}
```

Location tipi;

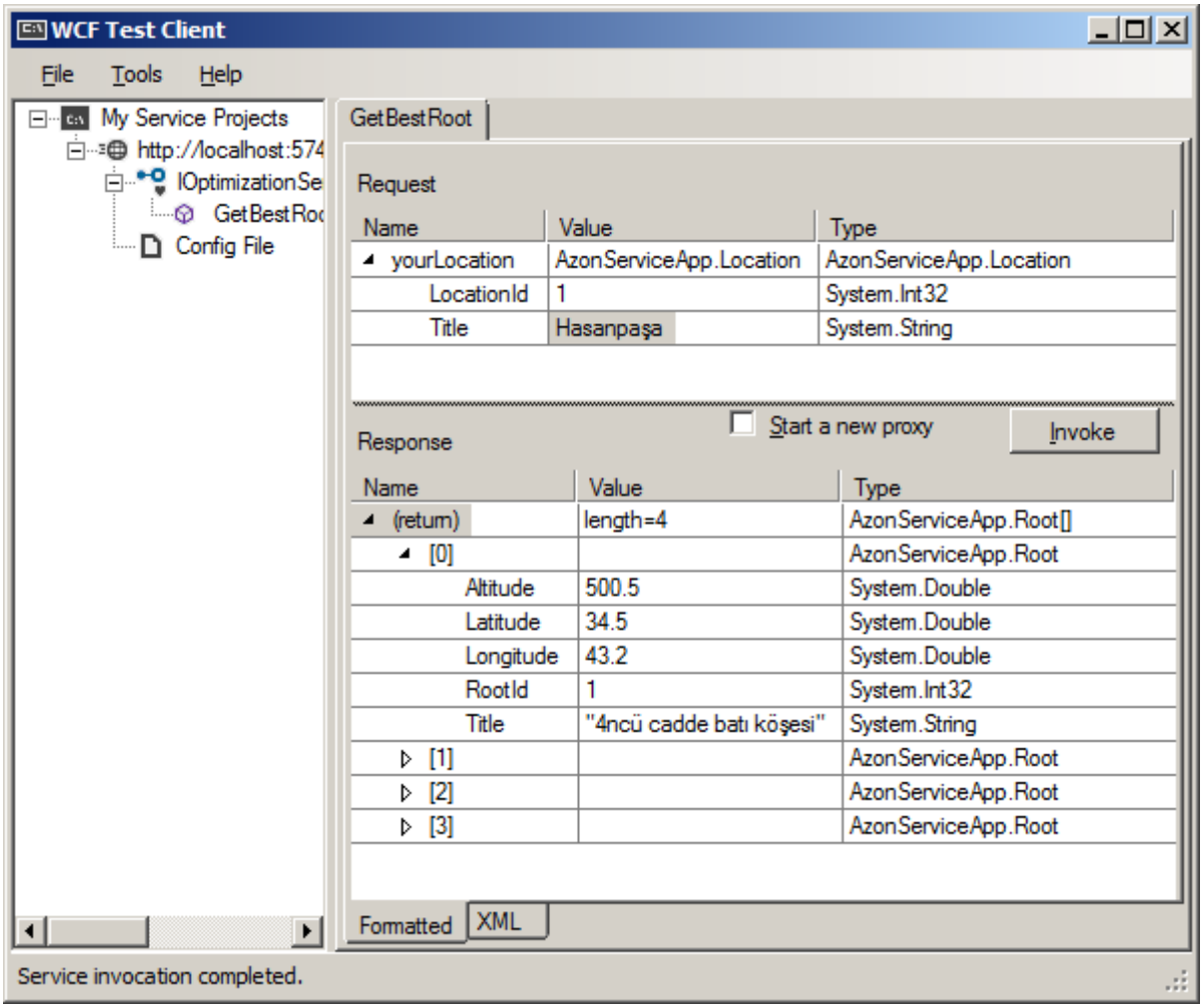
```
using System.Runtime.Serialization;
namespace AzonServiceApp
{
   [DataContract]
   public class Location
   {
      [DataMember]
      public int LocationId { get; set; }
      [DataMember]
      public string Title { get; set; }
```

```
    }
}
```

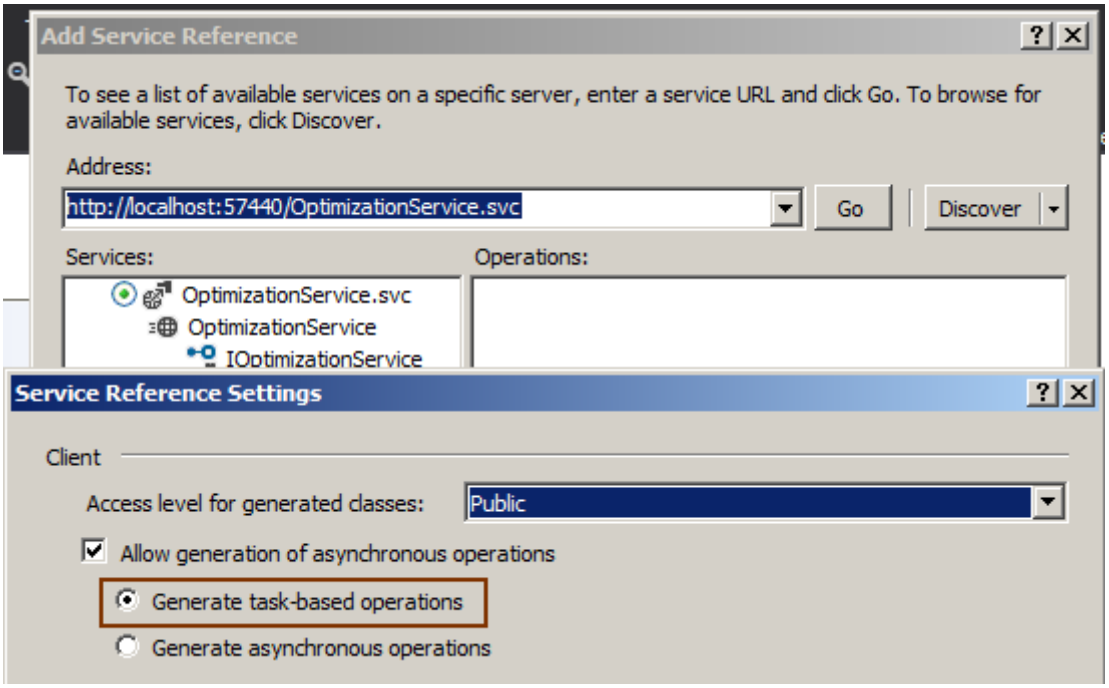
Root tipi;

```
using System.Runtime.Serialization;
namespace AzonServiceApp
{
    [DataContract]
    public class Root
    {
        [DataMember]
        public double Altitude { get; set; }
        [DataMember]
        public double Longitude { get; set; }
        [DataMember]
        public double Latitude { get; set; }
        [DataMember]
        public int RootId { get; set; }
        [DataMember]
        public string Title { get; set; }
    }
}
```

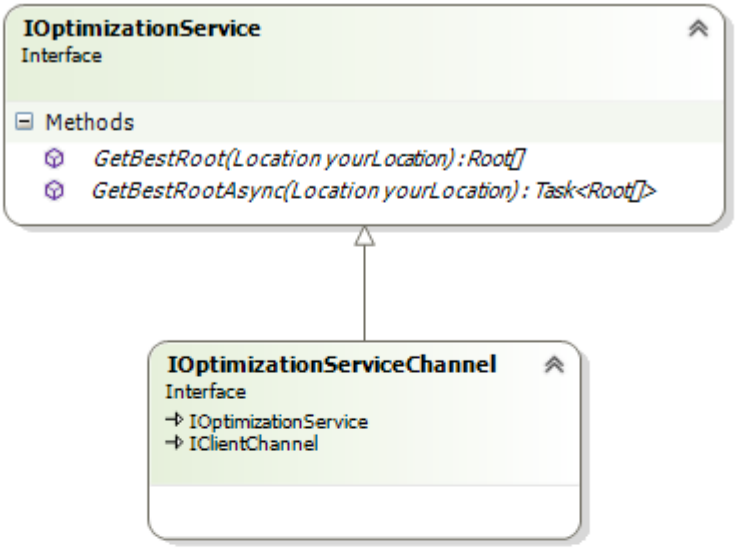
Eğer servis uygulamasını bu haliyle çalıştırıp test edersek, **WCF Test Client** uygulamasında yaklaşık olarak **10** saniyelik bir gecikme ile **Root** listesini alabildiğimizi görürüz*(Nitekim **GetBestRoot** servis operasyonu içerisinde çağırılan **FindBestRoot** metodunda, **Thread.Sleep** ile **10 saniyelik bir gecikme** uygulanmıştır)*

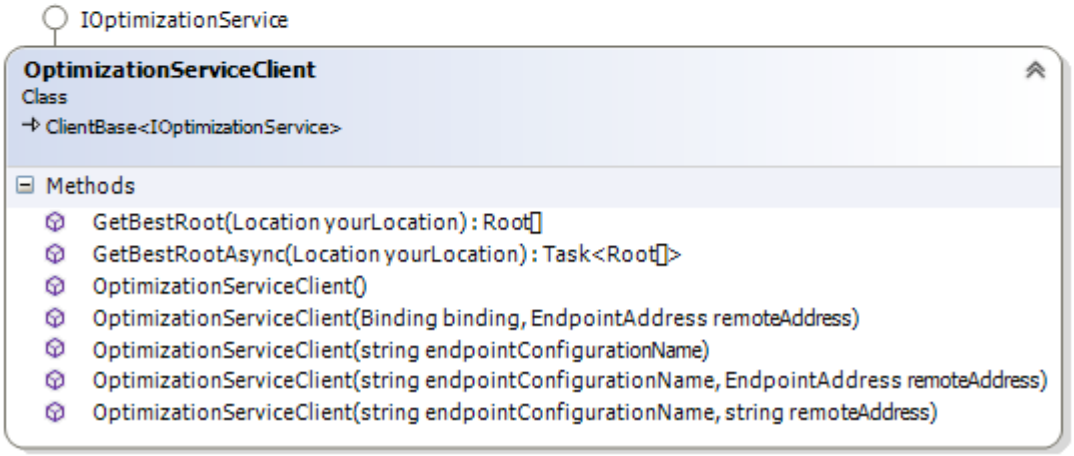


Tabi asıl konumuz bizim geliştireceğimiz istemci uygulamalardaki task bazlı operasyon desteğidir. İstemci tarafını bir **WPF Application** olarak geliştirdiğimizi düşünebiliriz. Uygulamaya **Add Service Reference** seçeneği ile servisimizi eklemek istediğimizde, **Advanced** sekmesinden ulaşacağımız arabirimde yer alan **Generate Task-Based Operations** kutucuğunun varsayılan olarak işaretli olduğunu fark edebiliriz.

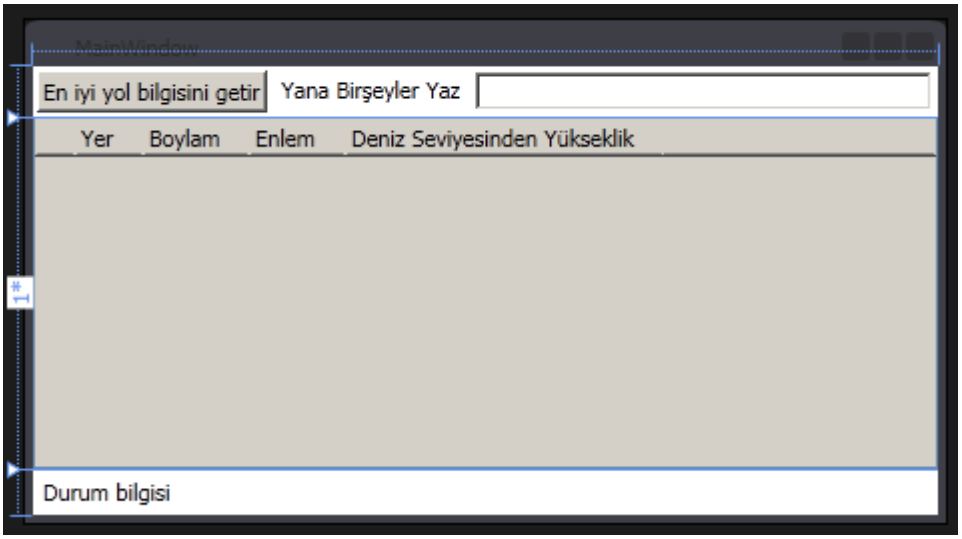


Bu duruma göre referansı eklediğimizde, istemci uygulama tarafında aşağıdaki sınıf diagramında yer alan tiplerin üretildiğini görürüz.





Dikkat edileceği üzere **OptimizationServiceClient** sınıfı içerisinde, geriye **Task<Root[]>** tipinden referans döndüren bir operasyon yer almaktadır; **GetBestRootAsync**. Bu dönüş tipi nedeniyle ilgili metod çağrısı **awaitable'** dır. Dolayısıyla **async** ile işaretlenmiş bir metod içerisindeyken **await** ile çağırılabilir. Dilerseniz bu durumu test etmeye çalışacak şekilde arayüzümüzü geliştirmeye devam edelim. Bu amaçla, **WPF*(Windows Presentation Foundation)*** uygulamamızda yer alan**MainWindow** öğesinin **XAML*(eXtensible Application Markup Language)*** içeriğini aşağıdaki gibi düzenleyelim.




```
<Window x:Class="WpfClientApp.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="MainWindow" Height="250" Width="460">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="auto"/>
            <RowDefinition Height="*"/>
            <RowDefinition Height="auto"/>
        </Grid.RowDefinitions>
        <StackPanel Orientation="Horizontal" Grid.Row="0">
            <Button x:Name="btnGetRoot" Content="En iyi yol bilgisini getir"
                    HorizontalAlignment="Left"
Margin="2,2,2,2" Click="btnGetRoot_Click_1"/>
            <Label Content="Yana Birşeyler Yaz"/>
            <TextBox Margin="3,3,3,3" Width="228"/>
        </StackPanel>
        <Label x:Name="lblStatus" Grid.Row="2" Content="Durum bilgisi"/>
        <DataGrid x:Name="grdRoots" Grid.Row="1" ItemsSource="{Binding}"AutoGe
nerateColumns="False">
            <DataGrid.Columns>
                <DataGridTextColumn Binding="{Binding Path=RootId}" Header=""/>
                <DataGridTextColumn Binding="{Binding Path=Title}" Header="Yer"/>
                <DataGridTextColumn Binding="{Binding
Path=Longitude}" Header="Boylam"/>
                <DataGridTextColumn Binding="{Binding
Path=Latitude}" Header="Enlem"/>
                <DataGridTextColumn Binding="{Binding Path=Altitude}" Header="Deniz
Seviyesinden Yükseklik"/>
```

```
        </DataGrid.Columns>
      </DataGrid>
   </Grid>
</Window>
```

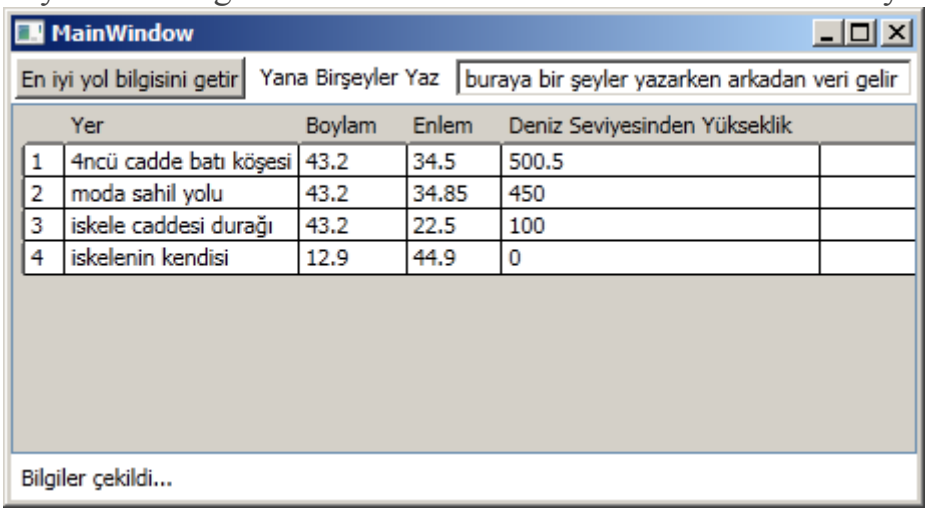
Burada yer alan **DataGrid** kontrolünün içeriğini, servis üzerinden yapacağımız çağrı sonrası gelen**Root[]** referansı ile doldurmaya çalışıyor olacağız. Bu sebepten bir **data bind** işlemi uyguladık ve**DataGrid** kontrolünün kolonlarında da **Root** tipine ait özelliklere yer verdik*(RootId, Title, Longitude, Latitude ve Altitude)* Gelelim yazımızın can alıcı noktasına Kod içeriğini aşağıdaki gibi düzenleyelim.

```
using System.Windows;
using WpfClientApp.AzonReference;
namespace WpfClientApp
{
   public partial class MainWindow : Window
   {
      OptimizationServiceClient proxy = new OptimizationServiceClient();
      public MainWindow()
      {
         InitializeComponent();
      }
      private async void btnGetRoot_Click_1(object sender, RoutedEventArgs e)
      {
         lblStatus.Content = "Bilgiler çekiliyor...";
         Root[] roots = await proxy.GetBestRootAsync(
            new Location {
               LocationId = 1
               , Title = "Hasanpaşa"
            }
            );
         grdRoots.DataContext = roots;
         lblStatus.Content = "Bilgiler çekildi...";
      }
   }
}
```

İlk dikkat çekici nokta **bntGetRoot_Click_1** olay metodunun **async** anahtar kelimesi ile işaretlenmiş olmasıdır. Bu işaretleme nedeniyle, ilgili olay metodu içerisinde asenkron yürütülebilecek bir operasyon çağrısı yapılabileceği de belirtilmiş olmaktadır. Nitekim **Root[]**dizisinin çekilmesi için **GetBestRootAsync** metoduna yapılan çağrıda, **await** anahtar kelimesine yer verilmiştir. Olay metodu başında ve veriler **DataGrid** kontrolüne bağlandıktan sonra da **lblStatus**kontrolü içerisinde kısa bilgilendirmeler yapılmaktadır.

İşin güzel yanı ise şudur; Asenkron olarak çağırılan servis metodunun işleyişi sırasında, **Form**, kullanıcı tepkilerine cevap verebilir durumdadır. Yani ekran üzerinde formu başka bir yere sürükleyebilir, içeride yer alan **TextBox** kontrolünde bir şeyler yazabiliriz

Oysaki eskiden, **Dispatcher**' lardan ve hatta daha eskiden de **Method Invoker**' lardan yararlanarak ekran arayüzünün cevap verebilir olmasını sağlamaya çalışırdık. Kafa karıştırıcı kodlar ile uğraşmak zorunda kalırdık. Bir kontrol için "hadi neyse…" derken, aynı anda yapılması gereken asenkron çağrı sayısının arttığı durumlarda kod kalabalığı ve karmaşıklığını daha da fazlalaştırırdık. Aslında uygulanan yeni model ile basitleşen bu durumu kendi gözlerinizle görmeniz daha iyi olacaktır. Ben sadece bir ekran görüntüsünü koyabilebileceğim. Siz mutlaka örnek kodu indirim test etmeye çalışın.



Görüldüğü üzere **async** ve **await** anahtar kelimelerinden de yararlandığımız bu senaryoda, kod daha az karmaşık olmakla beraber, istemci arayüzünün de asenkron işleyiş sırasında cevap verebilir olması sağlanmıştır. **WCF** operasyonlarının **Windows Phone 8***(Şu adresteki tartışmaya da bir kulak verin)* gibi cevap verebilir arayüz ihtiyaçları yüksek olan uygulama çeşitlerinde de kullanıldığı düşünüldüğünde, kazanılan kabiliyetin önemli olduğu aşikardır. Böylece geldik kısa bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HowTo_TaskBasedAsyncOperations.zip (167,69 kb)

Apache Cassandra ve .Net

Pazartesi, 17 Aralık 2012 10:35

Nosql, Apache Cassandra, .Net, Nuget, Not Only Sql

Merhaba Arkadaşlar,
Size bu günkü makale konumuzun yanda fotoğrafı görülen model*(youtube modeli de diyebiliriz)* Cassandra Bankson ile alakalı olduğunu söylemek isterdim ama yakınlarından bile geçmeyeceğiz. *(Zaten araştırırsanız aslında makyaj bidonu ile bu hale geldiğini keşfedeceksiniz)* Başlıktan da anlayacağınız üzere bu günkü yazımızın konusu **.Net**plaftormunda **Apache Cassandra**' yı kullanmak.
Uzun zamandır gündemimde olan konulardan birisi de**NoSQL** veritabanı sistemleri. **Internet** şirketlerinin pek çoğu*(Facebook, Twitter, Youtube, Netflix vb)* **NotOnly SQL** veritabanlarını kullanmakta ve hatta bir kısmının kendi geliştirdikleri **NoSQL** sistemleri bile var. **Amazon, Google** bu noktada öncüler diyebiliriz. **NoSQL** veritabanlarının popüler olmalarının elbette bazı sebepleri var.
Özellikle **RDMS'** lerin tipik özelliklerine ters gelen kabiliyetleri nedeni ile büyük veriler ile çalışılmasında, ölçeklemelerde, performans da öne çıkabilmektedirler. Tabi bu avantajlar, **NoSQL** yapısına uygun veri kümeleri için söz konusudur. Her veri yapısı veya modeli için NoSQL sistemleri uygun olmayabilir. *(Yani RDMS' i terk eden bir dünyadan bahsedemeyiz* 😊*)*
Ancak **NoSQL** sistemler çeşitlilikleri açısından da **RDMS'** lere göre farklılaşmakta ve bu yüzden daha fazla tercih edilir olmaktadır. Bu çeşitler kısaca şunlardır;

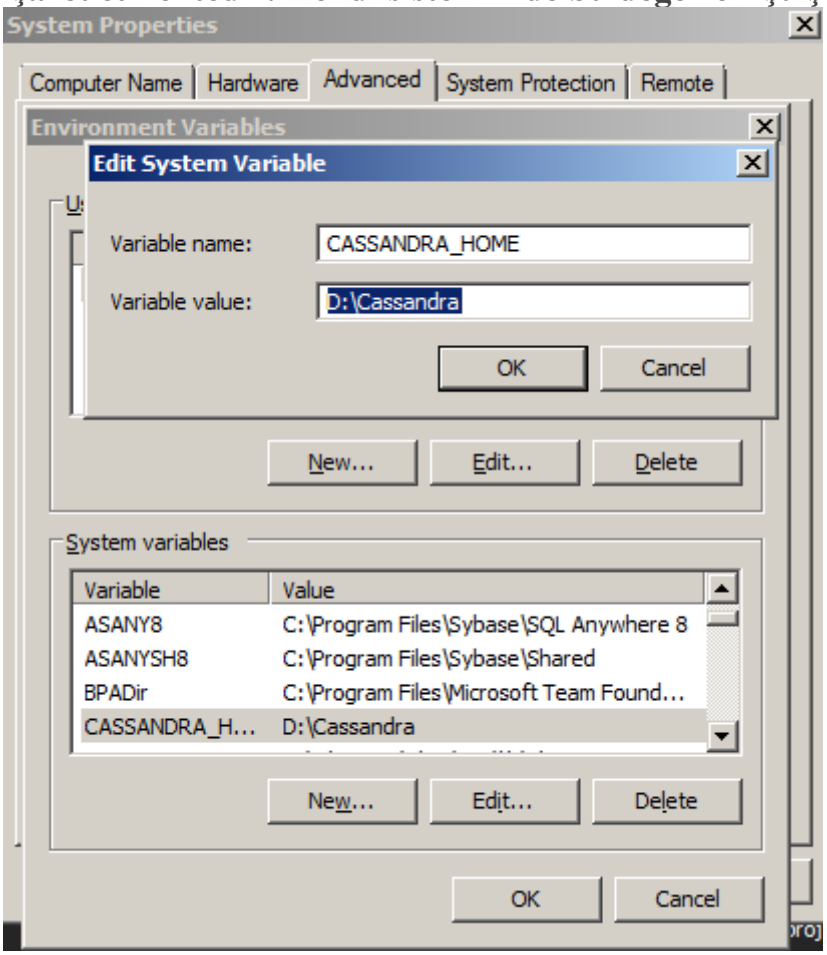
- Wide Column Store / Column Families *(Cassandra bu kategoride gösterilmiştir)*
- Document Store
- Key Value / Tuple Store
- Graph Databases
- Multimodel Databases
- Object Databases
- Grid & Cloud Databases
- XML Databases
- Multidimensional Databases
- Multivalue Database

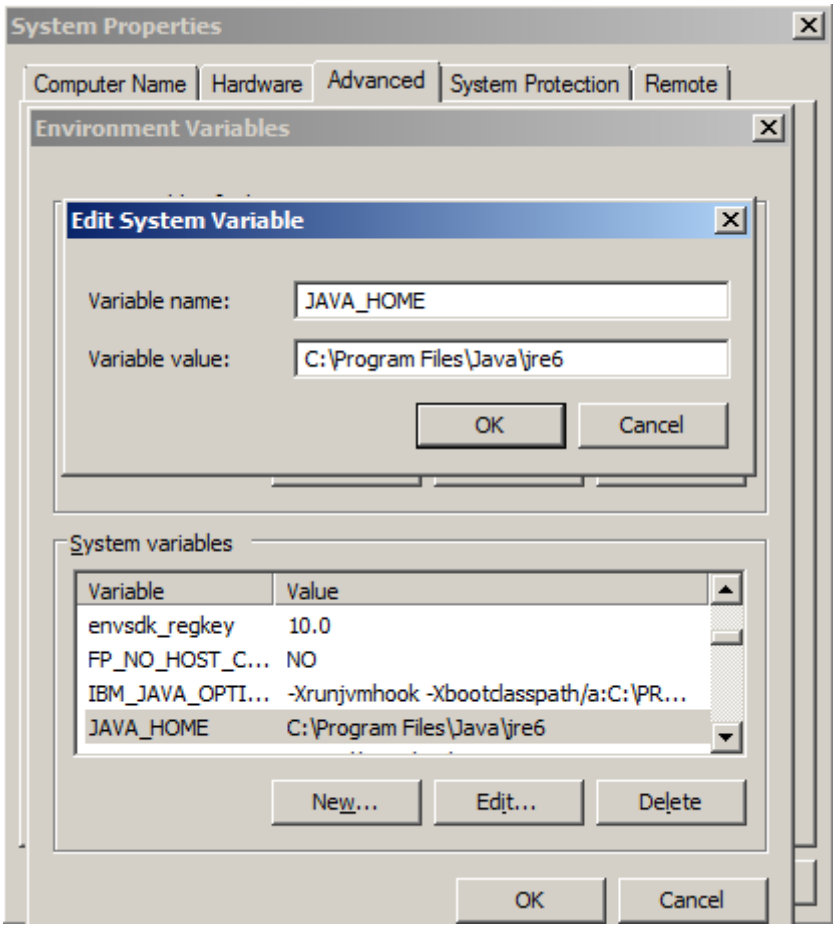
Aslında ortalıkta pek çok **NoSQL** sistemi var. Hatta çoooook uzun zaman önce **Berkley** üniversitesi tarafından üretilen **Berkley_DB** bu işin atasıdır diyebiliriz. Günümüzde ise MongoDb, Neo4j, db4o, bigtable, hadoop, Redis, MemcachedDB vb ürünler bulunmakta. *(Tam ve güncel bir listeye bu adresten ulaşabilirsiniz)* **NoSQL** sistemlerinin popüler olanlarından biriside ismini Yunan

Mitolojisinden aldığını düşündüğüm **Apache**' nin **Cassandra** isimli ürünüdür. Bu yazımızda**Cassandra**' yı **.Net** platformunda nasıl kullanabileceğimizi incelemeye çalışıyor olacağız.

Apache Cassandra' yı http://cassandra.apache.org/download/ adresinden indirebilirsiniz. Cassandra yı indirdiğinizde büyük ihtimalle tar.gz uzantılı bir dosya gelecektir. Dolayısıyla Windows tabanlı bir sistemde bunu açacak uygulamaya ihtiyacınız var. 7Zip bu konuda bana yardımcı oldu diyebilirim.

Cassandra' yı bir klasöre açtıktan sonra sisteme bazı çevresel değişkenlerin de ilave edilmesi gerekmektedir(Environment Variables). Bunlardan birisi JAVA_HOME dur ve sistem de kurulmuş olan Java klasörünü göstermektedir. Diğeri ise CASSANDRA_HOME olarak adlandırılmaktadır ve Cassandra' nın açıldığı klasörü işaret etmektedir. Kendi sistemimde bu değerleri şu şekilde ayarladım.





Kurulum işlemi için gerekli ayarları yaptıktan sonra **bin** klasörü altında yer alan **cassandra.bat**dosyasını çalıştırabiliriz. Bu **batch** dosyası sunucuyu etkinleştirecektir. **Cassandra** verilerin loglarını tutmak için **root** klasördeki **var** isimli alt klasörü kullanır *(Bu kuruluma göre d:\var altındadır ama istendiği takdirde conf klasörü içindeki cassandra.yaml' den konum değiştirilebilir)*

Genel Özellikler

Cassandra' nın genel özelliklerine baktığımızda aşağıdakileri ifade edebiliriz.

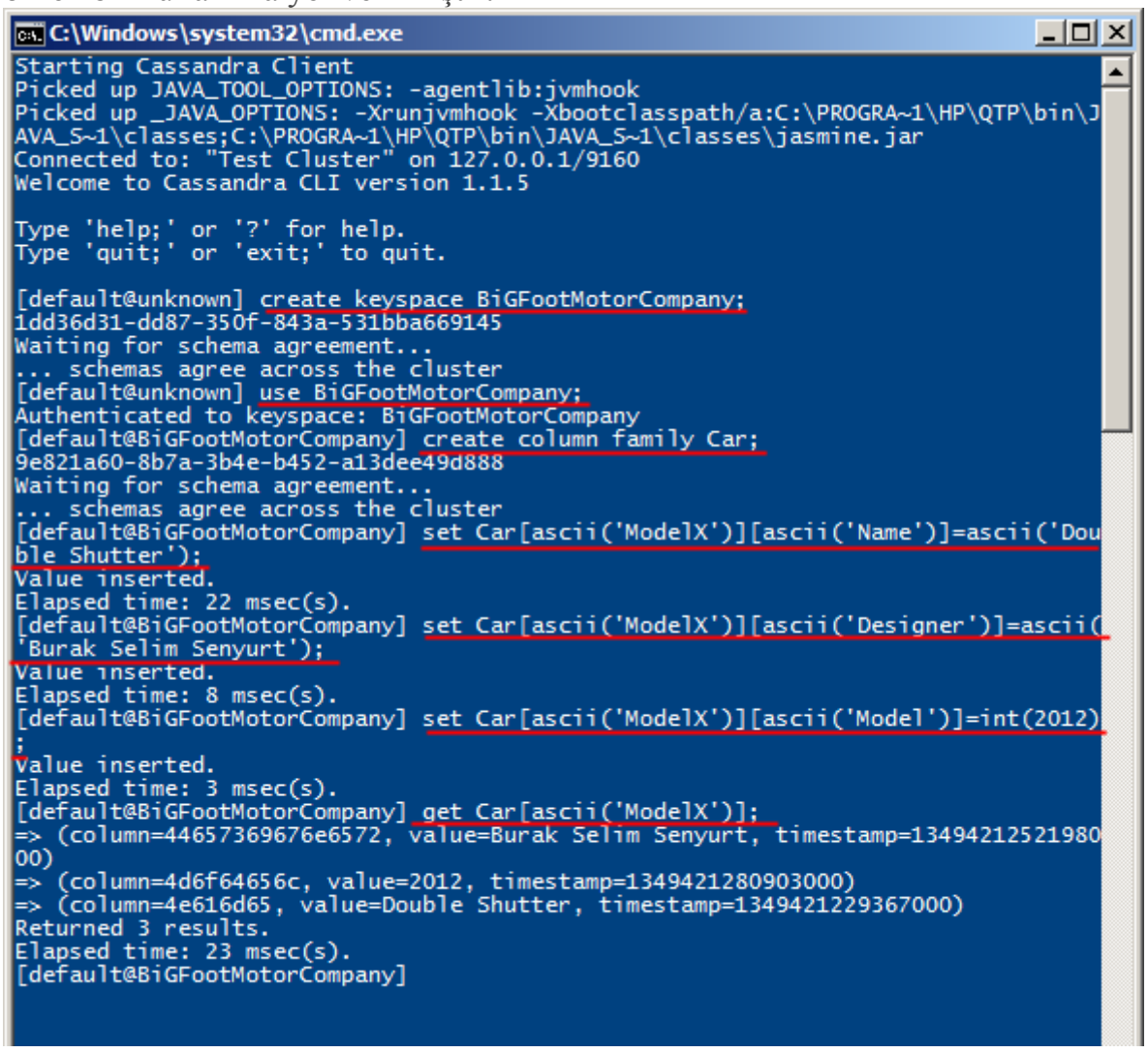
- Java ile geliştirilmiş bir veritabanı sistemidir ve bu nedenle kurulduğu makinede Java ortamının var olması gerekmektedir.
- Bir NoSQL(Not Only SQL) sistemidir. Dolayısıyla SQL, Oracle gibi ilişkisel bir veritabanı modeli değildir.
- Açık kaynaktır(Open Source)
- Bir Oracle veya SQL Server olmasa da Youtube, Netflix gibi pek çok dünya markası tarafından çeşitli ürünlerde kullanılmaktadır.
- Kolayca ölçeklenebilir(Scalable)
- Hata tolerans yönetimi mevcuttur.
- Sütun odaklı*(Wide Column Store/Column Families)* çalışan bir NoSQL tipidir.
- Distribution tasarım modeli Amazon' un Dynamo ürünü esaslıdır.
- Dağıtık modele destek verdiğinden veriyi n sayıda makine üzerinde genişletmek mümkündür. Bu anlamda RDMS(Relational Database Managament Systems) lerin tam aksine Ring düzenine göre çalışır. Bir başka deyişle dikey(Vertical) değil yatay(Horizontal) olarak ölçeklenir.

- Terabyte' larca veriyi tutabilir

Veri modeline üstten bakıldığında yapısı **RDMS** şemalarına benzetilebilir. Sütunlar ve**isimlendirilmiş değerler(Named Values)** söz konusudur. Ancak pratikte hiç de böyle değildir.**Cassandra** bir şema yapısı kullanmamaktadır. Ya bizim aşina olduğumuz tipte bir şema kullanmamaktadır diyebiliriz. Veriyi sütunlar topluluğu halinde tutar. Aslında bu konuda çok fazla detaya girmeyeceğiz.

İlk Çalışma

Cassandra' yı kurup çalıştırdıktan sonra*(cassandra.bat ile yapıyoruz)* komut satırından hemen veri girişi işlemleri yaptırılabilir. Bunun için
yine **bin** dizinindeki **cassandra_cli.bat** dosyasının çalıştırılması yeterlidir. Bu bir komut satırı istemcisidir ve cassandra sunucusuna bağlanacaktır. Aşağıdaki ekran görüntüsünde örnek bir kullanıma yer verilmiştir.



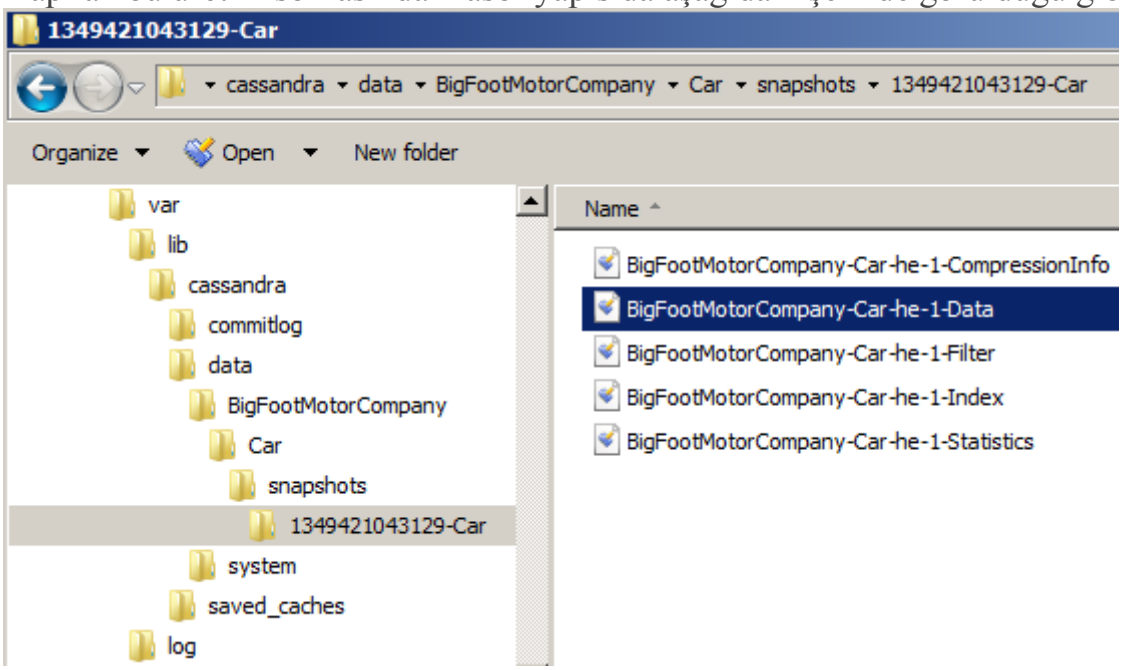
Buradaki komutlardan da anlaşılacağı üzere veritabanı aslında bir **keyspace'** dir.
Bu **keyspace**içerisinde bir tablo oluşturmak aslında bir **Column Family** yaratmak anlamına gelmektedir.**Column Family** içerisine **set** edilen **Row Key'** ler aslında bildiğimiz tablo satırlarına benzetilebilir.**Row Key'** ler içerisinde ise **key-value** şeklinde kolonlar ve verileri bulunmaktadır. Her **key-valuecolumn** aslında bir **Row Key** ile ilintilidir.

Dikkat edileceği gibi kolonlarda **key-value** şeklinde bir tutuluş söz konusudur. Ortam tamamen**case sensitive'** dir ve ifadelerin çalışması için **;** ile bitirilmesi gerekmektedir. Örnekte kullandığımız komutlar ise aşağıdaki gibidir.

İlk Komutlar

- **create keyspace** ile BigFootMotorCompany isimli bir **key space** üretilmiştir.
- **create column family** ile Car isimli bir **Row Key** üretilmiştir.
- **set** ile Car içerisinde sütun(Column) ve **key-value** eklenmiştir.
- **get** ile bir ModelX isimli column verisi çekilmiştir.

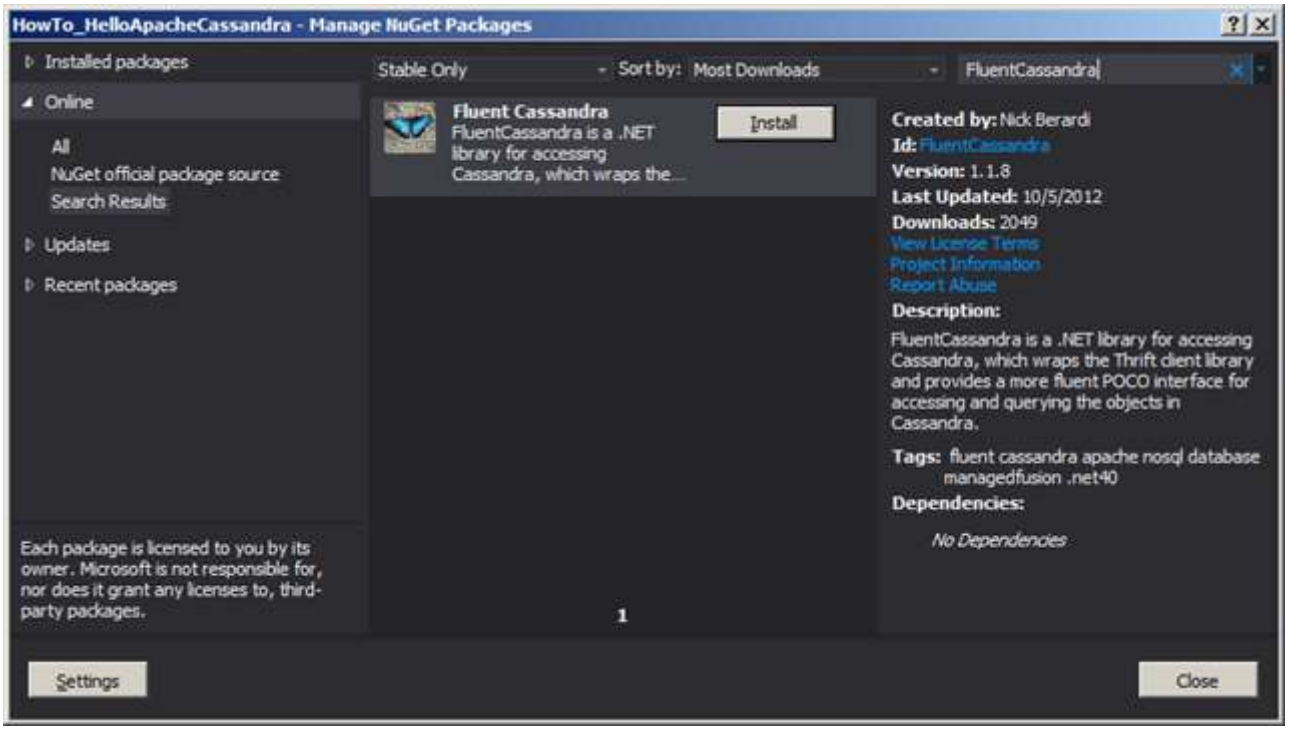
Yapılan bu üretim sonrasında klasör yapısıda aşağıdaki şekilde görüldüğü gibi oluşacaktır.



Veri binary formatta tutulmaktadır.

Gelelim .Net Framework tarafına

.Net tarafında **Cassandra** ile çalışabilmek için **NuGet** paketlerinden birisi olan **FluentCassandra**ile çalışabiliriz. İlk olarak basit bir Console uygulaması açalım ve NuGet paket yöneticisini kullanarak internetten Fluent Cassandra paketini indirelim(Dilerseniz komut satırından da install edebilirsiniz. Install-Package FluentCassandra ifadesini çalıştırmanız yeterli olacaktır)



İlk Kodlar

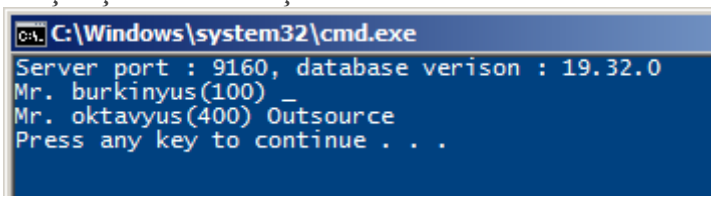
İşte ilk deneme kodlarımız.

```
using FluentCassandra;
using FluentCassandra.Connections;
using System;
namespace HowTo_HelloApacheCassandra
{
   class Program
   {
      static void Main(string[] args)
      {
         // Sunucu ile bağlant kuralım. Varsayılan olarak localhost ismiyle bağlanabiliriz.
         Server cassandraServer = new Server("localhost");
         // Komut satırından ürettiğimiz BigFootMotorCompany isimli key space ile çalışacağız
         using (var database = new CassandraContext(keyspace: "BiGFootMotorCompany", server: cassandraServer))
         {
            // sunucuya ve veritabanına bağlanıp bağlanamadığımız test etmek için bir kaç bilgi talep edelim.
            Console.WriteLine("Server port : {0}, database verison : {1}"
               ,cassandraServer.Port
               , database.DescribeVersion()
               );
            #region Yeni bir Column Family oluşturmak
```

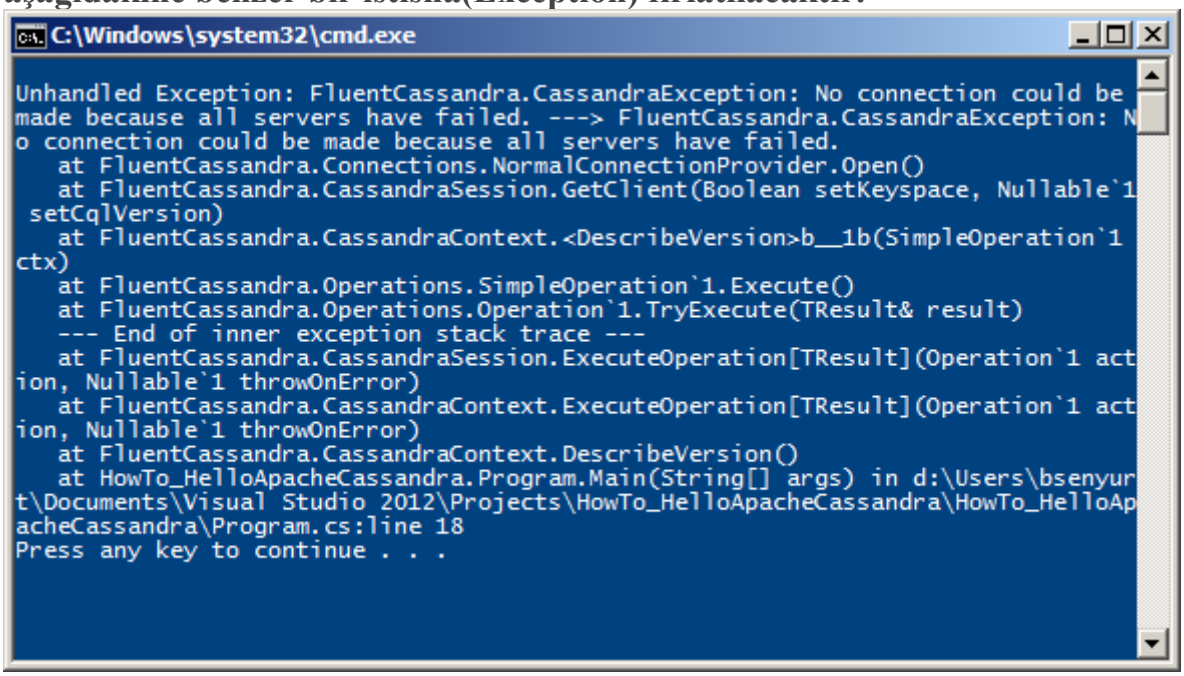
```
            CassandraColumnFamily newFamily =
database.GetColumnFamily("ModelDesigner");
            // Cassandra Query Language kullanarak bir Column Family oluşturuyor ve
içerisine bir kaç column ilave ediyoruz
            // Sonraki denemelerde bu satırda hata vermemesi için bir null kontrolü yapmak
da işe yarayabilir
            if(newFamily==null)
               database.ExecuteQuery(@"
               create columnfamily
               ModelDesigner(
                  ModelDesignerId ascii Primary Key
                  ,Title text
                 ,Nickname text
                  ,Level int
                  ,Outsource boolean);"
               );
            // Yeni bir satır oluşturalım
            // dynamic tipin çalışma zamanında çözümlenmesine neden olacaktır.
            dynamic burkinyus=newFamily.CreateRecord("burkinyus");
            // Key' lerin değerlerini atayalım
            burkinyus.Title = "Mr.";
            burkinyus.Nickname = "burkinyus";
            burkinyus.Level = 100;
            burkinyus.Outsource = false;
            // Yeni oluşturlan satırları Context' e ekleyelim
            database.Attach(burkinyus);
            // bir satır daha oluşturalım ve alanlarını set edelim
            dynamic oktavyus = newFamily.CreateRecord("oktavyus");
            oktavyus.Title = "Mr.";
            oktavyus.Nickname = "oktavyus";
            oktavyus.Level = 400;
            oktavyus.Outsource = true;

            database.Attach(oktavyus);
            // Değişiklikleri kayıt edelim
            database.SaveChanges();
            // şimdi de verileri çekip gösterelim
            var designers = database.ExecuteQuery("select * from ModelDesigner");
            foreach (dynamic designer in designers)
            {
               Console.WriteLine("{0} {1}({2}) {3}"
```

```
                ,designer.Title
                ,designer.Nickname
                ,designer.Level
                ,designer.Outsource==true?"Outsource":"_"
                );
            }
            #endregion
        }
    }
  }
}
```

ve çalışma zamanı çıktısı.



Uygulamanın başarılı bir şekilde çalışabilmesi için tahmin edeceğiniz üzere Cassandra sunucusunun da açık olması gerekir. Aksi durumda çalışma zamanında aşağıdakine benzer bir istisna(Exception) fırlatılacaktır.



Bu ilk örnekte BiGFootCompany **key space**' ine bağlanıp **ModelDesigner** isimli yeni bir **ColumnFamily** oluşturmaktayız. Bu **Column Family** içerisinde de, **Title, Nickname, Level** ve **Outsource**gibi kolonlar bulunmakta. Örnek olarak iki adet satır oluşturulmakta ve veritabanına ilave edilerek bir sonuç listesinin ekrana bastırılması sağlanmaktadır. Dikkat edileceği üzere **SQL** ifadelerine çok benzeyen bazı sorgular da kullandık.
Bunlar **CQL(Cassandra Query Language)** olarak adlandırılmaktadır. Ayrıca işlerimizi biraz daha kolaylaştırmak adına **dynamic** anahtar kelimesinden faydalanmaya çalıştık. Bu

sayede **Row Key'** lerin kolonlarına, birer özellikmiş gibi erişebilmemiz mümkün oldu. Böylece geldik bir makalemizin daha sonuna. Bir sonraki yazımızda görüşünceye dek hepinize mutlu günler dilerim

HowTo_HelloApacheCassandra.zip (457,46 kb)

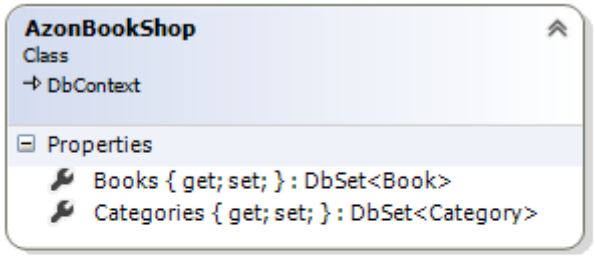
Entity Framework 6 – Code First için Convention Nedir?

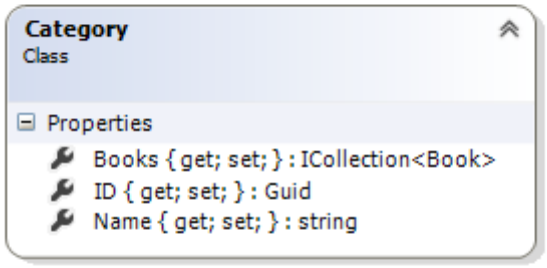
Çarşamba, 12 Aralık 2012 16:15

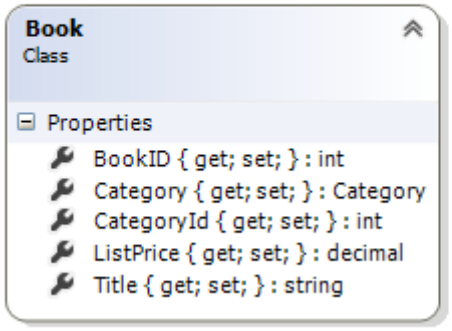
Entity Framework, Entity Framework 6.0, Dbcontent, Convention, Fluent Api, Dbset,Navigationproperty, Primary Key Convention, Relation Convention, Complex Type Convention, Connection String Convention, Lightweight Convention Model, Model Based Convention, Configuration Based Covention, Iedmconvention, Idbconvention,Idbmappingconvention

Merhaba Arkadaşlar,
Entity Framework takımı aldı başını gidiyor. Kim durduracak onları. Onlarda **The Mask** filmindeki karakter gibi **"Somebody stop me!"** demiyor ki
Aslında bakarsanız olaylar bana göre, Microsoft geliştirici takımlarının, diğer geliştiricilerin seslerini duymaya ve dikkate almaya başlamasından sonra epeyce gelişti. **Microsoft**' un çeşitli takımlarının açtığı anketler sayesinde, geliştiricilerin talepleri dinleniyor, değerlendiriliyor ve kayda değer olanlar planlanıp peyder pey yeni sürümlere ilave ediliyor. Hatta takımların ortaya koyduğu "şu da olsa nasıl olur?" ruh halindeki öğeler de geliştiriciler tarafından oylanıyor ve aynı sürece dahil edilebiliyor.*(Bloğumdaki takip ettiklerim listesinde bir kaç survey adresini bulabilirsiniz)*
Hal böyle olunca çok doğal olarak bir sürü sürüm çıkıyor ve var olanlar çabucak eskiyor. Takip edilmesi zor olan ve özellikle **Enterprise** seviyede ki projelerde "**acaba bu teknolojiyi kullanabilir miyiz?"** gibi soruların doğmasına ve ne yazık ki negatif olarak yanıtlanmasına neden olabilecek bir durum bu. Fakat biz yine de üstümüze düşen görevi yapalım ve gerekli anlatımımızı icra ederek öğrendiklerimizi sizlerle paylaşalım. Öyleyse başlayalım
[Makalede yazılanlar Entity Framework 6 Alpha 2 sürümünü baz almaktadır]
Entity Framework alt yapısının sunduğu önemli yaklaşımlardan birisi de **Code-First** modelidir. Bu modele göre geliştiriciler, önce sınıfları basit **POCO***(Plain Old CLR Objects)* tipler şeklinde tasarlar. Böylece **Conceptual(Domain) Model** oluşturulur. **POCO** tiplerinin tek başına tasarlanması elbette yeterli değildir. **DbContext** türevli bir sınıfında, model de kullanılması düşünülen **POCO** tiplerine ait koleksiyon bazlı özellikleri içeriyor olması gerekmektedir. Bu noktada **DbSet<T>** tipinden yararlanılır. Ayrıca tipler arasın ilişkileri betimleyen **Navigation Property**' ler de tasarlanır.
Sonrasında ise **Entity Framework** ilgili **Context** tipinden yararlanarak çalışma zamanında gerekli veritabanı üretimini icra eder. Peki hiç şu soru aklınıza geldi mi;

Bu üretim işlemi sırasında tablolar hangi kuralar göre oluşur, hangi alan Primary Key kabul edilir, tablolar arasındaki ilişkiler(Relations) nasıl belirlenir vb.
İşte bu yazımızda bu soruya biraz daha açıklık getirmeye çalışıyor olacağız.
Code First yaklaşımında, veritabanı tarafının üretilmesi aşamasında devreye girmekte olan bir takım kurallar bütünü bulunmaktadır. **Convention** olarak adlandırılan bu kurallar bütünü aslında**System.Data.Entity.ModelConfiguration.Conventions** isim alanı altında yer alan bazı tipler yardımıyla ifade edilmektedir. *(Entity Framework 5.0 sürümü için ilgili isim alanında(namespace) yer alan tipleri bu adresten inceleyebilirsiniz)*
Burada pek çok **Convention** tipi yer almaktadır. İlk olarak bu basit **Convention** tiplerinden bazılarını kavramsal olarak incelemeye çalışalım. Bu amaçla aşağıdaki basit içeriğe sahip olduğumuz bir örnek üzerinden ilerleyebiliriz.





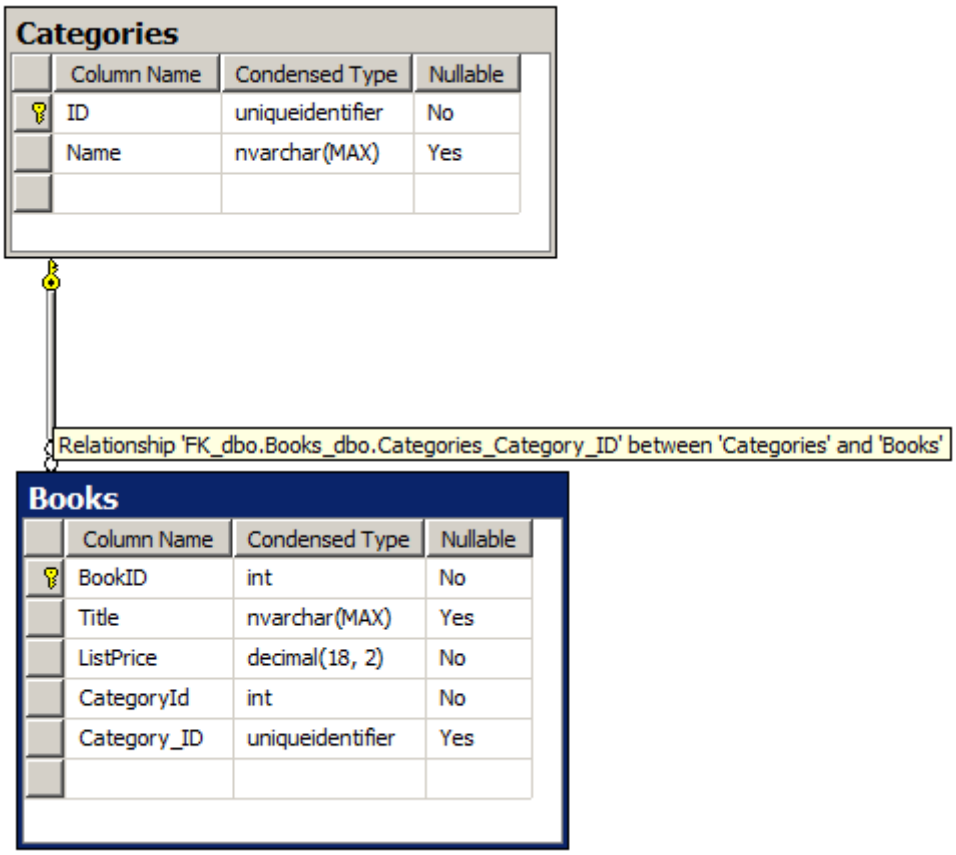



```
using System;
using System.Collections.Generic;
using System.Data.Entity;
namespace HowTo_EFCodeFirstConvetions
{
   public class AzonBookShop
     :DbContext
   {
```

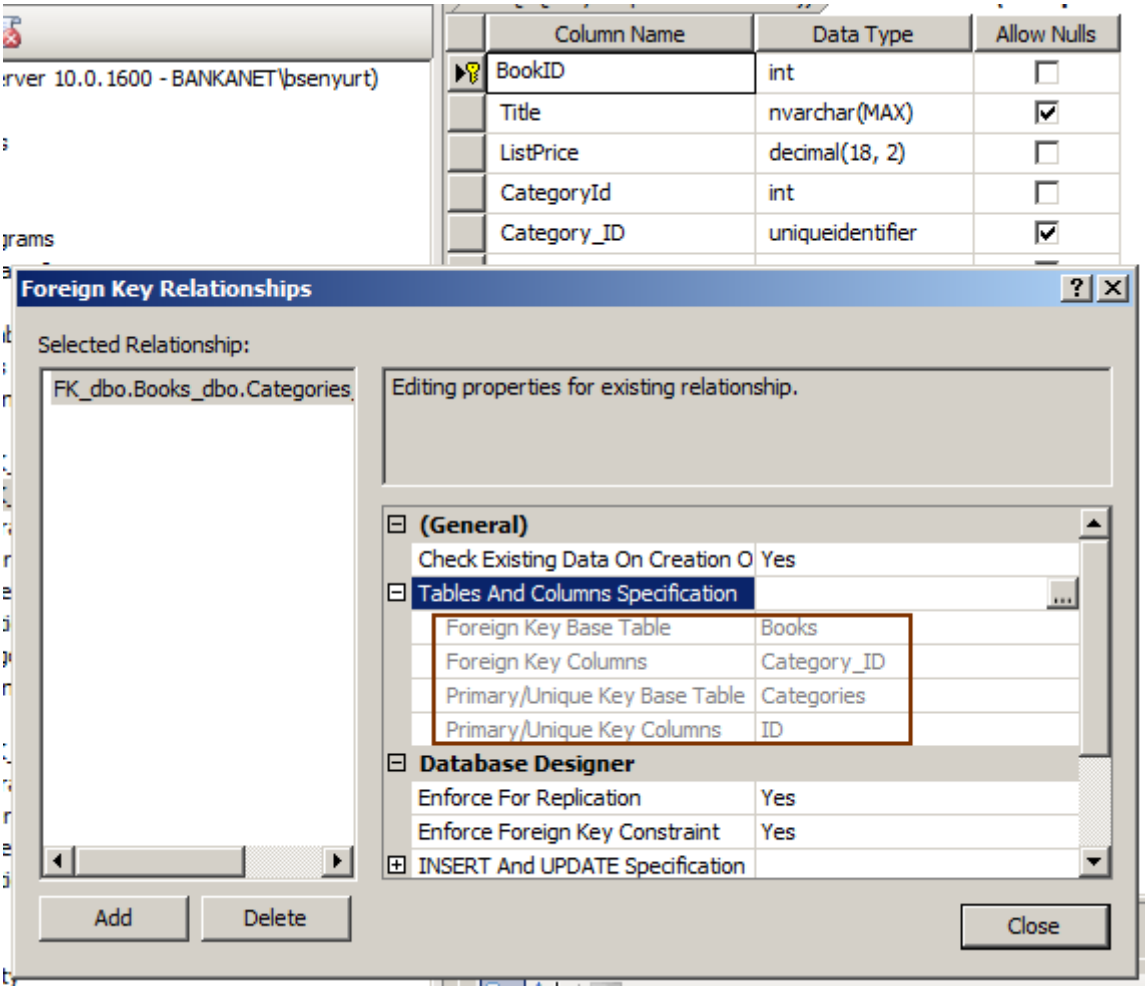
```
        public DbSet<Book> Books { get; set; }
        public DbSet<Category> Categories { get; set; }
        static AzonBookShop()
        {
            Database.SetInitializer(new
DropCreateDatabaseIfModelChanges<AzonBookShop>());
        }
    }
    public class Category
    {
        public Guid ID { get; set; }
        public string Name { get; set; }
        public virtual ICollection<Book> Books { get; set; }
    }
    public class Book
    {
        public int BookID { get; set; }
        public string Title { get; set; }
        public decimal ListPrice { get; set; }
        public virtual Category Category { get; set; }
        public int CategoryId { get; set; }
    }
}
```

Örnek üzerinde ilerlerken modeli sıkça değiştireceğimizden, static yapıcı metod*(Constructor)* içerisinde bir strateji seçerek, Initialize sırasında eğer model de bir değişiklik olmuşsa Drop işlemlerinin uygulanması gerektiği belirtilmiştir.

Şimdi bu modele göre üretilen veritabanı şemasını şöyle kısaca bir inceleyelim dilerseniz.



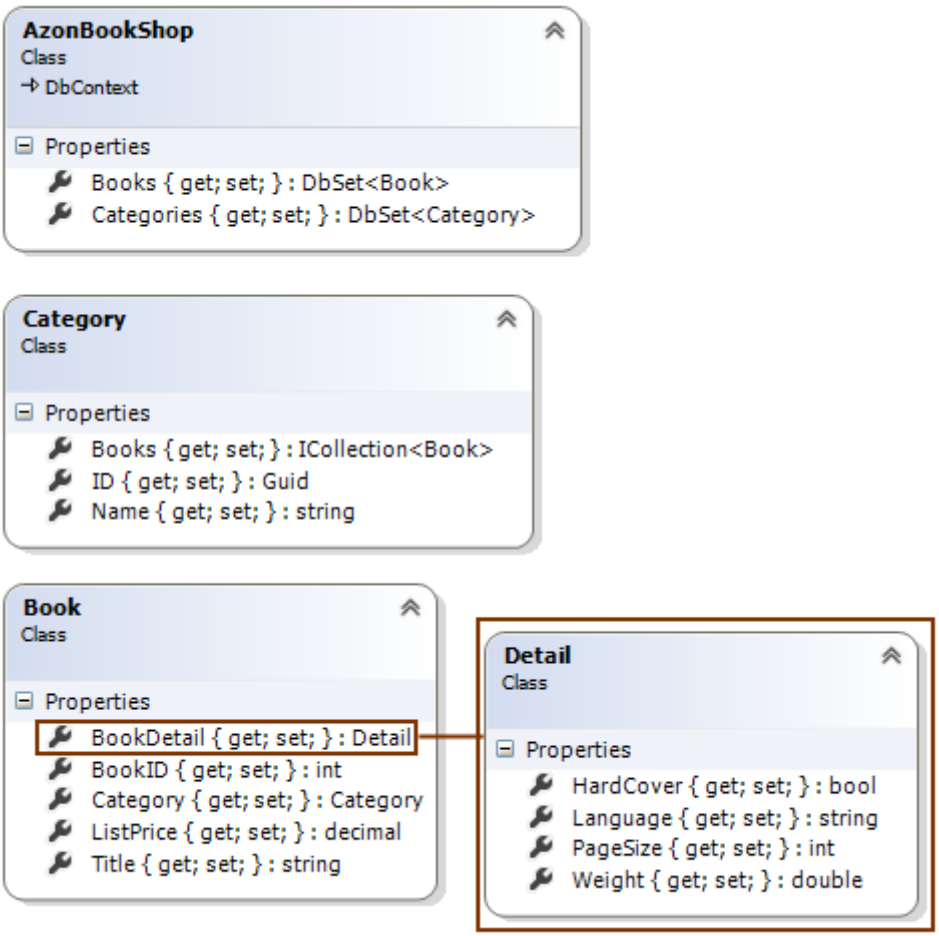
Dikkat edileceği üzere **Categories** ve **Books** isimli iki tablo üretilmiştir. Her iki tabloda birer**Primary Key** alan bulunmaktadır. İşte burada **Primary Key Convention** kural kümesi devreye girmektedir. Bu kural setine göre, **Guid** veya **int** tipinden olup adı **ID** veya **[SınıfAdı][Id]**notasyonunda olan özellikler, veritabanı şemasında birer **Identity** alan olarak üretilecek ve hatta**Primary Key** şeklinde işaretleneceklerdir. Senaryomuza **Class** seviyesinde baktığımızda, bir kategorinin altında birden fazla kitabın yer alabileceği görülmektedir. Nesneler arası kurulan bu **ilişkiyi*(association)*** tanımlamak adına**Category** sınıfı içerisinde **ICollection<Book>** tipinden bir özellik kullanılmıştır. Bunun karşılığı olarak veritabanı şemasında görüldüğü üzere iki tablo arasında bir **relation** kurulmuştur. Bu ilişki,**Categories** tablosundan **Books** tablosuna doğru **one-to-many** olacak şekildedir. Burada ise**Relation Convention** kuralları devreye girmektedir. Örnekte kasıtlı olarak **Book** sınıfı içerisidne **CategoryId** isimli ayrı bir özellik daha tanımlanmıştır. Mantıksal olarak bu özellik bir kitabın bağlı olduğu **Category** tipini işaret etmek üzere planlanmıştır. Ancak **Relation Convention** kurallarına göre isimlendirme de bir sorun vardır. **Category**tablosunun **Identity** şeklindeki **Primary Key** alanı **ID** olarak belirlenmiştir. Bu sebepten tablo şemasına bakıldığında **Category_ID** isimli bir alanın daha eklendiği ve iki tablo arasındaki bire çok ilişkinin, bu alan üzerinden sağlandığı görülmektedir.




```
USE [AzonBookShop]
GO
ALTER TABLE [dbo].[Books]  WITH CHECK ADD  CONSTRAINT
[FK_dbo.Books_dbo.Categories_Category_ID] FOREIGN KEY([Category_ID])
REFERENCES [dbo].[Categories] ([ID])
GO
ALTER TABLE [dbo].[Books] CHECK CONSTRAINT
[FK_dbo.Books_dbo.Categories_Category_ID]
GO
```

Relation ile ilişkili SQL script' i içinde de görüldüğü üzere **Books** tablosunda yer alan **Category_IDForeignKey** olarak belirlenmiş ve **Books** tablosundaki **ID** alanına bağlanmıştır.

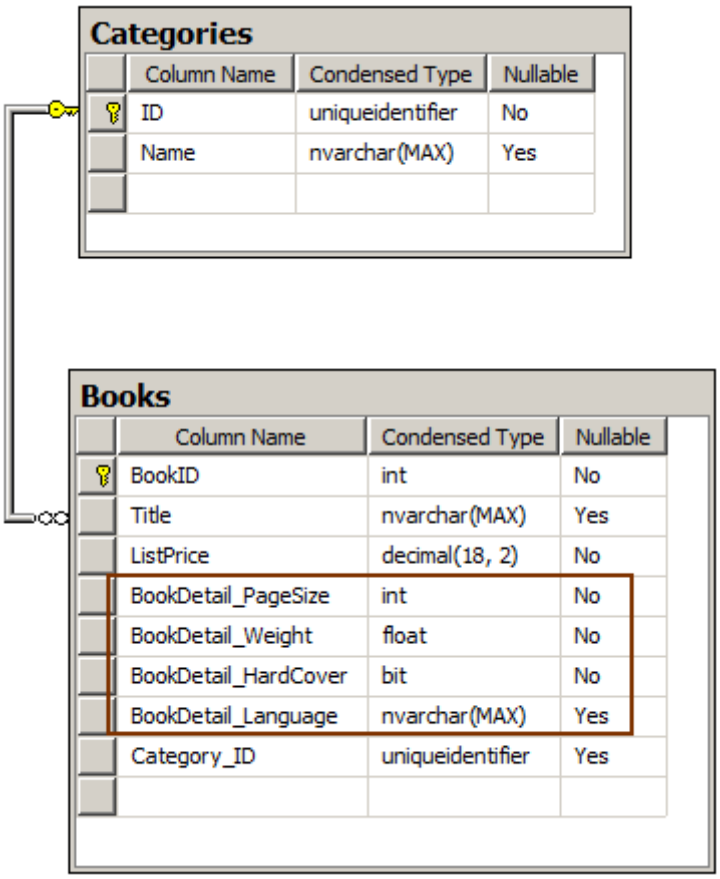
Şimdi dilerseniz örneğimizi biraz daha genişletelim ve Context için aşağıdaki sınıf çizelgesinde yer alan **POCO** tipini eklediğimizi düşünelim.




```
public class Book
{
   public int BookID { get; set; }
   public string Title { get; set; }
   public decimal ListPrice { get; set; }
   public virtual Category Category { get; set; }
   //public int CategoryId { get; set; }
   public Detail BookDetail { get; set; }
}
public class Detail
{
   public int PageSize { get; set; }
   public double Weight { get; set; }
   public bool HardCover { get; set; }
   public string Language { get; set; }
}
```

Book sınıfına **Detail** tipinden **BookDetail** isimli bir özellik eklenmiştir. **Detail** sınıfının dikkat çekici özelliği ise **Primary Key Convention** kuralına uygun bir özellik içermiyor

olmasıdır. Bu durumuda **Complex Type Convention** kural seti devreye girecektir ve veritabanı tarafında aşağıdaki sonuçların oluşmasına neden olacaktır.



Görüldüğü üzere **Detail** sınıfının özellikleri, **Book** tablosu içerisinde birer alan*(Field)* haline getirilmiştir.

Örnekte, makinede yüklü olan **SQL 2008** sunucusu kullanılmıştır. Bu nedenle **app.config** dosyası içerisinde aşağıdakine benzer bir **ConnectionString** bilgisi yer almaktadır. **name** niteliğinin değerinin **Context** sınıf adı ile aynı olması önemlidir.

```
<connectionStrings>
   <add name="AzonBookShop" connectionString="data
source=.;database=AzonBookShop;integrated security=SSPI"
providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Bu noktada aslında **Connection String Convention** kural seti devreye girmektedir. Bu kural setinin veritabanı şemasını oluştururken baktığı yerlerden birisi, **config** dosyasındaki **connectionStrings**elementi içeriğidir. Eğer **name** özelliğinin değeri ile **DbContext** türevli **Context** tipinin adı eşlenirse, o elemente ait **Connection** bilgisi kullanılaraktan bir şema üretimi gerçekleştirilecektir.

Ötesi

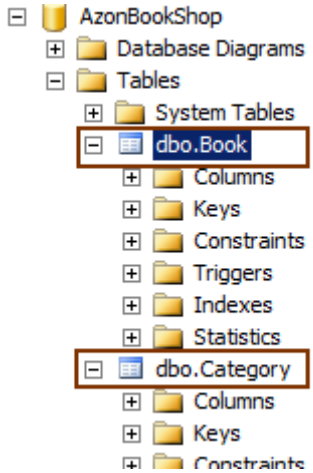
Buraya kadar anlatmaya çalıştığımız **Convention** kural setlerinin daha pek çok özelliği bulunmaktadır. **Data Annotations** ve **Fluent API** kullanımı gibi durumlarda ilgili **Convention**kural kümelerinin ezilmesi vb mümkündür. İstenirse

bir **Convention** kural seti devre dışı bırakılabilir de. Bunun için **DbContext** üzerinden gelen **OnModelCreating** metodunun ezilmesi ve içerisinde aşağıdakine benzer bir kodun kullanılması yeterlidir.

```
using System;
using System.Collections.Generic;
using System.Data.Entity;
using System.Data.Entity.ModelConfiguration.Conventions;
namespace HowTo_EFCodeFirstConvetions
{
   public class AzonBookShop
      :DbContext
   {
      public DbSet<Book> Books { get; set; }
      public DbSet<Category> Categories { get; set; }
      static AzonBookShop()
      {
         Database.SetInitializer(new
DropCreateDatabaseIfModelChanges<AzonBookShop>());
      }
      protected override void OnModelCreating(DbModelBuilder modelBuilder)
      {
         modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
      }
   }
   ...
```

modelBuilder tipi üzerinden **Conventions** özelliği ile **Remove** metoduna erişilmekte ve**PluralizingTableNameConvention** sınıfı generic parametre olarak belirtilmektedir. Örneğimizin önceki kısımlarında veritabanı tarafında üretilen tablo adları mutlaka dikkatinizi çekmiştir. Çoğul isimlendirme kuralına göre üretilmişlerdir.

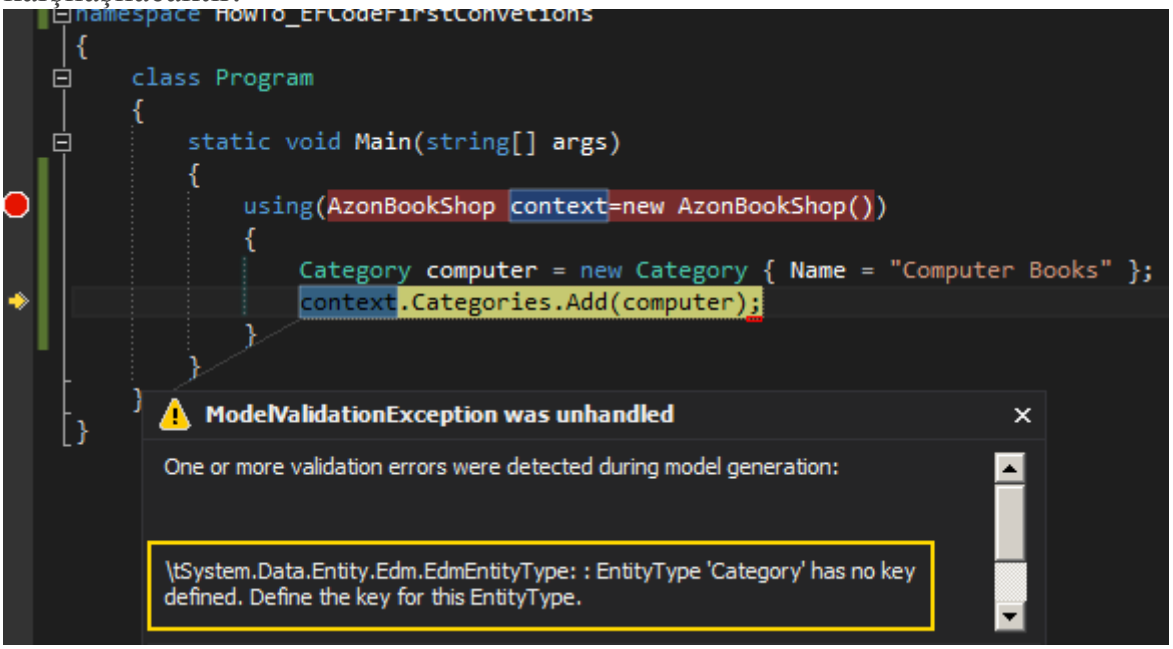
Ancak **OnModelCreating** içerisinden bu kural setini kaldırmamız, tablo adlarının sınıf adları olarak tanımlanmasını sağlamaktadır.



Peki, **Convention** kuralları ile ilişkili olarak daha başka neler yapabiliriz? Özellikle bunları manuel olarak ele abilir miyiz? Var olan Convention kurallarını geçersiz kılarak kendi istediğimiz ayarların devreye girmesini nasıl sağlarız?
Convetion kurallarını manuel olarak ele almanın bir kaç yolu bulunmaktadır. Bunlardan birisi**Lightweight** modelidir. Bu modelde önce bir filtreleme işlemi yapılır ve işlem sonucuna göre konfigurasyonun değiştirilerek uygulanması sağlanır. Örneğin modelimizde yer alan **Category**sınıfının içeriğini aşağıdaki gibi değiştirdiğimiz düşünelim.

```
public class Category
{
   public Guid Signature { get; set; }
   public string Name { get; set; }
   public virtual ICollection<Book> Books { get; set; }
}
```

Burada **ID** isimli özelliğin **Signature** olarak değiştirildiği görülmektedir. Bu değişiklik nedeniyle pek tabi **Primary Key Convention** kural seti uygulanamayacaktır. Daha da kötüsü, senaryomuz gereği**Category** ile **Book** arasında bir **relation** tesis edilebilmesi için gerekli **Foreign Key** bulunamayacak ve çalışma zamanında aşağıdaki **Exception** ile karşılaşılacaktır.

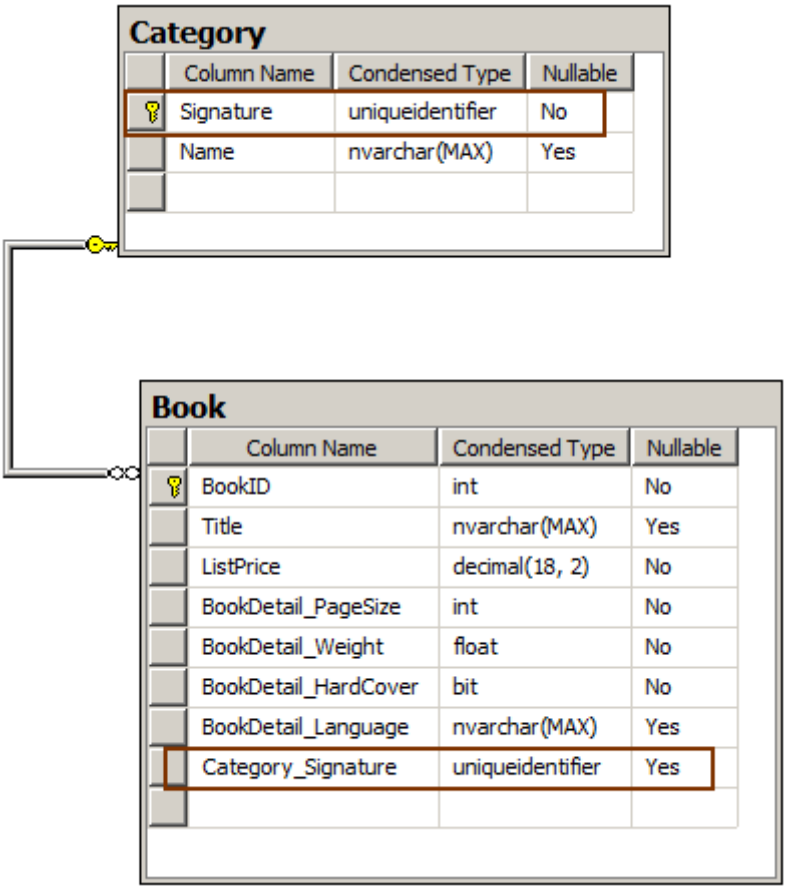


İşte bu noktada **Lightweight Convention** tekniği ile durum çözümlenebilir. Bunun için yine**OnModelCreating** içerisinde bazı işlemler yapılması gerekmektedir. Aynen aşağıda görüldüğü gibi.
Söz konusu kodda yer alan Properties özelliği EF 6.0 Alpha 2 sürümünde duyurulmuştur. Dolayısıyla bundan sonraki kodlar için güncel PreRelease sürümünü kurarak devam etmelisiniz. Kurulum için suradaki adresten yararlanabilirsiniz

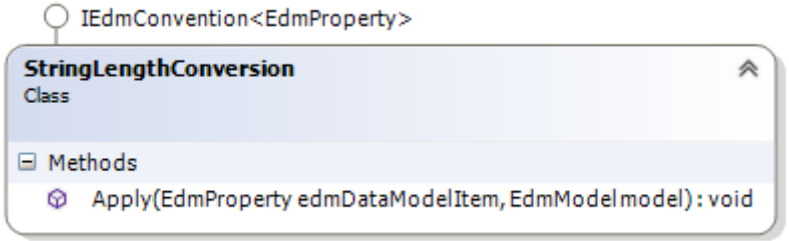
```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
   modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
```

```
    modelBuilder
        .Properties()
        .Where(p => p.Name.Contains("Signature"))
        .Configure(p => p.IsKey());
}
```

modelBuilder üzerinden **Properties** metodu kullanılarak, **Entity**' deki özellikler arasında**Signature** kelimesini içeren bir tane olup olmadığına bakılmakta ve eğer var ise **IsKey** metodu çağrısı ile bunun bir **Identity** alan olması gerektiği*(bir başka deyişle Primary Key Convention kurallarının uygulanması gerektiği)* vurgulanmaktadır. Buna göre çalışma zamanı sonucunda veritabanı tarafında aşağıdaki şemanın üretildiği gözlemlenecektir.



Convention çeşitlerinden bir diğeri de **Model-Based** olan versiyondur. Bu teknikte doğrudan model ile çalışma ve ayarlama şansına sahip oluruz. İlgili tekniği uygulayabilmek için**IEdmConvention**, **IDbConvention** ve **IDbMappingConvention arayüzlerini*(interface)***implemente eden sınıflardan yararlanırız. Aşağıdaki basit örneği göz önüne alalım.



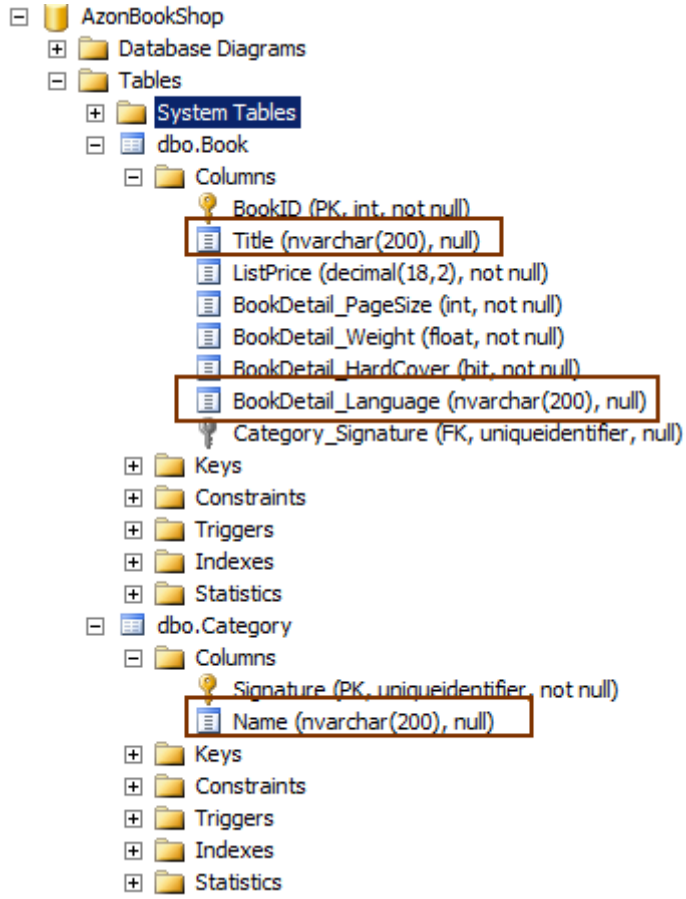
```
using System.Data.Entity.Core.Metadata.Edm;
using System.Data.Entity.ModelConfiguration.Conventions;
namespace HowTo_EFCodeFirstConvetions
{
   public class StringLengthConversion
      : IEdmConvention<EdmProperty>
   {
      public void Apply(EdmProperty edmDataModelItem, EdmModel model)
      {
         if (edmDataModelItem.PrimitiveType.PrimitiveTypeKind ==
PrimitiveTypeKind.String)
            edmDataModelItem.MaxLength = 200;
      }
   }
}
```

IEdmConvention<EdmProperty> interface' ini implemente eden **StringLengthConversion**sınıfı **Apply** metodunu uygulamaktadır. Bu metodun içerisinde, **edmDataModelItem** isimli değişkenin **String** tipi olup olmadığına bakılmakta ve eğer öyleyse **Max Length** değeri **200** karakter ile sınırlandırılmaktadır.

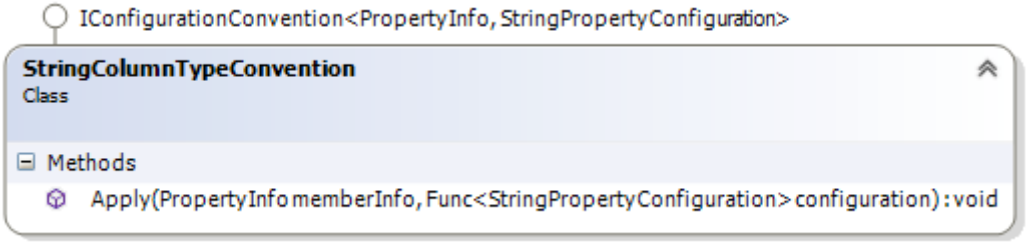
Bu işlem pek tabi **Model** içerisinde yer alan ne kadar **String** tipte öğe var ise geçerli olacktır. Tabi söz konusu sınıfın devreye girebilmesi için yine **OnModelCreating** içerisine müdahale edilmelidir. Aşağıdaki kod parçasında görüldüğü gibi.

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
   modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
   modelBuilder
      .Properties()
      .Where(p => p.Name.Contains("Signature"))
      .Configure(p => p.IsKey());
   modelBuilder
   .Conventions
   .AddBefore<StringLengthAttributeConvention>(new StringLengthConversion());
}
```

Bu işlem sonucunda veritabanı şemasında aşağıdaki sonuçların oluştuğu gözlemlenecektir.



Bir başka **Convention** modeli ise **Configuration** tabanlı olanıdır. Bu teknikte**IConfigurationConvention arayüzünün(intercace)** implenente edildiği bir sınıfın devreye girerek **Convention** kurallarına müdahale etmesi söz konusudur. Aşağıdaki örnek sınıfı göz önüne alalım.



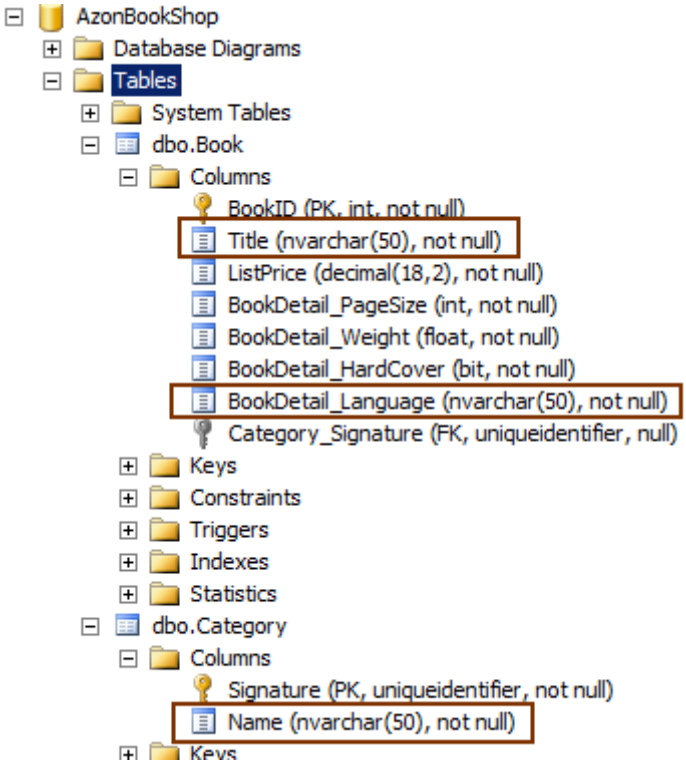

```
using System;
using System.Data.Entity.ModelConfiguration.Configuration.Properties.Primitive;
using System.Data.Entity.ModelConfiguration.Conventions;
using System.Reflection;
namespace HowTo_EFCodeFirstConvetions
{
   public class StringColumnTypeConvention
      :IConfigurationConvention<PropertyInfo,StringPropertyConfiguration>
   {
      public void Apply(PropertyInfo memberInfo
        , Func<StringPropertyConfiguration> configuration)
```

```
      {
        if (configuration().ColumnType == null)
         {
            configuration().ColumnType = "nvarchar";
            configuration().IsNullable = false;
            configuration().MaxLength = 50;
         }
      }
   }
}
```

Öncelikli olarak ColumnType özelliğinin değerinin null olup olmadığın bakılmaktadır. Bu işlem, ilgili alanın daha önceden oluşturulup oluşturulmadığını da işaret etmektedir. Pek tabi **Convention**kurallarının devreye girebilmesi için yine **OnModelCreating** metoduna bir müdahale de bulunmak gerekmektedir. Aşağıdaki kod parçasında bu durumu görebilirsiniz.

```
protected override void OnModelCreating(DbModelBuilder modelBuilder)
{
   modelBuilder.Conventions.Remove<PluralizingTableNameConvention>();
   modelBuilder
      .Properties()
      .Where(p => p.Name.Contains("Signature"))
      .Configure(p => p.IsKey());
   modelBuilder.Conventions.Add<StringColumnTypeConvention>();
}
```

Bu işlem sonrasında veri tabanı şemasının aşağıdaki gibi üretildiği görülecektir.



Dikkat edileceği üzere **String** özelliklerin karşılığı olarak **nvarchar** tipinde olan, **null** değer içeremeyen ve **maksimum 50** karakter uzunluğunda içerik tutabilen alanlar üretilmiştir.

Code First yaklaşımında **Convention** kullanımı ile ilişkili olarak daha ileri seviye uygulamalar da mevcuttur. Söz gelimi Custom Attribute' lar le yeni Convention kural setleri tanımlanabilir. Özellikle LightWeight modelinde kullanılabilecek epey fazla fonksiyonllik bulunmaktadır. Bu konuda şu adresteki yazının son kısımlarını da değerlendirebilir ve kendi denemelerinizi yaparak konuyu irdelemeye çalışabilirsiniz. Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim

[Makalede yazılanlar Entity Framework 6 Alpha 2 sürümünü baz almaktadır]

HowTo_EFCodeFirstConvetions.zip (2,14 mb) *(Dosya boyutunun büyümemesi için Packages klasörü çıkartılmıştır. EF' in 6ncı sürümünü projeye indirmeniz gerekebilir)*

Tek Fotoluk İpucu–74–SequenceEqual

Salı, 11 Aralık 2012 07:40

Linq, Generics, Sequenceequal, List<T>, C#, Extension Methods,Iequalitycomparer, Comparer

Merhaba Arkadaşlar,
Diyelim ki uygulamanızda zaman zaman da olsa farklı referanslar da duran ve aynı tipte elemanlardan oluşan koleksiyonlarınız oluşuyor ve bunları yeri geldiğinde birbirleri ile kıyaslamak istiyorsunuz. Ne yaparsınız?
Mantıksal olarak her iki koleksiyonu dolaşacak ortak bir döngü ile bire bir kıyaslama yolunu tercih edersiniz  Ama daha pratik yollar da var. Örneğin aşağıdaki gibi 

Bu kadar basit. Bir başka ip ucunda görüşmek dileğiyle.

Not : SequenceEqual metodu kaynak ve hedef koleksiyonların sıralı olduğunu varsayar. Yani yukarıdaki örnekte koleksiyonların içeriklerini farklı sırada tutarsak sonuç False dönecektir.

WCF Interceptors

Pazartesi, 3 Aralık 2012 06:53

Windows Communication Foundation, Wcf, Interceptor, Parameter Inspector,Message Inspector, Custom Behavior, Iparameterinspector, Ioperationbehavior,Attribute, Iendpointbehavior, Endpoint Behavior

Merhaba Arkadaşlar,
Hepimizin hafızasında yer eden ve defalarca seyretse de asla sıkılmayacağı kült filmler vardır. Hatta nesiller ilerledikçe, her neslin mutlaka en az bir kere uğradığı, uğraması gereken yapımlar vardır.
The Godfather, Starwars, Matrix, The Good the bad and the ugly, Back to the future vb…Bunlardan birisi de benim için **Mad Max**' dir.
Serinin ilginç konusu dışında en çok dikkatimi çeken*(pek çok erkeğin de dikkatini çetkiği üzere)*filmde kullanılan araçlardır. Bunlardan birisi de o zamanlar yol devriyesi olarak görev yapan **V8**motora sahip **1974** model **Ford Falcon XB** dir. Tabi bunu tahmin edeceğiniz üzere siyah renkli efsanevi **Ford GT351** takip etmiştir. Detaylar için wikipedia' dan bilgi alabilirsiniz
Peki bizimle ne alakası var Bu yazımızda farklı bir **Interceptor** kavramını göz önüne alıyor olacağız.
WCF(Windows Communication Foundation) alt yapısının popüler olmasının en büyük nedenlerinden birisi de, hemen her seviyede genişletilebilir olmasıdır. Genişleyebilirlik, bir**Framework** için oldukça önemli bir özelliktir. Nitekim bu yeteneğin olması, geliştiricilerin daha fazla noktada müdahalede bulunabilme ve ihtiyaçları daha fazla yerde çözümleyebilme kabiliyetini kazanabilmesi anlamına gelmektedir.
WCF tarafındaki **genişletilebilirlik***(extendability)* ve hatta yeniden yazabilme yetenekleri, **WCF** alt yapısı üzerine oturmuş yan ürünlerin de oluşmasına neden olmuştur. Söz gelimi **Data Service'** ler veya **WCF Web API** bu duruma verilebilecek en güzel örneklerdendir. Tabi bunun dışında daha ince ayarlar da yapılabilir. Söz gelimi kendi **ServiceHost** ortamımızı yazabiliriz veya özel **kanal yığını***(Channel Stack)* oluşturup güvenlik için daha etkin çözümler üretebiliriz. Ya da kendi **Binding**tipimizi geliştirerek şirketimize ait bir mesajlaşma protokolünün devreye alınmasını sağlayabiliriz.
WCF çalışma mantığı düşünüldüğünde istemci ve servis tarafı*(Dispatcher diyelim bu yazımız için)*arasındaki iletişim sırasında da araya girebileceğimiz konumlar bulunmaktadır. Sonuçta istemci bir mesajı servis tarafına gönderir ve karşılığında genellikle bir cevap bekler*(veya beklemez. OneWay tipindeki operasyonları düşünelim)* İşte bu süreçte

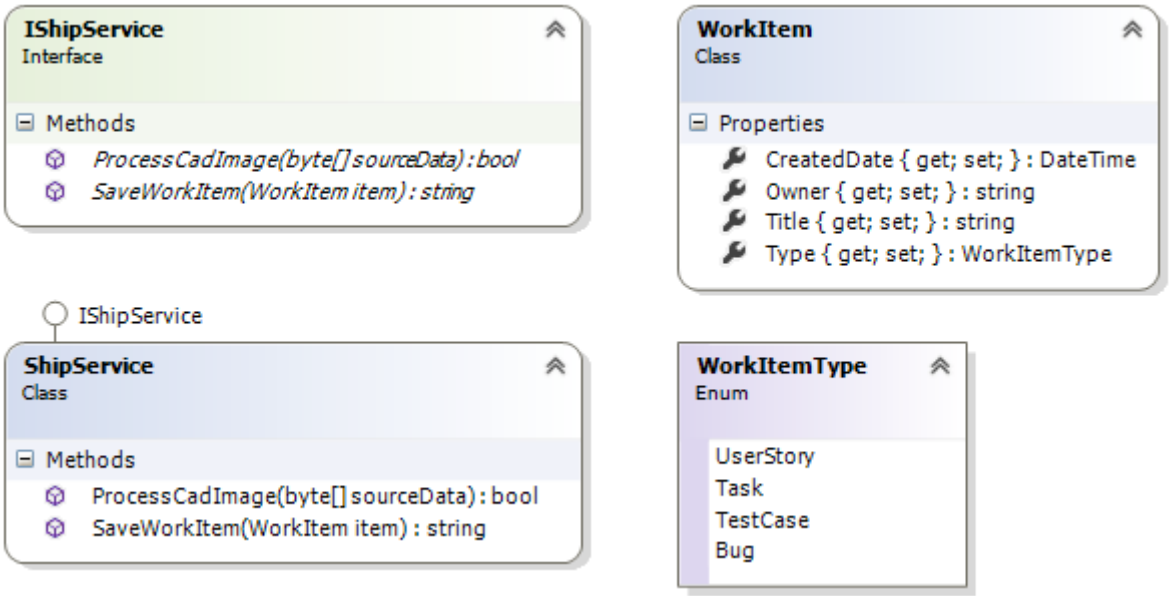
parametre dahil, mesaj seviyesinde dahi araya girebilir ve istediklerimizi gerçekleştirebiliriz. İşte bu yazımızın konusu da budur. **Interceptors**

Interceptor kelimesi durdurucu, yol kesen, önleme uçağı gibi anlamlara gelmektedir. Lakin ilerleyen kısımlarda göreceğiniz gibi kendi geliştireceğimiz özel kesici tipler Inspector kelimesini içeren arayüzleri(Interface) uygular. Inspector ise kontrolör, denetleyici, denetmen, müfettiş gibi anlamlara gelmektedir. Dolayısıyla kelimeler ve kavramlar birbirleri yerine geçebilirler. Algıda bir karmaşa olmaması adına belirtmek isterim.

Interceptor' ler bir servis operasyonu öncesinde veya sonrasında devreye girebilmektedirler. WCF tarafında kesmede bulunabileceğimiz 4 yer vardır. Öncelikli olarak bir servis operasyonunun parametrelerini değerlendirebilir ve hatta değiştirebiliriz. Yani bir servis operasyonunun parametresi işlenmeden önce müdahalede bulunma şansına sahibiz.

2nci olarak servis tarafına gelen bir mesajın yakalanması da mümkündür. Bu mesaj ele alınıp yine düzenlenebilir ve bir kontrol sürecine dahil edilebilir. Hatta dönecek olan mesaj içeriğ için de aynı durum söz konusudur. 3ncü olarak bir mesajın formatına müdahalede bulunulabilecek seviyede kesme yapılabilir. 4ncü ve son olarak da çağırılan bir operasyonun kendisi için kesme işlemi uygulanabilir.

Biz bu yazımızda mesaj ve parametre seviyesinde kesme işlemlerinin nasıl uygulanabileceğini incelemeye çalışıyor olacağız. Bu amaçla ilk olarak **Visual Studio***(2010 veya 2012 olabilir)*ortamında basit bir **WCF Service Application** projesi oluşturalım. Başlangıç için aşağıdaki sınıf diyagramında yer alan tipleri yazabiliriz.



Servis sözleşmesi olan **IShipService** aşağıdaki gibi yazılabilir.

```
using System.ServiceModel;
namespace SomeServiceApp
{
   [ServiceContract]
   public interface IShipService
```

```
    {
        [OperationContract]
        string SaveWorkItem(WorkItem item);
        [OperationContract]
        bool ProcessCadImage(byte[] sourceData);
    }
}
```

Servis sözleşmesine ait implementasyonu gerçekleştirdiğimiz tip ise şu şekilde yazılabilir.

```
using System;
namespace SomeServiceApp
{
    public class ShipService
      : IShipService
    {
        public string SaveWorkItem(WorkItem item)
        {
            return string.Format("{0}[{3}] {1} tarihinde {2} tarafından oluşturuldu"
                , item.Title
                ,item.CreatedDate
                ,item.Owner
                ,item.Type);
        }
        public bool ProcessCadImage(byte[] sourceData)
        {
            return true;
        }
    }
}
```

Bu basit servis içerisinde iki operasyon uygulanmaktadır. **SaveWorkItem** aşağıda içeriği verilen**WorkItem** tipinden bir parametre almakta ve bunu sembolik olarak işlemektedir.**ProcessCadImage** operasyonu ise **byte[]** tipinden bir parametre ile çalışmaktadır. Yine bu operasyonun iç yapısı da çok önemli olmadığından varsayılan olarak **true** döndürecek şekilde geliştirilmiştir.

WorkItem sınıfı içeriği;

```
using System;
namespace SomeServiceApp
{
    public class WorkItem
    {
        public string Title { get; set; }
        public DateTime CreatedDate { get; set; }
```

```
        public string Owner { get; set; }
        public WorkItemType Type { get; set; }
    }
}
```

WorkItemType Enum sabiti;

```
namespace SomeServiceApp
{
    public enum WorkItemType
    {
        UserStory,
        Task,
        TestCase,
        Bug
    }
}
```

Parametre Bazlı Kontrolör (Parameter Inspector)

Parametre bazlı kesme tiplerinin kullanılabileceği senaryolardan ilk akla gelen sanıyorumki doğrulama operasyonlarıdır. Söz gelimi örnek servisimizi göz önüne aldığımızda, **ProcessCadImage**operasyonuna gelen **byte[]** tipinden dizinin içeriğini kontrol ettirebilir ve gerçekten bir **Image** olup olmadığını denetleyebiliriz. Ama bu kadar komplike düşünmemize şu aşamada pek gerek yoktur  Sonuçta basit bir boyut kontrolü de pekala işimizi görecektir. Buna göre ilk yapmamız gereken bir parametre kesicisi yazmaktır. Bunun için **System.ServiceModel.Dispatcher** isim alanında bulunan**IParameterInspector** türevli bir tip geliştirmemiz gerekecek. Aynen aşağıda olduğu gibi.

```
using System.ServiceModel;
using System.ServiceModel.Dispatcher;
namespace SomeServiceApp
{
    public class CADSizeInspector
        :IParameterInspector
    {
        public double DefaultControlSize { get; set; }
        // Operasyon çağrısı tamamlandıktan sonra devreye girecektir.
        public void AfterCall(string operationName, object[] outputs, object
returnValue, object correlationState)
       {

        }
        // Çağırılan operasyon başlatılmadan önce devreye girecektir
        public object BeforeCall(string operationName, object[] inputs)
```

```
      {
         // Operasyon adını kontrol ediyoruz
         if (operationName == "ProcessCadImage")
         {
            // Gelen ilk parametre operasyonumuza göre byte[] dizisi. Eğer
DefaultControlSize' dan büyükse geriye bir FaultException döndürülmektedir
            if (((byte[])inputs[0]).Length > DefaultControlSize)
               throw new FaultException("CAD çizimine ait mesaj boyutu
varsayılandan büyük.");
         }
         // Herhangibir sıkıntı yoksa null dönerek operasyonun devam etmesini
sağlayabiliriz.
         return null;
      }
   }
}
```

IParameterInspector arayüzü*(Interface)* ile birlikte gelen iki kritik operasyon vardır. Bunlardan birisi **AfterCall** diğeri ise **BeforeCall'** dur. Geriye bir şey döndürmeyen **AfterCall** metodu, tahmin edileceği üzere ilgili servis operasyonu işleyişini tamamladıktan sonra devreye girmektedir. Tabiri yerinde ise iş işten geçtikten sonra diyebiliriz İşin aslı, burada servis operasyon çağrısının tamamlanması sonrası gerçekleştirilecek işlemlere yer verebiliriz. Belki loglama veya performans ölçümleri adına ideal bir yer olabilir.

Geriye **object** tipinden değer döndüren ve çağrıda bulunulan operasyon adı ile o operasyona gelen parametreler için **object** tipinden bir diziyi parametre olarak alan **BeforeCall** metodu ise, kesme işleminin uygulanabileceği en ideal yerdir. Örnekteki kod parçasında, servis operasyonunun **ProcessCadImage** olup olmadığı kontrol edilmiş ve bu işlemin ardından gelen ilk parametrenin **byte[]** tipinden olan karşılığının uzunluğuna bakılmıştır. Söz konusu uzunluk değeri bu sınıfın bir özelliğidir*(Property).* Dolayısıyla kesme sınıfını ürettiğimiz yerde belirlenebilir. Eğer servis operasyonuna gönderilen **byte[]** dizisinin boyutu kabul edilebilir olandan çok büyükse ortama bir **FaultException** fırlatılmaktadır. Çok doğal olarak bu istisna bilgisi istemci tarafına da gidecektir*(Eğer includeExceptionDetailInFaults niteliğinin değeri true ise)*

Aslında bu tip boyut kontrolleri için konfigurasyon dosyasında yer alan binding nitelikleri de kullanılabilir. Lakin bu ayarlarda daha çok mesajın boyutu göz önüne alınmaktadır. Bizim senaryomuzda ise dikkat edileceği üzere operasyona gelen bir parametreye özgü olacak şekilde boyut kontrolü gerçekleştirilmektedir.

Peki parametre için gerekli kesme tipini tanımladık. Bunu WCF çalışma zamanı nasıl bilebilir?

Söz konusu kesme operasyonu aslında bir servis metoduna ait olduğundan, nitelik bazlı bir davranış*(Attribute -ased Behavior)* olarak ortama bildirilebilir. Yani, bir **Custom Behavior** tipi söz konusudur. Bu amaçla aşağıdaki sınıfı geliştirmemiz yeterli olacaktır.

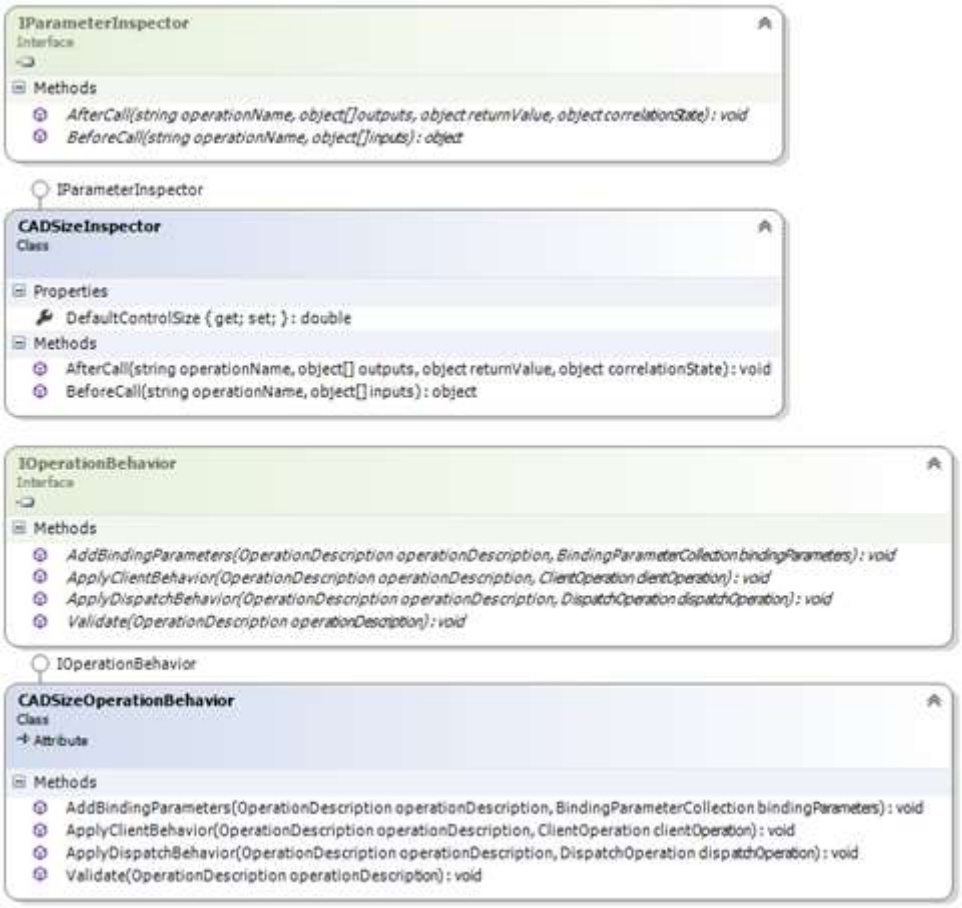
```
using System;
using System.ServiceModel.Channels;
using System.ServiceModel.Description;
using System.ServiceModel.Dispatcher;
namespace SomeServiceApp
{
   public class CADSizeOperationBehavior
      :Attribute,IOperationBehavior
   {
      public void AddBindingParameters(OperationDescription operationDescription,
BindingParameterCollection bindingParameters)
      {
      }
      public void ApplyClientBehavior(OperationDescription operationDescription,
ClientOperation clientOperation)
      {
      }
      // Servis tarafındaki dispatch sürecinde araya girilmesi için gerekli tanımlamanın
yapıldığı metoddur
      public void ApplyDispatchBehavior(OperationDescription operationDescription,
DispatchOperation dispatchOperation)
      {
         // sürece bir Parametre kesici tipi bildirilmektedir
         dispatchOperation.ParameterInspectors.Add(
            new CADSizeInspector
            {
               DefaultControlSize = 1024 // Bu kontrol değeri ve benzeri özellikler Behavior
tipi içerisinde tanımlanıp geçirilebilir de
            }
            );
      }
      public void Validate(OperationDescription operationDescription)
      {
      }
   }
}
```

CADSizeOperationBehavior tipi **Attribute** sınıfından türemekte ve **IOperationBehavior**arayüzünü*(Interface)* uygulamaktadır. Bu arayüz içerisinde gelen

ve **ezilmesi***(override)* gereken 4 temel metod bulunmaktadır. Senaryoya uygun olacak şekilde bu fonksiyonelliklerden biri veya bir kaçı ele alınabilir.
Bizim senaryomuzda parametre bazlı bir kesicinin, **Dispatch** işlemi sırasında devreye girmesi beklenmektedir. Bir başka deyişle servis tarafındaki işlem süreci sırasında yürürlüğe girecek bir kesici tipden bahsedilmektedir. Bu sebepten **ApplyDispatchBehavior** metodu içerisinde bir tanımlama yapılmıştır. Dikkat edileceği üzere **DispatchOperation** tipinden olan**dispatchOperation** parametresinin **ParameterInspectors** koleksiyonuna yeni bir**CADSizeInspector** sınıf örneği ilave edilmiştir.
Çok doğal olarak tanımlanan bu davranışın servis operasyonunun implemente edildiği metod içinde bildirilmesi gerekecektir. **Attribute** türevli bir tip geliştirdiğimizden **ProcessCadImage**operasyonuna bu davranışı aşağıdaki gibi enjekte edebiliriz.

```
// ProccessCad için özel bir çalışma zamanı davranışı belirtilmiştir.
[CADSizeOperationBehavior]
public bool ProcessCadImage(byte[] sourceData)
{
   return true;
}
```

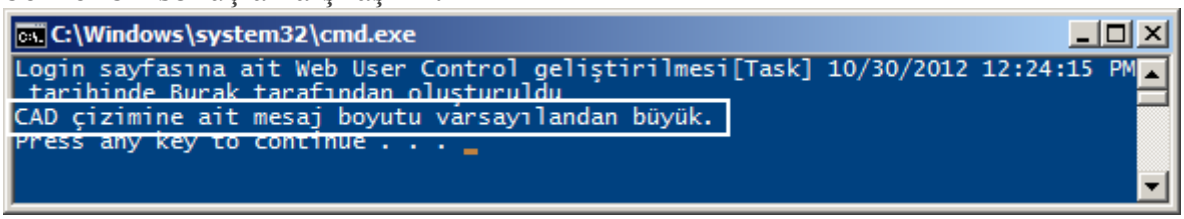
Özetle servis tarafında Parametre bazlı kesici için aşağıdaki sınıf diagramında görülen tiplerin geliştirildiğini ifade edebiliriz.



Artık istemci tarafı için örnek bir uygulama geliştirebilir ve sonuçları irdeleyebiliriz. Basit bir**Console** uygulaması işimizi görecektir. Servis referansını istemci tarafına ilave ettikten sonra aşağıdaki örnek kod satırlarını yazabiliriz.

```
using ClientApp.ShipServiceReference;
using System;
namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         ShipServiceClient proxy = null;
         try
         {
            proxy = new ShipServiceClient("BasicHttpBinding_IShipService");
            WorkItem wi = new WorkItem
            {
               CreatedDate = DateTime.Now,
               Owner = "Burak",
               Title = "Login sayfasına ait Web User Control geliştirilmesi",
               Type = WorkItemType.Task
            };
            string swiResult = proxy.SaveWorkItem(wi);
            Console.WriteLine(swiResult);
            byte[] someImageData = new byte[2048];
            bool pcResult = proxy.ProcessCadImage(someImageData);
            Console.WriteLine(pcResult);
         }
         catch (Exception excp)
         {
            Console.WriteLine(excp.Message);
         }
         finally
         {
            if (proxy != null
               && proxy.State == System.ServiceModel.CommunicationState.Opened)
               proxy.Close();
         }
      }
   }
}
```

İstemci tarafı sırasıyla **SaveWorkItem** ve **ProcessCadImage** isimli servis operasyonlarını çağırmaktadır. Söz konusu işlemler bir **try…catch…finally** bloğu içerisinde icra edilmektedir. İlk önce **Fault** durumunu kontrol etmek istediğimizden, **ProcessCadImage** metodu için **byte[]**tipinden olan dizinin parametresi kasıtlı olarak **1024**' ün üstünde tutulmuştur. Buna göre çalışma zamanında aşağıdakine benzer bir sonuçla karşılaşırız.



Görüldüğü gibi Parametre bazlı çalışan kesici tip devreye girmiş ve servis tarafında üretilen **Fault**mesajı istemci tarafında da ele alınabilmiştir. *(Bu noktada AfterCall metoduna hiç uğranılmayacağını da ifade edebiliriz)* Tabi tam tersi durumu da test etmemiz yerinde olacaktır. Yani uygun bir boyut gönderdiğimizde true değerini aldığımızı da görmeliyiz

Mesaj Bazlı Kesici (Message Interceptor)

Gelelim mesaj bazında kesme işleminin nasıl yapılabileceğine. Mesaj kesme işlemini istemci veya servis tarafında gerçekleştirebiliriz. Eğer servis tarafında*(Dispatch kısmı)* bir kesici söz konusu ise öncelikli olarak **IDispatchMessageInspector** arayüzünü implemente edecek bir sınıfın geliştirilmesi gerekmektedir. İstemci taraflı bir kesme operasyonu için de**IClientMessageInspector** arayüzü uygulanmalıdır. Bazı senaryolarda her iki arayüzün implemente edildiği tek bir hybrid tip de söz konusu olabilir. Bu sayede hem istemci hem de servis tarafı için çalışabilecek bir mesaj kesicisi geliştirilebilir. Biz örnek senaryomuzda mesajın toplandığı servis tarafını göz önüne alarak ilerleyeceğiz. Bu nedenle **IDispatchMessageInspector** arayüzünü değerlendiriyor olacağız.

Mesaj kesicilerini yazmanın en zorlu yanı gelen bilginin formatına göre bir parser ihtiyacı olmasıdır. Varsayılan ve bildiğimiz üzere mesajın içeriği tamamen **XML** formatındadır. Ancak **WCF Web API**gibi alt yapıların sunduğu **JSON** formatlı içerikler de servis tarafına gelebilir. Dolayısıyla böyle bir durumda **JSON** tabanlı bir **parsing** işlemi de işin içerisine girecektir. Aynı durum TCP bazlı çalışan mesajların olduğu durumda daha farklı bir hal alacaktır.

Fazla vakit kaybetmeden hemen bir implementasyona başlayalım dilerseniz. İlk olarak**IDispatchMessageInspector** arayüzünü uygulayan bir sınıf ile işe başlıyoruz.

```
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.ServiceModel.Dispatcher;
namespace SomeServiceApp
{
   public class ShipServiceMessageInspector
     :IDispatchMessageInspector
```

```
    {
        // Operasyon talebi servis tarafına ulaştıkan sonra devreye giriyor olacak
        public object AfterReceiveRequest(ref Message request, IClientChannel channel,
InstanceContext instanceContext)
        {
            // Gelen request nesnesinin içeriği ele alınır ;)
            return null;
        }
        // Cevap istemci tarafına gönderilmeden önce çağırılacak. Tabi eğer istisnai bir durum
oluşmamışsa
        public void BeforeSendReply(ref Message reply, object correlationState)
        {
        }
    }
}
```

AfterReceiveRequest tahmin edileceği üzere istemciden ilgili operasyon için gelen talep alındığında devreye girmektedir. Burada mesaj içeriği henüz işlenmemiştir. Bir başka deyişle taleb edilen operasyon henüz çalıştırılmamıştır. Dolayısıyla gelen mesaj alınıp, istenirse değiştirilebilir *(Message tipinden olan request değişkeninin ref ile tanımlandığına dikkat edelim. Dolayısıyla runtime' da mesajın geldiği yerde bulunan mesaj içeriği buradan değiştirilebilir)*

BeforeSendReply metodu ise istemcinin talep ettiği operasyonun işlenmesi sonrası cevap dönülmeden önce devreye giren fonksiyondur. Burada da dikkat edileceği üzere dönüş için kullanılan mesaj ele alınabilir ve istenirse değiştirilebilir.

Örneğimizde her iki operasyon içeriği de boş bırakılmıştır. Siz, söz konusu implementasyonu yaparken bir senaryo üzerinden gidebilirsiniz. Örneğin gelen mesaj içeriğinde yer alan Türkçe karakterleri ayrıştırma veya dönüş sırasında üretilen mesajı kendi geliştirdiğiniz ve şirketinize özel algoritmaya göre şifreleyerek göndermek gibi

Mesaj tabanlı kesici tip geliştirildikten sonra, parametre kesicilerindekine benzer olarak, çalışma zamanına bir bildirimde bulunmamız gerekecektir. Bu sefer bu bildirimi konfigurasyon bazlı olacak şekilde gerçekleştireceğiz. Bu amaçla bir **Endpoint** davranışı belirleyip bunu konfigurasyon dosyasında değerlendirilebilecek şekilde ele almamız yeterlidir.

Ancak istenirse sözleşme bazlı bir davranış da bildirilebilir. Bunun için de IContractBehavior arayüzünden yararlanılmalıdır.

İlk olarak aşağıdaki tipi geliştirelim.

```
using System.ServiceModel.Channels;
using System.ServiceModel.Description;
using System.ServiceModel.Dispatcher;
namespace SomeServiceApp
{
```

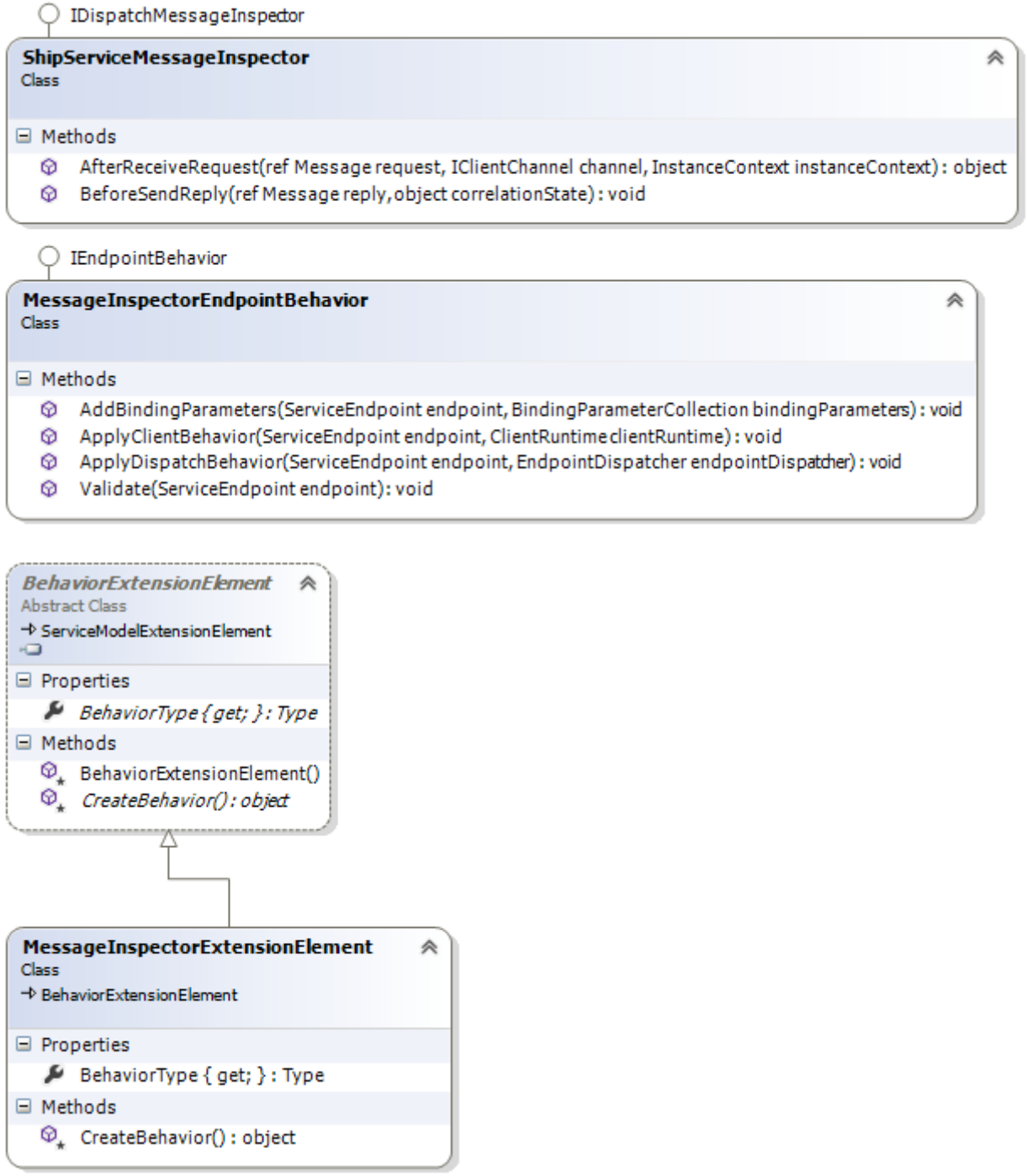
```
   public class MessageInspectorEndpointBehavior
      :IEndpointBehavior
   {
      public void AddBindingParameters(ServiceEndpoint endpoint,
BindingParameterCollection bindingParameters)
      {
      }
      public void ApplyClientBehavior(ServiceEndpoint endpoint, ClientRuntime
clientRuntime)
      {
      }
      public void ApplyDispatchBehavior(ServiceEndpoint endpoint,
EndpointDispatcher endpointDispatcher)
      {
         endpointDispatcher.DispatchRuntime.MessageInspectors.Add(new
ShipServiceMessageInspector());
      }
      public void Validate(ServiceEndpoint endpoint)
      {
      }
   }
}
```

IEndpointBehavior arayüzünü uygulayan **MessageInspectorEndpointBehavior** tipi içerisinde**ApplyDispatchBehavior** metodunun işlendiği görülmektedir. Bu metod içerisinde WCF çalışma zamanında bir **Endpoint** davranışının bildirilmesi işlemi icra edilmektedir. Buna göre herhangibir**Endpoint** için **ShipServiceMessageInspector** isimli bir davranış tipi belirlenebilecektir. İkinci olarak söz konusu davranışın konfigurasyon dosyasında ele alınabilmesini sağlamak adına bir**Element** tipi geliştirilmelidir. Aynen aşağıda görüldüğü gibi.

```
using System;
using System.ServiceModel.Configuration;
namespace SomeServiceApp
{
   public class MessageInspectorExtensionElement
      :BehaviorExtensionElement
   {
      public override Type BehaviorType
     {
         get { return typeof(MessageInspectorEndpointBehavior); }
      }
```

```
        protected override object CreateBehavior()
        {
          return new MessageInspectorEndpointBehavior();
        }
    }
}
```

Buraya kadar geliştirdiğimiz tiplerin özeti aşağıdaki sınıf çizelgesinde*(Class Diagram)* olduğu gibidir.



Artık servis uygulamamızı geliştirdiğimiz **WCF Serice Application** projesindeki **web.config**dosyasında aşağıdaki gibi özel bir Endpoint davranışı belirleyebiliriz.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
     <services>
       <service name="SomeServiceApp.ShipService">
         <endpoint address="" binding="basicHttpBinding"
contract="SomeServiceApp.IShipService"behaviorConfiguration="ShipServiceEndpoin
tBehavior" />
       <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange"
/>
       </service>
     </services>
       <behaviors>
          <serviceBehaviors>
            <behavior name="">
              <serviceMetadata httpGetEnabled="true"/>
              <serviceDebug includeExceptionDetailInFaults="true" />
            </behavior>
          </serviceBehaviors>
       <endpointBehaviors>
       <behavior name="ShipServiceEndpointBehavior">
         <messageInspector/>
       </behavior>
     </endpointBehaviors>
    </behaviors>
   <extensions>
     <behaviorExtensions>
       <add name="messageInspector"
type="SomeServiceApp.MessageInspectorExtensionElement, SomeServiceApp"/>
     </behaviorExtensions>
   </extensions>
   </system.serviceModel>
  <system.web>
   <compilation debug="true"/>
  </system.web>
</configuration>
```

İlk olarak **extensions** elementi altında, **MessageInspectorExtensionElement** tipi bildirilmiştir. Ancak bu bildirimden sonra **messageInspector** adıyla bir elementin

kullanılması ve geçerli olması söz konusu olabilir. Bu tanımlamanın ardından görüldüğü üzere **ShipServiceEndpointBehavior**isimli bir behavior bildirilmiş ve içerisinde **messageInspector** alt elementi kullanılmıştır. Son olarak bu davranışın **Endpoint** için bildiriminin yapılması yeterli olmuştur.
Yapılan bu ilavelerin ardından istemci tarafındaki **proxy** tipinin de güncelleştirilmesi gerekir. Eğer istemci ve servis tarafı **Debug** edilerek ilerlenirse mesaj bazlı kesici içerisinde yer alan**AfterReceiveRequest** ve **BeforeSendReply** operasyonlarında aşağıdaki mesaj içeriklerinin hareket ettiği gözlemlenebilir.
İlk çağırılan **SaveWorkItem** operasyonunda **AfterReceiveRequest'** e gelen mesaj içeriği,

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <To s:mustUnderstand="1"
xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">
http://localhost:56119/ShipService.svc</To>
    <Action s:mustUnderstand="1"
xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">
http://tempuri.org/IShipService/SaveWorkItem</Action>
  </s:Header>
  <s:Body>
    <SaveWorkItem xmlns="http://tempuri.org/">
      <item xmlns:a="http://schemas.datacontract.org/2004/07/SomeServiceApp"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
        <a:CreatedDate>2012-10-30T13:21:34.2638133+02:00</a:CreatedDate>
        <a:Owner>Burak</a:Owner>
        <a:Title>Login sayfasına ait Web User Control geliştirilmesi</a:Title>
        <a:Type>Task</a:Type>
      </item>
    </SaveWorkItem>
  </s:Body>
</s:Envelope>
```

İlk çağırılan **SaveWorkItem** operasyonunda dönüşe geçildikten sonra, **BeforeSendReply**metodunda ele alınan mesaj içeriği,

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1"
xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">
http://tempuri.org/IShipService/SaveWorkItemResponse</Action>
  </s:Header>
  <s:Body>
    <SaveWorkItemResponse xmlns="http://tempuri.org/">
      <SaveWorkItemResult>Login sayfasına ait Web User Control geliştirilmesi[Task]
```

10/30/2012 1:29:39 PM tarihinde Burak tarafından oluşturuldu</SaveWorkItemResult>
</SaveWorkItemResponse>
</s:Body>
</s:Envelope>

İkinci olarak yapılan **ProcessCadImage** operasyon için **AfterReceiveRequest** metoduna gelen mesaj içeriği,

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <To s:mustUnderstand="1"
xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">
http://localhost:56119/ShipService.svc</To>
    <Action s:mustUnderstand="1"
xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">
http://tempuri.org/IShipService/ProcessCadImage</Action>
  </s:Header>
  <s:Body>
    <ProcessCadImage xmlns="http://tempuri.org/">
      <sourceData>AAAAAAAA(burada bi sürü A vardı)=</sourceData>
    </ProcessCadImage>
  </s:Body>
</s:Envelope>
```

İkinci olarak yapılan ProcessCadImage operasyonundan dönülürken ele alınan mesaj içeriği,

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <Action s:mustUnderstand="1"
xmlns="http://schemas.microsoft.com/ws/2005/05/addressing/none">
http://tempuri.org/IShipService/ProcessCadImageResponse</Action>
  </s:Header>
  <s:Body>
    <ProcessCadImageResponse xmlns="http://tempuri.org/">
      <ProcessCadImageResult>true</ProcessCadImageResult>
    </ProcessCadImageResponse>
  </s:Body>
</s:Envelope>
```

Mesaj bazlı kesici tip, **Endpoint** seviyesinde bir davranış olarak tanımlandığından, bu **Endpoint**adına gelen ne kadar operasyon var ise her biri için devreye girecektir. Diğer yandan **Parametre** bazlı kesicimizin bir **FaultException** üretmesi halinde, **BeforeSendReply** metodları hatanın üretildiği servis operasyonu için devreye girmiyor olacaktır.

Görüldüğü üzere parametre ve mesaj seviyesinde kesme işlemlerini uygulamak çok zor olmadığı gibi dikkat gerektiren bir işlemler bütününden oluşmaktadır. Bu yazımızda **Parametre** ve **Mesaj** bazlı kesici tiplerin **WCF** tarafında nasıl yazılabileceğini incelemeye çalıştık. Bu kesicilerin bir diğer implementasyonu da **WCF Data Service** tarafındadır. Bu konuyu da incelemenizi tavsiye ederim. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

[Bu makale Formspring üzerinden gelen bir soru üzerine hazırlanmıştır. Soruyu soran arkadaşımıza teşekkür etmek isterim]

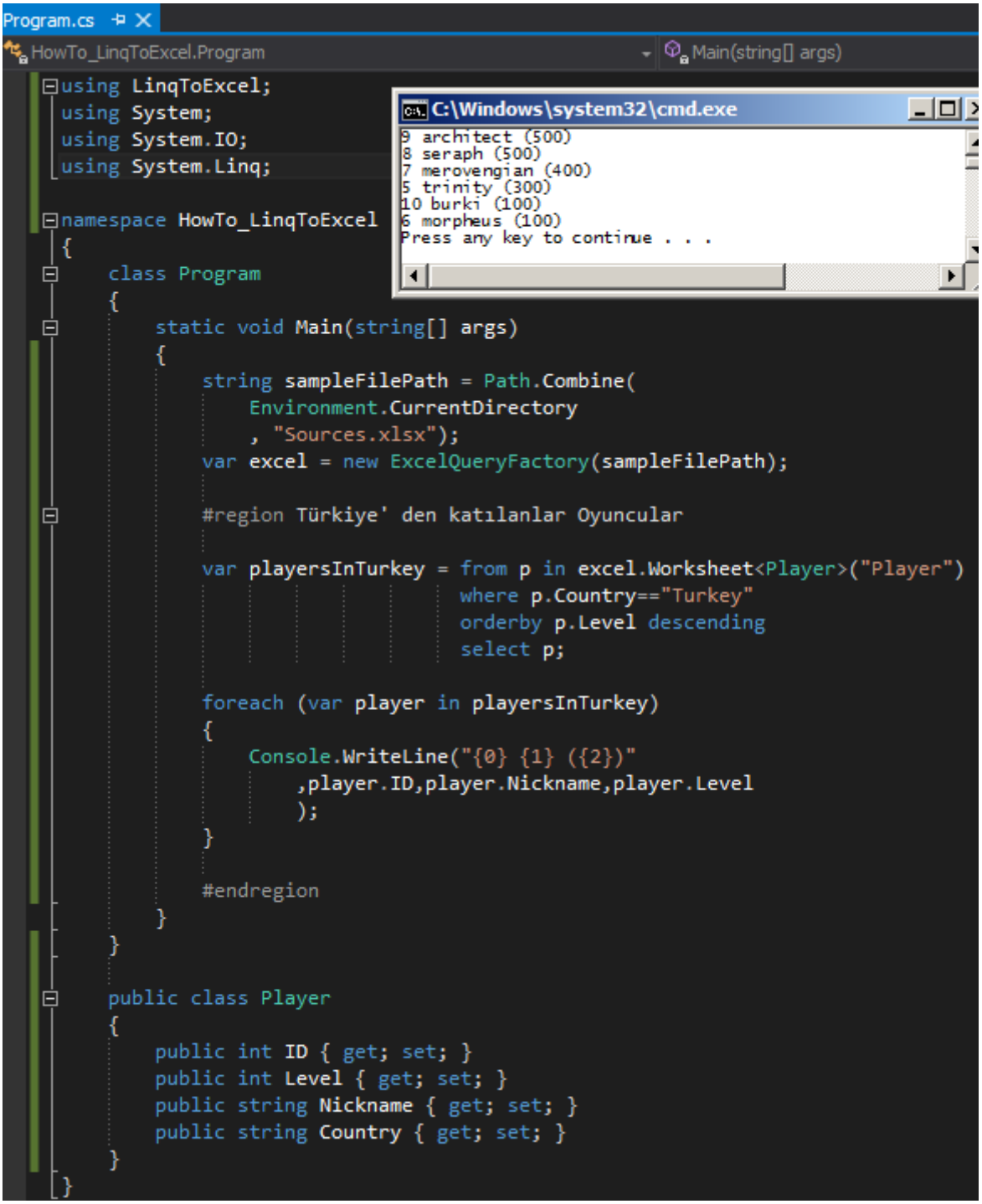
HowTo_WCF_Interceptors.zip (86,34 kb)

Tek Fotoluk İpucu–73–LINQ to Excel için Strongly Typed Tip Kullanmak

Salı, 27 Kasım 2012 15:10

Tek Fotoluk Ipucu, Linq To Excel, Excel, Linq, C#, Query

Merhaba Arkadaşlar,
Bir önceki ip ucunda **LINQ to Excel Provider**' dan yararlanmış ve bir **Excel** dosyasını kolayca nasıl sorgulayabileceğimizi görmüştük. Peki ya **Excel** tablosunda yer alan satırları, kod tarafında oluşturacağımız **Strongly Typed** sınıflar içerisindeki özelliklere karşılık gelecek şekilde ifade edebiliyor olsaydık Daha şık olmaz mıydı?
İşte size basit bir örnek. Tek dikkat etmemiz gereken, **Excel** tablosundaki kolonların başlıkları ile aynı isimde olan ve veri türü olarak da dönüştürülebilir tipteki **özellikleri(Property)** içeren basit bir**POCO** sınıfı tasarlamamız ve bunu **Worksheet<T>** çağrısında **T** yerine kullanmamız. Bir başka deyişle **Worksheet** ile doğru şekilde eşleştirilebilecek bir **POCO(Plain Old CLR Object)** sınıfına ihtiyacımız var; o kadar. Hepsi bu



Bir başka ip ucunda görüşünceye dek hepinize mutlu günler dilerim.

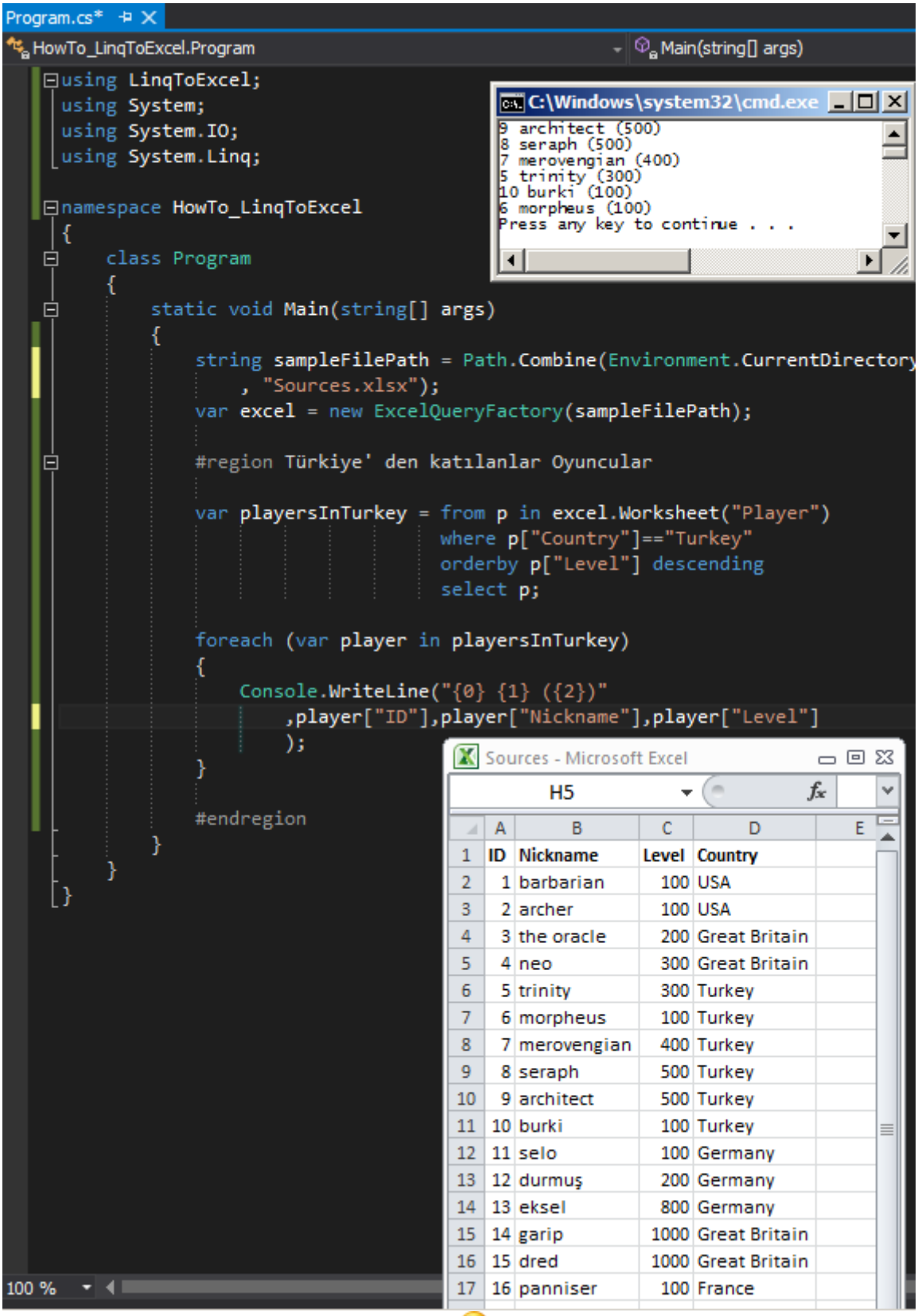
[Not:Provider' da değişiklik yapılmış olabilir ve yukarıdaki kod parçasının son sürümde daha farklı bir şekilde ele alınması gerekebilir. Buna dikkat edelim]

Tek Fotoluk İpucu–72–LINQ to Excel ile Basit Sorgulama

Cuma, 23 Kasım 2012 09:10

Tek Fotoluk Ipucu, Linq, Linq To Excel, Linq Providers, Excel, C#

Merhaba Arkadaşlar,
Zaman zaman siz de benim gibi **LINQ** ile yazılmış çeşitli **Provider'** lara şöyle bir göz gezdirenlerden misiniz? Belki siz de **LINQtoEXCEL** provider' ını duymuşsunuzdur. **NuGet** paket yönetim aracı yardımıyla indirip kullanabileceğiniz bu sağlayıcıyı duymadıysanız eğer işte size basit bir örnek.**Excel** kaynaklarını **LINQ** ile kolay bir şekilde sorgulamanıza olanak sağlayan bu sağlayıcının bu adreste daha detaylı kullanımı ile ilişkili güzel örnekler var. **WorkSheet**' ler içerisindeki **Row'** lara karşılık gelecek **property'** ler içeren **Strongly Typed** sınıfların kullanımı vb.



	A	B	C	D	E
1	ID	Nickname	Level	Country	
2	1	barbarian	100	USA	
3	2	archer	100	USA	
4	3	the oracle	200	Great Britain	
5	4	neo	300	Great Britain	
6	5	trinity	300	Turkey	
7	6	morpheus	100	Turkey	
8	7	merovengian	400	Turkey	
9	8	seraph	500	Turkey	
10	9	architect	500	Turkey	
11	10	burki	100	Turkey	
12	11	selo	100	Germany	
13	12	durmuş	200	Germany	
14	13	eksel	800	Germany	
15	14	garip	1000	Great Britain	
16	15	dred	1000	Great Britain	
17	16	panniser	100	France	

Başka bir ipucunda görüşmek dileğiyle

[Not:Provider' da değişiklik yapılmış olabilir ve yukarıdaki kod parçasının son sürümde daha farklı bir şekilde ele alınması gerekebilir. Buna dikkat edelim]

Tek Fotoluk İpucu–71–IQueryable veya IEnumerable

Salı, 20 Kasım 2012 16:01

Tek Fotoluk Ipucu, Entity Framework, Ienumerable<T>, Iqueryable<T>

Merhaba Arkadaşlar,
Bu sefer ki ip ucumuz biraz daha kışkırtıcı aslında. Aşağıdaki fotoğrafı bir inceleyin öncelikle ve nasıl bir fark olabileceğini düşünmeye çalışın. Yani kafanızda kod parçasını **debug** etmeye gayret edin.
(Visual Studio ve benzeri herhangibir geliştirme aracı kullanmamanız şiddetle tavsiye edilir 😆)

QueryableOrEnumerable.Program

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;

namespace QueryableOrEnumerable
{
    class Program
    {
        static void Main(string[] args)
        {
            IQueryable<Customer> customerList1 = null;
            IEnumerable<Customer> customerList2 = null;

            using (ChinookEntities context = new ChinookEntities())
            {
                customerList1 = from c in context.Customers
                                orderby c.LastName, c.FirstName
                                select c;

                customerList2 = from c in context.Customers
                                orderby c.LastName, c.FirstName
                                select c;

                foreach (var c in customerList1.Take(10))
                    Console.WriteLine(c.LastName);

                foreach (var c in customerList2.Take(10))
                    Console.WriteLine(c.LastName);
            }
        }
    }
}
```

Tabi fotoğrafa bakınca durumu görmek zor olabilir. Ama fotoğrafın arkasında yatan gerçeklere bakarsak*(örneğin SQL Server Profiler yardımıyla)* **customerList1** üzerinden uygulanan **Take(10)**çağrısı için aşağıdaki SQL sorgusunun çalıştırıldığını görürüz.

```
SELECT TOP (10)
[Extent1].[CustomerId] AS [CustomerId],
[Extent1].[FirstName] AS [FirstName],
```

```
[Extent1].[LastName] AS [LastName],
[Extent1].[Company] AS [Company],
[Extent1].[Address] AS [Address],
[Extent1].[City] AS [City],
[Extent1].[State] AS [State],
[Extent1].[Country] AS [Country],
[Extent1].[PostalCode] AS [PostalCode],
[Extent1].[Phone] AS [Phone],
[Extent1].[Fax] AS [Fax],
[Extent1].[Email] AS [Email],
[Extent1].[SupportRepId] AS [SupportRepId]
FROM [dbo].[Customer] AS [Extent1]
ORDER BY [Extent1].[LastName] ASC, [Extent1].[FirstName] ASC
```

customerList2 üzerinden yapılan **Take(10)** içinse benzer bir sorgu üretilir.

```
SELECT
[Extent1].[CustomerId] AS [CustomerId],
[Extent1].[FirstName] AS [FirstName],
[Extent1].[LastName] AS [LastName],
[Extent1].[Company] AS [Company],
[Extent1].[Address] AS [Address],
[Extent1].[City] AS [City],
[Extent1].[State] AS [State],
[Extent1].[Country] AS [Country],
[Extent1].[PostalCode] AS [PostalCode],
[Extent1].[Phone] AS [Phone],
[Extent1].[Fax] AS [Fax],
[Extent1].[Email] AS [Email],
[Extent1].[SupportRepId] AS [SupportRepId]
FROM [dbo].[Customer] AS [Extent1]
ORDER BY [Extent1].[LastName] ASC, [Extent1].[FirstName] ASC
```

 Uppsss!!! Benzer bir sorgu derken!

IQueryable üzerinden yapılan **Take(10)** çağrısı dikkat edileceği üzere **TOP 10** ifadesini kullanmıştır. Peki ya **IEnumerable** üzerinden yapılan **Take(10)** çağrısı ne yapmıştır  Aslında tüm liste çekilmiş sonrasında **Take** metodu, belleğe aldığı koleksiyon seti üzerinden ilk **10luk** parçayı almıştır.

Sanırım artık **IQueryable** mı olsun, **IEnumerable** mı olsun çıktı sonucu ya da **var** anahtar kelimesini kullanırsak hangisini göz önüne alır, hangisi daha avantajlıdır diye bir kuşku oluşturmuş bulunmaktayım içinizde  E hadi hayırlısı diyelim.

Başka bir ipucunda görüşmek dileğiyle.

Entity Framework Code First için Doğrulama(Validation) Stratejileri

Pazar, 18 Kasım 2012 17:38

Entity Framework, Code First, Validation, Attribute, Data Annotations

Merhaba Arkadaşlar,

[Bu yazıda ele alınan konuları içeren Webiner(Webcast) kayıtlarına Nedirtv?com sayfasından erişebilir, izleyebilir ve indirebilirsiniz]

Bir verinin çeşitli kurallara göre doğrulanması, verinin işlenmek üzere gönderilmeden önce yapılması gereken önemli işlemlerden birisidir. Özellikle **Entity Framework**gibi **veri merkezli(Data-Centric)** uygulama geliştirme alt yapılarında bu durum daha da önem arz etmektedir.

Burada söz konusu olan, görsel bir kontrolün içerik denetiminden ziyade, çalışma zamanı **Entity** örneklerine ait**özelliklerin(Property)** değerlerinin denetlenmesidir. Çok doğal olarak verilerde tutarsızlıklara neden olabilecek çeşitli ihlallerin tespit edilmesi, toplanması, gerektiğinde son kullanıcıya bildirilmesi ya da farklı bir yere raporlanması/loglanması gerekmektedir. Peki verinin doğrulanmasından tam olarak beklentilerimiz neler olabilir? Bunu bir kaç gerçek hayat ihtiyacı ile cevaplayabiliriz.

Örneğin,

- Kullanıcı isminin en az 5, en fazla 25 karakter olması istenebilir.
- Oyuncunun kullanmak istediği dil sadece ingilizce, almanca ve fransızca olsun, şeklinde bir zorlama yapılması söz konusu olabilir.
- Doğum tarihinin bu günün ötesinde olmaması gerekebilir.
- Girilen URL adresinin istenen formatta olması beklenebilir.
- Yazarın vermiş olduğu sosyal güvenlik numarasının gerçekten de var olmaması bir doğrulama ihlali olarak düşünülebilir.
- vb...

Örnekler duruma göre çoğaltılabilir elbette. **Entity Framework**, özelliklerin doğrulanması için bir kaç noktada araya girmemizi sağlayacak imkanlar sunmaktadır. Biz bu örneğimizde **Code-First**yaklaşımı üzerinden ilgili doğrulama kurallarını hangi noktalardan ve nasıl enjekte edebileceğimizi incelemeye çalışıyor olacağız. Doğrulama işlemlerini temelde **Entity** ve **Context** seviyesinde olmak üzere iki ana dala ayırabiliriz.

Doğrulama kuralları(Validation Rules), Entity seviyesinde iki şekilde yaptırılabilir.**Nitelik(Attribute)** bazlı veya **IValidatableObject** arayüzünün implementasyonu ile. Nitelik bazlı enjektelerde, **System.ComponentModel.DataAnnotations isim alanı(namespace)** altında yer alan bazı nitelik tiplerinden yararlanılır.

Context seviyesinde ise **ValidateEntity** sanal metodunun **ezilmesi(override)** suretiyle söz konusu denetimler yaptırılabilir.
Hangi teknik seçilirse seçilsin, ilgili doğrulama kurallarının geçersiz olması halinde, ortama fırlatılan**Exception'** ların da kümülatif olarak toplanıp sunulması önemlidir. Bilindiği üzere normal şartlarda, kod akarken oluşan bir **Exception** sonucu çalışma zamanı catch bloğuna atlayacak ve **try** bloğu içerisindeki akışına devam etmeyecektir. **Entity Framework** gibi kullanım alanlarında, birden fazla Entity söz konusu olmakla birlikte bunların herhangibirisine ait özelliklerde oluşacak olan doğrulama ihlallerin toplanıp sunulması çok daha doğru bir yaklaşım olacaktır.
Entity Framework bu noktada bize yardımcı olacak bir Exception tipi içerir;**DbEntityValidationException**. Bu **exception** tipine ait **EntityValidationErros** koleksiyonu,**Entity'** ler için söz konusu olabilecek tüm doğrulama ihlallerini bünyesinde toplamaktadır. Bu sayede kullanıcıya toplu bir hata listesini döndürmemiz mümkün olabilir.
Şimdi dilerseniz basit bir **Console** uygulaması üzerinden söz konusu teknikleri irdelemeye çalışalım.
Code-First yaklaşımını tercih ettiğimiz bu örnekte NuGet yardımıyla ilgili Entity Framework paketinin yüklenmiş olduğunu varsayıyoruz. Örneği Entity Framework' ün 4.5 sürümü üzerinde geliştirmekteyiz.
İlk olarak aşağıdaki sınıf çizelgesinde yer alan **Layer POCO** tipini geliştirdiğimizi düşünelim.


```
using System.ComponentModel.DataAnnotations;
namespace HowTo_Validation
{
   public class Layer
   {
      public int LayerId { get; set; }

      [MaxLength(25,ErrorMessage="Katman başlığı en fazla 25 karakter olabilir")
      , Required(ErrorMessage="Bir başlık girilmelidir")]
      public string Title { get; set; }

     [Required(ErrorMessage="Maximum oyuncu kapasitesini girmelisiniz")
```

```
      , Range(5,100,ErrorMessage="En az 5 en fazla 100 oyuncu olabilir")]
      public int MaxPlayerCapacity { get; set; }
      [RegularExpression(@"^([\w\.\-]+)@([\w\-]+)((\.(\w){2,3})+)$", ErrorMessage =
"Geçersiz posta adresi")]
      public string ModeratorMail { get; set; }
   }
}
```

Layer tipi içerisinde yer alan **Title**, **MaxPlayerCapacity** ve **ModeratorMail** özelliklerine çeşitli nitelikler uygulandığı görülmektedir. Bu niteliklerin **ErrorMessage** kısmında genellikle ihlallere ilişkin bir mesaj kullanılır. **MaxLength**, tahmin edileceği üzere ilgili özelliğin maksimum karakter sayısını ifade etmektedir. Diğer yandan **Range** niteliği ile sayısal bir özellik için gerekli olan alt ve üst sınır değerleri belirtilmektedir.
Required niteliği, ilgili özelliğin mutlaka girilmesini gerektirir. Kullanılabilecek olan bir diğer faydalı nitelikte*(ki bana göre en işe yararlarından birisidir)* **RegularExpression'** dır. İlk parametre olarak bir**RegEx** ifadesi almaktadır. Örneğimzde **ModeratorMail** özelliğinin taşıyacağı çalışma zamanı değerinin geçerli bir **e-mail** adresi olup olmadığı kontrol edilmektedir.
Entity' lere ait özellikler seviyesinde yapılabilen bu doğrulama işlemleri özellikle tipin görsel bileşenlere **bağlanabildiği(Binding)** uygulama çeşitlerinde epey kullanışlıdır. Bir başka deyişle**Model View Controller(MVC)** veya **Model View View Model(MVVM)** tarzı yapılarda değerlendirilebilir. Tabi **Entity** içerisinde çok fazla sayıda özellik olabilir ve her biri için ilgili doğrulama kriterleri ortaklık gösterebilir. Söz gelimi hiç bir **Entity** özelliğinin boş geçilmemesi istenebilir.
Diğer yandan nitelikler seviyesinde gerçekleştirilmesi pek kolay olmayan bazı doğrulama kuralları da söz konusu olabilir ve bunlar farklı yerlerde fonksiyonel hale gelmiş kütüphaneler içerisinde yer alabilirler. Bu durumda sanki **Entity** seviyesinde söz konusu olabilecek bir metod daha ideal olabilir. Bu tip bir vakayı karşılamak için **Entity** tipine yine **System.ComponentModel.DataAnnotations**içerisinde yer alan **IValidatableObject** arayüzünü uyarlamak ve beraberinde gelen Validate metodunu ezmek yeterlidir. Şimdi örneğimize aşağıdaki sınıf çizelgesinde görülen **Player POCO(Plain Old CLR Object)** tipini eklediğimizi düşünelim.

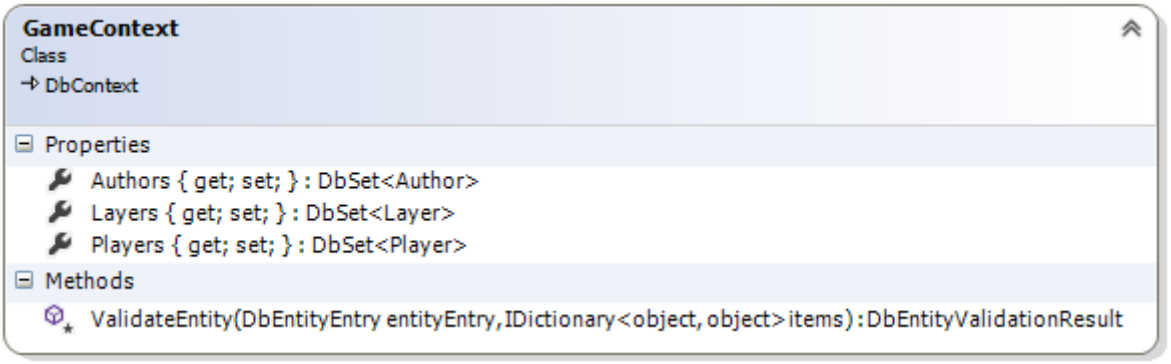

```
using System;
using System.Collections.Generic;
using System.ComponentModel.DataAnnotations;
using System.Linq;
namespace HowTo_Validation
{
   public class Player
     :IValidatableObject
   {
      public int PlayerId { get; set; }
      public string NickName { get; set; }
      public string FirstName { get; set; }
      public short Level { get; set; }
      public string Country { get; set; }
      public IEnumerable<ValidationResult> Validate(ValidationContext
validationContext)
      {
         if (String.IsNullOrEmpty(NickName) || String.IsNullOrEmpty(FirstName) ||
String.IsNullOrEmpty(Country))
            yield return new ValidationResult("Nickname, FirstName veya Country
değerleri boş bırakılamaz"
               , new string[] { "NickName", "FirstName", "Country" }
               );
```

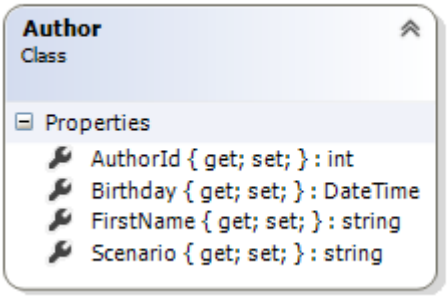
```
        if (NickName.Length < 3 || NickName.Length>10)
            yield return new ValidationResult("NickName en az 3 karakter en fazla 10
karakter olmalıdır",new string[]{"NickName"});
        if (!Constants.Levels.Contains(Level))
            yield return new ValidationResult("Geçersiz oyuncu seviyesi.", new string[]
{ "Level" });
        if (!Constants.Countries.Contains(Country))
            yield return new ValidationResult("Geçersiz ülke.", new string[] {
"Country" });
      }
   }
}
```

Bu örnekte **Player** tipinin **Nickname**, **FirstName**, **Country** ve **Level** özellikleri için uygulanmış olan bazı doğrulama kriterleri olduğu görülmektedir. Dikkat edileceği üzere Validate metodu, doğrulama işlemine ait çalışma zamanı içeriğini **ValidationContext** tipinden olan parametre ile kontrol altına almaktadır. Bu özelliğin değeri elbetteki çalışma zamanında asıl **Context** nesnesi tarafından dolduralacak ve o **Player** entity tipine ait canlı nesne değerleri ile beslenecektir. **Validate**metodu geriye bir numaralandırıcı döndürmektedir. Dolayısıyla **yield** anahtar kelimesinden yararlanılabilir ve bu sayede n sayıda doğrulama ihlalinin asıl ortama döndürülmesi mümkün olabilir. Dönüş koleksiyonu içerisinde yer alan tipler **ValidationResult** sınıfına ait örneklerdir. Bu sınıfa ait örnekler üretilirken genellikle ilk parametre olarak hata mesajı verilir. İkinci parametre ise ihlale sebebiyet veren özelliği ifade etmektedir. Bu iki bilgi yine çalışma zamanındaki **Catch** bloğunda yakalanan **DbEntityValidationException** örneği içerisinden alınabilir ve son kullanıcıya bilgilendirme de kullanılabilir.

Ancak bu yöntem için de bir dezavantaj da söz konusu olabilir. Uygulamada kullanılan **Entity**tiplerinin sayısı arttıkça ve benzer doğrulama kriterlerinin pek çok **Entity** için yapılması söz konusu ise attribute bazlı enjekte yöntemi terk edilebilir. Böyle bir durumda doğrudan Context nesnesi üzerinden bir doğrulama tekniği tercih edilebilir. Aynen aşağıda görüldüğü gibi.

Bu açılardan bakıldığında **Entity Framework'** deki doğrulama kontrolleri, **ASP.Net** tarafındaki hata yönetimini andırmaktadır. **Asp.Net** tarafında bildiğiniz üzere sırasıyla Metod, Sayfa(Page) ve uygulama(Application-global.asax.cs) seviyesinde hata kontrolleri gerçekleştirilir. **EF'** de de**property'** den başlayan, sınıf içi bir metod ile devam eden ve son olarak **context** nesnesi üzerinde ele alınabilen doğrulama enjekte noktaları mevcuttur. Hangi sırada çalıştıklarını merak ediyorsanız**Debug** edip denemenizi öneririm Belki de bu sıra duruma göre değişiklik arz, eder kim bilir






```
using System;
namespace HowTo_Validation
{
   public class Author
   {
      public int AuthorId { get; set; }
      public string FirstName { get; set; }
      public string Scenario { get; set; }
      public DateTime Birthday{ get; set; }
   }
}
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.Data.Entity.Validation;
namespace HowTo_Validation
{
   public class GameContext
      :DbContext
   {
      public DbSet<Player> Players { get; set; }
```

```
        public DbSet<Layer> Layers { get; set; }
        public DbSet<Author> Authors { get; set; }
        protected override DbEntityValidationResult ValidateEntity(DbEntityEntry
entityEntry,
IDictionary<object, object> items)
        {
            var result = base.ValidateEntity(entityEntry, items);
            if (entityEntry.State == EntityState.Added &&
                entityEntry.Entity is Author)
            {
                var author = entityEntry.Entity as Author;
                if (author.Birthday > DateTime.Today)
                {
                  result.ValidationErrors.Add(
                      new DbValidationError(
                        "Yazar doğum tarihi",
                        "Doğum tarihi bugünden büyük olamaz.")
                        );
                }
            }
            return result;
        }
    }
}
```

Standart olarak **DbContext** türevli olarak tasarlanan **GameContext** sınıfı içerisinde **ValidateEntity**metodunun ezildiği görülmektedir(override). Metod geriye **DbEntityValidationResult** tipnden bir örnek döndürmektedir. Olası **n sayıdaki kural ihlali**, bu tipin nesne örneğine ait **ValidationErros**koleksiyonunda toplanabilir. Örnekte o anda üzerinde işlem yapılan **Entity** örneği **DbEntityEntry** tipinden olan **entityEntry**isimli metod parametresidir. Bu parametreden yararlanılarak **State** özelliğine bakılır ve ayrıca ilgili**Entity'** nin bir **Author** tipi olup olmadığı tespit edilir. Eğer yeni bir yazar ekleniyorsa bu durumda doğrulama işlemi yaptırılmaktadır. Sembolik olarak yazarın doğum tarihinin bu günün tarihinden büyük olmaması istenmiştir. Eğer bir ihlal söz konusu ise bu durumda **ValidationErros** özelliğinin işaret ettiği koleksiyona yeni bir **DbValidationError** örneği eklenir.

Buraya kadar ki örneklerimizle doğrulama kriterlerini 3 farklı seviyede ele alabildiğimizi gördük. Şimdi **Program** kodu içerisinde gerekli **try...catch** bloğunu uygulayarak örneğimizi test edelim ve çalışma zamanı sonuçlarını irdeleyelim.

```
using System;
using System.Collections.Generic;
```

```
using System.Data.Entity;
using System.Data.Entity.Validation;
namespace HowTo_Validation
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Code First için Doğrulama Teknikleri
         // Initialization stratejisi olarak modelde bir değişiklik olursa yeni baştan
üretilmesini ifade ediyoruz
         Database.SetInitializer<GameContext>(new
DropCreateDatabaseIfModelChanges<GameContext>());
         // Context nesnesi örneklenir
         using (GameContext context = new GameContext())
         {
            try
            {
               Player jedi = new Player();
               jedi.NickName = "jd";
               jedi.Level = 5000;
               jedi.Country = "Coroban";
               context.Players.Add(jedi);
               Layer layerSubZero = new Layer();
               layerSubZero.Title = "bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla
bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla bla";
               layerSubZero.ModeratorMail = "birmailiste";
               layerSubZero.MaxPlayerCapacity = 200;
               context.Layers.Add(layerSubZero);
               Author me = new Author();
               me.FirstName = "Burak";
               me.Scenario = "Güzel bir oyun senaryosu var burada";
               me.Birthday = new DateTime(2014, 1, 1);
               context.Authors.Add(me);

               context.SaveChanges();
            }
            catch (DbEntityValidationException exception) // Validasyon ile ilişkili
Exception' lar yakalanır
            {
              WriteErrosToConsole(exception.EntityValidationErrors); // Exception
```

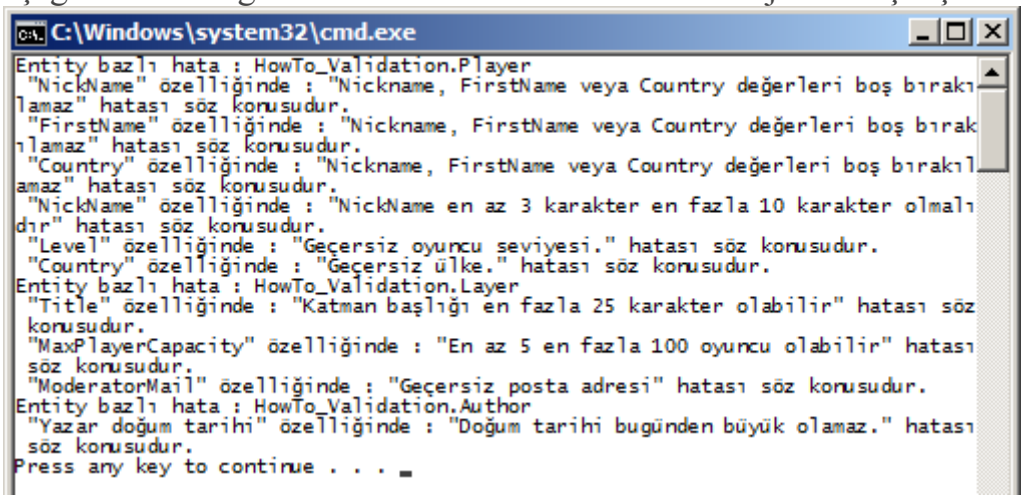
```
loglanmak amacıyla ilgili metoda gönderirilir
            }
          }
          #endregion
        }
        private static int
WriteErrosToConsole(IEnumerable<DbEntityValidationResult> validationErrors)
        {
          int errorCount = 0;
          // Her bir Validation Error dolaşılmaya başlanır
          foreach (DbEntityValidationResult validationError
            in validationErrors)
          {
            // Doğrulama hatasına neden olan Entity bilgisi verilir
            Console.WriteLine(
              "Entity bazlı hata : {0}",
              validationError.Entry.Entity);
            // Entity içerisinde doğrulama hatasına takılan özelliklerin her biri dolaşılır
            foreach (DbValidationError propertyError
              in validationError.ValidationErrors)
            {
              // Doğrulama hatasına takılan özelliğin adı ve hata mesajı yazdırılır
              Console.WriteLine(
                @" ""{0}"" özelliğinde : ""{1}"" hatası söz konusudur.",
                propertyError.PropertyName,
                propertyError.ErrorMessage);
            }
            errorCount++;
          }
          return errorCount;
        }
    }
}
```

Örnekte Code-First yaklaşımı kullanılmıştır. Bu yaklaşımda connection string bilgisi de önemlidir. Biz aksini belirtmedikçe uygulama SQL Express sürümü ve DbContext türevli tip adını baz alarak bir veritabanı oluşturacaktır. Biz örneğimizde aşağıdaki gibi bir connection string bilgisi kullandık.

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <configSections>
    <section name="entityFramework"
```

```
type="System.Data.Entity.Internal.ConfigFile.EntityFrameworkSection,
EntityFramework, Version=5.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" requirePermission="false" />
  </configSections>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
  </startup>
  <entityFramework>
    <defaultConnectionFactory
type="System.Data.Entity.Infrastructure.SqlConnectionFactory, EntityFramework"
/>
  </entityFramework>
  <connectionStrings>
    <add name="GameContext" connectionString="data
source=.;database=Gamers;integrated security=SSPI"
providerName="System.Data.SqlClient"/>
  </connectionStrings>
</configuration>
```

Main metodu içerisinde üretilen **GameContext** örneği için örnek bir **Player**, **Author** ve **Layer**örneğinin eklenmesi söz konusudur. Bu örnekler eklendikten sonra yapılan **SaveChanges** çağrısı sırasında ilgili doğrulama kodları devreye girecek ve olası hatalar toplanarak catch bloğu içerisinde yakalanacaktır. Bu hataları kolay bir şekilde yazdırabilmek amacıyla, **WriteErrorsToConsole** isimli bir metoddan yararlanılmaktadır. Dikkat edileceği üzere bu metod içerisinde ihlale neden olan Entity örneği**DbEntityValidationResult** örneklerine ait **Entry.Entity** özelliğinden yakalanmaktadır. **Entity'** den hemen bir alt seviye olan özelliklerdeki ihlallere inmek için de **ValidationErros** koleksiyonunda dolaşılmakta ve **PropertyName** ile **ErrorMessage** değerlerine bakılmaktadır. Örneği çalıştırdığımızda aşağıdaki ekran görüntüsündeki benzer bir hata mesajı ile karşılaşırız.

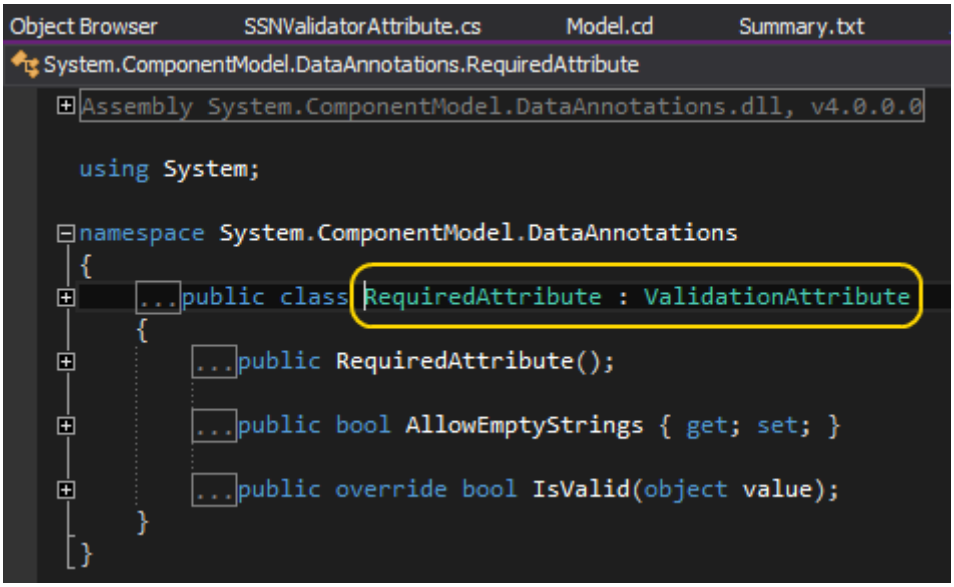


Görüldüğü gibi tüm seviyelerdeki kural ihlalleri toplu olarak yakalanabilmiştir.

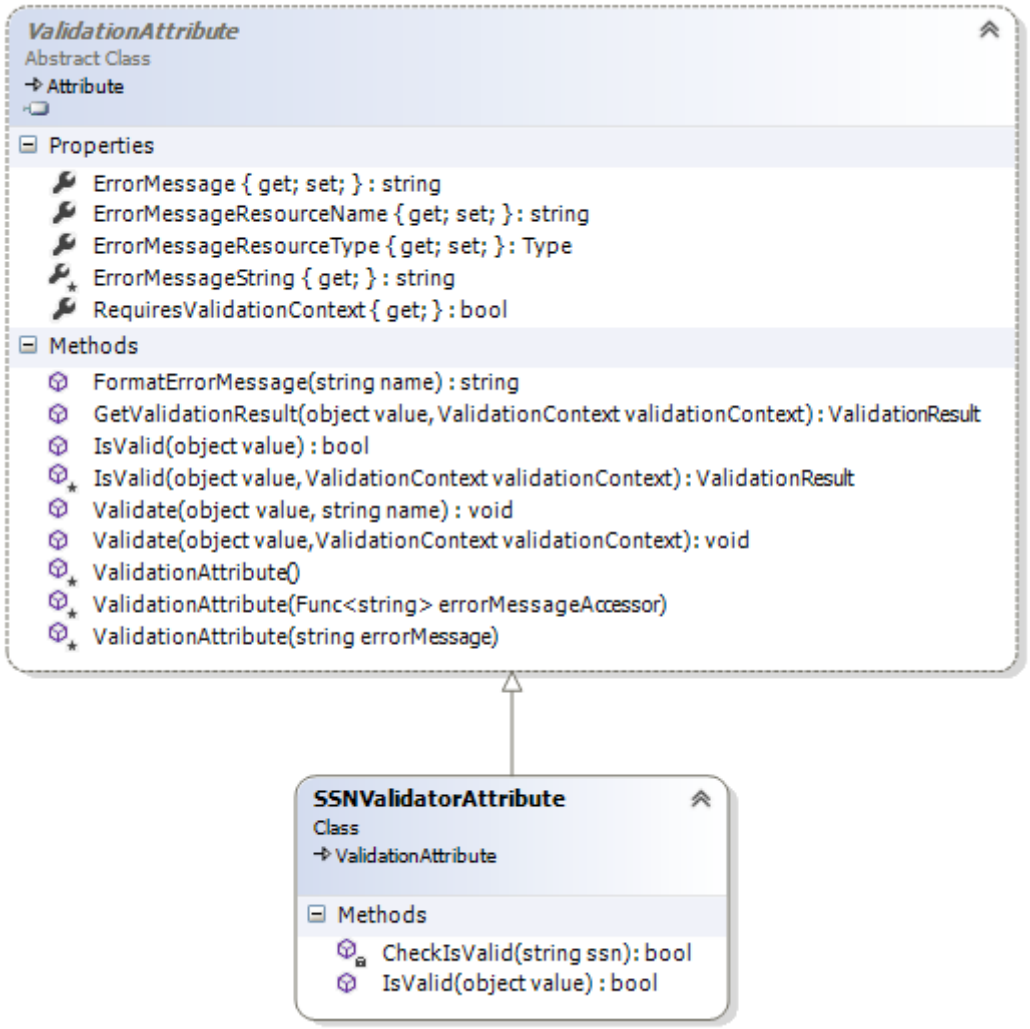
Örneği çalıştırırken mutlaka debug edip adım adım ilerlemenizi öneririm. Bu sayede kodun sırasıyla hangi doğrulama kriterlerini çalıştırdığını daha net görebilirisiniz ;)
Örnekte dikkati çeken noktalardan birisi de, ilgili doğrulama işlemlerinin **SaveChanges** metoduna yapılan çağrı ile devreye girmiş olmasıdır. Çok doğal olarak bu çağrıyı yapmadan önce bir yerlerde ilgili doğrulama kurallarını çalıştırmak ve olası ihlalleri toplamak isteyebiliriz. Bu tip bir durumda yine **Context** nesnesine ait olan **GetValidationErros** metodundan yararlanılabilir. Aşağıdaki kod parçasında olduğu gibi.

```
var validationErros = context.GetValidationErrors();
var errorCount=WriteErrosToConsole(validationErros);
if(errorCount==0)
   context.SaveChanges();
```

GetValidationErros metodu **IEnumerable<DbEntityValidationResult>** tipinden bir referans döndürmektedir. Çok doğal olarak bu içerik en az 1 ihlal dahi içerse **context** nesne örneğine ait**SaveChanges** metodunun çağırılması istenmeyebilir.
Peki özellikle **attribute** seviyesinde yapılan doğrulama kontrollerini göz önüne alırsak, kendi özel kriterlerimizi içeren nitelikler tanımlayamaz mıyız? Elbetteki böyle bir esnekli var Nitekim**System.ComponentModel.DataAnnotations** isim alanı altında yer alan doğrulama kriterlerinin ortak özelliği, **ValidationAttribute** niteliğinden türemiş olmalarıdır.



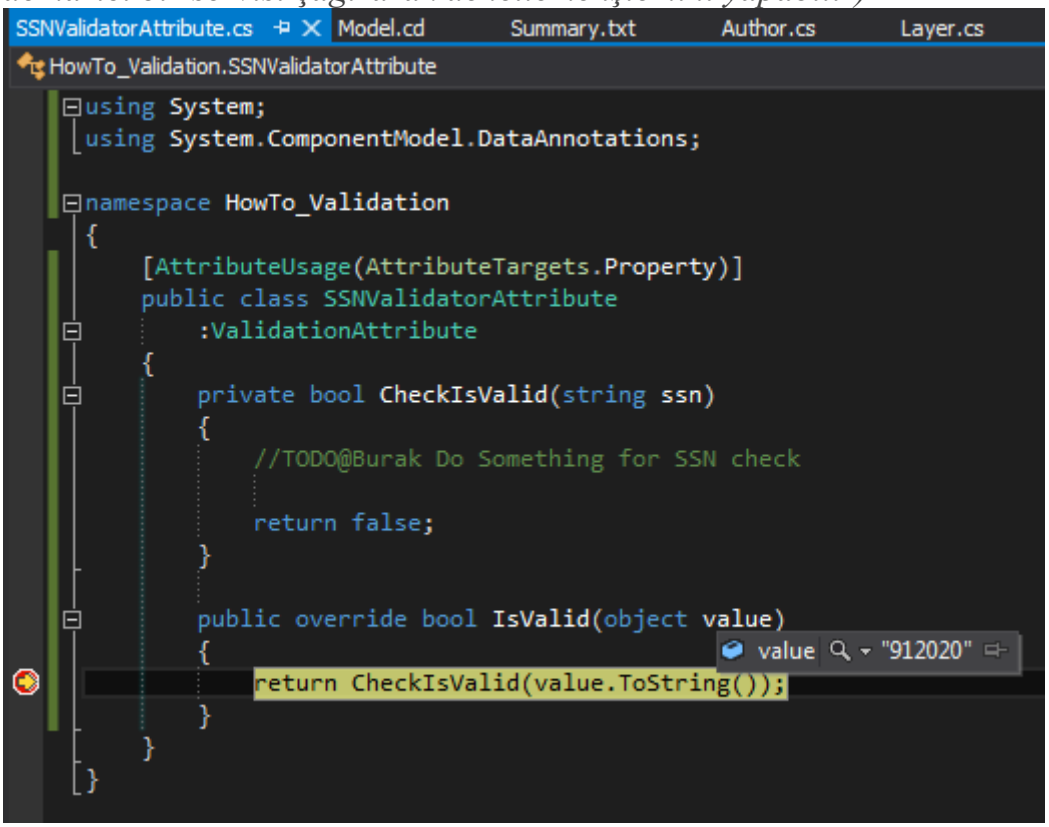
ValidationAttribute niteliği de doğal olarak **Attribute** tipinden türemektedir. Öyleyse kendi doğrulama niteliklerimizi yazmanın bir yolunu bulduğumuzu ifade edebiliriz Söz gelimi **Author**tipimize **SocialSecurityNumber** isimli **string** bir özellik eklediğimizi ve buraya girilen değerlerin geçerli bir numara olup olmadığını denetleyecek bir doğrulama niteliği geliştirmek istediğimizi farz edelim. Aşağıdaki şekilde ilerleyebiliriz.




```
using System;
using System.ComponentModel.DataAnnotations;
namespace HowTo_Validation
{
   [AttributeUsage(AttributeTargets.Property)]
   public class SSNValidatorAttribute
      :ValidationAttribute
   {
      private bool CheckIsValid(string ssn)
      {
         //TODO@Burak Do Something for SSN check
         return false;
      }
      public override bool IsValid(object value)
      {
         return CheckIsValid(value.ToString());
      }
```

```
    }
}
```

Tabi duruma göre söz konusu niteliğin ezmesi gereken üye sayısı daha fazla olabilir. Biz örneğimizde sadece **IsValid** metodunu ezdik. Çalışma zamanında bu metoda girildiğinde **object** tipinden olan**value** parametresinin değeri, niteliğin uygulandığı özelliğin çalışma zamanındaki içeriği olacaktır.*(Burada sembolik olarak kontrol işlemini üstlenen ayrı bir metod private olarak tanımlanmıştır. Gerçek hayatta bu metod gerçekten de harici bir servisi çağırarak denetleme işlemini yapabilir)*

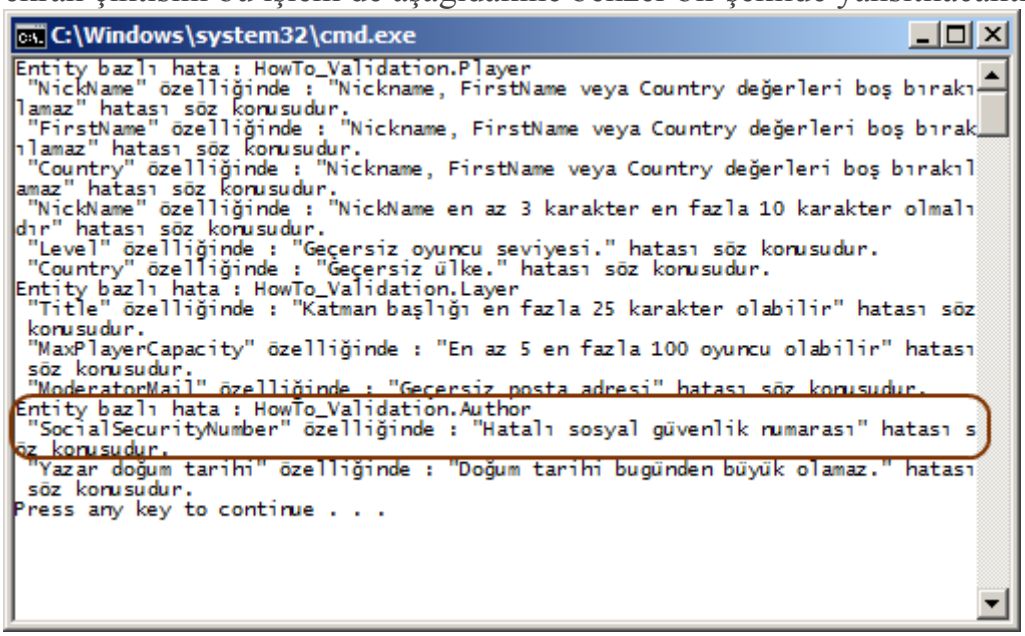


Bunu kontrol ederek duruma göre geriye **true** veya **false** değer döndürmemiz yeterlidir. Niteliği**Author Entity** tipi için aşağıdaki kod parçasında görüldüğü şekilde uygulayabiliriz.

```
using System;
namespace HowTo_Validation
{
   public class Author
   {
      public int AuthorId { get; set; }
      public string FirstName { get; set; }
      public string Scenario { get; set; }
      public DateTime Birthday{ get; set; }

      [SSNValidator(ErrorMessage="Hatalı sosyal güvenlik numarası")]
      public string SocialSecurityNumber { get; set; }
```

```
    }
}
```

Şu andaki **test** kodumuz her vaziyette **SSN** doğrulamasında false değer üretecektir. Sonuçta ekran çıktısını bu işlem de aşağıdakine benzer bir şekilde yansıtılacaktır.



Özetle **Entity Framework** tarafındaki doğrulama işlemlerini **Entity** seviyesinde**nitelikler(Attribute)** ve **IValidatableObject** arayüzü sayesinde gerçekleştirebilirken, **Context** tipi seviyesinde de **ezilebilen(overridable) ValidateEntity** metodu içerisinde yapabiliriz. Bu makalemizde çok basit seviyede de olsa, **Code-First Entity Framework** tabanlı doğrulama işlemlerini ele almaya çalıştık. Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HowTo_Validation.zip (2,59 mb)

Asp.Net Web API' leri ASPX' den Asenkron Çağırmak

Çarşamba, 14 Kasım 2012 09:00

Asp.Net Web Api, .Net Framework 4.5, Mvc, Newtonsoft, Json.Net, Task Parallel Library, Async, Await, Asynchronous Programming, Tpl

Merhaba Arkadaşlar,
Bateri çalanları her zaman için büyük bir hayranlıkla izlerim/izlemişimdir. Özellikle 4 uzuvlarını da(iki kol iki ayak) kullanırlar ama daha da önemlisi tüm bu unsurları eş zamanlı olarak çalıştırabilirler.
Sadece iki kolu çalıştırmak nispeten bir dereceye kadar kolay olabilir biz normal insan oğulları için ama bir de ayakları devreye sokmak. Hele de tüm bu hareketli parçalardan anlamlı bir melodi çıkartmak gerçekten çok ama çok zor bir iştir.
Tabi eğitimler ile belirli seviyede bateri çalabilmek pek çok insan için mümkün olabilir ama tabi ritim tutturmak, ataklarda bulunmak veya çok uzun süre boyunca aynı tempoyu hiç bozulmadan devam ettirebilmek inanılmaz bir konsantrasyon veya yetenek gerektirmektedir. Peki bu eş zamanlı çalışabilme bir yazılımcı için ne ifade edebilir? Tabi ki de asenkron çalışan kod parçalarını
Bilindiği gibi **Asenkron(Asynchronous)** çalışma, günümüz yazılım geliştirme araçlarının olmazsa olmaz parçalarından birisidir. Nitekim **kullanıcı deneyimi(User Experience)** yüksek olan uygulamalarda, istemcilerin uzun sürebilecek işlemleri beklemeden başkalarına devam edebilmesi tercih edilen ve işlem sürelerini kısaltan bir kabiliyettir.
Ne varki bu çalışma mantığı özellikle **Asp.Net** gibi sunucu tabanlı çalışan web uygulama modelleri düşünüldüğünde biraz daha farklı ele alınmalıdır/anlaşılmalıdır. Bunun en büyük sebeplerinden birisi de Web' in sunucu taraflı çalışması sırasında devreye girmekte olan sayfa yaşam döngüsüdür**(Page Life Cycle)**
Asp.Net tabanlı uygulamalar çok doğal olarak istemciden gelen taleplere cevap üretirken sunucu üzerinde yürüyen bir süreci devreye sokar. Bu süreç içerisinde talepte bulunulan sayfaların da belirli yaşam döngüleri ve dolayısıyla olay metodları devreye girer. Dolayısıyla şu senaryo her zaman için benim gibi yazılım geliştiricilerin kafasını karıştırır;
Bir web sayfası metodu içinden asenkron olarak bir servis çağrısı gerçekleştirirsem sunucu tarafındaki yaşam döngüsünde nasıl bir hareketlilik olur?
[Dikkat edileceği üzere burada konu, istemcinin AJAX tabanlı bir servis çağrısı yapmasından farklıdır. İrdelenmek istenen sunucu tarafındaki bir servis çağrısına ait asenkron çalışma mantığıdır]

Aslında senaryomuza göre istemci herhangibir şekilde web sayfasına talep de bulunduğunda, sayfanın yaşam döngüsü içerisinde kalan bazı servis çağrılarının asenkron olarak yürütülebilmesi ele alınmaktadır. İstemciler standart **HTTP Get** çağrısı dışında kendi tarayıcılarından bir düğmeye bastıklarında da sunucu tarafında bazı olayları tetiklerler. Ancak sıkıntı şudur. Sayfanın yaşam döngüsü gereği kontrollere ait **Click/Change** olay metodlarının da devreye girdiği bir an vardır. En azından o ana gelene kadar bile, bazı servislere olan çağrıların asenkron olarak başlatılıp sonuçlarının alınması düşünülebilir.

Sanırım kendi kafamı karıştırdığım kadar sizin kafanızı da epeyce karıştırmış olabilir!

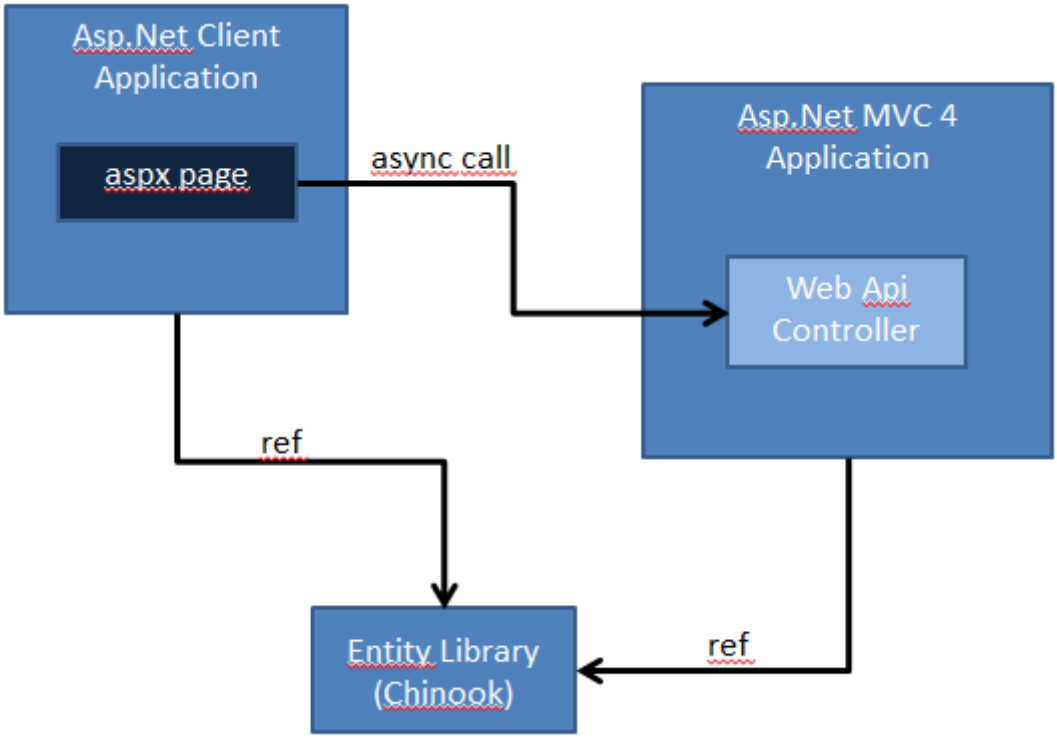
Başlangıç noktası olarak Asp.Net Server Based Web uygulamalarının yaşam döngülerine ve web dışında normal bir Console uygulaması üzerinde yapılabilen asenkron çalışma mantığına bakmanızı öneririm.

Peki biz bu yazımızda neyi değerlendiriyor olacağız? Bize ne kaldı

Malum **.Net Framework 4.5** ile birlikte, asenkron programlamayı biraz daha kolaylaştıran ve **Task Parallel Library**' nin tamamlayıcısı olarak da görebileceğimiz **async** ve **await** isimli iki yetenekli**keyword** ile tanıştık*(Yetenekli diyorum nitekim **Intermediate Language** tarafında önemli eklemeler yapıyorlar)* Dolayısıyla **Asp.Net Web Forms** uygulamalarında asenkron çalışma mantığını yeni baştan ele almamızı gerektirecek bazı kabiliyetler söz konusu.

İşte bu amaça bir **Asp.Net Web Forms** uygulamasından bir **Asp.Net Web API** servis operasyonunu asenkron olarak nasıl çağırabileceğimizi adım adım incelemeye çalışıyor olacağız. Bu sayede bir **Web**uygulamasında asenkron erişim tekniklerini nasıl ele alabileceğimizi de **.Net Framework 4.5**stilinde görmüş olacağız

İşe bir adet **Empty Asp.Net Web Application**, bir adet **Asp.Net MVC 4.0 Application***(ama Web API şablonunu kullanan)* ve bir adet te **Class Library** oluşturarak başlayalım. Web API uygulamamız içerisinde **Entity Framework** tabanlı bir **Web API** servisini de kullanıyor olacağız. Temel olarak solution içeriğimizin aşağıdaki şekilde tesis edileceğini ifade edebiliriz.



Her iki uygulamada da aynı Entity tipleri kullanılacağından basit olarak Class Library' nin her iki projeye de referans edilmesi yolunu tercih ettik.

MVC uygulamamızda örnek veritabanı olarak **Chinook'** u işaret eden bir **Entity Model'** imiz bulunmakta. Buna uygun olacak şekilde örneğin **Employee Entity** tipi ile çalışacak bir de**Controller** eklediğimizi düşünelim. **Controller** tipimizi **Empty Controller** şablonunda ekleyebiliriz. Nitekim veri çekme işlemine ait kod içeriğini kendimiz geliştiriyor olacağız.**EmployeeController** ismiyle üretilen sınıfımızın içeriğini ise aşağıdaki gibi geliştirebiliriz.

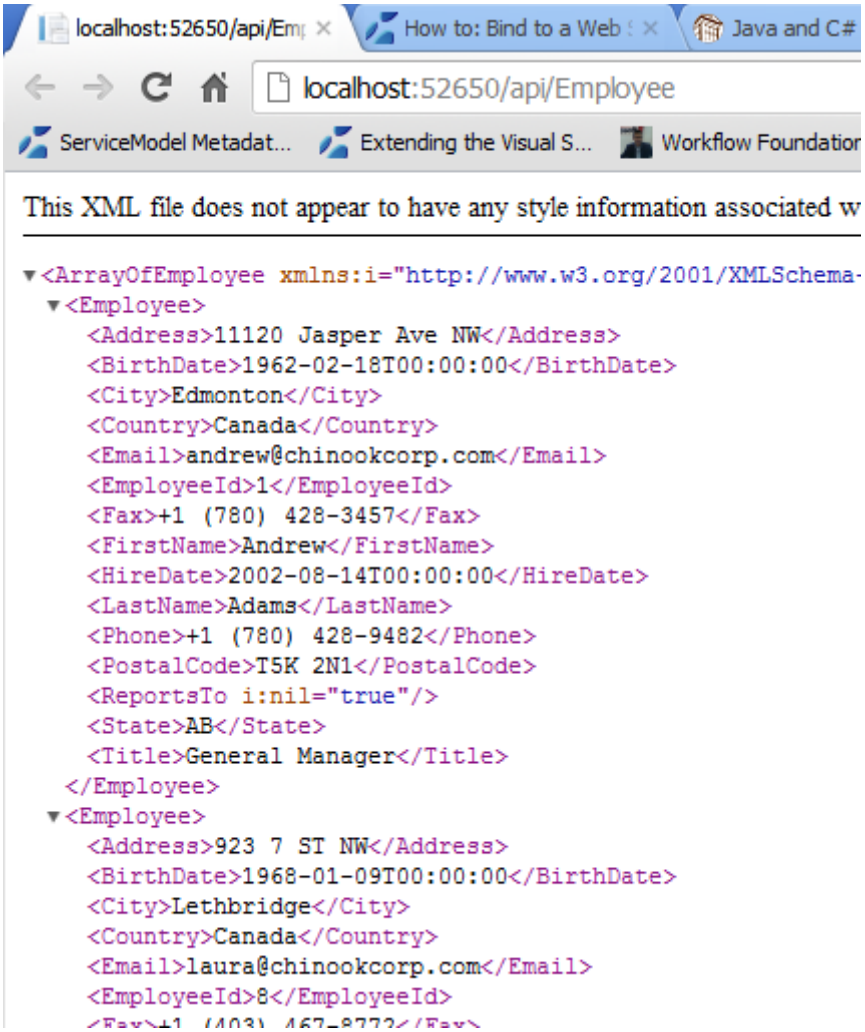
```
using ChinookEntityLibrary;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Web.Http;
namespace ChinookWebApi.Controllers
{
   public class EmployeeController
      : ApiController
   {
      private ChinookEntities db = new ChinookEntities();
     public IEnumerable<Employee> GetEmployees()
      {
         ChinookEntities db = new ChinookEntities();
         var allEmployees = from e in db.Employees
```

```
                    orderby e.LastName
                    select e;
        Thread.Sleep(5000); // Bilinçli olarak duraksatma yapıyoruz
        return allEmployees;
      }
   }
}
```

Tabi WebApi örneğinin çalışabilmesi için WebApiConfig sınıfı içerisinde yer alan Register metodunun da aşağıdaki şekilde güncellenmesi gerekmektedir.

```
using System.Web.Http;
namespace ChinookWebApi
{
   public static class WebApiConfig
   {
      public static void Register(HttpConfiguration config)
      {
         config.Routes.MapHttpRoute(
            name: "EmployeeApi",
            routeTemplate: "api/{controller}/{id}",
            defaults: new { id = RouteParameter.Optional }
         );
      }
   }
}
```

EmployeeController tipi içerisinde yer alan **Get** metodu klasik olarak **Chinook Entity Model'** i sorgulamakta ve tüm **Employee** listesini **LastName** alanına göre sıralayarak geriye döndürmektedir. Metodun örneğimiz için önemli olan kısmı ise çalışmakta olan **Thread'** in **5 saniye boyuncaduraksatıldığı** kısımdır. Bu, verinin en az 5 saniye geç gelmesine neden olacak ve asenkron için gerekli senaryoya zemin hazırlayacaktır.
Bu arada örneği test etmemizde ve servis çağrısı sonucu geçerli bir veri içeriğini elde edebildiğimizi görmemiz de yarar var. Eğer sıkıntı yoksa aşağıdakine benzer bir ekran görüntüsü almamız gerekecektir.



Şimdi elimizde **HTTP** tabanlı çalışan bir **REST** servisi bulunmakta. Bu servisi konuşlandırdığımız adresi kullanarak, diğer **Web** uygulamasındaki **Web Form** üzerinden bir **Request** gönderiyor olacağız. Ancak bu **Request'** in asenkron şekilde gerçekleştirilmesi de önemli. Bunu sağlamak için öncelikli olarak istemci **Web Form'** u içerisine aşağıdaki metodları yazdığımızı göz önüne alalım.

```
using ChinookEntityLibrary;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Diagnostics;
using System.Net.Http;
using System.Threading.Tasks;
using System.Web.UI;
namespace ClientApp
{
   public partial class AsyncTestPage : System.Web.UI.Page
   {
```

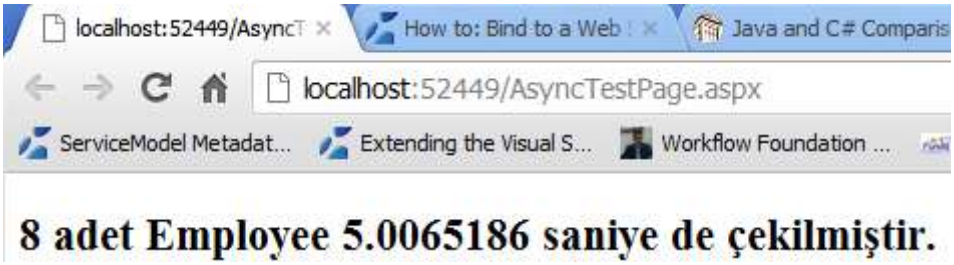
```
    protected void Page_Load(object sender, EventArgs e)
    {
        RegisterAsyncTask(new
PageAsyncTask(GetEmployeeDataFromServiceAsync));
    }
    public async Task<List<Employee>> InvokeEmployeeService()
    {
        using (HttpClient client = new HttpClient())
        {
            HttpResponseMessage serviceCallResponse = await
client.GetAsync(ConfigurationManager.AppSettings["EmployeeServiceAddress"]);
            string jsonContent = (await
serviceCallResponse.Content.ReadAsStringAsync());
            List<Employee> employees =
JsonConvert.DeserializeObject<List<Employee>>(jsonContent);
            return employees;
        }
    }
    private async Task GetEmployeeDataFromServiceAsync()
    {
        Stopwatch stopWatch = new Stopwatch();
        stopWatch.Start();
        var taskInvokeEmployee = InvokeEmployeeService();
        await taskInvokeEmployee;
        List<Employee> data1 = taskInvokeEmployee.Result;
        stopWatch.Stop();
        lblResult.Text = string.Format("<h2>{0} adet Employee {1} saniye de
çekilmiştir.</h2>"
            ,data1.Count, stopWatch.Elapsed.TotalSeconds);
    }
  }
}
```

Client uyulamasında HttpClient tipinin kullanımı için System.Net assembly' ının referans edilmesi gerekir. Ayrıca JSON içeriğinin daha kolay bir şekilde ele alınabilmesi için Newtonsoft' un Json.Net kütüphanesinden yararlanılmaktadır. Bu kütüphane NuGet Package Manager ile uygulamaya kolayca yüklenebilir.
InvokeEmployeeService metodu dikkat edileceği üzere **async** anahtar kelimesi(keyword) ile işaretlenmiştir. Bunun en önemli nedenlerinden birisi de, içerisinde **awatiable** iki fonksiyon içermesidir. Bu fonksiyonlardan birisi servis operasyonunu asenkron olarak çağıran **GetAsync'** dir. Diğeri ise servis operasyonuna

ait **response'** daki içeriği **Json** formatında çeken**ReadAsStringAsync'** dir. Her iki asenkron metodun çağrısı sırasında **await keyword'** ünün kullanıldığına dikkat edilmelidir. **InvokeEmployeeService** metodu aslında **Employee** tipinden olan **generic List** koleksiyonunu sarmallayan bir **Task** nesne örneğini döndürmektedir. **await** ile işaretlenmiş olan metodlar aslında bir sonraki satıra geçilmesi noktasında beklenilmesi gerektiğini belirtmektedir. Dolayısıyla**GetAsync** metodu ile **Response** alınmadan sonraki ifadeye geçilmemesi gerektiği **await** ile bildirilmektedir.
Gelelim **GetEmployeeDataFromServiceAsync** metoduna. Bu metod içerisinde aslında **Stopwatch**nesne örneği ile yapılan süre ölçümü haricinde, **InvokeEmployeeService** metoduna yapılan bir çağrı da söz konusudur. Bu çağrı yapılırken dikkat edileceği üzere, **taskInvokeEmployee** isimli**Task** tipinden nesne örneği için **await** keyword' ünün kullanıldığı görülmektedir.
Dolayısıyla**taskInvokeEmployee** nesne örneğinin sarmalladığı(Wrap) generic **Employee** listesi dönmeden, sonraki ifadeye geçilmemesi gerektiği ifade edilmektedir.
async keyword' ü ile işaretlediğimiz **GetEmployeeDataFromServiceAsync** metodunun **PageFramework'** e register edilmesi için **Page_Load** olay metodu içerisinde **Page** özelliği ile erişilebilen**RegisterAsyncTask** metodunun kullanıldığını görüyoruz. Bu metoda yapılan çağrı ile, parametre olarak gelen **async** şeklinde işaretlenmiş asenkron çağırılabilen metodun sayfa ile ilişkilendirilmesi sağlanmış olunmaktadır.
Ancak işlemlerimiz bunlarla sınırlı değil Son olarak sayfanın **Page** direktifi içerisindeki **Async**niteliğine **true** değerinin atanması gerekmektedir. Böylece web sayfanın asenkron olarak çalıştırılacağı belirtilmektedir.

```
<%@ Page Async="true" Language="C#" AutoEventWireup="true"
CodeBehind="AsyncTestPage.aspx.cs" Inherits="ClientApp.AsyncTestPage" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:Label ID="lblResult" runat="server" />

   </div>
   </form>
</body>
</html>
```

Artık Web sayfamızı çalıştırabiliriz. İlk çağrı sırasında servisin ayağa kalkması veya sayfanın ilk kez yürütülmesinden kaynaklanan bir gecikme sorunu yaşanabilir ve beklemediğimiz kadar uzun bir süre ile karşılaşabiliriz. Ancak sonraki çağrılarda aşağıdakine benzer bir çıktı alırız. Yaklaşık olarak 5 saniye civarında bir çalışma zamanı söz konusudur ki bu son derece normaldir.

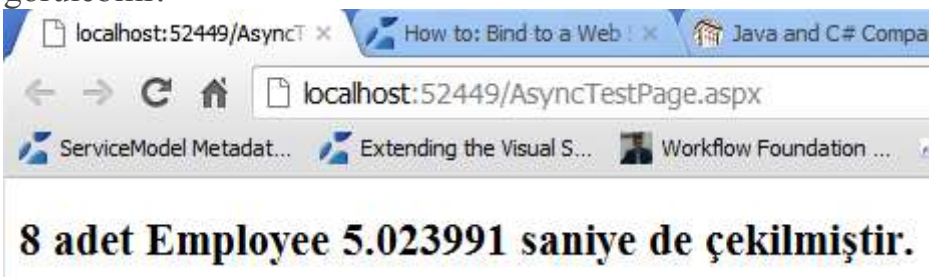


Yine de bu sayfanın tam anlamıyla asenkron çalıştığına dair bir kanıt değildir*(Unutmayın ki sayfanın sunucu tarafındaki yaşam döngüsü içerisinde bir asenkron çalışma senaryosu göz önüne alınmaktadır)* Bu kanıt için kodu biraz daha ilginçleştirelim ve sayfa içeriğini aşağıdaki hale getirelim.

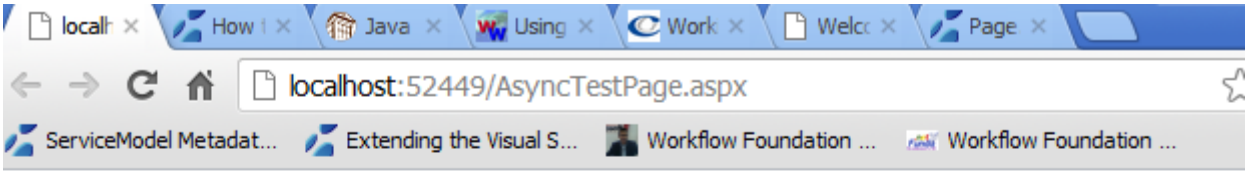
```
using ChinookEntityLibrary;
using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Diagnostics;
using System.Net.Http;
using System.Threading;
using System.Threading.Tasks;
using System.Web.UI;
namespace ClientApp
{
   public partial class AsyncTestPage : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         RegisterAsyncTask(new PageAsyncTask(GetEmployeeDataFromServiceAsync));
         DoSomething();
      }
      public async Task<List<Employee>> InvokeEmployeeService()
      {
            .
            .
            .
      }
```

```
        private async Task GetEmployeeDataFromServiceAsync()
        {
                .
                .
                .
        }
        private void DoSomething()
        {
           Thread.Sleep(5000);
        }
    }
}
```

Dikkat edileceği üzere **RegisterAsyncTask** metoduna yapılan çağrının hemen ardından içerisinde sayfaya ait **Thread'** i **5** saniye kadar geciktiren bir fonksiyon çağrısı daha yapılmıştır. Buna göre çalışma zamanında aşağıdakine benzer bir sonucun alındığı görülebilir.



Normal şartlarda sayfanın yaşam döngüsünü düşündüğümüzde, senkron yapılan bir işleyiş de servis tarafındaki 5 saniyelik gecikme ve içerideki DoSomething üzerinden gelen 5 saniyelik gecikme sonrası en az 10 saniyelik bir işlem süresi olması gerekmektedir. Aslında böyledir de  Eğer sayfanın **Trace** modunu açarsak aşağıdaki **Trace Information** sonuçları ile karşılaşırız.



8 adet Employee 5.1760688 saniye de çekilmiştir.

Request Details

Session Id:	gefpir1nsfjekvz50nftmaa3	**Request Type:**	GET
Time of Request:	03.09.2012 14:36:18	**Status Code:**	200
Request Encoding:	Unicode (UTF-8)	**Response Encoding:**	Unicode (UTF-8)

Trace Information

Category	Message	From First(s)	From Last(s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	0.020060	0.020060
aspx.page	Begin Init	0.020204	0.000144
aspx.page	End Init	0.020275	0.000071
aspx.page	Begin InitComplete	0.020308	0.000033
aspx.page	End InitComplete	0.020343	0.000035
aspx.page	Begin PreLoad	0.020373	0.000030
aspx.page	End PreLoad	0.020404	0.000031
aspx.page	Begin Load	0.020434	0.000029
aspx.page	End Load	6.437241	6.416807
aspx.page	Begin LoadComplete	6.437312	0.000071
aspx.page	End LoadComplete	6.437350	0.000038
aspx.page	Begin PreRender	6.437383	0.000033
aspx.page	End PreRender	6.437469	0.000086
aspx.page	Begin PreRenderComplete	11.616486	5.179016
aspx.page	End PreRenderComplete	11.616543	0.000058
aspx.page	Begin SaveState	11.617083	0.000540
aspx.page	End SaveState	11.625344	0.008261
aspx.page	Begin SaveStateComplete	11.625369	0.000025
aspx.page	End SaveStateComplete	11.625386	0.000017
aspx.page	Begin Render	11.625401	0.000016
aspx.page	End Render	11.625755	0.000354

Görüldüğü üzere sayfanın render edilme süresi yine **11** saniyeler civarındadır. Yaklaşık 5 saniyelik servis çağrı süresi + 5 saniyelik DoSomething süresi. Peki biz neyi başarmış olduk? 

Başarılan, servis çağrısının sayfanın yaşam döngüsü içerisinde asenkron olarak gerçekleştirilebilmesidir 

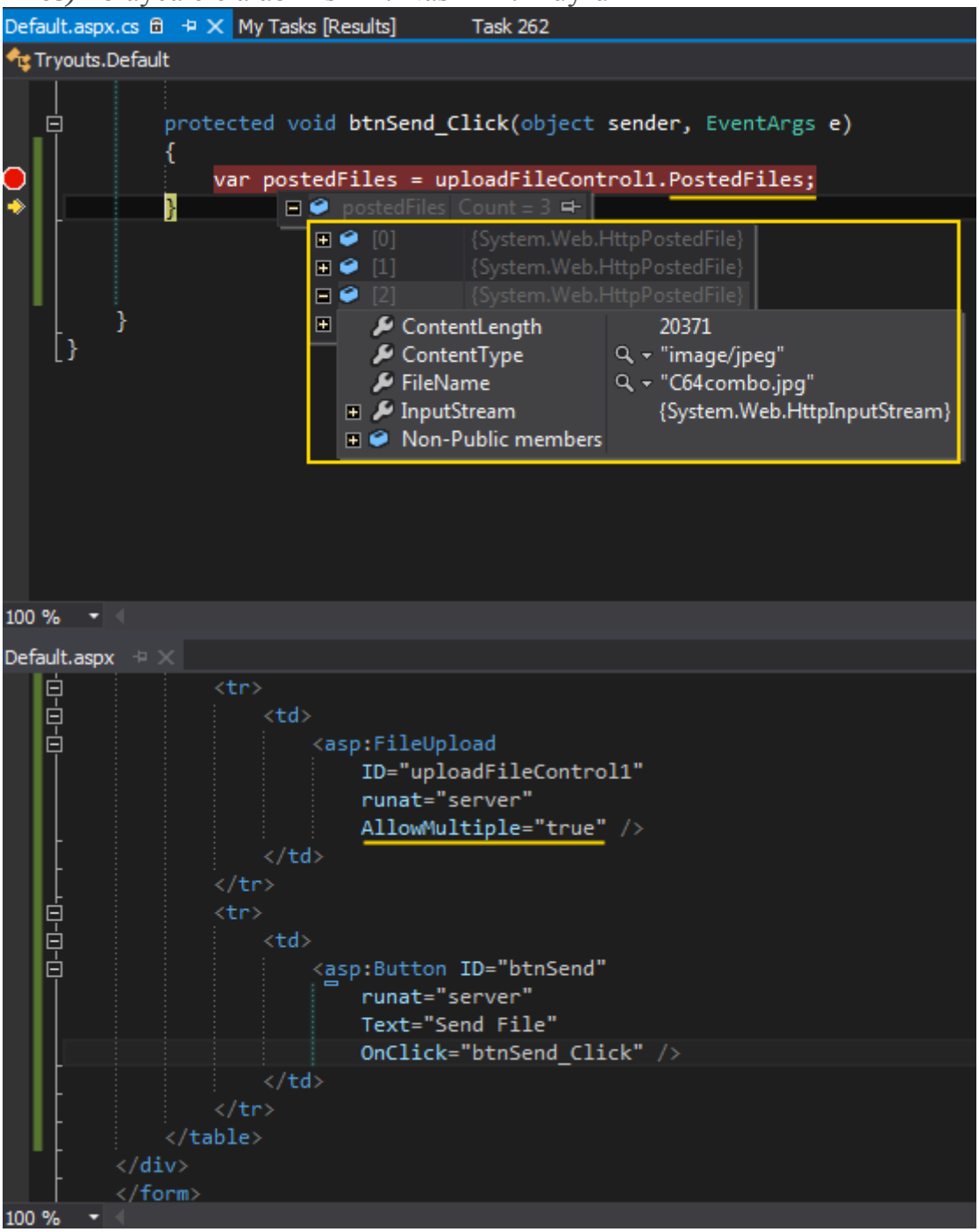
Böylece geldik bir yazımızın daha sonuna. Bu makalemizde bir **ASP.Net Web API** servisinin, bir Web uygulaması içerisinden asenkron olarak nasıl çağırılabileceğini, **.Net Framework 4.5** ile birlikte gelen **async** ve **await keyword'** lerini de işin içerisine katarak değerlendirmeye çalıştık. Benim için de oldukça yeni ve halen daha öğrenmeye çalıştığımı bir konu. Özellikle **MVC(Model View Controller)** tabanlı **Asp.Net** uygulamalarında bu asenkron çağırımları nasıl değerlendirebileceğimizi de incelemeye çalışıyorum. Elde ettiğim bulguları ve öğrendiklerimi her zaman ki gibi bloğumda sizlerle paylaşıyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Tek Fotoluk İpucu 70.5–Asp.Net Multiple File Upload

Salı, 13 Kasım 2012 17:57

Tek Fotoluk Ipucu, Asp.Net 4.5, Fileupload, Allow Multiple, Multiple File Upload,Postedfiles, Allowmultiple, Asp.Net

Merhaba Arkadaşlar,
Asp.Net 4.5 ile **FileUpload** kontrolüne gelen iki önemli özellik*(Property)* mevcuttur. Bunlardan birisi **AllowMultiple**, diğeri ise **PostedFiles**' dır. Bu iki özelliği kullanarak birden fazla dosyanın, istemciden sunucu tarafına yüklenme işlemlerini*(Multiple Upload Files)* kolayca ele alabilirsiniz. Nasıl mı? Buyrun



İşin özünde **AllowMultiple** özelliğine atanan **true** değeri*(ki varsayılan olarak false dur)* yatmaktadır.Bu sayede, istemci tarafında açılan pencerede birden fazla dosyasnın seçilmesine izin verilmektedir. Dolayısıyla **PostedFiles** özelliği **HttpPostedFile** tipinden birden fazla örnek ile dolar ve sunucu tarafında istenilen şekilde değerlendirilebilir.
Bir başka ipucunda görüşmek dileğiyle 

Tek Fotoluk İpucu–70–Yine Newtonsoft Json.net ve dynamic

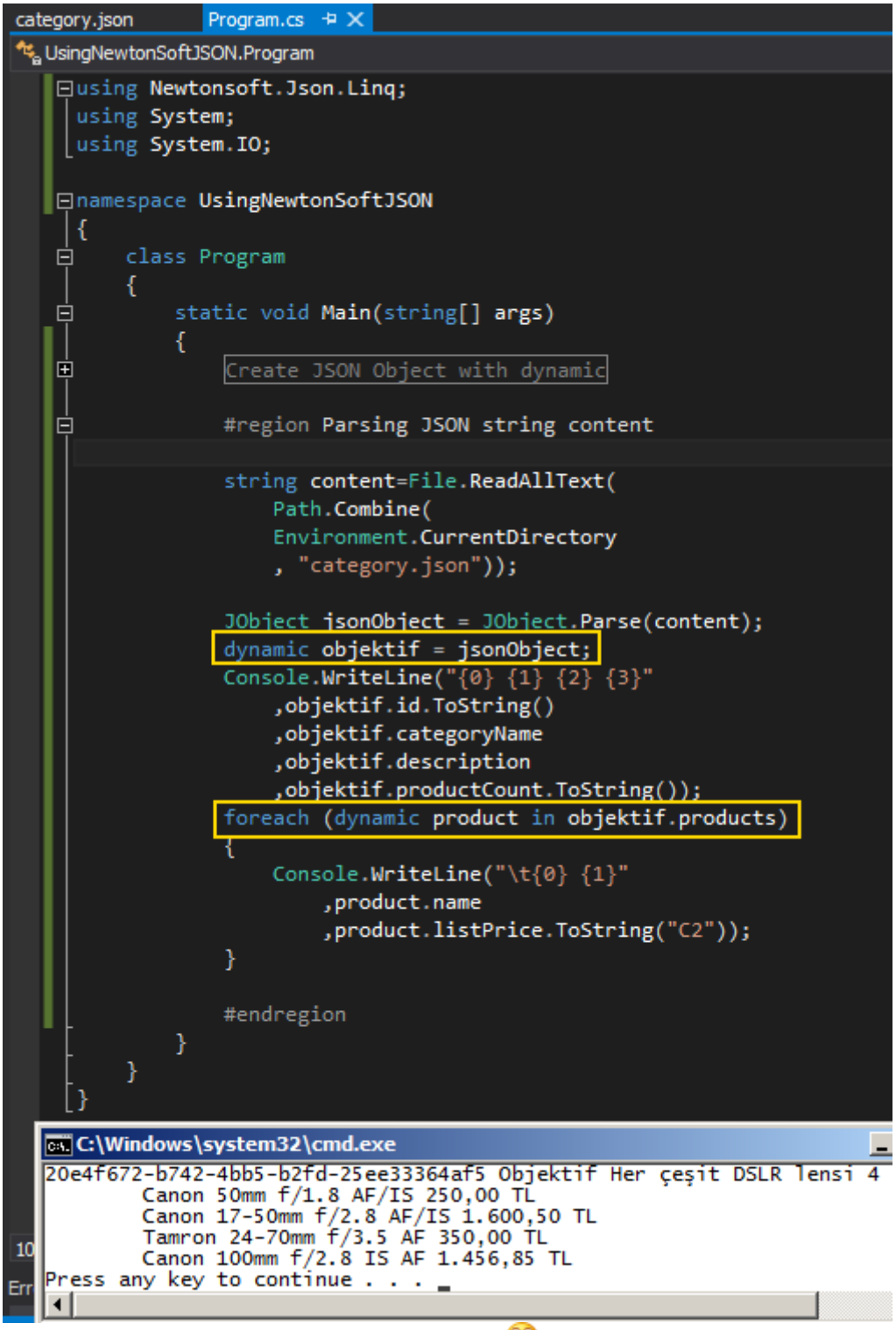
Pazartesi, 12 Kasım 2012 15:30

Tek Fotoluk Ipucu, Dynamic, C#, Json, Serialization, Json.Net, Newtonsoft, Jobject,Jarray

Merhaba arkadaşlar,
Diyelim ki elimizde aşağıdaki gibi bir JSON içeriği var.

```
{
  "categoryName": "Objektif",
  "description": "Her çeşit DSLR lensi",
  "productCount": 4,
  "id": "bb913579-ac93-4398-a77d-dd07db825df8",
  "products": [
    {
      "name": "Canon 50mm f/1.8 AF/IS",
      "listPrice": 250.0
    },
    {
      "name": "Canon 17-50mm f/2.8 AF/IS",
      "listPrice": 1600.5
    },
    {
      "name": "Tamron 24-70mm f/3.5 AF",
      "listPrice": 350.0
    },
    {
      "name": "Canon 100mm f/2.8 IS AF",
      "listPrice": 1456.85
    }
  ]
}
```

ve hatta NuGet ile eklediğimiz **Newtonsoft**' un **JSON.net** kütüphanesi.
Ha bir de **dynamic** keyword' ümüz var.
O halde bu doküman içeriğini okumamız ne kadar zor olabilir ki? 😉



Başka bir ipucunda görüşmek dileğiyle

Tek Fotoluk İpucu–69–Newtonsoft JSON.Net ve dynamic Keyword

Pazartesi, 5 Kasım 2012 09:00

Tek Fotoluk Ipucu, Json, Json.Net, Newtonsoft, Dynamic, C#, Mvc, Serialization

Bildiğiniz üzere JSON(JavaScriptObjectNotation) oldukça kompakt bir veri formatı sunuyor. Çoğu durumda veriyi anlamlı şekilde saklarken, XML serileştirme yerine tercih ediyoruz. Nitekim daha az yer kaplamakla birlikte nesnel olarak anlaşılabilirliği daha yüksek. Özellikle MVC tarafında çok kıymetli. JSON ile .Net tarafında çalışırken ise işleri kolaylaştırmak adına Newtonsoft' un NuGet ile indirebileceğimiz paketini kullanmaktayız.
install-package newtonsoft.json
Şimdi düşünün ki elinizde C#' ın dynamic gücü ve Newtonsoft' un JSON.net kütüphanesi var. Sizce bir JSON nesnesini oluşturmak ne kadar zor olabilir

Program.cs

UsingNewtonSoftJSON.Program

```
using Newtonsoft.Json.Linq;
using System;

namespace UsingNewtonSoftJSON
{
    class Program
    {
        static void Main(string[] args)
        {
            dynamic category = new JObject();
            category.categoryName = "Objektif";
            category.id = Guid.NewGuid();
            category.products = new JArray() as dynamic;

            dynamic product1 = new JObject();
            product1.name = "Canon 50mm f/1.8 IS";
            product1.listPrice = 250M;
            category.products.Add(product1);

            dynamic product2 = new JObject();
            product2.name = "Canon 17-50mm f/2.8 IS";
            product2.listPrice = 1600.50M;
            category.products.Add(product2);

            Console.WriteLine(category.ToString());
        }
    }
}
```

C:\Windows\system32\cmd.exe

```
{
  "categoryName": "Objektif",
  "id": "0853ec86-e37b-4810-83d1-28c8aaaaffdb",
  "products": [
    {
      "name": "Canon 50mm f/1.8 IS",
      "listPrice": 250.0
    },
    {
      "name": "Canon 17-50mm f/2.8 IS",
      "listPrice": 1600.5
    }
  ]
}
Press any key to continue . . .
```

İşte bu kadar  Başka bir ipucunda görüşmek dileğiyle.

Entity Framework 6 Alpha 1 ve async, await Özellikleri

Perşembe, 1 Kasım 2012 09:41

Entity Framework, Entity Framework 6.0, Alpha, Async, Await, Task Parallel Library, Asynchronous Programming, Nuget

Merhaba Arkadaşlar,
Doğruyu söylemek gerekirse yazılım hayatım boyunca en çok kurduğum cümlelerden birisi de sanıyorum ki şu olmuştur :**"Microsoft' un hızına yetişemiyoruz"** Bazı açılardan bakıldığında bu özellikle nihayi ürün ile geliştirme yapanlar için bir handikap olarak görülebilir. Çünkü yeni sürümler genellikle geliştiricilerin ve ürün yöneticilerinin arzu ettikleri, görmek istedikleri kabiliyetleri içermektedir. Diğer taraftan kişisel görüşüme göre, **Microsoft** yazılım ekiplerinin bu çalışkanlığı da takdir edilmesi gereken bir durumdur. Bu ekiplerin başında da **EntityFramework** geliştirilmesinden sorumlu ekip gelmektedir.
Sözü fazla uzatmıyayım ama daha bu ayki **Entity Framework 5.0** tabanlı **Nedirtv?com** Webinerime hazırlanırken bir kaç gün önce EF 6.0' nın alpha sürümünün yayınlandığını ve **NuGet** paket yönetim aracı ile indirilebileceğini öğrendim. **Paketi arayıp bulabilmek için, Manage Nuget Packages dialog penceresindeki Include Prerelease seçeneğini işaretlemeyi unutmayın. Aksi durumda Release sürümleri öncesindeki ürünler listelenmeyeceklerdir.**

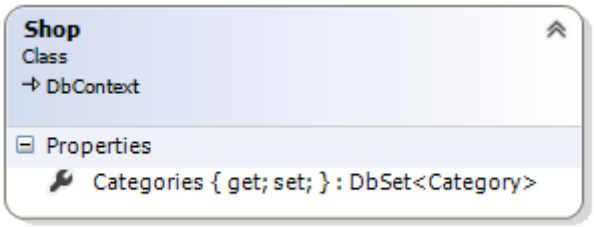
Entity Framework geliştirilmesinden sorumlu takım bildiğiniz gibi codeplex üzerinden kaynak kodları da açmış durumdadır. Dolayısıyla **alpha** sürümüne ait kodları açık kaynak olarak inceleyebilir ve hatta **Microsoft'** un beklediği gibi, ürünle ilişkili geri bildirimlerinizi*(Feedbacks)* ekibe iletebilirsiniz. Bizi dinliyorlar ve gerçekten bazı gerekli

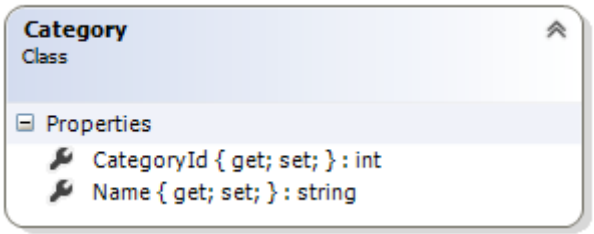
özellikleri yeni sürümlere dahil ediyor veya en azından yol haritasına*(Roadmap)* alıyorlar. *(Bu arada Ef tarafındaki Roadmap ile ilişkili olarak bu adresi takip edebilirsiniz)*

6.0 versiyonu ile birlikte **Entity Framework'** ün son sürümüne dahil edilmesi planlanan bir çok özellik de bulunmakta. Bunlardan birisi de artık **.Net Framework'** ün olmassa olmaz parçası haline gelen paralel programlama desteği ve pek tabi **Task** tabanlı kabiliyetleri dil seviyesinde kolaylaştıran**async**, **await** anahtar kelimelerinin kullanılabilmesi. Bu destek kendisini diğer alt yapılarda da göstermekte. **Entity Framework** tarafında da kayıt ve sorgulama operasyonları için asenkron çalışma desteği getirilmiş durumda*(alpha sürümü için)*. İşte bu yazımızda **async, await** kullanımını incelemeye çalışıyor olacağız.

Başlangıç

İlk etapta **Visual Studio 2012** üzerinde basit bir **Console** uygulaması oluşturup gerekli **EntityFramework** kütüphanesini **Nuget** yardımıyla dahil ederek işe başlayabiliriz. Konuyu basit ve kolay bir şekilde anlayabilmek adına **Code-First** yaklaşımını tercih ediyor olacağız. Başlangıçta ki tip hiyerarşisini aşağıki gibi oluşturabiliriz.






```
class Category
{
   public int CategoryId { get; set; }
   public string Name { get; set; }
}
class Shop
   : DbContext
{
   public DbSet<Category> Categories { get; set; }
}
```

Klasik olarak kobay tiplerimizden kategori sınıfını kullandığımız bir yapı söz konusudur Kategoriler ve isterseniz bunlara bağlı ürünleri de dahil edebileceğini veri modelini **Shop** isimli**DbContext** türevi içerisinden sunmaktayız.

Eğer kendi belirleyeceğiniz SQL bağlantısı üzerinde Shop context tipi için gerekli veritabanını üretmeyi planlıyorsanız, app.config dosyasında aşağıdakine benzer bir bildirimde bulunmanız gerekecektir.

```
<connectionStrings>
  <add name="Shop" connectionString="data
source=localhost\sqlinstancename;integrated security=SSPI"
providerName="System.Data.SqlClient"/>
</connectionStrings>
```

Klasik Veri Ekleme, Sorgulama

Şimdi bilinen yöntemi ile örnek kategoriler ilave edip, sonrasında sorgulamak istediğimizi düşünelim. Aşağıdakine benzer bir kod içeriği pekala işimizi görecektir.

```
class Program
{
   static void Main(string[] args)
   {
      InsertCategory("Kitap");
      InsertCategory("Kalem");
      InsertCategory("Defter");
      InsertCategory("Oyuncak");
      WriteCategories();
   }
   static void InsertCategory(string categoryName)
   {
      using (Shop context = new Shop())
      {
         Category newCategory = new Category { Name = categoryName };
         context.Categories.Add(newCategory);
         context.SaveChanges();
      }
   }
   static void WriteCategories()
   {
      using (Shop context = new Shop())
      {
         var categories = from c in context.Categories
                          orderby c.Name
                          select c;
         foreach (var category in categories)
         {
            Console.WriteLine("{0} {1}",category.CategoryId,category.Name);
         }
```

```
        }
    }
}
```

Örneği basit seviyede ele almak adına sadece kategori tipinden örneklerin eklenmesi ve sorgulanması simüle edilmiştir.

async ve await ile Beslemek

Yeni eklenen **Entity** nesne örneklerini veritabanına gönderirken ve sorgularken kullanabileceğimiz asenkron metodlar **SaveChangesAsync** ve **ForEachAsync** isimli fonksiyonlardır. *(Bir başka deyişle**Async** son eki ile biten operasyonları kullanmamız gerekmektedir)* Bu operasyonlar **await** edilebilir niteliktedirler. Şimdi dilerseniz örneğimizde yer alan **insert** ve **select** operasyonların asenkron modda çalışabilir hale getirelim. Bunun için aşağıdaki kod parçasını değerlendirebiliriz.

```
class Program
{
    static void Main(string[] args)
    {
        InsertAndSelectAsync().Wait();
    }
    static async Task InsertAndSelectAsync()
    {
        await InsertCategory("Kitap");
        await InsertCategory("Kalem");
        await InsertCategory("Defter");
        await InsertCategory("Oyuncak");
        await WriteCategories();
    }
    static async Task InsertCategory(string categoryName)
    {
        using (Shop context = new Shop())
        {
            Category newCategory = new Category
            {
                Name = categoryName
            };
            context.Categories.Add(newCategory);
            await context.SaveChangesAsync();
        }
    }
    static async Task WriteCategories()
    {
        using (Shop context = new Shop())
```

```
    {
        await context
            .Categories
            .ForEachAsync(c=>
        {
            Console.WriteLine("{0} {1}",c.CategoryId,c.Name);
        });
    }
  }
}
```

Dikkat edileceği üzere **InsertCategory** metodu **async** anahtar kelimesi ile imzalanmış ve içerisinde yapılan **SaveChangesAsync** fonksiyon çağırımında **await** kullanılmıştır. Benzer durum**WriteCategories** metodu için de söz konusudur. Bu metodda sorgu işlemini asenkron modda gerçekleştirmek için, **ForEachAsync** metoduna yapılan çağrıda da **await** kullanılmıştır.**WriteCategories** fonksiyonu da **InsertCategory** gibi **async** çalışacak şekilde işaretlenmiştir.
InsertCategory ve **WriteCategories** fonksiyonları **async** desenine uygun şekilde geliştirildiklerinden **await** edilebilirler. Bu sebepten tüm işlemleri kapsülleyen**InsertAndSelectAsync** metodu içerisinde **await** kullanımlarına yer verilerek ilerlenilmiştir. *(Bu metodu yazmak mecburi değildir)* Main metodu içerisinde asenkron çalışan metodların işlmeleri tamamlanıncaya kadar uygulamanın beklemesini sağlamak içinse **Wait** fonksiyonundan yararlanıldığında dikkat edilmelidir.
Pek tabi örnekteki operasyonlar çok basit ve aslında hızlı olduklarından asenkron bir modelin kullanılmasının performansa önemli bir katkısı bulunmamaktadır.
Hatta **Thread** seviyesinde yapılan hazırlıklar nedeni ile negatif bir etki de söz konusu olabilir.
Yine de bazı senaryolarda ve özellikle sunucu tarafında çok fazla sayıda **CRUD(Create Retrieve Update Delete)** işleminin gönderilebileceği vakalarda bu tip bir yaklaşım ele alınabilir. İstenirse söz konusu asenkron işleyiş için **Task** tiplerinden aşağıdaki kod parçasında olduğu gibi yararlanılabilir de. Bir başka deyişle InsertAndSelectAsync fonksiyonu bir zorunluluk değildir.

```
static void Main(string[] args)
{
   Task[] tasks = new Task[5];
   tasks[0] = InsertCategory("Kitap");
   tasks[1] = InsertCategory("Kalem");
   tasks[2] = InsertCategory("Defter");
   tasks[3] = InsertCategory("Oyuncak");
   tasks[4] = WriteCategories();
   Task.WaitAll(tasks);
}
```

async olarak işaretlenmiş **InsertCategory** ve **WriteCategories** metodları **Task** dönüş tipine sahip olduklarından**,** Task türevli bir dizinin elemanı olarak kullanılabilirler. Buna göre senaryomuzda yer alan asenkron fonksiyon çağrılarına ait bir **Task** dizisinin, **WaitAll** tekniğine göre ele alınması ve uygulama sonlanmadan önce bu işlemleri içeren tüm Task örneklerinin işlerinin bitirilmesinin beklenmesi sağlanabilir. Görüldüğü üzere **Entity Framework** tarafında asenkron programlama desteği de artık adım adım gelmektedir. Alpha sürümüne ait yapmış olduğumu bu örnek uygulamanın beta ve release sürümlerinde çok fazla değişikliğe uğramayacağını ama async takılı ek metodların da gelebileceğini düşünmekteyim. Entity Framework' ün yeni özelliklerine dair bilgileri ilerleyen zamanlarda paylaşmaya devam ediyor olacağım. Örneğin bu yeniliklerden birisi Code-First tarafında Stored Procedure ve Function tanımlanıp kullanılabilmesidir. Tekrardan görüşmek dileğiyle hepinize mutlu günler dilerim

EF6_Alpha_AsyncAwait.zip (2,25 mb)

[Örnekte Entity Framework 6.0 Alpha 1 sürümü kullanılmıştır zip boyutunu küçültmek için içerideki nuget bazlı package klasörü çıkartılmıştır.]

Tek Fotoluk İpucu–68–Reflection ile Workflow Activity Yüklemek, Çalıştırmak

Cuma, 12 Ekim 2012 10:20

Tek Fotoluk Ipucu, Workflow Foundation, Activity, Reflection, Linq, Let Keyword

Merhaba Arkadaşlar,
Diyelim ki elinizde içerisinde bi dünya **Workflow Activity**' si olan bir kütüphane var. Ancak bu kütüphane projenize referans edilmiş değil. Fiziki bir klasörde tutulmakta. Siz de istiyorsunuz ki, bu kütüphane içerisinde yer alan herhangibir **Workflow Activity**' sini örnekleyebileyim ve hatta**Workflow** çalışma zamanı motoruna devredip yürütebileyim. Aşağı yukarı yapmanız gereken şeyin içerisinde **Reflection** olduğunu tahmin ediyorsunuzdur. Belki de aşağıdaki gibi bir yaklaşım hayal ediyorsunuzdur

Program.cs

Program.cs

HowToTransaction.Program.Utility

GetActivities(string libraryAddress)

```
using System;
using System.Activities;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Reflection;

namespace HowToTransaction{
    class Program{
        static void Main(string[] args){
            var activityList = Utility.GetActivities(
                Path.Combine(Environment.CurrentDirectory, "CommonActivityLibrary.dll"));
            var activity = (from a in activityList
                            where a.DisplayName == "SumActivity"
                            select a.Activity).FirstOrDefault();

            var result = WorkflowInvoker.Invoke(activity
                , new Dictionary<string, object> { { "Param1", 1 }, { "Param2", 3 } });
            Console.WriteLine(result["Result"].ToString());
        }

        public static class Utility{
            public static List<ActivityContainer> GetActivities(string libraryAddress){
                try{
                    var assembly = Assembly.LoadFile(libraryAddress);
                    return (from type in assembly.GetTypes()
                            where type.IsClass && type.BaseType == typeof(Activity)
                            let activityInstance = (Activity)Activator.CreateInstance(type)
                            orderby type.Name
                            select new ActivityContainer
                            {
                                DisplayName = activityInstance.DisplayName,
                                Activity = activityInstance
                            }
                            ).ToList();
                }
                catch (Exception excp){
                    throw excp;
                }
            }
        }
        public class ActivityContainer{
            public string DisplayName { get; set; }
            public Activity Activity { get; set; }
        }
    }
}
```

C:\Windows\system32\cmd.exe

```
4
Press any key to continue . . .
```

SumActivity int *tipinde, dışarıdan gelen iki argümanı toplayıp, sonucu yine bir argüman ile geriye döndüren akışı içermektedir.*

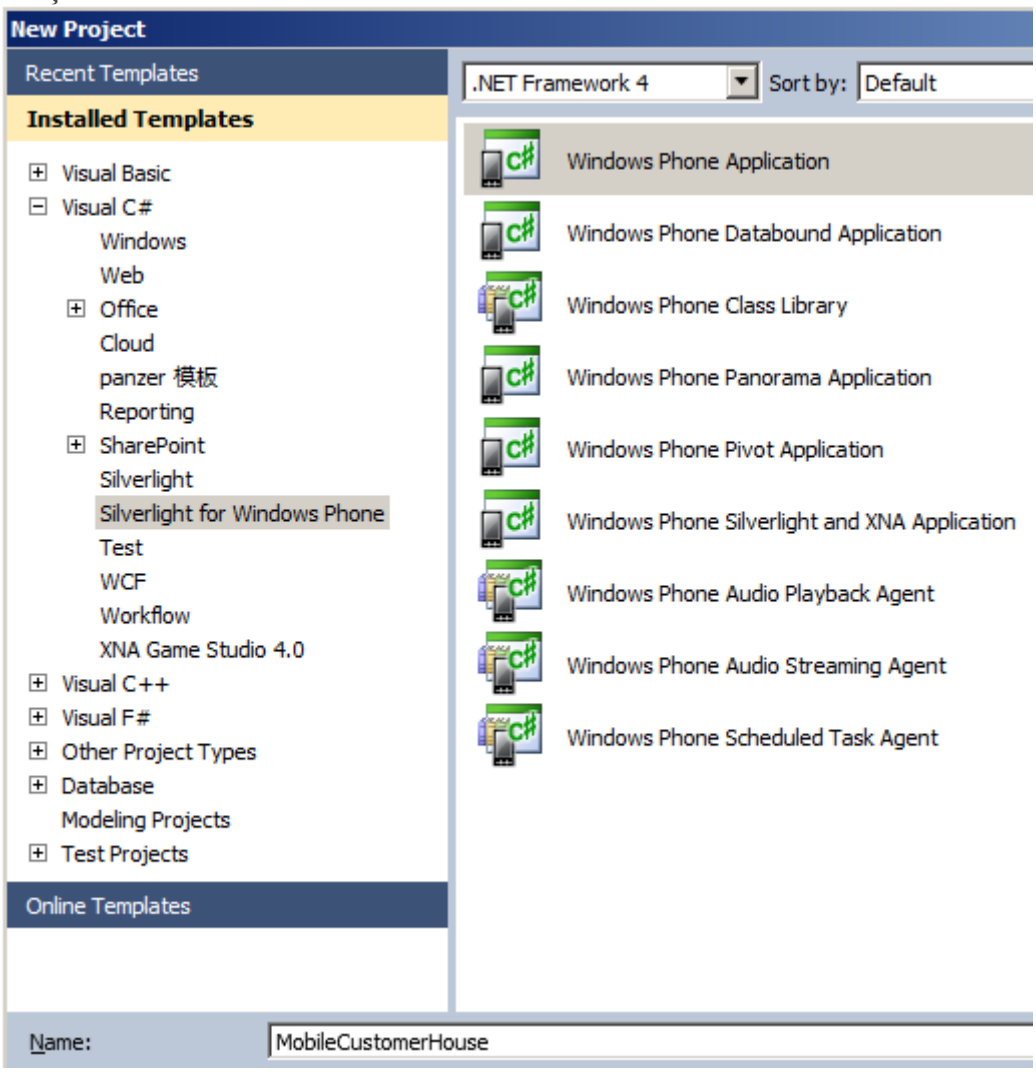
Windows Phone 7 Cihazlarda LINQ to SQL Kökenli Veritabanı ile Çalışmak

Perşembe, 11 Ekim 2012 12:00

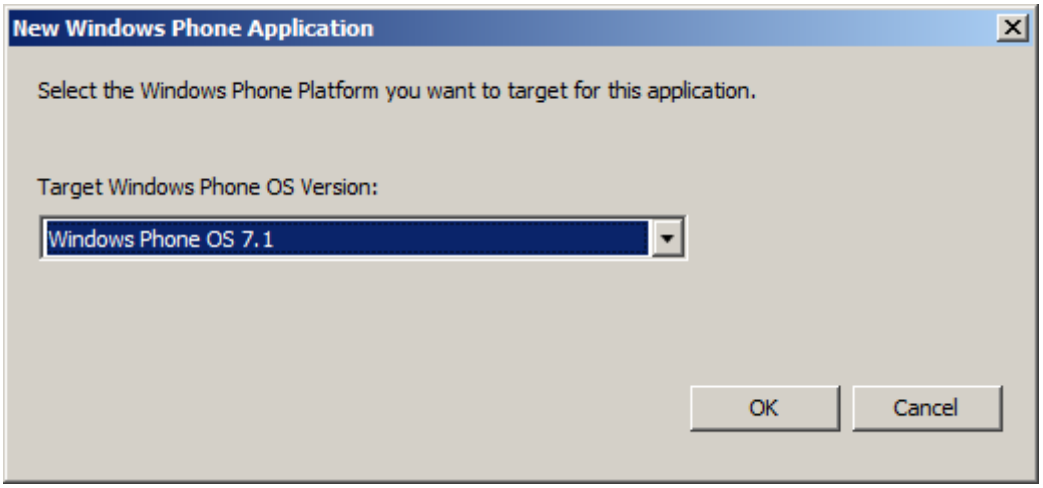
Windows Phone 7, Windows Phone Mango, Isolated Storage, Linq To Sql, C#, Linq

Merhaba Arkadaşlar,
Uzun zamandır bilgisayar yazılım teknolojileri ile ilgileniyor olmama rağmen zaman içerisinde belirli konularda uzmanlaşmaya çalıştığımı fark ettim. Bana göre normalde olması gereken bu. Nitekim insanın kapasitesini bilmesi ve her şeyden çok fazla anlamamaktansa, belirli bir konuda çok iyi bilgiye sahip olması daha anlamlıdır diye düşünüyorum
Ama tabi zaman zaman uzmanlık alanım dışındaki konulara da merak salmıyor değilim. Örneğin mobil platform üzerine geliştirme yapmak gibi. Her ne kadar **Microsoft** bu konuda elinden geleni yapıp işi son yıllarda daha da kolaylaştırıp **Windows Phone** gibi güzel bir zemin hazırlamış olsa da çok nadiren o tarafa gidip geliyorum.
Geçtiğimiz günlerde **Feedreader** üzerinden blogları şöyle bir tararken **Windows Phone** üzerinde kullanılabilecek olan veri depolama seçenekleri ile ilişkili kısa bir nota rast geldim.
Özellikle Isolated Storage tabanlı depolamalar üzerinde durulmaktaydı. Derken kendimi konuyu araştırır halde buldum. İşte bu yazının amacı elde edilen sonuçlar ve hoşunuza gidecek*(hoşuma gidecek)* bir örneği kaleme almak
Yıllar yıllar önce değil ama 2006 yılında Netron' da ilk Freelance eğitimimi bir ilaç firması*(Boehringer Ingelheim)* için vermiş ve Windows Mobile 6.5 üzerinde yazılım geliştirme anlatmıştım.
Compact .Net Framework ile ilişkili örnekleri ve konuları firmanın sağladığı HP marka akıllı telefonlarda ele almıştık. Styles Pen' ler ile çalışan ve kapasite olarak*(bellek, işlemci hızı, ekran çözünürlüğü vb)* sınırlı cihazlarda. Kim bilebilirdi ki iş bu noktaya kadar gelecek.
iPhone' lar, Blackberry' ler, Samsung Galaxy' ler ve tabi Windows Phone' lar. O zamanlarda Java tabanlı akıllı telefonlar yine .Net Compact Framework' lü olanlara göre çok daha iyiydi. Lakin bir süredir Windows Phone tarafının çok daha önemli bir atılım yaptığını ve arayı hızla kapattığını görüyoruz. En azından teknoloji ve yazılım geliştirme yetenekleri açısından.
Veri depolama sistemleri, mobil sistemlerdeki en önemli sıkıntılardan birisi olarak da karşımıza çıkıyor. Her ne kadar günümüz cihazlarında depolama alanlarının boyutu GB' lar cinsinden ifade edilebiliyor ve haricen kolayca genişleyebiliyor olsa da, ortada senkronizasyon gibi sıkıntılı iş senaryoları da bulunmakta. Yine de saha da çalışanların offline veri depolama kabiliyetleri ile çalışabilmesi önemli. Bilindiği üzere **Silverlight** ile

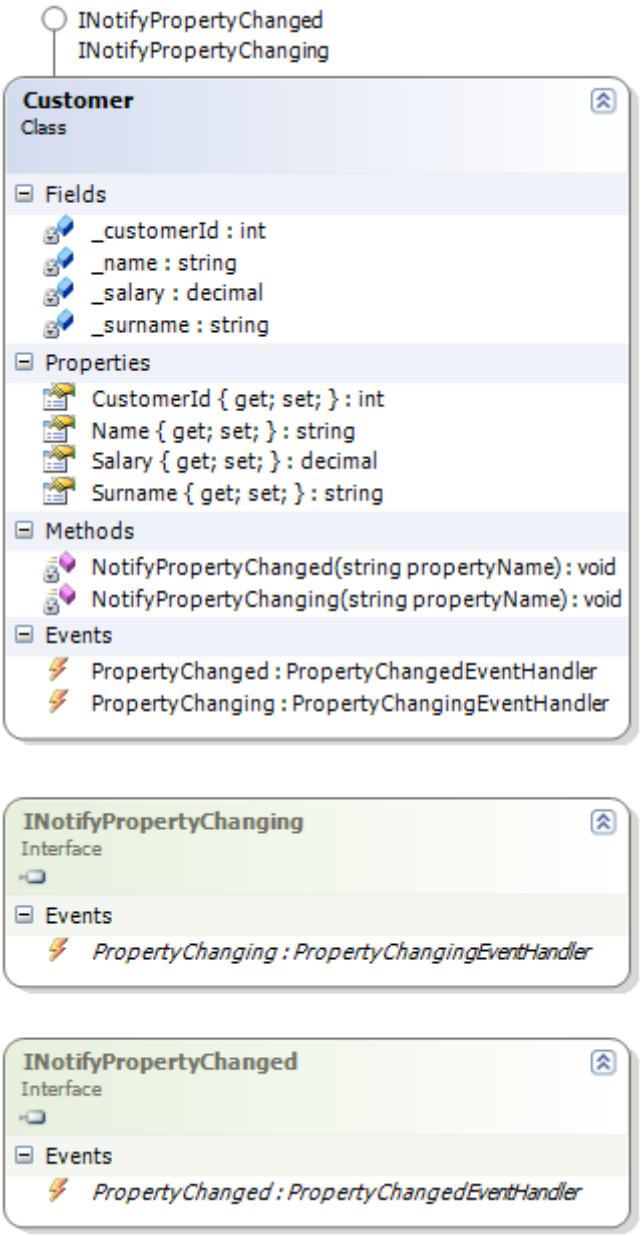
gelen **Isolated Storage** kavramı mobil taraf için de geçerli. Bu alan içerisinde veriyi saklamak için çeşitli yollara başvurabiliriz.
Örneğin veriyi **text** tabanlı olarak saklayabiliriz ve hatta bu sebepten **NoSQL(Not Only SQL)** gibi sistemleri de ele almayı düşünebiliriz. Diğer yandan bazı 3ncü parti kütüphanelerden veya SQL' in mobile taraf için kullanılabilecek sürümlerinden de yararlanabiliriz*(Bu adreste konu ile ilişkili detaylı bilgiye ulaşabilirsiniz)*
Biz bu günkü örneğimizde ise **LINQ to SQL** ve **Code First** benzeri*(benzeri diyorum çünkü tam olarak POCO tipleri söz konusu değil. Attribute' lar ile bezenmiş bir sınıf söz konusu olacak)* bir yaklaşımı ele alıyor olacağız. Dilerseniz hiç vakit kaybetmeden örneğimizi adım adım geliştirmeye başlayalım.
İlk olarak **Silverlight for Windows Phone** şablonunu kullanarak*(Windows Phone SDK' yı yüklemiş olduğunuzu varsayıyorum)* **MobileCustomerHouse** isimli bir uygulama oluşturalım.



Uygulamamızı **7.1** versiyonlu işletim sistemi sürümü için geliştiriyor olacağız.



Uygulamamızda **LINQ to SQL** tabanlı bir yapı kullanacağız. Bu sebepten **System.Data.Linqassembly'** ını projemize referans etmemiz gerekiyor. Bu işlemin ardından uygulamamıza aşağıda sınıf çizelgesi ve kod içeriği görülen **Customer** sınıfını ekleyebiliriz.




```
using System.ComponentModel;
using System.Data.Linq.Mapping;
namespace MobileCustomerHouse
{
   [Table(Name="Customer")]
   public class Customer
      :INotifyPropertyChanged,INotifyPropertyChanging
   {
      public event PropertyChangedEventHandler PropertyChanged;
      public event PropertyChangingEventHandler PropertyChanging;
      private void NotifyPropertyChanged(string propertyName)
      {
         if (PropertyChanged != null)
```

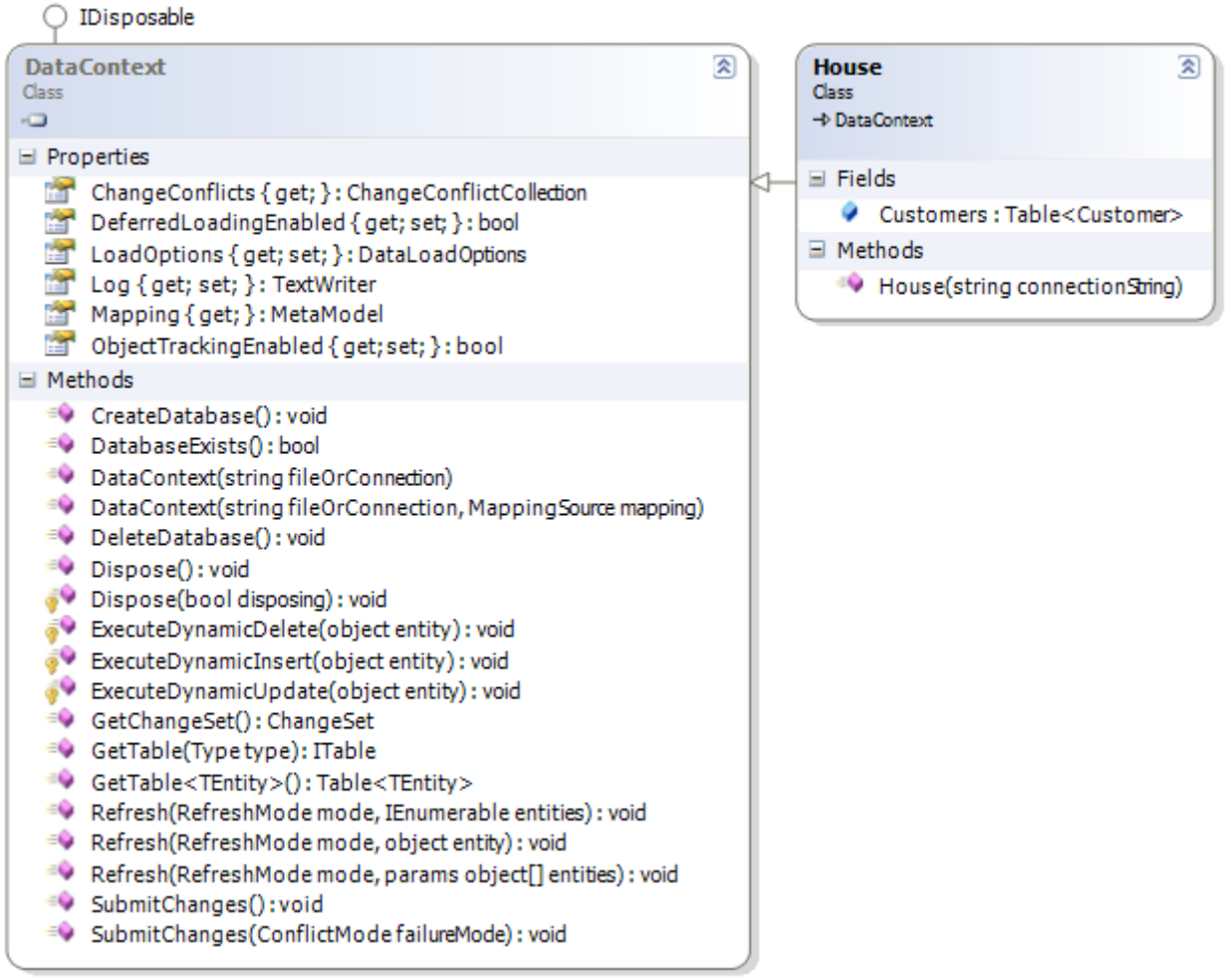
```
        {
            PropertyChanged(this, new PropertyChangedEventArgs(propertyName));
        }
    }
    private void NotifyPropertyChanging(string propertyName)
    {
        if (PropertyChanging != null)
        {
            PropertyChanging(this, new PropertyChangingEventArgs(propertyName));
        }
    }
    private int _customerId;
    private string _name;
    private string _surname;
    private decimal _salary;
    [Column(IsPrimaryKey = true, IsDbGenerated = true, DbType = "INT NOT
NULL Identity", CanBeNull = false, AutoSync = AutoSync.OnInsert)]
    public int CustomerId
    {
        get { return _customerId; }
        set
        {
            NotifyPropertyChanging("CustomerId");
            _customerId = value;
            NotifyPropertyChanged("CustomerId");
        }
    }
    [Column(DbType = "nvarchar(25) NOT NULL", CanBeNull = false, AutoSync =
AutoSync.OnInsert,Name="CustomerName")]
    public string Name
    {
        get { return _name; }
        set
        {
            NotifyPropertyChanging("Name");
            _name = value;
            NotifyPropertyChanged("Name");
        }
    }
    [Column(DbType = "nvarchar(25) NOT NULL", CanBeNull = false, AutoSync =
AutoSync.OnInsert,Name="CustomerSurname")]
```

```
        public string Surname
        {
            get { return _surname; }
            set
            {
                NotifyPropertyChanging("Surname");
                _surname = value;
                NotifyPropertyChanged("Surname");
            }
        }
        [Column(DbType = "money NOT NULL", CanBeNull = false, AutoSync =
AutoSync.OnInsert,Name="CustomerSalary")]
        public decimal Salary
        {
            get { return _salary; }
            set
            {
                NotifyPropertyChanging("Salary");
                _salary = value;
                NotifyPropertyChanged("Salary");
            }
        }
    }
}
```

Aslında örneğimizde View Model tarzı bir yaklaşımda bulunmayacağım. Bu sebepten Customer tipine INotifyPropertyChanged ve INotifyPropertyChanging arayüzlerini implemente etmesek de örneğimiz işlevsel olacaktır.
Ancak doğru olan, söz konusu tipin bir View ile birlikte ViewModel deseni içerisinde kullanılma ihtimalinin de olacağını göz önünde bulundurmaktır.
Siz, Model View View Model(MVVM) veya Model View Control(MVC) gibi desenleri göz önüne alarak bu yönde bir geliştirme yapmayı düşünebilirsiniz.
Customer tipi içerisinde dikkat edileceği üzere **CustomerId,Name,Surname** ve **Salary**özelliklerine ait **Set** bloklarına uygulanmış olan **Notify** çağrımları söz konusudur. **Customer** tipine**INotifyPropertyChanging** ve **INotifyPropertyChanged** aray üzlerini uyarladığımızdan, modelin görsel bir component ile bağlanması ve özelliklerde yapılan değişikliklerde **View** tarafının uyarılması/tepki verebilecek olması sağlanmaktadır. **Customer** tipinin diğer önemli özelliği ise tipin **Table** ve içeride yer alan özelliklerin de **Column**nitelikleri ile*(System.Data.Linq.Mapping isim alanından geliyorlar)* dekore edilmiş olmalarıdır. Bu**nitelikler(Attribute)** ile tahmin edeceğiniz üzere **Customer** tipine ait tablo şemasının içeriği belirlenmektedir. Örneğin **CustomerId**, tablo tarafında otomatik

olarak artan **integer** tipinde bir alandır. Ayrıca **Primary Key** olarak tanımlanmıştır. **Name** ve **Surname** alanları **nvarchar**tipindeyken, **Salary** alanı **Money** tipi ndedir.

Entity Framework ve **LINQ to SQL** den bildiğimiz üzere aslında **Entity** tiplerine ait koleksiyon bazlı özelliklerin tutulduğu ayrı bir **Context** sınıfı söz konusudur. **Code First** yaklaşımında**DbContext** türevli, klasik **Entity Framework** yaklaşımında ise **ObjectContext** türevli olan bu yapı**LINQ to SQL** tarafını göz önüne aldığımızda ise **DataContex'** den türemek anlamına gelmektedir. İşte biz de örneğimizde asıl veritabanını map eden bir **Context** tipinden yararlanıyor olacağız. Aşağıda ki sınıf çizelgesi ve kod parçasında görüldüğü gibi.




```
using System.Data.Linq;
namespace MobileCustomerHouse
{
   public class House
      :DataContext
   {
         public House(string connectionString)
            : base(connectionString)
         {
```

```
            Customers = this.GetTable<Customer>();
        }
    public Table<Customer> Customers;
  }
}
```

Örneğin basit tutulması amacıyla sadece Customer tipi ve buna ait veri kümesini işaret eden Customers isimli Table<Customer> sınıfından bir özellik kullanılmıştır. Size tavsiyem, örneği ilişkisel(Relational) bir/bir kaç tabloyu daha işin içerisine katarak ilerlemeye çalışmanız olacaktır. Örneğin müşterilerin hesap bilgilerini tutabileceğiniz ve One-to-Many ilişkiyi ifade eden bir geliştirmeyi nasıl yapacağınızı düşünebilirsiniz.

Peki veritabanımız ne zaman üretilecek? Arayüzümüz nasıl olacak? Dilerseniz uygulamamıza ait XAML içeriğini ve tasarımını aşağıdaki gibi geliştirererk ilerlemeye devam edelim.

MainPage.XAML


```
<phone:PhoneApplicationPage
    x:Class="MobileCustomerHouse.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:phone="clr-namespace:Microsoft.Phone.Controls;assembly=Microsoft.Phone"
    xmlns:shell="clr-namespace:Microsoft.Phone.Shell;assembly=Microsoft.Phone"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d" d:DesignWidth="480" d:DesignHeight="768"
    FontFamily="{StaticResource PhoneFontFamilyNormal}"
    FontSize="{StaticResource PhoneFontSizeNormal}"
    Foreground="{StaticResource PhoneForegroundBrush}"
```

```
    SupportedOrientations="Portrait" Orientation="Portrait"
    shell:SystemTray.IsVisible="True">
    <!--LayoutRoot is the root grid where all page content is placed-->
    <Grid x:Name="LayoutRoot" Background="Transparent">
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>
        <!--TitlePanel contains the name of the application and page title-->
        <StackPanel x:Name="TitlePanel" Grid.Row="0" Margin="12,17,0,28">
            <TextBlock x:Name="ApplicationTitle" Text="Customer House"
Style="{StaticResource PhoneTextNormalStyle}"/>
            <TextBlock x:Name="PageTitle" Text="Customer" Margin="9,-7,0,0"
Style="{StaticResource PhoneTextTitle1Style}"/>
        </StackPanel>
        <!--ContentPanel - place additional content here-->
        <Grid x:Name="ContentPanel" Grid.Row="1" Margin="12,0,12,0"></Grid>
        <StackPanel Height="607" HorizontalAlignment="Left" Margin="12,0,0,0"
Name="stackPanel1" VerticalAlignment="Top" Width="456" Grid.Row="1">
            <TextBlock Height="30" Name="textBlock1" Text="Name" />
            <TextBox Height="71" Name="txtCustomerName" Text="customer name"
Width="460" />
            <TextBlock Height="30" Name="textBlock2" Text="Surname" />
            <TextBox Height="71" Name="txtSurname" Text="customer surname"
Width="460" />
            <TextBlock Height="30" Name="textBlock3" Text="Salary" />
            <TextBox Height="71" Name="txtCustomerSalary" Text="1000" Width="460" />
            <Button Content="Insert" Height="71" Name="btnInsert" Width="160"
HorizontalContentAlignment="Center" Click="btnInsert_Click" />
            <ListBox Height="226" Name="lstCustomers"
Width="460" ItemsSource="{Binding}">
                <ListBox.ItemTemplate>
                    <DataTemplate>
                        <StackPanel Orientation="Horizontal">
                            <TextBlock Foreground="Gold" Margin="2,2,2,2" Text="{Binding
Name}"/>
                            <TextBlock Foreground="Gold" Margin="2,2,2,2" Text="{Binding
Surname}"/>
                            <TextBlock Foreground="Red" Margin="2,2,2,2" Text="{Binding
Salary}"/>
                        </StackPanel>
```

```
            </DataTemplate>
          </ListBox.ItemTemplate>
        </ListBox>
      </StackPanel>
   </Grid>
</phone:PhoneApplicationPage>
```

Görsel arabirim de çok özel bir şey yok aslına bakarsanız. Dikkate değer nokta **ListBox** kontrolünün bir **DataTemplate** ile ilişkilendirilmiş olmasıdır. **ListBox** kontrolü, **Context** nesnesinin ilgili özelliğine bağlandıktan **sonra(Binding),** içeride yer alan **TextBlock** bileşenleri de sırasıyla **Name**,**Surname** ve **Salary** özelliklerine bağlanmıştır*(CustomerId' yi unutmuşum onu da siz ekleyiverin)*Şimdi arka plan kodlarını yazarak örneğimizi genişletmeye devam edebiliriz.

```
using System;
using System.Linq;
using Microsoft.Phone.Controls;
namespace MobileCustomerHouse
{
   public partial class MainPage
      : PhoneApplicationPage
   {
      House houseDb = null;
      public MainPage()
      {
         InitializeComponent();
         houseDb = new House("Data Source = 'isostore:/House.sdf'; File Mode = read
write");
         if (!houseDb.DatabaseExists())
         {
            houseDb.CreateDatabase();
         }
      }
      private void btnInsert_Click(object sender, System.Windows.RoutedEventArgs e)
      {
         Customer newCustomer = new Customer
        {
            Name=txtCustomerName.Text,
            Surname=txtSurname.Text,
           Salary=Convert.ToDecimal(txtCustomerSalary.Text)
         };
```

```
            houseDb.Customers.InsertOnSubmit(newCustomer);
            houseDb.SubmitChanges();
            lstCustomers.ItemsSource = houseDb.Customers.ToList();
        }
    }
}
```

Yapıcı metod içerisinde House tipinden bir nesne örneklendiği görülmektedir. Bu nesne verdiğimiz Connection String bilgisine göre Isolated Storage üzerinde House.sdf isimli bir dosya oluşturulacaktır. Dikkat edilecek olursa **DatabaseExists()** metodu ile veritabanının önceden yaratılıp yaratılmadığı kontrol edilmektedir.

Eğer veritabanının şema(Schema) yapısında kod tarafında değişiklik yapılması planlanıyorsa Microsoft.Phone.Data.Linq isim alanında bulunda DatabaseSchemaUpdater tipi ve üyelerinden yararlanılabilir.

Button kontrolünde basıldığın **TextBox** kontrollerindeki verilerden yararlanılarak bir **Customer**nesnesini örneklenmekte ve **houseDb** isimli veritabanı içerisine ilave edilkmektedir. Nesnenin önce**Context**' eklenmesi için **InsertOnSubmit** metodundan faydalanılmış ve değişikliklerin veritabanı üzerine yazılması için de **SubmitChanges** fonksiyonu çağırılmıştır.

Ekleme işlemi tamamlandıktan sonra ise **ListBox** kontrolünün **ItemsSource** özelliğine gerekli **veri bağlama(Data Binding)** işlemi yapılmaktadır. Örneği test ettiğimizde ve bir kaç örnek veri içeriğini girdiğimizde aşağıdaki ekran görüntüsündekine benzer bir çalışma zamanı sonucu ile karşılaşırız.

Görüldüğü üzere örnek olarak eklenen **Customer** tipleri **ListBox** kontrolü içerisine basılmıştır.

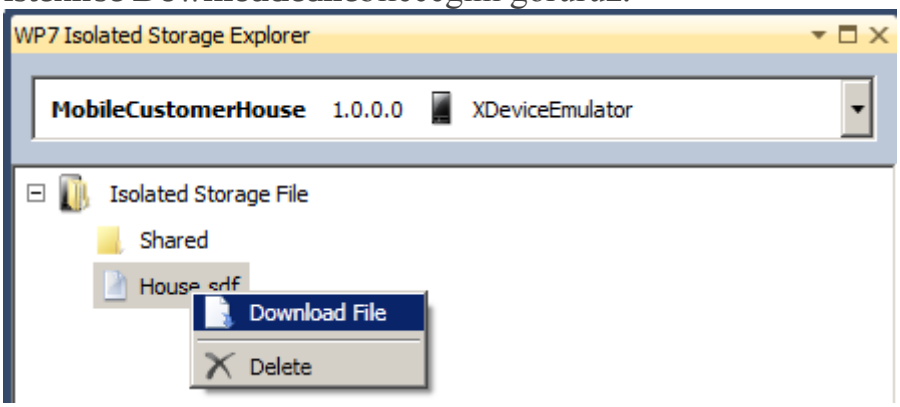
Peki gerçekten de **Isolated Storage** alanı içerisinde **sdf** veritabanı oluşturulmakta mıdır? Örneğin başarılı bir şekilde çalışması nedeni ile bunun doğru olduğu görülmektedir ama biz yine de bakmak istediğimizi farz edelim

Bunu görmek için **Isolated Storage** içeriğine kod yardımıyla da bakabiliriz. Ama benim önerim**Codeplex** sitesinde yayınlanan (http://wp7explorer.codeplex.com/) **Windows Phone 7 Isolated Storage Explorer**' ın kullanılması olacak. Bu ürünü kurduktan sonra **Visual Studio**' nun View menüsüne **Isolated Storage Explorer** penceresinin ilave

edildiği gözlemlenebilir*(View->Other Windows->WP7 Isolated Storage Explorer)*. Tabi buraya çalışmakta olan bir Emulator' ü ttach' lamak için uygulamamıza **IsolatedStorageExplorer assembly'** ını referans etmemiz ve **Appl.xaml.cs**içerisinde aşağıda görülen kod değişikliklerini yapmamız gerekmektedir.

```
private void Application_Activated(object sender, ActivatedEventArgs e)
{
   IsolatedStorageExplorer.Explorer.RestoreFromTombstone();
}

private void Application_Deactivated(object sender, DeactivatedEventArgs e)
{
   IsolatedStorageExplorer.Explorer.RestoreFromTombstone();
}
```

Uygulamamızı yeniden çalıştırdığımızda ve **Isolated Storage Explorer** penceresini açtığımızda**House.sdf** isimli veritabanı dosyasının başarılı bir şekilde üretildiğini ve hatta istenirse **Download**edilebileceğini görürüz.



[Bu arada söz konusu ürün makaleyi yazdığım tarih itibariyle Beta sürümündeydi. Dolayısıyla güncellenmiş ve farklı kabiliyetler ile donatılmış olabilir]

Böylece geldik kısa bir maceramızın daha sonuna. Görüldüğü üzere **LINQ to SQL**' i çok basit anlamda ele alarak özel depolama alanında bir veritabanının tutulmasını sağlayabildik. Yukarıda belirtiğim gibi aslında örneği **MVVM** veya **MVC** çerçvesinde göz önüne alarak arayüz ile daha güçlü entegre olacak şekilde geliştirmeye çalışmanızı öneririm. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

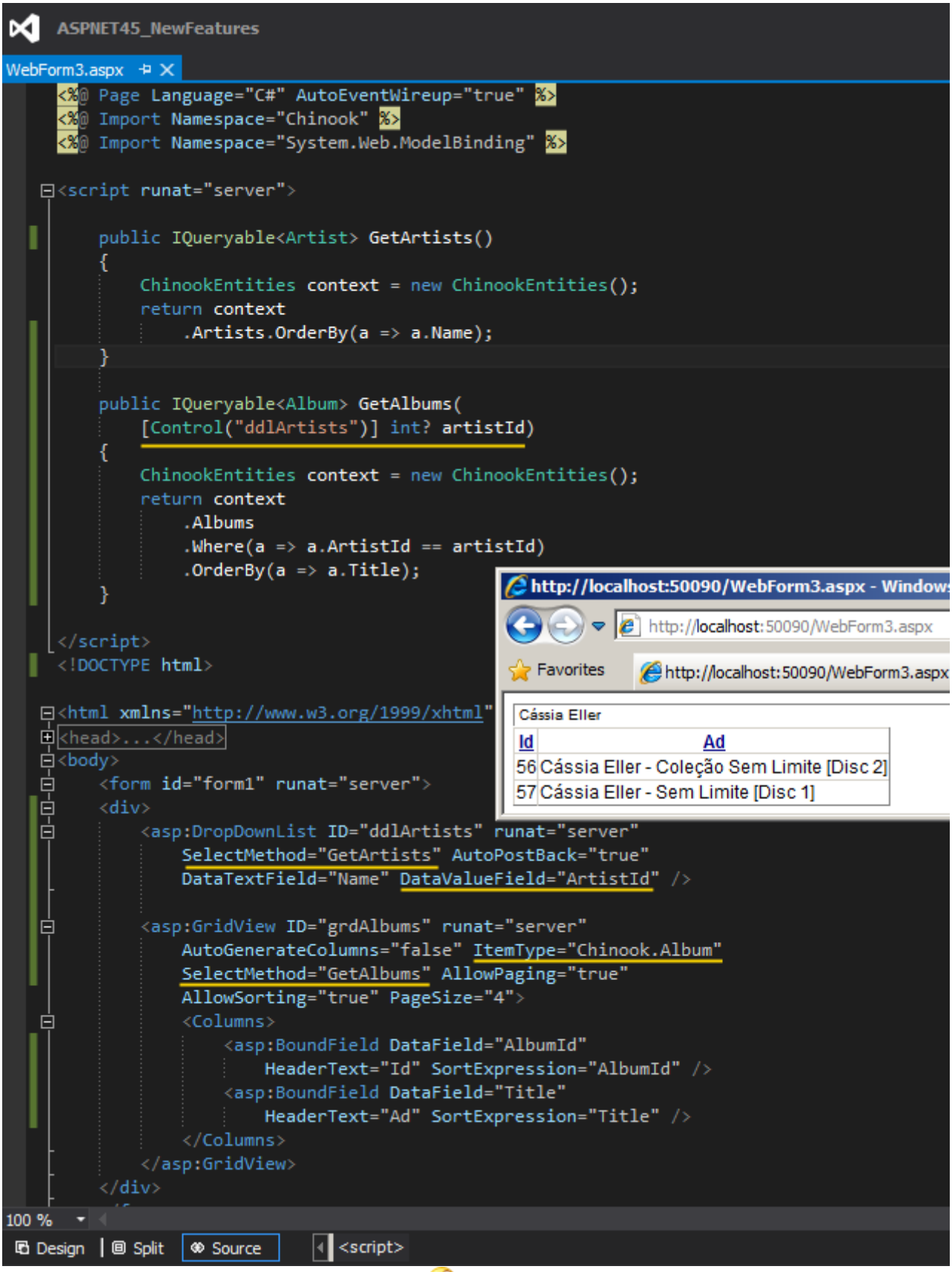
MobileCustomerHouse.zip (133,80 kb)

Tek Fotoluk İpucu 67.75–Asp.Net 4.5 ControlAttribute

Salı, 2 Ekim 2012 11:00

Tek Fotoluk Ipucu, Asp.Net 4.5, Linq, Iqueryable<T>, Controlattribute,System.Web.Modelbinding, Attribute, Nitelik, C #

Merhaba Arkadaşlar,
Asp.Net 4.5 ile gelen önemli tiplerden birisi de, **System.Web.ModelBinding** isim alanı*(System.Web.dll assembly' ı içerisindedir)* altında yer alan **ControlAttribute**niteliğidir*(Attribute)*. **Metod** parametrelerine uygulanabilen bu nitelik ile, veri bağlı kontrollerin*(GridView gibi)* filtre bazlı çalıştığı senaryolarda, filtreleme kriterinin/kriterlerinin nereden alınacağı, kod seviyesinde kolayca belirtilebilir. Aşağıdaki fotoğrafta görülen örnekte, albümlerin sorgulanmasında kullanılan **ArtistId** değerinin bir **DropDownList** öğesinden çekileceği,**GetAlbums** metodu içerisindeki **Control** niteliği yardımıyla ifade edilmiştir

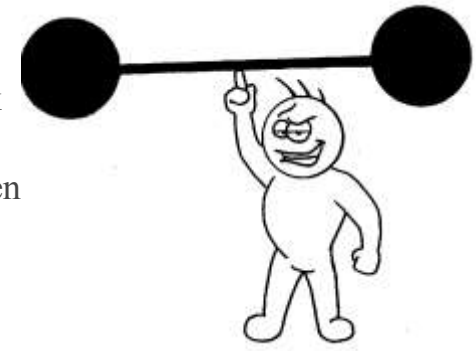


Bir başka ipucunda görüşmek dileğiyle

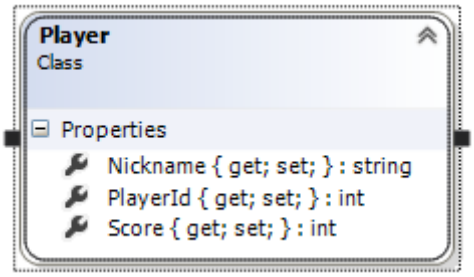
Asp.Net 4.5- Strongly Typed Data Control

Pazartesi, 1 Ekim 2012 11:45

Asp.Net 4.5, Strongly Typed Data Controls, Binditem, Bind, Eval, C#, Asp.Net



Merhaba Arkadaşlar,
Malumunuz Web tarafı ile aram pek iyi değildir.
Ancak **.Net Framework**' ün her sürümünde genel olarak gelen yeniliklere bakmaya çalışıyorum/çalışmaktayım.
Geçtiğimiz hafta içerisinde de **Asp.Net 4.5** tarafında gelen yenilikleri incelemeye başladım.
Bunlar arasında dikkatimi çekenlerden birisi de, **Web Form**' larda veri bağlı kontroller*(Data Bind Controls)* için gelen**strongly typed** ve **intelli-sense** desteğiydi. Durumu daha iyi aktarabilmem için basit bir örnek üzerinden ilerlemeye çalışım. İlk etapta aşağıdaki gibi bir **POCO(Plain OLD CLR object)** tipimiz olduğunu düşünelim.
(Bu arada yandaki halter kaldıran adam ne alak diyebilirsiniz. Giriş yazısını düşünürken, Strongly kelimesinden Strong ifadesine gelince, bunu anlatabilecek fotoğraflardan birisi olarak karşıma çıktı)




```
namespace STDC_Old
{
   public class Player
   {
      public int PlayerId { get; set; }
      public string Nickname { get; set; }
      public int Score { get; set; }
   }
}
```

Senaryomuzda bu basit **POCO** tipine ait nesne örneklerinden oluşan **generic** bir koleksiyonu, sayfa üzerindeki bir **FormView** kontrolüne bağlıyor olacağız*(senaryoda örnek olarak **FormView**kontrolünü göz önüne aldık. Pek tabi diğer **data bind** bileşenleri de örneğe katabilirsiniz)* Web uygulamamıza ait arka plan kodlarını aşağıdaki gibi

tasarlayabiliriz. *(Projemiz **Asp.Net Empty Web Application** tipindedir ve **.Net Framework 4.0** hedefli olarak oluşturulmuştur)*

```
using System;
using System.Collections.Generic;
using System.Web.UI;
namespace STDC_Old
{
   public partial class Default
      : System.Web.UI.Page
   {
      static List<Player> players = new List<Player>
      {
         new Player{ PlayerId=1, Nickname="Brit", Score=125},
         new Player{ PlayerId=2, Nickname="Kuşbeyin", Score=250},
         new Player{ PlayerId=3, Nickname="Zeyno", Score=90},
         new Player{ PlayerId=4, Nickname="Nikol", Score=175}
      };
      protected void Page_Load(object sender, EventArgs e)
      {
         if (!Page.IsPostBack)
         {
            formviewPlayers.DataSource = players;
            formviewPlayers.DataBind();
         }
      }
   }
}
```

Kod parçasında görüldüğü üzere **formviewPlayers** isimli **FormView** kontrolüne **players** isimli bir koleksiyon içeriği bağlanmaktadır. Peki ya tasarım ortamında durum nedir? Tahmin edileceğiz üzere çalışma zamanında, bir **Player** nesne örneğinin **PlayerId**, **Nickname** ve **Score** gibi özelliklerinin değerlerini tek(one way) veya çift yönlü(two way) olacak şekilde göstermek için **Bind** veya **Eval**tiplerinden yararlanılmaktadır. Bu tipleri genellikle **Item Template** elementleri içerisindeki kontrollerde sıklıkla kullanırız. Söz gelimi aşağıdaki gibi

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="STDC_Old.Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
```

```
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:FormView runat="server" ID="formviewPlayers" DefaultMode="Edit">
      <EditItemTemplate>
               Player Id
               <asp:Label ID="lblPlayerId" runat="server" Text='<%#Bind("PlayerId")
%>'/>
               <br />
               Nick name
               <asp:TextBox ID="txtNickname"
runat="server" Text='<%#Bind("Nickname") %>' />
               <br />
               Score
               <asp:TextBox ID="txtScore" runat="server" Text='<%#Bind("Score")
%>' />
               <br />
         <asp:Button ID="btnUpdate" Text="Update Player Informations" runat="server"
CommandName="Update"/>
          <br />
      </EditItemTemplate>
</asp:FormView>
   </div>
   </form>
</body>
</html>
```

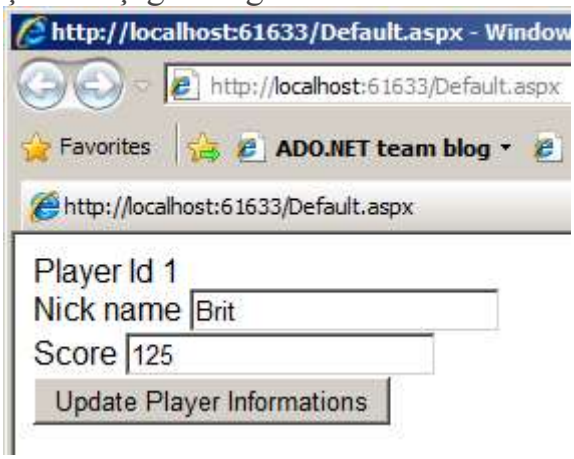
Dikkat edileceği üzere **lblPlayerId**, **txtNickname**, **txtScore** isimli bileşenlerin **Text** nitelikleri,**Player** tipinin sırasıyla **PlayerId, Nickname** ve **Score özelliklerine(Properties)** bağlanmışlardır. FormView bileşeninde, **Bind** tipi kullanılmış ve çift yönlü veri bağlama imkanı sunulmuştur. Ama bildiğiniz üzere **Eval** de kullanılabilir ve tek yönlü bir veri akışı da sağlanabilir. Kaldı ki hangisini kullandığımızı şu anda pek bir önemi yok

Peki buradaki kodları yazarken hiç şöyle bir şey de olsun ister miydiniz?

Bind veya Eval kullanırken keşke kontrolü bağladığımız tipin özelliklerini de görebilsekde, neyi neye bağladığımızı daha kolay takip edebilsek. Hatta burada Intelli-sense kabiliyetleri de bize yardımcı olsa

Çünkü yukarıdaki kod parçasını yazarken **Eval** veya **Bind** kullanımlarında tamamen ezbere olacak şekilde **string** veriler yazılmaktadır. Bu özellikle kalabalık veri toplulukları göz önüne alındığında dikkat dağıtıcı ve zorlayıcı bir durumdur. Ayrıca hata yapma riski vardır. İfadeler yanlış yazılabilir ve sonuçlar ancak çalışma zamanında görülecektir.

Diğer yandan arka plandaki çalışma sistematiğine baktığımızda, bağlanan özelliğin elde edilebilmesi için **string** içeriğin çözümlenmesi gerektiği de aşikardır. Bu da çok doğal olarak **reflection** tabanlı bir yaklaşımı gerektirmektedir. Bu arada yukarıdaki kodun ekran çıktısı aşağıdaki gibi olacaktır onu da ifade edeyim.



Gelelim **Asp.Net 4.5** tarafındaki duruma. Yine aynı örnek senaryoyu ele alacağız ancak bu kez veri bağlı kontrolümüzün **ItemType** isimli özelliğine, bağlanacak olan tipin*(kuvvetle muhtemel bir sınıf adı olacaktır)* adını vereceğiz. Aşağıdaki ekran görüntüsünde olduğu gibi. Üstelik **intelli-sense**desteğimiz de var

```
<form id="form1" runat="server">
<div>
        <br />
    <asp:FormView runat="server" ID="formviewPlayers" DefaultMode="Edit"
        ItemType=""
                STDC_New.Player
```

Volaaaa! demek için aslında çok erken. Nitekim iş asıl tipi belirttikten sonra güzelleşiyor. **ItemType**tanımlaması ile söz konusu kontrolün veri bağlı içeriğinin*(bir başka deyişle child element olacak kontrollerin)* hangi **.Net** tipini değerlendirmesi gerektiğini ifade etmiş oluyoruz. İşte bu noktadan sonra sunucu bazlı kontrollerin ilgili özelliklerine veri bağlama işlemlerini yaparken **intelli-sense**özelliğinden yararlanabiliriz. Artık elimizde **Strongly Typed** bir veri bağlı kontrol bulunmakta. Aşağıdaki ekran görüntüsünde bu durum ve kullanım kolaylığı daha net bir şekilde görülebilir.

```
<asp:Label ID="Label1" runat="server" Text='<%#BindItem.%>'/>

<asp:Label ID="Label2" runat="server" AssociatedControlI
        Nick name </asp:Label>
<asp:TextBox ID="nickName" runat="server"
        Text='<%#BindItem.Nickname %>' />

    <div>
<asp:Label ID="Label3" runat="server" AssociatedControlI
```

Equals
GetHashCode
GetType
Nickname
PlayerId
Score
ToString
int Player.Score

Sanırım şimdi volaaaaaa!!! diyebiliriz

Buradaki yaklaşım XAML tarafından size tanıdık gelecektir. XAML tarafında da asıl kontrolün bir veri kaynağına bağlanması(yani bir Resource bağlanması), iç

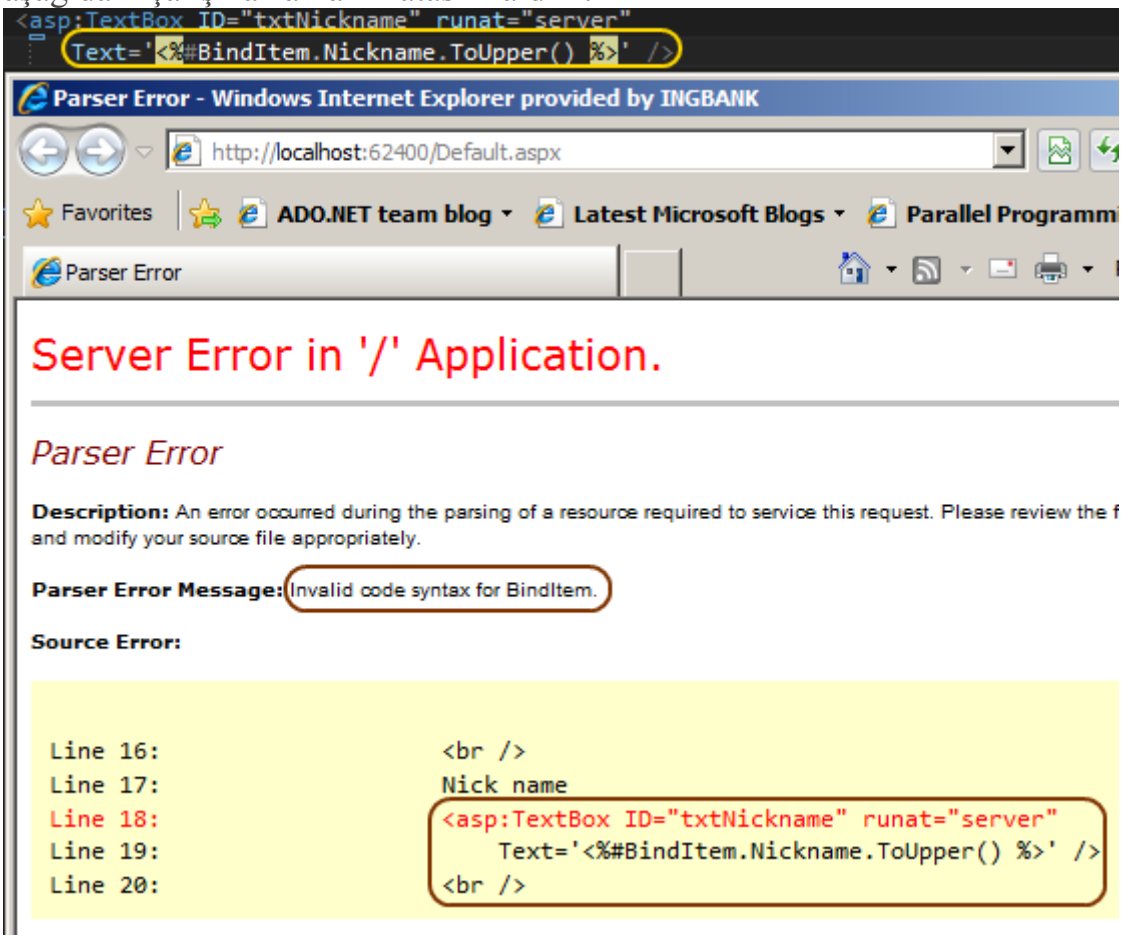
elementlerin özelliklerinin de bu verinin niteliklerine atanması işlemi, dekleratif olarak yapılmaktadır.

Bu durumda **Asp.Net 4.5** versiyonu ile senaryomuzun **aspx** içeriği aşağıdaki gibi olacaktır.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="STDC_New.Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
            <br />
      <asp:FormView runat="server" ID="formviewPlayers"
DefaultMode="Edit"ItemType="STDC_New.Player">
      <EditItemTemplate>
            Player Id
            <asp:Label ID="lblPlayerId" runat="server" Text='<%#BindItem.PlayerId
%>'/>
            <br />
            Nick name
            <asp:TextBox ID="txtNickname"
runat="server" Text='<%#BindItem.Nickname %>'/>
            <br />
            Score
            <asp:TextBox ID="txtScore" runat="server" Text='<%#BindItem.Score
%>' />
            <br />
        <asp:Button ID="btnUpdate" Text="Update Player Informations" runat="server"
CommandName="Update"/>
      </EditItemTemplate>
</asp:FormView>
   </div>
   </form>
</body>
</html>
```

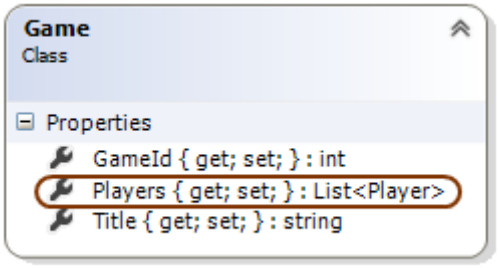
Tabi bu esnekliği kullanırken aklınıza **“acaba noktaya basıp özellik adını yazdıktan sonra gelen fonksiyonellikleri de kullanabilir miyiz?”** diye bir soru gelebilir Söz gelimi “string tipinden olan **Nickname’** i bağladıktan sonra bir de **ToUpper** metodunu

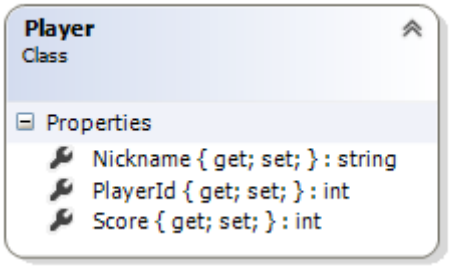
çağırsak da, ismi büyük harf yazsa olmaz mı?" diyebilirsiniz. Ben bunu denediğimde aşağıdaki çalışma zamanı hatasını aldım.



Bu gibi durumlara dikkat etmek gerektiğini ifade edebiliriz.

Şimdi senaryomuzu biraz daha ilginçleştirelim Bu sefer özelliklerinden birisi veri kontrolüne bağlanabilir liste koleksiyonu tipinden olan bir sınıfı ele alıp çalışma zamanı içeriğini **master-detail**formatında göstermeye çalışıyor olacağız. Senaryomuzda yine yeni gelen **ItemType** ve **BindItem**özelliklerine yer vereceğiz. İlk etapta uygulamamıza **Game** isimli yeni bir POCO tipi eklediğimizi düşünelim. Aşağıdaki sınıf diyagramında görüldüğü gibi.





Dikkat edileceği üzere **Game** tipi içerisinde **Player** sınıfına ait örnekleri taşıyacak **List** tipinden**generic** bir özellik de bulunmaktadır. Dolayısıyla bir Game alanındaki n sayıda Player' ı taşıyabiliriz.**Default.aspx.cs** içeriğini de aşağıdaki gibi kodlayabiliriz.

```
using System;
using System.Collections.Generic;
using System.Web.UI;
namespace STDC_New
{
   public partial class Default
      : System.Web.UI.Page
   {
      static List<Player> players1 = new List<Player>
      {
         new Player{ PlayerId=1, Nickname="Brit", Score=125},
         new Player{ PlayerId=2, Nickname="Kuşbeyin", Score=250},
         new Player{ PlayerId=3, Nickname="Zeyno", Score=90},
         new Player{ PlayerId=4, Nickname="Nikol", Score=175}
      };
      static List<Player> players2 = new List<Player>
      {
         new Player{ PlayerId=6, Nickname="Legolat", Score=450},
         new Player{ PlayerId=7, Nickname="Aragorn", Score=485},
         new Player{ PlayerId=8, Nickname="Gandalf the Gray", Score=555},
         new Player{ PlayerId=9, Nickname="Arven", Score=1005}
      };
```

```
        static List<Game> mordor = new List<Game>
        {
            new Game{
             GameId=1001,
             Title="Mordor Game Zone",
             Players=players1
            },
            new Game{
             GameId=4587,
             Title="Gondor Game Zone",
             Players=players2
            }
        };
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!Page.IsPostBack)
            {
                listViewGame.DataSource = mordor;
                listViewGame.DataBind();
            }
        }
    }
}
```

Bizim için önemli olan nokta listViewGame isimli ListView kontrolünün içeriğidir. Bu içeriği aşağıdaki gibi ürettiğimizi düşünelim.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="STDC_New.Default" %>
<!DOCTYPE html>
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:ListView runat="server" ID="listViewGame"
              ItemType="STDC_New.Game">
               <ItemTemplate>
                  <div style="background-color: gold">
                     <asp:Label ID="lblGameIt" runat="server"
                       Text="<%#BindItem.GameId %>" />
```

```
                Title :
                <asp:Label ID="lblGameTitle" runat="server"
                  Text="<%#BindItem.Title %>" /><br />
              </div>
              <div style="border: medium; border-color: black; background-color:
lightgray">
                <asp:ListView ID="listViewPlayers"
                  DataSource="<%#BindItem.Players %>"
                  ItemType="STDC_New.Player" runat="server">
                  <ItemTemplate>
                    <div style="border: double; border-color: red">
                      <asp:Label ID="lblPlayerId" runat="server"
                        Text='<%#BindItem.PlayerId %>' /><br />
                      Nickname       :
                      <asp:Label ID="txtNickname" runat="server"
                        Text='<%#BindItem.Nickname %>' /><br />
                      Current Score   :
                      <asp:Label ID="txtScore" runat="server"
                        Text='<%#BindItem.Score %>' /><br />
                    </div>
                  </ItemTemplate>
                </asp:ListView>
              </div>
            </ItemTemplate>
          </asp:ListView>
        </div>
    </form>
</body>
</html>
```

İlk olarak en dıştaki **ListView** kontrolünün **ItemType** özelliğine **STDC_New.Game** değerini atadığımızı görmekteyiz. Buna göre **listViewGame** kontrolünün alt elementlerindeki bileşenlere ait özelliklerde, **Game** sınıfının özelliklerini kullanabiliriz. **lblGameId** ve **lblGameTitle** isimli **Label**kontrollerindeki **Text** özelliklerini, **Game** tipinin sırasıyla **GameId** ve **Title** özelliklerine bağlıyoruz.

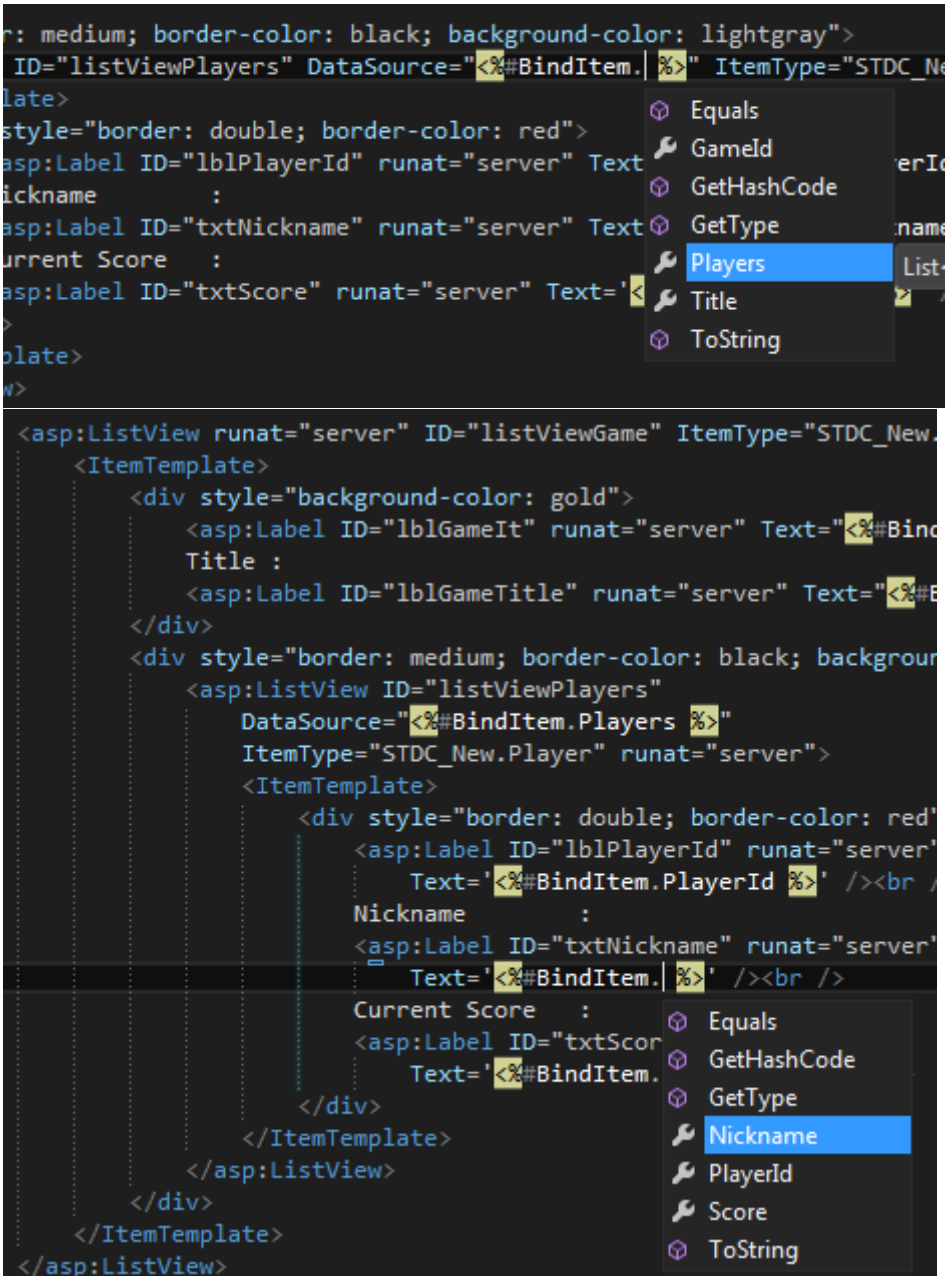
Yine alt elementlerden birisi olarak iç kısımda yer alan **listViewPlayers** isimli **ListView** bileşeninde ise, daha farklı bir veri bağlama işlemi yapıldığı hemen gözünüze çarpmış olmalıdır. Dikkat edileceği üzere **DataSource** elementine **BindItem.Players** şeklinde bir atama yapılmıştır. Bir başka deyişle çalışma zamanında üst tarafa bağlı olan **Game** nesne örneğinin

içerisindeki **Players** özelliğinin işaret ettiği koleksiyonun veri kaynağı olarak kullanılacağı belirtilmektedir.

Eğer DataSource özelliğine bu şekilde bir atama işlemi yapılmassa, Game örneklerinin Players koleksiyonlarına ait içerikleri ekrana basılmayacaktır. Böyle bir durumda aşağıdakine benzer bir ekran çıktısı elde edilir.

Ayrıca **listViewPlayer ListView** kontrolünün **ItemType** özelliğine de **SDTC_New.Player** değeri verilmiştir. Buna göre alt elementlerdeki kontrollerin özelliklerinde **Player** sınıfının özellikleri kullanılabilecektir.

Pek tabi kodu yazarken intelli-sense özelliği de devreye girmekte ve aşağıdaki ekran çıktılarında olduğu gibi bizlere kolaylık sağlamaktadır.



Örneği çalıştırdığımızda çalışma zamanında aşağıdaki ekran görüntüsünde yer alan sonuçları aldığımızı görürüz.

Görüldüğü gibi iç içe iki **ListView** bileşeni **Master-Detail** formasyonda oyun sahalarını ve bu sahalardaki oyuncuları gösterecek şekilde üretilebilmiştir. Böylece geldik bir yazımızın daha sonuna. Bu kısa çerez tadındaki yazı ile **Asp.Net 4.5** ile gelen yeni kabiliyetlerden birisine değinmiş olduk. Bir başka yazımızda görüşmek dileğiyle, hepinize mutlu günler dilerim.

ASPNET45_NewFeatures.zip (44,72 kb)

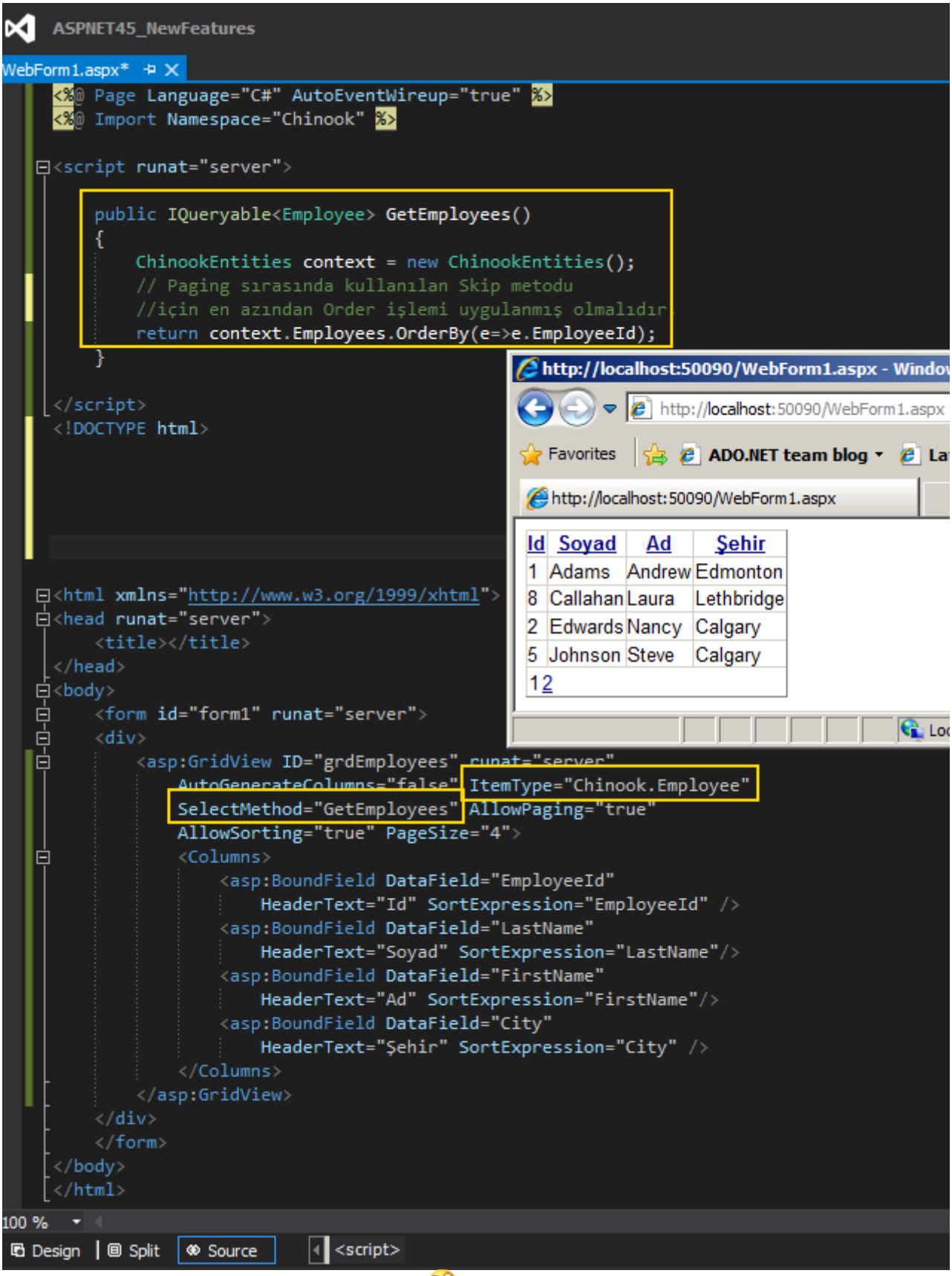
Tek Fotoluk İpucu 67.5–Asp.Net 4.5 No More DataBind

Salı, 25 Eylül 2012 14:45

Tek Fotoluk Ipucu, Asp.Net 4.5, Model Binding, Linq, Entity Framework

Merhaba Arkadaşlar,
Asp.Net 4.5 Web Forms tarafında gelen yeniliklerden birisi de, veri bağlı kontrolleri**IQueryable<T>** veya **IEnumerable<T>** tipinden arayüz referanslarına bağlarken **DataBind**fonksiyon çağrısı yapılması zorunluluğu olmamasıdır. Bu sayede Markup tarafında sadece **Select**metodunun bildirilmesi yeterli olmaktadır. Aşağıdaki ekran görüntüsünde olduğu gibi

[Örnekte kullanılan Chinook isim alanı, meşhur Chinook veritabanının Entity Framework tabanlı modelinin oluşturulduğu yerdir]



Bir başka ipucunda görüşmek dileğiyle

Tek Fotoluk İpucu–67–Fibonacci, LINQ, Skip ve Take

Pazar, 23 Eylül 2012 09:23

Tek Fotoluk Ipucu, Biginteger, Linq, Fibonacci, Skip, Take, Linq, Ienumerable<T>,Yield

Merhaba Arkadaşlar,
Biliyorsunuz **.Net Framework 4.0** ile birlikte **BigInteger** tipi geldi ve çok büyük sayıları kullanabilir olduk. Şimdi diyelimki eğlencelik olsun diye **Fibonacci** sayılarının sonlu kümesine ve bu kümedende istediğiniz indexte başlayıp istediğiniz miktarda alabileceğiniz bir fonksiyonelliğe ihtiyacınız var. Ha bir de elinizin altında **yield** keyword' ü. Taaaa **.Net Framework 2.0**zamanlarından. Naparsınız? 😉

FunnyCodes - Program.cs

Program.cs

FunnyCodes.Program — Main(string[] args)

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Numerics;

namespace FunnyCodes{
    class Program{
        static void Main(string[] args){
            var array1 = FibonacciProvider.Get(0, 10);
            WriteToConsole(array1);
            var array2 = FibonacciProvider.Get(50, 5);
            WriteToConsole(array2);
        }

        private static void WriteToConsole(IEnumerable<BigInteger> sourceArray){
            foreach (var number in sourceArray)
                Console.Write("{0}\t",number.ToString());
        }
    }
    public class FibonacciProvider{
        public static IEnumerable<BigInteger> Get(int startIndex, int count){
            return GetInfinity().Skip(startIndex).Take(count);
        }
        private static IEnumerable<BigInteger> GetInfinity(){
            BigInteger number1 = 1;
            BigInteger number2 = 1;

            yield return 1;
            yield return 1;

            while (true)
            {
                BigInteger number3 = number1 + number2;
                number1 = number2;
                number2 = number3;
                yield return number3;
            }
        }
    }
}
```

C:\Windows\system32\cmd.exe

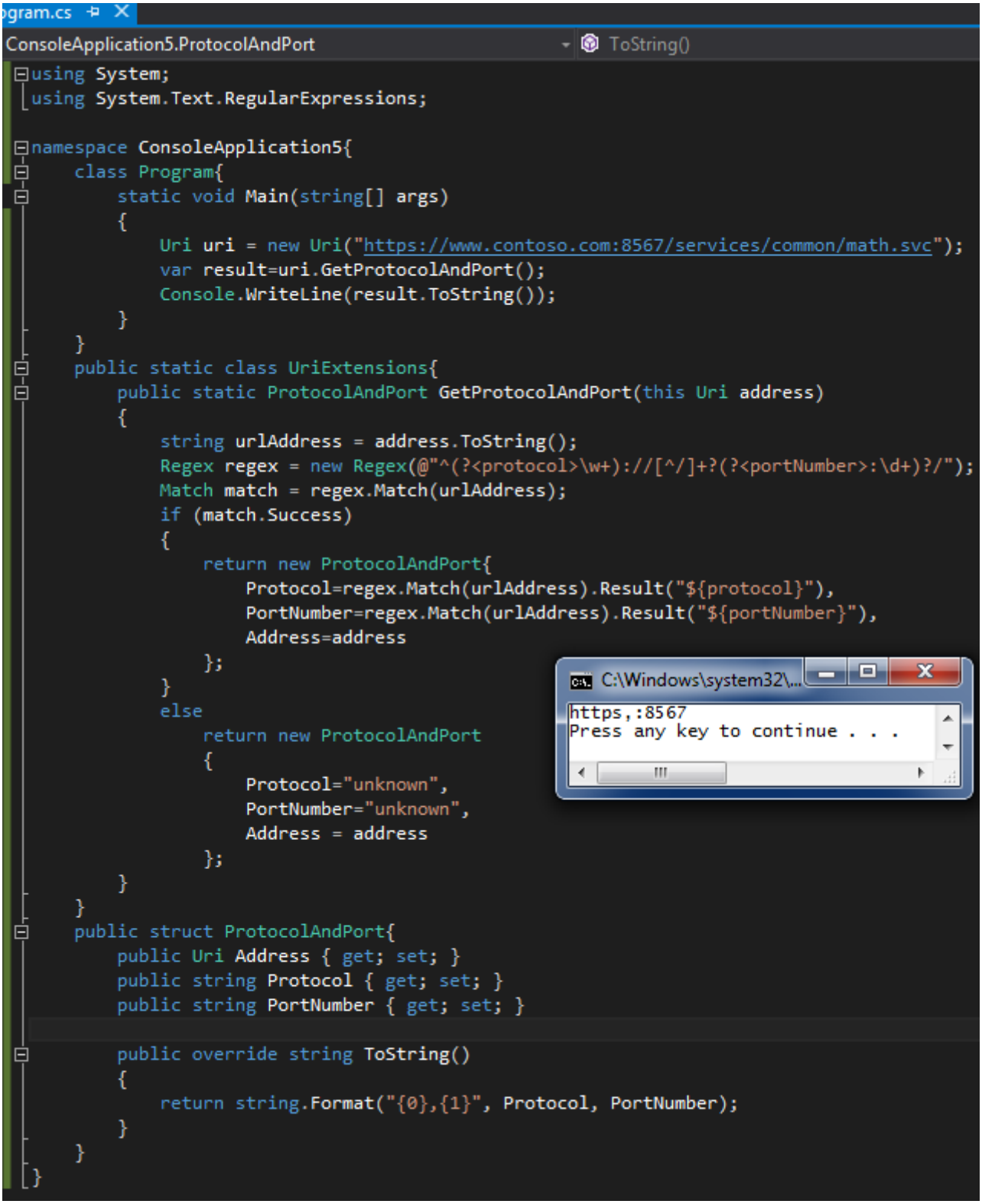
```
1       1       2       3       5       8       13      21      34      55
20365011074     32951280099     53316291173     86267571272     139583862445
Press any key to continue . . .
```

Tek Fotoluk İpucu–66–Protokol ve Port Numarasını Bulmak

Pazar, 23 Eylül 2012 09:00

Extension Methods, Regex, Tek Fotoluk Ipucu, Uri, Struct, Port, Protocol

Merhaba Arkadaşlar,
Elimizde bir Uri nesne örneği olduğunu varsayalım. Bu Uri adres bilgisinden de port numarası ve protocol bilgisine ulaşmak istiyorsunuz. Aslında string tabanlı fonksiyonellikler ile bu iş gerçekleştirilebilir ama Regex tipini işin içerisine katar ve bir de Extension Method haline getirirsek tadından yinmez



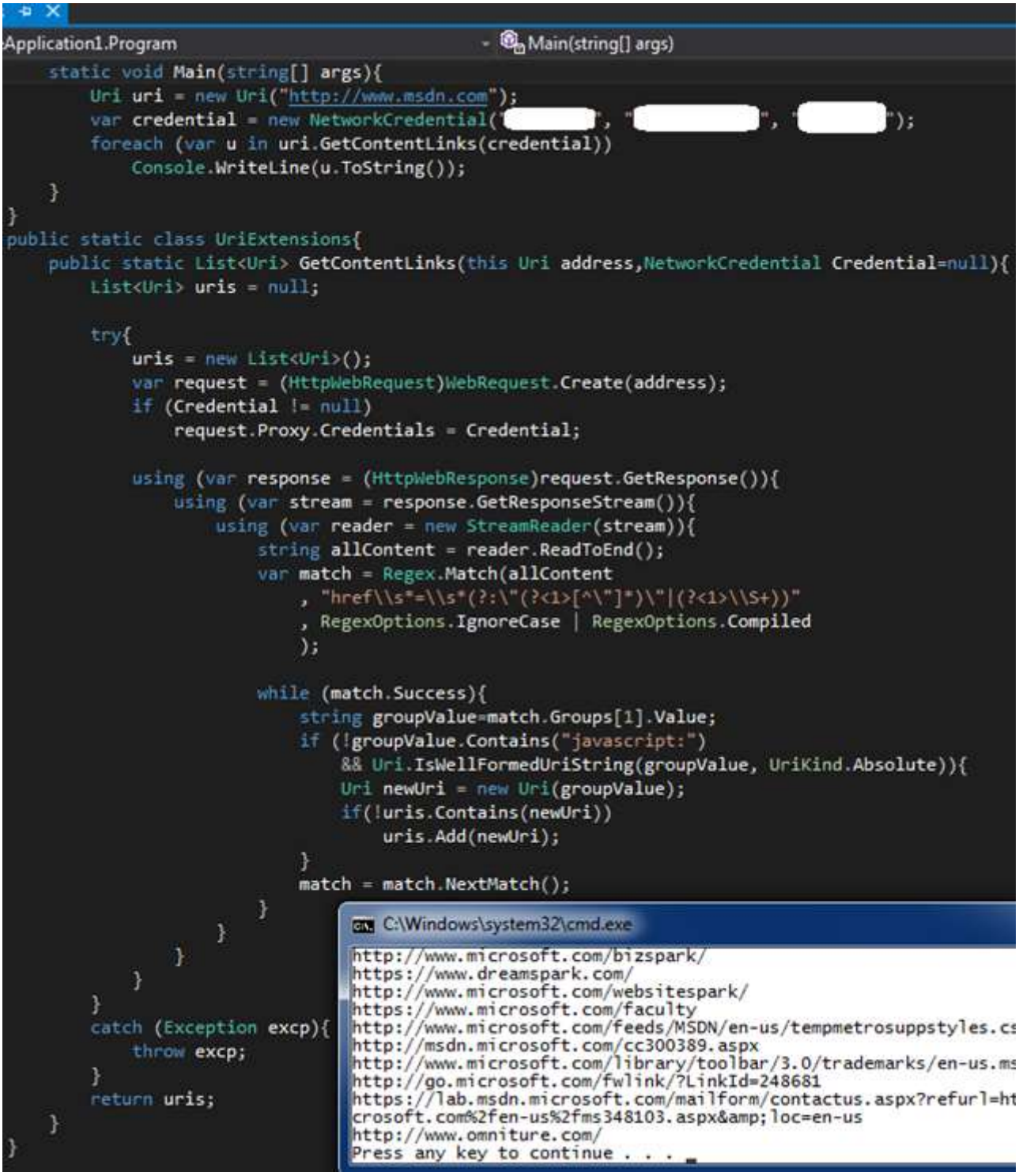
Bir başka tek fotoluk ip ucunda görüşmek dileğiyle.

Tek Fotoluk İpucu–65–Bir Web Sayfasının External Link' lerini Yakalamak

Perşembe, 20 Eylül 2012 11:15

Regex, Extension Methods, Tek Fotoluk
Ipucu, Httpwebrequest, Httpwebresponse,Streamreader, Html, Uri

Merhaba Arkadaşlar,
Diyelim ki her hangibir **Uri** tipinin işaret ettiği **Web** içeriğinde yer alan **a href=** takılarını yakalamak ve bir listeye doldurmak istediniz. Napardınız? Yoksa aşağıdaki gibi bir **Extension Method** mu geliştirirdiniz?

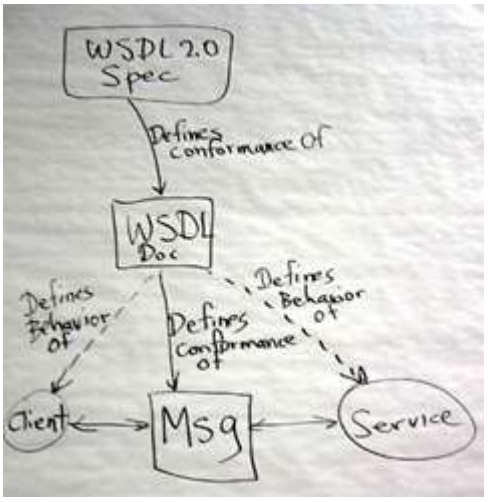


Sanırım Regex ifadesini farklı **desenler(Pattern)** ile denediğiniz de çok daha fazla bilgiye sahip olabildiğinizi görebilirsiniz 😉

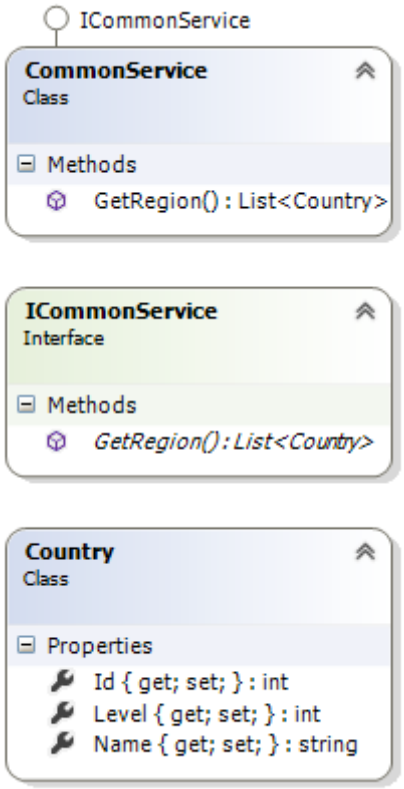
WCF 4.5–SingleWSDL

Salı, 18 Eylül 2012 14:50

Wcf, Windows Communication Foundation, Wcf 4.5, Wsdl, Singlewsdl



Merhaba Arkadaşlar,
Daha dün gibi hatırlıyorum. **Windows Communication Foundation 4.0** ile gelen yenilikleri inceliyor, öğrendiklerimi derhal bloğumda paylaşıyordum. Zaman ya çok hızlı akıyor ya da Microsoft zamanın önünde koşuyor Nitekim **.Net Framework 4.5** ha çıktı çıkacak derken, çoktan çıkmış da profesyonel projelerde kullanılmaya başlanmış bile.
Dolayısıyla zaman yine benim gibi aciz geliştiriciler için bir şeyleri daha öğrenme vaktidir. İşte ben de bu hevesle çok fazla makale tadında olmayan ama yeniliklerden haberdar olmamızı*(en çok da balık hafızamın unuttuklarını kayıt almamı sağlıyor)* sağlayacak bir kaç yazıya yer vermek istedim. İlk konumuz **SingleWSDL**
Windows Communication Foundation 4.5 sürümü ile birlikte gelen yeniliklerden birisi de **WSDL**içeriklerinin tekil bir dosya halinde elde edilebilme imkanıdır. **SingleWSDL** anahtar kelimesi sayesinde tek bir **WSDL(Web Service Description Language)** içeriğine, servis tarafının kullandığı tüm **veri sözleşmelerinin(Data Contract)** eklenmesi mümkündür.
Bir WSDL içeriği ile, servisin hangi operasyonları sunduğu, bu operasyonlardan dönen tiplerin ne olduğu ve bu tiplerin şema yapılarının nasıl oluşturulması gerektiği öğrenilebilir. WSDL, XML dili ile üretilen bir servis tanımlama/keşfetme protokolüdür.
Bunun ne anlama geldiğini anlamak için dilerseniz basit bir örnek üzerinden ilerlemeye çalışalım. Bu amaçla **Visual Studio 2012** ortamında bir **WCF Service Application** projesi oluşturalım. Proje içerisinde yer alan sözleşme ve diğer tipler aşağıdaki sınıf diagramında görüldüğü gibidir.



ICommonService isimli **servis sözleşmesi(Service Contract)** içerisinde yer alan **GetRegion**isimli fonksiyon, **Country** tipinden elemanlardan oluşan bir listeyi döndürmekte olan servis operasyonunu tanımlamaktadır. **Country** tipi ise basit bir **POCO** olup aşağıdaki içeriğe sahiptir.

```
namespace ContosoService
{
   public class Country
   {
      public int Level { get; set; }
      public string Name { get; set; }
      public int Id { get; set; }
   }
}
```

Servis sözleşmesi ve uygulayıcı tip içerikleri ise şu şekildedir.

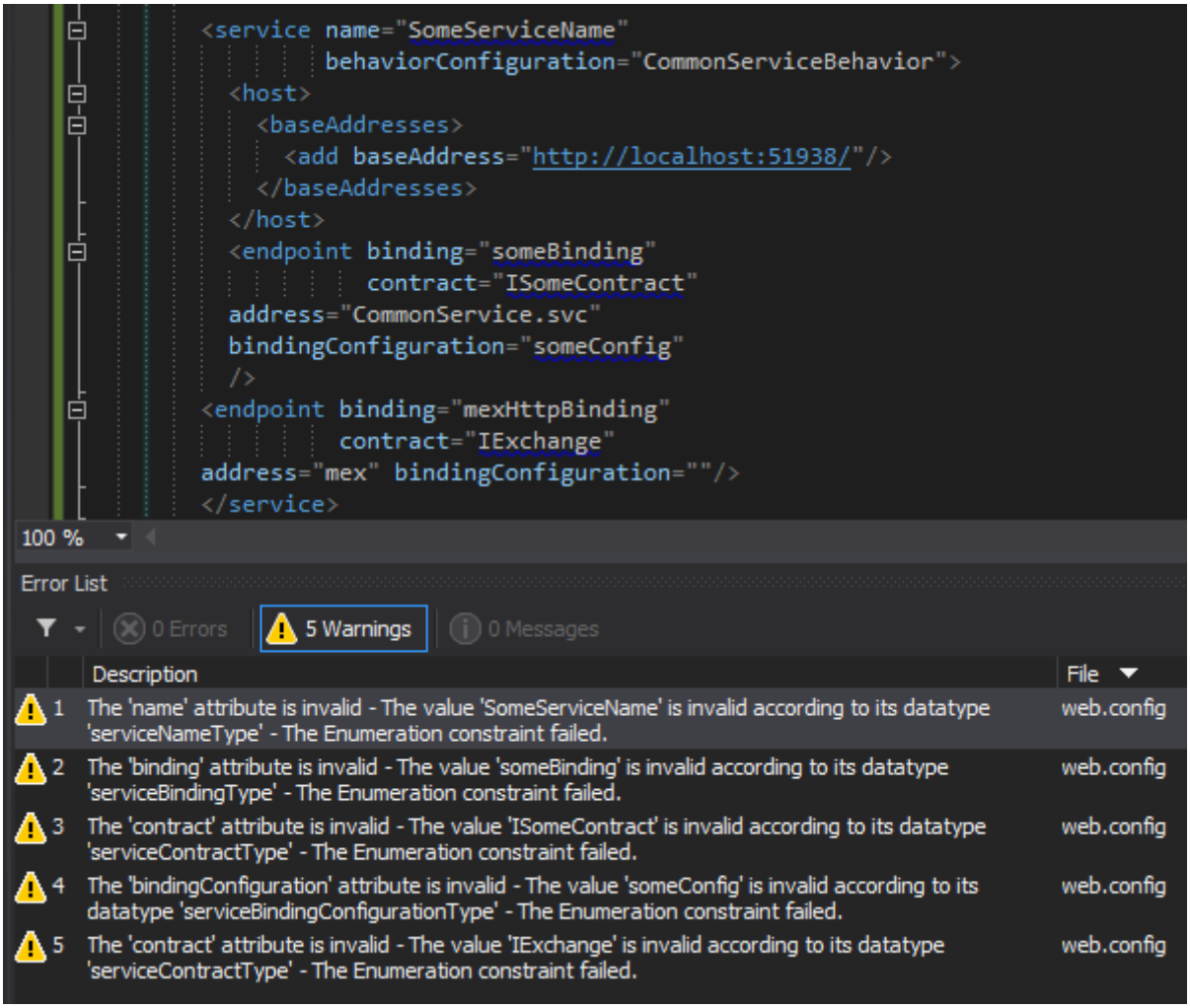
Servis sözleşmesi;

```
using System.Collections.Generic;
using System.ServiceModel;
namespace ContosoService
{
   [ServiceContract]
   public interface ICommonService
   {
      [OperationContract]
```

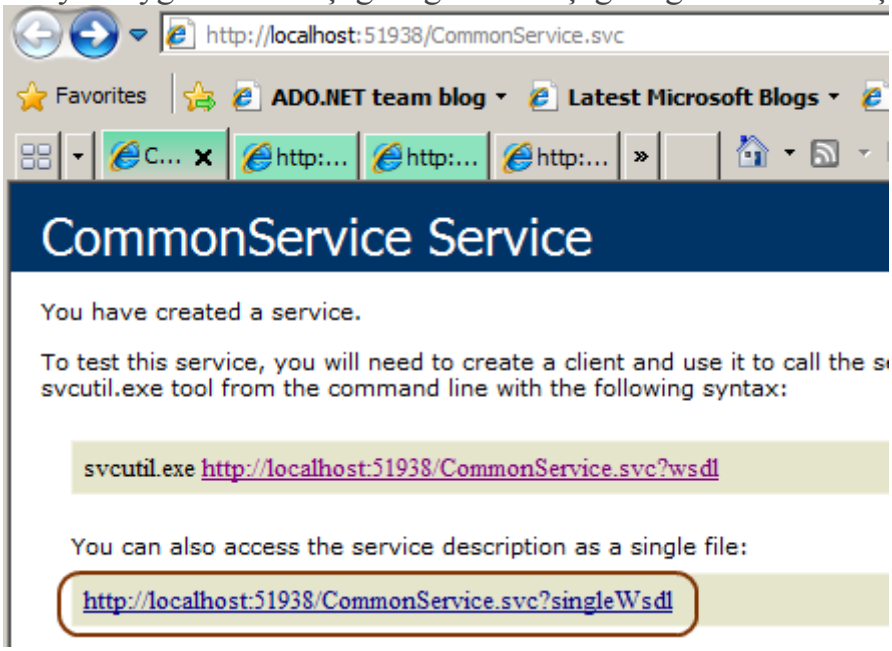
```
        List<Country> GetRegion();
    }
}
using System;
using System.Collections.Generic;
namespace ContosoService
{
    public class CommonService
        : ICommonService
    {
      public List<Country> GetRegion()
        {
            List<Country> asiaRegion = new List<Country>
            {
                new Country{Id=1,Name="Japan",Level=100},
                new Country{Id=2,Name="S. Korea",Level=200},
                new Country{Id=3,Name="Singapur",Level=300}
            };
            return asiaRegion;
        }
    }
}
```

Bonus Yenilik:

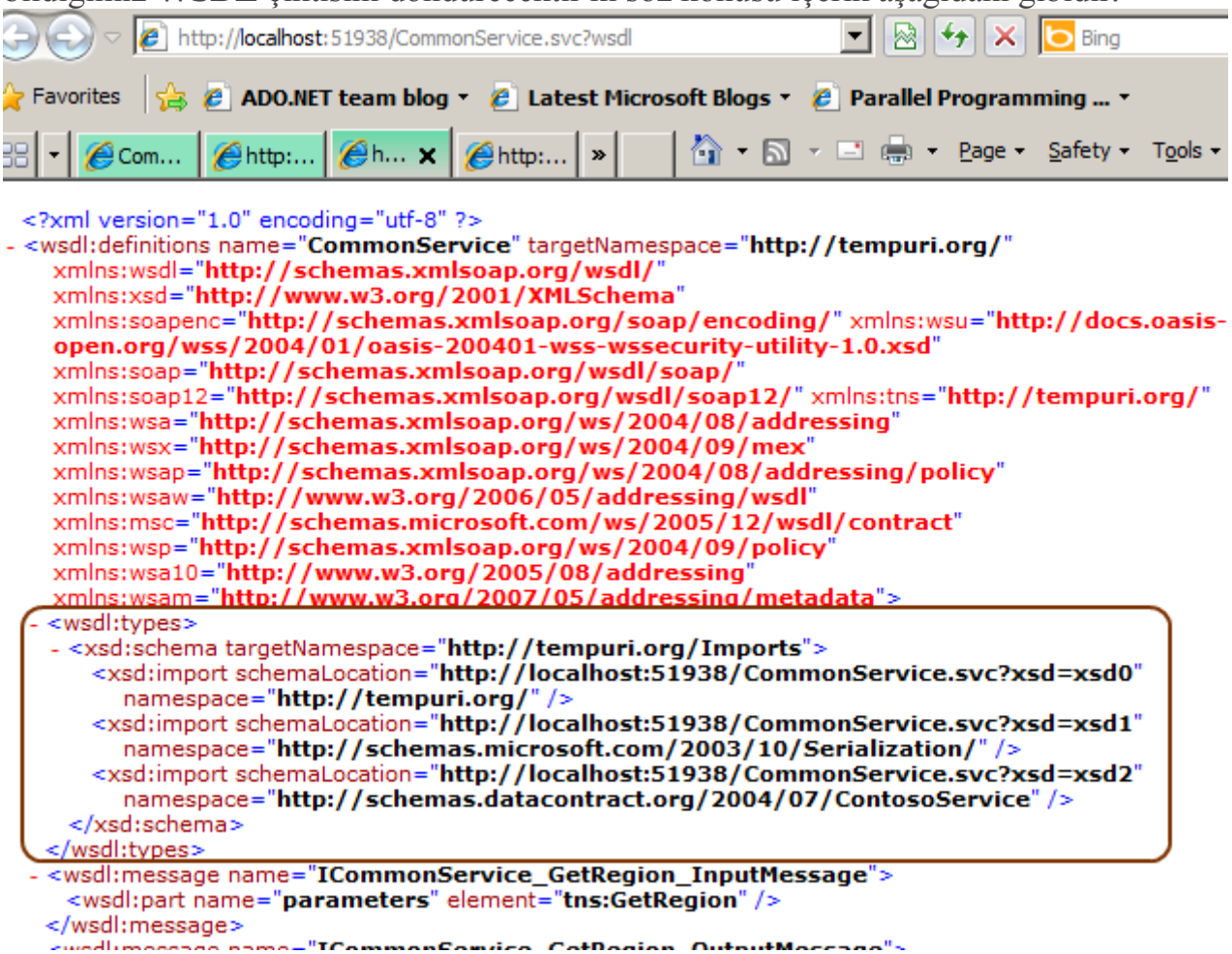
WCF 4.5 konfigurasyon içeriklerinde doğrulama(Validation) artık daha etkili ve kabiliyetli. Aşağıdaki şekilde yer alan Warning mesajlarına dikkat



Amacımız **WSDL** çıktılarına bakmak olduğundan sadece **POCO(Plain Old CLR Object)** tip kullanan bir **Dummy** servis geliştirdiğimizi düşünebiliriz. Şu noktada servisi tarayıcı uygulamadan çağırdığımızda aşağıda görülen ekran çıktısı ile karşılaşırız.



Dikkat edileceği üzere **WSDL** içeriklerine iki farklı adres ile talepte bulunabiliriz. Üstte yer alan**http://localhost:51938/CommonService.svc?wsdl** adresinden yapılan standart sorgu, bildiğimiz **WSDL** çıktısını döndürecektir ki söz konusu içerik aşağıdaki gibidir.

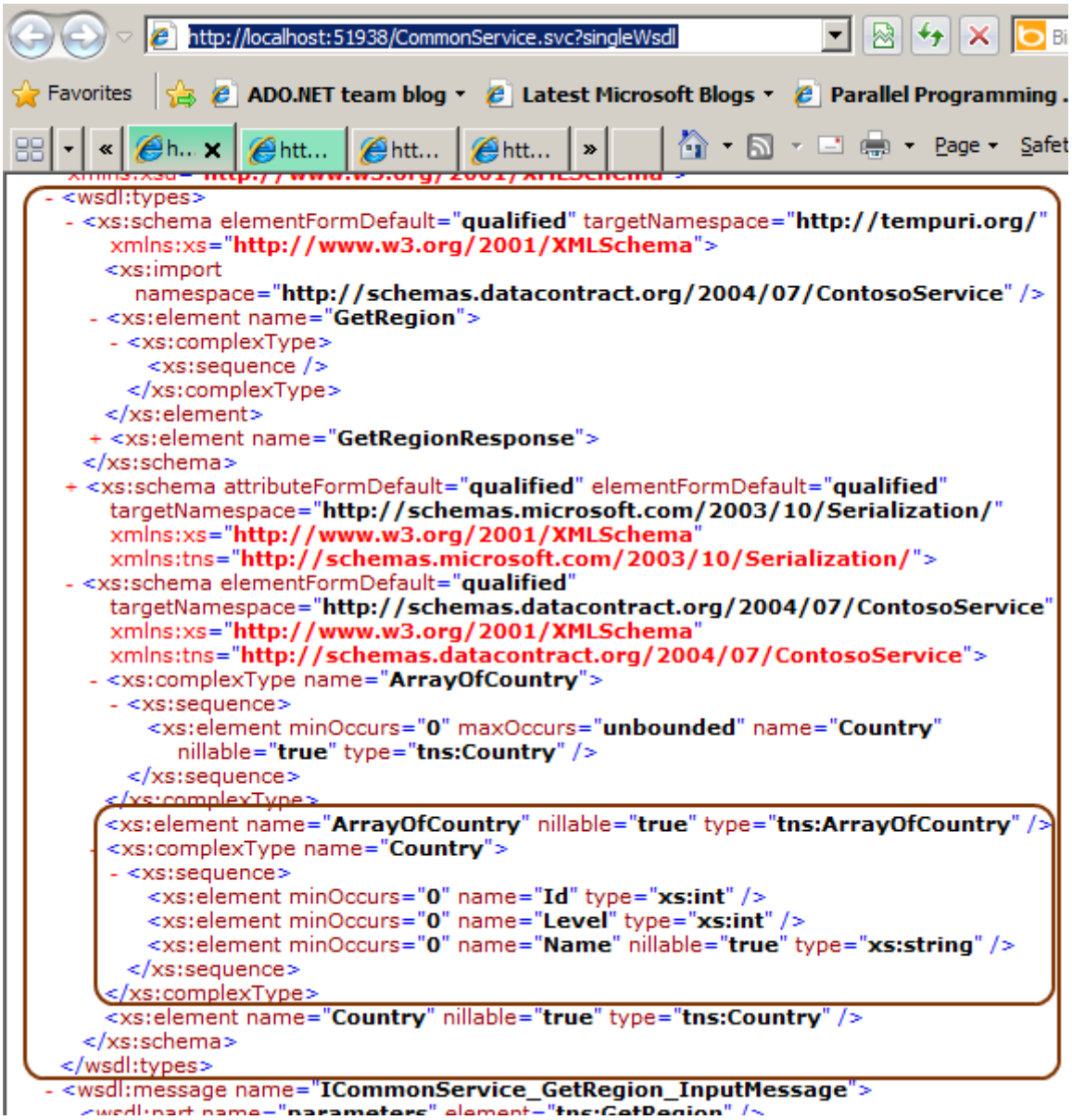


Şimdi burada duralım. Servisimiz bilindiği üzere **Country** tipi ile çalışacak şekilde tasarlanmıştır. Dolayısıyla istemci tarafı için gerekli proxy üretimi sırasında bu tipinde karşı tarafta oluşturulması gerekir. Bunun içinse ilgili tip tanımlamalarının **WSDL** dökümanında bulunması şarttır ki, **svcutil**aracı veya **Add Service Reference** seçeneği hangi şemaya bakarak nasıl bir sınıf üreteceğini bilsin.

Yukarıdaki ekran görüntüsüne bakıldığında **XSD(Xml Schema Definition)** tanımlamaları için harici referanslar verildiği de görülmektedir. Söz gelimi **http://localhost:51938/CommonService.svc?xsd=xsd2** talebi gerçekleştirildiğinde aşağıdaki ekran görüntüsünde yer alan çıktıya ulaşılır.


```
<?xml version="1.0" encoding="utf-8" ?>
- <xs:schema elementFormDefault="qualified"
    targetNamespace="http://schemas.datacontract.org/2004/07/ContosoService"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tns="http://schemas.datacontract.org/2004/07/ContosoService">
  - <xs:complexType name="ArrayOfCountry">
    - <xs:sequence>
        <xs:element minOccurs="0" maxOccurs="unbounded" name="Country" nillable="true"
          type="tns:Country" />
      </xs:sequence>
    </xs:complexType>
    <xs:element name="ArrayOfCountry" nillable="true" type="tns:ArrayOfCountry" />
  - <xs:complexType name="Country">
    - <xs:sequence>
        <xs:element minOccurs="0" name="Id" type="xs:int" />
        <xs:element minOccurs="0" name="Level" type="xs:int" />
        <xs:element minOccurs="0" name="Name" nillable="true" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
    <xs:element name="Country" nillable="true" type="tns:Country" />
  </xs:schema>
```

Fark edileceği üzere bu **XSD** içeriğinde **Country** tipi ve **Country** tipinden oluşan dizi tanımı**(ArrayOfCountry)** söz konusudur. Bu şema da **Country** tipinin içerdiği**özellikler(Properties)** ve bu özelliklerin tipleri yer almaktadır.
Şimdi de **http://localhost:51938/CommonService.svc?singleWsdl** adresi için yapılan talebin sonuçlarına bakalım.



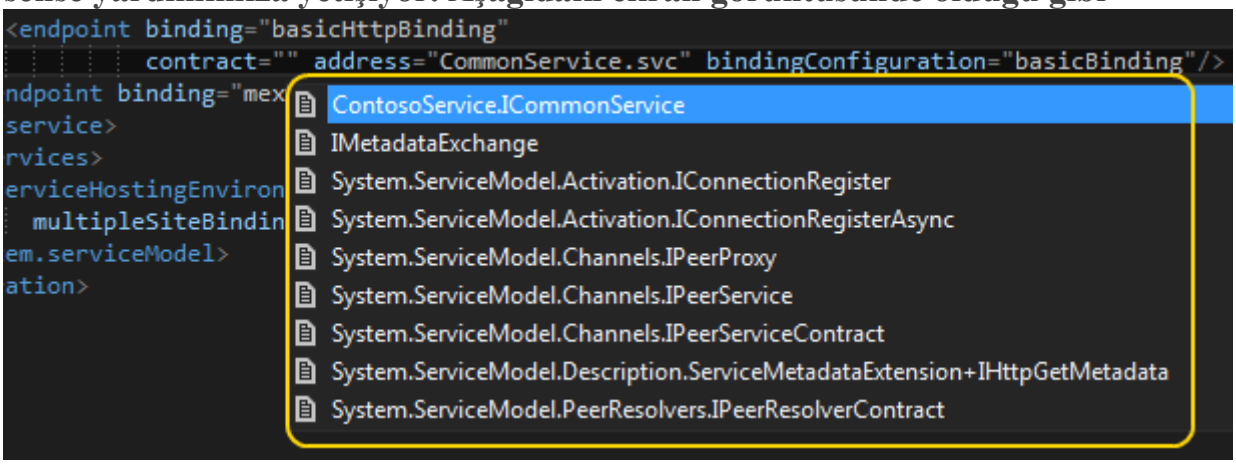
Görüldüğü gibi servis tarafının kullandığı **veri sözleşmesi(Data Contract)** ve gerekli diğer tip tanımlamaları aynı **WSDL** dökümanı içerisine bir element ağacı olarak entegre edilmiştir. Bir başka deyişle **XSD** kaynaklarının harici **URL** adresleri üzerinden referans edilmediği, bunun yerine aynı döküman içerisine enjekte edildikleri görülmektedir.

WCF 4.5 tarafında gelmiş olan bir kaç yenilik daha bulunmaktadır. Genelde çok küçük ve pek de göze çarpmayacak yenilikler olmasına rağmen, önemlidirler. Söz gelimi configurasyon dosyası içerikleri için **doğrulama(Validation)** kabiliyetleri arttırılmış, **IIS** için otomatik **HTTPS end point**tanımlaması getirilmiş, **WebSockets** üzerinden haberleşme imkanı sunulmuş vb…Bu tip yenilikleri kısa kısa da olsa paylaşmaya çalışıyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Bonus Yenilik :

WCF konfigurasyon dosyası üzerinde artık daha fazla hakimiyetimiz var. Eğer sizlerde benim gibi WCF Service Configuration Editor yerine config dosyası

içerisinde ilerleyenlerdenseniz özellikle intelli-sense desteğinin eksik olmasından yakınanlardansınızdır.
Ancak WCF 4.5 ile bu konuda önemli avantajlarımız var. Örneğin hatırlayamadığımız veya içeride hangi sözleşme var bilmediğimiz durumlarda intelli-sense yardımımıza yetişiyor. Aşağıdaki ekran görüntüsünde olduğu gibi



ContosoService.zip (23,61 kb)

WCF 4.5 WebSockets Kullanımı [Taslak]

Perşembe, 13 Eylül 2012 00:04

Wcf 4.5, Asp.Net 4.5, Windows 8, Websockets, Windows Communication Foundation, Html 5, Http, Polling, Streaming

Merhaba Arkadaşlar,

Bazen yemek yemek için dışarı çıkar ve daha önceden gitmediğimiz bir yere oturup hiç bakmadığımız tadlara yelken açarız. Bu, bazen çok başarılı sonuçlanır ve bize büyük bir keyif verir. Bazen de yapmış olduğumuz tercihlerimiz için pişmanlık duyarız. Hatta bazı zamanlarda yerken iyi gelen o tadlar, çıkışta büyük sıkıntılara yol açabilir
İşin garip olan ve belki de heyecan verici yanı, sonuçları tam olarak kestiremediğimiz bir deneyimi yaşayacak olmamızdandır. Hatta biz ilk kez denerken, aynı tadları denemiş başkaları var ise onların tavsiyelerine de kulak verir, kimine inanır, kimine inanmayız. İşte bu günkü makalemizin konusu da, hiç tadmadığımız bir yemeğin bilemediğimiz sonuçlarına benzer nitelikte. Nitekim kodlamayı yapacak bir ortamımız var ama test yapacak bir çevre yok. Nitekim senaryomuz gereği şu anda sadece **Windows 8** platformu ve üzerinde yüklü *IIS(Internet Information Services)*tarafından desteklenen ama **.Net Framework 4.5** içerisinde yer aldığı için **Windows 7** gibi bir platform üzerinde de geliştirilebilen bir konu söz konusu. Bu noktaya nasıl geldik bir bakalım
[İzleyen makalede yazılmış olan kodlar test edilememiştir. Çalışma zamanında kuvvetle muhtemel olası hatalar olduğu aşikardır. Yazıyıyı bunu düşünerek yargılamanızı öneririm.]
Web teknolojileri hızla gelişmeye devam etmekte. Bir yıldan fazla bir süredir gündemde olan yeni maceramız ise, **WebSockets** üzerinden haberleşebilme imkanı.
Özellikle **HTML 5** standartları arasında da yer alan bu yeni oluşum*(ki Google' un Doodle ürünlerinde yer alan pek çok oyunda bu tekniğin kullanımı söz konusudur)*, **HTTP** protokolünün özellikle **Real-Time** veri akışlarındaki kısıtlamalarını, güçlüklerini ortadan kaldırır nitelikte.
Bilindiği üzere **HTTP** protokolü üzerinden gerçekleştirilmekte olan **Request-Response** tabanlı çalışma modelinde, istemcilerin göndereceği taleplere karşılık olarak sunucunun vereceği cevaplar söz konusudur. Dolayısıyla istemciler, örneğin borsa hareketliliği gibi anlık değişim gösteren içerikleri elde etmek istediklerinde, çeşitli teknikleri işin içerisine katmak zorundadırlar. Bunun için Polling adı verilen teknik sıklıkla kullanılmaktadır. İstemci belirli periyot aralıklarında sunucudan gerekli veriyi talep eder ve içeriği okur. Polling dışında bir de Streaming tekniği ile verinin çekilmesi sağlanabilir, ancak hangisi olursa olsun istemci ve sunucu arasındaki haberleşme şekli, tek**talebe(Request)** karşın, tek bir **cevap(Response)** gelecek şekilde tesis edilir.

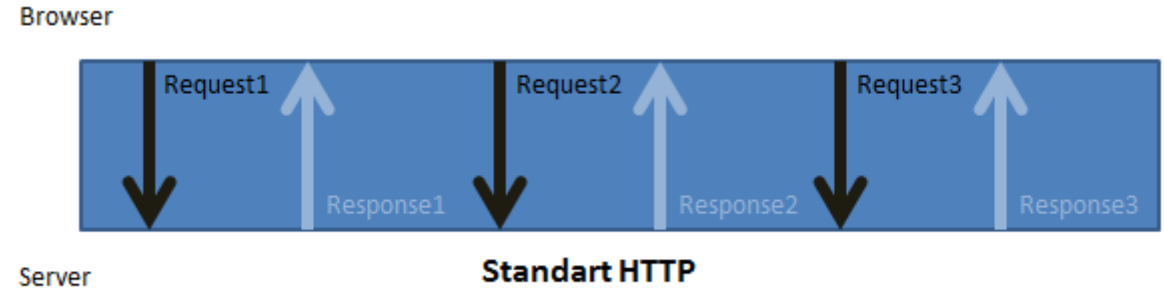
Aslında duyulan ihtiyaç özellikle anlık veriler için aynı kanal üzerinden çift yönlü iletişimi sağlayabilmektir. **Duplex Channel** iletişim olarak da adlandırabileceğimiz bu yapı, **WebSockets**'ler sayesinde **HTTP** üzerinden kolayca sağlanabilmektedir.
WebSockets' i yeni bir protokolden ziyade HTTP protokolü üzerinden kullanılabilen ve Real-Time veri transferleri için aynı kanal üzerinden çift yönlü iletişimi kolaylaştıran bir teknik olarak düşünmeyi daha doğru buluyorum.
WebSockets' in **HTTP'** nin standart çalışma modeline göre daha önemli avantajları bulunmaktadır.**HTTP** protokolüne baktığımızda aşağıdaki handikaplara sahip olduğunu görmekteyiz.

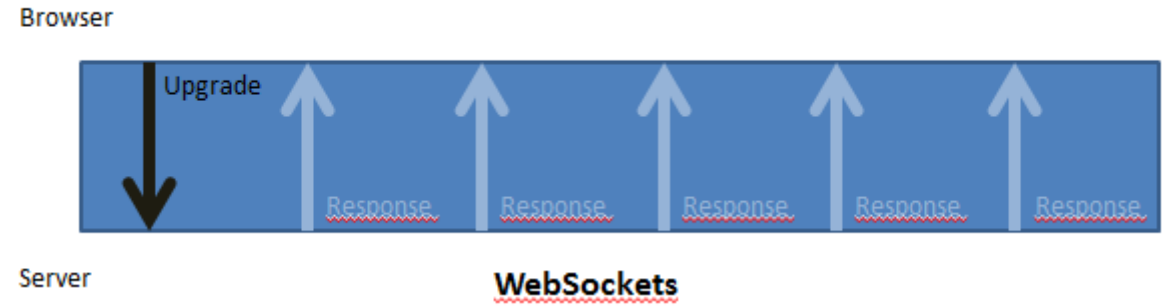
- Hızlı bir iletişim olması için tasarlanmamıştır.
- Her bir request' de mesaj başlığı(Message Header) gibi kısımlar veri ile dolduklarından paket hareket eden paket boyutları artmaktadır.
- Özellikle istemci tarafındaki uygulamaya ait verileri güncel tutabilmek için tarayıcı üzerinden sürekli olarak bir talep gönderilmesi gerekir.
- Daha kısıtlı bir Cross Domain desteği bulunmaktadır.
- Bazı vakalarda ve özellike Long Polling ve Streaming tekniklerinin uygulandığı hallerde, Proxy veya Firewall gibi ara katmanlar cevap sürelerinde gecikmelere neden olabilir.
- Diğer yandan Long Polling ve Streaming teknikleri ölçeklenebilir(Scalable) değildir.

WebSockets' in ise bu dezavantajlar karşısında sunduğu önemli avantajlar mevcuttur. Bunları şu şekilde sıralayabiliriz;

- Gerektiği kadar verinin gönderilmesi söz konusudur.
- Bandwith daha efektif olarak kullanılır.
- Cross Domain desteği sağlamaktadır.
- Firewall ve Proxy' ler üzerinden de iletişim sunabilir.
- Şu an için bazı Javascript implementasyonları haricinde Binary veri akışı desteği de sunar.
- TCP tabanlı yük dengeleyicileri(Load Balancer) içerir.

Peki bu güçlü özellikleri hangi hallerde ele almalıyız? Başlarda da belirttiğimiz üzere Real-Time veri transferi gerektiren pek çok senaryoda, **WebSockets** biçilmiş kaftan olarak görülebilir. Online Oyunlar, finansal uygulamalar, sohbet(Chat) programları, haber akışları vb...Zaten bu, **HTML 5** için de ne kadar popüler olduğunun bir göstergesidir.
Yine de ortada bazı sorunlar mevcut. Yazıyı hazırladığım tarih itibariyle örneğin Asp.Net 4.5 ile olan kullanımında ciddi kısıtlamalar var. Sadece IIS 8 üzerinde host edilen Asp.Net uygulamalarında çalışabilmekte ve bunun için, IIS tarafında WebSockets özelliğinin de etkinleştirilmiş olması gerekmektedir.
Diğer yandan, tarayıcı uygulamaların bazı versiyonlarının da bu tekniğe tam olarak destek vermediği görülmektedir. IE10, Chrome 13+, Firefox 7, Safari 5+, Opera 11+ şu an için desteklenen tarayıcı versiyonları olarak görünüyor. Ancak bu henüz tam olarak oturmamış HTML 5 protokolünün meyvelerinden birisi. Dolayısıya zaman içerisinde iyice yerleşecek ve bence vazgeçilmez teknolojilerden birisi olacaktır.
Dilerseniz **HTTP** ve **WebSockets** üzerinden yapılan iletişimler arasındaki farkı şekilsel olarak da ifade edelim.





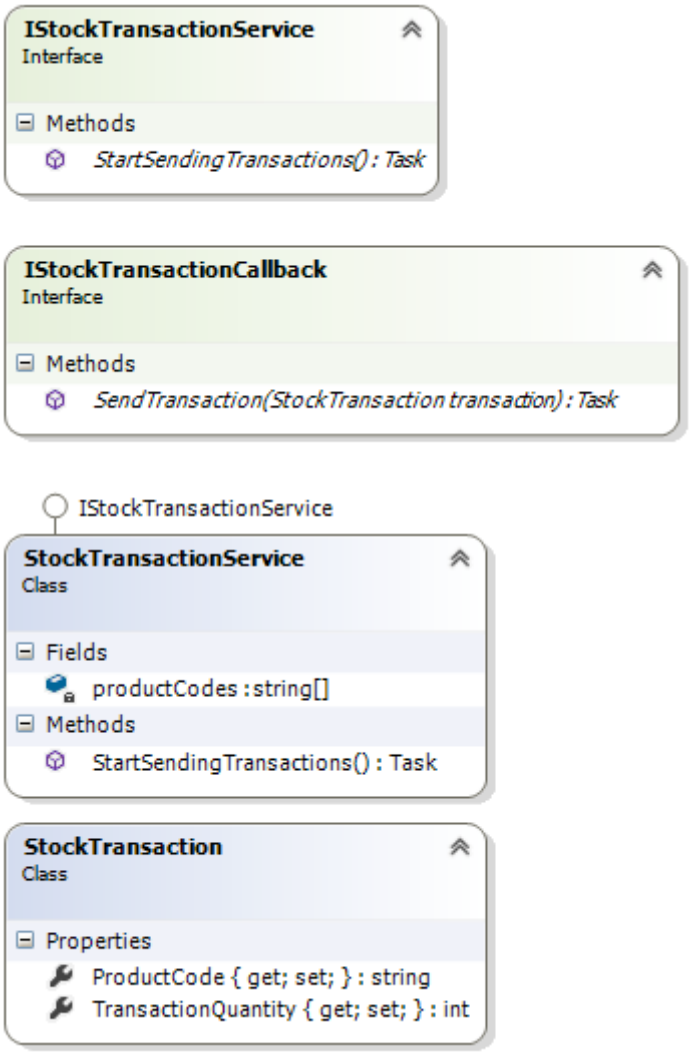
Sanırım şimdi biraz daha netleşmiştir. En azından benim kafamda şöyle bir imajı var. Bir **Request** ile kanalı açarım ve ben tekrardan bir talepte bulunana kadar, sunucu bana mesaj yollamaya ve beni bilgilendirmeye devam eder

Tabi bizim bu makalemizdeki amacımız **WebSockets** tekonolojisini derinlemesine incelemekten ziyade bunu **.Net Framework 4.5** tarafında nasıl değerlendirebileceğimizi incelemeye çalışmak. Ben**Asp.Net 4.5** yerine **WCF 4.5** tarafında konuyu irdelemeye çalışmak niyetindeyim. Bu amaçla da basit bir **How To** çalışması yapmaya gayret ediyor olacağız.

WCF 4.5 ile gelen **Binding Type'** larından birisi de, **NetHttpBinding** sınıfıdır. Özellikle, **HTTPRequest/Response** modeli dışında, **Duplex Channel** iletişimine izin veren **WebSockets** desteği nedeni ile de ön plana çıkmaktadır. Varsayılan olarak da **Binary** mesajlaşmaya da izin vermektedir. Dolayısıyla bir **WCF** servisi ile istemciler arasında **Real-Time** veri akışını sağlayacak senaryoların gerçekleştirilmesi mümkündür. Dilerseniz konuyu basit bir örnek ile ilerletelim.*(Gerçi bu biraz kör uçuşa benziyor ama en azından teoriyi anlamamıza yardımcı olacaktır kanaatindeyim)*

İlk olarak bir **WCF Service Library** projesi oluşturup içerisine aşağıdaki kod parçalarını ekleyerek işe başlayabiliriz.

Sınıf diagramı;




```
using System.ServiceModel;
using System.Threading.Tasks;
namespace StockServiceLibrary
{
   [ServiceContract(CallbackContract = typeof(IStockTransactionCallback))]
   public interface IStockTransactionService
   {
      [OperationContract(IsOneWay = true)]
      Task StartSendingTransactions();
   }
   [ServiceContract()]
   public interface IStockTransactionCallback
   {
      [OperationContract(IsOneWay = true)]
      Task SendTransaction(StockTransaction transaction);
   }
}
```

Burada görüldüğü gibi bir **Callback** tekniği söz konusudur. Malum **Duplex** kanal iletişimde bu tip bir geri bildirim sözleşmesinin de, servisin istemci tarafında tetikleme yapabilmesi için uygulanması gerektiğini biliyoruz. Asıl servis sözleşmesi olan **IStockTransactionService** geri bildirim sözleşmesi olarak **IStockTransactionCallback** arayüzünü kullanmaktadır.
Her iki sözleşme de kendi içerisinde tanımladıkları operasyonların asenkron olarak çalıştırılabilmesine olanak tanımaktadır*(async anahtar kelimeleri ile imzalandıklarına dikkat edelim)*Tabi bu asenkron işleyiş sadece servisin kendi iç çalışmasında söz konusu değildir. Geri bildirim sözleşmelerinin uygulanma kurallarından birisi olarak **StartSendingTransaction** ve**SendTransaction** isimli operasyonlar **tek yönlü(One Way)** çalışacak şekilde işaretlenmiştir.
Servisi implemente ettiğimiz **StockTransactionService** sınıfının içeriği de aşağıdaki gibidir.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.Threading.Tasks;
namespace StockServiceLibrary
{
   public class StockTransactionService
     : IStockTransactionService
   {
      static string[] productCodes = {
                        "PRD1001", "PRD45", "PRD2451",
                        "PRD2501", "PRD2301", "PRD2001",
                        "PRD1190", "PRD1007", "PRD1006",
                        "PRD1004", "PRD1023" };
      public async Task StartSendingTransactions()
      {
         var callback =
OperationContext.Current.GetCallbackChannel<IStockTransactionCallback>();
         var random = new Random();
         double price = 29.00;
         while (((IChannel)callback).State == CommunicationState.Opened)
         {
           await callback.SendTransaction(
               new StockTransaction{
                  ProductCode=productCodes[random.Next(0,productCodes.Length)],
                  TransactionQuantity=random.Next(50,5000)
               });
            price += random.NextDouble();
```

```
            await Task.Delay(1000);
          }
        }
    }
}
```

StartSendingTransactions metodu içerisinde geri bildirim sözleşmesinden yararlanılarak güncel**Context** elde edilmekte ve istemci ile olan bağlantı açık kaldığı sürece **SendTransaction**metodundan yararlanılarak rastgele **StockTransaction** örnekleri hazırlanıp **Client** uygulamaya gönderilmektedir. İstemci tarafındaki çalışma şeklini daha iyi görebilmek adına buradaki operasyon içerisinde 1 saniyelik bir gecikme işlemi de uygulanmaktadır. Ayrıca, **SendTransaction** ile **Task**tipinin **static Delay** metodlarına yapılan çağrıların **await** anahtar kelimesi eşliğinde gerçekleştirildiklerine dikkat edilmelidir.

Servis tarafında kullandığımız ve mesajlar içerisinde dolaştırdığımız **veri sözleşmesine(Data Contract)** ait **POCO(Plain Old CLR Object)** tipi tasarımı da şu şekildedir.

```
using System.Runtime.Serialization;
namespace StockServiceLibrary
{
    [DataContract]
    public class StockTransaction
    {
        [DataMember]
        public string ProductCode { get; set; }
        [DataMember]
        public int TransactionQuantity { get; set; }
    }
}
```

Sadece örnek olması açısından içerisinde ürün kodu ve işleme alınan mitar bilgisi değerlendirilmektedir. Servisin rastgele üreteceği işlemlere ait bilgileri bağlı olan stemciye/istemciler göndereceği beklenmektedir.

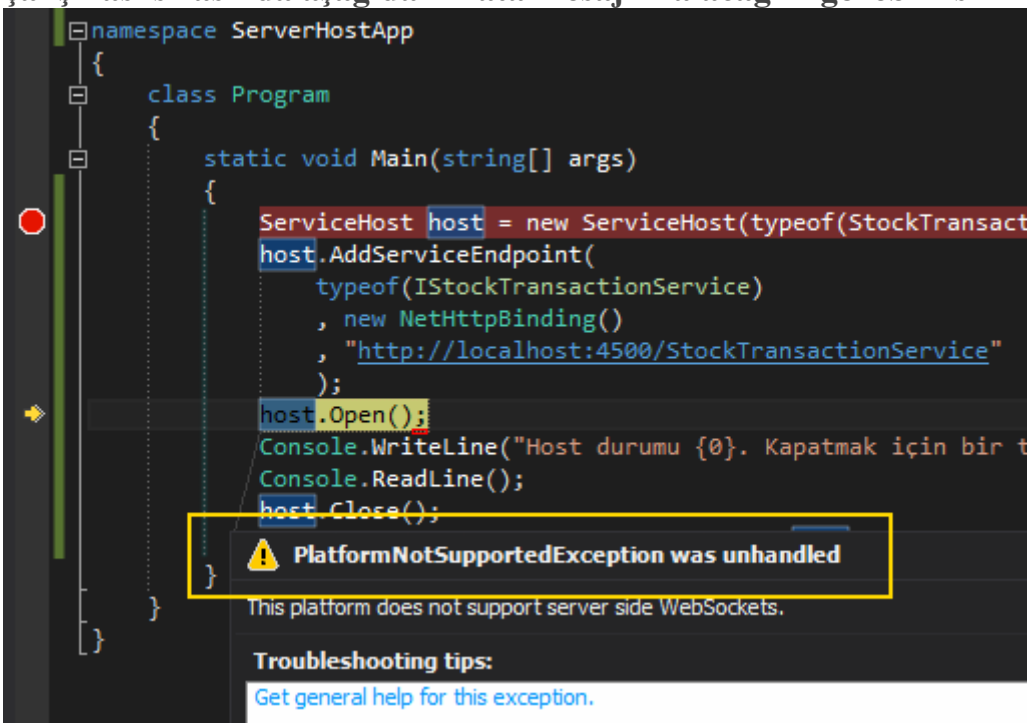
Artık servis host uygulamasını yazmaya başlayabiliriz. Örnekte **IIS** yerine, **self-host** tekniğini ele almaktayız. Bu nedenle servis kütüphanesini referans eden bir **Console** uygulamasını aşağıdaki şekilde geliştirebiliriz.

```
using StockServiceLibrary;
using System;
using System.ServiceModel;
namespace ServerHostApp
{
    class Program
    {
```

```
        static void Main(string[] args)
        {
            ServiceHost host = new ServiceHost(typeof(StockTransactionService));
            host.AddServiceEndpoint(
                typeof(IStockTransactionService)
              , new NetHttpBinding()
               , "ws://localhost/StockTransactionService"
               );
            host.Open();
            Console.WriteLine("Host durumu {0}. Kapatmak için bir tuşa basınız.",host.State);
            Console.ReadLine();
            host.Close();
            Console.WriteLine("Host durumu {0}.", host.State);
        }
    }
}
```

Görüldüğü üzere **ServiceHost** nesne örneğine eklenen **ServiceEndpoint** tipi, **NetHttpBinding**bağlayıcısını kullanmaktadır. Bu nedenle **WebSockets** desteğini içermektedir.

Eğer uygulamayı Windows 7 tabanlı bir sistemde çalıştırırsanız host uygulamanın çalışması sırasında aşağıdaki hata mesajını alacağını görebilirsiniz 😢



İstemci tarafını geliştirmek için öncelikli olarak **proxy** sınıfının üretilmiş olması da gerekmektedir. Örneğimizde **Metadata Publishing** özelliğini bilinçli olarak etkinleştirmedik. Bu nedenle komut satırından **svcutil** aracını kullanarak gerekli üretimi

gerçekleştirebiliriz. **Visual Studio CommandPrompt** üzerinden bunu aşağıdaki ifadeler ile sağlayabiliriz.

/>svcutil StockServiceLibrary.dll

ve ardından

/>svcutil *.wsdl *.xsd /out:proxy.cs

Üretilen **Proxy.cs** dosyasını istemci uygulamaya ekledikten sonra **Program** sınıfının içeriğini de aşağıdaki şekilde kodladığımızı düşünelim.

```
using StockServiceLibrary;
using System;
using System.ServiceModel;
namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("İşlemlere başlamak için bir tuşa basınız.");
         Console.ReadLine();
         var context = new InstanceContext(new CallbackHandler());
         var client = new StockTransactionServiceClient(context);
        client.StartSendingTransactions();
         Console.WriteLine("Çıkmak için bir tuşa basınız.");
         Console.ReadLine();
      }
   }
   class CallbackHandler
      : IStockTransactionServiceCallback
   {
      public void SendTransaction(StockTransaction transaction)
      {
         Console.WriteLine("{0}
{1}",transaction.ProductCode,transaction.TransactionQuantity.ToString());
      }
   }
}
```

Tabi istemci tarafında bir de **WCF** çalışma zamanı için gerekli konfigurasyon içeriğine ihtiyacımız olacaktır. Yine kör uçuş yaparak bu içeriği belirleyelim

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <startup>
    <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.5" />
```

```
</startup>
<system.serviceModel>
  <bindings>
    <netHttpBinding>
      <binding name="BindingConfig">
        <webSocketSettings transportUsage="Always" />
      </binding>
    </netHttpBinding>
  </bindings>
  <client>
    <endpoint address="ws://localhost/StockTransactionService"
              binding="netHttpBinding"
              bindingConfiguration="BindingConfig"
              contract="IStockTransactionService"
              name="NetHttpBinding_IStockTransactionService" />
  </client>
 </system.serviceModel>
</configuration>
```

Mutlaka servisin host edildiği ve istemcinin talepte bulunduğu adres dikkatinizi çekmiştir. **ws** ile başlayan bir adresleme söz konusudur. İstemci tarafı da çok doğal olarak **NetHttpBinding** bağlayıcı tipini kullanacaktır. İnce bir ayar olarak da **transportUsage** niteliği **Always** olarak set edilmiştir.

Bakalım çalışma zamanında nasıl sonuçlar elde edeceğiz? Aslında nasıl sonuçlar elde edeceksiniz demem gerekiyordu. Nitekim kodu **Windows 7** platformunda yazmak zorunda kaldığım için zaten platform desteği olmadığından deneyemedim Bunu deneme şansını ilerleyen zamanlarda umarım bulurum ki o vakit sonuçları ve koddaki hataları da düzelterek size sunarım Şimdilik bu haliyle bırakmak zorundayım. Böylece geldik bir makalemizin*(Makale adayımızın)* daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Not : *Bu arada .Net 4.5 deki WebSockets ler için Windows 8 dışında bir platform desteği olmayacağı söylenmekte. http://forums.asp.net/t/1732788.aspx/1 adresindeki tartışma oldukça taze ve bu konu üzerine yapılmış. Tabi burada dikkat edilecek olursa Kaazing WebSockets Gateway aracından yararlanılabileceği söylenmekte. Ben şu an için bunu denemedim ama siz deneyebilirsiniz*

HowTo_WCFandWebSockets_.zip (107,55 kb) [Her ihtimale karşın oynayabilmeniz için örneği ilave ettim]

Tek Fotoluk İpucu 64 – Assembly Adresinden Object Üretmek

Pazartesi, 10 Eylül 2012 08:32

Reflection, Extension Methods, Linq, Tek Fotoluk Ipucu

Merhaba Arkadaşlar,
Bazen reflection tekniklerini kullanarak harici assembly' lar içerisinden bulduğumuz tiplerin örneklerini ürettirme ihtiyacı duyabiliriz. Bunun için kullanabileceğimiz pek çok yol vardır aslında. Örneğin tipin bulunduğu Assembly dosya adresini tutan bir string değişken üzerinden dahi istenilen nesne örneğinin üretilmesini sağlayabiliriz. Nasıl mı?

Program.cs

Program.cs

UsefulExtensions.Program

Main(string[] args)

```
using System;
using System.IO;
using System.Linq;
using System.Reflection;

namespace UsefulExtensions
{
    class Program
    {
        static void Main(string[] args)
        {
            string systemData = @"C:\Windows\Microsoft.NET
                                  \Framework\v4.0.30319\System.Data.dll";
            object tbl = systemData.CreateObject("DataTable", "Employees");
            if(tbl!=null)
                Console.WriteLine(tbl.ToString());
        }
    }
    public static class Extensions
    {
        public static object CreateObject(
            this string assemblyPath
            , string className
            , params object[] arguments)
        {
            object instance = null;
            if (File.Exists(assemblyPath))
            {
                Assembly assembly = Assembly.LoadFrom(assemblyPath);
                Type createdType = assembly
                    .GetTypes()
                    .Where(t => t.Name == className)
                    .FirstOrDefault();
                if(createdType!=null)
                    instance = Activator.CreateInstance(createdType, arguments);
            }
            return instance;
        }
    }
}
```

C:\Windows\system32\c...

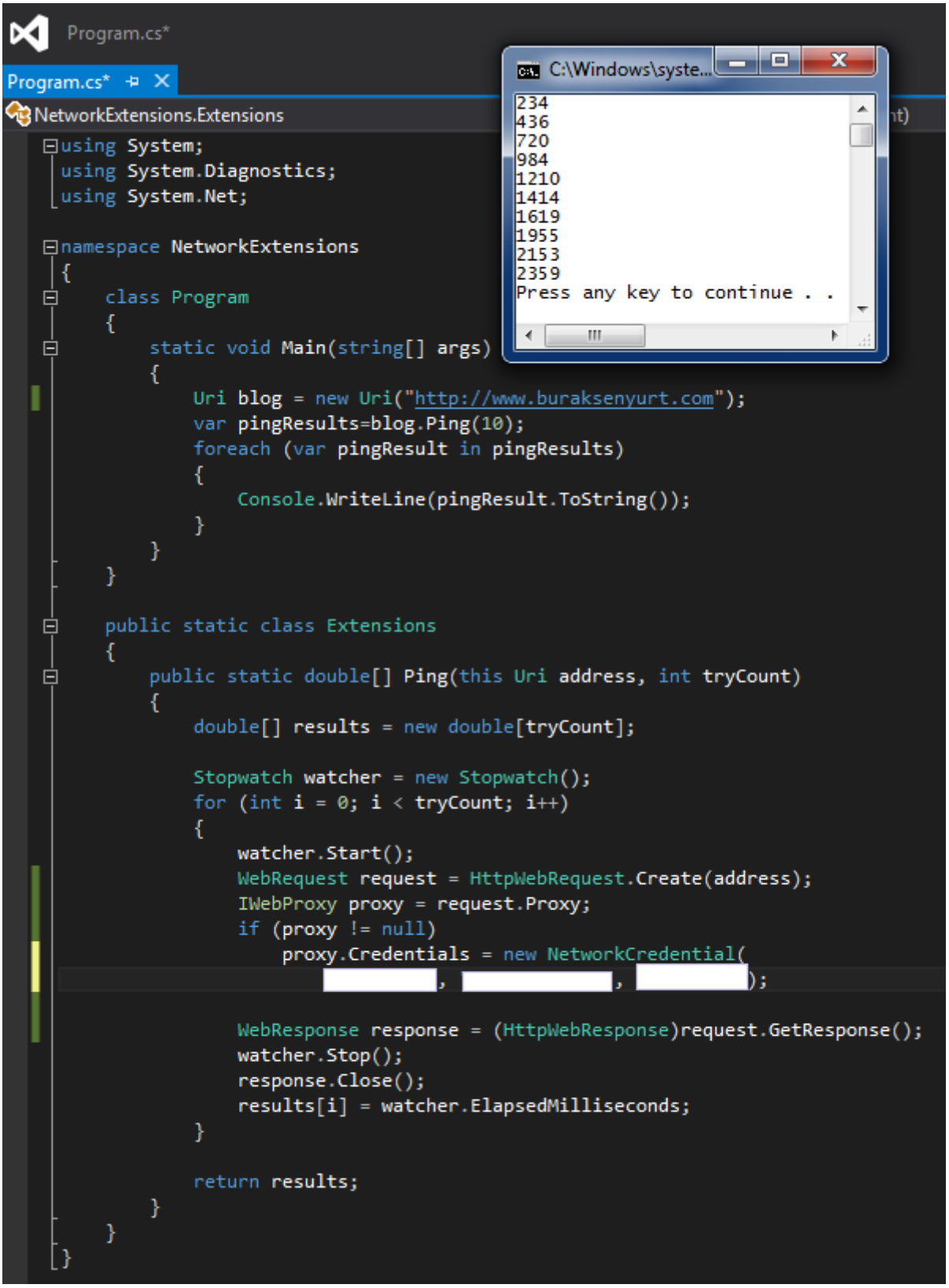
```
Employees
Press any key to continue . . .
```

Tek Fotoluk İpucu 63–Uri Üzerinden Ping Sürelerine Bakmak

Pazartesi, 3 Eylül 2012 07:50

Uri, Extension Methods, Httpwebrequest, Httpwebresponse, Ping, Web Request,Web Response, Stopwatch

Merhaba Arkadaşlar,
Hani bazen komut satırından bir URL adresine talep gönderip cevap sürelerine bakarız. Peki kod tarafından bu işi nasıl taklit edebiliriz? Örneğin Uri tipine bir Extension Method dahil etsek nasıl olur? Buyrun öyleyse



NetworkCredential parametreleri sırasıya Username, Password ve Domain olarak set edilir. Eğer proxy kullanılıyorsa.

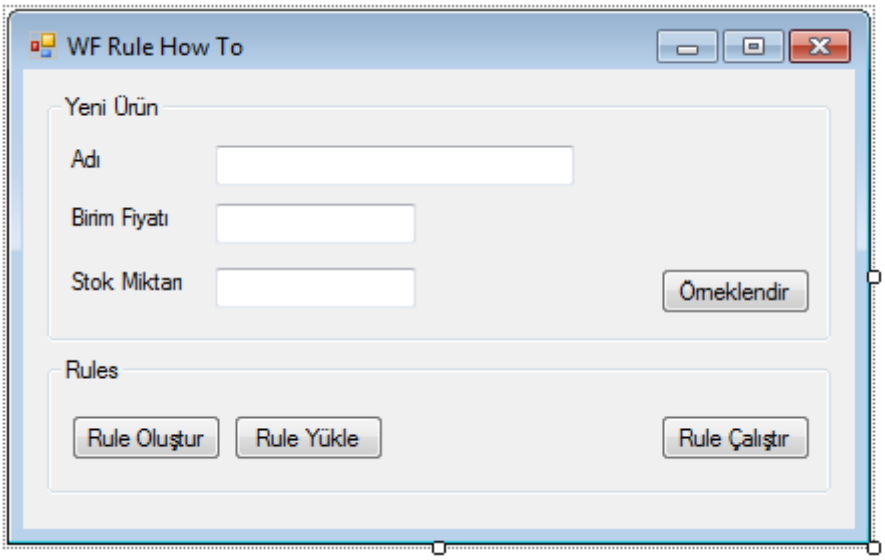
Bir başka ip ucunda görüşmek dileğiyle.

WF Rule Engine' i Dışarıdan Kullanmak

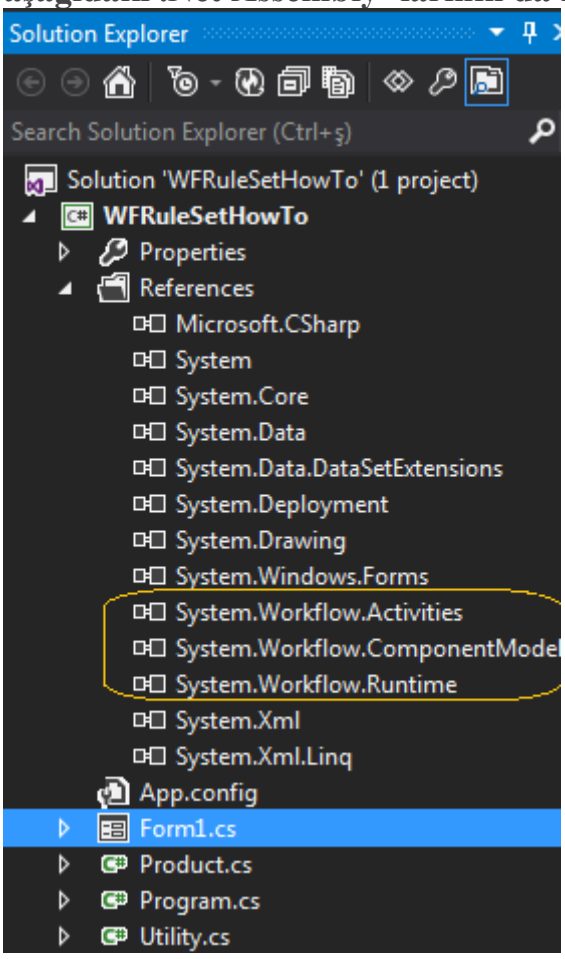
Pazartesi, 3 Eylül 2012 07:45

Rule Engine, Workflow Foundation, Workflow Foundation 4.0, Wf Rule Engine, Rule Set, Ruleset, Windows Forms

Merhaba Arkadaşlar,
Kurallar, kurallar, kurallar! Hayatın hemen her noktasında karşımıza tonlarca kural çıkar. Tabi mevzumuz kuralların zorlayıcılığı ve saire değil, kuralların ihlal edilmesi veya uyulması halinde gerçekleşen aksiyonların neler olduğu ile ilgilidir. Ki bu düşünce tarzı aslına bakarsanız iş dünyasının çekirdek süreçlerinden tutun, en ince ve hatta uç noktalarına kadar yayılır.
Hangi sektörde olunursa olunsun, işler ister kağıt üstünde, ister elektronik ortamda yürüsün, iş süreçleri kendi içerisinde tanımlı bir çok kural kümesi içerir. İş bu kural kümeleri, gerekli durumlarda doğal yollarla sistemin bir parçası olarak ya da yürütme usulü ile manuel olarak devreye girerek, sürecin şekillenmesi ve bir takım aksiyonların alınması noktasında önemli rol üstlenirler.
Biz geliştiricilerde, iş akışı mantığına dayalı sistemleri tasarladığımız durumlarda bu kural kümelerinin esnekliklerine sahip olmak isteriz. Bu Biztalk, Sharepoint, TIBCO vb iş akışı modellerini içeren gelişmiş ürünlerde çoğu zaman karşımıza çıkmaktadır.
Aslına bakarsınız uzun bir süre önce Business Rule Engine ile Programlama(Biztalk Server 2006)başlıklı bir makale yazmıştım. Bu makalede ana konu **Biz Talk'** un gelişmiş **Business Rule Engine**ara birimini **BizTalk** ortamı dışında nasıl kullanabileceğimizi görmekti. O zamanlar bir **POC(Proof of Concept)** çalışması için yapmış olduğum araştırmanın sonuçlarının kayıt altına alınmış hali olan bu yazı pek çoğu gibi atıl oldu tabi
Ancak nasıl ki **Biztalk** tarafında dışarıdan kullanabileceğimiz bir **Rule Engine** bulunmaktadır, benzer şekilde **Workflow Foundation** tarafında da kural setleri tanımlayıp, uygulatabileceğimiz bir çalışma zamanı motoru *(Runtime Rule Engine)* mevcuttur. İşte bu yazımızda söz konusu **WF Rule Engine** arabirimini herhangibir .Net uygulamasında basit anlamda nasıl kullanabileceğimizi bir kaç temel adımla görmeye/anlamaya çalışıyor olacağız.
Örnek senaryomuzda **Product** isimli bir **POCO(Plain Old CLR Object)** tip üzerinden *kural tanımlanması, kuralların kayıt edilmesi, yeniden yüklenmesi* ve istenildiği zaman canlı bir **Product**örneği üzerinde *işletilmesi* gibi fonksiyonellikleri sağlıyor olacağız. İşe ilk olarak aşağıdaki basit görünüme sahip olan bir **Windows Forms** uygulaması geliştirerek başlayabiliriz.



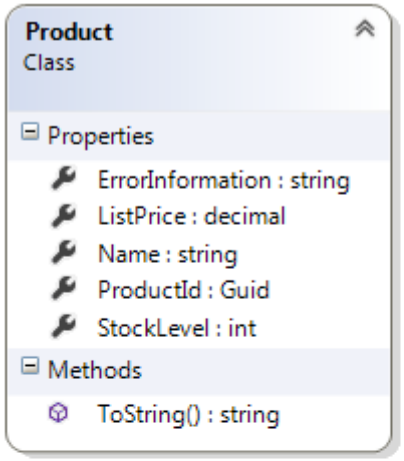
Test amaçlı olarak kullanacağımız bu **Windows Form** uygulamasında **Product** tipine ait basit kuralları tanımlayabileceğimiz, kayıt altına alabileceğimiz, tekrardan yükleyebileceğimiz ve çalıştırıp sonuçlarını görebileceğimiz işlevsellikler söz konusudur. **Pek tabi Workflow Foundation Rule Set Engine kullanılmak istendiğinden, projeye aşağıdaki .Net Assembly' larının da referans edilmesi gerekmektedir.**

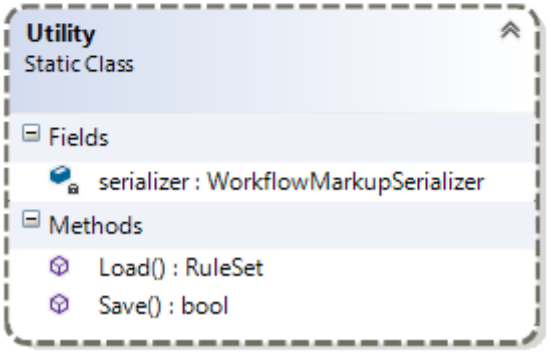


Şimdi arka planda gerekli olan kod üretimlerini gerçekleştirerek işlemlerimize devam edelim. Kural motoru için kullanacağımız **Product** tipi ve bir kuralın **XAML(eXtensible**

Application Markup Language) olarak *serileştirilmesi(Serialization)* ile tekrardan *geri yüklenmesi(DeSerialization)* için gerekli fonksiyonellikleri içeren **Utility** sınıfının kod içerikleri aşağıdaki gibidir.

Sınıf diyagramı;





Product.cs;

```
using System;
namespace WFRuleSetHowTo
{
   public class Product
   {
      public Guid ProductId { get; set; }
      public string Name { get; set; }
      public decimal ListPrice { get; set; }
      public int StockLevel { get; set; }
     public string ErrorInformation { get; set; }
      public override string ToString()
      {
         return string.Format("{0} {1} {2} {3} [{4}]"
            , ProductId
```

```
            , Name
            , ListPrice
            , StockLevel
            ,ErrorInformation
            );
        }
    }
}
```

Utility.cs;

```
using System;
using System.Workflow.Activities.Rules;
using System.Workflow.ComponentModel.Serialization;
using System.Xml;
namespace WFRuleSetHowTo
{
    public static class Utility
    {
        // Workflow RuleSet' lerin XAML bazlı serileştirilmesi/ters serileştirilmesi için gerekli nesne örneklenir
        private static WorkflowMarkupSerializer serializer = new WorkflowMarkupSerializer();

        public static RuleSet Load(string ruleSetFileName)
        {
            RuleSet ruleSet = null;
            try
            {
                // RuleSet dosyası okunmak üzere reader' a yüklenir
                XmlTextReader reader = new XmlTextReader(ruleSetFileName);
                // Ters serileştirme işlemi uygulanarak RuleSet içeriği nesnelleştirilir
                ruleSet = serializer.Deserialize(reader) as RuleSet;
                reader.Close();
            }
            catch (Exception excp)
            {
                //TODO@Burak Do Something
            }
            return ruleSet;
        }
        public static bool Save(string ruleSetFileName, RuleSet ruleset)
        {
```

```
            bool result = false;
            try
            {
               // RuleSet' i kaydetmek için bir writer oluşturulur
               XmlTextWriter writer = new XmlTextWriter(ruleSetFileName, null);
               // RuleSet ilgili dosya içerisine serileştirilir
               serializer.Serialize(writer, ruleset);
               result = true;
               writer.Flush();
               writer.Close();
            }
            catch (Exception excp)
            {
               //TODO@Burak Do Something
            }
            return result;
         }
      }
   }
```

Utility sınıfı temel olarak bir **RuleSet** örneğinin fiziki dosyaya **XAML** formatında serileştirilmesi veya tam tersi olarak **XAML** formatından geriye yüklenerek çalışma zamanında kullanılabilmesi işlevlerini barındırmaktadır. **Product** sınıfı ise örneğimizde kullanacağımız ve kural seti içerisinde ele alacağımız nesne şablonu olarak düşünülmelidir. Gelelim **Form** üzerindeki Button arkası kod parçalarına.

```
using System;
using System.IO;
using System.Windows.Forms;
using System.Workflow.Activities.Rules;
using System.Workflow.Activities.Rules.Design;
namespace WFRuleSetHowTo
{
   public partial class Form1 : Form
   {
      Product sampleProduct = null;
      RuleSet sampleRuleSet = null;
      string ruleSetFileName = Path.Combine(Environment.CurrentDirectory,
"Product.rules");
      public Form1()
      {
         InitializeComponent();
         // Kullanılacak olan RuleSet örneklenir
```

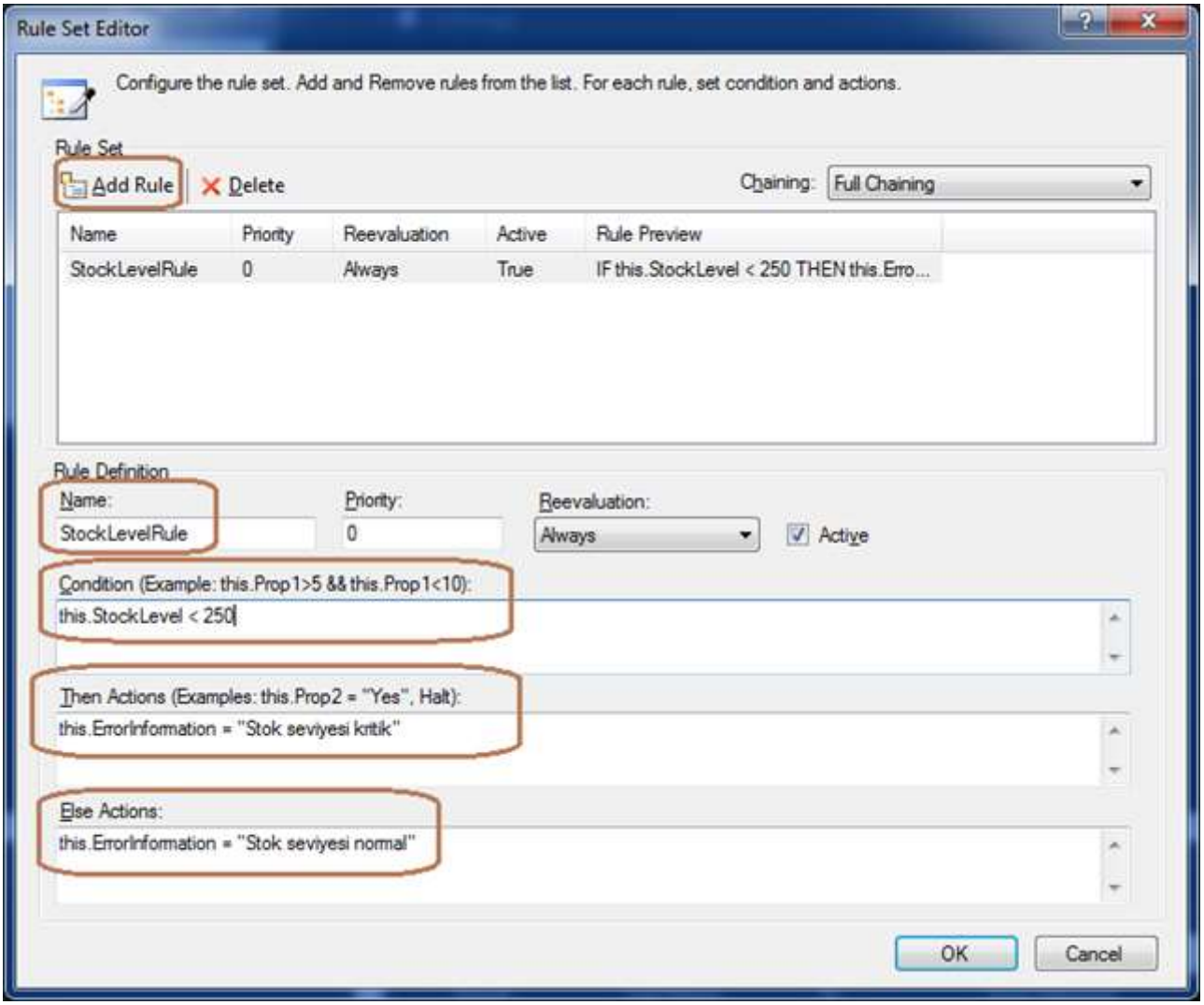
```
        sampleRuleSet = new RuleSet();
    }
    private void btnCreateProduct_Click(object sender, EventArgs e)
    {
        decimal price;
        int stockLevel;

        // RuleSet testi için örnek bir Product instance' ı oluşturulur
        sampleProduct = new Product
        {
            ProductId=Guid.NewGuid(),
            Name=!String.IsNullOrEmpty(txtProductName.Text)?txtProductName.Text:"Or
nektir",
            ListPrice=decimal.TryParse(txtProductListPrice.Text,out price)?price:1M,
            StockLevel=int.TryParse(txtStockLevel.Text,out stockLevel)?stockLevel:100
        };
        MessageBox.Show(string.Format("{0} örnek kullanım için
üretildi",sampleProduct.ToString()));
    }
    private void btnCreateRule_Click(object sender, EventArgs e)
    {
        // RuleSet' in oluşturulacağı Dialog nesnesi örneklenir
        // ilk parametre kuralın uygulanacağı nesne tipidir
        RuleSetDialog rsDialog = new RuleSetDialog(typeof(Product), null,
sampleRuleSet);
        if (rsDialog.ShowDialog() == System.Windows.Forms.DialogResult.OK)
        {
          if (Utility.Save(ruleSetFileName,rsDialog.RuleSet))
            MessageBox.Show("Rule Set başarılı bir şekilde kayıt edildi");
          else
            MessageBox.Show("İşlemlerinizi gözden geçiriniz. RuleSet kayıt
edilemedi");
        }
    }
    private void btnLoadRule_Click(object sender, EventArgs e)
    {
        sampleRuleSet = Utility.Load(ruleSetFileName);
        if (sampleRuleSet != null)
            MessageBox.Show("RuleSet başarılı bir şekilde yüklendi");
        else
```

```
            MessageBox.Show("RuleSet yüklenemedi!");
        }
        private void btnRunRule_Click(object sender, EventArgs e)
        {
            // Elimizde bir RuleSet' imiz var ise
            if (sampleRuleSet != null)
            {
                // Kuralı işletmek için gerekli doğrulama nesnesi örnek Product tipi için üretilir
                RuleValidation validation = new RuleValidation(sampleProduct.GetType(),
null);
                // Kuralı işletecek olan motor örneklenir. İlk parametre doğrulama kriterlerini
ikinci parametre ise doğrulamaya tabi olacak canlı nesne örneğini içerir
                RuleExecution engine = new RuleExecution(validation, sampleProduct);
                // Kural işletilir.
                sampleRuleSet.Execute(engine);
                MessageBox.Show(sampleProduct.ToString());
            }
        }
    }
}
```

Geliştirici bu arabirim üzerinden bir **Product** tipi için yeni **RuleSet** tanımlayabilir, kayıt altına alabilir, kayıtlı olanı yükleyebilir ve işletebilir. İşin temelinde **RuleSetDialog, RuleValidation, RuleExecution** ve **RuleSet** tipleri yer almaktadır.

RuleSet nesne örneğine ait **Execute** metodu parametre olarak gelen **RuleExecution instance'** ını baz alarak bir kural kümesi işletimini gerçekleştirmektedir. **RuleExecution**, hangi nesne örneği için ilgili kural kümesinin çalıştırılacağını, ikinci parametresi sayesinde bilmekte olup ilk parametre ile de bir doğrulama işlemini sürece dahil etmektedir. Bu doğrulama, **RuleValidation** örneğine göre bir**.Net** tipi için*(örneğimizde Product sınıfıdır)* icra edilmektedir.

RuleSetDialog tipi ile aşağıdakine benzer bir iletişim kutucuğunun*(Rule Set Editor)* çalışma zamanında açılması ve yine resimde görüldüğü gibi örnek bir kuralın tanımlanması mümkündür.



Örnekte tanımlanan **StockLevelRule** ile, herhangibir **Product** nesne örneğinin **StockLeveldeğerinin 250' nin altında olması** hali ele alınmış ve durumun **true** veya **false** olmasına göre yine o anki canlı **Product** nesne örneğinin **ErrorInformation** özelliğine bazı bilgilendirme mesajları atanmıştır. *(Çok doğal olarak burada başka atamaların yapılması da söz konusu olabilir)* Tanımlanan bu kural seti serileştirilerek kayıt altına alındığında ise aşağıdaki **XAML** içeriğinin üretildiği görülür.

<**RuleSet** Description="{p1:Null}" Name="{p1:Null}" ChainingBehavior="Full" xmlns:p1="http://schemas.microsoft.com/winfx/2006/xaml"xmlns="http://schemas.microsoft.com/winfx/2006/xaml/workflow">
  <RuleSet.Rules>
    <Rule Priority="0" ReevaluationBehavior="Always" Description="{p1:Null}" Active="True" Name="StockLevelRule">
      <Rule.Condition>
        <RuleExpressionCondition Name="{p1:Null}">
          <RuleExpressionCondition.Expression>
            <ns0:CodeBinaryOperatorExpression Operator="LessThan"

```
xmlns:ns0="clr-namespace:System.CodeDom;Assembly=System, Version=4.0.0.0,
Culture=neutral, PublicKeyToken=b77a5c561934e089">
                    <ns0:CodeBinaryOperatorExpression.Right>
                      <ns0:CodePrimitiveExpression>
                        <ns0:CodePrimitiveExpression.Value>
                          <ns1:Int32 xmlns:ns1="clr-
namespace:System;Assembly=mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089">250</ns1:Int32>
                        </ns0:CodePrimitiveExpression.Value>
                      </ns0:CodePrimitiveExpression>
                    </ns0:CodeBinaryOperatorExpression.Right>
                    <ns0:CodeBinaryOperatorExpression.Left>
                      <ns0:CodePropertyReferenceExpression
PropertyName="StockLevel">
                        <ns0:CodePropertyReferenceExpression.TargetObject>
                          <ns0:CodeThisReferenceExpression />
                        </ns0:CodePropertyReferenceExpression.TargetObject>
                      </ns0:CodePropertyReferenceExpression>
                    </ns0:CodeBinaryOperatorExpression.Left>
                  </ns0:CodeBinaryOperatorExpression>
                </RuleExpressionCondition.Expression>
              </RuleExpressionCondition>
            </Rule.Condition>
            <Rule.ThenActions>
              <RuleStatementAction>
                <RuleStatementAction.CodeDomStatement>
                  <ns0:CodeAssignStatement LinePragma="{p1:Null}" xmlns:ns0="clr-
namespace:System.CodeDom;Assembly=System, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089">
                    <ns0:CodeAssignStatement.Left>
                      <ns0:CodePropertyReferenceExpression
PropertyName="ErrorInformation">
                        <ns0:CodePropertyReferenceExpression.TargetObject>
                          <ns0:CodeThisReferenceExpression />
                        </ns0:CodePropertyReferenceExpression.TargetObject>
                      </ns0:CodePropertyReferenceExpression>
                    </ns0:CodeAssignStatement.Left>
                    <ns0:CodeAssignStatement.Right>
                      <ns0:CodePrimitiveExpression>
                        <ns0:CodePrimitiveExpression.Value>
                          <ns1:String xmlns:ns1="clr-
```

namespace:System;Assembly=mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">Stok seviyesi kritik</ns1:String>
</ns0:CodePrimitiveExpression.Value>
</ns0:CodePrimitiveExpression>
</ns0:CodeAssignStatement.Right>
</ns0:CodeAssignStatement>
</RuleStatementAction.CodeDomStatement>
</RuleStatementAction>
</Rule.ThenActions>
<Rule.ElseActions>
<RuleStatementAction>
<RuleStatementAction.CodeDomStatement>
<ns0:CodeAssignStatement LinePragma="{p1:Null}" xmlns:ns0="clr-namespace:System.CodeDom;Assembly=System, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">
<ns0:CodeAssignStatement.Left>
<ns0:CodePropertyReferenceExpression PropertyName="ErrorInformation">
<ns0:CodePropertyReferenceExpression.TargetObject>
<ns0:CodeThisReferenceExpression />
</ns0:CodePropertyReferenceExpression.TargetObject>
</ns0:CodePropertyReferenceExpression>
</ns0:CodeAssignStatement.Left>
<ns0:CodeAssignStatement.Right>
<ns0:CodePrimitiveExpression>
<ns0:CodePrimitiveExpression.Value>
<ns1:String xmlns:ns1="clr-namespace:System;Assembly=mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089">Stok seviyesi normal</ns1:String>
</ns0:CodePrimitiveExpression.Value>
</ns0:CodePrimitiveExpression>
</ns0:CodeAssignStatement.Right>
</ns0:CodeAssignStatement>
</RuleStatementAction.CodeDomStatement>
</RuleStatementAction>
</Rule.ElseActions>
</Rule>
</RuleSet.Rules>
</RuleSet>

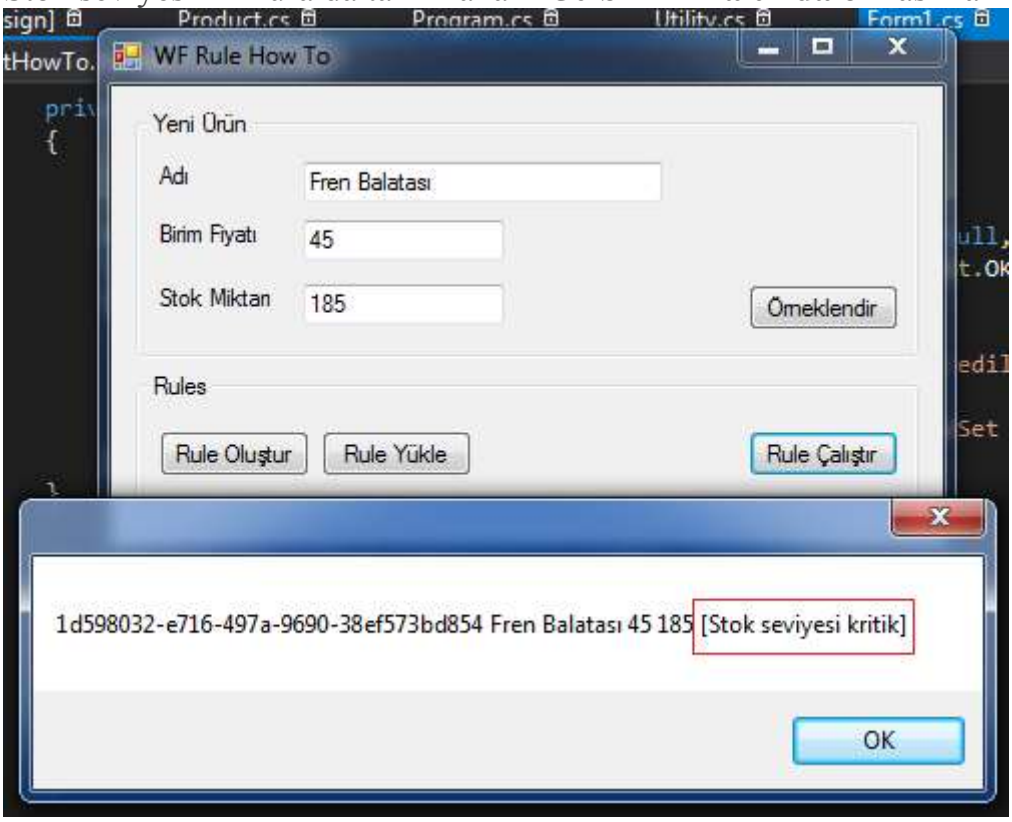
Her ne kadar bu çıktıyı gözle takip etmek zor olsa da şu noktaya dikkat edilmelidir.

XAML olarak üretilen içerik Notepad gibi basit bir metin editörü ile açılıp düzenlenebilir. Bir başka deyişle kuralların dekleratif olarak tanımlanabilmesi, güncellenmesi ve devreye alınması söz konusudur.

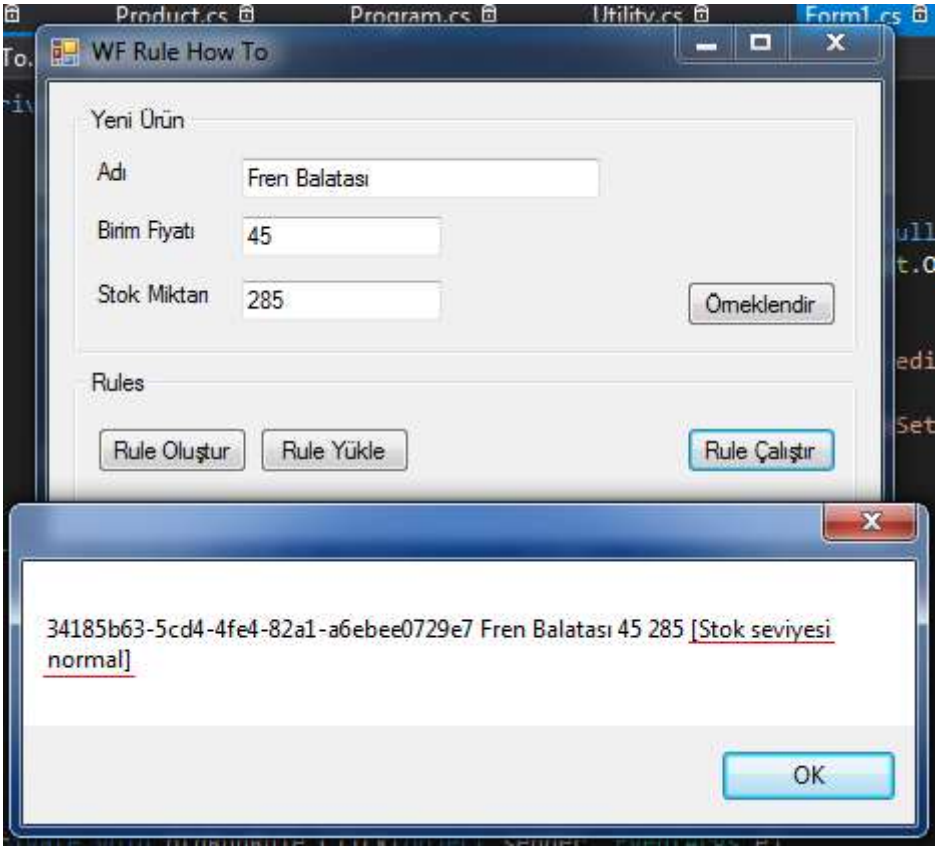
Pek tabi kayıt altına serileştirerek almış olduğumuz bu **XAML** içeriğini uygulamayı kapatsak bile tekrardan aynı veya farklı uygulamalara yükleyebilir ve işletebiliriz.

Uygulamamızda örnek bir**Product** için kural çalıştırıldığında aşağıdaki sonucun alındığı gözlemlenir.

Stok seviyesinin kuralda tanımlanan **250 birimin altında** olması halinde,



Stok seviyesinin kuralda tanımlanan **250 birimin üstünde** olması halinde,



Görüldüğü gibi **Workflow Foundation** ile birlikte gelen kural motorunun herhangibir **.Net**uygulaması üzerinden kullanılabilmesi son derece kolaydır. Hatta bu tip bir arabirim yardımıyla, iş analistlerinin çeşitli kurallar tanımlayıp kayıt altına alabilecekleri ve aslında süreç yönetim araçlarında önemli yere sahip olan bir takım depolama programlarının geliştirilmesi de kolaylaşmaktadır. Çok doğal olarak bu kurallar bir servis arkasında işletilebilirler de

WF Rule Set Editor arayüzü, kullanıcısına daha esnek bir şekilde kural tanımlayabilme ve bunları kayıt altına alarak saklayabilme imkanı sunmaktadır. XAML formatlı olarak kayıt altına alınabilen kural kümeleri(RuleSet) istenildiği zaman çalışma zamanına yüklenebilir ve tanımın ait olduğu nesne örneği/örnekleri için işletilebilir.

Rule Set Editör içerisinde birden fazla kural tanımlanabilir ve bunlar çalışma zamanında yürütülebilir.

Bu yazımızda çok basit olarak **Workfow Rule Engine** alt yapısına bir Merhaba demeye çalıştık. Kapıyı aralamak benden içeri girip yürümek ise sizden Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

[Örnek her nedense halen RC sürümünde olan Visual Studio 2012 ile geliştirilmiştir. Ancak Visual Studio 2010 ile de çalışmaktadır]

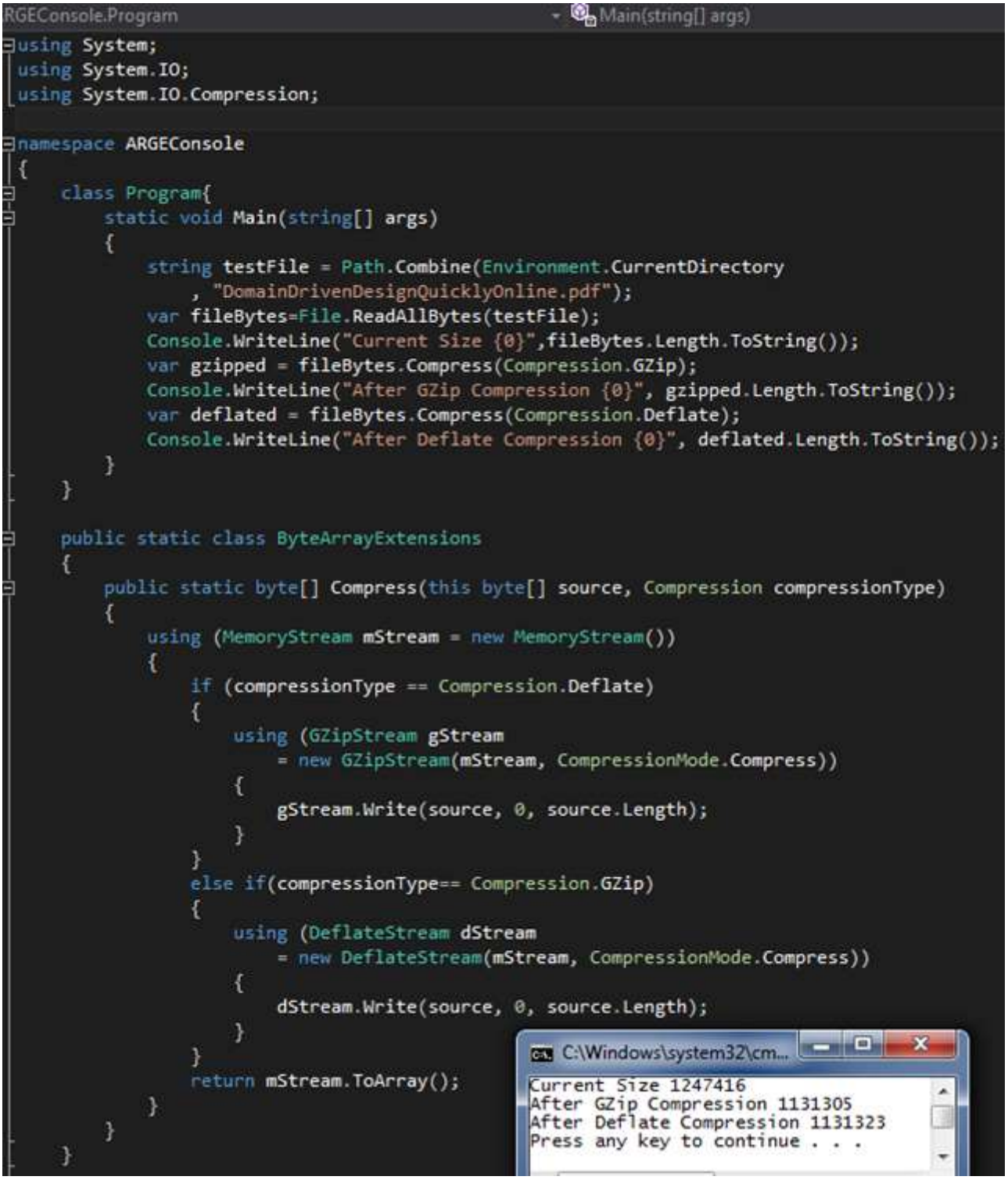
WFRuleSetHowTo.zip (60,57 kb)

Tek Fotoluk İpucu 62–Byte Array için Sıkıştırma

Perşembe, 26 Temmuz 2012 18:00 by bsenyurt

Byte, Compression, Gzip, Deflate, Gzipstream, Deflatestream, Memorystream,Stream, Compress, Extension Methods

Merhaba Arkadaşlar,
Kod içerisinde bir yerlerde öyle ya da böyle elde ettiğiniz ama boyutu azcık da olsa küçülebilse dediğiniz byte tipinden array' ler olduğunu düşünün. Kimi zaman bir dosyanın içeriği olabileceği gibi, sistem içerisinde üretilmiş bir byte dizisi bile olabilir bu. Peki söz konusu içeriği var olan GZip veya Deflate algoritmalarına göre sıkıştırmak isterseniz Aşağıdaki gibi bir Extension Method eminim ki işinize yarayacaktır.



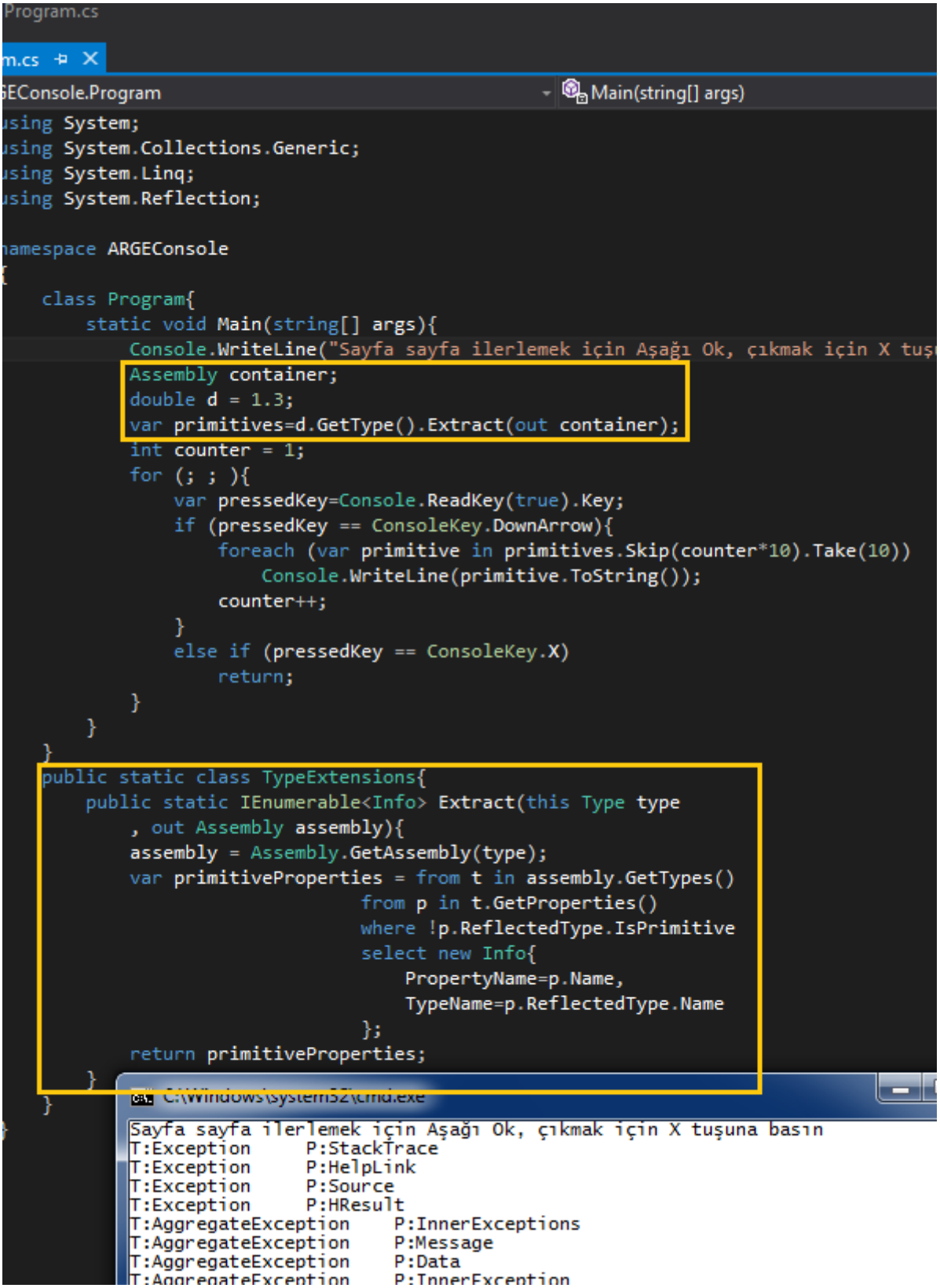
Bir başka ipucunda görüşmek dileğiyle.

Tek Fotoluk İpucu 61–Primitive Olmayan Property' leri Bulmak

Perşembe, 26 Temmuz 2012 07:51

Reflection, Linq, Isprimitive, Cts, Common Type System, .Net Types, Assembly,Extension Methods

Merhaba Arkadaşlar,
Diyelim ki bir değişkenin tipinin içerisinde yer aldığı Assembly' daki diğer tiplerin Primitive olmayan(int,double,char vb) özelliklerini bulmak gibi bir ihtiyacınız var. Nasıl bir yol izlersiniz? Kuvvetle muhtemel Reflection' dan yararlanırsınız. Hatta belki biraz da LINQ katarsınız işin içine. Ya da aklınızdan geçen tam olarak aşağıdaki gibi bir Extension Method' dur



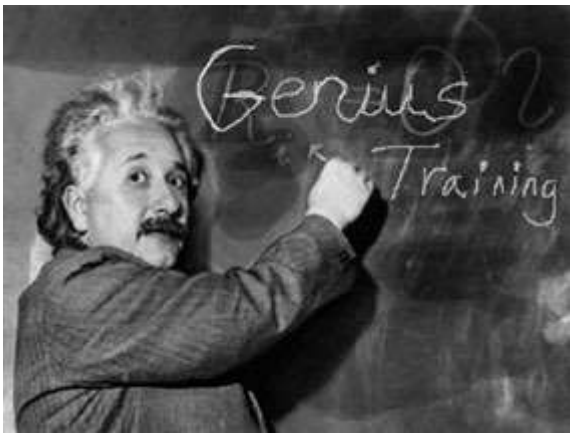
Hazır primitive tip demişken. String ve Decimal' in primitive olmadıklarını biliyor muydunuz?

Kodla Saçmalamaca

Çarşamba, 18 Temmuz 2012 17:05

C#, Xml, Programming, C# Temelleri

[İzleyen yazı Level 100 altı bir deneyimi içermekte olup üstünde kalan geliştiricileri pekala sıkabilir]



Merhaba Arkadaşlar,

Programlamaya ister yeni başlamış olun ister yıllardır bu işin içerisinde bulunun, hızlı çözüm üretmek, analitik düşünmek ve olabildiğince işe yarar parçalar çıkartmak en büyük hedeflerimizden birisi olmalıdır. Elbette yıllar içerisinde elde edinilen, kazanılan tecrübe ve bilgi birikimine bağlı olarak kendinize ait bir**geliştirme(Development)** tarzı da ister istemez oluşacak ve hatta sonrasında değiştirilemez/değiştirilmesi zor bir alışkanlık haline gelecektir.

Makbul olan pek çok geliştirici gibi ortak bazı kurallar veya standartlar üzerinde buluşabiliyor olmaktır tabiki. Şimdi diyeceksiniz ki "yazının başlığı ve içeriği arasında nasıl bir bağ kurdun be adam?". Aslında ispatlamak istediğim basit bir teori var.

Her ne kadar saçma görünen bir fikrin icrası da söz konusu olsa, developer işini kafasında veya kağıt üstünde titizce planlar, araştırır, sırayla adımlar ve hatta satranç oynarmışçasına bir kaç hamle ilerisini düşünerek kodlama yapar. Sonrasında ise…Okuyalım ve görelim

Doğruyu söylemek gerekirse bu tip felsefik söylemleri veya yaklaşımları kanıtlamak veya kabul ettirmek zordur. Hatta tepki almak çok ama çok daha kolaydır. Felsefeyi anlatabilmenin belki de en kolay yolu kendi beyninizi açıp bir işi yaparken canlı canlı kağıda dökmekten geçmektedir.

Lafı fazla uzatmadan felsefemizi örnek bir fikir ile ilişkilendirip ilerlemeye çalışalım. Örneğin geliştireceğiniz **Freelance** uygulamalarınızda sıklıkla kullandığınız ama aslında dünya bakış açısına göre çok uzun bir zaman boyunca sabit kalan belirli veri içeriklerine ihtiyacınız oldu. Ülke adları, kodları, telefon alan kodları vb…Karar verdiniz ve dediniz ki, **Ülkelerin isim, kod, ISO, telefon alan kodu bilgilerini tek bir XML kaynağında ele almak ve hatta gerektiğinde POCO olarak da dış ortama sunmak istiyorum. Ve…**

İşte gerisini beynim nasıl yorumlamış görelim

1- Ülkeler için öncelikli olarak **XML** veri kaynağı araştırılır ancak kafamızada tasarladığımız gibi tüm alanları içermediği görülebilir. Bulunan içerik her ihtimale karşı kayıt altına alınır. ***(5 dakika)***

İşin en zevkli kısımlarından birisi de ön araştırmalardır. Ancak geliştirici kendisini araştırmaya çok fazla kaptırıp, bir anda hedeflediği üretime ulaşma noktasında süre bazında sapabilir. Bu yüzden dikkatli olunmalıdır.

```
<?xml version='1.0' encoding='UTF-8'?>
<Countries>
  <Country Code="AF" ISOCode="4">Afghanistan</Country>
  <Country Code="AL" ISOCode="8">Albania</Country>
  <Country Code="DZ" ISOCode="12">Algeria</Country>
  <Country Code="AS" ISOCode="16">American Samoa</Country>
  <Country Code="AD" ISOCode="20">Andorra</Country>
  <Country Code="AO" ISOCode="24">Angola</Country>
…
</Countries>
```

2 – Telefon Alan kodlarının olmadığı görülünce, bu kez bunlara ilişkin bir **XML** veri kaynağı araştırılır ve aşağıdakine benzer bir içerik bulunur. ***(5 Dakika)***

```
<?xml version='1.0' encoding='UTF-8'?>
<TelephoneCode>
    <AF>93</AF>
    <AL>355</AL>
    <DZ>213</DZ>
    <AD>376</AD>
    <AO>244</AO>
…
</TelephoneCode>
```

3 – Elde edilen **XML** içeriklerine kısaca bir göz atılır. Alanlar analiz edilir. Her iki içeriğin bir arada ele alınarak tek bir çıktı üretilebileceği gözlemlenir ki bu oldukça iyidir  Hedef **XML** içeriğinin nasıl olacağı planlanır. Bunun için kağıt kalem bile kullanılabilir. ***(2 Dakika)***

```
<?xml version="1.0" encoding="utf-8" standalone="yes"?>
<Countries>
  <Country Name="" PhoneCode="" Code="" ISO="" />
  <Country Name="" PhoneCode="" Code="" ISO="" />
…
</Countries>
```

4 – Bir sınıf kütüphanesi üretilip içerisine **Country** bilgisini taşıyacak bir **POCO(Plain Old CLR Objects)** tipi ile **XML** kaynaklarını birleştirip, tekil halde çıktı olarak verecek fonksiyonları içeren yardımcı bir sınıf yazılır. Bunu doğrudan **Client** uygulama içerisinde de gerçekleştirebiliriz. Ama farklı istemcilerin ihtiyacı olabileceğini düşünerekten*(ve hatta belki servis olarak açarız dışarıya)*atomik olarak fonksiyonlaştırmak yerinde bir hareket olacaktır. ***(20 Dakika)***

Country POCO tipi;

```
using System;
namespace Common
{
```

```
    /// <summary>
    /// Ülke
    /// </summary>
    public class Country
    {
        /// <summary>
        /// İki karakterden oluşan ülke kodu
        /// </summary>
        public string CountryCode { get; set; }
        /// <summary>
        /// ISO kodu
        /// </summary>
        public short ISO { get; set; }
        /// <summary>
        /// Ülke adı
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// Uluslararası telefon kodu
        /// </summary>
        public short PhoneCode { get; set; }
        /// <summary>
        /// Bir ülkenin özelliklerini string formatta geri döndürür
        /// </summary>
        /// <returns>Ülke bilgileri</returns>
        public override string ToString()
        {
            return String.Format("{0},{1},({2}),({3})", Name, CountryCode,
ISO.ToString(),PhoneCode.ToString());
        }
    }
}
```

XML Comment' lerin eklenmesi zaman alan bir iştir. Bu istenirse en son adımlarda icra edilebilir.

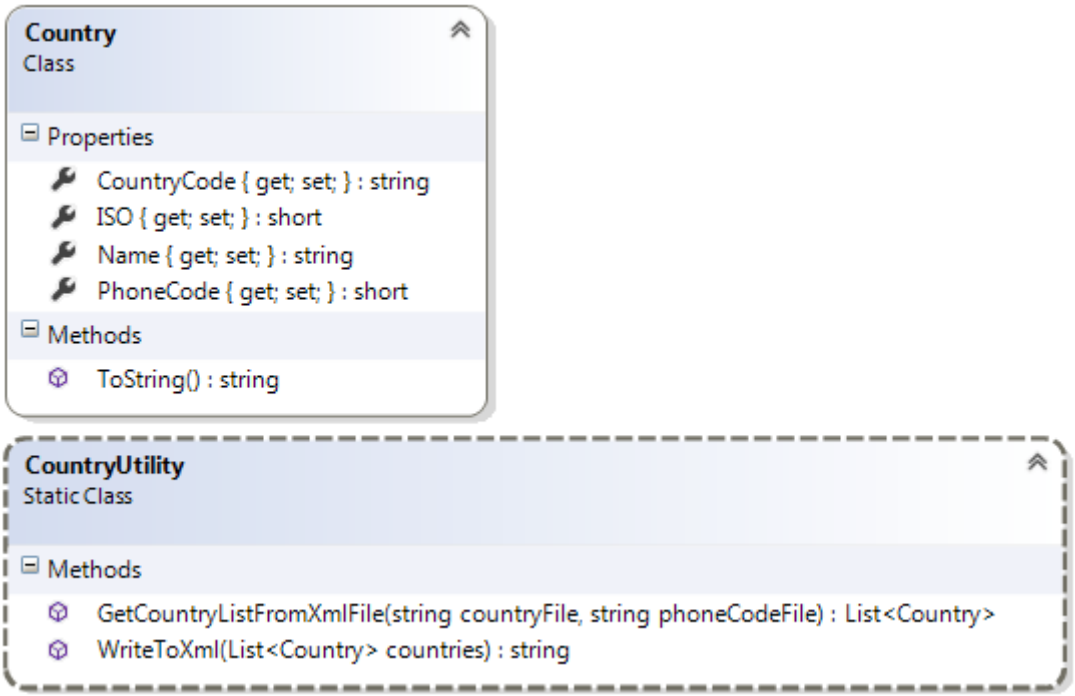
CountryUtility yardımcı tipi;

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Xml.Linq;
```

```
namespace Common
{
   /// <summary>
   /// Country listesinin çekilmesi için gerekli yardımcı sınıftır
   /// </summary>
   public static class CountryUtility
   {
      /// <summary>
      /// Ülke listesini geriye döndürür
      /// </summary>
      /// <param name="countryFile">Ülke ad,kod ve iso bilgilerinin tutulduğu XML dosya adresidir</param>
      /// <param name="phoneCodeFile">Kod ve telefon kodu bilgilerinin tutulduğu XML dosya adresidir</param>
      /// <returns>Ülke adı, kodu, telefon alan kodu ve ISO kod bilgilerini tutan Country tipine ait bir listedir</returns>
      public static List<Country> GetCountryListFromXmlFile(string countryFile,string phoneCodeFile)
      {
         List<Country> countryList = null;
         try
         {
            XDocument countryDocument = XDocument.Load(countryFile);
            XDocument telephoneDocument = XDocument.Load(phoneCodeFile);
            countryList = (from countryNode in countryDocument.Document.Root.Elements("Country")
                           let code=countryNode.Attribute("Code").Value
                           select new Country
                           {
                              Name = countryNode.Value,
                              ISO = Convert.ToInt16(countryNode.Attribute("ISOCode").Value),
                              CountryCode = code,
                              PhoneCode = telephoneDocument.Root.Element(code)!=null?Convert.ToInt16(telephoneDocument.Root.Element(code).Value):(short)-1
                           })
                          .OrderBy(c => c.Name)
                          .ToList();
         }
         catch (Exception excp)
```

```
            {
                throw excp;
            }
            return countryList;
        }
        /// <summary>
        /// Toparlanan ülke listesini XML dosyasına kaydetmek üzere kullanılır
        /// </summary>
        /// <param name="countries">Ülkerin listesi</param>
        /// <returns>Kayıt başarılı bir şekilde yapılmış ise fiziki dosya adres bilgisini
döndürür</returns>
        public static string WriteToXml(List<Country> countries)
        {
            string fileOutputPath = String.Empty;
            if (countries.Count != 0)
            {
                fileOutputPath = Path.Combine(Environment.CurrentDirectory,
"Country.xml");
                XDocument xDoc = new XDocument(new XDeclaration("1.0", "utf-8",
"yes"));
                XElement root = new XElement("Countries");
                foreach (var country in countries)
                {
                    XElement element = new XElement(
                            "Country"
                            , new XAttribute("Name", country.Name)
                            , new XAttribute("PhoneCode", country.PhoneCode)
                            , new XAttribute("Code", country.CountryCode)
                            , new XAttribute("ISO", country.ISO));
                    root.Add(element);
                }
                xDoc.Add(root);
                xDoc.Save(fileOutputPath);
            }
            return fileOutputPath;
        }
    }
}
```

Sınıf kütüphanesinin tip modelinin aşağıdaki şekilde görüldüğü gibi tesis edildiği



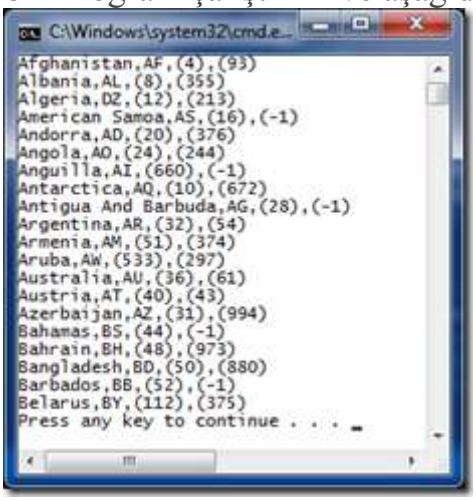
doğrulanır.

5 – Araçları kullanacak olan basit bir **Console** uygulaması açılır, sınıf kütüphanesi referans edilir ve içeriği aşağıdaki gibi geliştirilir. ***(3 Dakika)***

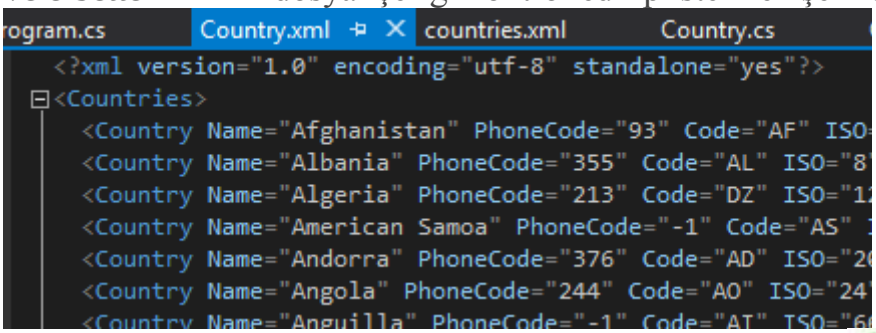
```
using System;
using System.Linq;
using System.IO;
using Common;
namespace ConsoleApplication3
{
   class Program
   {
      static void Main(string[] args)
      {
         // Country listesini tutan fiziki dosya adresini ve telefon kod bilgilerini tutan fiziki dosya adreslerini alalım
         string countryXmlFilePath = Path.Combine(Environment.CurrentDirectory, "Countries.xml");
         string telephoneCodeXmlFilePath = Path.Combine(Environment.CurrentDirectory, "TelephoneCodes.xml");
         try
         {
            // Country tipinden generic List koleksiyonunu yardımcı kütüphanedeki static fonksiyondan üretelim.
            var countries = CountryUtility.GetCountryListFromXmlFile(countryXmlFilePath,
```

```
telephoneCodeXmlFilePath);
            // Test sonuçlarını görmek amacıyla ülke listesinin bir kısmını yazdıralım
            foreach (var country in countries.Take(20))
               Console.WriteLine(country.ToString());

            CountryUtility.WriteToXml(countries);
         }
         catch (Exception excp)
         {
            Console.WriteLine(excp.Message);
         }
      }
   }
}
```

6 – Program çalıştırılır ve aşağıdaki çıktıların üretilip üretilmediği gözlemlenir. ***(1 Dakika)***



ve elbetteki **XML** dosya içeriği kontrol edilip istenilen şekilde olup olmadığına bakılır.



7 – İlk 6 adım başarılı bir şekilde aşıldıktan sonra ise mutfağa gidilir, havanın durumuna göre soğuk veya sıcak bir içecek alınır ***(15 Dakika)***

Görüldüğü üzere **toplamda 36** dakikalık sürede*(kahve süresi hariç)* istediğimiz üretimi gerçekleştirdik. Buraya kadarki adımları siz de sabırla uyguladıysanız eğer basit

bir kod antrenmanı da yapmış olmuşsunuz demektir. Özetle neler yapmışız gelin bir bakalım.

- İki farklı **XML** içeriğini tek bir **LINQ** sorgusunda harmanladık ve**POCO** tabanlı generic bir **List** koleksiyonu ürettik*(LINQ sorgusunda Let keyword' ünü kullandık)*
- Çıktı olarak üretilecek **XML** şemasını düşündük ve tasarladık.
- İlerki kullanımlar için bir **POCO** tipi tasarladık.
- Generic **List** koleksiyonundan yararlanıp bir **XML** dökümanı ve içeriğini oluşturduk.
- **XML** işlemleri için **XDocument**, **XElement**, **XAttribute** gibi tiplerden yararlandık.
- **XML** dökümanındaki **Country** elementlerini üretirken, **XElement** tipinin **Constructor** metodu içerisinde birden fazla **XAttribute** örneğini oluşturarak ekleme yolunu tercih ettik.

Peki her şey istenildiği gibi mi acaba? Mükemmel mi? Eksik bir şeyler hiç mi yok? Yoksa gözden kaçırdığımız bir şeyler var mı?

Doğruyu söylemek gerekirse sonuçları irdelemeden projeyi kapatırsak pek de iyi bir iş yapmamış oluruz. Söz gelimi elimizde istediğimiz ülke bilgilerini içeren bir **XML** dökümanımız ve bunu karşılayan **POCO** türevli bir nesne koleksiyonumz artık var. O halde **Utility** içerisindeki farklı **XML**kaynaklarını birleştiren metodumuz atıl olmuş durumda. Ya yeni **XML** kaynağından veri yükleme işini üstlenmeliyiz ya da içeriye daha akıllı bir parça koyup sonuç **XML** dökümanı yoksa veya içeriği boş ise tekrardan orjinal kaynaklardan çekip üretme işini ele alacak bir kod parçası eklemeliyiz. İşte bu da bizim için 8nci maddemiz oluyor.

8 – Kodun işleyişinin gözden geçirip düzeltilmesi veya eklenmesi gereken yerler var ise bunların tespit edilerek kodlanması. **(??? Dakika)**

???

9 – Çalışmanın tekrardan test edilmesi ve sonuçların yeniden irdelenerek istenen üretimin gerçekleştirilip gerçekleştirilmediğine bakılması ki bu 8nci madde içerisinde ela alınabilir **(??? Dakika)**

???

Soru işaretlerinden de anlaşılacağı üzere bu kısımlar tamamen size ait değerli okurlarım  Ama düşünce yapısını az da olsa ifade edebildiğimi düşünüyorum. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ConsoleApplication3.zip (76,18 kb) *[Örnek bilinmez ama öyle denk geldiği için Visual Studio 2012 RC sürümünde geliştirilmiştir]*

Text Template ve VSIX Project Template Kullanımı

Salı, 17 Temmuz 2012 17:04

Text Template, Visual Studio Sdk, Vsix Project Template, Visual Studio 2012

[Aşağıdaki örnek Visual Studio 2012 RC sürümü üzerinde ele alınmıştır]

Merhaba Arkadaşlar,

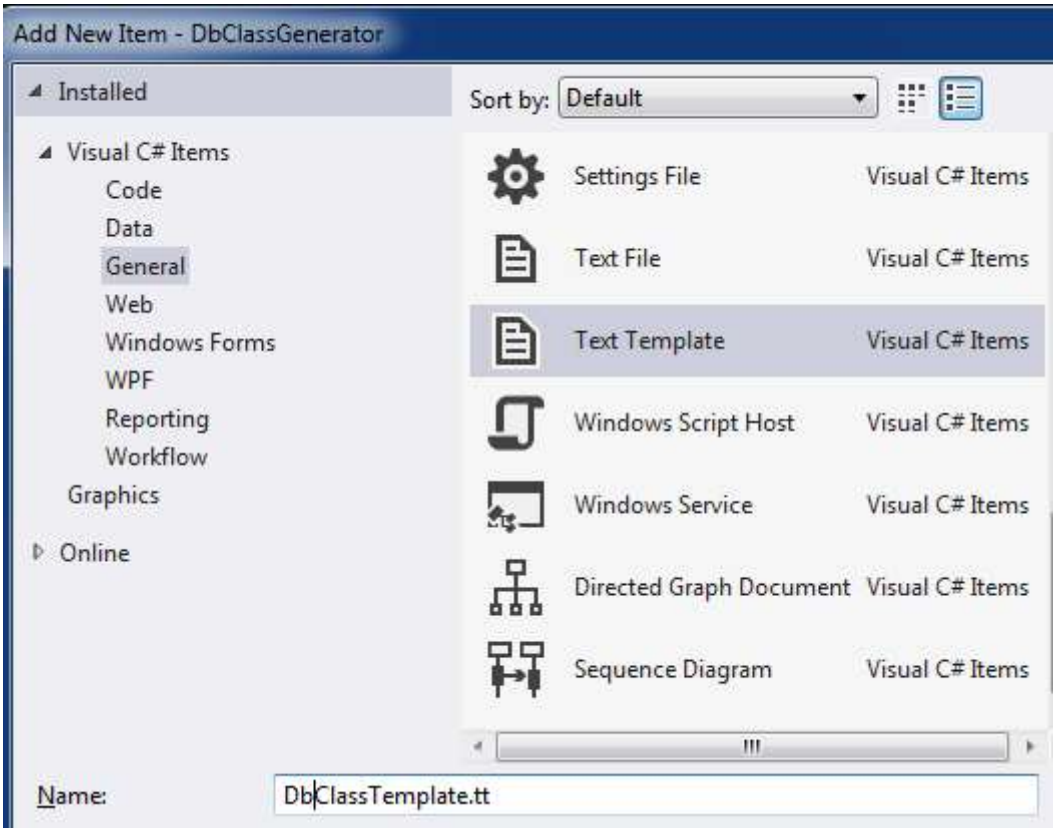
Çoğu zaman geliştiricilerin karşısına zaman kısıtı olan projelerde, sıklıkla tekrar eden çözümsel ihtiyaçlar çıkar. Örneğin, ürünün içerisinde n sayıda ekran kullanıldığını ve bunların aslında belirli bir noktaya kadar bir kaç parametre ile değişen ama standart kod içeriklerine sahip olduğunu düşünün. Hatta bu tip ekranları bir kaç proje için aynı şekilde ürettiğinizi.

Yazılımcılar bu gibi durumlarda yükü azaltmak ve özellikle Deadline sürelerini eritmek adına, parçaları otomatik olarak üreten kodlar ile çözümleme yoluna gitmeye gayret ederler. Bir başka deyişle **otomatik kod üreticilerini(Auto Code Generator)**yazmak için çaba gösterirler. Bu oldukça etkili ve önemli bir yaklaşımdır. Tekrarlı işleri azaltmakla kalmaz, aynı zamanda yeniden yapılması gereken üretimlerde veya toplu güncellemelerde işleri merkezi bir noktadan kolaylaştırır.

Ancak yazılımcıların bu gibi ihtiyaçlarda yine de yaptıkları bazı temel hatalar vardır. Söz gelimi dosya üretme ve devreye alma işlemlerini yapmak için herşeyi sıfırdan yazma yoluna gidebilirler. Oysaki kullanılan geliştirme ortamlarının bu gibi noktalarda ürettikleri bazı kolaylaştırıcı çözüm yolları da bulunmaktadır. Söz gelimi **Visual Studio** tarafından bakıldığında, **Text Template**' ler kod dosyalarının otomatik üretiminde kullanılabilir. Hatta **Visual Studio SDK** ile birlikte gelen **VSIXProject Template**' ler ile bu gibi üretimlerin birer **Extension** olarak şablonlaştırılması da mümkündür.

İşte bu yazımızda bu tip bir ihtiyacı göz önüne alarak **Text Template** ve bununla ilişkili **VSIX Project Template** üretimi işinin içerisinde yer almaya çalışıyor olacağız. **Text Template** ve **VSIXProject Template** öğeleri ile ilişkili teknik detayları **MSDN**' de bulabilirsiniz. Biz bu yazımızda çok daha basit bebek adımları ile ilerleyerek çözüme ulaşmaya çalışacak ve adımlarımız arasında, aslında neleri yapıp neleri elde ettiğimizi göreceğiz.

İşe ilk olarak **Class Library** projesi oluşturup, **Text Template** tipinden bir öğeyi içeri dahil ederek başlayabiliriz.

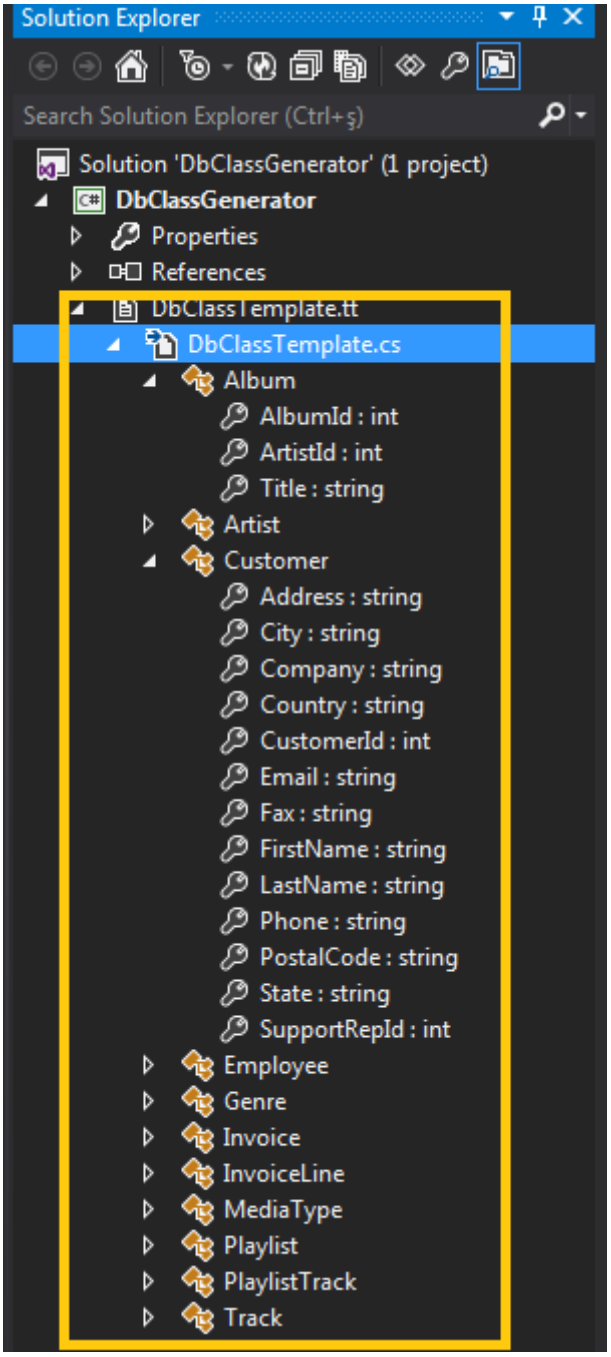


Öğemize **DbClassTemplate** ismini verebiliriz. **TT(Text Template)** uzantılı olan bu dosya aslında metinsel içerik ile **C#/Vb.Net** tabanlı kod parçalarını bir arada ele alıp üretimi gerçekleştirmek üzere kullanılmaktadır. **Visual Studio** ile entegre çalışmakta olup yaptığımız her **Save** işlemi sonrası bir üretim gerçekleşmektedir. Söz konusu dosyanın içeriğini ise aşağıdaki gibi oluşturduğumuzu düşünelim.

```
<#@ template debug="True" hostspecific="True" language="C#" #>
<#@ output extension=".cs" #>
<#@ Assembly Name="System.Data" #>
<#@ Import Namespace="System.Data.SqlClient" #>
<#@ Import Namespace="System.Data" #>
<#@ Import Namespace="System.Text" #>
// Özet       :Bu dosya içerisinde seçilen veritabanı içerisindeki tablolara eş düşen birer sınıf söz konusudur
// Yazan     :Unknown Developer
// Yazım tarihi    :Uzay Tarihi 2012
<#
var connectionString = "data source=.;database=Chinook;integrated security=SSPI;MultipleActiveResultSets=True";
var builder = new StringBuilder();
var dbName = "Chinook";
using (SqlConnection connection = new SqlConnection(connectionString))
{
   var commandTables = new SqlCommand("select object_id,name from sys.tables where
```

```
type='U' order by name",connection);
   var commandColumns = new SqlCommand("select C.name, (select top 1 name from
sys.types T where T.system_type_id=C.system_type_id) as TypeName from sys.columns C
where object_id=@ObjectId order by C.name",connection);
   commandColumns.Parameters.Add("@ObjectId", SqlDbType.Int);
   connection.Open();
   SqlDataReader readerTables = commandTables.ExecuteReader();
   while (readerTables.Read())
   {
      builder.AppendLine(string.Format("\tpublic class {0}",
readerTables["name"].ToString()));
      builder.AppendLine("\t{");
      commandColumns.Parameters["@ObjectId"].Value =
Convert.ToInt32(readerTables["object_id"]);
      SqlDataReader readerColumns = commandColumns.ExecuteReader();
      while (readerColumns.Read())
      {
         // Bu kısımda daha etkili bir tip dönüşüm fonksiyonelliği kullanılmalıdır.
         string dbType=readerColumns["TypeName"].ToString();
         string dotNetType = "object";
         if (dbType.Contains("varchar") || dbType.Contains("text")) dotNetType = "string";
         if (dbType.Contains("int") || dbType == "number" || dbType=="numeric")
dotNetType = "int";
         if (dbType == "decimal" ||dbType == "money") dotNetType = "decimal";
         if (dbType.Contains("date") || dbType.Contains("Date")) dotNetType =
"DateTime";
         if (dbType.Contains("binary")) dotNetType = "byte[]";
         builder.AppendLine(string.Format("\t\tpublic {0} {1}{{get;set;}}", dotNetType,
readerColumns["name"].ToString()));
      }
      readerColumns.Close();
      builder.AppendLine("\t}");
   }
   readerTables.Close();
}
#>
using System;
using System.Linq;
namespace <#= dbName #>
{
```

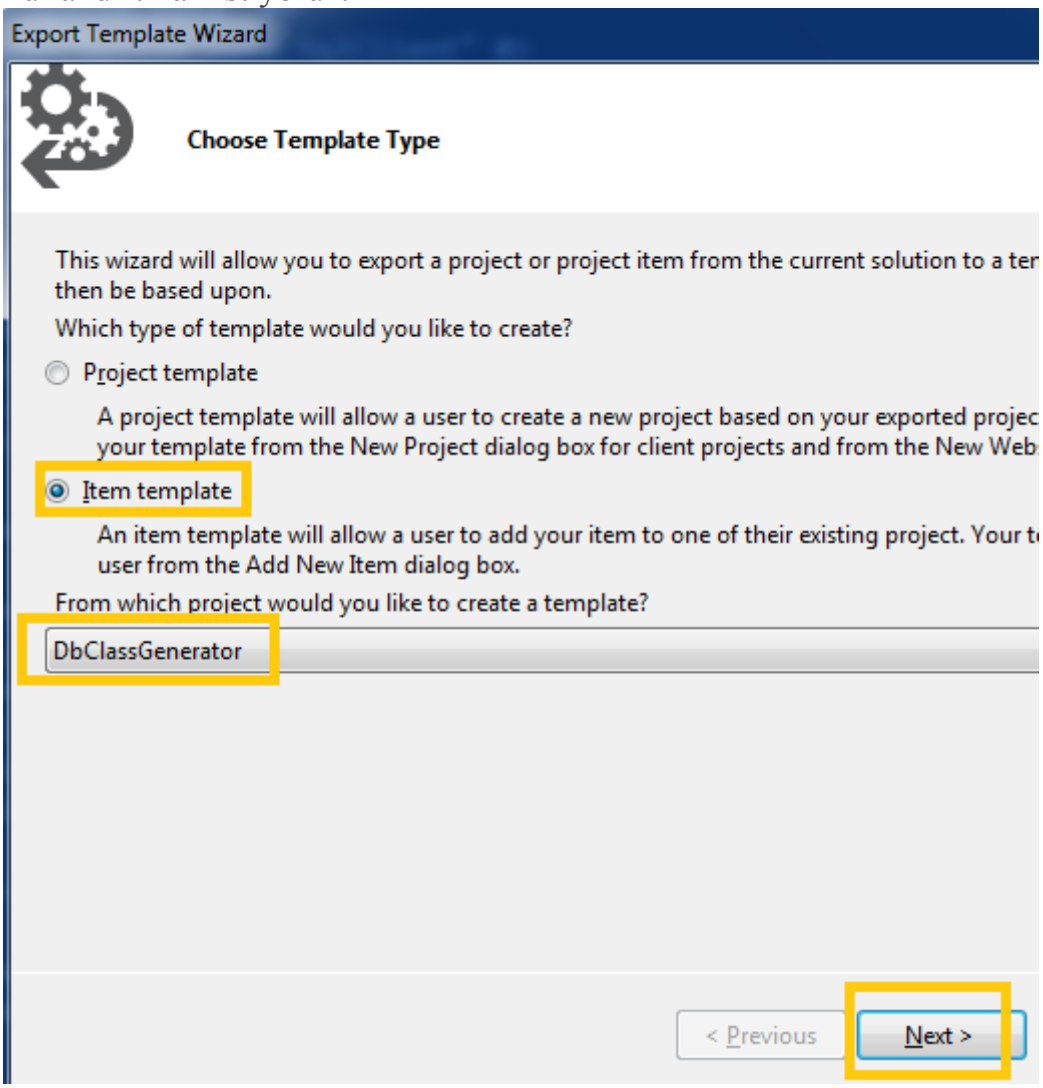
<#= **builder.ToString()** #>
}
Şimdi bu kod parçasında ne yaptığımıza bir bakalım.
<#@ ile başlayıp biten kısımlarda, üretimi yapılacak olan çıktıya ve TT dosyası içeriğine ait bir takım bilgiler vermekteyiz. Örneğimizde çıktının **cs** uzantılı bir **CSharp** dosyası olacağı belirtiliyor. Bunun dışında **System.Data Assembly'** ının çıktının bulunacağı yerde var olması gerektiği ifade ediliyor. Ayrıca kod içerisinde kullanılacak olan **isim alanları(Namespaces)** da belirtiliyor.
<# ile başlayıp biten kısımların haricinde kalan bölümler, tipik olarak çıktı dosyası içerisine yazılan metinsel bilgiler olarak da düşünülebilirler. Her ne zaman bir kod parçasını çalıştırmak istersek, <#ile başlayan blokları ele almamız gerekmektedir.
Örneğimizde yer alan ilk bloğun yaptığı iş, veritabanına bağlanıp **Chinook** tablolarını ve bu tablolara ait kolonların karşılığı olan **class** ve **property** içeriklerini üretmektir. Bu amaçla içeride**StringBuilder** tipinden yararlanıldığı görülmektedir. Yanlız şu kod parçasına dikkat etmeliyiz
using System;
using System.Linq;
namespace <#= **dbName** #>
{
<#= **builder.ToString()** #>
}
Burada **<#=builder.ToString()#>** yazan kısmın yerine, yukarıdaki kod parçasının içerisinde kullanılan **builder** isimli **StringBuilder** değişkeninin çıktısı yerleştiriliyor olacaktır. Hatta **dbName**değişkeni de **namespace** adı olarak alınmaktadır. Geri kalan kısımlar sabit metinsel bilgilerdir.
Dosyamızı bu haliyle kayıt ettiğimizde, **Visual Studio IDE'** si otomatik olarak içeriği çalıştıracak ve bir **C#** kod dosyasını üreterek içeriğini aşağıdaki şekilde görülen hale getirecektir.



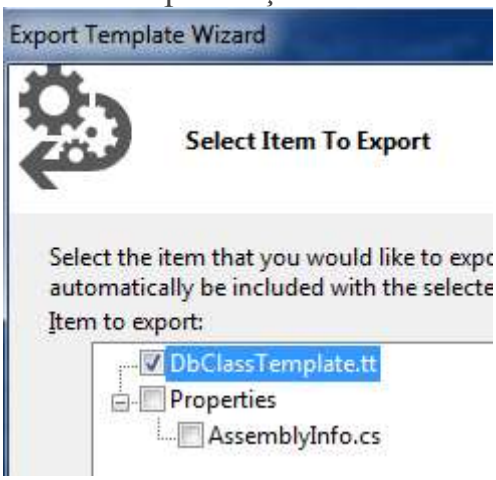
Dikkat edileceği üzere **Chinook** veritabanında yer alan tablolara karşılık gelen sınıf dosyaları üretilmiş olup, **Column'** ların veri tiplerine göre de uygun **Property'** ler ilgili tipler içerisine dahil edilmiştir. Artık elimizde **cs** tabanlı otomatik olarak üretilmiş bir içerik mevcuttur. Bu haliyle de içeriği herhangibir **.Net** projesine ekleyip kullanabilme şansına sahibiz.

Şu anda yapmak istediğimiz ise söz konusu **Text Template** içeriğini ve üretimini bir **Extension**haline getirmektir. Bu sayede bu TT öğesini herhangibir projeye ekleyebilir ve gerekli otomatik üretimleri yaptırtabiliriz. Tabi bu yapının parametrik olarak çalışması çok daha önemlidir. Nitekim geliştiriciler üretimini yapacakları veritabanına ait Connection String bilgisini isterlerse seçebilmelidir. Şimdi yazımızın ikinci kısmını ele almaya başlayabiliriz

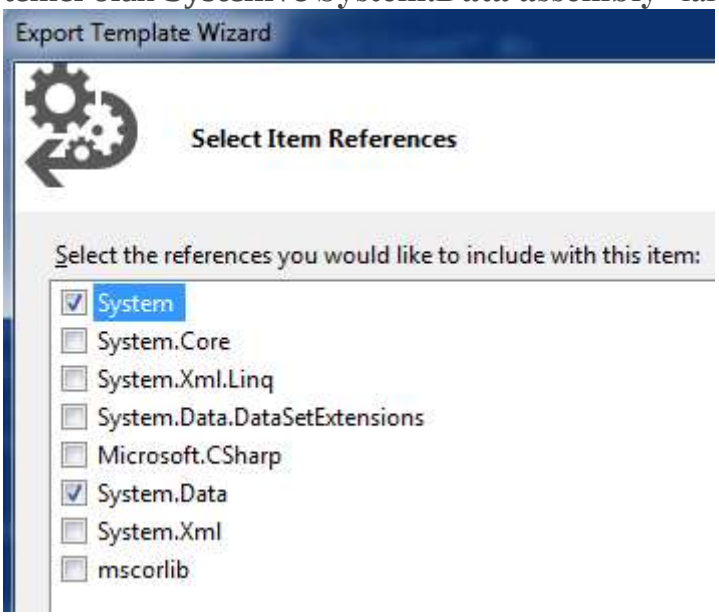
Bu amaçla ilk olarak **File** menüsünden **Export Template** seçeneğini işaretleyip**DbClassGenerator'** ü sıkıştırılmış bir paket haline getirmemiz gerekmektedir. Söz konusu paketin üretimi sırasında **Visual Studio** bir **Wizard** yardımıyla bizden bir kaç adımı tamamlamamızı isteyecektir. Söz konusu adımları aşağıdaki sırayla icra edebiliriz. Adım 1de Item Template seçimi yapılabilir. Nitekim TT' mizi bir proje öğesi olarak kullandırtmak istiyoruz.



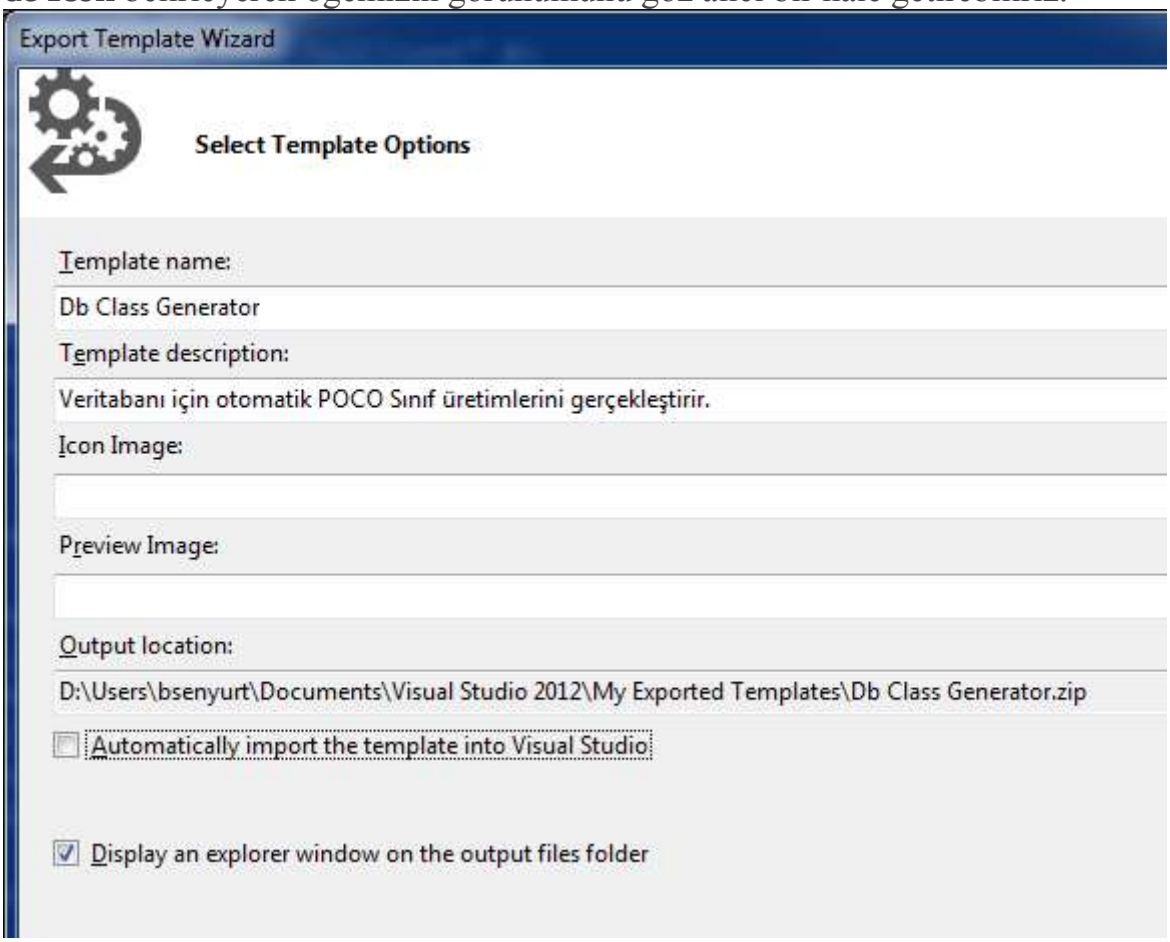
Adım 2 de paket içerisine almak istediğimiz **Text Tempalte** dosyalarını seçebiliriz.



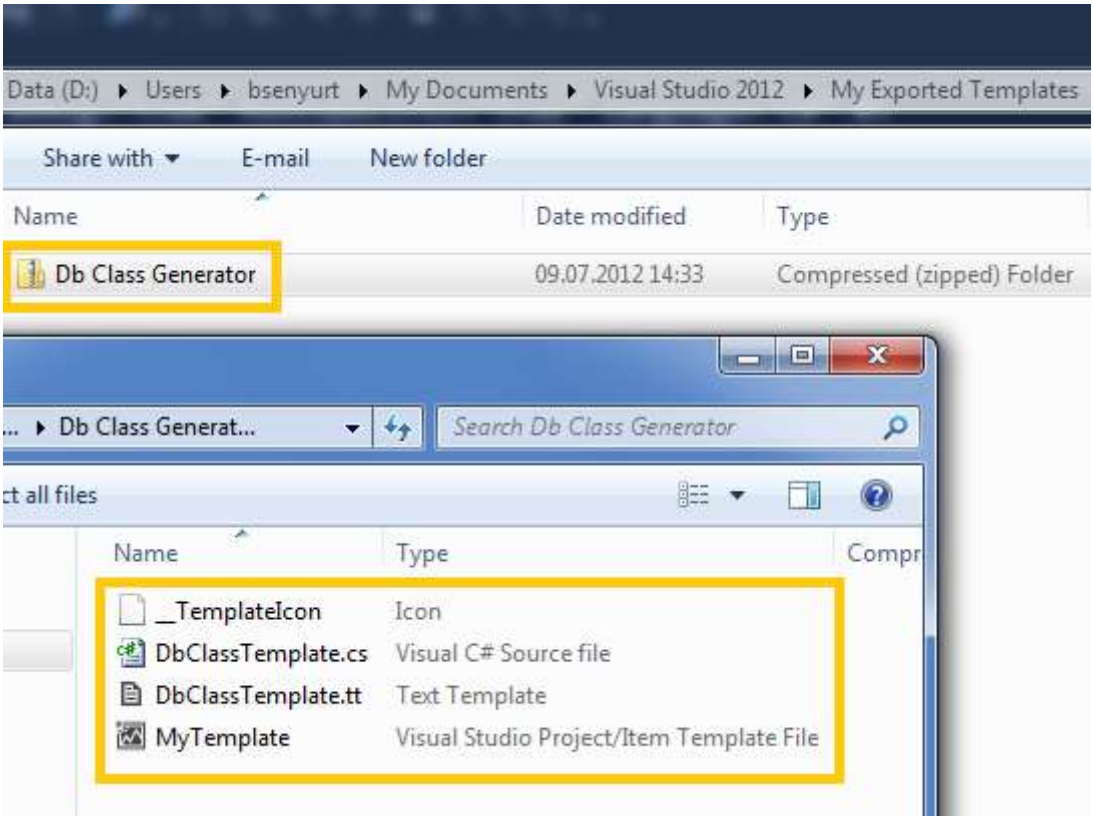
3ncü adımda, **Text Template'** in çalışması sonrası ortaya çıkacak ürünün dahil olduğu projenin referans etmesi gereken **Assembly'** lar var ise bunu belirtebiliriz. Örneğimizde temel olan **System**ve **System.Data assembly'** ları seçilmiştir.



4ncü adımda **Template** ile ilişkili isim ve açıklama bilgilerini verip dilersek bir de **Icon** belirleyerek öğemizin görünümünü göz alıcı bir hale getirebiliriz.

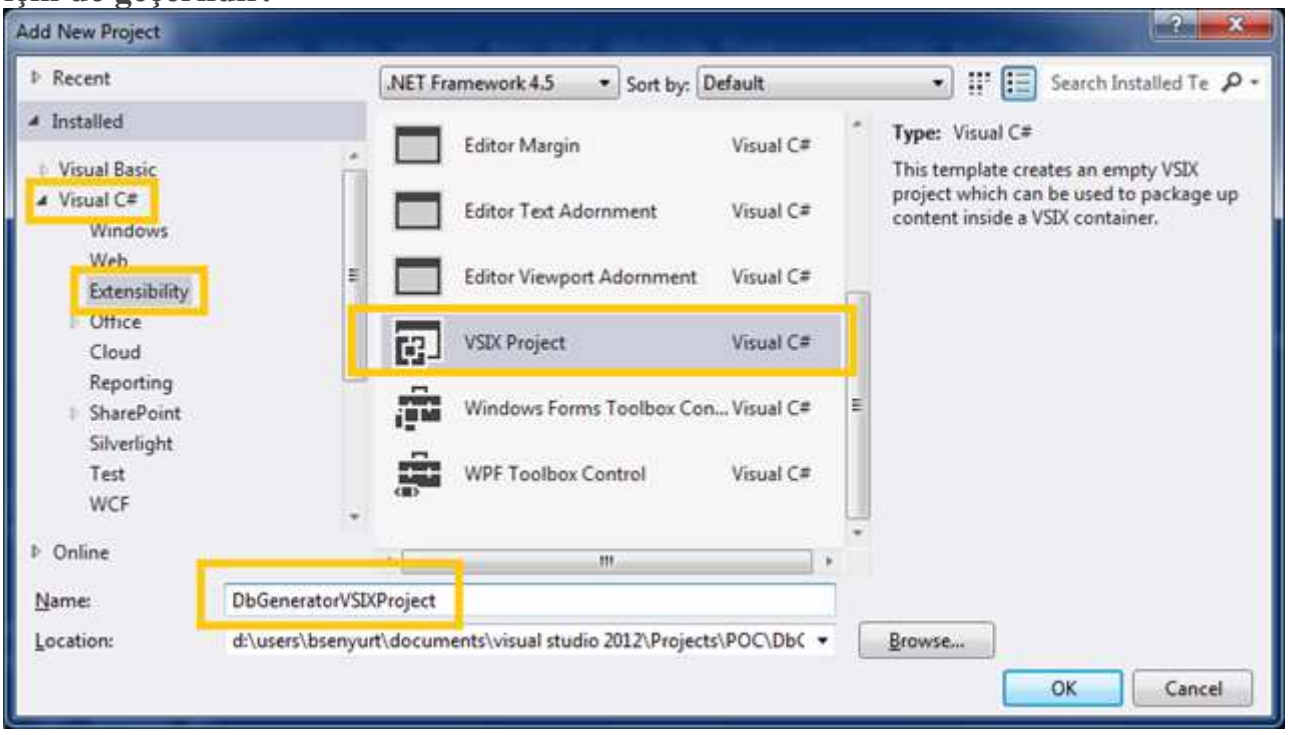


Bu işlemleri tamamladığımızda ise aşağıdaki ekran görüntüsünde yer alan sıkıştırılmış dosyanın üretildiğini görmüş olacağız.

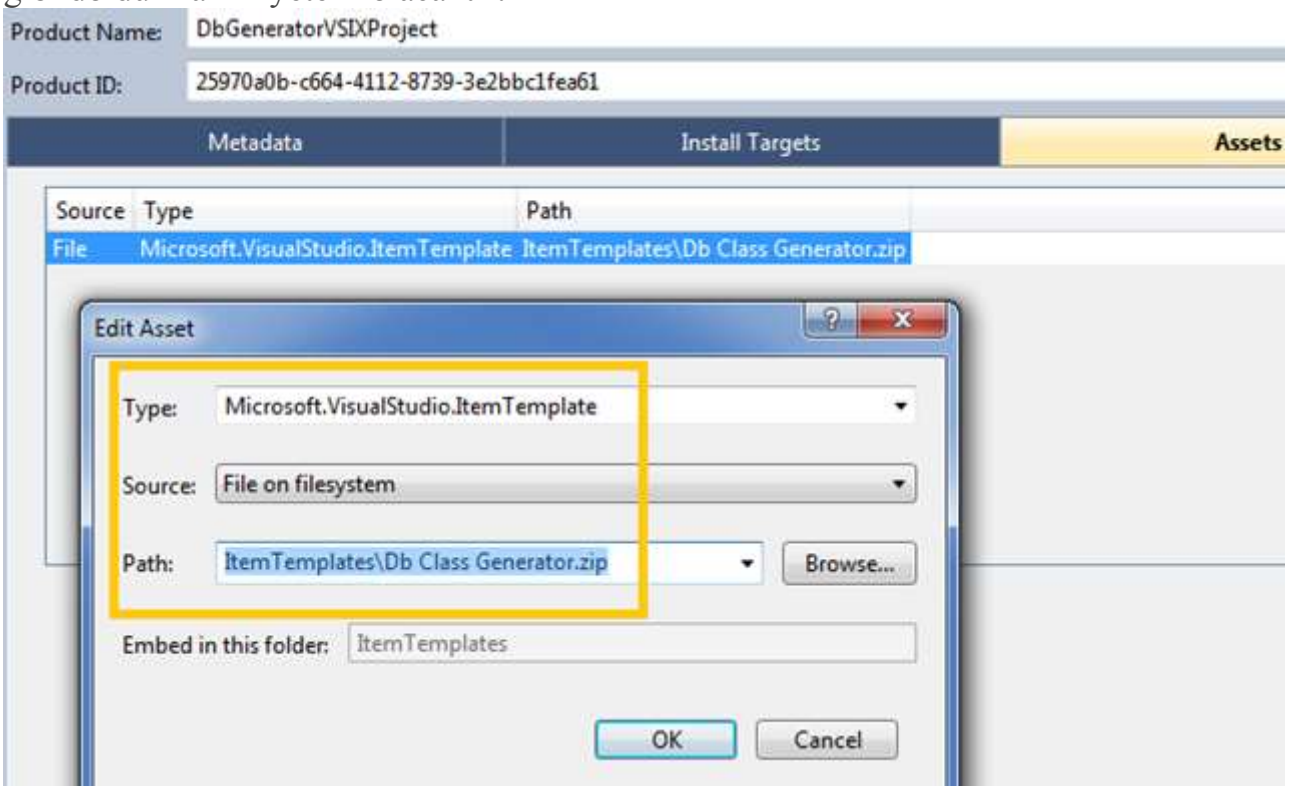


Üretilen dosya adresi bizim için önemlidir nitekim **VSIX Project Template'** de bunu kullanıyor olacağız. Çünkü VSIX projesi ilgili dosyayı alıp kendi içerisine entegre ediyor olacak. Öyleyse Visual C# – Extensibility sekmesinden bir VSIX Project öğesi seçerek ilerlemeye devam edelim.

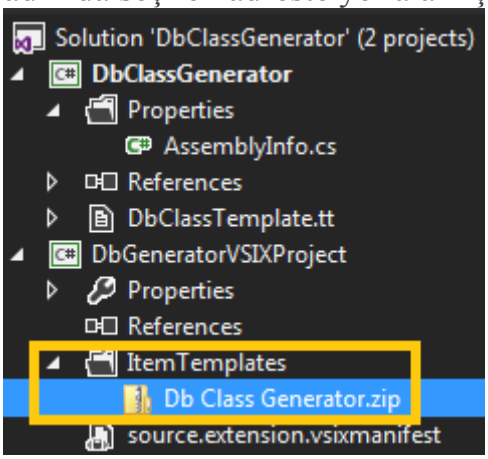
Buradaki VSIX Project öğesinin kullanılabilir olması için Visual Studio 2012 SDK' nın yüklü olması gerektiğini unutmayalım. Aynı durum Visual Studio 2010 sürümü için de geçerlidir.



Bu işlem sonrasında **Visual Studio IDE**' si karşımıza detaylı bir özellik penceresi çıkartacaktır. Bu özellikler temel olarak **vsixmanifest** uzantılı dosya içerisinde bulunmaktadır. **Designer** tarafında ki en önemli kısımlardan birisi de **Assets** bölümüdür. Burada daha önceden **Zip** çıktısı haline getirdiğimiz **Template** içeriğinin işaretlenmesi esasına dayalı bazı seçimler yapmamız gerekmektedir. Örneğimizde bu kısmı aşağıdaki gibi doldurmamız yeterli olacaktır.



Type kısmında **ItemTemplate** seçildiği görülmektedir *(Template' i hangi tipte Export ettiğimizi hatırlayın)* Kaynak olarak **File on filesystem** seçeneği işaretlenmiştir. **Path** bölümünde zip uzantılı dosyanın lokasyonunun verilmesi yeterli olacaktır. İlgili adres bilgisi verildikten sonra **Asset**kısmına tekrardan girilirse yol bilgisinin artık projedeki **ItemTemplates** klasörünü işaret ettiği gözlemlenebilir. Bunun sebebi ilk adımda seçilen adreste yer alan içeriğin **VSIX** projesine kopyalanmış olmasıdır.

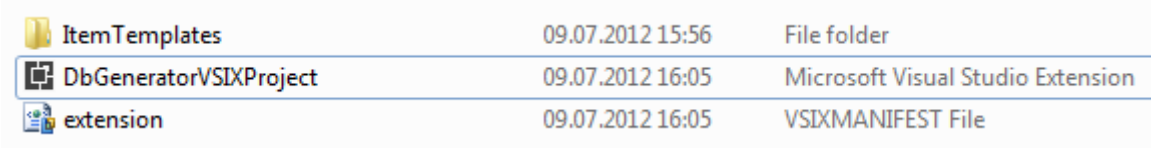


Manifest bilgilerini düzenlediğimiz kısımda set ettiğimiz özellikler çok doğal olarak arka plana **XML**tabanlı olarak yazılmaktadır. Sonuç itibariyle **vsixmanifest** dosyasının içeriği de aşağıdaki gibi olabilir.

```
<?xml version="1.0" encoding="utf-8"?>
<PackageManifest Version="2.0.0" xmlns="http://schemas.microsoft.com/developer/vsx-schema/2011" xmlns:d="http://schemas.microsoft.com/developer/vsx-schema-design/2011">
  <Metadata>
    <Identity Id="25970a0b-c664-4112-8739-3e2bbc1fea61" Version="1.0" Language="en-US" Publisher="BurakSenyurt" />
    <DisplayName>Db POCO Generator</DisplayName>
    <Description>Veritabanı tabloları için otomatik olarak POCO tip üretimi gerçekleştirmek üzere tanımlanmış bir Extension dır.</Description>
  </Metadata>
  <Installation>
    <InstallationTarget Id="Microsoft.VisualStudio.Pro" Version="11.0" />
  </Installation>
  <Dependencies>
    <Dependency Id="Microsoft.Framework.NDP" DisplayName="Microsoft .NET Framework" d:Source="Manual" Version="4.5" />
  </Dependencies>
 <Assets>
    <Asset Type="Microsoft.VisualStudio.ItemTemplate" d:Source="File" Path="ItemTemplates" d:TargetPath="ItemTemplates\Db Class Generator.zip" />
  </Assets>
</PackageManifest>
```

Biz örneğimizde çok basit olarak **Display Name, Description, Id, Version** gibi kısımlara müdahale ettik. Ancak daha ileri seviyede müdahalelerde de bulunabiliriz. Söz gelimi Framework hedef versiyonlarını değiştirebilir ki örnemğizde bu bağımlılık 4.5 sürümüne göre yapılmıştır.

Yaptığımız çalışma sonucu projeyi **build** edersek eğer aşağıdaki gibi **vsix** uzantılı **ExtensionInstaller** dosyasının yer aldığı bir sonuca ulaşmış oluruz.

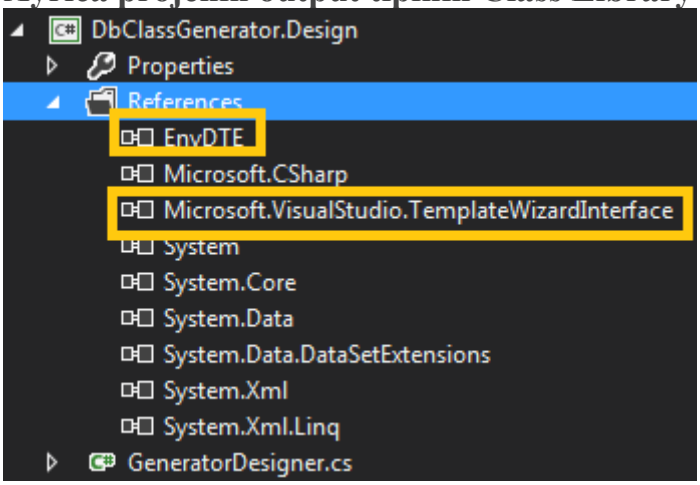


Bir başka deyişle söz konusu **installer** dosyasını çalıştırıp **template'** imizin **Visual Studio**ortamına entegre edilmesini sağlayabiliriz. Lakin halen eksik olan bir kaç şey bulunmaktadır. Projemizde kullandığımız **Text Template** içerisinde yer alan **Connection String** ve **DbName** gibi bilgiler **hard coded** olarak tutulmaktadır. Oysaki bağlantı bilgisini dışarıdan verebilmiş olsaydık çok daha generic bir proje öğemiz olurdu. Şimdi bu parametreleri dışarıdan nasıl alabileceğimize bir bakalım

İlk olarak **DbClassGenerator.Design** isimli bir **Windows** uygulaması oluşturalım ve söz konusu projeye aşağıdaki referansları ekleyelim. Bu proje öğeyi eklediğimiz sırada devreye girecek bir arabirimi geliştiricinin karşısına çıkartmak için kullanılacaktır.

Projenin VSIX dosyasının install edilmesi sırasında devreye girebilmesi için, output olarak C:\Program Files\Microsoft Visual Studio 11.0\Common7\IDE\ adresine çıktı vermesi gerekmektedir. Bu nedenle projenin özelliklerinden Build->Output kısmında ilgili lokasyon bildirilmelidir.

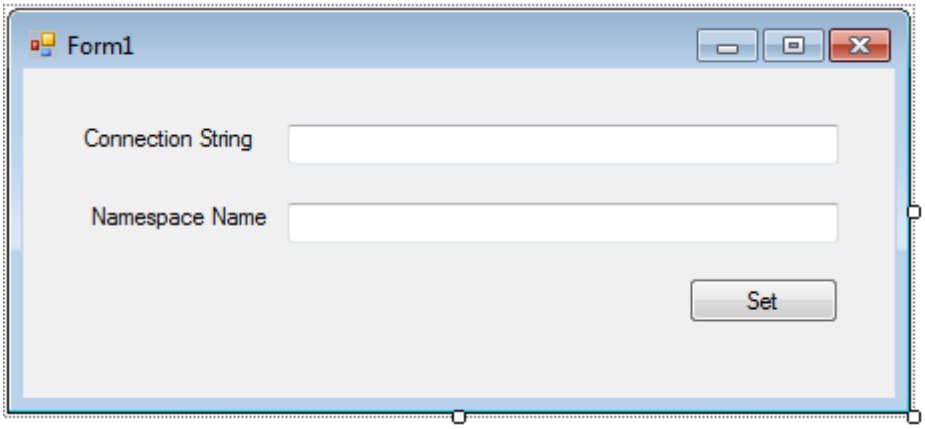
Ayrıca projenin output tipinin Class Library olarak set edilmesi gerekmektedir.



EnvDTE ve **Microsoft.VisualStudio.TemplateWizardInterface assembly'** larının referans edilmesinin ardından, aşağıdaki basit **Windows Forms** tasarımını ve kod içeriğini geliştirerek ilerleyebilir.

Örneğimizde çok basit bir arabirim söz konusudur. Ancak burada daha kompleks bir Connection String designer' ı da söz konusu olabilir. Bu tamamen tasarlanan Template' in kullanacağı parametrelerinin nasıl ve ne şekilde set edileceğine bağlı olarak değişir.

Yine örnekte hata yönetimi çok fazla dikkate alınmamıştır. Zero Data testleri yapılmamıştır. Bu gibi hususları kendi geliştireceğiniz örnek içerisinde dikkatli bir şekilde ele almalısınız.




```
using System;
using System.Windows.Forms;
```

```
namespace DbGenerator.Design
{
   public partial class Form1
      : Form
   {
      public string ConnectionString { get; set; }
      public string NamespaceName { get; set; }
      public Form1()
      {
         InitializeComponent();
      }
      private void Form1_Load(object sender, EventArgs e)
      {
      }
      private void btnSet_Click(object sender, EventArgs e)
      {
         if (!String.IsNullOrEmpty(txtConnectionString.Text))
            ConnectionString = txtConnectionString.Text;
         if (!string.IsNullOrEmpty(txtDbName.Text))
            NamespaceName = txtDbName.Text;
         DialogResult = System.Windows.Forms.DialogResult.OK;
      }
   }
}
```

Dikkat edileceği üzere bu basit formdan **TT** içerisindeki parametrelere set edeceğimiz değişken değerlerini almaktayız. Tabi işin önemli kısmı bu pencerenin, öğe **Visual Studio** projesine eklenirken karşımıza çıkmasını sağlamaktır. Bunun için **IWizard arayüzünü(interface)** implemente edecek bir sınıfı projemize dahil etmemiz gerekiyor. **GeneratorDesigner** olarak adlandırabileceğimiz sınıfın kod içeriğini ise aşağıdaki gibi geliştirdiğimizi düşünelim.

```
using System.Collections.Generic;
using System.Windows.Forms;
using EnvDTE;
using Microsoft.VisualStudio.TemplateWizard;
namespace DbGenerator.Design
{
   public class GeneratorDesigner
      :IWizard
   {
      private bool CanAddProjectItem;
```

```
    public void RunStarted(object automationObject, Dictionary<string, string>
replacementsDictionary, WizardRunKind runKind, object[] customParams)
    {
      Form1 frm = new Form1();
      if (frm.ShowDialog() == DialogResult.OK)
      {
        replacementsDictionary.Add("$connectionString$", frm.ConnectionString);
        replacementsDictionary.Add("$dbnameString$", frm.NamespaceName);
      }
      else
      {
        // Default bir değer kısımlarına geçerli bir takım bilgiler yazmakta yarar olabilir.
Nitekim geçerli bir ConnectionString bilgisi olmaması halinde uygulama hata verecek ve cs
dosyaları başarılı bir şekilde üretilmeyecektir.
        replacementsDictionary.Add("$connectionString$", "default bir değer");
        replacementsDictionary.Add("$dbnameString$", "Default bir değer");
      }
      CanAddProjectItem = true;
    }
    public bool ShouldAddProjectItem(string filePath)
    {
      return CanAddProjectItem;
    }
    public void BeforeOpeningFile(ProjectItem projectItem)
    {
    }
    public void ProjectFinishedGenerating(Project project)
    {
    }
    public void ProjectItemFinishedGenerating(ProjectItem projectItem)
    {
    }
    public void RunFinished()
    {
    }
  }
}
```

IWizard türevli bu sınıf içerisinde basit bir Application Life Cycle söz konusudur aslında. Biz ilk girişte bazı parametrelerin set edilmesi için araya girerek müdahale de bulunuyoruz. Bu nedenle dikkat edileceği üzere RunStarted metodunda, **$** sembolleri arasında tanımlanmış olan bazı parametrelerin değiştirilmesi işlemi söz konusudur. Bunun için

tasarladığımız Windows Form' una ait bir örnek oluşturulmuş, ardından form içerisine girilen metinsel bilgiler alınarak koleksiyona dahil edilmiştir. Tabi çok doğal şu anda kafamızda bir soru oluşması muhtemeldir. Acaba bu değişkenler nerede tanımlanmalıdır  **Application Life Cylce içerisindeki metodlara bakıldığında Loglama işlemlerinin de yapılabileceği uygun noktalar olduğu görülmektedir.**

Tahmin edeceğiniz üzere **Text Template** içeriğini kullanan kısım **VSIX** projemizdir. Bu proje**ItemTemplates** klasörüne **zip** olarak ilgili **Template** içeriğini ve dosyalarını indirmiştir. Parametreleri sisteme entegre etmek için bu zip içeriğindeki bazı dosyaları değiştirmemiz şarttır*(Ya da en baştan TT' yi bunu düşünerek tasarlamak gerekmektedir)*. İlk olarak **vstemplate** uzantılı dosya içeriğinde aşağıdaki değişiklikleri yapalım.

```
<VSTemplate Version="3.0.0"
xmlns="http://schemas.microsoft.com/developer/vstemplate/2005"Type="Item">
  <TemplateData>
    <DefaultName>Db Class Generator.tt</DefaultName>
    <Name>Db Class Generator</Name>
    <Description>Veritabanı için otomatik POCO Sınıf üretimlerini
gerçekleştirir.</Description>
    <ProjectType>CSharp</ProjectType>
    <SortOrder>10</SortOrder>
    <Icon>__TemplateIcon.ico</Icon>
  </TemplateData>
  <TemplateContent>
    <References>
      <Reference>
        <Assembly>System</Assembly>
      </Reference>
      <Reference>
        <Assembly>System.Data</Assembly>
      </Reference>
    </References>
    <ProjectItem SubType=""
TargetFileName="$fileinputname$.tt"ReplaceParameters="true">DbClassTemplate.tt</
ProjectItem>
    <!--<ProjectItem SubType="Code" TargetFileName="$fileinputname$.cs"
ReplaceParameters="true">DbClassTemplate.cs</ProjectItem>-->
  </TemplateContent>
    <WizardExtension>
      <Assembly>DbClassGenerator.Design</Assembly>
      <FullClassName>DbClassGenerator.Design.GeneratorDesigner</FullClassName
>
```

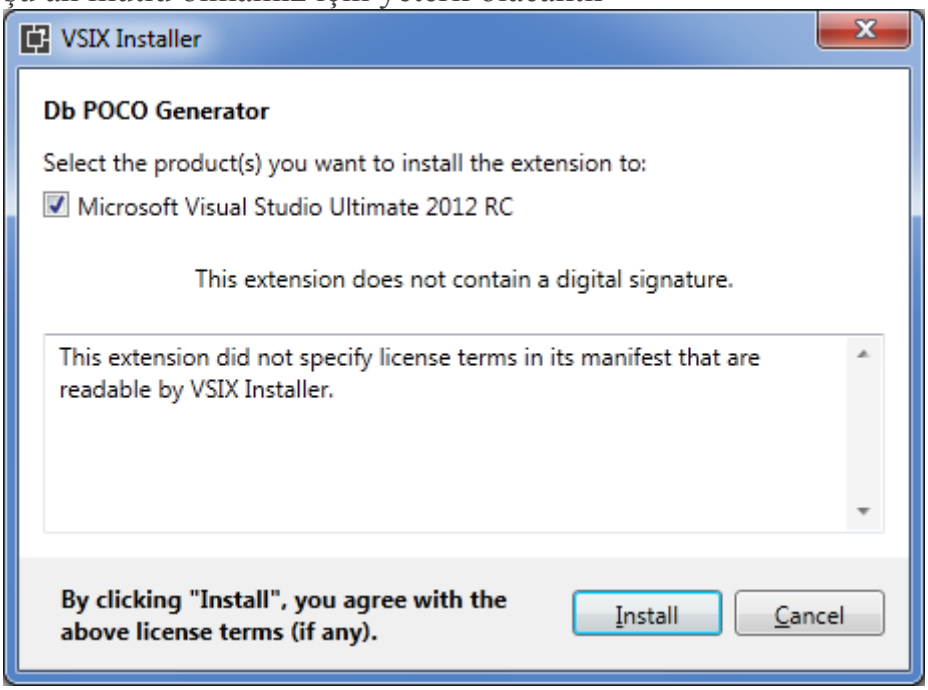
</WizardExtension>
</VSTemplate>
3 önemli değişiklik vardır.

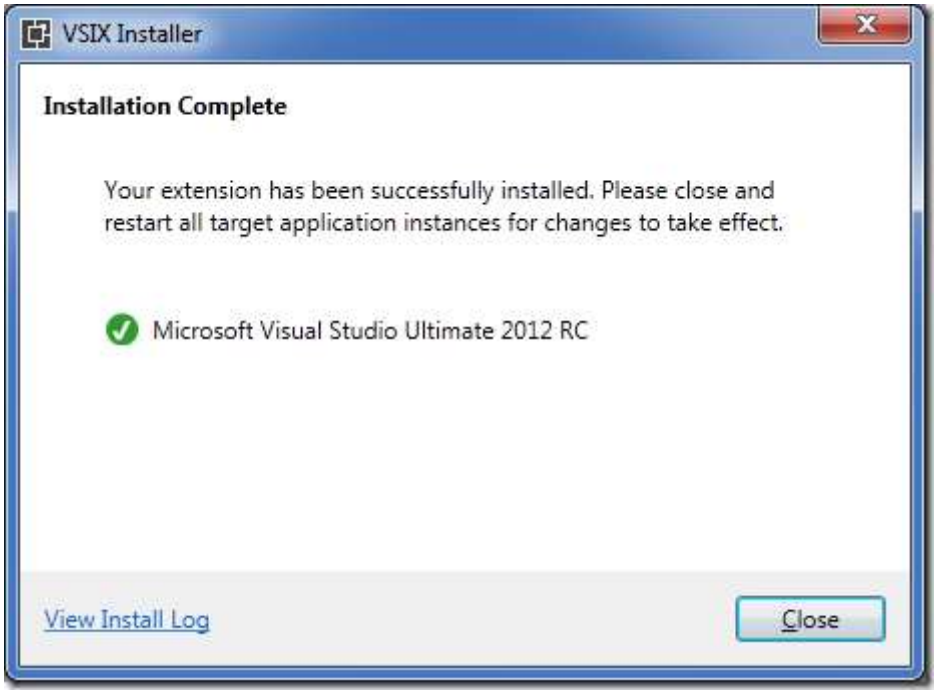
- **cs** dosyasının üretimine ait bildirimde bulunan **ProjectItem** elementi kaldırılmıştır.*(Örnekte yorum satırı yaptık)*
- İkinci olarak **tt** için kullanılan **ProjectItem** elementindeki **ReplaceParameters** değeri **true** yapılmıştır. Nitekim **$** işaretleri arasına aldığımız parametrelerin değiş tokuş edileceğinin belirtilmesi gerekmektedir.
- 3ncü ve en önemli değişiklik ise **WizardExtension** bloğunun eklenmiş olmasıdır. Bu blok tahmin edileceği üzere **$** işareti ile belirlenmiş parametrelerin alınması için gerekli arabirimin devreye alınması sırasında rol oynamakta olup, hangi **IWizard** türevli tipin değerlendirileceğini belirtmektedir.

Tabi **tt** uzantılı dosyamızda da bazı değişikliklerin yapılması şarttır. Yani **$** işaretli parametrelerin eklenmesi gerekmektedir.

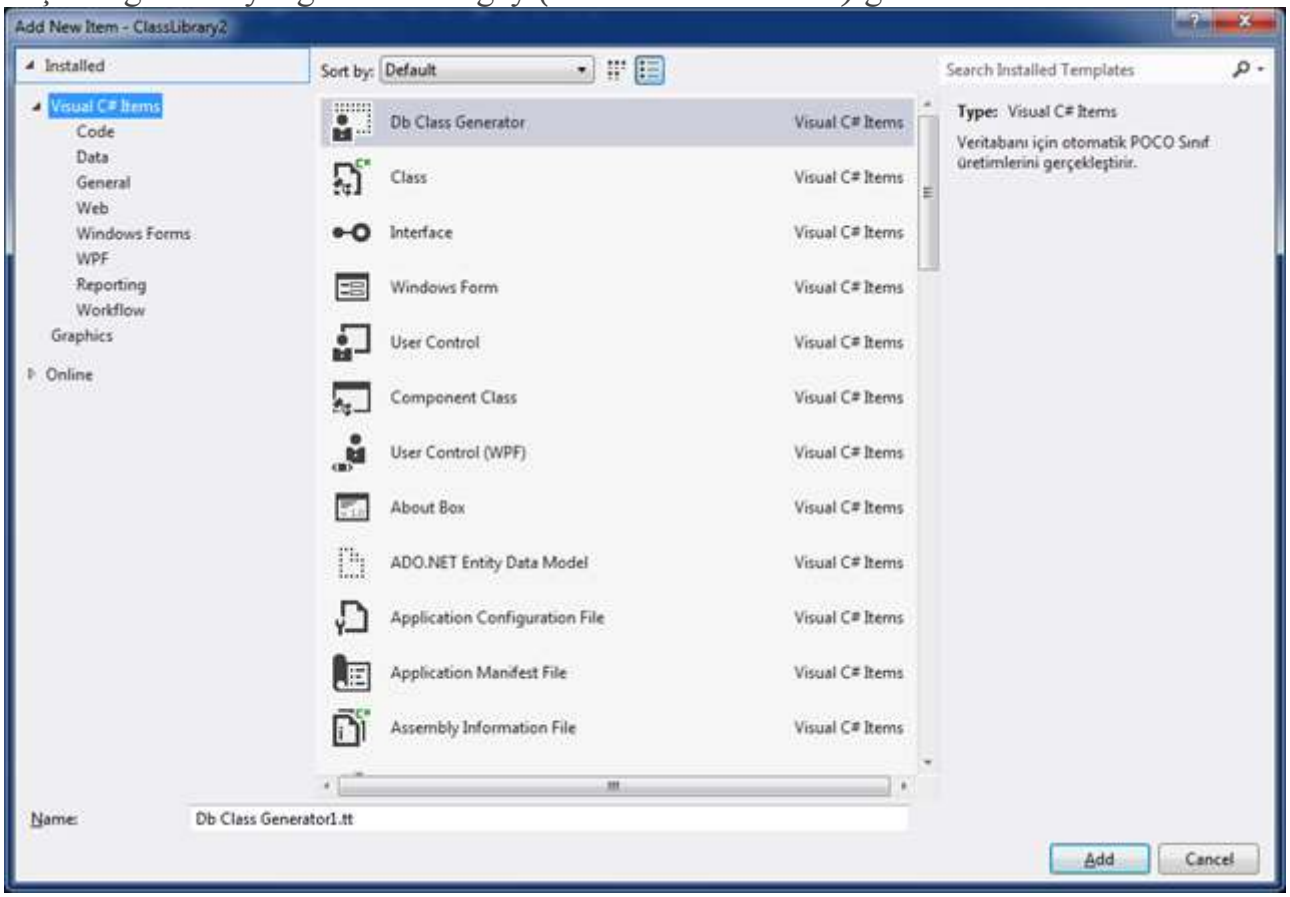
<#
var connectionString = "$connectionString$";
var builder = new StringBuilder();
var dbName = "$dbnameString$";

Bu son işlemlerden sonra **VSIX** Projesini bir kere daha derlememiz gerekir. *(Hatta daha önceden install etmişsek uninstall edip son bir build yapıp tekrardan install etmemiz gerekmektedir)* Tüm bu işlemlerin arından **VSIX** dosyasını çalıştırıp template' in yüklenmesini sağlayabiliriz. Eğer işler yolunda giderse aşağıdaki iki pencereyi görmemiz şu an mutlu olmamız için yeterli olacaktır





Şimdi her hangibir proje açalım ve daha sonra **Add New Item** ile ilerleyelim. Listenin en başında gurur kaynağımız olan öğeyi*(Db Class Generator)* görebiliriz.



Öğeyi seçtikten sonra ise karşımıza tasarladığımız **Windows Forms** penceresi çıkacaktır. Eğer doğru **Connection String** bilgisini girersek*(bu örnekte **MultipleActiveResultSets** değeri için true vermeyi unutmayın)* **cs** içeriğinin otomatik olarak üretildiğine şahit olabilirsiniz 😊

Görüldüğü üzere bir TT projesi VSIX ile bir arada değerlendirildiğinde Visual Studio tarafına Extension yazılması söz konusudur. Biraz uzun ve yorucu bir makale olduğu kadar uygulama sırasında da oldukça dikkat ve titizlik isteyen bir çalışma söz konusudur. Umarım istediğiniz sonuçları sizlerde elde edersiniz. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

Size tavsiyem otomatik Windows Forms veya ASP.Net Web Pages ya da MVC öğeleri üretecek bir TT üzerinde çalışmanızdır. Ayrıca bu örnekteki Windows Forms' u hatalara neden olmayacak ve developer' ın hayatını daha da kolaylaştıracak şekilde geliştirmeyi ciddi anlamda düşünmelisiniz.

[Aşağıdaki örnek Visual Studio 2012 RC sürümü üzerinde ele alınmıştır]

DbClassGenerator.zip (105,75 kb)

Tek Fotoluk İpucu 60 - string Kökenli JSON İçeriği Ters Serileştirmek

Salı, 17 Temmuz 2012 09:03

Tek Fotoluk Ipucu, Json, Datacontractjsonserializer, Extension Methods

Merhaba Arkadaşlar,
Tek Fotoluk İpucu 60' da generic bir nesne örneğinin JSON formatında serileştirilmesini ve serileştirilen içeriğinde string olarak geriye döndürülmesini sağlayan bir extension metod geliştirmiştik. E tabi çok haklı olarak hani bunun ters serileştirmesi nerede diyebilirsiniz. Dediniz di mi? 😀 Buyrun öyleyse.

SomeExtensions - Program.cs

Program.cs

SomeExtensions.Program

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Runtime.Serialization.Json;

namespace SomeExtensions
{
    class Program{
        static void Main(string[] args){
            List<Person> hrDepartment = new List<Person> {
                new Person{ PersonId=1, FirstName="Burak"},
                new Person{ PersonId=2, FirstName="Selçuk"},
                new Person{ PersonId=3, FirstName="Hakan"},
                new Person{ PersonId=4, FirstName="Özcan"},
                new Person{ PersonId=1, FirstName="Uğur"},
            };
            string jsonContent = hrDepartment.SerializeToJson();
            var hrDepartmantList=jsonContent.DeserializeFromJson<List<Person>>();
            foreach (var person in hrDepartmantList)
            {
                Console.WriteLine(person.FirstName);
            }
        }
    }
    public static class ObjectExtensions{
        public static T DeserializeFromJson<T>(this string source)
        {
            T result = default(T);
            DataContractJsonSerializer serializer =
                new DataContractJsonSerializer(typeof(T));

            using (MemoryStream stream = new MemoryStream())
            {
                using (StreamWriter writer = new StreamWriter(stream))
                {
                    writer.Write(source);
                    writer.Flush();
                    stream.Position = 0;
                    result = (T)serializer.ReadObject(stream);
                }
            }
            return result;
        }

        public static string SerializeToJson<T>(this T sourceObject)
```

C:\Windows\system3...

```
Burak
Selçuk
Hakan
Özcan
Uğur
Press any key to continue . . .
```

Workflow Designer' ı Yeniden Host Etmek (WF 4.0)

Cuma, 13 Temmuz 2012 08:05

Workflow Foundation 4.0, Rehosted Workflow Designer, Xaml, Workflowdesigner,Activity, Custom Activity Designer, Code Activity, Native Activity, Workflow Context

Merhaba Arkadaşlar,
Çoğu zaman sinemada daha önceden vizyona girmiş olan bir filmin yeniden çekilmiş bir versiyonuna rastlarız. Örneğin **Batman Begins** veya vizyona bu yaz girecek **Total Recall** gibi. Hatta bazen **Cover** olarak adlandırdığımız bir durum söz konusu olur ve çeşitli müzik guruplarının önemli parçalarının tekrardan, aynı ekipçe veya başkalarınca yorumlandığını görür, duyarız.
Sonuç itibariyle insanlar zaman zaman yapılmış olan bazı çalışmaları hem teknolojinin yeni nimetleri, hem de farklı şekilde yorumlayabilme isteği nedeni ile tekrardan ele alabilirler.
Hatta bu felsefe yazılım dünyasında da zaman zaman vuku bulan bir senaryodur. Özellikle **IDE**tarafında. Bir **IDE'** nin kabuğu üstüne giyidirilebilen parçaları farklılaştırabildiğinizi veya var olan**IDE'** lerden farklı olan alternatiflerini üretebildiğinizi düşünün. Örneğin **SharpDevelop**
Aslına bakarsanız **Visual Studio** gerçekten harika bir **IDE** ortamı sunmaktadır. Hatta **UX** olarak bilien **User eXperience** değil de tam anlamıya **Developer eXperience'** ın hat safhada olduğu bir geliştirme ortamıdır. Lakin genişletilebilir olması*(Extension Manager' a dikkatiniz çekmek isterim)*haricinde çok gelişmiş özellikleri olmakla birlikte, zaman zaman daha hafif bir sürüme ihtiyaç duyabiliriz. Örneğin **Workflow Foundation** tabanlı olarak bir iş akışı tasarım uygulaması geliştirmek istediğinizi düşünün
Bu tip uygulamalarda herşeyi baştan ele alıp Amerikayı tekrardan keşfetmeyi deneyebilirsiniz elbette. Ancak zaten elimizde var olan bir **Designer** ortamı var ise, sadece bunu alıp yeni bir kabuk giydirmeye çalışmak daha etkili ve hızlı bir çözüm olabilir. İşte bu yazımızda çok basit olarak**Workflow Designer** ortamının **Visual Studio** dışarısında nasıl kullanılabileceğini öğrenmeye çalışıyor olacağız. Aracımızdan beklediğimiz özellikler temel olarak aşağıdaki maddeler halinde ifade edilebilir.

1. Var olan **Primitive Workflow Component'** leri veya bizim tarafımızdan geliştirilmiş bileşenleri içeren**Toolbox'** a sahip olmalıdır.
2. **Workflow** içeriğinin tasarlanabileceği **Visual Studio** içerisindeki **Designer** bulunmalıdır.
3. Herhangibir **Workflow** bileşeni seçildiğinde, buna ait özelliklerin dolacağı ve tabiki değiştirilebileceği bir**Properties** penceresi yer almalıdır.

4. Tasarlanan **Workflow** örnekleri kayıt edilebilmeli veya **XAML*(eXtensible Application Markup Language)*** içerikli dosyalardan yüklenebilmelidir.
5. Tasarlanan veya yüklenen **Workflow** örnekleri çalıştırılabilmelidir.
6. Kullanıcı deneyimini yüksek tutmak istediğimizden **WPF(Windows Presentation Foundation)** tabanlı bir arayüz sunulabilmelidir.

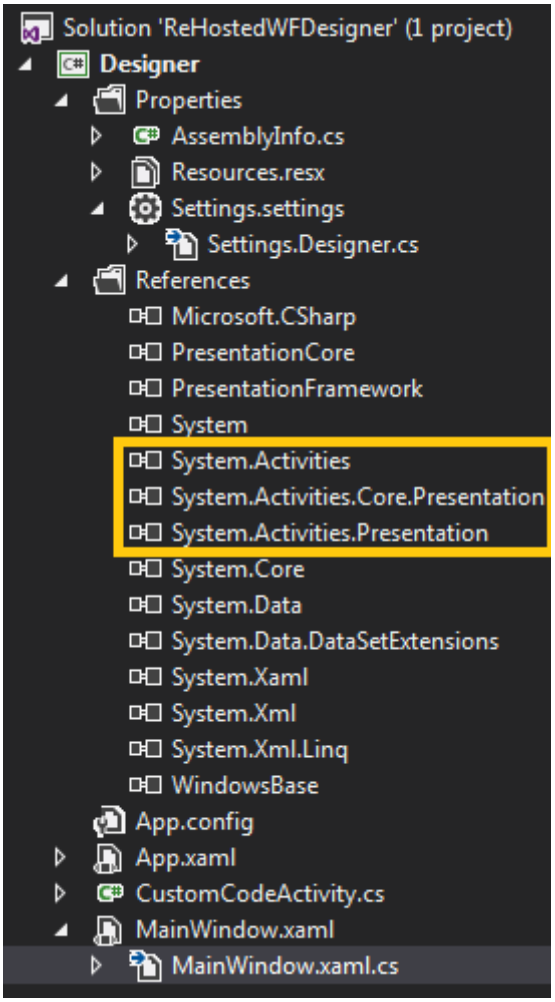
Bu temel özellikleri gerçekleştirdiğimiz takdirde elimizde basit bir **Workflow** geliştirme aracı oluşacaktır. Söz konusu aracın daha da etkin hale getirilmesi için genişletilebilir bir yapıda tasarlanması önemlidir, ancak bu örneğimizde bu biraz daha göz ardı edilecek bir unsurdur Peki bu tip bir uygulama geliştirmek için elimizde neler var bir de buna bakalım dilerseniz.

1. **WorkflowDesigner** sınıfı ile tasarım ortamının birerbir kullanılabilmesi mümkün olacaktır.
2. **ToolboxCategory, ToolboxControl, ToolboxItemWrapper** tiplerinden yararlanarak **Toolbox**oluşturulabilir ve içeriğine **Workflow** bileşenleri atılabilir.
3. **WorkflowDesigner** tipinin **PropertyInspectorView** özelliği ile, **Property** penceresinin set edilmesi sağlanabilecektir.
4. **WorkflowDesigner'** ın sunduğu **Load** ve **Save** metodları ile, bir akışın yüklenmesi veya kayıt altına alınması işlemleri gerçekleştirilebilir. Bu akışlar XAML tabanlı dosyalardan gelebileceği gibi*(ki buna göre istediğimiz yerde bir Workflow Repository' miz olabilir)* canlı çalışma zamanı Activity örnekleri de olabilir.
5. **XAML** formatında saklanacak olan **Workflow** içeriklerinin **çalışma zamanında(Runtime)** yürütülebilmesi için elimizde **ActivityXamlServices** sınıfı bulunmaktadır.
6. Yüklenen bir **Workflow'** un asenkron olarak çalıştırılabilmesi sağlamak için de **WorkflowApplication**tipinden yararlanılabilir.

Görüldüğü üzere elimizde hayal ettiğimiz gibi(?) bir **Designer'** ın geliştirilebilmesi için gerekli materyaller bulunmaktadır. Tabi ilgili düşüncenin gerçek bir ürün haline getirilmesi için epey bir çaba da sarf etmek gerekecektir.

Biz şu an için sadece giriş noktasını tasarladığımızı ve aslında **Workflow Designer'** ı **Visual Studio IDE'** si dışında çalıştırabildiğimizi ispatlarsak önemli bir aşamayı geçmiş olduğumuzu var sayabiliriz. Öyleyse gelin hiç vakit kaybetmeden işe koyulalım. Örneğimizi **Visual Studio 2012 RC**sürümü üzerinde geliştiriyor olacağız ancak **Visual Studio 2010** ortamında da test ettiğimizi ve çalıştırdığımızı ifade edebiliriz. Dolayısıyla kod parçalarını aynen **Copy-Paste** yöntemi ile 2010 ortamında da uygulatabilirsiniz.

WPF uygulaması olarak geliştireceğimiz projemizde, aşağıdaki şekilde görülen ve sarı kutucuk içerisine alınmış referansların bulunması gerekmektedir.

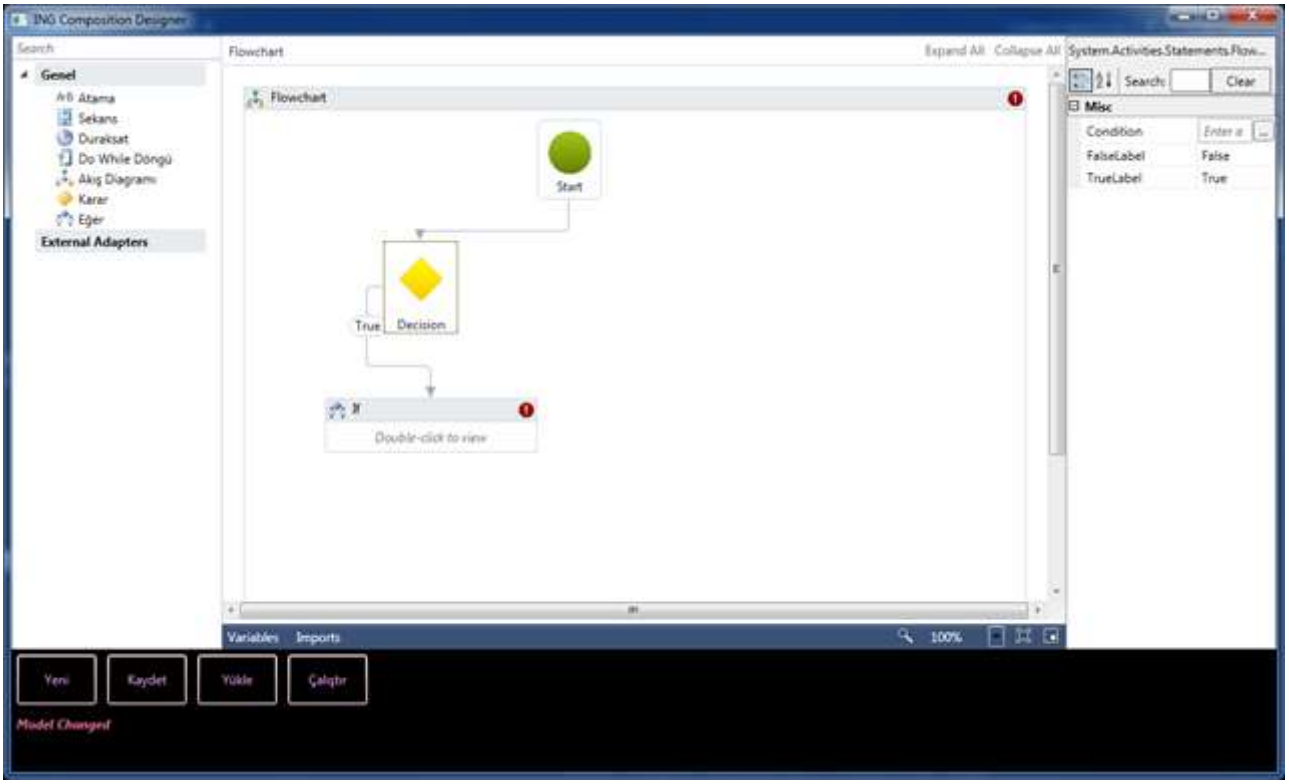


Dikkat edileceği üzere
System.Activities
System.Activities.Core.Presentation
ve **System.Activities.Presentation**
assembly' larının yüklenmesi yukarıda bahsettiğimiz temel **Designer** tipleri için gereklidir. Kendi IDE' mizin ana ekranını oluşturacak **MainWindow.xaml** içeriğini ise aşağıdaki kod parçasında görüldüğü gibi tasarlayabiliriz.

```
<Window x:Class="Designer.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="ING Composition Designer" Height="350" Width="800"
WindowState="Maximized">
   <Grid x:Name="grdScene" Background="Black">
      <Grid.RowDefinitions>
         <RowDefinition Height="10*"/>
         <RowDefinition/>
         <RowDefinition Height="1*"/>
      </Grid.RowDefinitions>
      <Grid.ColumnDefinitions>
```

```
            <ColumnDefinition/>
            <ColumnDefinition Width="4*"/>
            <ColumnDefinition/>
        </Grid.ColumnDefinitions>
        <StackPanel Orientation="Horizontal" Grid.Row="1" Width="auto"
Grid.ColumnSpan="3" Background="Black">
            <Button Background="Black" BorderBrush="Pink" Content="Yeni"
Foreground="Plum" Width="75" x:Name="btnNew" Click="btnNew_Click"
Margin="5,5,5,5" />
            <Button Background="Black" Foreground="Plum" BorderBrush="Pink"
Content="Kaydet" x:Name="btnSave" Click="btnSave_Click" Margin="5,5,5,5"
Width="75" />
            <Button Background="Black" Foreground="Plum" BorderBrush="Pink"
Content="Yükle" x:Name="btnLoad" Click="btnLoad_Click"  Margin="5,5,5,5"
Width="75" />
            <Button Background="Black" BorderBrush="Pink" Content="Çalıştır"
Foreground="Plum" Width="75" Margin="5,5,5,5" x:Name="btnRun"
Click="btnRun_Click" />
        </StackPanel>
        <StackPanel Background="Black" Orientation="Horizontal" Grid.Row="2"
Width="auto" Grid.ColumnSpan="3">
            <Label x:Name="labelStatus" Foreground="PaleVioletRed" FontWeight="Bold"
FontStyle="Italic" HorizontalAlignment="Right"/>
        </StackPanel>
    </Grid>
</Window>
```

Aslında tasarım olarak **Visual Studio IDE'** sini sadece ucundan andıran bir görselliğimiz bulunmakta Uygulamanın sol tarafında **Toolbox'** ımız, ortasında **Workflow** tasarımının yapılacağı alanımız ve en sağda bileşenlere ait **Properties** penceremiz bulunmaktadır. Şimdilik basit bir **POC(Proof of Concept)** çalışması olarak öngördüğümüzden temel fonksiyonelliklerimiz*(Load,Save,Run,New)* birer **Button** halinde pencerenin alt kısmında yer alacaktır. Olayı kafamızda daha iyi canlandırmak için uygulamamızın bitmiş halinin çalışma zamanındaki bir görüntüsüne bakalım arzu ederseniz.



Sanırım bu ekran görüntüsüne bakınca biraz daha heyecanlanmış ve iştahlanmış olabilirsiniz yanılıyor muyum? O halde kod tarafında neler yaptığımıza bir bakalım. İşte kodlarımız.

```
using System;
using System.Activities;
using System.Activities.Core.Presentation;
using System.Activities.Presentation;
using System.Activities.Presentation.Toolbox;
using System.Activities.Statements;
using System.Activities.XamlIntegration;
using System.IO;
using System.Threading;
using System.Windows;
using System.Windows.Controls;
using Microsoft.Win32;
namespace Designer
{
   // Illegal Cross Thread Exception' dan kaçmak için kullandığımız temsilci tipi
   delegate void
BindCompletionStateToLabelDelegate(WorkflowApplicationCompletedEventArgs
state);
   public partial class MainWindow
      : Window
   {
```

```
// WorkflowDesigner
private WorkflowDesigner wfDesigner;
private string currentFileName = String.Empty;
public MainWindow()
{
   InitializeComponent();
   RegisterMetadata();
   AddDesigner(new Flowchart());
   AddToolBox();
   AddPropertyInspector();
}

// Designer için Metadata içeriği register edilir
private void RegisterMetadata()
{
   DesignerMetadata dMetadata = new DesignerMetadata();
   dMetadata.Register();
}
// Bir instance' dan WorkflowDesigner' ın yüklenilmesi sağlanır. Örneğin bir
Flowchart bileşeni veya Sequence designer içerisine açılabilir.
private void AddDesigner(object instance)
{
   wfDesigner = new WorkflowDesigner();
   wfDesigner.Load(instance);
   BindDesignerEvents(wfDesigner);
   BindWfDesignerToGrid();
}
// WorkflowDesigner içeriğinin XAML tabanlı bir dosya içeriğindeki activity ile
yüklenmesini sağlar
private void AddDesigner(string xamlFileName)
{
   wfDesigner = new WorkflowDesigner();
   wfDesigner.Load(xamlFileName);
   BindDesignerEvents(wfDesigner);
   BindWfDesignerToGrid();
}
// WorkflowDesigner' ın örnek olay metodlarını yükler
private void BindDesignerEvents(WorkflowDesigner wfDesigner)
{
   wfDesigner.ModelChanged += (obj, e) =>
   {
```

```
            labelStatus.Content = "Model Changed";
        };
    }
    // WorkflowDesigner bileşeninin WPF içeriğine element olarak eklenmesini sağlar
    private void BindWfDesignerToGrid()
    {
        Grid.SetColumn(wfDesigner.View, 1);
        if (Grid.GetColumn(wfDesigner.View) == 1)
            grdScene.Children.Remove(wfDesigner.View);
        grdScene.Children.Add(wfDesigner.View);
    }
    // Toolbox içeriğinin WPF penceresine eklenmesini sağlar
    private void AddToolBox()
    {
        ToolboxControl control = GetToolboxControl();
        Grid.SetColumn(control, 0);
        grdScene.Children.Add(control);
    }
    // Örnek Workflow Component' lerinin Toolbox içeriğine eklenmesini sağlar
    private ToolboxControl GetToolboxControl()
    {
        ToolboxControl tbxControl = new ToolboxControl();
        #region Genel
        ToolboxCategory tbxStandartControls = new ToolboxCategory("Genel");
        ToolboxItemWrapper toolAssign = new ToolboxItemWrapper(typeof(Assign),
"Atama");
        ToolboxItemWrapper toolFlowDecision = new
ToolboxItemWrapper(typeof(FlowDecision), "Karar");
        ToolboxItemWrapper toolFlowchart = new
ToolboxItemWrapper(typeof(Flowchart), "Akış Diagramı");
        ToolboxItemWrapper toolIf = new ToolboxItemWrapper(typeof(If), "Eğer");
        ToolboxItemWrapper toolSequence = new ToolboxItemWrapper(typeof(Sequence),
"Sekans");
        ToolboxItemWrapper toolDelay = new ToolboxItemWrapper(typeof(Delay),
"Duraksat");
        ToolboxItemWrapper toolDoWhile = new ToolboxItemWrapper(typeof(DoWhile),
"Do While Döngü");
        tbxStandartControls.Add(toolAssign);
        tbxStandartControls.Add(toolSequence);
        tbxStandartControls.Add(toolDelay);
        tbxStandartControls.Add(toolDoWhile);
```

```
            tbxStandartControls.Add(toolFlowchart);
            tbxStandartControls.Add(toolFlowDecision);
            tbxStandartControls.Add(toolIf);
            #endregion
            #region Adapterler
            ToolboxCategory tbxAdapterControls = new ToolboxCategory("External
Adapters");
            tbxControl.Categories.Add(tbxStandartControls);
            tbxControl.Categories.Add(tbxAdapterControls);
            #endregion
            return tbxControl;
        }
        // Seçilen Workflow bileşeni veya component için gerekli özelliklerin yüklenmesini
sağlar
        private void AddPropertyInspector()
        {
            if (Grid.GetColumn(wfDesigner.PropertyInspectorView) == 2)
            {
                grdScene.Children.Remove(wfDesigner.PropertyInspectorView);
            }
            else
            {
                Grid.SetColumn(wfDesigner.PropertyInspectorView, 2);
                grdScene.Children.Add(wfDesigner.PropertyInspectorView);
            }
        }
        // Tasarlanan Workflow içeriğinin XAML uzantılı olarak kayıt edilmesini sağlar
        private void btnSave_Click(object sender, RoutedEventArgs e)
        {
            SaveFileDialog sfd = new SaveFileDialog();
            sfd.Filter = "XAML Files|*.xaml";
            sfd.InitialDirectory = Environment.CurrentDirectory;
            if (sfd.ShowDialog().Value == true)
            {
                wfDesigner.Save(sfd.FileName);
                currentFileName = sfd.FileName;
                labelStatus.Content = string.Format("{0}(Saved)", sfd.FileName);
            }
        }
        // Workflow' un designer içerisine XAML tabanlı bir içerikten okunmasını sağlar
        private void btnLoad_Click(object sender, RoutedEventArgs e)
```

```
        {
            OpenFileDialog ofd = new OpenFileDialog();
            ofd.Filter = "XAML Files|*.xaml";
            ofd.InitialDirectory = Environment.CurrentDirectory;
            if (ofd.ShowDialog().Value == true)
            {
                AddDesigner(ofd.FileName);
                AddPropertyInspector();
                currentFileName = ofd.FileName;
                labelStatus.Content = string.Format("{0}(Loaded)", ofd.FileName);
            }
        }
        // Boş bir Flowchart içerikli Workflow oluşturulmasını sağlar
        private void btnNew_Click(object sender, RoutedEventArgs e)
        {
            AddDesigner(new Flowchart());
            AddPropertyInspector();
            labelStatus.Content = "Yeni Akışı";
            currentFileName = String.Empty;
        }
        // Workflow' un Run edilmesi için kullanılır
        private void btnRun_Click(object sender, RoutedEventArgs e)
        {
            try
            {
                Activity activity = LoadActivity(currentFileName);
                ExecuteActivity(activity);
            }
            catch (Exception excp)
            {
                labelStatus.Content = string.Format("{0} nedeni ile activity başarılı bir şekilde
çalıştırılamadı", excp.Message);
            }
        }
        // Workflow icrasını gerçekleştirir
        private void ExecuteActivity(Activity activity)
        {
            AutoResetEvent aReseter = new AutoResetEvent(false);
            WorkflowApplication wfApp = new WorkflowApplication(activity);
            wfApp.Completed += ea =>
            {
```

```
            aReseter.Set();
            labelStatus
               .Dispatcher
               .Invoke(
               new BindCompletionStateToLabelDelegate(BindCompletaionStateToLabel)
               , ea
               );
         };
         wfApp.Run();
         aReseter.WaitOne();
      }
      // Illegal Cross Thread Exception' dan kaçtığımız ve Label kontrolüne diğer Thread
içerisinden değiştirmemizi sağlayan metod
      private void
BindCompletaionStateToLabel(WorkflowApplicationCompletedEventArgs state)
      {
         labelStatus.Content = string.Format(
            "{0} ID li akış için Completion State {1}"
            , state.InstanceId, state.CompletionState
            );
      }
      // XAML içeriğinden Activity bileşenini üretmek için kullanılır
      private Activity LoadActivity(string currentFileName)
      {
         Activity loadedActivity = null;
         if (!string.IsNullOrEmpty(currentFileName)
            && Path.GetExtension(currentFileName).ToUpper() == ".XAML")
         {
            loadedActivity = ActivityXamlServices.Load(currentFileName);
            labelStatus.Content = string.Format(
               "{0}({2}) run edilmek üzere {1} lokasyonundan yüklendi"
               , loadedActivity.DisplayName
               , currentFileName
               , loadedActivity.Id
               );
         }
         return loadedActivity;
      }
   }
}
```

Kodlar uzun görünmesine rağmen çok karmaşık değildir. **Designer'** ın yüklenmesi veya designer içeriğinin kayıt edilmesi gibi işlevsellikler **WorkflowDesigner** tipi üzerinden gerçekleştirilebilmektedir. **Toolbox** içeriğinin yüklenmesi veya **Properties** penceresinin üretilmesi için gerekli tipler de başta belirttiğimiz gibidir. Örnek olması açısında basit bir kaç Workflow bileşeni yüklenmiştir*(Assign, Delay vb)* Burada önemli olan bir diğer nokta da Toolbox üzerinde istediğimiz gibi kategorilendirme yapabilmemizdir.
Hatta **ToolboxItemWrapper** tipinin aşırı yükleniş yapıcılarına bakıldığında farklı şekillerde **Component** yükleyebileceğimizi de görebiliriz ki bu bize önemli bir esneklikte sağlayacaktır. **Component'** lerin herhangibir **Repository'** den alınması gibi. Bunun dışında kalan kısımlar aslında bakarsanız **Workflow** tiplerinin **Runtime'** daki davranışları için kullandığımız basit tiplerdir*(WorkflowApplication, ActivityXamlServices vb)*
Ne yazık ki yeni oluşturmak istediğimiz her Workflow örneği için WorkflowDesigner tipinin tekrardan örneklenmesi ve Toolbox ile Property penceresi içeriklerinin WPF Grid kontrolüne sıfırdan bağlanması gibi henüz aşmayı başaramadığım bazı zorunluluklar/aksaklıklar da bulunmaktadır.
Aslında bu tip **Visual Studio IDE'** si dışında bir **Designer** geliştirmenin ne gibi artıları olabileceğini de bir düşünmemiz ve masaya koymamız gerekmektedir. Bunları aşağıdaki maddeler halinde sıralayabiliriz.

1. **Visual Studio** ürünü dışında daha basit içeriğe sahip olup örneğin sadece **İş Analistlerini** hedef alan bir araca sahip olabiliriz.
2. **Workflow Foundation** içerisindeki tüm **Component** seti yerine sadece işe ve ihtiyaca yönelik bileşenlerin yer aldığı bir **Toolbox'** un ürün bazlı olarak sunulabilmesini sağlayabiliriz.
3. Söz konusu **Toolbox** içeriği **yetkilendirilebilir(Authorization)** ve ürünü kullananların pozisyonlarına göre yapabilecekleri sınırlanabilir.
4. Bir dezavantaj olarak görebileceğimiz **Debug** etme güçlüğüne karşılık ürünün **Developer** profili dışında kullanılacağı tezini öne sürebiliriz
5. Geliştirilen araç, **Visual Studio** bağımsız bir ürün olarak düşünülüp lisanslanabilir veya ücretsiz olarak şirket içi çalışmalarda değerlendirilebilir.
6. Özellikle görsel yeteneğe sahip **Workflow Activity** bileşenlerinin, bu aracın **Visual Studio** ile birlikte çalıştırılması halinde, **Designer'** a bağlandıklarındaki davranışlarının **Debug** edilmesi çok daha kolay olacaktır*(Acısını çok çektim o yüzden kulak verin bu avantajı yaban atmayın)*

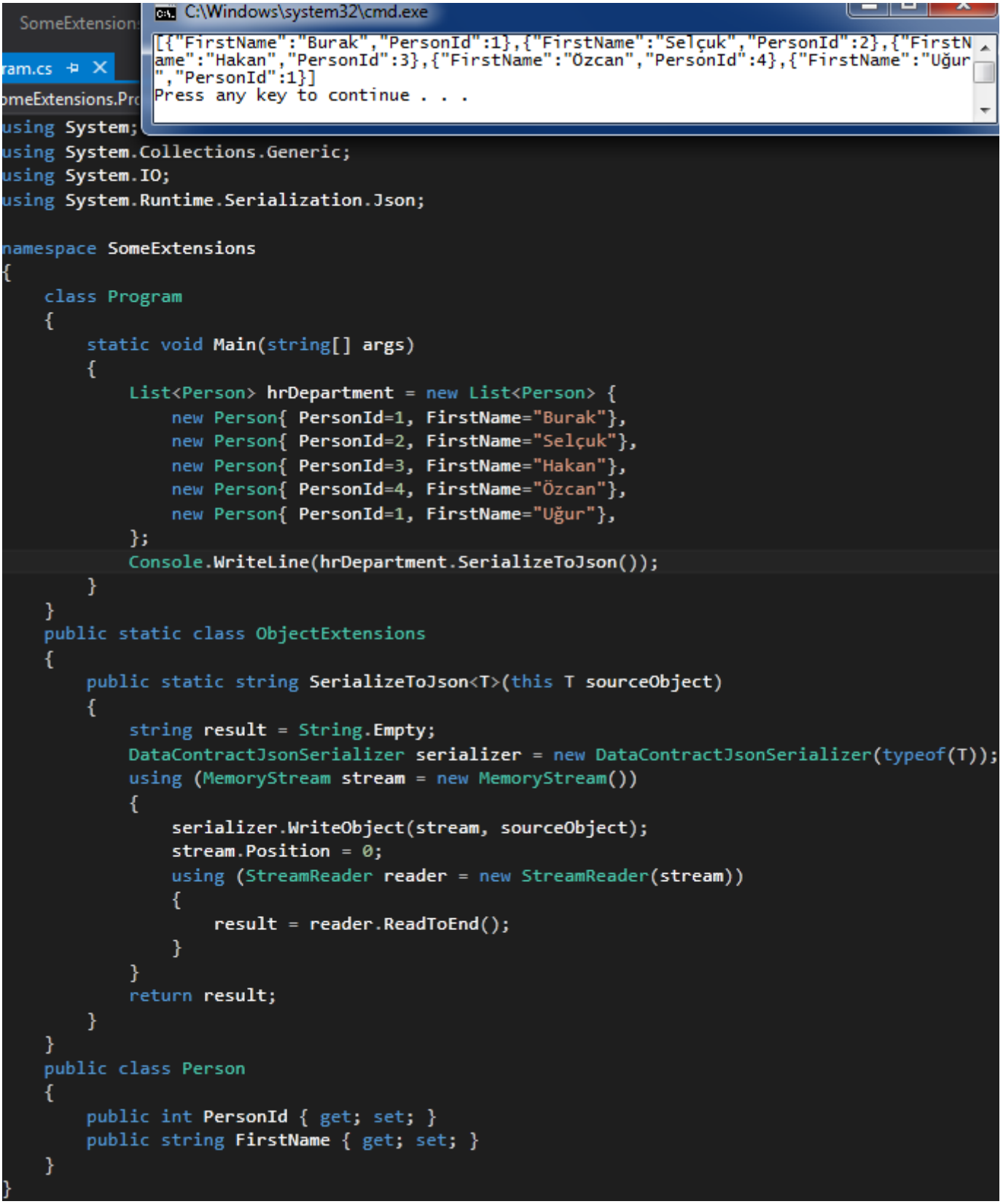
Görüldüğü üzere **Visual Studio IDE'** sinin bir parçası olarak sunulan ve kullanılan **Workflow Designer'** ın harici bir ürün haline dönüştürülmesi son derece kolaydır. Umarım vizyonunuza değer katacak bir çalışma olmuştur. Bir başka yazımızda görüşünceye dek hepinize mutlu günler dilerim.
ReHostedWFDesigner.zip (75,05 kb)
(Örnek Visual Studio 2012 RC sürümünde geliştirilmiştir ancak kodlar Visual Studio 2010 üzerinde de çalışmaktadır)

Tek Fotoluk İpucu 59–Nesneyi JSON String Olarak Serileştirmek

Perşembe, 12 Temmuz 2012 12:52

Tek Fotoluk Ipucu, Json, Datacontractjsonserializer, Extension Methods

Merhaba Arkadaşlar,
Diyelimki generic T tipine yazacağınız bir Extension metod ile, JSON formatında serileştirme işlemi yaptırmak ve serileştirme sonucunuda string olarak geriye döndürmek istiyorsunuz. Ne yaparsınız? Aşağıdaki fotoğraf bir ip ucu verebilir mi? 😉


C:\Windows\system32\cmd.exe
[{"FirstName":"Burak","PersonId":1},{"FirstName":"Selçuk","PersonId":2},{"FirstName":"Hakan","PersonId":3},{"FirstName":"Özcan","PersonId":4},{"FirstName":"Uğur","PersonId":1}]
Press any key to continue . . .
using System;
using System.Collections.Generic;
using System.IO;
using System.Runtime.Serialization.Json;
namespace SomeExtensions
{
class Program
{
static void Main(string[] args)
{
List<Person> hrDepartment = new List<Person> {
new Person{ PersonId=1, FirstName="Burak"},
new Person{ PersonId=2, FirstName="Selçuk"},
new Person{ PersonId=3, FirstName="Hakan"},
new Person{ PersonId=4, FirstName="Özcan"},
new Person{ PersonId=1, FirstName="Uğur"},
};
Console.WriteLine(hrDepartment.SerializeToJson());
}
}
public static class ObjectExtensions
{
public static string SerializeToJson<T>(this T sourceObject)
{
string result = String.Empty;
DataContractJsonSerializer serializer = new DataContractJsonSerializer(typeof(T));
using (MemoryStream stream = new MemoryStream())
{
serializer.WriteObject(stream, sourceObject);
stream.Position = 0;
using (StreamReader reader = new StreamReader(stream))
{
result = reader.ReadToEnd();
}
}
return result;
}
}
public class Person
{
public int PersonId { get; set; }
public string FirstName { get; set; }
}
}

Tek Fotoluk ipucu - 58 Derived Tipler için XElement Converter

Perşembe, 12 Temmuz 2012 09:15

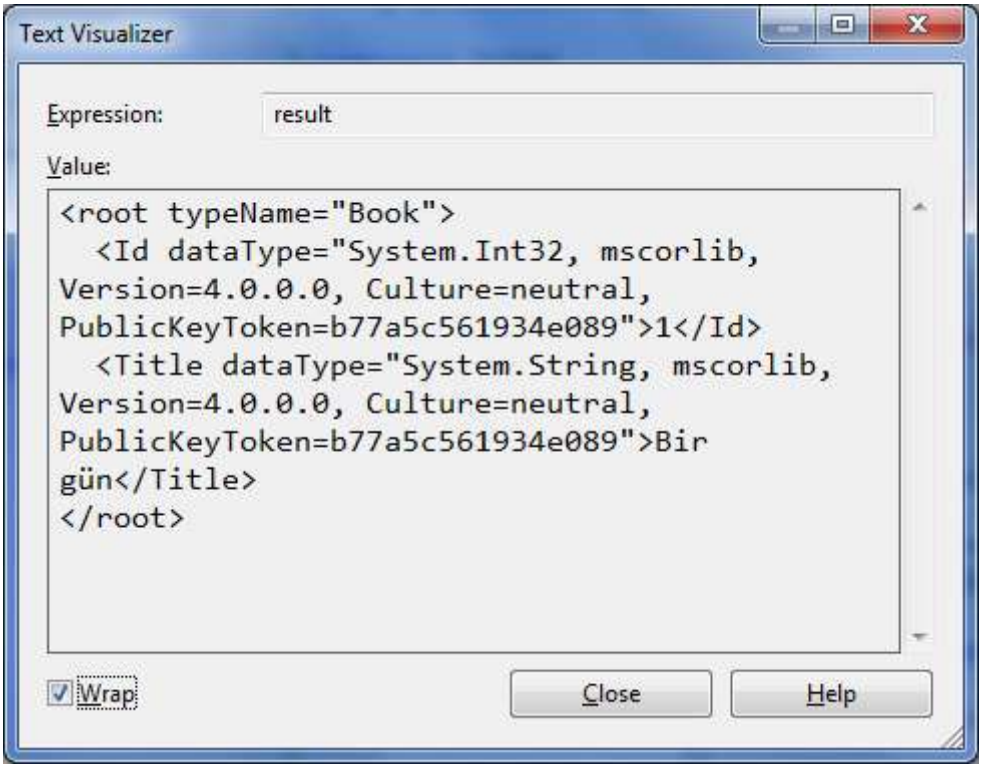
Tek Fotoluk Ipucu, Extension Method, Xelement, Reflection

Merhaba Arkadaşlar,
Farz edelim ki elimizden tonlarca POCO(Plain Old CLR Object) tip var. Hatta laf aramızda tonlarca otomatik olarak üretilmiş SQL User Defined Type karşılığı sınıf var. İstiyorsunuz ki, bu tiplerin çalışma zamanındaki canlı örnekleri, XElement tipine dönüştürülebilsin. Hatta XElement içerisinde özelliklerin adları ile birlikte .Net tarafında ki type bilgileri de detaylı olarak bulunsun. Bulunsun ki başka bir yerden tekrar ayağa kaldırabilelim. Her tip için birer Extension method' mu yazarsınız? Ya da tüm tipler için Convert ile ilişkili bir Interface implementasyonu mu? Belki de bu tipler size Oracle' dan gelmiştir de türedikleri bir base type' da vardır  İşte size fikir verecek bir ipucu daha.

```
using System;
using System.Xml.Linq;

namespace TFI59{
    class Program{
        static void Main(string[] args){
            ShopContent content = new Book { Id = 1, Title = "Bir gün" };
            XElement result=content.ConvertToXml();
        }
    }
    static class Converter{
        public static XElement ConvertToXml(this ShopContent content){
            XElement root = null;
            try
            {
                Type type = content.GetType();
                root = new XElement("root", new XAttribute("typeName",
type.Name));
                var properties = type.GetProperties();
                foreach (var property in properties)
                {
                    XElement newElement = new XElement(
                        property.Name
                        , new XAttribute("dataType",
property.PropertyType.AssemblyQualifiedName)
                        );
                    newElement.Value = property.GetValue(content,
null).ToString();
                    root.Add(newElement);
                }
            }
            catch{            }
            return root;
        }
    }
    abstract class ShopContent{}
    class Book
        :ShopContent{
        public int Id { get; set; }
        public string Title { get; set; }
    }
}
```

86 %

Debug time görüntüsü ise,

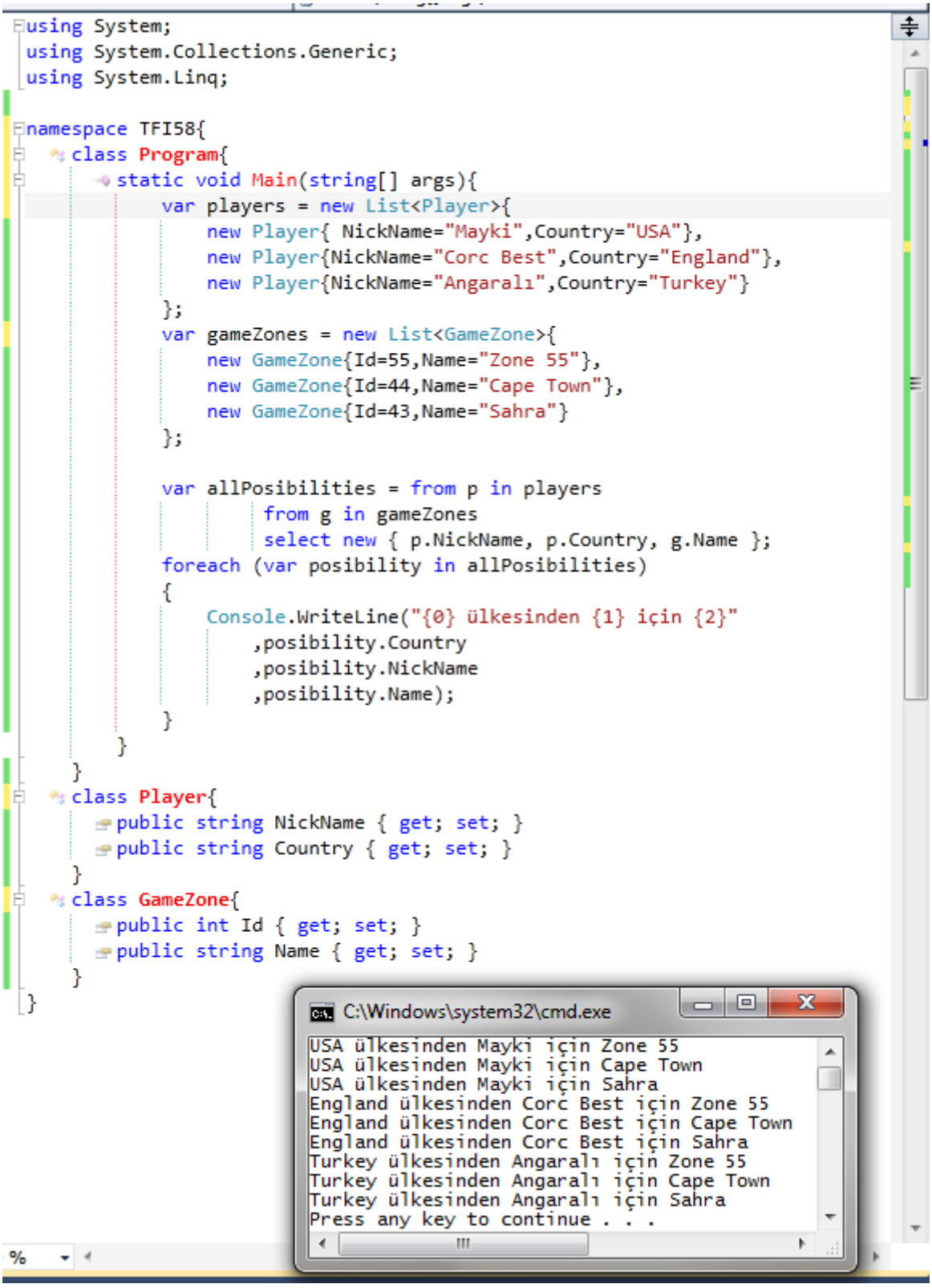

Text Visualizer
Expression:
result
Value:
<root typeName="Book">
<Id dataType="System.Int32, mscorlib,
Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089">1</Id>
<Title dataType="System.String, mscorlib,
Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089">Bir
gün</Title>
</root>
Wrap
Close
Help

Tek Fotoluk ipucu - 57 LINQ Tarafında Cross Join

Çarşamba, 11 Temmuz 2012 16:00

Tek Fotoluk Ipucu, Linq, Cross Join

Merhaba Arkadaşlar,
Elinizde iki adet nesne koleksiyonu olduğunu ve bunların veri satırı bazındaki olası eşleşmelerine ait kartezyen tablosunu elde etmek istediğinizi düşünün. Aşağıdaki gibi bir sorgu, SQL tarafındaki Cross Join etkisini LINQ ile de gerçekleştirebileceğimizi göstermektedir. Buna göre hangi ülkeden kimin hangi oyun alanlarında yer alabileceğine dair bir kartezyen çarpım içeriği elde etmiş oluruz



```
using System;
using System.Collections.Generic;
using System.Linq;

namespace TFI58{
    class Program{
        static void Main(string[] args){
            var players = new List<Player>{
                new Player{ NickName="Mayki",Country="USA"},
                new Player{NickName="Corc Best",Country="England"},
                new Player{NickName="Angaralı",Country="Turkey"}
            };
            var gameZones = new List<GameZone>{
                new GameZone{Id=55,Name="Zone 55"},
                new GameZone{Id=44,Name="Cape Town"},
                new GameZone{Id=43,Name="Sahra"}
            };

            var allPosibilities = from p in players
                                  from g in gameZones
                                  select new { p.NickName, p.Country, g.Name };
            foreach (var posibility in allPosibilities)
            {
                Console.WriteLine("{0} ülkesinden {1} için {2}"
                    ,posibility.Country
                    ,posibility.NickName
                    ,posibility.Name);
            }
        }
    }
    class Player{
        public string NickName { get; set; }
        public string Country { get; set; }
    }
    class GameZone{
        public int Id { get; set; }
        public string Name { get; set; }
    }
}
```

```
C:\Windows\system32\cmd.exe
USA ülkesinden Mayki için Zone 55
USA ülkesinden Mayki için Cape Town
USA ülkesinden Mayki için Sahra
England ülkesinden Corc Best için Zone 55
England ülkesinden Corc Best için Cape Town
England ülkesinden Corc Best için Sahra
Turkey ülkesinden Angaralı için Zone 55
Turkey ülkesinden Angaralı için Cape Town
Turkey ülkesinden Angaralı için Sahra
Press any key to continue . . .
```

Tek Fotoluk Ipucu 56 – LINQ Metodlarında String Sorgular

Pazartesi, 9 Temmuz 2012 07:45

Linq, Extension Methods, String Expressions

Merhaba Arkadaşlar,
Bazı durumlarda Entity Framework tabanlı nesne koleksiyonlarını sorgularken, Extension Method' lar içerisine gelecek olan sorgulama ifadelerinin string bazlı olarak gelmesi söz konusu olabilir. Örneğin servis metodlarının istemci tarafından parametre olarak bu tip sorgu ifadeleri aldığı sıklıkla görülmektedir. Peki ama nasıl? Bunun bir örneği var mıdır? Hani elimizin altında dursa ve bir fikir verse iyi olmaz mı? Buyrun öyleyse.

Program.cs

StringLINQExpression.Program

```
using System;

namespace StringLINQExpression
{
    class Program
    {
        static void Main(string[] args)
        {
            using (AdventureWorksEntities context
                = new AdventureWorksEntities())
            {
                var result = context.Products
                    .Where("it.Name like 'M%' && it.ListPrice>=3000")
                    .OrderBy("it.Name desc")
                    .Select("it.Name,it.Class,it.ListPrice");
                foreach (var r in result)
                    Console.WriteLine("{0}\t{1}\t{2}"
                        ,r.GetValue(0)
                        ,r["Class"].ToString()
                        ,r.GetDecimal(2).ToString()
                        );
            }
        }
    }
}
```

C:\Windows\system32\cmd.exe

```
Mountain-100 Silver, 48 H       3404,9900
Mountain-100 Black, 38  H       3379,9900
Mountain-100 Black, 42  H       3379,9900
Mountain-100 Black, 44  H       3379,9900
Mountain-100 Black, 48  H       3379,9900
Press any key to continue . . .
```

95 %

Bu koda göre arka planda hareket eden SQL sorgusu da şöyledir.

```
SELECT
1 AS [C1],
[Extent1].[Name] AS [Name],
[Extent1].[Class] AS [Class],
[Extent1].[ListPrice] AS [ListPrice]
FROM [Production].[Product] AS [Extent1]
WHERE ([Extent1].[Name] LIKE 'M%') AND ([Extent1].[ListPrice] >= 3000)
```

Bir diğer ipucunda görüşmek dileğiyle.

Tek Fotoluk İpucu 55 - Distinct ve IEqualityComparer

Çarşamba, 4 Temmuz 2012 09:00

Distinct, Extension Method, Linq, Iequalitycomparer, C#

Merhaba Arkadaşlar,
Diyelim ki elinizde kendi tipinize ait generic bir liste ve bu liste içerisinde veri bazında tekrarlı nesne örnekleri var. Örneğin bir ürün listesi ve bu liste içerisinde aynı üretici adına ait pek çok kayıt olduğunu düşünelim. Normal şartlar altında **SQL** tarafında yazacağınız basit bir sorgu ile üreticilerin adlarını tekrarsız olarak elde edebilirsiniz.
Peki **LINQ** tarafındaki **Distinct** fonksiyonunu kullanarak aynı işi yapabilir misiniz? Ufak bir **interface**' den yararlanarak bu sorunu aşmanız mümkün. Sorun diyoruz, çünkü **interface** implementasyonunu yapmassanız, **Distinct genişletme metodu(Extension Method)** yine çalışır ama beklemediğiniz şekilde

ogram.cs

DistinctForOwnTypes.BasketEqualityComparer | GetHashCode(Basket obj)

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace DistinctForOwnTypes
{
    class Basket
    {
        public int BasketId { get; set; }
        public string ProductName { get; set; }
        public string Producer { get; set; }
    }
    class BasketEqualityComparer
        :IEqualityComparer<Basket>
    {
        public bool Equals(Basket x, Basket y)
        {
            if (Object.ReferenceEquals(x, y))
                return true;

            if (Object.ReferenceEquals(x, null)
                || Object.ReferenceEquals(y, null))
                return false;

            return x.Producer == y.Producer;
        }

        public int GetHashCode(Basket obj)
        {
            if (Object.ReferenceEquals(obj, null))
                return 0;
            int hashProductName = obj.Producer == null ? 0 : obj.Producer.GetHashCode();
            return hashProductName;
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            List<Basket> baskets = new List<Basket>
            {
                new Basket{ BasketId=1,Producer="Sony",ProductName  ="PSP2"},
                new Basket{ BasketId=1,Producer="Sony",ProductName  ="PSP2"},
                new Basket{ BasketId=1,Producer="Canon",ProductName  ="EOS450D"},
                new Basket{ BasketId=1,Producer="Nikkon",ProductName  ="D90"},
                new Basket{ BasketId=1,Producer="Sony",ProductName  ="Vaio 17inch"},
                new Basket{ BasketId=1,Producer="Sony",ProductName  ="Vaio 17inch"},
            };

            var result = (from b in baskets
                          select b)
                         .Distinct(new BasketEqualityComparer());

            foreach (var r in result)
                Console.WriteLine(r.Producer);
        }
    }
}
```

```
C:\WINNT\system32\cmd.exe
Sony
Canon
Nikkon
Press any key to continue . . .
```

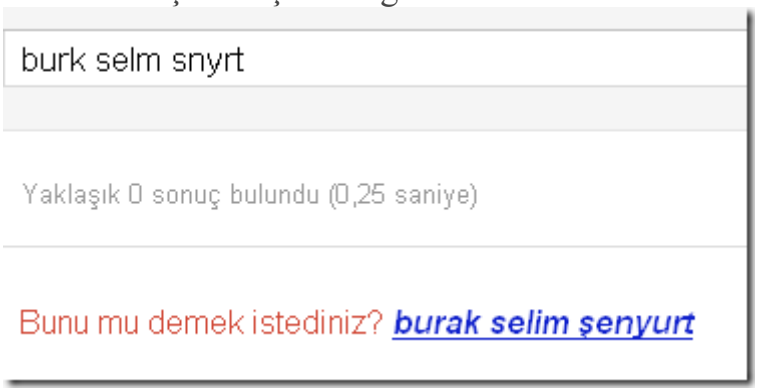
00 %

Levenshtein Distance Algoritması

Pazartesi, 2 Temmuz 2012 08:05

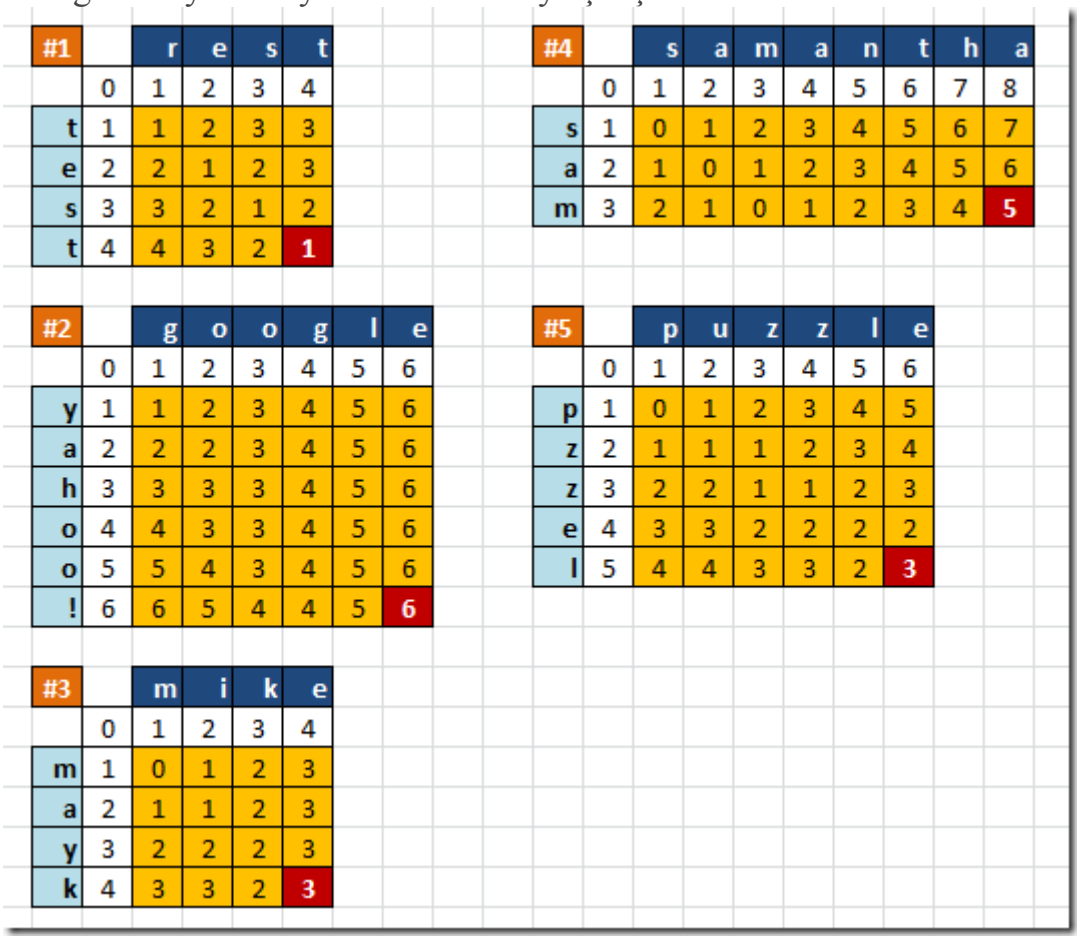
C#, Levenshtein Distance, Algoritma, Extension Methods

Merhaba Arkadaşlar,
Bir süredir yazılım dünyasında sıklıkla kullanılan basit algoritmalara merak salmış durumdayım. Bazıları kafayı yedirtecek cinsten olsalarda arada sırada bunları değerlendirmekte ve paslanan dimamızı açmaya çalışmakta yarar olduğu kanısındayım.
Aslına bakarsanız bilgisayar bilimlerinde uygulanabilen, gerçekten çok işe yarayan ve onları keşfedenleri saygıyla hatırlamamız gereken algoritmalar mevcut. Örneğin bunlardan birisi olanLevenshtein Distance algoritması ve mucidi **Vladimir Levenshtein**
Bu algoritma bizlere, özellikle arama motorlarında da kullanılabilen bir model sunmaktadır. Son kullanıcıların aradıkları kelimeleri tam olarak belirleyemedikleri veya kestiremedikleri durumlarda, öneri olarak sunulan kelimelerin tespit edilmesi sırasında ele alınan bir algoritmadır. Örneğin ben**Google** sitesindeki arama kutucuğunda kendi ismimi eksik karakterler ile yazdığımda, **google** daha önceden yapmış olduğu indekslenmiş içeriklere göre bir öneri de bulunmuştur*(Bunu mu demek istediniz kısmı)* Aşağıdaki şekilde bu durum açık bir şekilde görülmektedir.



Arama motorları dışında, özellikle imla kontrolü yapan uygulamalarda da*(Söz gelimi **Microsoft Outlook** veya **Microsoft Word'** ün **Spell Checking** mekanizmalarında)* bu algoritmanın kullanımına sıklıkla şahit olmaktayız.
Biz bu yazımızda söz konusu algoritmanın kullanılması için gerekli olan temel fonksiyonu, sıklıkla yaptığımız üzere bir **Extension Method** olarak geliştirmeye ve test etmeye çalışıyor olacağız. Ancak kodlama kısmına geçmeden önce algoritmanın nasıl çalıştığına ve işlediğine bakmamızda yarar olacağı kanısındayım.
Aslında algoritma temel olarak iki kelimenin birbirlerine olan benzerliklerini ölçümlemek amacıyla kullanılmaktadır. Sonuç tek bir sayısal değerdir ve iki kelimeden birinin diğerine

dönüştürülebilmesi için gerekli olan işlem sayısını ya da maliyetini vermektedir. Çok doğal olarak bu sayınının düşük olması arzu edilen neticedir. Nitekim daha az değişiklik anlamına gelmektedir. Çok doğal olarak bir kelimenin, bir öneri kelime kümesi içerisindekiler ile karşılaştırılması sonucu ortaya çıkan sayısal değerlerden en küçüğü veya küçükleri, sonuca ulaşılması ve doğru önerilerde bulunulması açısından önemlidir.
Peki bu yakınlık değeri nasıl hesaplanmaktadır? Bunun için kelimeler arası iki boyutlu bir matris dizisi kullanılır. Lakin söz konusu matrisin içereceği değerlerin tespiti çok da kolay değildir. Dilerseniz aşağıdaki Excel görüntüsünde yer alan örneklemelere bir bakalım ve algoritmayı daha yakından tanımaya çalışalım.



#1		r	e	s	t
	0	1	2	3	4
t	1	1	2	3	3
e	2	2	1	2	3
s	3	3	2	1	2
t	4	4	3	2	1

#4		s	a	m	a	n	t	h	a
	0	1	2	3	4	5	6	7	8
s	1	0	1	2	3	4	5	6	7
a	2	1	0	1	2	3	4	5	6
m	3	2	1	0	1	2	3	4	5

#2		g	o	o	g	l	e
	0	1	2	3	4	5	6
y	1	1	2	3	4	5	6
a	2	2	2	3	4	5	6
h	3	3	3	3	4	5	6
o	4	4	3	3	4	5	6
o	5	5	4	3	4	5	6
!	6	6	5	4	4	5	6

#5		p	u	z	z	l	e
	0	1	2	3	4	5	6
p	1	0	1	2	3	4	5
z	2	1	1	1	2	3	4
z	3	2	2	1	1	2	3
e	4	3	3	2	2	2	2
l	5	4	4	3	3	2	3

#3		m	i	k	e
	0	1	2	3	4
m	1	0	1	2	3
a	2	1	1	2	3
y	3	2	2	2	3
k	4	3	3	2	3

Bu grafikte, 5 farklı örnek ile 10 kelimenin birbirleri ile yakınlıklarının **Levensthein Distance**algoritmasına göre nasıl hesap edildiği gösterilmektedir. İlk olarak **rest** kelimesinin **test** kelimesi ile olan yakınlığı bulunmaya çalışılmıştır. Aslına bakarsanız bu iki kelime arasında sadece **1 işlem**yaparak sonuca ulaşılabilir. Bu işleme göre **rest kelimesindeki r harfi yerine, t harfinin gelmesi** yeterlidir. Matris içerisinde yer alan sayılar o andaki sütuna veya satıra kadar olan harf topluluklarının birbirleri ile eş düşmeleri için gerekli işlem sayılarını içermektedir.
Şimdi de **google** ve **yahoo!** kelimelerinin yakınlık hesabını göz önüne alalım. Normal şartlarda iki kelime içerisinde ortak olan **2 "o"** harfi bulunmaktadır ancak yerleri farklıdır. Diğer harfler ise zaten birbirlerinde yoktur. Bu nedenle **6 işlemlik bir operasyon yapılması** gerekmektedir.

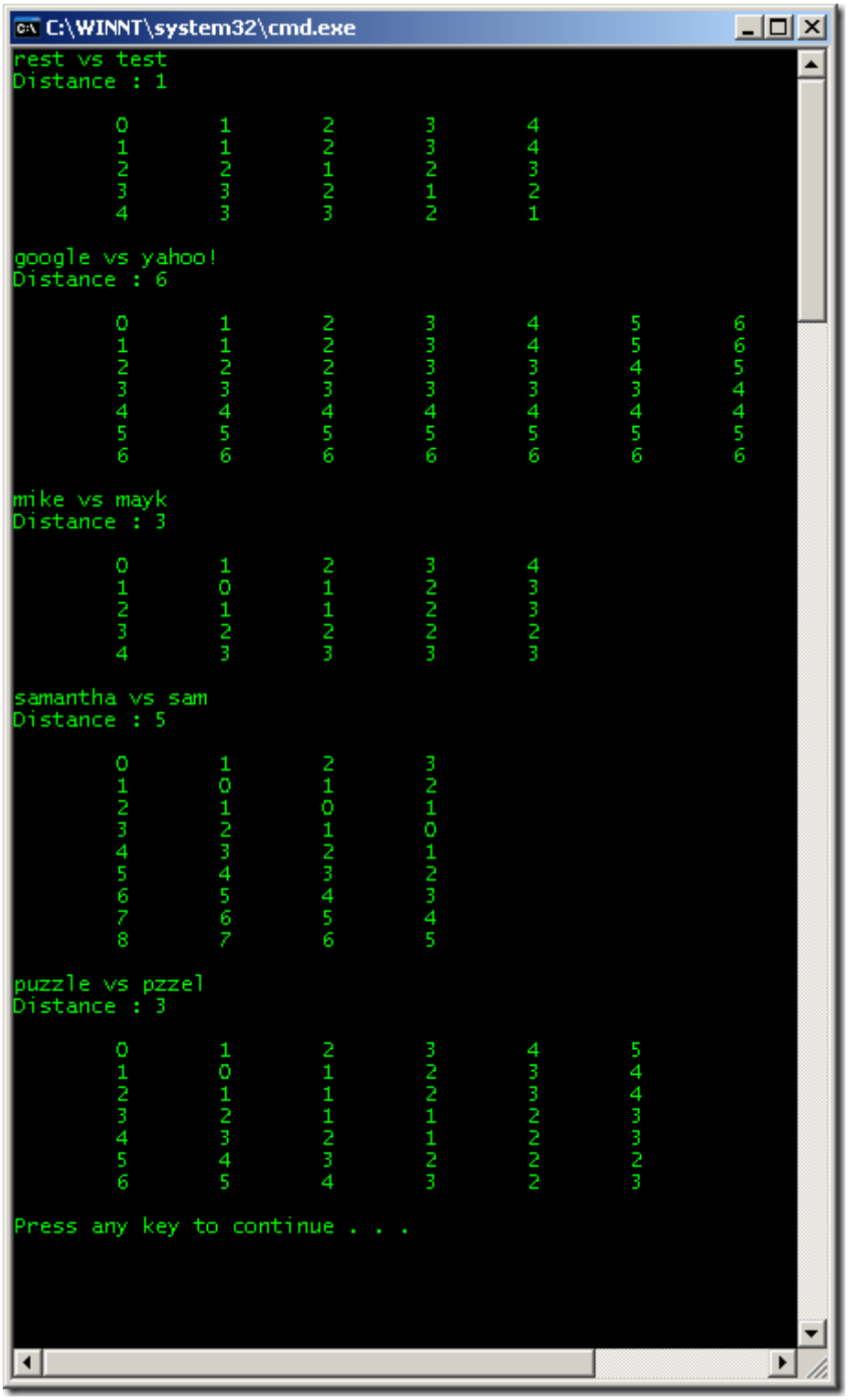
Peki sayılar tam olarak nasıl yerleştirilmekte veya okunmaktadırlar?
Hemen **Samantha** ile **Sam'** in karşılaştırılmasını ele alalım. Şimdi **0 indisli** olacak şekilde **1nci sütun** ve **1inci satırı** göz önüne alalım. 1nci sütunda **"s"** harfi ve 1nci satırda yine **"s"** harfi bulunmaktadır. Dolayısıyla o anki karşılaştırmada, her iki harfte aynı olduğunda bir işlem yapılmasına gerek yoktur. Dolayısıyla **işlem maliyeti 0dır.**
Şimdi **2ncü sütuna** ve **1nci satıra** bakalım. 2nci sütuna kadar olan kısımda **"sa"**hecesi oluşmuştur. Satır tarafında ise sadece **"s"** harfi bulunmaktadır. Dolayısıyla eşleştirme için satır kısmındaki **"s"** harfine bir de **"a"** harfinin eklenmiş olması gerekir. Ki bu da **1 işlem maliyeti**olarak ifade edilmektedir.
Durumu biraz daha öteleyelim. 5 numaralı örnekte yer alan **puzzle** ve **pzzel** kelimelerinin karşılaştırılmasında **5nci sütun** ve **4ncü satıra** bakalım. 5nci sütuna kadar **puzz** kelimesi 4ncü satıra kadar da **pzz** kelimesi söz konusudur.**pzz**' un **puzz** kelimesine benzemesi için araya bir **"u"**harfinin konulması yeterlidir. Diğer kısımlar satır ve sütun bazında da eşleşmektedir. Bu yüzden buradaki **işlem maliyeti değeri 1** dir. Ancak yine 0 indisli baktığımızda ve **7nci sütun** ve **6ncı satıra** kadar olan kısımda **puzzle** ve **pzzel** kelimeleri göz önüne alındığında ise; pzzel' dan puzzle'a geçmek istenildiğinde ilk olarak araya bir **"u"** harfi konulur.
p**u**zzel
Ardından **"el"** hecesinde **e' nin l yerine, l' nin e yerine** geçmesi gerekir.
p**u**zz**le**
Dolayısıyla toplamda **3 işlem maliyeti** söz konusu olmuştur.
Bu algoritma gereği iki kelime arasındaki yakınlık derecesi, matrisin sağ alt hücresindeki sayısal değer ile ifade edilmektedir. Buna göre **puzzle** ile **pzzel** kelimeleri arasındaki mesafe **3 işlemoperayonu** ile ölçülürken, bu **Samantha** ve **Sam** kelimeleri arasında **5** işlemlik bir maliyet oluşması söz konusudur*(Samantha' dan **antha** kısmının atılması nedeni ile 5 işlemlik bir maliyet oluşmaktadır)*
Algoritmayı biraz kavradığımıza göre dilerseniz bunu **C#** tarafında bir **Extension Method** içerisine dahil edelim ve test uygulamamıza çıkalım. Bu amaçla aşağıdaki örnek Console uygulamasını göz önüne alabiliriz.

```
using System;
namespace UsingLevenshtein
{
   class Program
   {
      static void Main(string[] args)
      {
         TestMethod("rest", "test");
         TestMethod("google", "yahoo!");
         TestMethod("mike", "mayk");
         TestMethod("samantha", "sam");
```

```
            TestMethod("puzzle", "pzzel");
        }
        private static void TestMethod(string Source,string Target)
        {
            int[,] matrix3 = new int[Source.Length, Target.Length];
            int distance3 = Source.FindLevenshteinDistance(Target, out matrix3);
            Console.WriteLine("{0} vs {1}\nDistance : {2}\n",Source,Target, distance3);
            WriteToConsole(matrix3);
        }
        static void WriteToConsole(int[,] Matrix)
        {
            for (int i = 0; i < Matrix.GetLength(0); i++)
            {
                for (int j = 0; j < Matrix.GetLength(1); j++)
                {
                    Console.Write("\t{0}  ", Matrix[i, j]);
                }
                Console.WriteLine();
            }
            Console.WriteLine();
        }
    }
    public static class StringExtensions
    {
        // Genişletme metodu, karşılaştırma matrisini de out parametresi olarak
döndürmektedir.
        public static int FindLevenshteinDistance(this string Source, string Target,out
int[,] Matrix)
        {
            int n = Source.Length;
            int m = Target.Length;
            Matrix = new int[n + 1, m + 1]; // Hesaplama matrisi üretilir. 2 boyutlu matrisin
boyut uzunlukları ise kaynak ve hedef metinlerin karakter uzunluklarına göre set edilir
            if (n == 0) // Eğer kaynak metin yoksa zaten hedef metnin tüm harflerinin değişimi
söz konusu olduğundan, hedef metnin uzunluğu kadar bir yakınlık değeri mümkün olabilir
                return m;
            if (m == 0) // Yukarıdaki durum hedefin karakter içermemesi halinde de geçerlidir
                return n;
            // Aşağıdaki iki döngü ile yatay ve düşey eksenlerdeki standart 0,1,2,3,4...n
elemanları doldurulur
            for (int i = 0; i <= n;i++)
```

```
            Matrix[i, 0] = i;

         for (int j = 0; j <= m; j++)
            Matrix[0, j] = j;
         // Kıyaslama ve derecelendirme operasyonu yapılır
         for (int i = 1; i <= n; i++)
            for (int j = 1; j <= m; j++)
            {
               int cost = (Target[j - 1] == Source[i - 1]) ? 0 : 1;
               Matrix[i, j] = Math.Min(Math.Min(Matrix[i - 1, j] + 1, Matrix[i, j - 1] + 1),
Matrix[i - 1, j - 1] + cost);
            }
         return Matrix[n, m]; // sağ alt taraftaki hücre değeri döndürülür
      }
   }
}
```

Uygulamamız içerisinde dikkat edeceğiniz üzere **Excel** tablosunda yer alan kelimelere ait bir test işlemi gerçekleştirilmektedir.**FindLevenshteinDistance** isimli metodumuz bir genişletme fonksiyonu olarak herhangibir **string** tipine uygulanabilecek şekilde tasarlanmıştır. Bununla birlikte söz konusu metod hem **Levenshtein Distance** matrisini, hemde yakınlık derecesini döndürmektedir. Uygulama içerisinde kelimeler arası testi kolaylaştırmak adına **TestMethod** isimli bir fonksiyon da ele alınmıştır. Programın çalışma zamanındaki çıktısı ise aşağıdaki gibi olacaktır.



Artık bundan sonrasında yapılması gereken, bir text kutucuğuna girilen metni, bir metin kümesi içerisinde söz konusu algoritmaya göre aramak ve yakınlık derecesi, bir başka deyişle operasyon işlem maliyeti en düşük olan kelime veya kelimeleri kullanıcıya sunmaya çalışmaktan ibarettir. Dilerseniz bu konuyu bir düşünün ve uygulamaya çalışın Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

UsingLevenshtein.zip (15,85 kb)

Custom Activity Designer Geliştirmek

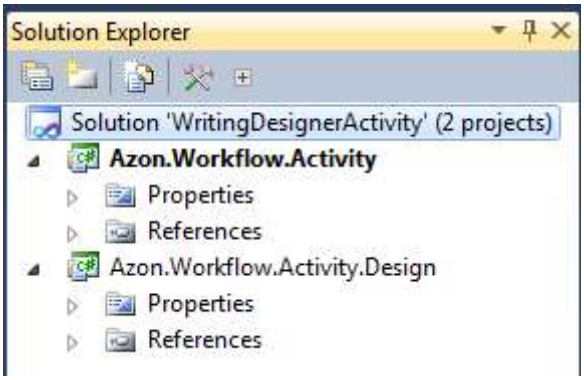
Salı, 12 Haziran 2012 16:00

Workflow Foundation, Activity Designer, Native Activity, Custom Workflow Component, Xaml, Xaml Data Binding, Ivalueconverter, Observablecollection,Visual Studio 2010 Workflow Designer

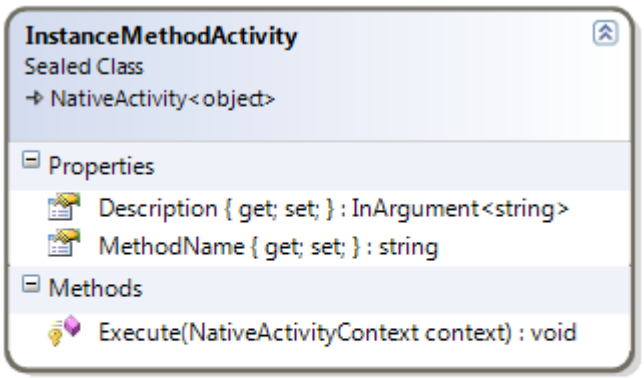
Merhaba Arkadaşlar,

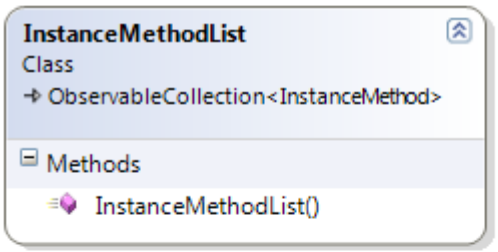
İnsanın kendisini en çok geliştireceği yer gerçek çalışma sahaları/ortamlarıdır. Ortaya konan ihtiyaçlar ne zaman ki sizin kullanmakta olduğunuz araçların(Tools) sınırlarını zorlamaya başlar, bu noktadan itibaren içerisine gireceğiniz her çeşit mücadele size inanılmaz derece tecrübe ve bilgi katacaktır. Tabi bu know-how bilgisini saklayabilir, kendiniz için dökümante edebilir veya kuralları çerçevesinde paylaşabilirsiniz 😉 Geçtiğimiz günlerde**Workflow Foundation** tarafında bir **Component Set'** in geliştirilmesi üzerine açılan **POC(Proof of Concept)** projesinde görev aldım. Bu anlamda yoğun bir şekilde **Custom Activity Designer**konusu ile yakın ilişki içerisinde yer almam gerekti. **Workflow Foundation'** ın bileşen seti her ne kadar geniş bir yelpazeye sahip olsa da, özellikle uygulama geliştiricilerin hızlı bir şekilde Workflow*(Flow Chart, Sequential vb)* tasarlaması gerektiği durumlarda, işleri kolaylaştıracak**Component** setlerinin üretilmesi son derece önemlidir. Ne varki **XAML** tabanlı çalışan**Activity Designer** örnekleri, **Visual Studio IDE'** si ile pek kardeşçe yaşamamaktadır*(Bu durumun Visual Studio 2012' de devam etmediğini umuyorum)*. Dikkat edilmesi gereken pek çok nokta ve ip ucu bulunmakta. Dilerseniz ne demek istediğimi örnek bir senaryo üzerinden görmeye çalışalım.

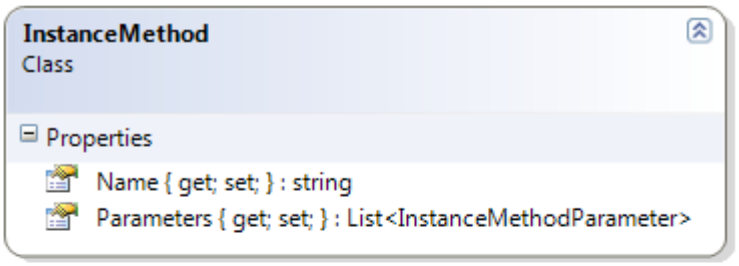
Senaryomuzda metod adlarını ve bu fonksiyonlara bağlı parametre listelerini gösteren basit bir**Workflow Activity** bileşenini tasarlamaya çalışıyor olacağız. Bileşenimiz standart bir **Code/Native Activity'** den farklı olarak görsel arayüze sahip olacak ve **Visual Studio IDE'** si içerisinden de kullanılabilecek. Bir başka deyişle **ToolBox** sekmesinden **designer** ortamına sürükleyip bıraktığımızda, **IDE** kullanıcısı ile etkileşim içerisinde olacak. Dolayısıyla **Activity Designer** tipini ele alacağımız bir örnek üzerinde çalışıyor olacağız. İlk olarak projelerimizi oluşturarak işe başlayalım. Bu anlamda **Solution** içeriğini aşağıdaki şekilde görüldüğü gibi tasarlayabiliriz.

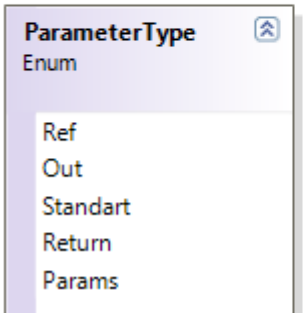


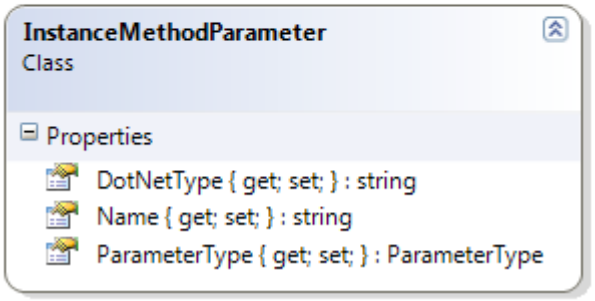
Solution yapısı oldukça önemlidir. **Activity** projesi **NativeActivity** türevli tipleri barındırıyor iken,**Design** kütüphanesinde sadece görsel tasarımlar yer alacaktır. **Azon.Workflow.Activity** projesi**Activity Library** tipinden iken **Azon.Workflow.Activity.Design**, **Activity Designer Library**tipindendir. Burada **Visual Studio 2010 IDE'** sinin beklediği bir isimlendirme standartı bulunmaktadır. Buna göre, **Component'** in görsel arayüzünün tasarlanacağı kütüphane adının mutlaka **Design** kelimesi ile bitmesi gerekmektedir*(Bu bilgiyi bulmak oldukça fazla vakit kaybına neden oldu. Ben en başından söylemek istiyorum)*
Yolumuza **Native Activity** bileşenimizi geliştirerek devam edelim. **Azon.Workflow.Activity**kütüphanesi içerisinde aşağıdaki sınıf diagramında görülen tipleri üretiyor olacağız. Senaryomuza göre bileşenimiz, kaynak bir listede yer alan metod adlarını ve bunlara ait parametreleri gösteriyor olacak. İlk hedefimiz bu.











Şimdi tiplerimiz içeriklerini biraz değerlendirelim.

InstanceMethodActivity.cs

```
using System.Activities;
namespace Azon.Workflow.Activity
{
   public sealed class InstanceMethodActivity
      :NativeActivity<object>
   {
      public InArgument<string> Description { get; set; }
      public string MethodName { get; set; }
      protected override void Execute(NativeActivityContext context)
      {
         //TODO@Burak burada bir takım kodlar işletilir
      }
   }
}
```

InstanceMethodActivity, **türetilemeyen(Sealed)** ve **NativeActivity<object>** türevli bir tiptir.**CodeActivity** türevli tiplere benzer olarak, çalışma zamanındaki işlerini **Execute** metodu içerisinde icra etmektedir. Örneğimizde söz konusu **Activity** bileşeni için herhangibir **Runtime** işlemi uygulatmıyor olacağız. Asıl hedefimiz **Visual Studio 2010 IDE'** sinde **Design Time Support'** unu sağlayabilmektir. Tipimizin içerisinde **InArgument<string>** tipinden **Description** ve **string**türünden **MethodName** isimli iki **özellik(Property)** yer almaktadır. **Designer** tarafında işimize yarayacak olan sınıflar ise **InstanceMethod, InstanceMethodParameter, ParameterType(Enum sabiti)** ve **InstanceMethodList'** tir.

```
namespace Azon.Workflow.Activity
{
   using System.Collections.Generic;
   public class InstanceMethod
   {
      public string Name { get; set; }
      public List<InstanceMethodParameter> Parameters{ get; set; }
   }
}
```

InstanceMethod, aslında bir metodun adını ve parametrik yapısını taşımak üzere tasarlanmış bir**POCO(Plain Old Clr Object)** tipidir. **Parameters** özelliği **InstanceMethodParameter** tipinden**generic** bir **List** koleksiyonudur ve ilgili sınıfın içeriği de aşağıdaki gibidir.

```
namespace Azon.Workflow.Activity
{
   public class InstanceMethodParameter
   {
      public string Name { get; set; }
      public string DotNetType { get; set; }
      public ParameterType ParameterType{ get; set; }
   }
}
```

Bu tip içerisinde ise sembolik olarak metod parametrelerine ait çeşitli bilgiler yer almaktadır. Örneğin parametrenin adı, **.Net Framework Common Type System** deki karşılığı gibi. **ParameterTypeenum** sabiti ile de ilgili parametrenin ne çeşitte olduğu belirtilmektedir.

```
namespace Azon.Workflow.Activity
{
   public enum ParameterType
   {
      Ref,
```

```
        Out,
        Standart,
        Return,
        Params
    }
}
```

Bu kütüphane içerisindeki en önemli tip ise **ObservableCollection<InstanceMethod>** türevli olan **InstanceMethodList'** dir.

```
namespace Azon.Workflow.Activity
{
    using System.Collections.Generic;
    using System.Collections.ObjectModel;
    public class InstanceMethodList
        :ObservableCollection<InstanceMethod>
    {
        public InstanceMethodList()
        {
            Add(new InstanceMethod
            {
                Name = "Sum",
                Parameters = new List<InstanceMethodParameter>{
                    new InstanceMethodParameter{ Name="X", DotNetType="System.Int32",
ParameterType= ParameterType.Standart},
                    new InstanceMethodParameter{ Name="Y", DotNetType="System.Int32",
ParameterType= ParameterType.Standart},
                    new InstanceMethodParameter{ Name="Result",
DotNetType="System.Int32", ParameterType= ParameterType.Return}
                }
            });
            Add(new InstanceMethod
            {
                Name = "TotalSum",
                Parameters = new List<InstanceMethodParameter>{
                    new InstanceMethodParameter{ Name="Values",
DotNetType="System.Int32[]", ParameterType= ParameterType.Params},
                    new InstanceMethodParameter{ Name="Result",
DotNetType="System.Int32", ParameterType= ParameterType.Return}
                }
            });
            Add(new InstanceMethod
            {
```

```
            Name = "CallSp",
            Parameters = new List<InstanceMethodParameter>{
               new InstanceMethodParameter{ Name="SpName",
DotNetType="System.String", ParameterType= ParameterType.Standart},
               new InstanceMethodParameter{ Name="ResultSet",
DotNetType="System.Data.DataTable", ParameterType= ParameterType.Ref}
            }
         });
      }
   }
}
```

Bu tip aslında **Activity Designer'** ın **XAML** tabanlı içeriğinde ele alacağımız **Data Binding** işlemleri için kullanılmaktadır. **ObservableCollection** türevli olmasının sebebi de budur. Amacımız tipin kendisini **ComboBox** ve **DataGrid** kontrollerine bağlamaktır. **Yapıcı(Constructor)** metod içerisinde, örnek metod bilgilerinin eklendiği görülmektedir. Elbetteki bir gerçek hayat senaryosunda ilgili içeriklerin farklı veri ortamlarından tedarik edilmesi de düşünülebilir. Örneğin bu bilgileri bir servis üzerinden veya doğrudan erişilebilen ve **InProc** modda kullanabildiğimiz bir **Assembly**içerisinden de getirtebiliriz.

Gelelim bileşenimizin arayüzünü tasarlayacağımız **Activity Designer** öğesine.**Azon.Workflow.Activity.Design** kütüphanesinde oluşturacağımız**InstanceMethodActivityDesigner.xaml** tipinin içeriğini aşağıdaki gibi tasarlayabiliriz.

```
<sap:ActivityDesigner
x:Class="Azon.Workflow.Activity.Design.InstanceMethodActivityDesigner"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:s="clr-
namespace:System;assembly=mscorlib"
   xmlns:sap="clr-
namespace:System.Activities.Presentation;assembly=System.Activities.Presentation"
   xmlns:sapv="clr-
namespace:System.Activities.Presentation.View;assembly=System.Activities.Presentation"
   xmlns:Model="clr-
namespace:System.Activities.Presentation.Model;assembly=System.Activities.Presentation
"
   xmlns:sapc="clr-
namespace:System.Activities.Presentation.Converters;assembly=System.Activities.Present
ation"
   xmlns:activity="clr-
namespace:Azon.Workflow.Activity;assembly=Azon.Workflow.Activity"
   mc:Ignorable="d" xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
```

```
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006">
   <sap:ActivityDesigner.Resources>
      <ResourceDictionary x:Uid="ResourceDictionary_0">
         <sapc:ModelToObjectValueConverter
x:Key="ModelToObjectValueConverter" />
         <ObjectDataProvider x:Key="dsInstanceMethods" ObjectType="{x:Type
activity:InstanceMethodList}">
         </ObjectDataProvider>
         <DataTemplate x:Key="Collapsed">
            <StackPanel Orientation="Horizontal">
               <TextBlock VerticalAlignment="Center" Margin="5" Text="Method Caller"
/>
            </StackPanel>
         </DataTemplate>
         <DataTemplate x:Key="Expanded">
            <Grid>
               <Grid.RowDefinitions>
                  <RowDefinition/>
                  <RowDefinition/>
                  <RowDefinition/>
                  <RowDefinition/>
               </Grid.RowDefinitions>
               <Grid.ColumnDefinitions>
                  <ColumnDefinition/>
               </Grid.ColumnDefinitions>
               <TextBlock Text="Metodlar" Grid.Row="0"/>
               <ComboBox x:Name="cmbInstanceMethods" Grid.Row="1"
ItemsSource="{Binding Source={StaticResource dsInstanceMethods}}"
DisplayMemberPath="Name" IsSynchronizedWithCurrentItem="True" />
               <TextBlock Text="Metod Parametreleri" Grid.Row="2"/>
               <DataGrid x:Name="grdMethodParameters" Grid.Row="3"
IsSynchronizedWithCurrentItem="True" ItemsSource="{Binding
Source={StaticResource dsInstanceMethods}, Path=Parameters}"/>
            </Grid>
         </DataTemplate>
         <Style x:Key="ExpandOrCollapsedStyle" TargetType="{x:Type
ContentPresenter}">
            <Setter Property="ContentTemplate" Value="{DynamicResource Expanded}" />
            <Style.Triggers>
               <DataTrigger Binding="{Binding Path=ShowExpanded}" Value="true">
                  <Setter Property="ContentTemplate" Value="{DynamicResource
```

```
Collapsed}" />
                </DataTrigger>
              </Style.Triggers>
            </Style>
        </ResourceDictionary>
    </sap:ActivityDesigner.Resources>
    <Grid>
        <ContentPresenter Style="{DynamicResource ExpandOrCollapsedStyle}"
Content="{Binding}" />
    </Grid>
</sap:ActivityDesigner>
```

Vuuuuu!!! 😬 Biraz korkutucu bir içerik gibi görünebilir. Ama korkmayın. Tek tek açıklamaya çalışalım.

Herşeyden önce bileşenimiz içerisinde bazı **Static Resource'** lar tanımlandığı görülmektedir.**dsInstanceMethods** isimli **ObjectDataProvider**, **InstanceMethodList** isi mli sınıfa bağlanmaktadır. Dolayısıyla **XAML** içerisinde yer alan bileşenler bu veri kaynağına bağlanıp**InstanceMethod** nesne örnekleri ile etkileşimde bulunabilirler. Örneğin **cmbInstanceMethods**isimli **ComboBox** kontrolü, **ItemsSource** özelliğine **static** bir veri kaynağı olarak bu**ObjectDataProvider** örneğini bağlamıştır. **DisplayMemberPath** özelliğine atanan **Name** değeri ise, **InstanceMethod** örnekleri içerisindeki **Name** özelliğini işaret etmekte olup **ComboBox'** un üzerinde nelerin gösterileceğini belirtmektedir. **ComboBox** üzerinde hareket edildikçe alt tarafta yer alan **grdMethodParameters** isimli **DataGrid** içeriğininde, ilgili metoda ait parametre listesi ile doldurulması beklenmektedir. Bu nedenle her iki bileşenin **IsSychnronizedWithCurrentItem**özelliği **true** değerine sahiptir**. DataGrid** bileşeninin **ItemsSource** özelliği de **static** veri kaynağı olan **dsInstanceMethods'** a bağlanmıştır. Ama!

Path özelliğinin değerine dikkat edelim. **Parameters** değeri aslında **InstanceMethodList**sınıfındaki özelliğin adıdır.

Dolayısıyla **ComboBox** kontrolünde bir öğe seçildiğinde, buna bağlı**Parameters** özelliğinin karşılığı olan liste**, DataGrid** içerisine basılıyor olacaktır. Görüldüğü üzere tipik olarak bir **WPF Data Binding** işlevselliği söz konusudur. Bunun dışında kalan kısımlarda bileşenin **Collapse** veya **Expand** edilmesi hallerinde nasıl görüneceği ifade edilmiştir. Dikkat edilecek olursa iki adet **DataTemplate** elementi vardır. Bunlardan birisi **Collapsed** diğer ise**Expanded** olarak isimlendirilmiştir. Son satırlarda yer alan **Grid** elementi içerisindeki**ContentPresenter'** da buna uygun olacak şekilde bileşenin **Collapsed** veya **Expanded** olarak**designer** üzerinde gösterilebilmesini sağlamaktadır*(Ne varki ben Collapsed hale bir türlü getirmeyi başaramadım. Yani örneğimizde şimdiden bir Bug' ımız olduğunu ifade etmek isterim 😕)*

Bileşenimizin **XAML** içeriğini bu şekilde oluşturmak yeterli değildir. Ayrıca **Visual Studio Designer'** ına söz konusu bileşeni bildirmemiz gerekmektedir. Bunun için ilk olarak **Activity Designer** sınıfının koda tarafını aşağıdaki hale getirmeliyiz.

```
using System.Activities.Presentation.Metadata;
using System.ComponentModel;
namespace Azon.Workflow.Activity.Design
{
   public partial class InstanceMethodActivityDesigner
   {
      #region Constructors and Destructors
      public InstanceMethodActivityDesigner()
      {
         this.InitializeComponent();
      }
      #endregion
      #region Public Methods
      public static void RegisterMetadata(AttributeTableBuilder builder)
      {
         builder.AddCustomAttributes(
            typeof(InstanceMethodActivity),
            new DesignerAttribute(typeof(InstanceMethodActivityDesigner)),
            new DescriptionAttribute("Instance Method Activity"));
      }
      #endregion
   }
}
```

Sınıf içerisindeki en önemli metod **RegisterMetadata** isimli **static** fonksiyondur. Bu metod içerisinde parametre olarak gelen **Attribute** tablosuna bazı bildirimlerde bulunularak yeni**niteliklerin(Attribute)** ilave edilmesi sağlanmaktadır. Hatta dilerseniz burada **Component** için bir**Icon(16X16 boyutlarında bir PNG olabilir)** dahi belirtebilirsiniz. Biz şimdilik bu detayı atlıyor olacağız.

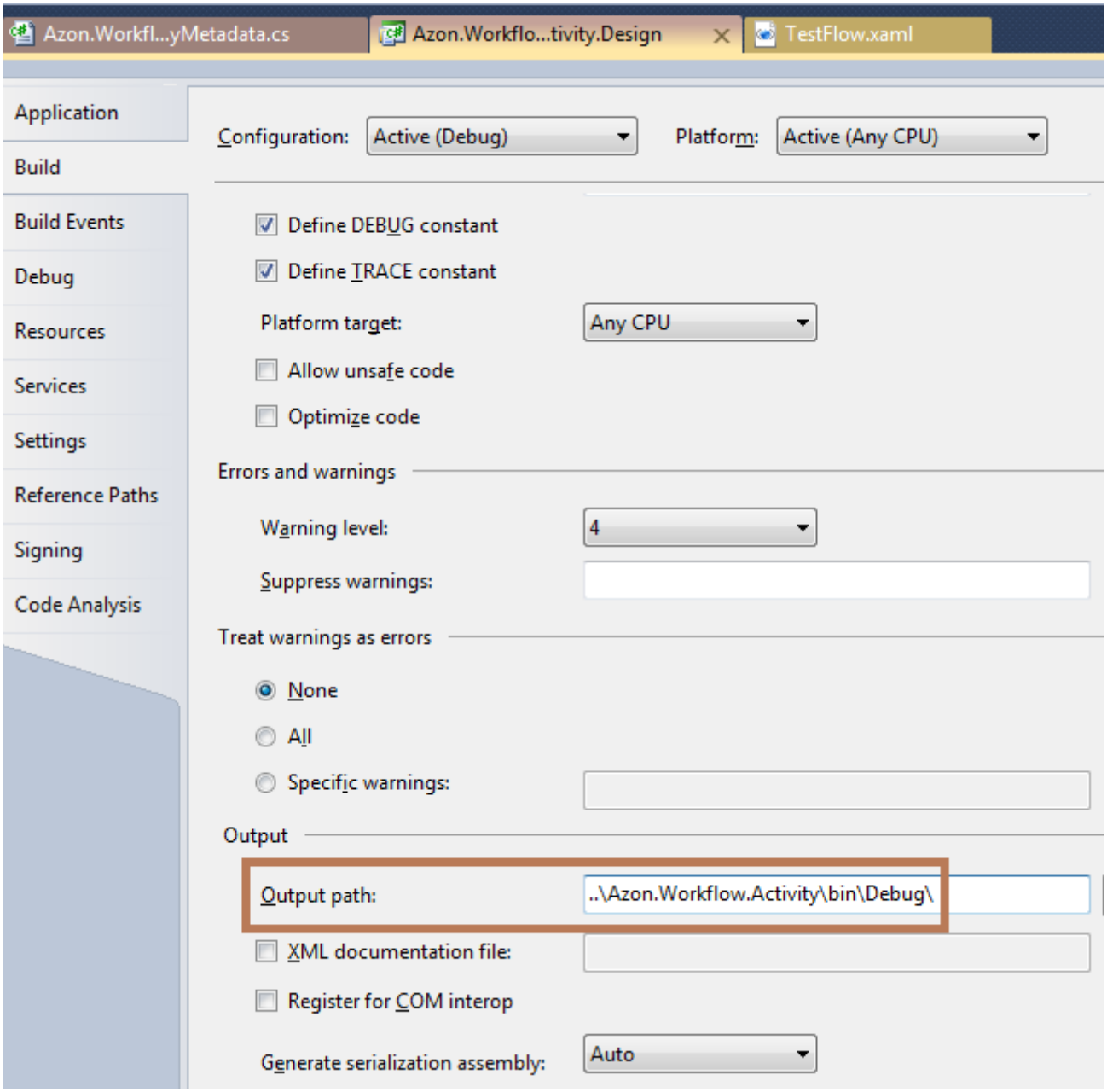
Peki söz konusu **static** metod nerede çağırılacaktır? 🤔 Bunun için**Azon.Workflow.Activity.Design** kütüphanesine **IRegisterMetadata** arayüzünü implemente eden bir sınıfın eklenmesi gerekmektedir. **IRegisterMetadata** arayüzünden gelen **Register** metodu içerisinde ise, **InstanceMethodActivityDesigner** sınıfına dahil edilmiş olan **RegisterMetadata**isimli **static** metod çağrısı gerçekleştirilmektedir.

```
using System.Activities.Presentation.Metadata;
namespace Azon.Workflow.Activity.Design
{
   public sealed class ActivityLibraryMetadata
      : IRegisterMetadata
```

```
    {
        public void Register()
        {
            RegisterAll();
        }
        public static void RegisterAll()
        {
            var builder = new AttributeTableBuilder();
            InstanceMethodActivityDesigner.RegisterMetadata(builder);
            MetadataStore.AddAttributeTable(builder.CreateTable());
        }
    }
}
```

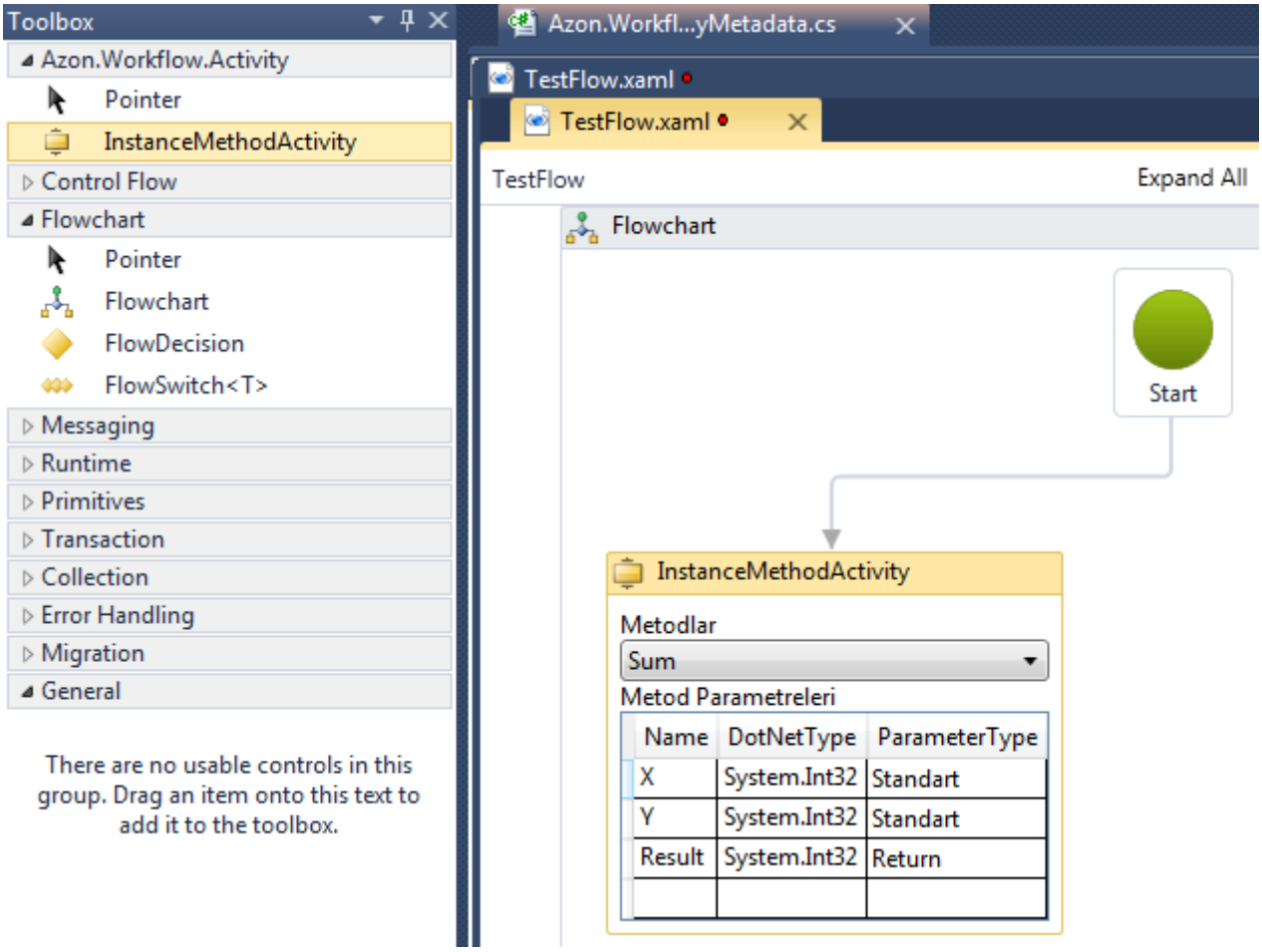
Burada kullanılan sınıfın adının çok önemi yoktur. Nitekim **Visual Studio IDE'** si, kendi çalışma zamanı ortamında, ilgili **Activity Designer** kütüphanesinde **IRegisterMetadata** arayüzünü uygulamış olan bir tipe bakmaktadır. Tipik bir **Plug-In** tasarım mantığı olduğunu rahatlıkla ifade edebiliriz.

Artık bileşenimizi deneyebiliriz demek isterdim ama son olarak yapmamız gereken ufak bir işlem daha var. Adı **Design** kelimesi ile biten kütüphanenin **dll** çıktısının, **NativeActivity** bileşenlerini içeren kütüphanenin olduğu yere doğru yapılması gerekmektedir. Aşağıdaki şekilde görüldüğü gibi.

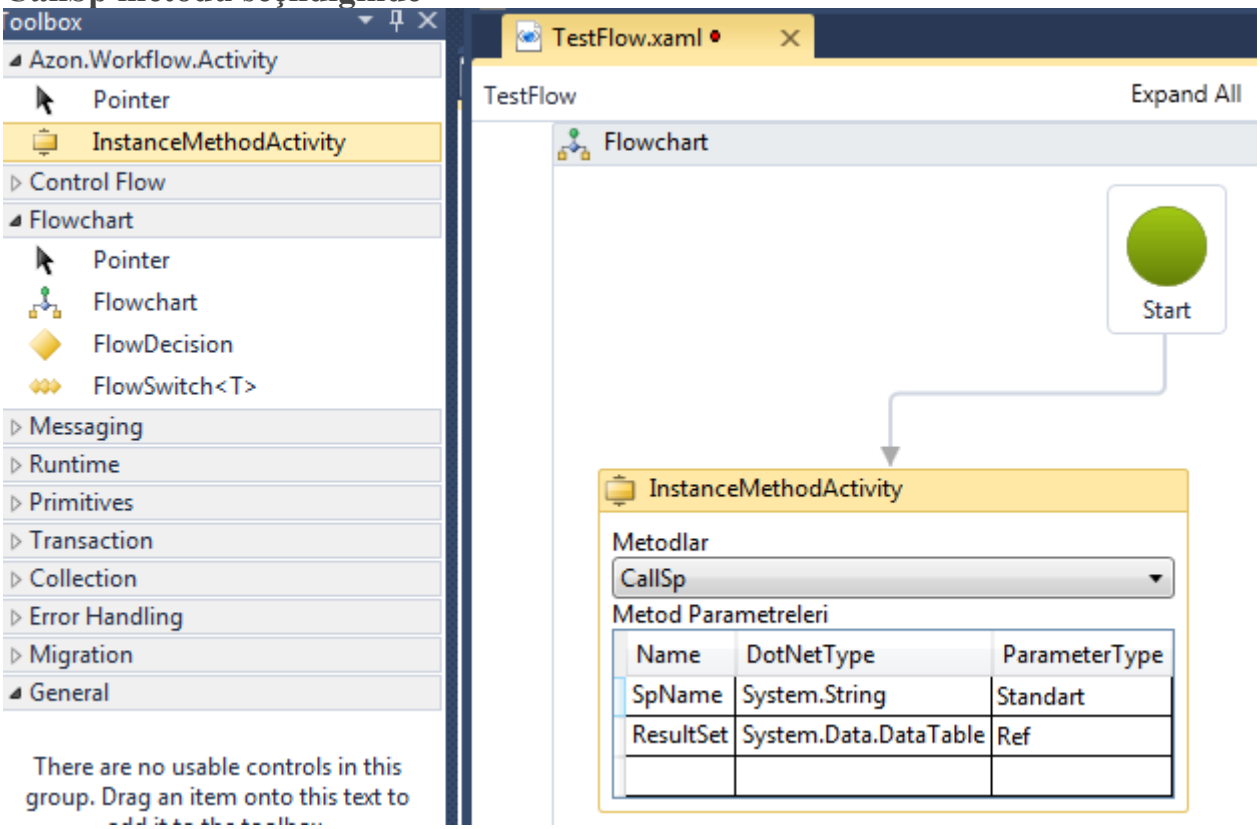


Artık basit bir **Workflow** üzerinden bileşenimizi deneyebiliriz. Bileşenimiz otomatik olarak **Toolbox**sekmesinde görünecektir. İşte **Visual Studio 2010** çalışma ortamına ait bir kaç örnek görüntü.

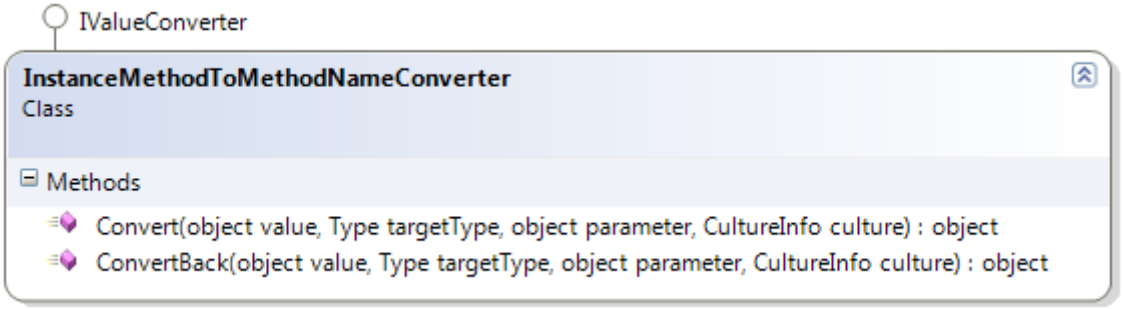
Sum metodu seçildiğinde



CallSp metodu seçildiğinde



Görüldüğü üzere bileşenimiz içerisinde **Data Binding** tekniklerini de kullanarak bir etkileşim gerçekleştirmeyi başardık. Şimdi bileşenimizi biraz daha geliştirmeyi deniyor olacağız. Buna göre**ComboBox** kontrolünde bir öğe seçildiğinde, **Name** alanının değerinin, o anki**InstanceMethodActivity'** ye ait **Property'** lerden **MethodName** alanında gösterilmesini sağlamaya çalışacağız. Bu bonus senaryoda işi zorlaştıran kısım şu; **ComboBox** bileşeni içerisinde **Binding** sebebi ile **InstanceMethodList** sınıfına ait değerler taşınmaktadır. Bu değerler **InstanceMethod** türünden nesne örnekleridir aslında.**InstanceMethodActivity** bileşeninin, **MethodName** özelliği ise **string** tipindendir. Dolayısıyla**ComboBox** kontrolünün **SelectedValue** özelliği içerisinde **XAML** tarafında bildirilecek şekilde özel bir **Convert** işleminin uygulanması gerekmektedir. Bu amaçla öncelikli olarak bir **Converter** tipini**Azon.Workflow.Activity** kütüphanesine aşağıdaki gibi ilave edelim.




```
namespace Azon.Workflow.Activity
{
   using System;
   using System.Globalization;
   using System.Windows.Data;

   public class InstanceMethodToMethodNameConverter
      :IValueConverter
   {
      public object Convert(object value, Type targetType, object parameter,
CultureInfo culture)
      {
         return null; //BURASI SİZE ÖDEV OLSUN
      }
      public object ConvertBack(object value, Type targetType, object parameter,
CultureInfo culture)
      {
         InstanceMethod instanceMethod = (InstanceMethod)value;
         return instanceMethod.Name;
      }
   }
}
```

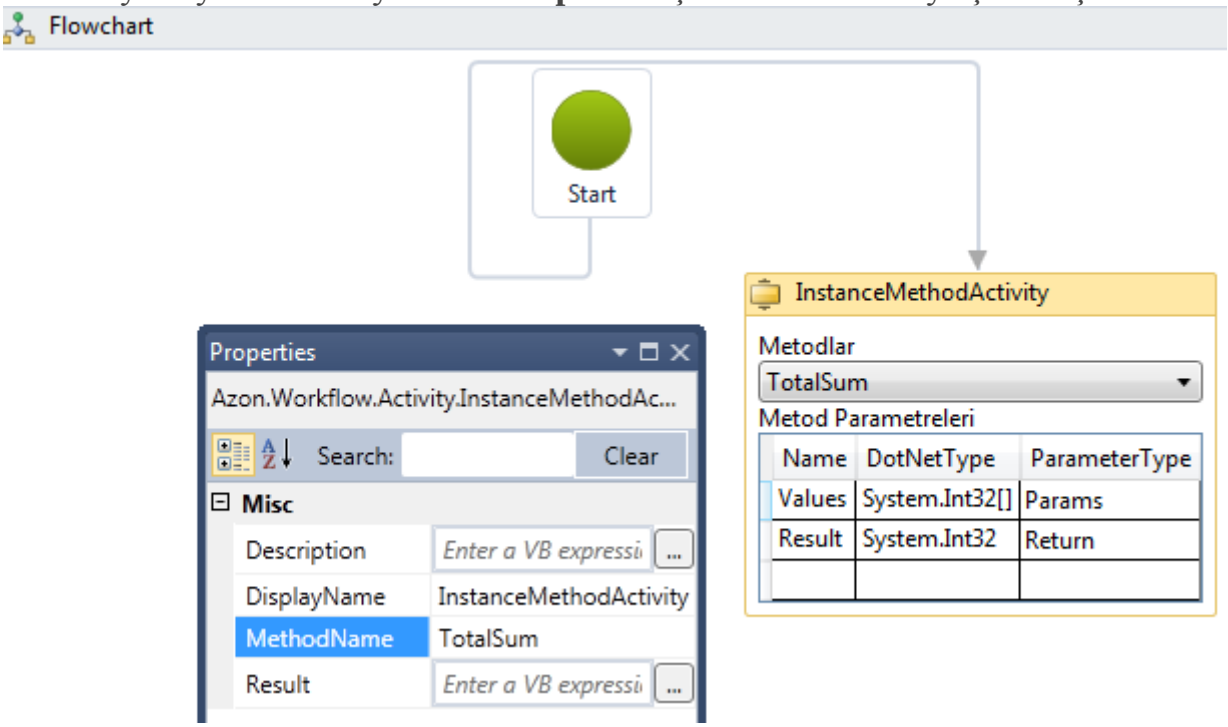
InstanceMethodToMethodNameConverter tipi, **System.Windows.Data** isim alanında*(ki PresentationFramework.dll assembly' ının projede referans edilmesi gerekmektedir)* yer alan**IValueConverter arayüzünü(Inteface)** uygulamaktadır. Buna göre **TwoWay Binding'** i destekleyecek şekilde **Convert** ve **ConvertBack** metodlarının implemantasyonunu istemektedir.**Convert** metodu, **Properties** penceresinden girilen değere göre **ComboBox** içerisinde ilgili öğeye gidilmesini sağlamaktadır. **ConvertBack** metodu ise tam tersi işlevi üstlenmekte olup, **ComboBox**'ta seçilen **InstanceMethod** nesne örneğinin **Name** özelliğinin değerini **Properties** penceresindeki**MethodName** alanına basmaktadır. Tabi söz konusu tipin yazılması yeterli değildir. Bu **Converter**tipinin **XAML** tarafında da dekleratif olarak bildirilmesi ve **ComboBox** bileşeni ile ilişkilendirilmesi gerekmektedir. Bunun için **InstanceMethodActivityDesigner.xaml** içeriğini aşağıdaki gibi güncellememiz yeterli olacaktır.

```
<sap:ActivityDesigner
x:Class="Azon.Workflow.Activity.Design.InstanceMethodActivityDesigner"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns:s="clr-
namespace:System;assembly=mscorlib"
   xmlns:sap="clr-
namespace:System.Activities.Presentation;assembly=System.Activities.Presentation"
   xmlns:sapv="clr-
namespace:System.Activities.Presentation.View;assembly=System.Activities.Presentation"
   xmlns:Model="clr-
namespace:System.Activities.Presentation.Model;assembly=System.Activities.Presentation
"
   xmlns:sapc="clr-
namespace:System.Activities.Presentation.Converters;assembly=System.Activities.Present
ation"
   xmlns:activity="clr-
namespace:Azon.Workflow.Activity;assembly=Azon.Workflow.Activity"
   mc:Ignorable="d" xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006">
   <sap:ActivityDesigner.Resources>
      <ResourceDictionary x:Uid="ResourceDictionary_0">
         <sapc:ModelToObjectValueConverter x:Key="ModelToObjectValueConverter" />
         <activity:InstanceMethodToMethodNameConverter
x:Key="MethodToMethodNameConverter"/>
         <ObjectDataProvider x:Key="dsInstanceMethods" ObjectType="{x:Type
activity:InstanceMethodList}">
         </ObjectDataProvider>
```

```
        <DataTemplate x:Key="Collapsed">
           <StackPanel Orientation="Horizontal">
              <TextBlock VerticalAlignment="Center" Margin="5" Text="Method Caller"
/>
           </StackPanel>
        </DataTemplate>
        <DataTemplate x:Key="Expanded">
           <Grid>
              <Grid.RowDefinitions>
                 <RowDefinition/>
                 <RowDefinition/>
                 <RowDefinition/>
                 <RowDefinition/>
              </Grid.RowDefinitions>
              <Grid.ColumnDefinitions>
                 <ColumnDefinition/>
              </Grid.ColumnDefinitions>
              <TextBlock Text="Metodlar" Grid.Row="0"/>
              <ComboBox x:Name="cmbInstanceMethods" Grid.Row="1"
ItemsSource="{Binding Source={StaticResource dsInstanceMethods}}"
DisplayMemberPath="Name"
IsSynchronizedWithCurrentItem="True" SelectedValue="{Binding
Path=ModelItem.MethodName, Mode=TwoWay, Converter={StaticResource
MethodToMethodNameConverter}}" />
              <TextBlock Text="Metod Parametreleri" Grid.Row="2"/>
              <DataGrid x:Name="grdMethodParameters" Grid.Row="3"
IsSynchronizedWithCurrentItem="True" ItemsSource="{Binding Source={StaticResource
dsInstanceMethods}, Path=Parameters}"/>
           </Grid>
        </DataTemplate>
        <Style x:Key="ExpandOrCollapsedStyle" TargetType="{x:Type
ContentPresenter}">
           <Setter Property="ContentTemplate" Value="{DynamicResource Expanded}" />
           <Style.Triggers>
              <DataTrigger Binding="{Binding Path=ShowExpanded}" Value="true">
                 <Setter Property="ContentTemplate" Value="{DynamicResource
Collapsed}" />
              </DataTrigger>
           </Style.Triggers>
        </Style>
```

```
        </ResourceDictionary>
    </sap:ActivityDesigner.Resources>
    <Grid>
        <ContentPresenter Style="{DynamicResource ExpandOrCollapsedStyle}"
Content="{Binding}" />
    </Grid>
</sap:ActivityDesigner>
```

Buna göre **ComboBox** kontrolünde bir **Metod** adı seçilirse bu **Properties** penceresine de bu isim yansıyacaktır. Böylece **developer'** ın işi biraz daha kolaylaştırılmış olmaktadır.



Şimdi olayı biraz daha renklendireceğiz. Örneğin **Visual Studio Designer'** ı üzerinde çalışırken,**Activity** bileşenlerine ait **event** methodları kullanmak istediğinizi ve hatta bu **event** metodlar içerisinde, diğer kontrollerin içeriklerine ulaşmak istediğimizi düşünelim. Bu senaryoyu irdelemek için **InstanceMethodActivityDesigner.xaml** içeriğine bir **Button** kontrolü ekleyerek ilerleyebiliriz.

```
<DataTemplate x:Key="Expanded">
            <Grid>
                <Grid.RowDefinitions>
                    <RowDefinition/>
                    <RowDefinition/>
                    <RowDefinition/>
                    <RowDefinition/>
                    <RowDefinition/>
                </Grid.RowDefinitions>
                <Grid.ColumnDefinitions>
                    <ColumnDefinition/>
```

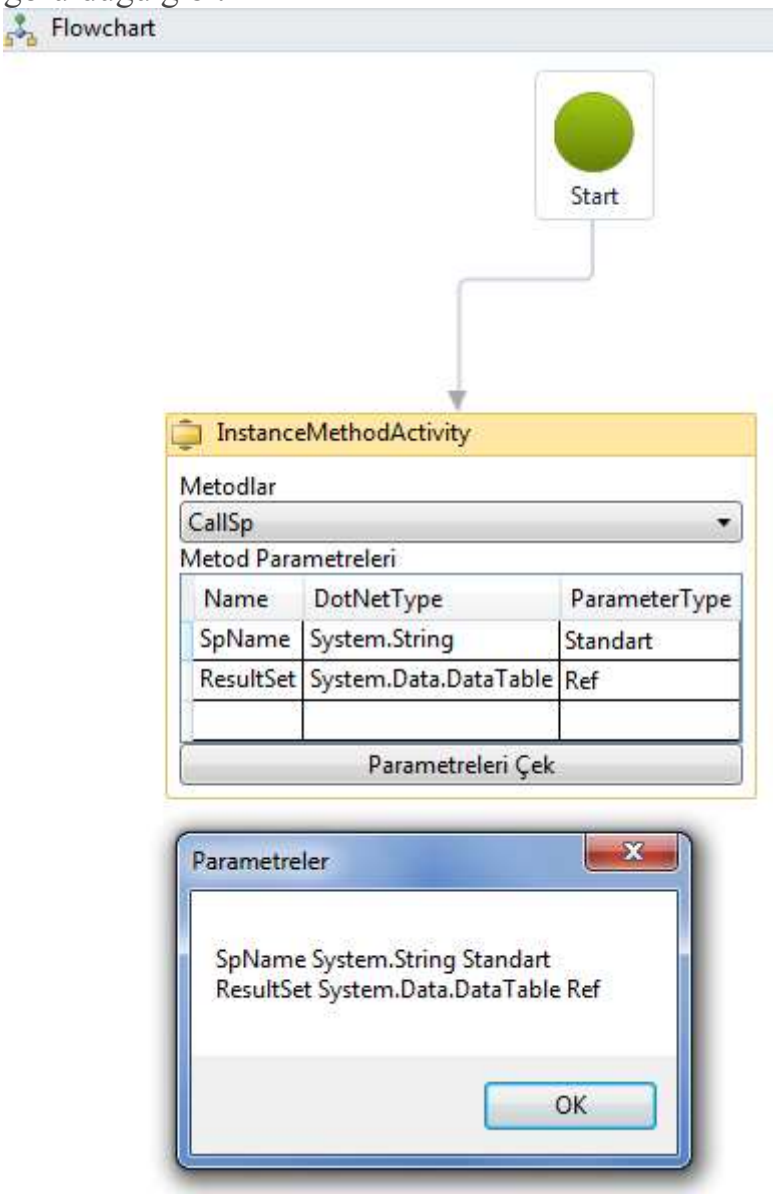
```
            </Grid.ColumnDefinitions>
            <TextBlock Text="Metodlar" Grid.Row="0"/>
            <ComboBox x:Name="cmbInstanceMethods" Grid.Row="1"
ItemsSource="{Binding Source={StaticResource dsInstanceMethods}}"
DisplayMemberPath="Name" IsSynchronizedWithCurrentItem="True"
SelectedValue="{Binding Path=ModelItem.MethodName, Mode=TwoWay,
Converter={StaticResource MethodToMethodNameConverter}}" />
            <TextBlock Text="Metod Parametreleri" Grid.Row="2"/>
            <DataGrid x:Name="grdMethodParameters" Grid.Row="3"
IsSynchronizedWithCurrentItem="True" ItemsSource="{Binding Source={StaticResource
dsInstanceMethods}, Path=Parameters}"/>
            <Button x:Name="btnCatchParameters"
Click="btnCatchParameters_Click" Content="Parametreleri Çek" Grid.Row="4">
            </Button>
          </Grid>
        </DataTemplate>
```

btnCatchParameters isimli **Button** kontrolünün **Click** olay metodunun yüklendiği görülmektedir. Bu olay metodu çok doğal olarak **InstanceMethodActivityDesigner.cs** içerisine açılıyor olacaktır. Olay metodu içeriğini aşağıdaki kod parçasında görüldüğü gibi geliştirebiliriz.

```
private void btnCatchParameters_Click(object sender, System.Windows.RoutedEventArgs
e)
{
  Grid parent=((Grid)((Button)e.Source).Parent);
  foreach (UIElement control in parent.Children)
  {
    DataGrid grid=control as DataGrid;
    if(grid!=null)
    {
      StringBuilder builder = new StringBuilder();
      foreach (var item in grid.ItemsSource)
      {
        builder.AppendLine(item.ToString());
      }
      MessageBox.Show(builder.ToString(),"Parametreler");
    }
  }
}
```

Olay metodu içerisindeki felsefe oldukça basittir. **Button** kontrolü aslında bir **Grid** içerisinde yer almaktadır ve **DataGrid** bileşeni de aynı **seviyede(Level)** durmakta olan bir elementtir. Dolayısıyla**Button** bileşeninin **Parent** elementine(**Container** da

diyebiliriz) çıkıp, tüm alt kontrolleri dolaşabilir ve **Grid** tipinde olana vardığımızda da **ItemsSource** özelliğine ait koleksiyon içeriğini ele alabiliriz. Kulağımızı farklı bir şekilde tuttuğumuzu ifade edebiliriz aslında ama şu anda elimizden en iyi çözüm bu. Böylece **Visual Studio Designer**' ı içerisindeyken, **DataGrid** elementlerine ve seçili olan metodun parametre listesine ulaşmamız mümkün olacaktır. Aynen aşağıdaki şekilde görüldüğü gibi.



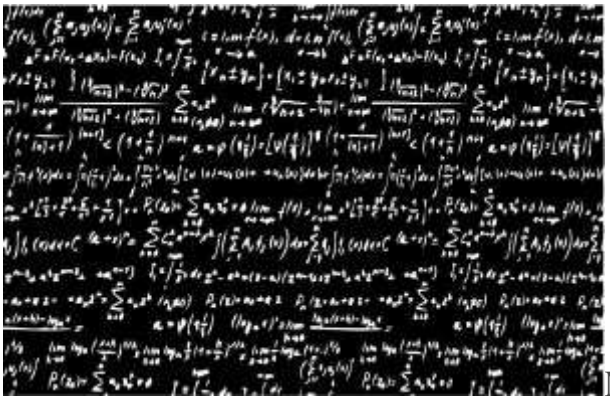
Görüldüğü üzere **Custom Activity** geliştirmek kolay olsa da, bu bileşeni **Designer** desteğine sahip olacak şekilde genişletmek bir kaç ipucu içeren ve dikkat edilmesi gereken bir süreci gerektirmektedir. Geliştirmiş olduğumuz örnekte bazı eksik kısımlar da bulunmaktadır. Örneğin**XAML** tarafında dekleratif olarak **Event** bazlı etkileşimler çok fazla ele alınmamıştır.*(Bir veritabanı bağlantısını seçtiren ve hatta **design** tarafında bir **SQL** sorgusunu çalıştırtıp sonuçları bir **DataGrid**kontrolüne basan bir **Activity Designer** yazmaya çalıştığınızı hayal edin. Üstelik Connection' ı tanımladığınızda Test' de edebilmelisiniz vs)* Bu konuda detaylı ve derinlemesine araştırmalarıma devam ediyorum. Yeni bilgiler edindikçe sizinle paylaşmaya

gayret ediyor olacağım. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.
WritingDesignerActivityV2.zip (155,91 kb)

Çerezlik Algoritmalar ve Extension Methodlar

Cuma, 8 Haziran 2012 08:30

Extension Methods, Fisher Yates Shuffle, Ceaser Chipper, Palindromic Words,Palindromic Numbers, C#

 Merhaba Arkadaşlar,
Akademik yıllarımızda çoğumuz karmaşık matematik algoritmaları ile uğraşmak durumunda kalmışızdır*(Sınav stresini hatırlamak bile istemiyorum)* Özellikle veri yapıları ve algoritmalar*(Data Structures and Alogirthms)* veya Numeric Analiz gibi derslerde yoğun algoritma tasarımları üzerinde çalışılmaktadır. Doğruyu söylemek gerekirse ülkemizde bu dersleri layıkıyla veren kurum sayısı oldukça azdır. Konular genellikle sırlama algoritmalarının*(özellikle Quick Sort' un)*ötesine pek geçmemektedir. En fazla yüksek lisans öğreniminde farklı konulara girilmesi söz konusudur.
Pek tabi yazılım dünyası söz konusu olduğunda var olan hemen her algoritmanın karşılığı olan kodlamaların geliştirilmesi de önemli bir mevzudur. Bilimsel uygulamalarda, finansal model çözümlerinde, endusturi alanındaki planlama tekniklerinde vb...Ben bu yazımda sizleri o karmaşık ve anlaşılması zor algoritmalar ile yormayacağım. Bunun yerine eğilenceli sayılabilecek ve özellikle oyun programlamada oyunculara keyifli dakikalar yaşatmanızı sağlayabilecek basit bir kaç algoritma üzerinde durmaya çalışacağım. Söz konusu algoritmaları birer **Extension Method** olarak geliştireceğiz*(Extension Method kavramını hatırlayalım. C# 3.0 - Derinlemesine Extension Method Kavramı)* Dilerseniz hiç vakit kaybetmeden ilk algoritmamız ile işe başlayalım.

 Biraz eskilere gidiyor olacağız.
Hatta **Roma** imparatoru **Ceaser** zamanına **Ceaser**, **Roma** imparatorluğunun şaşalı dönemlerinde generalleri ile haberleşirken basit bir şifreleme metodolojisini

kullanmaktaymış. Büyük bir ihtimalle sonradan **Ceaser Cheaper** olarak adlandırılan bu algoritmanın çalışma şekli aslında son derece basitmiş. Algoritmaya göre bir cümleyi veya metni, alfabe üzerinde belirlenen sayı kadar sağa(ileri) veya sola(geri) doğru ötelenmek suretiyle karşılık gelen harfler ile dizmek söz konusudur. Sonuçta ortaya, okunabilirliği pek olmayan anlamsız bir veri çıkmaktadır ama generaller tarafından bu, merkez sayı noktasına göre tekrardan geriye doğru ötelenerek anlamlı hale getirilebilir. Elbetteki bizim gerçek hayat uygulamalarımızda kullanacağımız bir şifreleme algoritması değildir bu. Ancak basit bir zeka oyununda neden kullanılmasın ki. Eğlenceli olabilir *(Algoritma hakkında detaylı bilgilere bu adresten ulaşabilirsiniz.)* Haydi gelin bu algoritma için bir genişletme metodu yazalım.

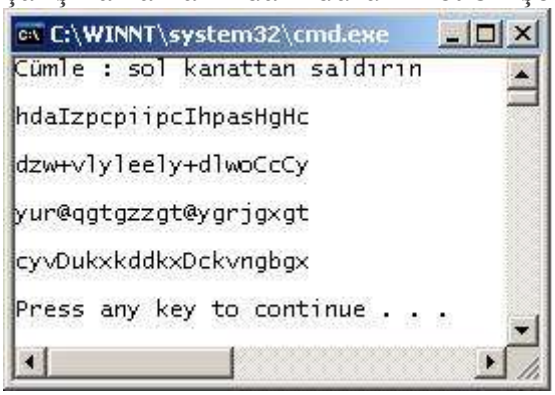
```
public static class AlgorithmExtensions
{
    #region Ceaser Cheaper ile karıştırma

    public static string CaesarChiper(this string Word, int ShiftNumber)
    {
        char[] chars = Word.ToCharArray();

        for (int i = 0; i < chars.Length; i++)
        {
            char currentLetter = chars[i];
            currentLetter = (char)(currentLetter + ShiftNumber);

            if (currentLetter > 'z')
                currentLetter = (char)(currentLetter - 26);
            else if (currentLetter < 'a')
                currentLetter = (char)(currentLetter + 26);

            chars[i] = currentLetter;
        }
        return new string(chars);
    }

    #endregion
}
```

Aslında algoritma oldukça basit gördüğünüz gibi. Girilen **Shift** değerine göre **ASCII** tablosu üzerinden sağa veya sola doğru hareket ediliyor. **z** ve **a** aralığında bir öteleme hareketi yapıldığına dikkat edelim. Öteleme noktalarında elde edilen karakterler ardışıl olarak dizilerekten metnin karıştırılmış hali geriye döndürülüyor. Bir **Extension Method** olarak yazdığımız için herhangibir**String** değişken üzerinden uygulanabilir. Peki nasıl kullanacağız?

Gelin aşağıdaki kod parçası ile ilerleyelim.

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace TestApp
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Ceaser Chipper Test

         string word = "sol kanattan saldırın";
         Console.WriteLine("Cümle : {0}\n",word);
         Console.WriteLine("{0}\n",word.CaesarChiper(15));
         Console.WriteLine("{0}\n", word.CaesarChiper(-15));
         Console.WriteLine("{0}\n", word.CaesarChiper(6));
         Console.WriteLine("{0}\n", word.CaesarChiper(10));

         #endregion

      }
   }
}
```

Örnek uygulamamızda girilen cümlenin **Shift** değerine göre farklı çıktılarının üretildiği görülecektir. İlk seferde 15 karakter sağa gidilerek işe başlanırken ikinci denemede 15 karakter sola gidilmek suretiyle bir üretim söz konusudur. Aşağıdaki ekran çıktısında çalışma zamanındaki durum net bir şekilde görülmektedir.



Dikkat edileceği üzere eğlenceli görünen karmaşık veri içerikleri üretilmiş durumda. Tabi bunu çözümlemeye çalışmak oyuncunun işi olacak. Oldukça fazla zorlanacağından emin olabilirsiniz.

Ceaser' ın hakkına Ceaser' a verip ve**Ceaser**' a elveda diyerek yolumuza devam edelim. Sırada yer alan algoritmamız ise **Fisher-Yates Shuffle**olarak literatürde yer almaktadır. Bu algoritma yardımıyla bir sayı veya kelime dizisinin ya da farklı bir veri kümesinin her defasında farklı olacak şekilde karıştırılarak elde edilmesi söz konusudur. Bir başka deyişle farklı**permütasyonların** hesap edilerek bir karma veri içeriği üretilmesi gibi bir durum mevcuttur. *(1938 yılında keşfedilmiş olan bu algoritma hakkında ki detaylı bilgileri yine wikipedia adresi üzerinden elde edebilirsiniz.)* Biz hiç vakit kaybetmeden bu algoritma için bir extension metod geliştirerek ilerleyelim.

```
using System;
using System.Collections.Generic;

public static class AlgorithmExtensions
{
   #region Fisher-Yates ile Shuffling

   public static void Shuffle<T>(this List<T> SourceArray) // Fisher-Yates Shuffle
Algoritmasına göre karıştırır
   {
      Random randomizer = new Random();

      for (int i = SourceArray.Count; i > 1; i--)
      {
         int j = randomizer.Next(i); // 0 ile i-1 arasında rastgele bir değer üretecektir
         T temp = SourceArray[j];
         SourceArray[j] = SourceArray[i - 1];
         SourceArray[i - 1] = temp;
      }
   }

   #endregion
}
```

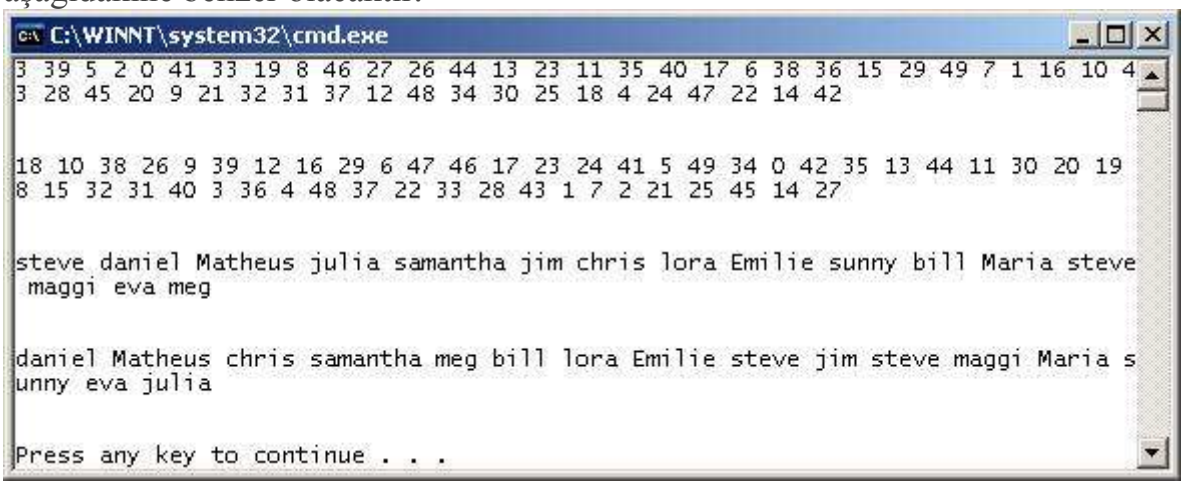
Görüldüğü gibi algoritmamız kaynak olan dizi içerisindeki elemanları sondan başa doğru gezmektedir. Bu işlem sırasında o anki iterasyon değerine kadarki aralıkta üretilen bir

rastgele sayı ve bunun dizideki karşılığı olan değer geçici bir değişkene atanır. Ardından o anki iterasyonun bir önceki değerine karşılık gelen dizi elemanı rastgele elde edilen değerin işaret ettiği indise taşınır. Son olarak da geçici değişkene alınan eleman o anki iterasyon değerinin bir öncesine karşılık gelen indise yerleştirilir. Genişletme metodunu generic olarak tasarladığımızı ve herhangibir **List** tipine uygulayabildiğimizi fark etmişsinizdir. Temel olarak hedefimiz sayısal veya metin tabanlı koleksiyon listelerinin karıştırılmış çıktılarını elde etmektir *(ki bir Puzzle uygulamasında bu teknik oldukça işe yarayabilir)* Metodun kalbinde **Random** tipi yer almaktadır. Metodumuzu aşağıdaki kod parçasında olduğu gibi test sürüşüne çıkartabiliriz.

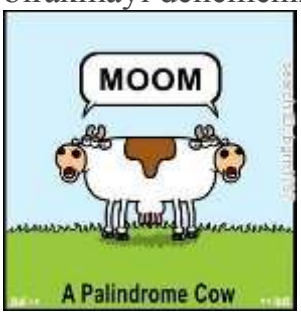
```
using System;
using System.Collections.Generic;
using System.Linq;

namespace TestApp
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Fisher - Yates Shuttle Alogirtması

         List<int> numbers = Enumerable.Range(0, 50).ToList();
         List<string> names = new List<string> {
            "bill","steve","daniel","meg","julia"
            ,"sunny","Matheus","chris","lora"
            ,"maggi","jim","steve","Emilie"
            ,"Maria","eva","samantha"
         };

         numbers.Shuffle();
         WriteToConsole(numbers);

         numbers.Shuffle();
         WriteToConsole(numbers);

         names.Shuffle();
         WriteToConsole(names);

         names.Shuffle();
         WriteToConsole(names);
```

```
        #endregion
    }

    private static void WriteToConsole<T>(List<T> SourceArray)
    {
        foreach (var element in SourceArray)
        {
            Console.Write("{0} ", element);
        }
        Console.WriteLine("\n\n");
    }
  }
}
```

Test kodunda **string** tipte isimlerden oluşan bir **List** koleksiyonu ve benzer şekilde sayılardan oluşan bir veri içeriği söz konusudur. Uygulamanın çalışma zamanı çıktısı ise aşağıdakine benzer olacaktır.



Size tavsiyem basit bir fotoğrafı n sayıda kareye böldükten sonra, bu parçaları işaret eden sınıfa ait nesne örneklerinden oluşan bir **List** koleksiyonunu, **Fisher-Yates Shuffle** algoritmasını kullanarak, oyuncuyu her seferinde farklı bir karmaşa ile baş başa bırakmayı denemeniz olacaktır



Geldik bu yazımızda size aktarmak istediğim son algoritmaya. Bu kez**Palindromik** veri tespiti yapmak için kullanılan bir algoritma üzerinde duracağız. **Palindromic** sayılar **181, 191, 55** gibi tersten okunduklarında da aynı sayı değerini veren kavramlar olarak düşünülebilirler. Aslında**Palindromic** sayılar olarak düşünebileceğimiz bu modeli

kelimeler için de ele alabiliriz. **ANA, KAÇAK** gibi örnekler bu anlamda düşünülebilir. Senaryo olarak baktığımızda ise, söz gelimi metin içerikli bir döküman içerisinde yer alan **Palindromic** kelimeleri veya cümleleri oyuncuya buldurabilir ve süre bazında başarısını ölçümlemeye çalışabiliriz. *(Bu algoritma ile ilişkili detaylı bilgileri yine Wikipedia üzerinden okuyabilirsiniz.)* Ancak öncesinde algoritma için gerekli olan genişletme metodumuzu yazalım.

```
using System;
using System.Collections.Generic;

public static class AlgorithmExtensions
{
   #region Palindromic Kelimeleri/Cümleleri Bulmak

   public static List<string> PalindromicCheck(this List<string> Words)
   {
      List<string> result = new List<string>();

      foreach (string word in Words)
      {
         if (IsPalindromicData(word))
            result.Add(word);
      }

      return result;
   }

   private static bool IsPalindromicData(string SourceValue)
   {
      int minValue = 0;
      int maxValue = SourceValue.Length - 1;
      while (true)
      {
         if (minValue > maxValue)
            return true;

         char a = SourceValue[minValue];
         char b = SourceValue[maxValue];

         if (char.ToLower(a) != char.ToLower(b))
            return false;
```

```
            minValue++;
            maxValue--;
        }
    }

    #endregion
}
```

Sonsuz while döngüsü aslında kelimenin başından ve sonundan itibaren orta noktasına gelinceye kadar bir iterasyonu kullanmaktadır. İlk karakter ve son karakterin aynı olup olmadığı noktasında devreye giren algoritma orta noktadaki karaktere kadar devam edecektir. Genişletme metodumuz bu versiyonu ile **string** tipinden **List** koleksiyonuna uygulanabilecek şekilde tasarlanmıştır. Koleksiyon içerisindeki veriler, **IsPalindromicData** metodu yardımıyla kontrol edilmektedir. Metod sonuç olarak **Palindromic** veri içeriğini barındıran başka bir **List** koleksiyonunu geriye döndürür. Haydi gelin örneğimizi test edelim. Bu amaçla aşağıdaki kod parçasını göz önüne alabiliriz.

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace TestApp
{
    class Program
    {
        static void Main(string[] args)
        {
            #region Palindromik kelimelerin tespiti

            List<string> words = new List<string>{
                "ana","baba","amaç","abla","aha","kaçak"
                ,"kayak","kırık","cam","uzak","yakın","meşale"
                ,"neden","sus","süs","'ey edip adanada pide ye'"
            };


            var result = words.PalindromicCheck();
            WriteToConsole(result);

            #endregion
        }

        private static void WriteToConsole<T>(List<T> SourceArray)
```

```
        {
            foreach (var element in SourceArray)
            {
                Console.Write("{0} ", element);
            }
            Console.WriteLine("\n\n");
        }
    }
}
```

Örnekte kullanılan koleksiyon içerisinde tertsten okunduklarında da aynı olan bazı kelimeler ve hatta bir cümle mevcuttur. İşte çalışma zamanı sonuçları.

Bu algoritmayı oyuncudan ziyade oyun motoru kullanıyor olabilir. Ya da siz geniş bir kelime kümesini ekrana basıp bu küme içerisindeki **Palindromic** kelimeleri tespit etmesi için henüz ilk okul çağında olan bir oyuncuyu tercih edebilir ve süre bazlı bir ortam sağlayarak, onun dikkat, kavrama, fark etme, görsel hafıza gibi yeteneklerini arttırmaya çalışabilirsiniz

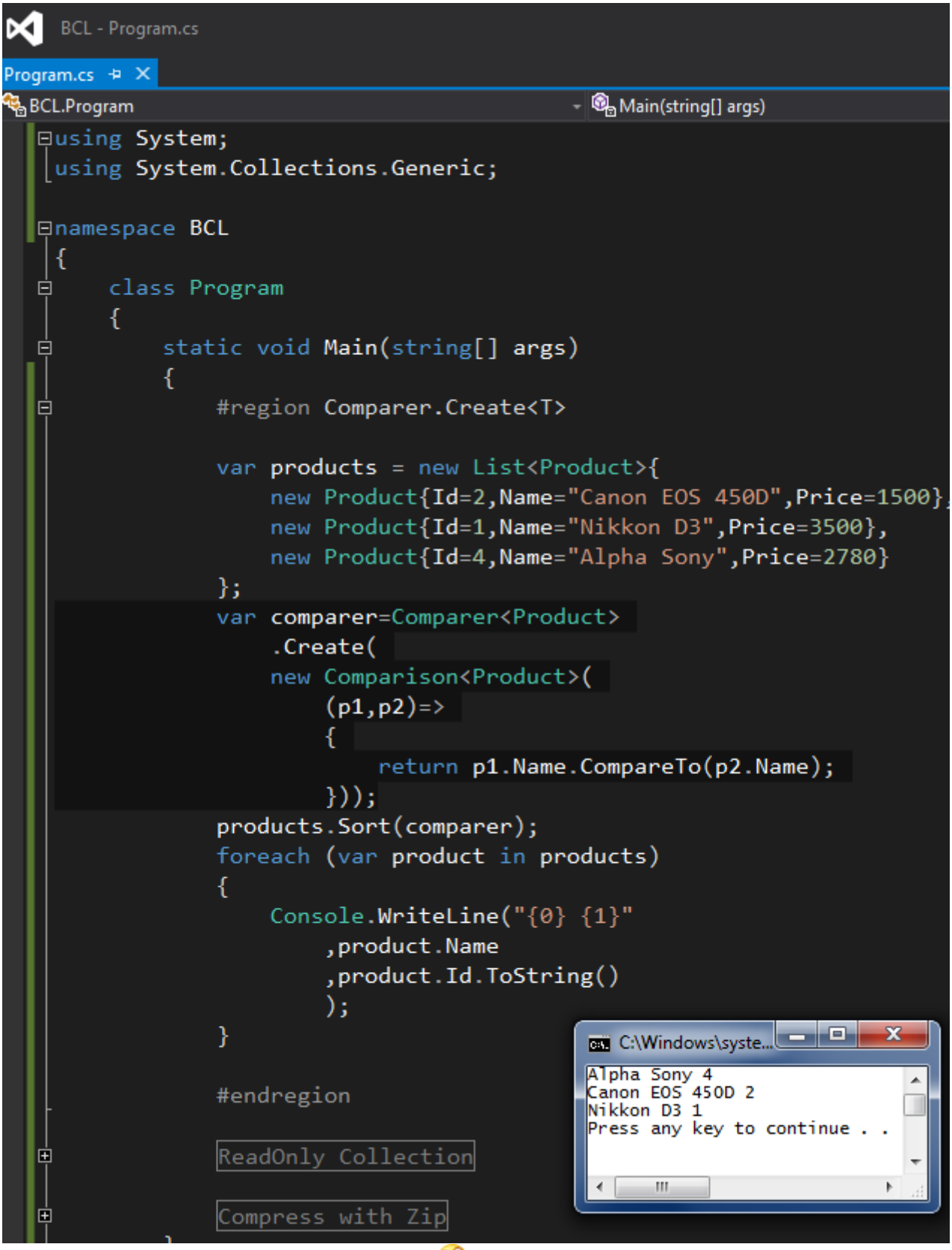
Aslında oyun programlama denilince çok basit ve yararlı algoritmalar olduğunu görebiliyoruz. Ben bu yazımızda sadece 3 tanesini sizlere aktarmaya çalıştım. Elbetteki çok daha fazlası var. Araştırmak, denemek, öğrenmek, test etmek ve kullanmak sizin göreviniz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim

Tek Fotoluk İpucu–54 Onda 75

Salı, 5 Haziran 2012 03:16

Collections, Comparer, .Net Framework 4.5

Merhaba Arkadaşlar,
Kendi tiplerimize ait koleksiyon nesnelerini kullanırken, **Sort** metodunu ele aldığımız durumlarda mutlaka neye göre karşılaştırma yapacağımızı belirtmemiz gerekmektedir. Bu amaçla **IComparer**veya **IComparable** gibi **arayüzleri(Interface)** ve bunların **generic** versiyonlarını kullanırız.
.Net Framework 4.5 ile birlikte ise, karşılaştırma işlemini tek satırda belirtebileceğimiz bir metod gelmektedir*(Tabi RC sürümü için konuştuğumuzu hatırlatalım)*
Comparer<T> tipinin **Create** isimli metodu, **Sort** fonksiyonu için gerekli olan karşılaştırma tipini kolayca üretebilmemizi sağlamaktadır. Parametre olarak aldığı **temsilci(Delegate)** metodunun kullanımı sırasında, **primitive type** seviyesine inip **Compare** operasyonunu çağırmaız yeterlidir. İşte size basit bir örnek



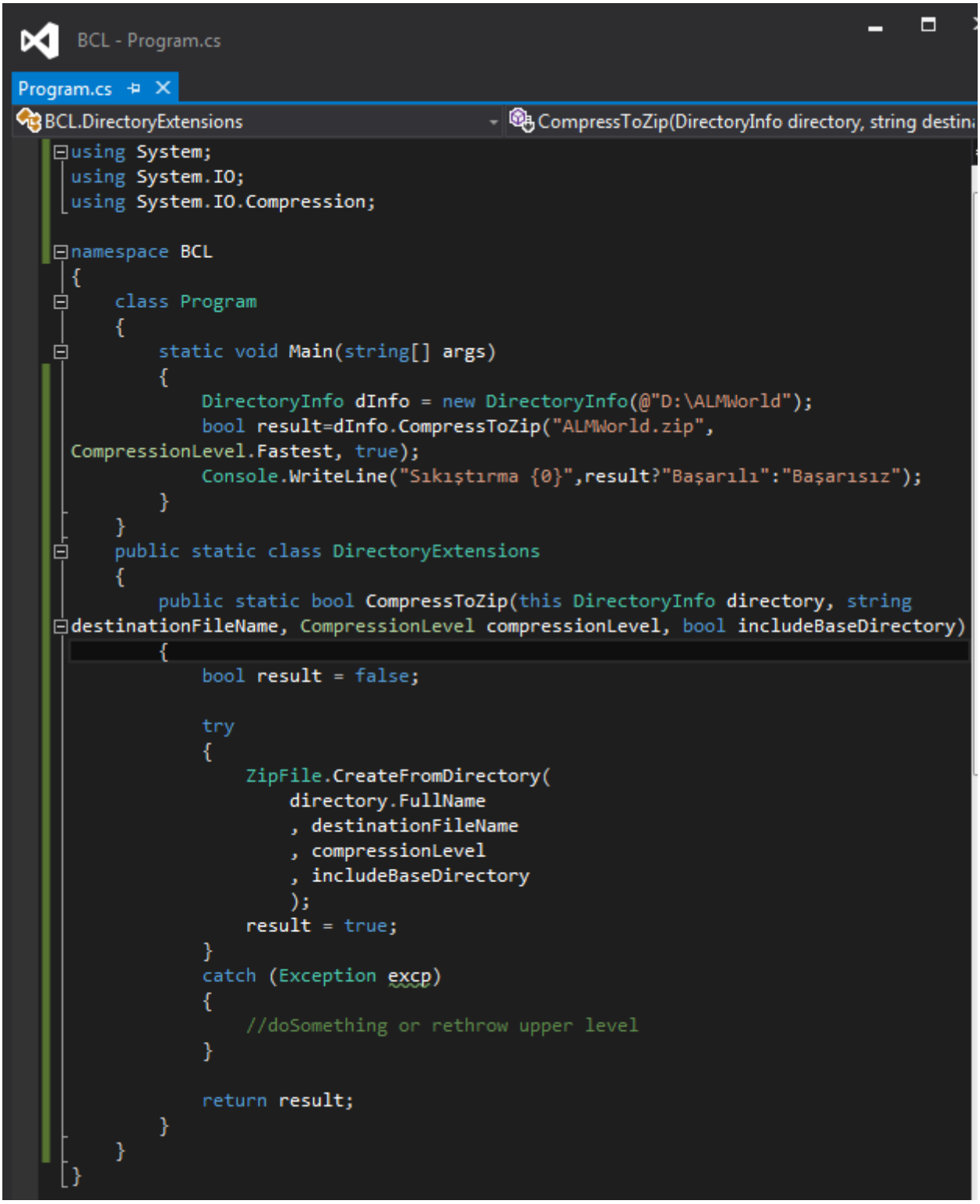
Başka bir ip ucunda görüşmek dileğiyle

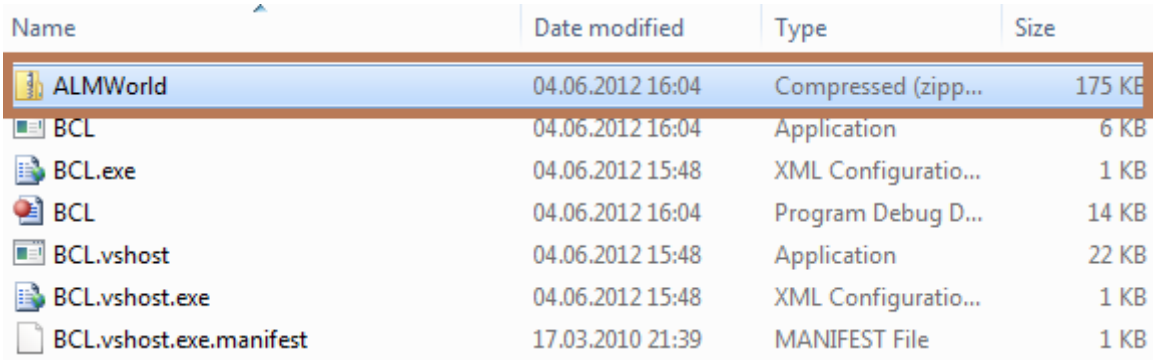
Tek Fotoluk İpucu–54Buçuk

Pazartesi, 4 Haziran 2012 23:12

.Net Framework
4.5, Zipfile, Compression, System.Io.Compression,System.Io.Compression.Filesystem, Zip

Merhaba Arkadaşlar,
Malum Visual Studio 2012 sürümünün RC sürümü geçtiğimiz hafta içerisinde yayınlandı ve internet üzerinden bu konu ile ilişkili yazılarda yayılmaya başlandı. Sadece Visual Studio 2012 değil ama .Net Framework 4.5 tarafında da epey önemli yenilikler geliyor. Ağırlık noktası her ne kadar paralel programlama tarafı ve doğal olarak async ile await anahtar kelimeleri olsa da, temel bazı yenilikler de var. Örneğin artık Zip formatında sıkıştırma desteği var. Söz gelimi bir klasör içeriğini ZIP formatında sıkıştıracak bir Extension metod yazmak istediniz  İşte buyrun.


BCL - Program.cs
Program.cs
BCL.DirectoryExtensions
CompressToZip(DirectoryInfo directory, string destin
using System;
using System.IO;
using System.IO.Compression;
namespace BCL
{
class Program
{
static void Main(string[] args)
{
DirectoryInfo dInfo = new DirectoryInfo(@"D:\ALMWorld");
bool result=dInfo.CompressToZip("ALMWorld.zip",
CompressionLevel.Fastest, true);
Console.WriteLine("Sıkıştırma {0}",result?"Başarılı":"Başarısız");
}
}
public static class DirectoryExtensions
{
public static bool CompressToZip(this DirectoryInfo directory, string
destinationFileName, CompressionLevel compressionLevel, bool includeBaseDirectory)
{
bool result = false;
try
{
ZipFile.CreateFromDirectory(
directory.FullName
, destinationFileName
, compressionLevel
, includeBaseDirectory
);
result = true;
}
catch (Exception excp)
{
//doSomething or rethrow upper level
}
return result;
}
}
}



Name	Date modified	Type	Size
ALMWorld	04.06.2012 16:04	Compressed (zipp...	175 KB
BCL	04.06.2012 16:04	Application	6 KB
BCL.exe	04.06.2012 15:48	XML Configuratio...	1 KB
BCL	04.06.2012 16:04	Program Debug D...	14 KB
BCL.vshost	04.06.2012 15:48	Application	22 KB
BCL.vshost.exe	04.06.2012 15:48	XML Configuratio...	1 KB
BCL.vshost.exe.manifest	17.03.2010 21:39	MANIFEST File	1 KB

Not : System.IO.Compression.dll ile System.IO.Compression.FileSystem.dll referanslarını eklemek gerekiyor. Ayrıca örnek RC(Release Candidate) sürümüdür. Yani Release sürümde değişiklikler olabilir unutmayın 

Tek Fotoluk İpucu 54 - Control Nerede?

Pazartesi, 4 Haziran 2012 08:15

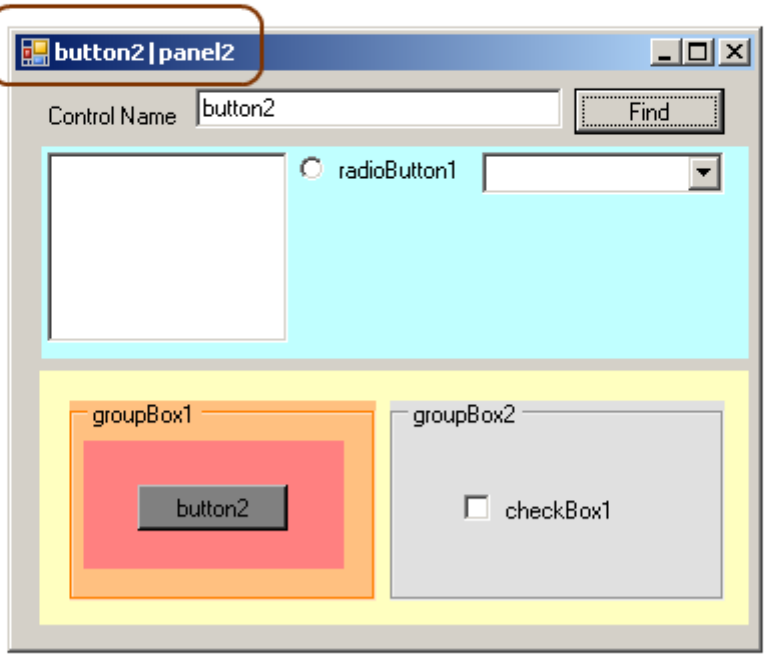
Tek Fotoluk Ipucu, Windows Forms, Controls, Recursive Methods, Extension Methods

Merhaba Arkadaşlar,
Diyelim ki çalışma zamanında, **Windows Forms**' un içerisindeki bir kontrolü*(Control tipinden bir nesne örneğini)* kodla buldurmanız ve üzerinde bir işlem yaptırmanız gerekiyor. Hatta formunuzun da, otomatik olarak bir veri kaynağına göre üretildiğini ve kontrollerin de iç içe gelecek şekilde yerleştirildiğini düşünün. Kodun belirli bir kontrol üzerinde işlem yapması için önce onu bulması gerekir değil mi? Ancak control ağaç yapısının da çalışma zamanında üretilmesi söz konusudur.
Ne yaparsınız? Bir **Recursive** metod ile bunu çözebilir misiniz? Hatta bunu bir **Extension Method**olarak tasarlayıp herhangibir **Container Control** için de çalışacak hale getirmek istemez misiniz? Buyrun öyleyse

Form1.cs

WindowsFormsApplication1.Form1 | btnFindControl_Click(object sender, System.Ev

```
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        public Form1()
        {
            InitializeComponent();
        }

        private void btnFindControl_Click(object sender, System.EventArgs e)
        {
            Control myControl=this.WhereIsMyControl(txtControlName.Text);
            if(myControl!=null)
                Text = string.Format("{0}|{1}"
                    , myControl.Name
                    , myControl.Parent.Name);
        }
    }

    public static class ControlExtensions
    {
        public static Control WhereIsMyControl(this Control control
            , string controlName)
        {
            foreach (Control currentControl in control.Controls)
            {
                if (currentControl.Name.ToLower() == controlName.ToLower())
                    return currentControl;

                Control childControl =
                    WhereIsMyControl(currentControl, controlName);
                if (childControl != null)
                    return childControl;
            }
            return null;
        }
    }
}
```



100 %

Bir sonraki ipucunda görüşmek dileğiyle

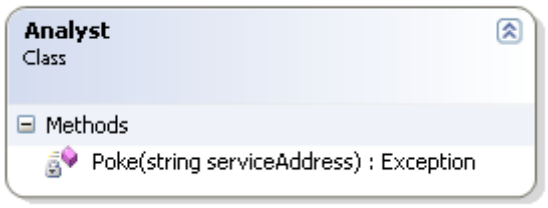
Servisleri Monitor Edelim

Cuma, 1 Haziran 2012 10:12

Wcf, Xml Web Services, Asmx, Httpwebrequest, Httpwebresponse

Merhaba Arkadaşlar,
Banka gibi, pek çok farklı sistemin bir arada yer aldığı ve çalıştığı, çoğu zaman heterojen yapıda olan büyük çaplı çözümlerde, servislerin sıklıkla kullanıldığını görürüz. Çok basit bir operasyonel uygulama bile, çalışacağı veri kümesini sadece veritabanı kaynağı üzerinden değil, sistem içerisinde yer alan başka kanallardan da almak durumunda kalabilir. Tam tersi durumda söz konusudur. Gerçekleşen bir toplu işlemin içerisinde, akışın çeşitli noktalarında yine servisler devreye girerek diğer sistemlerin haberdar edilmesi de söz konusudur. İşte böyle durumlarda, sistem içerisindeki parçalar arasındaki entegrasyonun sağlanabilmesi amacıyla, servis bazlı çözümlere sıklıkla başvurulur.
İşin içerisine servisler girdiğinde, bunların anlık durumunlarını izlemek, ayakta olup olmadıklarını görmek veya zaman içerisindeki hareketliliklerine bakarak istikrarlı yapılarının nasıl olduğunu analiz etmek isteyebiliriz. Pek tabi bunun için birden çok 3ncü parti tool olduğunu biliyoruz. Söz gelimi IIS tarafında **Windows Server AppFabric** aracından yararlanılarak son derece etkili ve gelişmiş izleme ve kontrol mekanizmaları gerçekleştirilebilmektedir. Lakin bazı bankacılık sistemlerinde teknoloji adaptasyonu beklendiği kadar hızlı değildir. **Windows Server AppFabric** gibi bir tool' un geçişi, yıllarca **XP** üzerinde çalışan bankanın **Windows 7'** ye geçişindeki gecikme kadar sancılı ve sıkıntılı olabilir. Hatta bankanın bir sonraki teknolojik yenilenme sürecinde ortada **Windows Server AppFabric**' ten tamamen farklı bir ürün de bulunabilir 😕
Peki böyle bir durumda ne yapabiliriz?
Elbette kendi başımızın çaresine bakmamız gerekecektir. Bir başka deyişle **Monitoring** aracını kendi imkanlarımızla yazmayı düşünebiliriz. İşte bu yazımızda **ASMX** tabanlı servisleri hatta **HTTP/HTTPS**protokolüne göre çalışan **WCF** servislerini nasıl izleyebileceğimizi görmeye çalışacak ve bununla ilişkili çekirdek bir tool yazacağız. İlk soru şu;
"Bir servisin ayakta olduğunu anlamak için ne yapabiliriz?"
Bazı servisler kendi durumlarını belirli periyotlarda çeşitli monitoring araçlarına bildirmek üzere genişletilmiştir. Hatta bazılarının çalışma zamanı motoru bunu default olarak sunmaktadır. Bazı monitoring araçlarıda, listelerinde yer alan servisleri belirli periyotlarda veya yöneticinin karar verdiği zaman aralıklarında kontrol ederek hayatta olup olmadıklarını anlamaya çalışır. Biz bu yolu tercih ediyor olacağız. Teorimiz ise oldukça

basit. Servis **URL** adresine bir talepte bulunacağız. Eğer bir**Exception/Error** dönmüyorsa bir başka deyişle bir **response** alabiliyorsak servisimizin ayakta olduğunu düşünebiliriz. Öyleyse ilk etapta bir **URL** adresi ile gelen sayfaya nasıl **request** atabiliriz buna bir bakalım. Bu amaçla **Kernel** isimli bir **kütüphane(Class Library)** projesi oluşturalım ve içerisine **Analyst** isimli bir sınıf ekleyelim.




```
using System;
using System.Text;
using System.Collections.Generic;
using System.Net;
namespace Kernel
{
   public class Analyst
   {
      private Exception Poke(string serviceAddress)
      {
         Exception result = null;
         HttpWebRequest request = null;
         HttpWebResponse response = null;
         try
         {
            request = (HttpWebRequest) WebRequest.Create(serviceAddress);
            request.Timeout = 2000;
            response = (HttpWebResponse) request.GetResponse();
         }
         catch (Exception exception)
         {
            result = exception;
         }
         finally
         {
            if(response!=null)
               response.Close();
         }
         return result;
      }
```

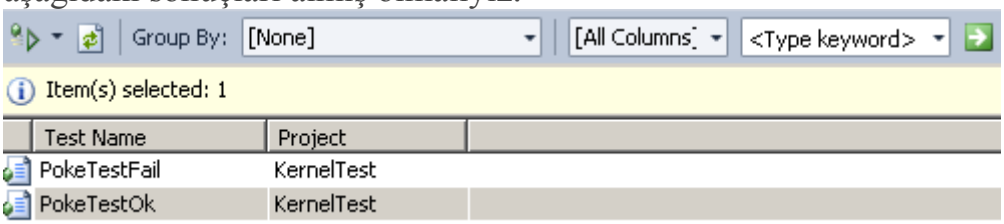
```
    }
}
```

Poke isimli metodumuz bir adres bilgisini alıp oluşan bir **Exception** var ise bunu geriye döndürmek üzere tasarlanmıştır. Eğer bir hata söz konusu değilse **null** değer dönecektir. Metodumuz içerisinde**HttpWebRequest** tipini kullanarak gelen adrese doğru bir **Http** talebinde bulunulmaktadır. Ardından bu **request** üzerinden belirtilen **timeout** süresi içerisinde bir **response** çekilip çekilemediğine bakılır. Eğer **response** geliyorsa söz konusu adresteki **resource'** un ayakta olduğunu düşünebiliriz. Şimdi yazmış olduğumuz bu metodu bir test edelim isterseniz. Bunun için **Unit Test** projesi oluşturarak ilerleyebiliriz. Örnek olarak aşağıdaki test metodları göz önüne alınabilir.

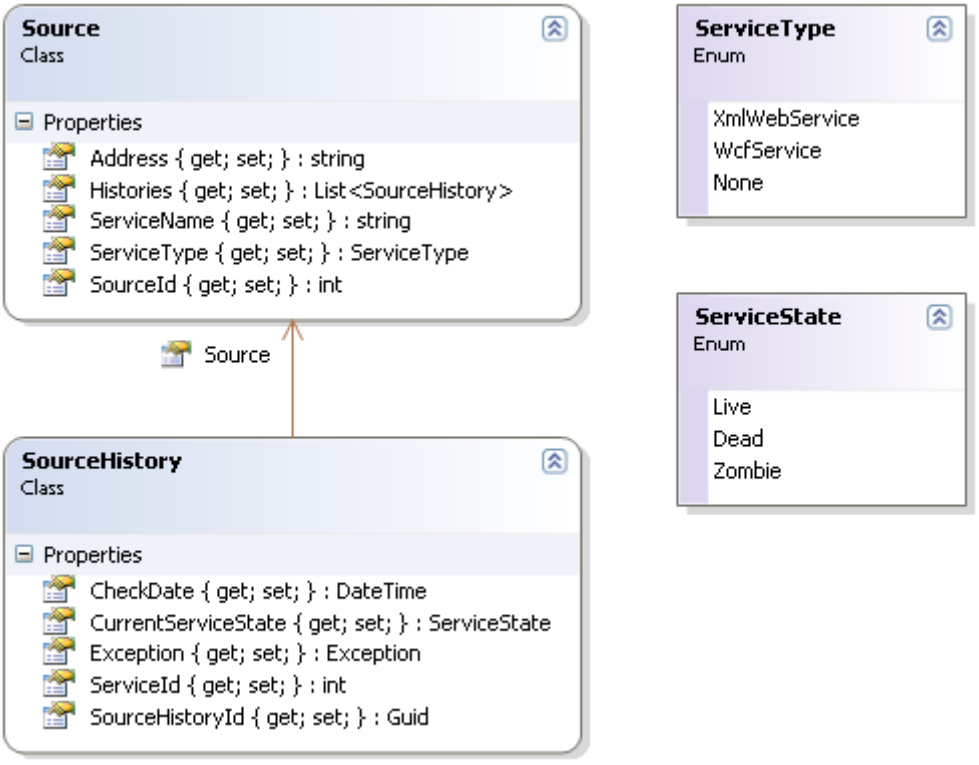
```
/// <summary>
/// Olmayan bir web adresi için test yapar
///</summary>
[TestMethod()]
[DeploymentItem("Kernel.dll")]
public void PokeTestFail()
{
    Analyst_Accessor target = new Analyst_Accessor();
    string serviceAddress = "http://www.yok.com/yok.asmx";
    Exception actual;
    actual = target.Poke(serviceAddress);
    Assert.AreNotEqual(null, actual);
}
/// <summary>
/// Var olan bir adres için test yapar
///</summary>
[TestMethod()]
[DeploymentItem("Kernel.dll")]
public void PokeTestOk()
{
    Analyst_Accessor target = new Analyst_Accessor();
    string serviceAddress =
"http://www.w3schools.com/webservices/tempconvert.asmx";
    Exception actual;
    actual = target.Poke(serviceAddress);
    Assert.AreEqual(null, actual);
}
```

İlk test metodumuzda servis adresi olarak olmayan bir **URL** bilgisi giriyoruz. Bu teste göre **Poke**metodundan bir **Exception** nesne örneğinin dönmesini beklemekteyiz. Diğer taraftan ikinci test metodu

içerisinde http://www.w3schools.com/webservices/tempconvert.asmx adresine bir talepte bulunuyoruz. Burada beklediğimiz ise herhangibir **Exception'** ın dönmemesi. Bir başka deyişle **Poke**metodunun **null** değer döndürmesini bekliyoruz. Testlerimizi çalıştırdığımızda aşağıdaki sonuçları almış olmalıyız.

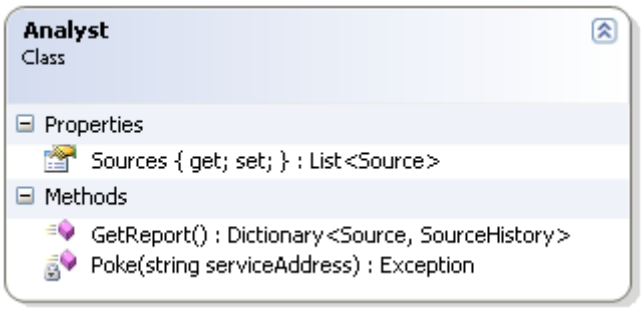


Sonuç itibariyle iki test de başarılı bir şekilde tamamlandı. Dolayısıyla bu çekirdek fonksiyonelliğimizin işe yarayacağını düşünebiliriz. Öyleyse artık uygulamamızın kalan kısmını geliştirmeye devam edelim.
Sistemsel olarak takip etmek istediğimiz servisler ve bu servislerin tarih içerisindeki yaşam durumlarını öğrenmek istiyoruz. Buna göre servislerimizin bilgisini ve her servisin tarih içerisindeki durum raporlarını tutacağımız bir yapı tasarlamamız gerekiyor. Aslında aşağıdaki tip modeli düşünüldüğünde söz konusu ilişkiyi bir ölçüde kurduğumuzu ifade edebiliriz.



Source tipi içerisinde servise ait adres bilgisini, tipini saklıyor olacağız. Bununla birlikte bir servisin tarih içerisindeki hareketliliklerini de saklamak istediğimizden arada bire çok ilişkiyi kuracağımız ikinci bir tipimiz daha yer alıyor. **SourceHistory** tipi içerisinde ise, yapılan **Poke** işlemine ait bilgiler tutulmaktadır. Bu işlemin **ne zaman** yapıldığı, **Exception** var ise buna ait bilgi, **hangi servisin**kontrol ediliği ve servisin **o anki durumu** gibi bilgiler tutulmaktadır. Dikkat edileceği üzere iki tip arasında

bir **Association** ilişkisi de kurulmuş durumdadır. Şimdi dilerseniz **Analyst** tipimizin geri kalan metodlarını yazmaya çalışalım. İlk olarak toplu bir servis listesi üzerinde işlem yapacak olan ana fonksiyonelliğimizi geliştirerek ilerleyebiliriz.




```
using System;
using System.Text;
using System.Collections.Generic;
using System.Net;
namespace Kernel
{
   public class Analyst
   {
      public List<Source> Sources { get; set; }
      public Dictionary<Source,SourceHistory> GetReport()
      {
         Dictionary<Source, SourceHistory> result = new Dictionary<Source,
SourceHistory>();
         Exception exception = null;
         foreach (Source source in Sources)
         {
            exception = Poke(source.Address);
            SourceHistory history = new SourceHistory
                                    {
                                       ServiceId=source.SourceId,
                                       CheckDate=DateTime.Now,
                                       CurrentServiceState =
exception!=null?ServiceState.Dead:ServiceState.Live,
                                       Exception=exception,
                                       Source = source,
                                       SourceHistoryId = Guid.NewGuid()
                                    };
            result.Add(source,history);
         }
         return result;
      }
```

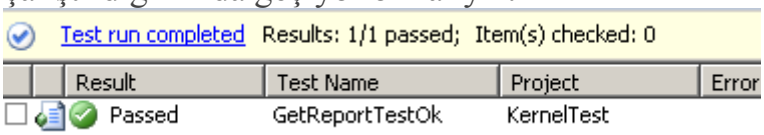
…

GetReport isimli metodumuz **Analyst** sınıfı içerisinde tanımlı olan **Sources** özelliğinin tüm içeriğini dolaşarak, her bir **Source** örneği için **Poke** metodunu çağırmakta ve sonuçları generic bir**Dictionary<Source,SourceHistory>** koleksiyonu içerisinde toplamaktadır. Yazmış olduğumuz metodun işe yarayıp yaramadığını görmek için yine bir test metodunu kullanabiliriz. Aynen aşağıda olduğu gibi

```
/// <summary>
/// Bir servis adres listesi için gerekli testi yapar
///</summary>
[TestMethod()]
public void GetReportTestOk()
{
   Analyst target = new Analyst();
   target.Sources = new List<Source>
                    {
                        new Source
                        {
                            Address = "http://www.yok.com/yok.svc",
                            Histories = null,
                            ServiceName = "Yok Servisi",
                            ServiceType = ServiceType.WcfService,
                            SourceId = 1
                        },
                        new Source
                        {
                            Address =
"http://www.w3schools.com/webservices/tempconvert.asmx",
                            Histories = null,
                            ServiceName = "Temp Convert",
                            ServiceType = ServiceType.XmlWebService,
                            SourceId = 2
                        }
                    };
   Dictionary<Source, SourceHistory> actual;
   actual = target.GetReport();
   Assert.AreEqual(target.Sources.Count, actual.Keys.Count);
}
```

Test metodumuzun çalışması sonrasında beklentimiz koleksiyon içerisinde yer alan her bir **Source**örneğine karşılık metoddan geriye bir karşılığının dönmesi. Bir başka deyişle **GetReport** metodunun dönüş referansına ait **Count** değeri ile **Sources** özelliğine

atanan koleksiyonun **Count** değerlerinin eşit olmasını bekliyoruz. Testimizi çalıştırdığımızda geçiyor olmalıyız.



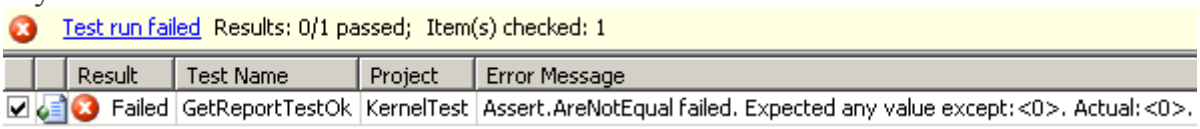
Elimizde temel fonksiyonellikler mevcut gibi. Ama halen daha eksiklikler var. Söz gelimi **Analyst**tipimizin tutarlı veriler ile çalışması gerekiyor. Yazımızın başında belirttiğimiz üzere biz **XML Web Servislerini** ve **HTTP** tabanlı **WCF** servislerini kontrol etmeyi planlıyoruz. Bu durumda **Http** veya**https** ile başlamayan ve bunlara ek olarak **svc** veya **asmx** ile bitmeyen adresleri işlemekatmamalıyız. Bunu koleksiyonu oluşturacağımız yerde yapacağımız bir kontrolle engelleyebileceğimiz gibi **Analyst** sınıfı içerisinde de ele alabiliriz. Dilerseniz ikinci seçeneği göz önüne alarak sınıf yapımızı biraz daha değiştirelim.

```
using System;
using System.Collections.Generic;
using System.Net;
using System.Text.RegularExpressions;
namespace Kernel
{
   public class Analyst
   {
      //public List<Source> Sources { get; set; }
      private List<Source> Sources = new List<Source>();
      public void AddSource(Source source)
      {
         Regex regex = new Regex(@"(http|https):\/\/[\w\-_]+(\.[\w\-_]+)+([\w\-
\.,@?^=%&amp;:/~\+#]*[\w\-\@?^=%&amp;/~\+#])?");
         string address = source.Address;
         if(regex.IsMatch(address))
            if(address.EndsWith(".asmx")||address.EndsWith(".svc"))
               Sources.Add(source);
      }
```

İlk olarak dışarıdan erişilebilen **Sources** koleksiyonunu **private** hale getirdik. Nitekim eleman ekleme işlemi sırasında çalıştırılması gereken bir doğrulama işlemi söz konusu. Bu koleksiyona eleman ekleme adımını **AddSource** metoduna verdik. Bu metod içerisinde **Regex** ifadesinden yararlanarak, gelen **Source** örneğine ait address bilgisinin geçerli bir **URL** olup olmadığını kontrol etmekteyiz. Bu işlemin ardından iş kuralımıza göre adresin **svc** veya **asmx** uzantılı olup olmadığına bakıyoruz. Eğer bu kriterler sağlanıyorsa güncel **Source** nesne örneğinin ilgili koleksiyona eklenmesi sağlanıyor. Tabi yapmış olduğumuz bu değişiklik nedeni ile test metodumuzu da güncellememiz gerekecek. Keşke en başından düşünseymişiz değil mi? 🙂

```
/// <summary>
/// Bir servis adres listesi için gerekli testi yapar
///</summary>
[TestMethod()]
public void GetReportTestOk()
{
   Analyst target = new Analyst();
   target.AddSource(
      new Source
         {
            Address = "http://www.yok.com/yok.sv",
            Histories = null,
            ServiceName = "Yok Servisi",
            ServiceType = ServiceType.WcfService,
            SourceId = 1
         }
      );
   target.AddSource(new Source
                  {
                     Address = "ftp://www.w3schools.com/webservices/tempconvert.asmx",
                     Histories = null,
                     ServiceName = "Temp Convert",
                     ServiceType = ServiceType.XmlWebService,
                     SourceId = 2
                  }
      );
   Dictionary<Source, SourceHistory> actual;
   actual = target.GetReport();
   Assert.AreNotEqual(0, actual.Keys.Count);
}
```

Dikkat edileceği üzere iki adreste istediğimiz normlara uygun değil. İlk adres **sv** ile biterken ikinci adresimiz **ftp** ile başlıyor. Buna göre test metodumuzun **Fail** etmesi gerekmektedir. Çünkü geçer şartımız **Keys.Count değerinin 0 dan farklı olmasıdır**. Buradaki senaryoda iki adreste geçersiz olduğundan koleksiyona eklenmeyecek ve eleman sayısı 0 olarak dönecektir.



Artık **Analyst** sınıfımızı terk edip arayüz tarafına geçebiliriz. **Windows Forms** şablonu olarak tasarlayacağımız **Monitor** uygulaması farklı bir proje de olabilir.

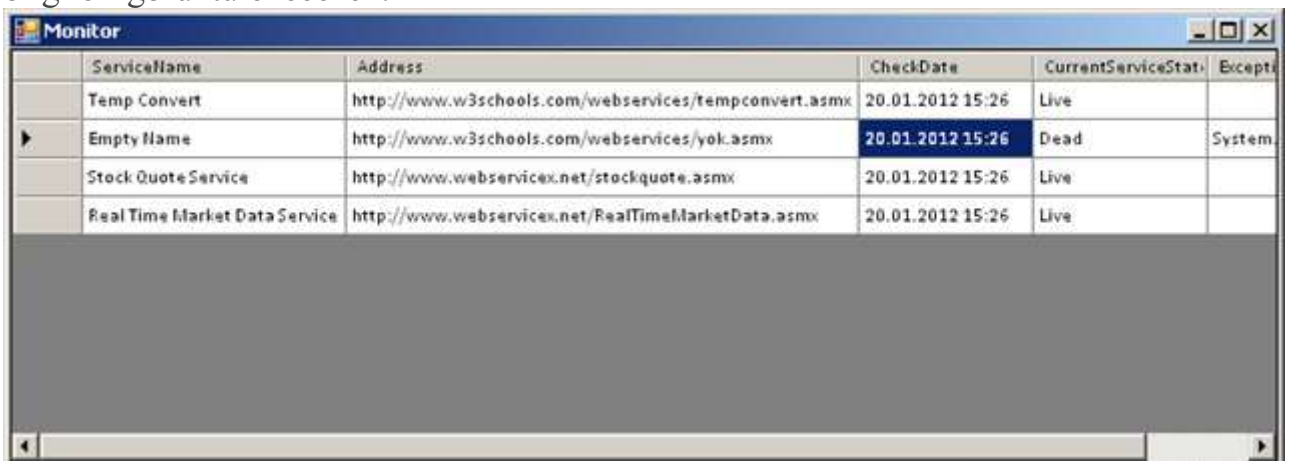
Nitekim **Kernel** isimli çekirdek kütüphanemiz herhangibir projeye referans edilerek kullanılabilir.
Form tasarımımız son derece sade. Sadece bir **DataGridView** kontrolü bulunmaktadır.
Kod içeriğini ise aşağıdaki gibi geliştirmeyi düşünebiliriz.

```
using System;
using System.Configuration;
using System.Linq;
using System.Windows.Forms;
using Kernel;
using System.Linq;
namespace Monitor
{
   public partial class frmMonitor : Form
   {
      private Analyst jackRyan = new Analyst();
      public frmMonitor()
      {
         InitializeComponent();
      }
      private void frmMonitor_Load(object sender, EventArgs e)
      {
         int interval = 10000;
         Int32.TryParse(ConfigurationManager.AppSettings["Interval"], out interval);
         timer1.Enabled = false;
         LoadSamples();
        timer1.Enabled = true;
      }
      private void LoadSamples()
      {
         jackRyan.AddSource(
            new Source
               {
                  Address = "http://www.w3schools.com/webservices/tempconvert.asmx",
                  Histories = null,
                  ServiceName = "Temp Convert",
                  ServiceType = ServiceType.XmlWebService,
                  SourceId = 1
               }
            );
        jackRyan.AddSource(
            new Source
```

```
            {
               Address = "http://www.w3schools.com/webservices/yok.asmx",
               Histories = null,
               ServiceName = "Empty Name",
               ServiceType = ServiceType.None,
               SourceId = 2
            }
         );
      jackRyan.AddSource(
         new Source
            {
               Address = "http://www.webservicex.net/stockquote.asmx",
               Histories = null,
               ServiceName = "Stock Quote Service",
               ServiceType = ServiceType.XmlWebService,
               SourceId = 3
            }
         );
      jackRyan.AddSource(
         new Source
            {
               Address = "http://www.webservicex.net/RealTimeMarketData.asmx",
               Histories = null,
               ServiceName = "Real Time Market Data Service",
               ServiceType = ServiceType.XmlWebService,
               SourceId = 4
            }
         );
   }
   private void timer1_Tick(object sender, EventArgs e)
   {
      var report = jackRyan.GetReport().Select((s, h) => new
                                                {
                                                   s.Key.ServiceName
                                                  ,s.Key.Address
                                                   ,s.Value.CheckDate
                                                   ,s.Value.CurrentServiceState
                                                   ,s.Value.Exception
                                                }).ToList();
      grdReport.DataSource = report;
   }
```

```
    }
}
```

Windows uygulamamızda bir **Timer** nesnesinden yararlanılmaktadır. **Form** ilk yüklenirken takip edilecek olan servis listesi yüklenir. Ardından **config** dosyasında belirtilen **Interval** değerine göre**Timer** nesne örneğinin **Tick** metodu belirli periyotlarda devreye girerek servislerin durum bilgisini çeker. Sonuçlar bir **anonymous type** içerisine örneklenip liste haline getirilerek **DataGridView**kontrolünde gösterilmektedir. Örneği çalıştırdığımızda 10 saniye de bir yapılan tetiklemeler sonucu, ilgili servislerin anlık bilgileri görüntülenecektir.



Buraya kadar yaptıklarımızı düşündüğümüzde uygulamamızın eksik kalan pek çok kısmı olduğu fark edilmektedir. Söz gelim;

- **Servis** bilgilerinin aslında bir **Repository** ortamı üzerinden yükleniyor olması ve yine **History** bilgilerinin de bu ortama kayıt ediliyor olması çok daha doğru bir yaklaşım olacaktır. Burada bir **SQL** veritabanı gibi ilişkisel yapıyı veri üzerinde kurgulayabileceğimiz bir sistem son derece yararlı olabilir.
- Buna ek olarak Windows uygulamasının daha responsible olması için **Timer** tipi yerine**BackgroundWorker** kontrolünün kullanılması göz önüne alınabilir.
- Diğer yandan **DataGridView** kontrolü dışında daha farklı bir **DashBoard** görünümü tasarlanabilir. Örneğin grafiksel bir gösterim yapılabilir.

Bu tip kısımların geliştirilmesini siz değerli okurlarıma bırakıyorum. Hatta sıkı takipçilerim için bir ödev olsun. Ben belki bir sonraki makalede bu geliştirmeleri de ele alabilirim. Kim bilir 😃

Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim 😉

Monitoring.zip (2,27 mb)

WCF Tarafında Task Bazlı Asenkron Operasyonlar

Perşembe, 24 Mayıs 2012 23:50

Wcf, Windows Communication Foundation, Task Parallel Library, Tpl, Iasyncresult

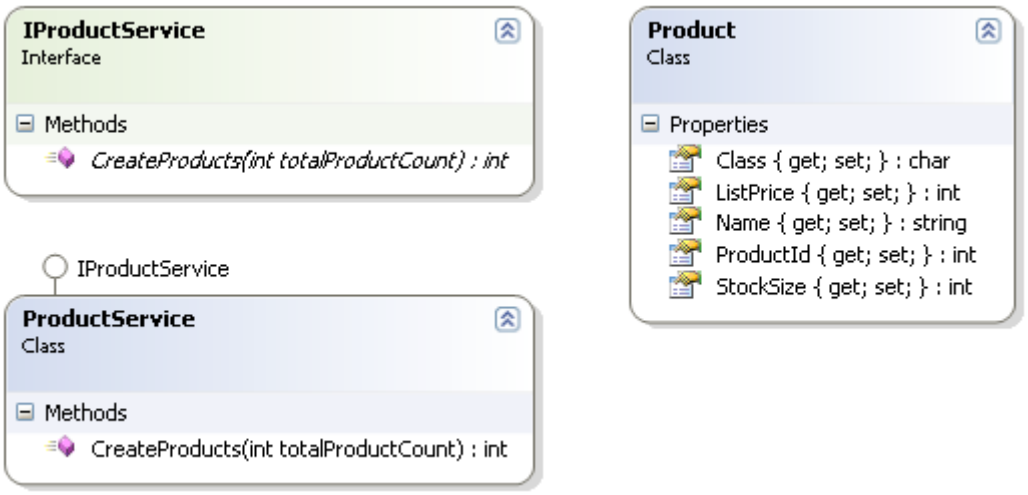
Merhaba Arkadaşlar,
Yandaki karikatür, aşağıdaki yazıyı bitirdiğim zaman aradığım giriş resmi ile ilişkili olarak karşıma çıkan örneklerden sadece bir tanesiydi. Beni epey bir güldürdüğünü ve neşelendirdiğini ifade edebilirim 😃 Konumuz işlerimizi birilerine yönlendirip yan gelip yatmak değil elbette, ama ona benzer olduğunu ifade edebilirim.
Bu adamın görevlerini başkaların atadıktan sonra, söz konusu işler yapılırken başka işlere*(örneğin koltuğunda şöyle bir geriye doğru yaslanarak vakit geçirmek)* yönelebildiğine odaklanmaya çalışalım..**Net Framework** tarafındaki **Delegate** tipleri de benzer bir ihtiyacı karşılamıyor mu? 😉 Asenkron olarak fonksiyonların çağırılabilmesini sağlamak*(Elbette başka yetenekleri de var ama bu en önemlileri arasında sayılabilir)*
Uzun bir zamandır **.Net Framework** içerisinde, fonksiyonların asenkronize edilmesi üzerinde çalışılmaktadır. Daha önceleri **Thread** bazlı veya **Delegate** tipleri ile gerçekleştirdiğimiz asenkron çağırımlar, **.Net Framework 4.0'** a gömülü olarak gelen **Task Parallel Library** sayesinde daha da gelişmiş ve alt yapının her noktasına enjekte edilebilir olmuştur. Şu günlerde **.Net Framework 4.5**ile birlikte gündeme gelen ve uzun zamandır da haberdar olduğumuz **async**, **await** gibi anahtar kelimeler de, temel de **Task** tiplerine dayanmaktadır. Bir başka deyişle **TPL** kütüphanesi ve içeriği, ilerleyen zamanlarda **.Net Framework'** ün pek çok önemli alt yapısında etkisini hissettirecektir.
Bu yazımızda **Task** tiplerinden yararlanarak, **WCF(Windows Communication Foundation)**servislerinde asenkron operasyon tanımlamalarının nasıl yapılabileceğini ve incelemeye çalışıyor olacağız. Gerçekleştirmeyi planladığımız işlemlerde önemli olan nokta ise, asenkron yürütmelerin**istemci(Client)** tarafında değil, servis tarafındaki operasyonlar için söz konusu olmasıdır. Bir başka deyişle **eş zamanlı olarak çalışabilen(Concurrent)** servis operasyonlarının, **Task** tiplerinden yararlanarak nasıl asenkron hale getirilebileceğini görmeye çalışacağız. Elbette bu asenkronize edilmiş metodlar servis tarafında değerlendirilen bir yaklaşımı içerecektir.
Aslında bir servis operasyonunun asenkron çalışacak hale getirilmesi, **delegate** tipleri ile kullanabildiğimiz*(hatta Ado.Net 2.0' dan bu yana pek çok XCommand gibi tipinde var olan)* **Begin**… ve **End**… ön ekli metod oluşumlarının uygulanmasından ibarettir.
Aşağıdaki örnek kod parçasında sadece hatırlatıcı olması açısından bir **Delegate** tipi

üzerinden ilgili **BeginInvoke** ve **EndInvoke**metodlarına nasıl ulaşılabildiği gösterilmeye çalışılmıştır. Örnekte **Async Callback** modeli değerlendirilmiştir. Bir başla deyişle asenkron olarak başlatılan metod işleyişini tamamladığında, uygulama ortamındaki başka bir geri bildirim fonksiyonu tetiklenmektedir.

```
using System;
using System.Collections.Generic;
namespace AzonTestClient
{
   // Örnek bir generic temsilci
   delegate int SaveToFileDelegate<T>(IEnumerable<T> list);
   class Program
   {
      static void Main(string[] args)
      {
         #region Async Callback modeli
         // Temsilci örneklenir
         SaveToFileDelegate<object> dlg=new
SaveToFileDelegate<object>(SaveToFile);
         // BeginInvoke çağrısı ile dlg' nin işaret ettiği SaveToFile metodunun çağırılması ve
kodun ifade sonundan itibaren akmaya devam etmesi sağlanır
         // ilk parametre SaveToFile metodunun alacağı değişken, ikinci parametre işlem
bittiğinde tetiklenecek geri bildirim fonksiyonunun işaretçisi olan temsilci, üçüncü
parametre ise geri bildirim metodunda AsyncState özelliği üzerinden yakalanacan Delegate
referansı
         IAsyncResult asyncResult = dlg.BeginInvoke(new List<object>(), new
AsyncCallback(Callback), dlg);
         // akış buradan kesilmeden devam eder
         #endregion
         Console.ReadLine();
      }
      // Uzun süreli işlem içeren örnek metod
      static int SaveToFile<T>(IEnumerable<T> list)
      {
         //TODO: Çok büyük boyutlu bir veri içeriğinin dosyaya yazılması söz konusudur.
Zaman alan bir işlem olarak düşünülebilir
         return 1;
      }
      // Async Callback tekniğine göre SaveToFileDelegate<T> ile işaret edilen metod
sonlandığında devreye girecek olan geri bildirim fonksiyonu
      static void Callback(IAsyncResult asyncResult)
      {
```

```
            // EndInvoke çağrısı ile ilişkili olan delegate tipi yakalanır
            var dlg = asyncResult.AsyncState as SaveToFileDelegate<object>;
            // delegate tipi üzerinden EndInvoke çağrısı yapılarak SaveToFile metodunun
sonucu alınır
            int result = dlg.EndInvoke(asyncResult);
            // Sonuç değerlendirilir
        }
    }
}
```

Delegate tiplerinin kullanımı göz önüne alındığında, **Polling**, **Callback**, **WaitHandle** gibi modelleri destekleyebilen **BeginInvoke** ve **EndInvoke** metod çağırımlarının söz konusu olduğu bilinmektedir. **BeginInvoke** metodu **IAsyncResult arayüzü(Interface)** tipinden bir referans döndürmekte olup, program kod akışının izleyen satırdan devam edebilmesini sağlmaktadır. Pek tabi, **EndInvoke** metodu içerisinde de ilgili **IAsyncResult** arayüz referansından yararlanarak sonuçların alınması söz konusudur. Bu iki metod arasındaki çalışma süreci, ana sürece*(varsayılan olarak uygulamanın Main Thread' i olarak da düşünebiliriz)* veya diğer süreçlere paralel olarak yürütülmektedir.
Dolayısıyla benzer bir yaklaşımı **WCF** servislerinde, asenkron hale getirilmek istenen operasyonlar için de düşünebiliriz. Gelin öncelikle senkron olarak uzun süren işlem içeren örnek bir **WCF** servisini geliştirelim. Bu amaçla aşağıdaki **servis sözleşmesini(Service Contract)** içeren bir servis uygulamasını göz önüne alabiliriz.



IProductService arayüz tipi(Interface);

```
using System.ServiceModel;
namespace AzonServices
{
    [ServiceContract]
    public interface IProductService
    {
        [OperationContract]
```

```
        int CreateProducts(int totalProductCount);
    }
}
```

ProductService sınıfı;

```
using System;
using System.Collections.Generic;
namespace AzonServices
{
    public class ProductService
      : IProductService
    {
        public int CreateProducts(int totalProductCount)
        {
            int createdProductCount = 0;
            char[] classes = {'C', 'D', 'E', 'L', 'S'};
            List<Product> products = new List<Product>();
            Random randomizer = new Random();
            for (int i = 0; i < totalProductCount; i++)
            {
                Product newProduct = new Product
                                     {
                                        ProductId=i,
                                        Name="PRD-"+i.ToString(),
                                        ListPrice=randomizer.Next(1,100),
                                        StockSize=randomizer.Next(50,500),
                                        Class=classes[randomizer.Next(0,classes.Length)]
                                     };
                products.Add(newProduct);
                createdProductCount++;
            }
            //TODO: product listesinin veritabanın yazılma veya dosyaya kayıt edilme işlemi
yapılacak
            return createdProductCount;
        }
    }
}
```

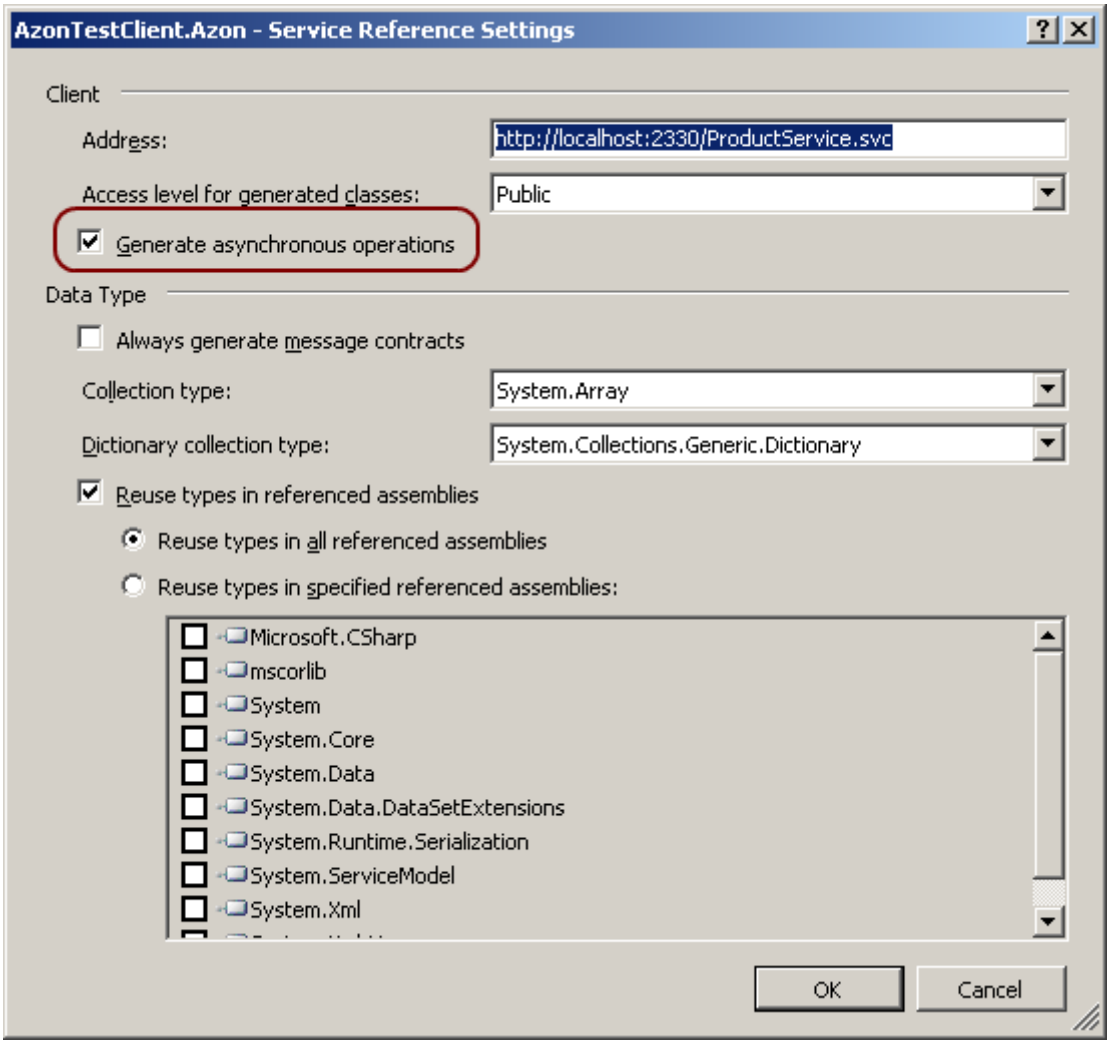
Product POCO sınıfı;

```
namespace AzonServices
{
    class Product
    {
```

```
        public char Class { get; set; }
        public int StockSize { get; set; }
        public int ListPrice { get; set; }
        public string Name { get; set; }
        public int ProductId { get; set; }
    }
}
```

CreateProducts metodu istemciden aldığı toplam miktara göre bir **Product** listesi üretmektedir. Test amaçlı olarak üretilen bu listenin içerisinde yer alan ürün bilgileri, rastgele değerlerden oluşmaktadır. İstemcinin vereceği maksimum ürün sayısına göre ilgili operasyonun uzun sürmesi olasıdır. İstemci açısından bakıldığında, söz konusu operasyonunun tamamlanana kadar uygulama içerisinde beklenmesine gerek yoktur. Bu, zaten istemci tarafında sahip olduğumuz ilgili servisi asenkron olarak çağırma yeteneğidir. Hatta servisi istemci tarafına eklerken **Add Service Reference** arabirimindeki ilgili opsiyon etkinleştirilerek servis operasyonlarının **olay bazlı(Event Based)** asenkron çağırım versiyonlarının üretilmesi sağlanabilmektedir.

İstemci tarafında servis çağırımlarına ait asenkron operasyon desteğini etkinleştirmek için, Generate asynchronous operations özelliğinin işaretlenmiş olması gerekmektedir.



Sonuç olarak üretilen **Async** uzantılı asenkron çağırım metodu ve servis operasyon işleminin tamamlanması sonrası devreye girecek fonksiyonu işaret edecek olan **olay(Event),** kod tarafında değerlendirilebilir olacaktır.

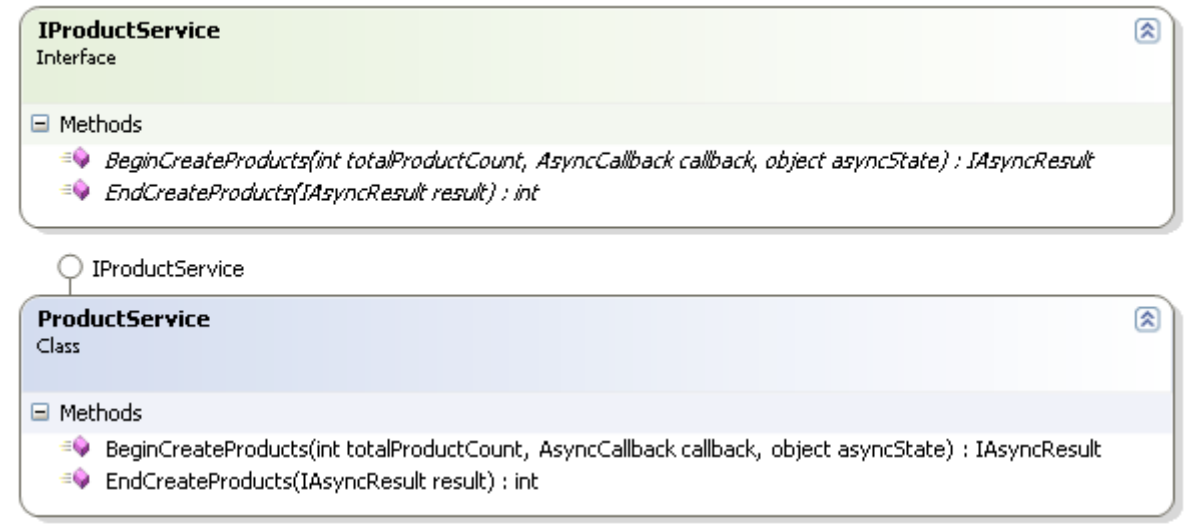
```
ProductServiceClient proxy = new ProductServiceClient("BasicHttpBinding_IProductService");
int result=proxy.CreateProducts(1000000);
```

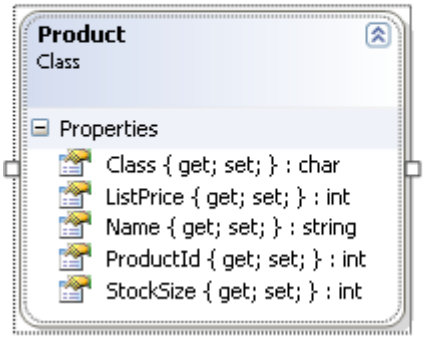
```
proxy.
    Abort
    BeginCreateProducts
    ChannelFactory
    ClientCredentials
    Close
    CreateProducts
    CreateProductsAsync
    CreateProductsCompleted
    DisplayInitializationUI
    EndCreateProducts
    Endpoint
));
ir tuşa basınız");
(int totalProductCount):void
(int totalProductCount, object userState):void
```

Gayet güzel 🙂 Buraya kadar ki kısımda zaten pek bir sıkıntı yok açıkçası. Ancak şöyle bir durum da söz konusu,

Servise eş zamanlı olarak gelen çağrılarda ve tek bir servis örneğinin(Instance) oluşmasının tercih edildiği durumlarda, servis üzerindeki bu yük nasıl asenkronize edilebilir?

İşte yazımızın asıl konusu da budur.
Başlarda da belirttiğimiz gibi servis tarafında **IAsyncResult** arayüzü ve **Task** tiplerini kullanarak, **eş zamanlı(Concurrent)** olarak gelen çağrılarda, ilgili operasyonların servis tarafını gereksiz yere duraksatması engellenebilir. Bunun için **IProductService servis sözleşmesini(Service Contract)**ve **ProductService** sınıfını aşağıdaki gibi değiştirmemiz yeterli olacaktır.
Önce **Class Diagram** üzerinden ilgili değişikliklere bir bakalım.





IProductService arayüzünün yeni versiyonu;

```
using System.ServiceModel;
using System;
namespace AzonServices
{
   [ServiceContract]
   public interface IProductService
   {
      [OperationContract(AsyncPattern=true, Action="CreateProducts",
Name="CreateProducts", ReplyAction ="CreateProductsReply")]
      IAsyncResult BeginCreateProducts(int totalProductCount,AsyncCallback
callback,object asyncState);
```

```
        int EndCreateProducts(IAsyncResult result);
    }
}
```

Servis sözleşmesinde **Begin** ve **End** ön ekleri ile başlayan iki metod yer almaktadır. Dikkat edilmesi gereken noktalardan
birisi, **EndCreateProducts** metodunun **OperationContractniteliği(Attribute)** ile imzalan mamış oluşudur. Nitekim bu metod, **BeginCreateProducts**metodunun tamamlanması sonucu devreye giren metod olmakla birlikte, sadece servis tarafını ilgilendiren bir fonksiyondur. Dolayısıyla istemci tarafına açılmasına söz konusu değildir.
BeginCreateProducts metoduna ait **OperationContract** niteliğinde ise bazı özellikler set edilmiştir. Herşeyden önce ilgili operasyonun **Asenkron** desene uygun olarak çalışacağının belirtilmesi gerekmektedir. Bu amaçla **AsyncPattern** özelliğine **true** değeri verilmiştir. Diğer taraftan aksiyon, operasyonun istemci tarafından görünecek olan adı ve istemci tarafında verilecek olan cevaba ait **Action** bilgisi de ilgili özelliklerce set edilmiştir.
BeginCreateProducts metodu ve **EndCreateProducts** metodlarının şema yapılarına bakıldığında,**temsilci(Delegate)** tiplerinin **BeginInvoke** ve **EndInvoke** metodlarından farksız oldukları gözlemlenmektedir. *(İsimlendirme standartdını bozmamak açısından da bu şekilde bir adlandırma tercih edilmelidir)*
BeginCreateProducts metodu, ilk parametre olarak istemciden gelecek
olan **integer** değeri almaktadır. Son iki parametre ise değişmez sırada olmalıdır. Bunlardan birisi **Callback** metodunu işaret edecek olan **AsyncCallback** temsilcisi iken, son parametre de **EndCreateProducts** metodu içerisinde **BeginCreateProducts** fonksiyonunda başlatılan **Task** örneğini yakalamak ve dolasıyıyla sonucunu almak üzere
kullanılan **object** referansıdır.
ProductService sınıfının yeni versiyonu;

```
using System;
using System.Collections.Generic;
using System.ServiceModel;
using System.Threading.Tasks;
namespace AzonServices
{
    [ServiceBehavior(InstanceContextMode= InstanceContextMode.Single,
ConcurrencyMode= ConcurrencyMode.Multiple)]
    public class ProductService
        : IProductService
    {
        public IAsyncResult BeginCreateProducts(int totalProductCount, AsyncCallback
callback, object asyncState)
        {
            var task = new Task<int>((s) =>
                                {
```

```
                        int createdProductCount = 0;
                        char[] classes = {'C', 'D', 'E', 'L', 'S'};
                        List<Product> products = new List<Product>();
                        Random randomizer = new Random();
                        for (int i = 0; i < totalProductCount; i++)
                        {
                            Product newProduct = new Product
                                        {
                                            ProductId = i,
                                            Name = "PRD-" + i.ToString(),
                                            ListPrice = randomizer.Next(1, 100),
                                            StockSize = randomizer.Next(50, 500),
                                            Class =
                                                classes[
                                                    randomizer.Next(0, classes.Length)
                                                ]
                                        };
                            products.Add(newProduct);
                            createdProductCount++;
                        }
                        return createdProductCount;
                    }, asyncState);
            task.ContinueWith((t) => { callback(t); });
            task.Start();
            return task;
        }
        public int EndCreateProducts(IAsyncResult result)
        {
            var task = (Task<int>) result;
            return task.Result;
        }
    }
}
```

ProductService tipinin ilk dikkat çeken noktalarından birisi, **ServiceBehvaior** niteliğinde set edilen özelliklerdir. Servisin bellek üzerinde tek bir örnek olarak oluşturulması belirlendikten sonra**(InstanceContextMode.Single),** servis operasyonlarında **Async** değeri **true** olanlara da eş zamanlı olarak erşilebileceği ifade edilmektedir**(ConcurrencyMode.Multiple)**

BeginCreateProducts metodunun içerisinde **Task** tipinden yararlanıldığı görülmektedir. **Asenkron**olarak yürütülmek istenen operasyona ait kod parçaları **Task** tipi örneklenirken ilgili **isimsiz metod(Anonymous Method)** içerisine yazılmıştır. Bu

örnekte **Task** tipinin **generic** bir versiyonun kullanıldığı görülmektedir. Nitekim metod geriye **int** tipinden bir sonuç döndürecek şekilde tasarlanmıştır. **Task** tipinin örneklenmesi sırasında kullanılan **asyncState** nesne örneği**EndCreateProducts** metodu içerisinde yakalanacak olan **Task** nesne örneğine ait referans olacaktır.
İlerleyen satırlarda **task** nesne örneğinin tamamlanması sonucu **callback** değişkeni ile ifade edilen çalışma zamanı metodunun tetiklenmesi ve **t** isimli **Task** değişkeninin*(ki bu t, task isimli değişkeni ifade etmektedir)* ilgili geri bildirim metodunun*(ki EndCreateProducts olmaktadır)* **IAsyncResult**arayüzüne atanması
sağlanır*(**ContinueWith** kısmı).* **Start** metodu ile bildiğiniz üzere ilgili task örneği başlatılmaktadır. Bundan sonraki aşama ise oldukça
basittir. **EndCreateProducts** metoduna gelecek olan **IAsyncResult** arayüzünden yararlanılarak çalışma zamanındaki **Task<int>** referansı yakalanmakta ve **Result** özelliği ile çalışma sonucu üretilen tamsayı değeri geriye, bir başka deyişle itemci tarafına döndürülmektedir.
Biraz karmaşık gözüken bir desen olduğunun farkındayım 😕 Ancak kalıp olarak düşünüldüğünde pek çok servis operasyonuna kolayca entegre edilebilecek bir yapı olarak düşünülebilir. Elbette bu konuyu bir de, **.Net Framework 4.5'** e entegre olarak gelen yeni **async** ve **await** anahtar kelimelerini göz önüne alarak değerlendirmekte yarar vardır. Bunu da ilerleyen zamanlarda incelemeyi planlamaktayım 🙂 Böylece geldik bir yazımızın daha sonuna. Bir sonraki yazımızda görüşünceye dek hepinize mutlu günler dilerim.
AzonServices.zip (67,73 kb)

Tek Fotoluk İpucu 53 - Tarih Dönüşümünde Extract Kullanımı

Salı, 15 Mayıs 2012 11:48

Datetime, Tryparseexact, C#, String, Convert

Merhaba Arkadaşlar,
Diyelim ki text tabanlı veya benzeri bir dosyadan satır bazlı veri okuma ve aktarma işlemi gerçekleştiriyorsunuz. Bu veri dosyasındaki alanlardan birisinde **16022012 094500** gibi bir tarih bilgisi tutulduğunu varsayalım. Kodunuzun bu alanı **DateTime** tipine çevirmesi işleminde **Try**kontrolünü kendi içinde yapan ve **string** içerisindeki harflerin tarihsel anlamda neye karşılık geldiğini belirtmemize yarayan bir fonksiyon olduğunu biliyor muydunuz?

Program.cs

ConsoleApplication4.Program

```
using System;
using System.Globalization;

namespace ConsoleApplication4
{
    class Program
    {
        static void Main(string[] args)
        {
            string unitDateString = "16022012 094700";
            DateTime unitDate=DateTime.Now;
            string format = "ddMMyyyy hhmmss";
            if (DateTime.TryParseExact(
                unitDateString
                ,format
                ,null
                , DateTimeStyles.None
                ,out unitDate))
            {
                Console.WriteLine(unitDate.ToString());
            }
        }
    }
}
```

```
C:\WINNT\system32\cmd.exe
16.02.2012 09:47:00
Press any key to continue . . .
```

WorldBank, OData ve ASP.Net Web API HttpClient Kullanımı

Pazartesi, 14 Mayıs 2012 18:30

Asp.Net Web Api, Wcf Web Api, Httpclient, Wcf, Odata, Worldbank, Xml, Json

Merhaba Arkadaşlar,
1999 yılında girdiğim yüksek lisans**(MBA-Master of Business Administrator)**programını tamamlarken, bitirme projemde**"Türkiye' nin Dünya Bankası borçlanmalarını"** ele almaya çalışmıştım. Haliyle o dönemlerde ve geçmişte, ülkemizin Worldbank üzerinden yaptığı borçlanmalara ait istatistiki bilgilere oldukça fazla ihtiyacım vardı. O kütüphane bu kütüphane gezmek dışında, dünya bankası internet sitesinden yayınlanan istatistik bazlı raporları da değerlendirmeye alıyordum. Yaklaşık olarak 60 sayfalık bir döküman oluşturmayı başarmıştım. Sunumumu yaptım, vardığım sonuçları değerli hocalarım ile paylaştım
Zaman hızla geçti tabi ki. Bir baktım **Microsoft .Net** teknolojileri ile uğraşıyor ve yazılım geliştirici olarak kariyerimi devam ettiriyorum. Zaman hızla geçiyordu geçmesine ancak ondan daha hızlı hareket etmek isteyen de bir teknoloji vardı ortada. **.Net Framework** platformunun gelişmesine ayak uydurabilmek gerçekten zorlaşıyordu. Ama tabi hepimiz için ortaya kullanışlı ve vizyonumuzu geliştiren ürünler çıkarttıkları da bir gerçekti.
Bugüne baktığımızda pek çok dünya firmasının dışarıya açık kaynaklı veri sunarken*(özellike OData-Open Data Protocol formatında)*, **Microsoft'** un da bu veri sunumlarını etkin bir şekilde ele almamız için getirdiği yeniliklerini görmekteyiz. Şöyle bir kaç sene geriye gidelim dilerseniz
Microsoft önce servis odaklı yaklaşımını değiştirerek **Windows Communication Foundation**altyapısını duyurdu. Hemen ardından **WCF** hızla gelişti ve pek çok **Microsoft** ürününün servis uç noktalarında yerini almaya başladı.
Derken **WCF'** e **Web programlama modeli(Web Programming Model)** için destekler eklendi. Artık **REST(Representational State Transefer)** odaklı servisleri yayınlamak ve hatta kullanmak mümkün hale gelmeye başladı. Çok basit anlamda **HTTP**protokolünün **GET,POST,PUT,DELETE** gibi metodlarına göre hizmet verebilen ve bu nedenle bir proxy ihtiyacını ortadan kaldırıp platform bağımsızlığı getiren servis yayınlama modeline destek söz konusu idi. Söz konusu model daha da geliştirildi. Özellikle **Code Plex** tarafında **WCF Rest Service API'** si duyuruldu ve programlama modeli **Astoria** kod adlı **WCF Data Service'** lerinde çekirdek yapı taşı haline geldi.

Söz konusu web programlama modeli odaklı WCF servisleri o kadar popüler olmaya başladı ki, onu**Asp.Net MVC** gibi **data-centric** uygulamalarda daha sık görmeye başladık. Hal böyle olunca **WCF**takımı, **WCF Web API** isimli bir kütüphaneyi kullanıma sundu. Şu an geldiğimiz noktada ise bu yapı isim değiştirerek*(ASP.NET Web API)* doğrudan **ASP.NET MVC 4.0 Beta** sürümüne entegre edildi. Artık **.Net Framework 4.5** sürümünde **ASP.NET Web API** gömülü olarak gelecek ve **2012**sonundan itibaren **WCF Web API**' ye olan destek kalkacak.
İster **WCF Web API** olsun, ister gündemimize yeni yeni giren **ASP.NET Web API** olsun, birbirlerinden ayrıldıkarı yönler olduğu kadar, ortak noktalarıda bulunmakta.
Örneğin, **OData, XML, JSON** gibi çıktı üretimleri sunan **REST** servislerinin daha kolay kullanımı için geliştirilmiş olan**HttpClient** tipi
Şimdi buraya kadar yazdıklarımızı bir toparlayalım. Dünya bankası artık verilerini **OData** formatında olacak şekilde dış dünyaya sunmakta. Bu anlamda **Developer'** lar için bir web sayfaları bile bulunuyor*(Developer' lar için kaynak http://data.worldbank.org/node/209)* Bu sayfada söz konusu**Worldbank Data API'** sinin nasıl kullanılacağı ve hatta **URL** bazlı sorguların nasıl gönderileceği anlatılmakta. Eee, elimizde bu servisin verisini kullanabilmek için **ASP.NET Web API** ile de gelen**HttpClient** tipi var. Daha ne bekliyoruz öyleyse Gelin bir kaç dünya bankası verisini **.Net**kodlarımız ile sorgulayalım.
Önce **örnek bir kaç sorguyu tarayıcı uygulama üzerinden göndermeye çalışalım. Örneğin**ilk 50 ülkenin bilgilerini çekelim. Bunun için aşağıdaki sorguyu kullanmamız yeterli olacaktır.
http://api.worldbank.org/countries
işte sonuç,

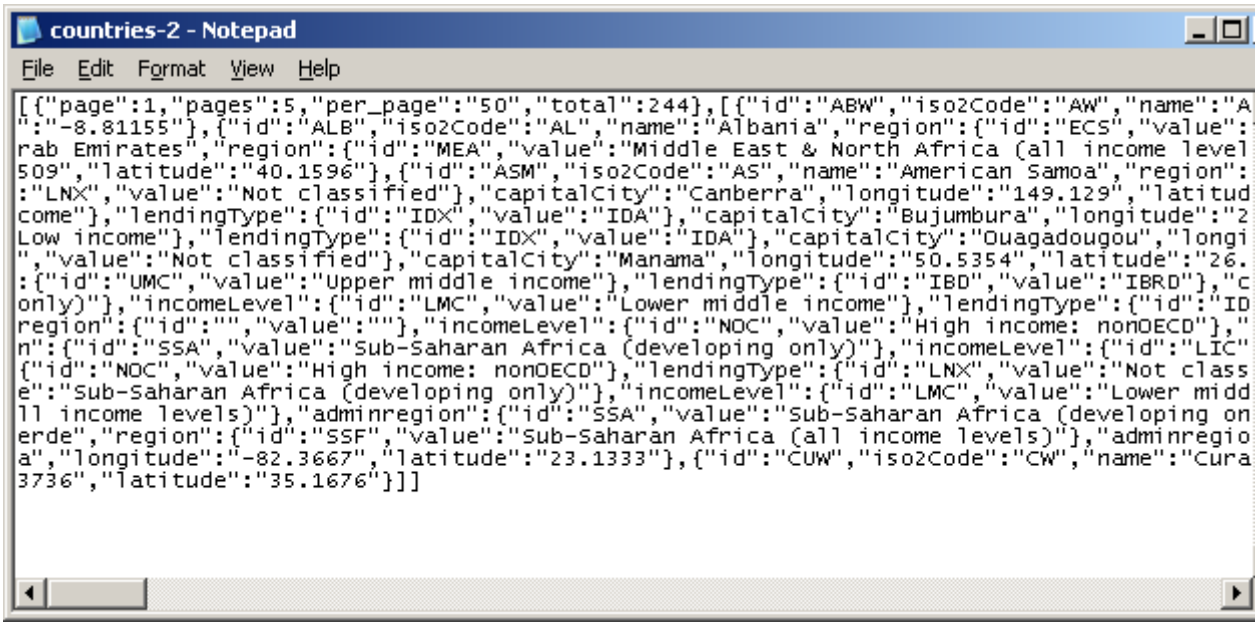
```
- <wb:countries page="1" pages="5" per_page="50" total="244">
  - <wb:country id="ABW">
      <wb:iso2Code>AW</wb:iso2Code>
      <wb:name>Aruba</wb:name>
      <wb:region id="LCN">Latin America & Caribbean (all income levels)</wb:region>
      <wb:adminregion id=""/>
      <wb:incomeLevel id="NOC">High income: nonOECD</wb:incomeLevel>
      <wb:lendingType id="LNX">Not classified</wb:lendingType>
      <wb:capitalCity>Oranjestad</wb:capitalCity>
      <wb:longitude>-70.0167</wb:longitude>
      <wb:latitude>12.5167</wb:latitude>
    </wb:country>
  - <wb:country id="AFG">
      <wb:iso2Code>AF</wb:iso2Code>
      <wb:name>Afghanistan</wb:name>
      <wb:region id="SAS">South Asia</wb:region>
      <wb:adminregion id="SAS">South Asia</wb:adminregion>
      <wb:incomeLevel id="LIC">Low income</wb:incomeLevel>
      <wb:lendingType id="IDX">IDA</wb:lendingType>
      <wb:capitalCity>Kabul</wb:capitalCity>
      <wb:longitude>69.1761</wb:longitude>
      <wb:latitude>34.5228</wb:latitude>
    </wb:country>
  - <wb:country id="AGO">
      <wb:iso2Code>AO</wb:iso2Code>
      <wb:name>Angola</wb:name>
      <wb:region id="SSF">Sub-Saharan Africa (all income levels)</wb:region>
      <wb:adminregion id="SSA">Sub-Saharan Africa (developing only)</wb:adminregion>
      <wb:incomeLevel id="LMC">Lower middle income</wb:incomeLevel>
      <wb:lendingType id="IDX">IDA</wb:lendingType>
      <wb:capitalCity>Luanda</wb:capitalCity>
      <wb:longitude>13.242</wb:longitude>
      <wb:latitude>-8.81155</wb:latitude>
```

Görüldüğü üzere **XML** tabanlı olarak ilk 50 ülkenin bilgilerine ulaşmış durumdayız. **page**, **per_page**gibi nitelikleri de sorgulara parametre olarak katarak sayfalar arasında gezinebiliriz de *(Örneğin 2nci sayfaya gitmeyi ve her sayfada 25 ülke göstermeyi bir deneyin)*

Tabi istersek bu sorguyu **JSON(JavaScript Object Notation)** formatında da ele alabiliriz.

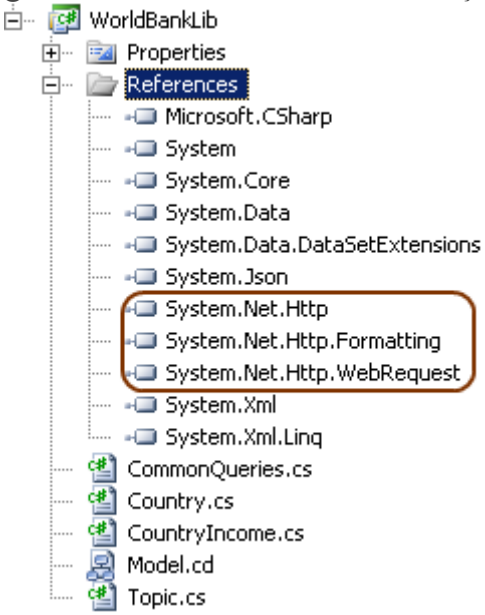
http://api.worldbank.org/countries?format=json

Bu durumda daha küçük boyutlu bir içeriğe ulaşmış oluruz.

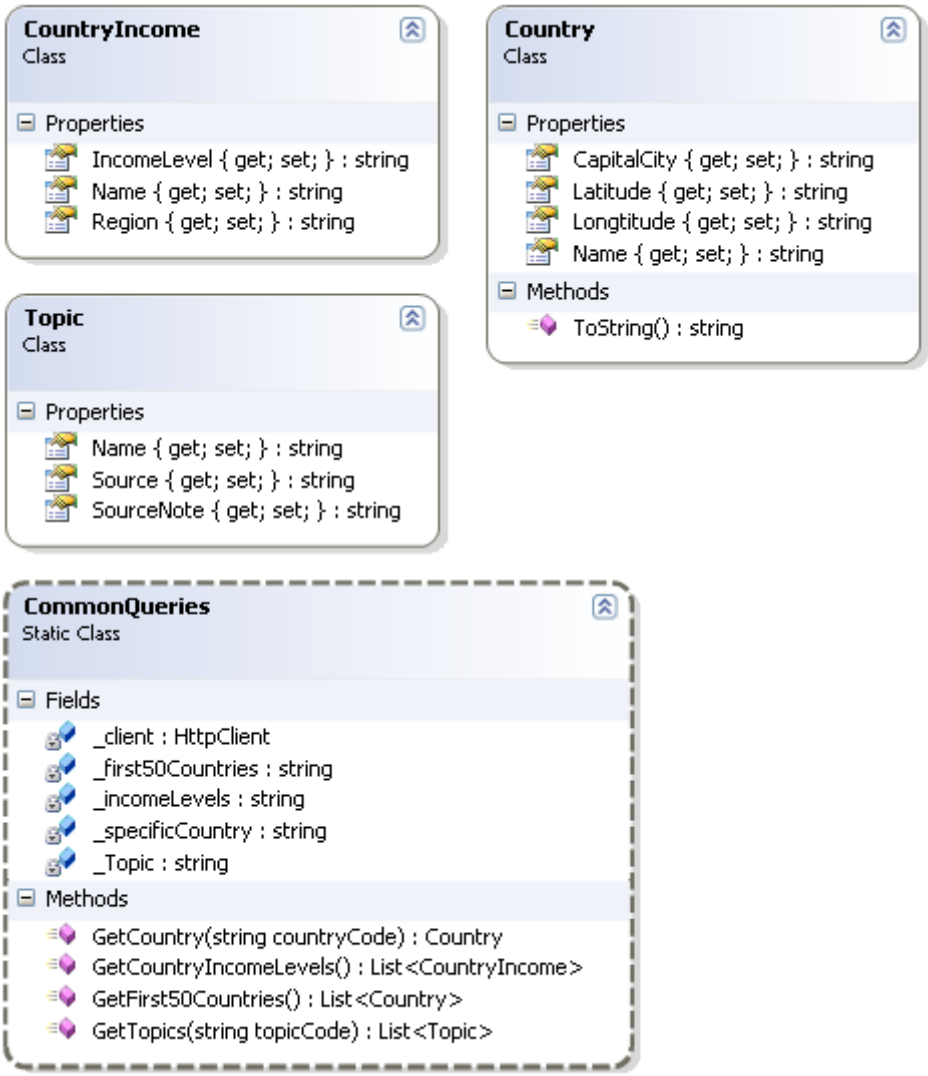


Teori gördüğünüz gibi oldukça basit. **HTTP Get** metoduna göre gönderdiğimiz sorgular sonucunda**XML** veya **JSON** formatında içeriklere ulaşabiliyoruz. Peki bunu kod tarafında nasıl kullanabiliriz?

Bu amaçla örnek bir **Solution** üzerinden hareket edeceğiz. **Solution** içerisinde temel sorgu fonksiyonellikleri ile **POCO(Plain Old CLR Object)** tiplerini barındıran bir kütüphane ile bunu kullanan bir **Windows Forms** uygulaması olması yeterli olacaktır. Sınıf kütüphanesine **NuGet** aracı ile veya dışarıdan harici olarak ilgili referansları da eklememiz gerekmektedir. Bu referanslar aşağıdaki şekilde görüldüğü gibidir.



WorldbankLib isimli sınıf kütüphanemizin içeriğini aşağıdaki **sınıf diagramı(Class Diagram)** ve kod parçalarında olduğu gibi geliştirebiliriz.



Country sınıf;

```
namespace WorldBankLib
{
   /// <summary>
   /// Ülke bilgilerini taşır
   /// </summary>
   public class Country
   {
      /// <summary>
      /// ülkenin adı
      /// </summary>
      public string Name { get; set; }
      /// <summary>
      /// ülkenin başkenti
      /// </summary>
      public string CapitalCity { get; set; }
      /// <summary>
```

```
        /// Enlem
        /// </summary>
        public string Latitude { get; set; }
        /// <summary>
        /// Boylam
        /// </summary>
        public string Longtitude { get; set; }
        public override string ToString()
        {
            return string.Format("{0}\t\t{1}\t\t({2};{3})"
                , Name
                , CapitalCity
                , Latitude
                , Longtitude
                );
        }
    }
}
```

CountryIncome sınıfı;

```
namespace WorldBankLib
{
    /// <summary>
    /// Gelir düzeyi bilgilerini içerir
    /// </summary>
    public class CountryIncome
    {
        /// <summary>
        /// Ülkenin adı
        /// </summary>
        public string Name { get; set; }
        /// <summary>
        /// Gelir düzeyi seviyesi
        /// </summary>
        public string IncomeLevel { get; set; }
        /// <summary>
        /// Bağlı bulunulan bölge
        /// </summary>
        public string Region { get; set; }
    }
}
```

Topic sınıfı;

```
namespace WorldBankLib
{
   /// <summary>
   /// Topic bilgilerini verir
   /// </summary>
   public class Topic
   {
      /// <summary>
      /// Topiğin adı
      /// </summary>
      public string Name { get; set; }
      /// <summary>
      /// Kaynak bilgisi(Education gibi)
      /// </summary>
      public string Source { get; set; }
      /// <summary>
      /// Kaynağa ait bir açıklama bilgisi
      /// </summary>
      public string SourceNote { get; set; }
   }
}
```

ve asıl fonksiyonellikleri üstlenen **CommonQueries** sınıfı;

```
using System.Collections.Generic;
using System.Json;
using System.Net.Http;
namespace WorldBankLib
{
   /// <summary>
   /// Worldbank OData servisi için genel sorguları içerir
   /// </summary>
   public static class CommonQueries
   {
      #region Temel sorgularımız
      // İlk 50 ülkenin bilgilerini JSON formatında verir
      private static string _first50Countries =
"http://api.worldbank.org/countries?format=json";
      //{0} yerine gelen ülkenin bilgisini JSON formatında verir. Bu bilgi BR gibi ülke kodu
şeklindedir
      private static string _specificCountry =
"http://api.worldbank.org/countries/{0}?format=json";
      //ilk 50 ülkenin gelir düzeyi durumları JSON formatında verilir
```

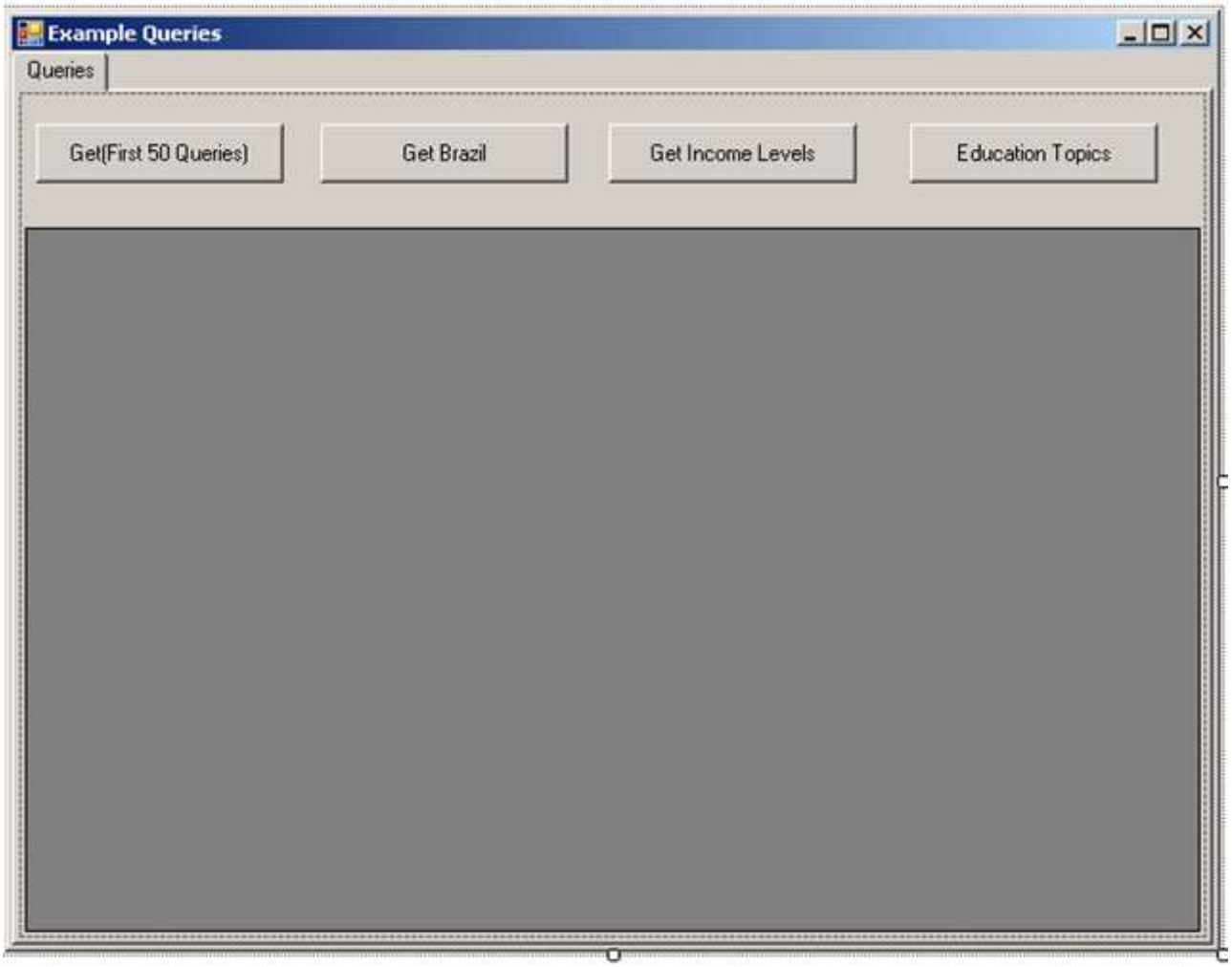
```
        private static string _incomeLevels =
"http://api.worldbank.org/countries?incomeLevels&format=json";
        // {0} yerine(Örneğin Education için 4) gelen konuya ait olacak şekilde bazı topic
bilgileri json formatında çekilir.
        private static string _Topic =
"http://api.worldbank.org/topics/{0}/indicators?format=json";
        #endregion
        /// <summary>
        /// Http Get taleplerini göndermemizde devreye giren yardımcı tipimiz
        /// </summary>
        private static HttpClient _client = new HttpClient();
        /// <summary>
        /// İlk 50 ülkenin ad, başkent, enlem ve boylam bilgilerini bulur
        /// </summary>
        /// <returns>ülke listesi döner</returns>
        public static List<Country> GetFirst50Countries()
        {
            List<Country> countries = new List<Country>();
            _client.GetAsync(_first50Countries)
                .ContinueWith((r) =>
                {
                    HttpResponseMessage responseMessage = r.Result;
                    responseMessage.EnsureSuccessStatusCode();
                    responseMessage.Content.ReadAsAsync<JsonArray>().
                        ContinueWith((rt) =>
                                {
                                    foreach (var country in rt.Result[1])
                                    {
                                        countries.Add(
                                            new Country()
                                                {
                                                    Name = country.Value["name"].ToString(),
                                                    CapitalCity
=country.Value["capitalCity"].ToString(),
                                                    Latitude =country.Value["latitude"].ToString(),
                                                    Longtitude =country.Value["longitude"].ToString()
                                                }
                                            );
                                    }
                                });
                }).Wait();
```

```
    return countries;
}
/// <summary>
/// Belirli bir ülkeye ait isim, başkent, enlem ve boylam bilgilerini verir
/// </summary>
/// <param name="countryCode">Ülke kodu(brazilya için BR gibi)</param>
/// <returns>ülke bilgisi</returns>
public static Country GetCountry(string countryCode)
{
    Country resultCountry=null;
    _client.GetAsync(string.Format(_specificCountry,countryCode))
        .ContinueWith((r) =>
        {
            HttpResponseMessage responseMessage = r.Result;
            responseMessage.EnsureSuccessStatusCode();
            responseMessage.Content.ReadAsAsync<JsonArray>().
                ContinueWith((rt) =>
                            {
                                var cntry = rt.Result[1];
                                resultCountry = new Country()
                                        {
                                            Name = cntry[0]["name"].ToString(),
                                            CapitalCity = cntry[0]["capitalCity"].ToString(),
                                            Latitude = cntry[0]["latitude"].ToString(),
                                            Longtitude = cntry[0]["longitude"].ToString()
                                        };
                            });
        }).Wait();
    return resultCountry;
}
/// <summary>
/// İlk 50 ülkenin Gelir düzeyi bilgilerini döndürür
/// </summary>
/// <returns>Gelir düzeyi bilgileri</returns>
public static List<CountryIncome> GetCountryIncomeLevels()
{
    List<CountryIncome> countries = new List<CountryIncome>();
    _client.GetAsync(_incomeLevels)
        .ContinueWith((r) =>
        {
```

```
            HttpResponseMessage responseMessage = r.Result;
            responseMessage.EnsureSuccessStatusCode();
            responseMessage.Content.ReadAsAsync<JsonArray>().
               ContinueWith((rt) =>
               {
                  foreach (var country in rt.Result[1])
                  {
                     countries.Add(
                        new CountryIncome()
                        {
                           Name = country.Value["name"].ToString(),
                           IncomeLevel =
country.Value["incomeLevel"]["value"].ToString(),
                           Region = country.Value["region"]["value"].ToString()
                        }
                        );
                  }
               });
         }).Wait();
      return countries;
   }
   /// <summary>
   /// Topic(Education) bazlı rapor bilgilerini döndürür
   /// </summary>
   /// <param name="topicCode">Topic kodu(Education=4 gibi)</param>
   /// <returns>Topic içeriği</returns>
   public static List<Topic> GetTopics(string topicCode)
   {
      List<Topic> topics = new List<Topic>();
      _client.GetAsync(string.Format(_Topic,topicCode))
         .ContinueWith((r) =>
         {
            HttpResponseMessage responseMessage = r.Result;
            responseMessage.EnsureSuccessStatusCode();
            responseMessage.Content.ReadAsAsync<JsonArray>().
               ContinueWith((rt) =>
               {
                  foreach (var topic in rt.Result[1])
                  {
                     topics.Add(
                        new Topic()
```

```
                        {
                            Name = topic.Value["name"].ToString(),
                            Source = topic.Value["source"]["value"].ToString(),
                            SourceNote = topic.Value["sourceNote"].ToString()
                        }
                        );
                    }
                });
            }).Wait();
        return topics;
    }
  }
}
```

CommonQueries sınıfı içerisinde **HttpClient** tipinin kullanıldığı çeşitli metodlar söz konusudur. Her metod **Worldbank OData** servisine doğru bir sorgu göndermekte, gelen içeriği **JSON**formatında olacak şekilde ele almaktadır.
Çok doğal olarak veri çekme işlemi uzun sürebilir. Bu nedenle **HttpClient** tipinin **GetAsync**metodundan yararlanılmaktadır. Bu metoda olan çağrı asenkron olarak verinin çekilmesi noktasında devreye girmektedir. Yanlız dikkat edilmesi gereken hususlardan birisi, bu metodları kullanan uygulamaların söz konusu asenkron çalışmaların sonucunu beklemeden sonlanmasını engellemeye çalışmaktır. Bir **Windows Forms** uygulaması söz konusu olacağından, **client'** ı uyarmak adına **Wait**metodundan yararlanılmış ve buradaki metodlardan ancak sonuçlar alındığında çıkılması garanti edilmiştir. Tabi **Windows Forms** tarafında bu duraksatmaların ekranı dondurması da engellenmeli ve kullanıcının eş zamanlı başka işlemler yapabilmesine de izin verilmelidir. Bunun için **Forms**tarafında **BackgroundWorker** kontrolünden yararlanıyor olacağız. Örnek **Windows Forms**tasarımımızı ve kod içeriğini aşağıdaki gibi geliştirebiliriz.



ve kod içeriğimiz

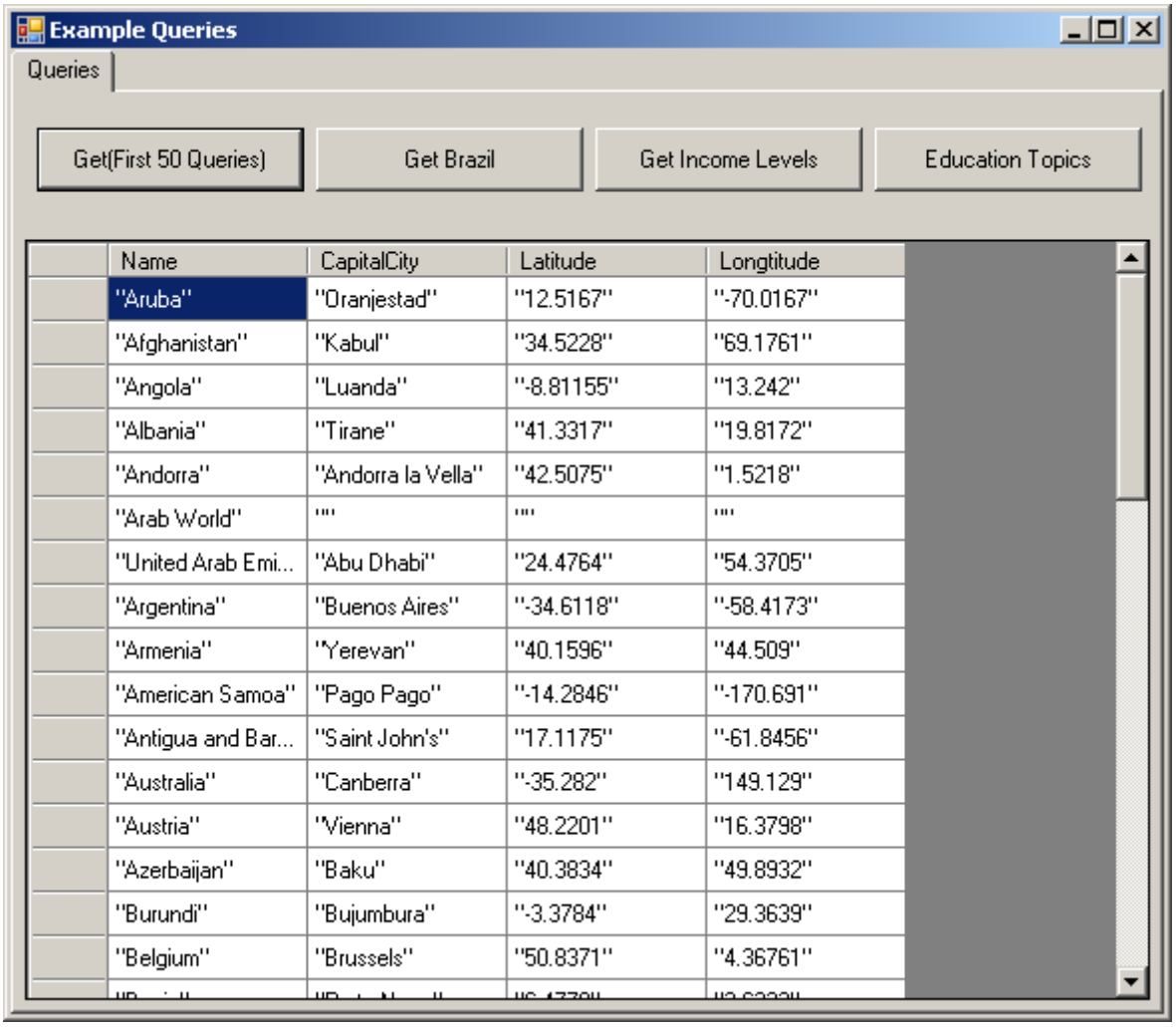
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Windows.Forms;
using WorldBankLib;
namespace WinClientApp
{
   public partial class Form1 : Form
   {
      private List<Country> countries = null;
      private Country brazil = null;
      public Form1()
      {
```

```
        InitializeComponent();
    }
    private void btnGetFirst50Countries_Click(object sender, EventArgs e)
    {
        bgwGetFirst50Countries.RunWorkerAsync();
    }
    private void bgwGetFirst50Countries_DoWork(object sender, DoWorkEventArgs e)
    {
        e.Result = CommonQueries.GetFirst50Countries();
    }
    private void bgwGetFirst50Countries_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
    {
        if(!e.Cancelled || e.Error==null)
            dgvResult.DataSource = (List<Country>)e.Result;
    }
    private void bgwGetBrazil_DoWork(object sender, DoWorkEventArgs e)
    {
        e.Result = CommonQueries.GetCountry("br");
    }
    private void bgwGetBrazil_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
    {
        if (!e.Cancelled || e.Error == null)
        {
            var country=e.Result as Country;
            if(country!=null)
                MessageBox.Show(country.ToString());
        }
    }
    private void btnGetBrazil_Click(object sender, EventArgs e)
    {
        bgwGetBrazil.RunWorkerAsync();
    }
    private void btnGetIncomeLevels_Click(object sender, EventArgs e)
    {
        bgwGetIncomeLevels.RunWorkerAsync();
    }
    private void bgwGetIncomeLevels_DoWork(object sender, DoWorkEventArgs e)
    {
```

```
            e.Result = CommonQueries.GetCountryIncomeLevels();
        }
        private void bgwGetIncomeLevels_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
        {
            if (!e.Cancelled || e.Error == null)
            {
                dgvResult.DataSource=(List<CountryIncome>)e.Result;
            }
        }
        private void btnGetEducationTopics_Click(object sender, EventArgs e)
        {
            bgwGetEducationTopics.RunWorkerAsync();
        }
        private void bgwGetEducationTopics_DoWork(object sender, DoWorkEventArgs e)
        {
            e.Result = CommonQueries.GetTopics("4");
        }
        private void bgwGetEducationTopics_RunWorkerCompleted(object sender,
RunWorkerCompletedEventArgs e)
        {
            if (!e.Cancelled || e.Error == null)
            {
                dgvResult.DataSource = (List<Topic>)e.Result;
            }
        }
    }
}
```

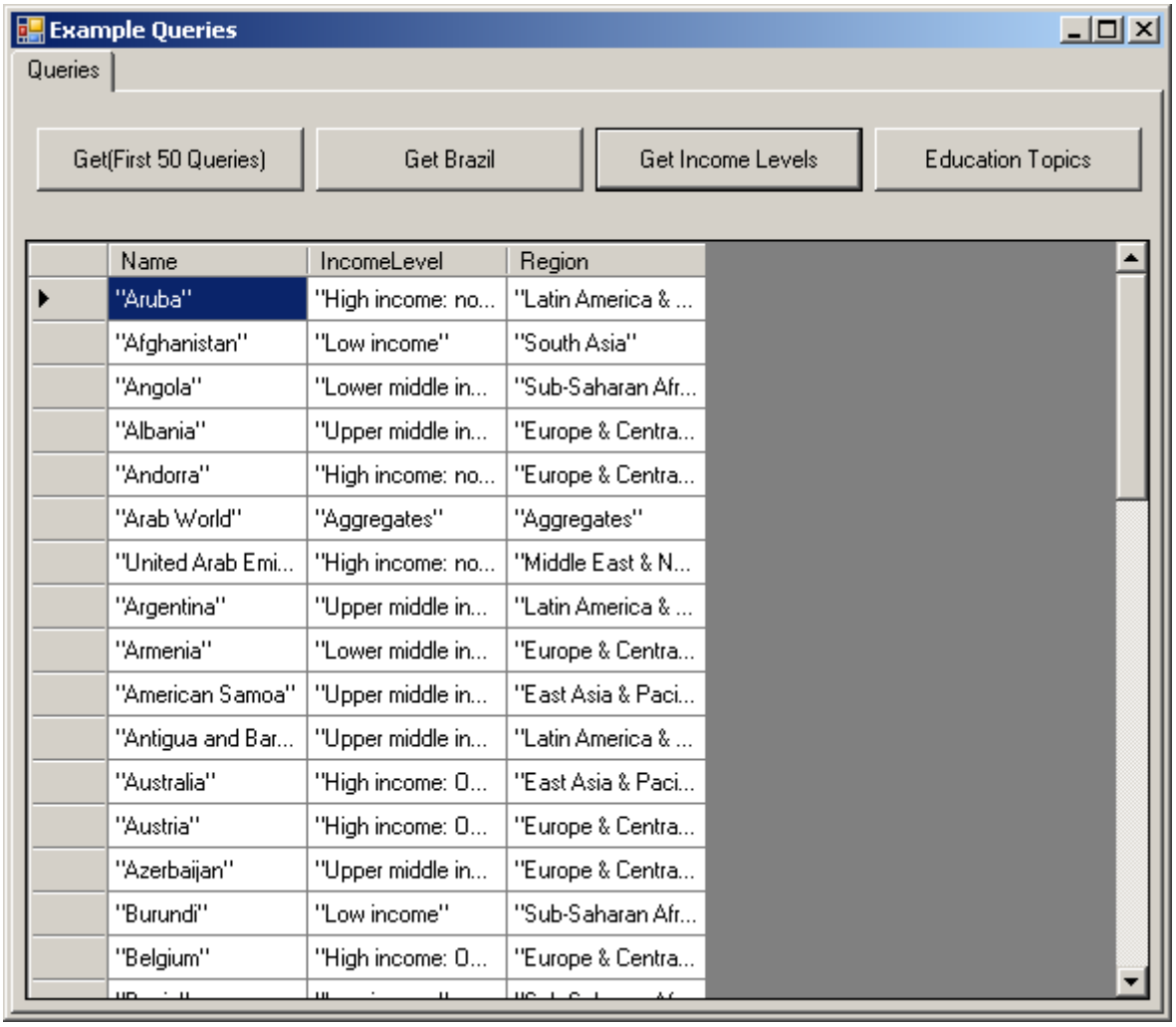
Dikkat edileceği üzere test düğmelerine basıldıkça ilgli **BackgroundWorker** nesne örneği çalıştırılmaktadır. Bu **BackgroundWorker** tiplerine ait **DoWork** ve **RunWorkerCompleted** olay metodlarında ise, çalıştırma ve işlem sonuçlarının ele alınması işlemleri sağlanmaktadır. Örneğimizi test ettiğimizde aşağıdakilere benzer sonuçlar elde ederiz.

İlk 50 ülke bilgisinin çekilmesi;

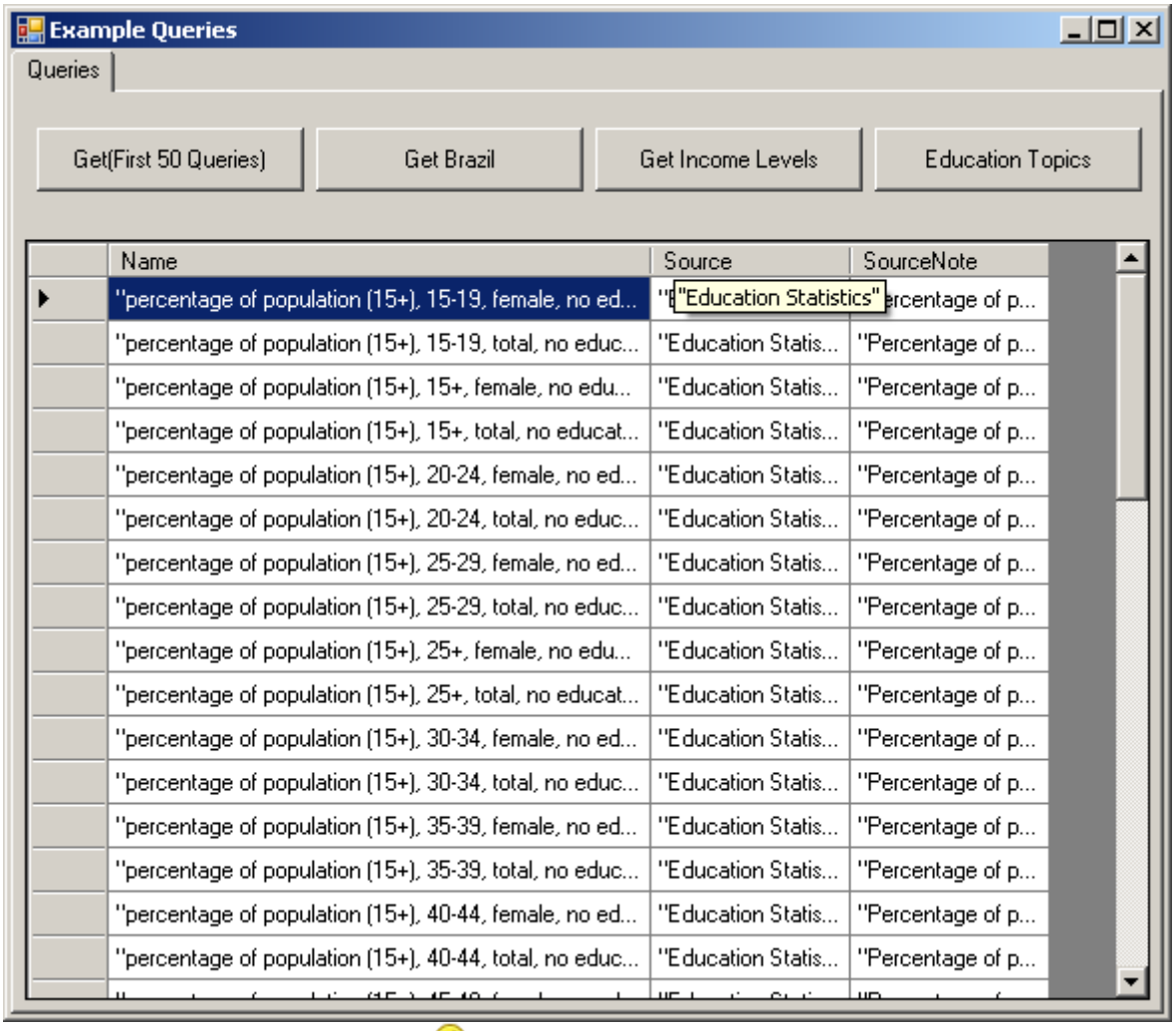


Brezilyaya ait bilginin çekilmesi sonucu;

Gelir düzeyi bilgilerinin elde edilmesi;



Eğitim odaklı verilerin elde edilmesi;



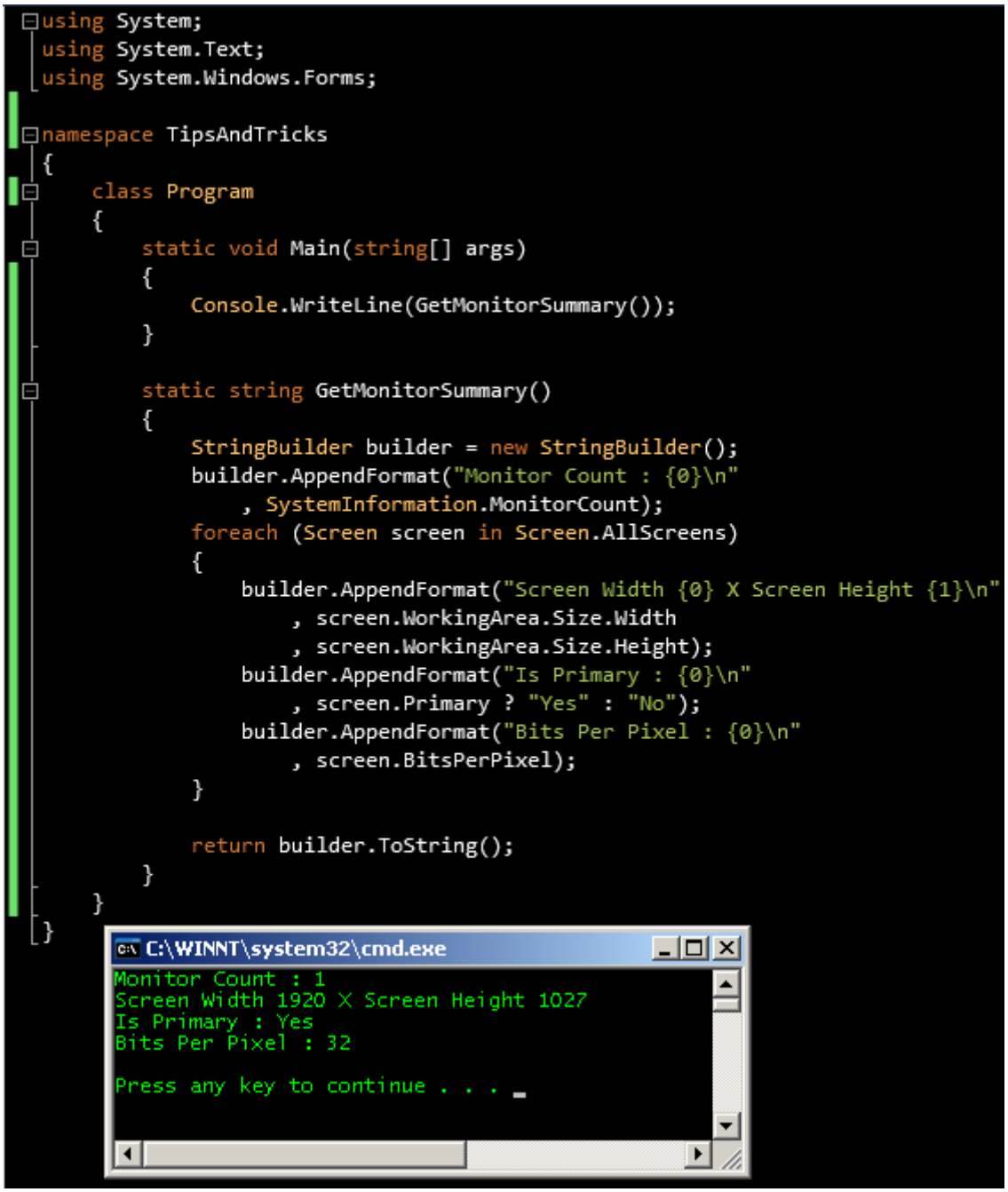
Gördüğünüz gibi gayet kolay Tabi örneğimizin pek çok noktasında hata bulunmaktadır. *(Biraz acele ile yazdığımdan dolayı)* Örneğin **Exception Handling** mekanizması eksiktir ve hatta**DataGridView** kontrolü bazı noktalarda **index hatası** vermektedir. Diğer yandan daha fazla sorgu da işin içerisine katılabilir. Ben kapıyı gösteren kişi olarak yazımı burada sonlandırıyorum Artık**Worldbank** verilerinin **ASP.NET Web API(ve tabi WCF Web API)** ile gelen **HttpClient** tipi yardımıyla ve özellikle **JSON** formatında nasıl ele alabileceğinizi öğrendiniz. Bunun geliştirmek tamamen sizlerin elinde. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.
WorldBank.zip (88,93 kb)

Tek Fotoluk İpucu 52 - Monitor Bilgileri

Çarşamba, 9 Mayıs 2012 00:10

C#, Windows Forms, Monitor, Screen, System Informations

Merhaba Arkadaşlar,
Malum Visual Studio dahil pek çok programın birden fazla monitor desteği bulunmakta. Hatta geliştirdiğimiz uygulamaların çoğu birden fazla monitor desteği olacak şekilde yapılandırılabiliyor. Peki sistemde yüklü kaç monitor var, bunların çözünürlükleri, pixel başına byte değerleri nelerdir, öğrenebilir miyiz? Tabi ki

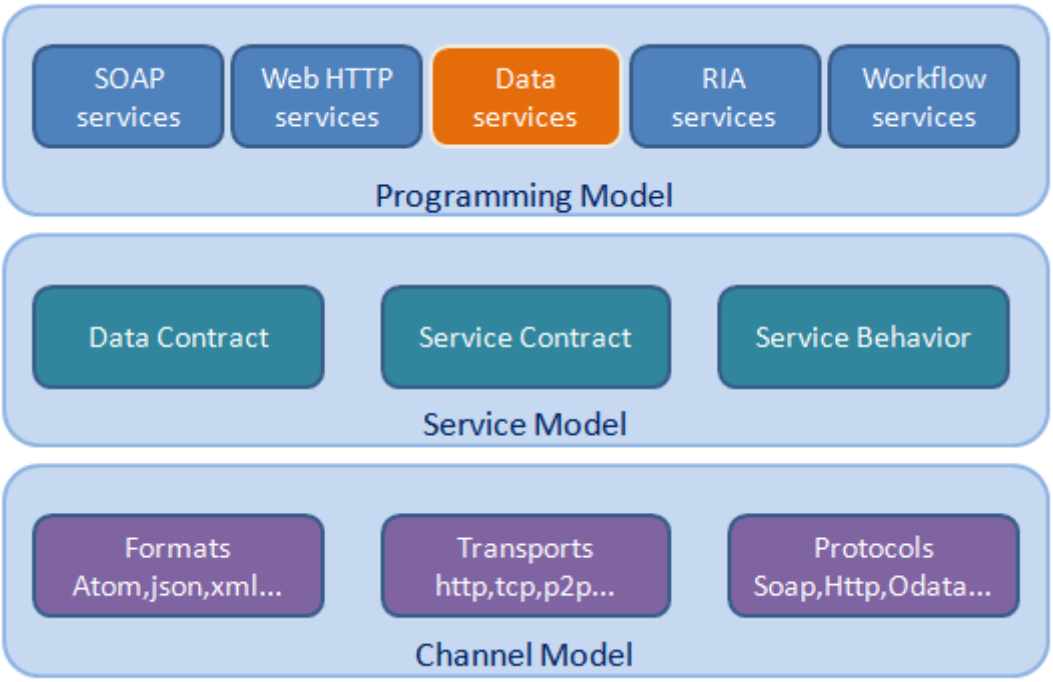


WCF Data Services – Reflection Provider Kullanımı

Salı, 1 Mayıs 2012 10:25

Wcf, Wcf Data Services, Reflection Provider

Merhaba Arkadaşlar,
Bundan bir kaç yıl öncesiydi. Daha dün gibi hatırlıyorum. **.Net Framework 3.0** sürümünde çıkması beklenen yeni **Foundation** alt yapıları üzerine **Microsoft** ekibinden gelen elektronik bir postayı okuyordum. O zamana kadar **XML Web Service**' leri ve özellikle **.Net Remoting** ile yakın ilişkiler içerisinde bulunduğumdan, dikkatimi ilk çeken **Windows Communication Foundation**isimli konsept olmuştu. Kısaca **WCF** olarak adlandırılıyordu. Mutlaka üzerine eğilmem gerektiğini düşündüğüm bir konuydu. Benim için yeni bir macera başlıyordu.
Hem heyecanlıydım hem de biraz korkmuştum 😉 Çünkü var olan tüm servis yaklaşımlarını tek bir çatı altında toplayacak bir modelden bahsediliyordu*(Onun şimdiki halini ve yaygınlığını düşündüğümüzde, Microsoft' un bu vizyonunda gerçekten de haklı olduğunu bir kez daha görebiliyorum)* Hemen araştırmalara başlamalıydım. Internet Explorer' ımı açtım ve **WCF** kelimesini googleladım. Bir de ne göreyim. Karşıma **Dünya Bisiklet Federasyonu(World Cycling Federations)** ile ilişkili sonuçlar çıktı. Yandaki fotoğrafın anlamı bundandır 😛 Geyiği bir kenara bırakıp konumuza dönelim.
Windows Communication Foundation(WCF) yıllardır **Microsoft** mimarisinde önemli bir yere sahip. Özellikle servis bazlı uygulama geliştirme modeline kazandırdığı pek çok yeni yaklaşım sayesinde, var olan ve gelecek **Microsoft** ürünlerinin de pek çok noktasında kullanılmaya başlandı. Bana göre **WCF** modeli özellikle **3.5** sürümü ile başlayan gelişmeler ile birlikte programlama modelini önemli derece de geliştirdi. Bu gün mimari programlama modeline kuşbakışı baktığımızda aşağıdaki çizelge de görülen servis çeşitlerinin kullanımda olduğunu biliyoruz.



Standart **SOAP(Simple Object Access Protocol)** servisleri*(klasik XML Web Service yaklaşımı olarak da düşünebiliriz)*, bügünlerde ASP.NET takımınca ele alınan ve **Asp.Net Web API** olarak adlandırılan **HTTP** bazlı **REST(Representational State Transfer)** modeli, özellikle **veri odaklı(Data-Centric)** çalışmak üzere tasarlanmış **Entity Framework** odaklı **Data Service'** ler,**Silverlight** tarafında kullanılan **RIA(Rich Internet Application)** çeşidi ve tabiki **Workflow Foundation** içerisinde yerini almış **Workflow Service'** ler.

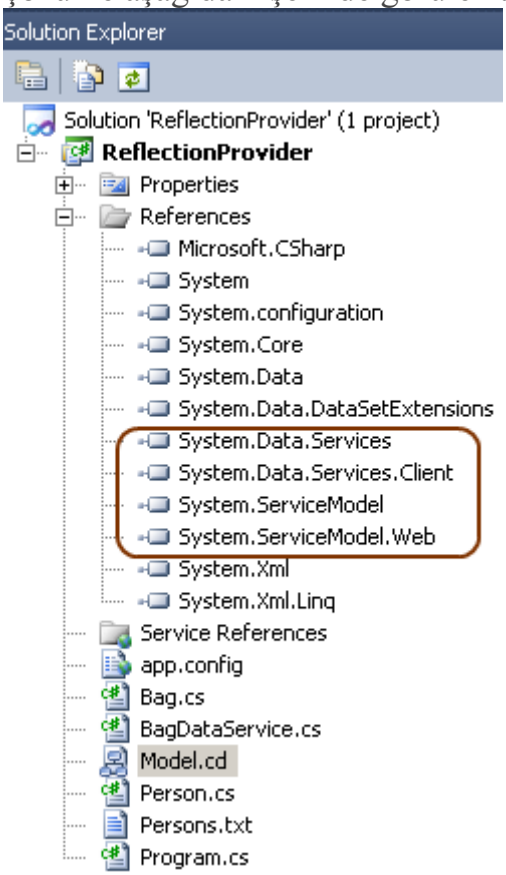
Bu geniş servis yelpazesinin uygulandığı alanlar göz önüne alındığında, **SQL Server, Sharepoint, BizTalk** gibi pek çok ürün grubu işin içerisine giriyor ki buna bir de Cloud üzerindeki WCF uç noktalarını etkilediğimizde modelin ne kadar etkili bir alana yayıldığını daha iyi görebiliyoruz. Dolayısıyla **WCF,** etkisini pek çok noktadan pozitif anlamda hissettiriyor. Peki biz bu yazımızda neyi inceliyor olacağız? 🙂

Çoğunluka **Entity Framework** odaklı olarak kullanılan **Data Service'** ler, basit **HTTP** protokolü üzerinden, **Querystring** bazlı olacak şekilde içerik yayınlanmasına izin vermektedir. Bu anlamda**HTTP** protokolünün basit **Get, Post, Put, Delete** metodları ile çalışabilen, **XML, JSON** veya **OData**standardında **Entity** çıktılarını verebilen bir servis yapısı söz konusudur. Burada kafamıza takılan veya çok fazla ilişmediğimiz konulardan birisi ise,

Data Service' leri her zaman Entity Framework tabanlı olarak kullanmak zorunda olup olmadığımızdır?

Güzel soru 😬 Acaba var olan **Data Service** çalışma modeli esnetilip **Entity Framework** yerine kendi **veri sağlayıcılarımız(Provider)** ile çalışabilir miyiz? Bir başka deyişle, örneğin **ASP.NET**üzerindeki **Membership Provider, Profile Provider** gibi yapılarda uygulanabilen özelleştirme mantığını, **Data Service'** ler tarafında da yapabilir miyiz? İşte bu yazımızda bu konuyu ele alıyor olacağız.

Olayı kolay ve hızlı bir şekilde kavrayabilmek adına İşe basit bir **Console** uygulaması açıp, çözüme aşağıdaki şekilde görülen **.Net assembly'** larını referans ederek başlamalıyız.

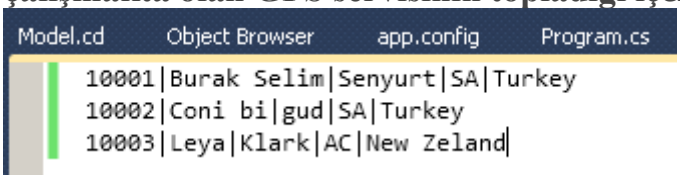


Burada görüldüğü üzere uygulamamızda **DataService** tipini kullanarak bir **Self-Host** işlemi gerçekleştiriyor olacağız. Diğer yandan kendi **provider'** ımızı yazacağımız için de bazı **assembly'**lara ihtiyacımız var

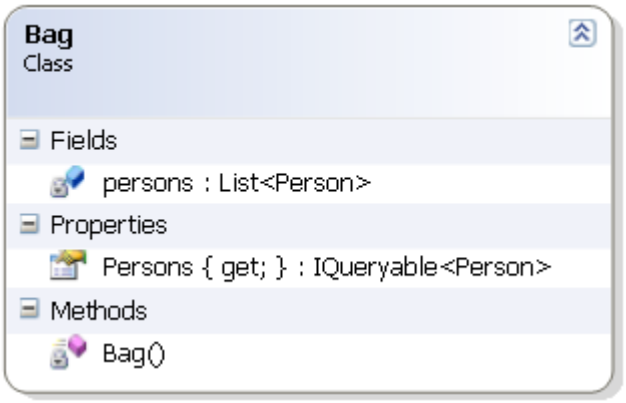
- System.Data.Services
- System.Data.Services.Client
- System.ServiceModel
- System.ServiceModel.Web

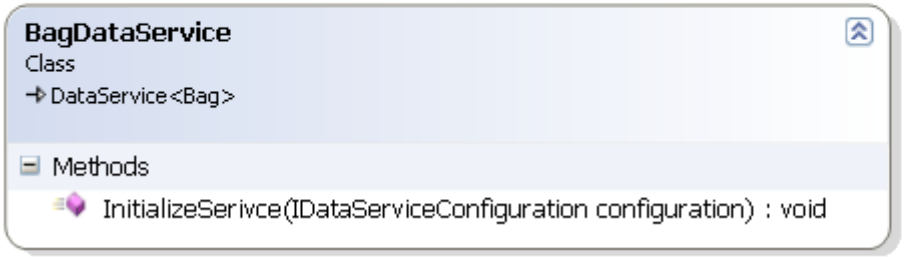
Örnek uygulamamızda **felsefeyi** anlamaya odaklanacağımız için çok basit bir **Text** dosyasını veri kaynağı olarak kullanıyor olacağız. Bu text dosya içerisinde | işaretleri ile ayrılmış şekilde bir personel verisini tutmayı planlıyoruz. Personel numarası, adı, soyadı, ünvanı ve bulunduğu ülke örnek veri alanlarımız olarak düşünülebilir.

Pek tabi çok daha farklı bir veri kaynağını da senaryoya dahil edebilirsiniz. Örneğin donanımsal bir arayüzün verilerini bu şekilde Data Service 'ler üzerinden sunabileceğinizi düşünün. Söz gelimi bu bir mobil cihaz üzerinde bir süredir çalışmakta olan GPS servisinin topladığı içerik olabilir 🙂



Gelelim uygulamamız içerisindeki asıl kodlara. Önce **Class Diagram'** a bir bakalım.





Person sınıfı aslında bizim için **POCO(Plain Old CLR Object)** niteliği gösteren bir tip olarak düşünülebilir. Bu tipi, **text** dosya içerisindeki satırların nesnel karşılıklarını ifade etmek için kullanacağız.

```
using System.Data.Services.Common;
namespace ReflectionProvider
{
   [DataServiceKey("PersonId")]
   public class Person
   {
      public int PersonId { get; set; }
      public string Name { get; set; }
      public string Surname { get; set; }
      public string Title { get; set; }
      public string Country { get; set; }
   }
}
```

Tabi bu tip içerisinde sınıf seviyesinde uygulanan **DataServiceKey** niteliğine dikkat etmemiz gerekiyor. Bu nitelik ile **Person** tiplerinin **Unique'** liğini **PersonId** özelliklerine bağladığımızı ifade etmiş oluyoruz. Nitekim **Data Service** sınıfları, **Entity Framework Entity** tipleri ile çalışırken ilgili sınıflarda bu niteliğin kullanılmasını beklemektedir. Aynı davranışın burada da uyarlanması gerekmektedir. Gelelim **Bag** sınıfına.

```
using System;
using System.Collections.Generic;
```

```
using System.Configuration;
using System.IO;
using System.Linq;
namespace ReflectionProvider
{
   public class Bag
   {
      private static List<Person> persons = new List<Person>();
       static Bag()
       {
          string[]
personLines=File.ReadAllLines(ConfigurationManager.AppSettings["FilePath"]);
          foreach (string personLine in personLines)
          {
             string[] columns=personLine.Split('|');
             Person person = new Person();
             person.PersonId = Int32.Parse(columns[0]);
             person.Name = columns[1];
             person.Surname = columns[2];
             person.Title = columns[3];
             person.Country = columns[4];
             persons.Add(person);
          }
       }
       public IQueryable<Person> Persons {
         get
          {
             return persons.AsQueryable();
          }
       }
   }
}
```

Bag sınıfı içerisinde **text** tabanlı dosya içeriğinin okunması ve **Person** tipinden **generic** bir **List**koleksiyonuna atanması işlemleri yapılmaktadır. Bu işlemler **static yapıcı metod(Constructor)**içerisinde gerçekleştirilmektedir. Diğer yandan sınıfın olabilecek belki de en önemli özelliği **Persons**üyesidir. Dikkat edileceği üzere bu üye geriye **IQueryable<Person>** tipinden bir referans döndürmektedir. Bu sayede biz **Data Service host** ortamına, **URL** bazlı olarak sorgulanabilir bir içerik sağlayacağımızı da belirtmiş olacağız*(Bir başka deyişle **EntitySet** tipini ifade etmiş olmaktayız)* Tabi bu belirtme işlemini asıl **DataService** sınıfı içerisinde gerçekleştireceğiz. Aynen aşağıda olduğu gibi

```
using System.Data.Services;
namespace ReflectionProvider
{
   public class BagDataService
      :DataService<Bag>
   {
      public static void InitializeService(IDataServiceConfiguration configuration)
      {
         configuration.SetEntitySetAccessRule("Persons", EntitySetRights.All);
      }
   }
}
```

InitializeService metodu, servisi host edeceğimiz ortamda tetikleniyor olacak. Bu metod içerisinde standart **Data Service** şablonunda da yapıldığı gibi gerekli erişim kuralları belirlenmektedir. Tabi bu sefer **Bag** tipi içerisinde yer alan **Persons** isimli özellik için **full** erişim açılması söz konusudur. Bir başka deyişle **GET, POST, PUT ve DELETE** metodlarına hizmet edecektir. *(Ancak **POST, PUT, DELETE** gibi metod etkileşimleri için Bag tipine **IUpdatable** arayüzünün de uygulanması gerekmektedir ki bu yazımızda bu konu ele alınmamıştır. Bir sonraki yazımızda kısmetse)*

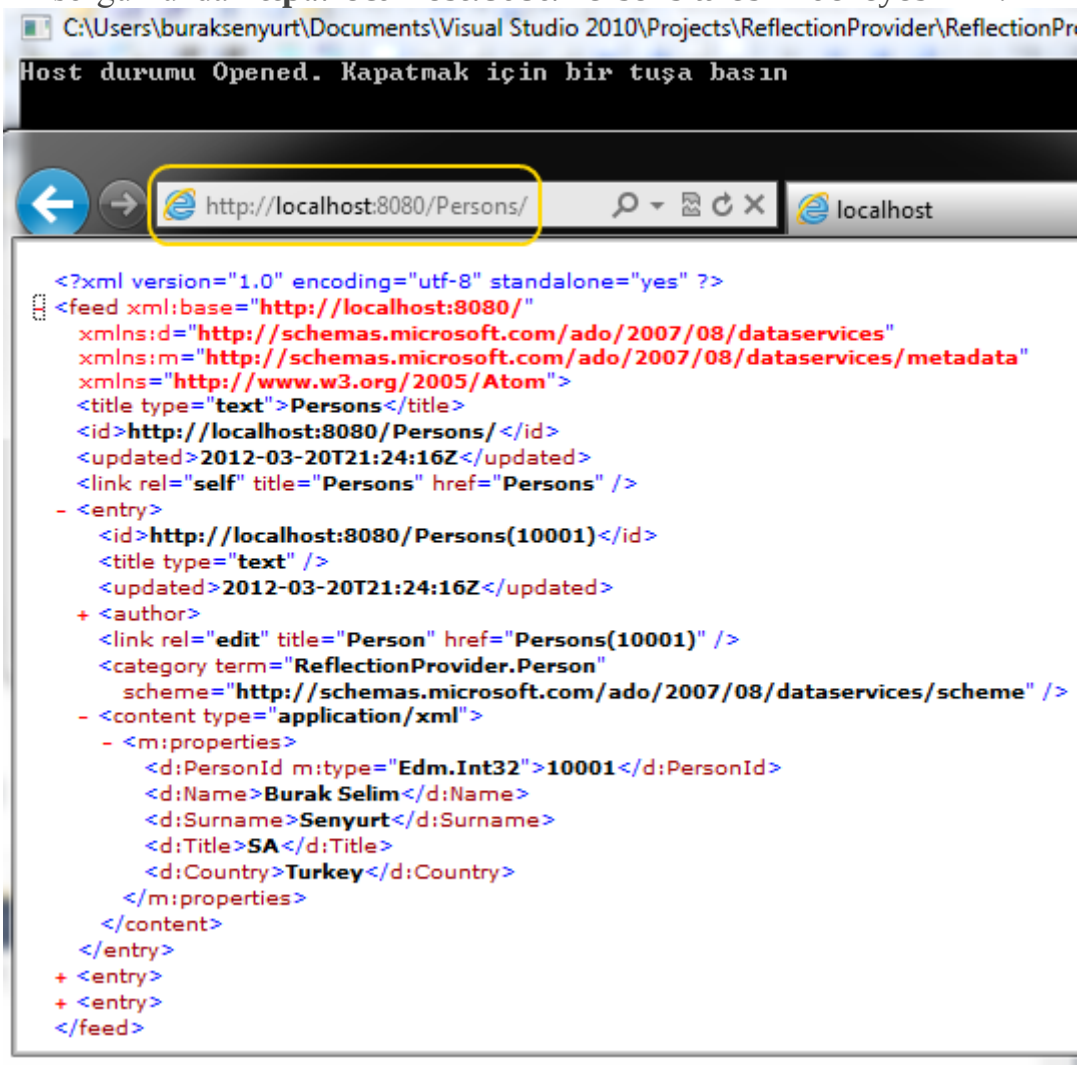
Artık tek yapılması gereken **DataService** tipinden yararlanarak bir **Host** örneğini oluşturmak ve belirli bir **URL** üzerinden istemci taleplerini dinleyecek şekilde ayağa kaldırmaktan ibarettir. Bu amaçla **Console** uygulamamıza ait Main metodu içeriğini aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.Data.Services;
namespace ReflectionProvider
{
   class Program
   {
      static void Main(string[] args)
      {
         Type serviceType = typeof(BagDataService);
         Uri baseAddress = new Uri("http://localhost:8080");
        Uri[] baseAddresses = new Uri[] { baseAddress };
         DataServiceHost host = new DataServiceHost(
            serviceType,
           baseAddresses);
         host.Open();
         Console.WriteLine("Host durumu {0}. Kapatmak için bir tuşa basın",host.State);
         Console.ReadLine();
         host.Close();
```

```
            Console.WriteLine("Host duruumu {0}. Güle güle!",host.State);
        }
    }
}
```

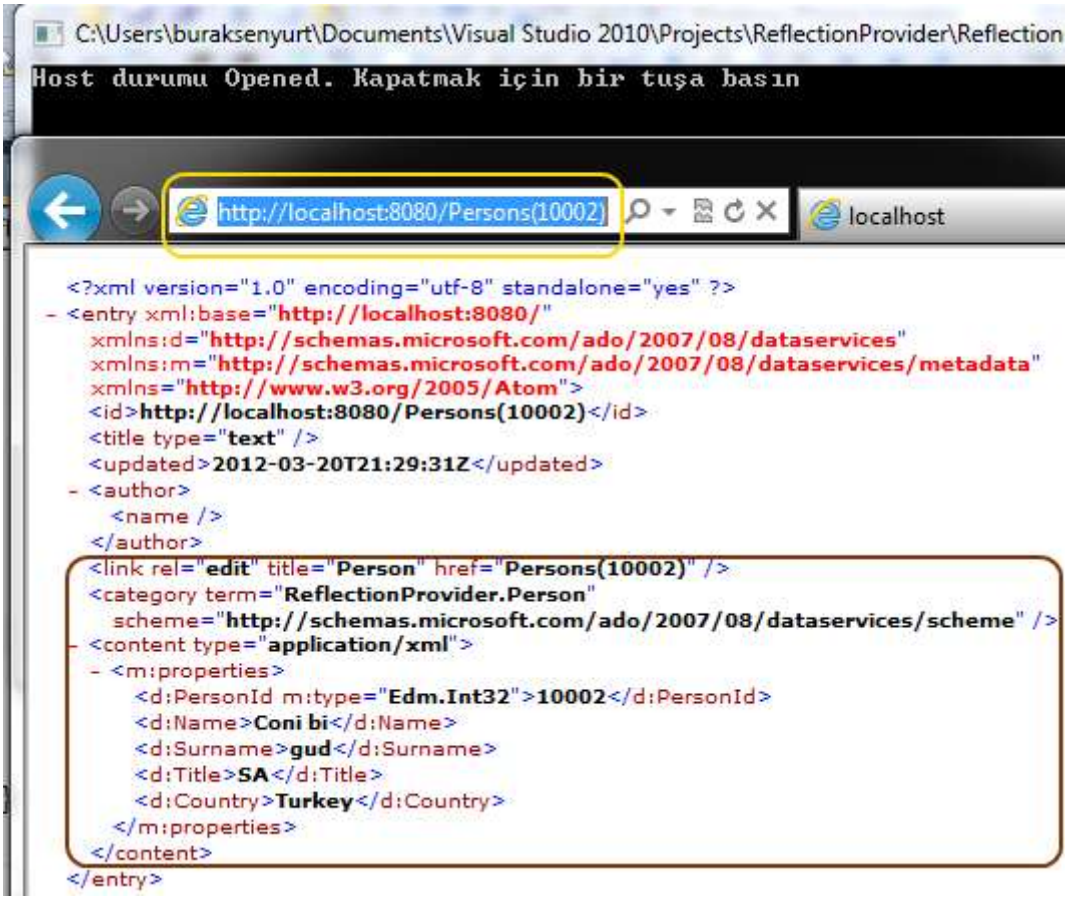
Şimdi buraya kadar yaptıklarımızı bir test edelim. İlk olarak **Console** uygulamamızı çalıştıralım. Sonrasında ise herhanbiri tarayıcı üzerinden uygulamamıza sorgulama talepleri gönderelim.

İlk sorgumuzda **http://localhost:8080/Persons** talebini deneyebiliriz.

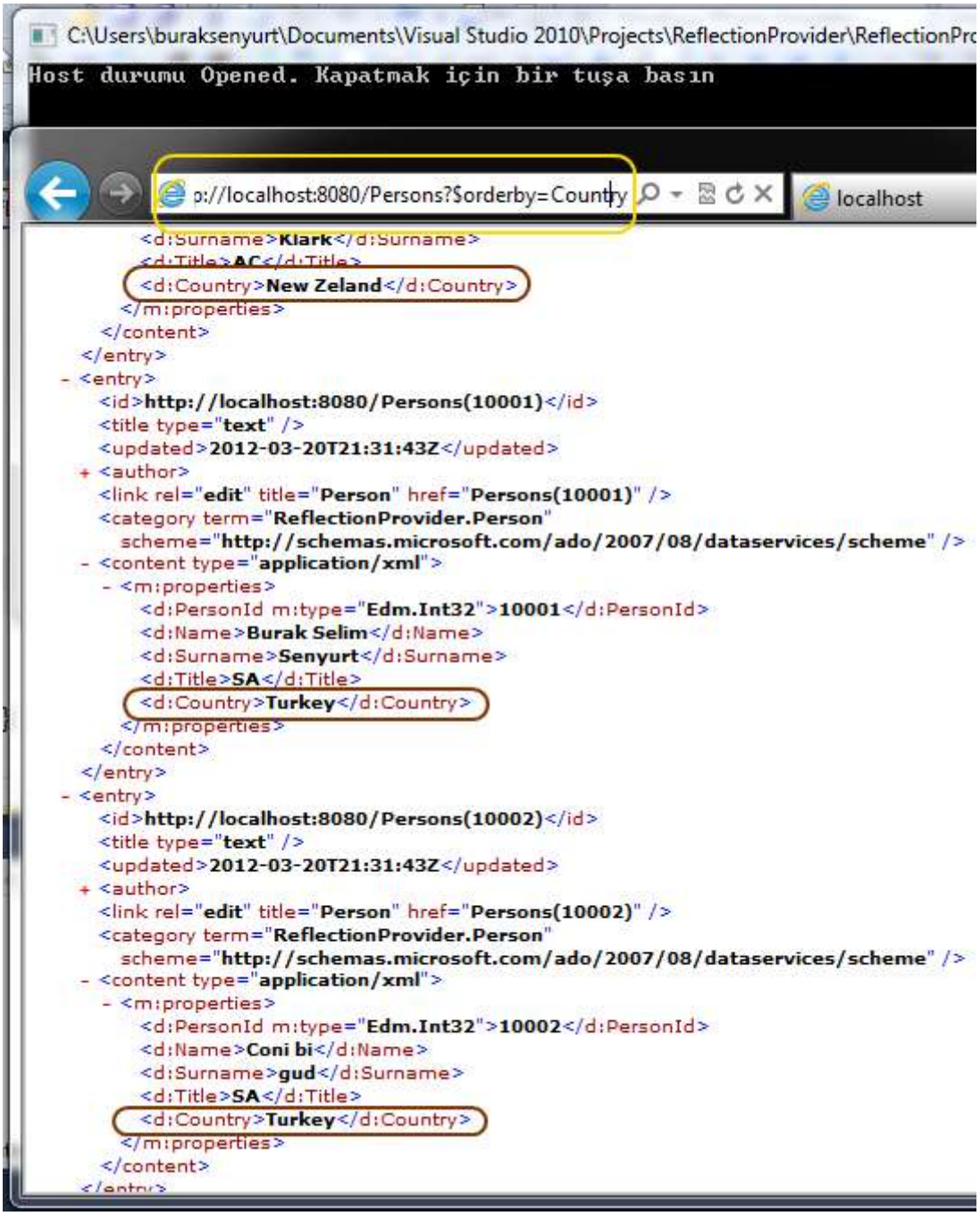


Dikkat edileceği üzere **Text** dosyamız içerisinde yer alan tüm içerik **XML** formatında tarayıcı uygulamaya gelmiştir.

İkinci sorgumuzda ise belirli bir **PersonId** değerine sahip içeriği getirmeye çalışacağız. Hatırlayacağınız gibi **Person** tipine uyguladığımız **DataServiceKey** niteliği ile, **PersonId** özelliğinin**unique** anahtar olduğunu belirtmiştik. Bu amaçla **http://localhost:8080/Persons(10002)** URL' ini deneyebiliriz. İşte sonuç



Çok doğal olarak servise başka sorgular da gerçekleştirilebilir.
Örneğin **Country** özelliğinin değerine göre alfabetik sırada listenin elde edilmesi sağlanabilir. Bunun için **orderby** anahtar kelimesini kullanmak yeterlidir. Tabi dikkat edilmesi gereken noktalardan birisi de, sorgu içerisinde yer alan özellik adlarının **case-sensitive** olarak ele alınması gerekliliğidir. Bir başka
deyişle, **Country**yerine **country** yazmak hataya neden olacak ve bir sonuç kümesi döndürülmeyecektir.
Dolayısıyla**http://localhost:8080/Persons?$orderby=Country** şeklindeki **URL** ifadesi aşağıdaki sonucun üretilmesini sağlayacaktır.



Görüldüğü gibi **WCF Data Service**' lerde **Reflection Provider**' larını kullanarak **Entity Framework** dışındaki kaynakları kullanmak son derece kolaydır. Bu yazıda geliştirdiğimiz örnekte,**POST, PUT** ve **DELETE** sorguları için gerekli destek yoktur. Bunun için **IUpdatable** arayüzünün ve onunla birlikte tanımlanan üyelerin **ezilmesi(override)** gerekmektedir. Tekrardan görüşünceye dek hepinize mutlu günler dilerim 🙂

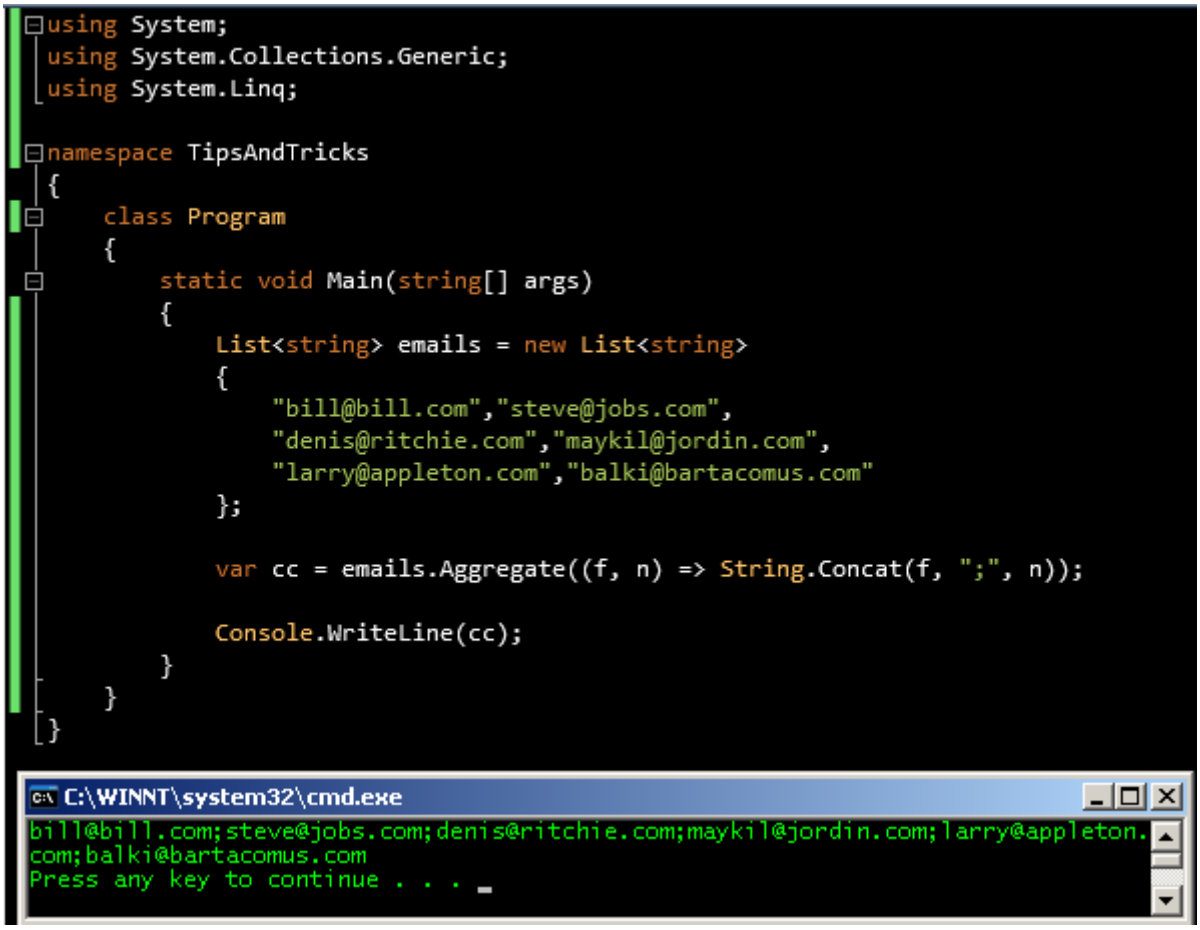
ReflectionProvider.rar (36,11 kb)

Tek Fotoluk İpucu 51 - String Birleştirirken Aggregate Kullanmak

Çarşamba, 25 Nisan 2012 00:40

C#, List, Aggregate, Extension Methods

Diyelim ki elinizde n sayıda e-mail adresi var ve bunları kod içerisinde string tipinden generic bir List koleksiyonunda saklıyorsunuz. Bu mail adreslerinin tamamına toplu olarak mail göndermek isterseniz genellikle aralarına virgül veya noktalı virgül işareti koyarak birleştirmeniz gerekir. Aslında bu amaçla basit bir for döngüsü/foreach döngüsü işinize yarayacaktır. Ya da aşağıdaki gibi LINQ' in getirdiği bazı extension method nimetlerinden de yararlanabilirsiniz



Görüşmek üzere

Tek Fotoluk İpucu 50 - Pivot Taklidi Yapan LINQ

Perşembe, 5 Nisan 2012 13:55

Linq, Pivot

Elimizde ülke bazlı bir toplam satış rakamlarını içeren bir veri listesi olduğunu düşünelim. Normal şartlarda bu tip bir çıktıyı sorguladığımızda veri içeriği ülke bazlı olacak şekilde dikine akacaktır. Ancak istediğimiz çıktı, ükle bazlı satışların toplam tutarlarını yatay eksene taşıyabilmek. Bir nevi**SQL** tarafındaki **PIVOT** hareketini gerçekleştirmek istiyoruz. Bunu bir **LINQ** sorgusu ile yapmaya ne dersiniz? Burdan buyrun

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace LINQPivot{
    class Program{
        static void Main(string[] args){
            List<CountrySale> sales = new List<CountrySale>{
                new CountrySale{Sequence=1,Country="Turkey",Prefix="TR",TotalSales=9035.45M},
                new CountrySale{Sequence=2,Country="Germany",Prefix="GR",TotalSales=1210.50M},
                new CountrySale{Sequence=3,Country="Norway",Prefix="NR",TotalSales=1050M},
                new CountrySale{Sequence=4,Country="Spain",Prefix="SP",TotalSales=1500.50M},
                new CountrySale{Sequence=5,Country="Italy",Prefix="IT",TotalSales=4500.99M},
                new CountrySale{Sequence=6,Country="Belgium",Prefix="BG",TotalSales=1350.25M}
            };

            Console.WriteLine("Country Sales");
            foreach (CountrySale sale in sales){
                Console.WriteLine("{0}\t\t{1}\t\t{2}",
                    sale.Sequence.ToString(),sale.Prefix,sale.TotalSales.ToString());
            }

            Console.WriteLine("\n\nCountry Sales");
            var result = from cs in sales
                         group cs by "Anything" into g
                         select new{
                             g.Key,
                             Turkey=g.Where(s=>s.Country=="Turkey").Sum(s=>s.TotalSales),
                             Germany= g.Where(s => s.Country == "Germany").Sum(s => s.TotalSales),
                             Norway = g.Where(s => s.Country == "Norway").Sum(s => s.TotalSales),
                             Spain= g.Where(s => s.Country == "Spain").Sum(s => s.TotalSales),
                             Italy = g.Where(s => s.Country == "Italy").Sum(s => s.TotalSales),
                             Belgium = g.Where(s => s.Country == "Belgium").Sum(s => s.TotalSales),
                         };
            foreach (var hr in result){
                Console.WriteLine("TR\tGR\tNR\tSP\tIT\tBG"
                    , hr.Turkey, hr.Germany, hr.Norway, hr.Spain, hr.Italy, hr.Belgium);
                Console.WriteLine("{0}\t{1}\t{2}\t{3}\t{4}\t{5}"
                    , hr.Turkey, hr.Germany, hr.Norway, hr.Spain, hr.Italy, hr.Belgium);
            }
            Console.WriteLine("\n");
        }
    }
    class CountrySale{
        public int Sequence { get; set; }
        public string Country { get; set; }
        public decimal TotalSales { get; set; }
        public string Prefix { get; set; }
    }
}
```

```
C:\WINNT\system32\cmd.exe
Country Sales
1               TR              9035,45
2               GR              1210,50
3               NR              1050
4               SP              1500,50
5               IT              4500,99
6               BG              1350,25


Country Sales
TR      GR      NR      SP      IT      BG
9035,45 1210,50 1050    1500,50 4500,99 1350,25


Press any key to continue . . .
```

BING Maps WCF Rest Servislerini Kullanmak

Pazartesi, 2 Nisan 2012 13:00

Bing Maps, Wcf Rest, Wcf

Merhaba Arkadaşlar,
Bazen öğrenmek istediklerimiz bize inanılmaz karşık gelir. Ne zaman kitabı açsak ya da bilgisayarın başına geçsek işe zaten demoralize olmuş bir şekilde başlarız. Özellikle tez hazırlıkları safhasındayken veya yazacağımız kitap için gerekli araştırmaları yaparken çok yoğun, ağır, sıkıcı ve uğraştırıcı unsurlarla karşı karşıya kalabiliriz. Yüksek Lisans yaptığım dönemlerdeki hocalarımdan birisi bu konuda şöyle bir tavsiye de bulunmuştu…
Araştırmanızda bir nokta da tıkandınız, takıldınız mı?(Bir süre sessizlik)…Tatile çıkın. Bir iki hafta o konu ile hiç uğraşmayın. Döndüğünüzde soruna çok daha farklı bir şekilde bakacağınızı göreceksiniz 😉
Doğruyu söylemek gerekirse hangi yolu kullanırsak kullanalım, kendimizi nasıl motive etmek istersek isteyelim, bazen araştırdığımız konu da öyle bir nokta yakalarız ki, gerisi çorap söküğü gibi gelir. İşte bu yazımızda bu çorap söküğünü bulmaya çalışıyor olacağız. Hatırlayacağınız üzere **BING Maps WCF Servislerini** değerlendirdiğimiz bir önceki yazımızda, söz konusu hizmetlerden yararlanabilmek için **proxy** tiplerinden faydalanmıştık*(Add Service Reference)*. Ancak bu servisleri sadece **proxy** tipleri üzerinden kullanmak gibi bir zorunluluğumuz bulunmamaktadır. Özellikle son yıllarda ön plana çıkan **REST** tabanlı servis yaklaşımı sayesinde, ilgili hizmetlerden **HTTP** protokolünün **GET** metodunu kullanaraktan da yararlanabiliriz**.** Bu çok doğal olarak platform bağımsızlık avantajını da beraberinde getirecektir. İşte bu yazımızda **BING Maps** servislerine ait **REST(Representational State Transfer)** arayüz noktalarını nasıl kullanabileceğimizi çok basit bir örnek üzerinden incelemeye çalışıyor olacağız. Gerisi çorağ söküğü gibi gelecek
BING Maps Rest servisleri de toplamda dört arayüz ile karşımıza çıkmaktadır. Adres, nokta bazlı yer bulma işlemleri vb için **Locations API**, harita bazlı rota gösterme ve metadata bilgisi elde edebilmek vb için **Imagery API**, yürüme yolu, araç yolu veya transit geliş gidişlere yönelik rota çıkartılması vb için **Routes API**, coğrafi bir alandaki trafik durum bilgisini vb öğrenebilmek için de **Traffic API REST** servislerinden yararlanılabilmektedir.
Aslında tüm **API** arayüzleri **HTTP** protokolünün **GET** metoduna göre talep kabul etmekte ve istemci tarafına buna uygun olacak şekilde **XML(eXtensible Markup Language)**veya **JSON(JavaScript and Object Notation)** çıktısı döndürmektedir. Bu iki çıktı içeriği de **W3C** tarafından kabul görmüş birer standart olduğundan, herhangibir platform tarafından kolaylıkla kullanılabilir. Hatta en basit anlamda tarayıcı uygulamalar

tarafından kolayca gösterilebilir. Dolayısıyla bilinmesi gereken, bu **API** arayüzlerine gönderilecek olan **URL** formatının nasıl olması gerektiği ve çıktıların kod tarafında nasıl ele alınabileceğidir.
İşe ilk olarak basit bir adım ile başlarsak gerisi çorap söküğü gibi gelecektir. Bu amaçla öncelikle bir adres bazlı lokasyon konumlandırma işinden başlayalım derim. Örneğin herhangibir tarayıcı üzerinden aşağıdaki talepte bulunduğumuzu düşünelim.
http://dev.virtualearth.net/REST/v1/Locations/US/NH/manchester/?o=xml&key={Buraya Developer API Key gelmeli}
Bu **URL** ifadesinden **Locations REST** arayüzüne*(ve hatta 1.0 versiyonuna)* bir talepte bulunduğu görülmektedir. **US** ile başlayan kısımda tahmin edileceği üzere ülke kodu belirtilir. Bu örnekte **Amerika Birleşik Devleteri(United States)** seçilmiştir. **US** kodunu izleyen kısımda ise bölge adının yazıldığı görülmektedir, **NH.** Sonraki kısımda ise şehir adı gelmektedir ki bu örnekte **Manchester** aranmaktadır. **?o** ile başlayan bölümde ise çıktı formatının tipi seçilir ki burada **XML** biçimi ele alınmıştır. Çok doğal **BING Maps** hizmetlerinden yararlanabilmek için bir **Developer Key** olması gerekmektedir**. key** anahtar kelimesinden sonra gelen kısımda da bu değer yazılır. Ben kendi developer key değişkenimi kullandığımda aşağıdaki **XML** çıktısını elde ettiğimi gördüm.

```
<Response>
  <Copyright>
    Copyright © 2012 Microsoft and its suppliers. All rights reserved. This API cannot be accessed and the content and any results may not be used, reproduced or transmitted in any manner without express written permission from Microsoft Corporation.
  </Copyright>
  <BrandLogoUri>
http://dev.virtualearth.net/Branding/logo_powered_by.png
  </BrandLogoUri>
  <StatusCode>200</StatusCode>
  <StatusDescription>OK</StatusDescription>
  <AuthenticationResultCode>ValidCredentials</AuthenticationResultCode>
  <TraceId>
    3c54bda6227b45a58f6c9947a79021d8|LTSM001153|02.00.83.500|LTSMSNVM002004, LTSMSNVM001455, LTSMSNVM001474, LTSMSNVM001465, LTSMSNVM001451, LTSMSNVM002052
  </TraceId>
  <ResourceSets>
    <ResourceSet>
      <EstimatedTotal>1</EstimatedTotal>
      <Resources>
        <Location>
          <Name>Manchester, NH</Name>
```

```
        <Point>
          <Latitude>42.991168975830078</Latitude>
          <Longitude>-71.463088989257812</Longitude>
        </Point>
        <BoundingBox>
          <SouthLatitude>42.936458587646484</SouthLatitude>
          <WestLongitude>-71.510566711425781</WestLongitude>
          <NorthLatitude>43.043960571289062</NorthLatitude>
          <EastLongitude>-71.415084838867188</EastLongitude>
        </BoundingBox>
        <EntityType>PopulatedPlace</EntityType>
        <Address>
          <AdminDistrict>NH</AdminDistrict>
          <AdminDistrict2>Hillsborough Co.</AdminDistrict2>
          <CountryRegion>United States</CountryRegion>
          <FormattedAddress>Manchester, NH</FormattedAddress>
          <Locality>Manchester</Locality>
        </Address>
        <Confidence>Medium</Confidence>
        <MatchCode>Good</MatchCode>
        <MatchCode>UpHierarchy</MatchCode>
        <GeocodePoint>
          <Latitude>42.991168975830078</Latitude>
          <Longitude>-71.463088989257812</Longitude>
          <CalculationMethod>Rooftop</CalculationMethod>
          <UsageType>Display</UsageType>
        </GeocodePoint>
      </Location>
    </Resources>
  </ResourceSet>
 </ResourceSets>
</Response>
```


Görüldüğü üzere söz konusu çıktı içerisinde Manchester mevkisinin coğrafik lokasyon bilgileri yer almaktadır. İlgili **XML** içeriğinin şemasını çıkarttığımızda ağaç yapısını daha kolay bir şekilde görebiliriz ve anlayabiliriz.

Aslında lokasyon ile ilişkili olarak asıl önemli bilgiler **Resources/Location** elementi altındaki boğumlarda yer almaktadır. Söz gelimi **Name** elementinde aranan kritere uygun olarak gelen lokasyonun adı, **BoundingBox** içerisinde kuzey, güney, doğu ve batı koordinatları, **Point** elementinde enlem ve boylam bilgileri vb yer almaktadır. Aranan içeriğin eşleşme oranı ise*(yani bulunan sonucun aranan ile ne kadar yakın olduğu bilgisi de)* **MatchCode** elementi içerisinde belirtilmektedir.
Aranan kritere göre çok daha fazla sonuç gelmesi de olasıdır. Örneğin,
http://dev.virtualearth.net/REST/v1/Locations/manchester/?o=xml&key={developer key}
şeklinde bir **URL** talebinde bulunduğumuzda bize an itibariyle 5 adet sonuç dönecektir. Nitekim burada ülke veya lokasyonu tam onikiden vurmak için gerekli ekstra bilgiler verilmemiştir. Sadece **BING** serverlarında kayıtlı olan **manchester** mevkisine ait veriler getirilmiş ve makalenin yazıldığı tarih itibariyle de 5 yakın sonuç bulunmuştur. *(Örnek erkan görüntüsünün bir kısmı aşağıdaki gibidir)*

```
<Location>
  <Name>Manchester, NH</Name>
+ <Point></Point>
+ <BoundingBox></BoundingBox>
  <EntityType>PopulatedPlace</EntityType>
- <Address>
    <AdminDistrict>NH</AdminDistrict>
    <AdminDistrict2>Hillsborough Co.</AdminDistrict2>
    <CountryRegion>United States</CountryRegion>
    <FormattedAddress>Manchester, NH</FormattedAddress>
    <Locality>Manchester</Locality>
  </Address>
  <Confidence>High</Confidence>
  <MatchCode>Good</MatchCode>
- <GeocodePoint>
    <Latitude>42.991168975830078</Latitude>
    <Longitude>-71.463088989257812</Longitude>
    <CalculationMethod>Rooftop</CalculationMethod>
    <UsageType>Display</UsageType>
  </GeocodePoint>
</Location>
<Location>
  <Name>Manchester, Manchester, United Kingdom</Name>
+ <Point></Point>
+ <BoundingBox></BoundingBox>
  <EntityType>PopulatedPlace</EntityType>
+ <Address></Address>
  <Confidence>High</Confidence>
  <MatchCode>Good</MatchCode>
+ <GeocodePoint></GeocodePoint>
</Location>
<Location>
  <Name>Manchester, TN</Name>
+ <Point></Point>
+ <BoundingBox></BoundingBox>
  <EntityType>PopulatedPlace</EntityType>
- <Address>
    <AdminDistrict>TN</AdminDistrict>
    <AdminDistrict2>Coffee Co.</AdminDistrict2>
    <CountryRegion>United States</CountryRegion>
    <FormattedAddress>Manchester, TN</FormattedAddress>
    <Locality>Manchester</Locality>
```

Bir kaç farklı örnek daha ilave ederek **REST** arayüz içeriklerini incelemeye devam edelim.

http://dev.virtualearth.net/REST/v1/Locations/turkey/kadiköy/?output=xml&key={Developer key}

Yukarıdaki sorgu ile Türkiye' deki Kadıköy ilçesinin lokasyon bilgisi elde edilebilir.

```
<EstimatedTotal>1</EstimatedTotal>
<Resources>
- <Location>
    <Name>Kadiköy, Turkey</Name>
  - <Point>
      <Latitude>40.986038208007812</Latitude>
      <Longitude>29.070089340209961</Longitude>
    </Point>
  - <BoundingBox>
      <SouthLatitude>40.949970245361328</SouthLatitude>
      <WestLongitude>29.014259338378906</WestLongitude>
      <NorthLatitude>41.012283325195312</NorthLatitude>
      <EastLongitude>29.161420822143555</EastLongitude>
    </BoundingBox>
    <EntityType>Neighborhood</EntityType>
  - <Address>
      <AdminDistrict>Istanbul</AdminDistrict>
      <CountryRegion>Turkey</CountryRegion>
      <FormattedAddress>Kadiköy, Turkey</FormattedAddress>
      <Locality>Kadiköy</Locality>
    </Address>
    <Confidence>High</Confidence>
    <MatchCode>Good</MatchCode>
  - <GeocodePoint>
      <Latitude>40.986038208007812</Latitude>
      <Longitude>29.070089340209961</Longitude>
      <CalculationMethod>Rooftop</CalculationMethod>
      <UsageType>Display</UsageType>
    </GeocodePoint>
  </Location>
</Resources>
```

http://dev.virtualearth.net/REST/v1/Locations?output=xml&countryRegion=DE&key={Developer Key}

Bu seferki sorgu ile de countryRegion=DE anahtar değer çiftini kullanarak Almanya' nın merkez koordinatlarını elde edebiliriz.

Peki çıktıyı **JSON** formatında almak istersek? Bu durumda **URL** sorgusundaki ufak bir değişiklik yapmamız yeterli olacaktır. Özellike **WCF Data Service** geliştiricileri, **URL**içerisinde önceden tanımlı pek çok anahtar kelimenin kullanıldığını bilirler. **BING Maps REST** servislerinde de benzer bir durum söz konusudur. Nitekim **?** den sonra gelen kısımlarda **anahtar kelime=değer** şeklinde **key-value** çiftleri yer almaktadır. Bu çiftlerlerin sayısı & operatörü ile arttırılabilir ve birden fazla kriterin hesaba katılması sağlanabilir. Eğer **o** parametresinin değeri **xml'** den **json'** a çekilirse, bu karşı taraftaki **BING Map Locations REST** servisi için çıktının **JSON** formatında hazırlanması gerektiği anlamına gelecektir.

http://dev.virtualearth.net/REST/v1/Locations/us/nh/manchester/?o=json&key={developer key}

Bu **URL** talebinin sonucunda aşağıdaki içerikte görülen **JSON** formatlı çıktı elde edilmiştir.

{"authenticationResultCode":"ValidCredentials","brandLogoUri":
"http:\/\/dev.virtualearth.net\/Branding\/logo_powered_by.png","copyright":"Copyright © 2012 Microsoft and its suppliers. All rights reserved. This API cannot be accessed and the content and any results may not be used, reproduced or transmitted in any manner without express written permission from Microsoft Corporation.","resourceSets":
[
{"estimatedTotal":1,"resources":
[
{"__type":
"Location:http:\/\/schemas.microsoft.com\/search\/local\/ws\/rest\/v1","bbox":
[42.936458587646484, -71.510566711425781,43.043960571289062, -71.415084838867188],"name":"Manchester, NH","point":
{"type":"Point","coordinates":[42.991168975830078,-71.463088989257812]},"address":
{"adminDistrict":"NH","adminDistrict2":"Hillsborough Co.",
"countryRegion":"United States","formattedAddress":"Manchester, NH","locality":"Manchester"},"confidence":"High","entityType":"PopulatedPlace","geocodePoints":
[{"type":"Point","coordinates":[42.991168975830078,-71.463088989257812]
,"calculationMethod":"Rooftop","usageTypes":["Display"]}],"matchCodes":["Good"]
}
]}
],"statusCode":200,"statusDescription":"OK","traceId":"d9816448d7e340c0a457ba230bd56310| LTSM001158|02.00.83.500|LTSMSNVM001472, LTSMSNVM002206, LTSMSNVM001475, LTSMSNVM001455, LTSMSNVM001465"
}

Pek tabi **JSON** formatlı çıktılar, **XML** formatlı çıktılara nazaran çok daha az yer kaplamaktadır.

Diğer servislerinde **REST** arayüzlerini kullanmak suretiyle çeşitli aramalar yapabiliriz. Örneğin **Routes API** arayüzüne kısa bir bakış atalım. Bu arayüzü kullanarak yürüyüş, sürüş veya transit geliş gidişler için rota bilgisi elde etmemiz mümkündür.

Eğer **İstanbul'** dan **Ankara'** ya doğru araba ile gideceğimiz bir rota bilgisi istersek, aşağıdaki gibi bir URL sorgusunu göndermemiz yeterli olacaktır.

http://dev.virtualearth.net/REST/V1/Routes/Driving?o=xml&wp.0=istanbul&wp.1=ankara&avoid=minimizeTolls&distanceUnit=km&key={Developer Key}

Sorgu sonucu elde edilen uzun XML çıktısına ait küçük bir ekran görüntüsü

```
<CompassDirection>east</CompassDirection>
- <Detail>
  <ManeuverType>DepartStart</ManeuverType>
    <StartPathIndex>0</StartPathIndex>
    <EndPathIndex>2</EndPathIndex>
  <Name>Yedikuyular Caddesi</Name>
    <CompassDegrees>105</CompassDegrees>
    <Mode>Driving</Mode>
    <PreviousEntityId>0</PreviousEntityId>
    <NextEntityId>0</NextEntityId>
   <RoadType>Arterial</RoadType>
  </Detail>
  <Exit/>
  <TollZone/>
  <TransitTerminus/>
  <IconType>Auto</IconType>
  <Time>0001-01-01T00:00:00</Time>
  <TransitStopId>0</TransitStopId>
 <TowardsRoadName>Cumhuriyet Caddesi</TowardsRoadName>
  <SideOfStreet>Unknown</SideOfStreet>
</ItineraryItem>
- <ItineraryItem>
 <TravelMode>Driving</TravelMode>
  <TravelDistance>0.358</TravelDistance>
  <TravelDuration>30</TravelDuration>
- <ManeuverPoint>
    <Latitude>41.040791</Latitude>
    <Longitude>28.98637</Longitude>
  </ManeuverPoint>
 <Instruction maneuverType="TurnRight">Turn right onto Cumhuriyet Caddesi</Instruction>
  <CompassDirection>south</CompassDirection>
- <Detail>
    <ManeuverType>TurnRight</ManeuverType>
    <StartPathIndex>2</StartPathIndex>
    <EndPathIndex>5</EndPathIndex>
    <Name>Cumhuriyet Caddesi</Name>
    <CompassDegrees>196</CompassDegrees>
    <Mode>Driving</Mode>
    <PreviousEntityId>0</PreviousEntityId>
    <NextEntityId>0</NextEntityId>
```

Bu **URL** sorgusunda önemli olan bazı **key'** ler vardır. **wp.0** ve **wp.1** ile tanımlanan anahtar kelimelere atanan değerler, sırasıyla **WayPoint 1** ve **WayPoint 2** anlamına gelmektedir. Bir başka deyişle, rota için gerekli **başlangıç** ve **bitiş noktaları** bilgileridir. **avoid** kelimesi seçimliktir ve burada atanan **minimizeTolls** değeri ile paralı yolların mümkün mertebe rota tanımından çıkartılması talep edilmektedir. **distanceUnit=km** anahtar değer çifti ile mesafelerin **km** cinsinden bildirilmesi sağlanmaktadır ki diğer seçenekte **mil** anlamına gelen **mi'** dir.

Diğer kullanılabilecek key değerleri için developer sayfasına bir göz atmanızı öneririm. Oldukça fazla sayıda alternatif bulunmaktadır.
Diğer API arayüzlerinden olan Static MAP Rest arayüzü ile de standart olarak 350X350 boyutlarında harita elde edilmesi mümkün olmaktadır. Bu haritayı uydu görüntüsü şeklinde, yol haritası şeklinde elde etmemiz de söz konusudur. Aslına bakarsanız BING Maps hizmetlerinin bana kalırsa en eğlencelilerinden birisi de bu API' dir. Örneğin
http://dev.virtualearth.net/REST/v1/Imagery/Map/AerialWithLabels/istanbul?mapSize=400,300&key={developer key}
URL sorgusu sonucunda aşağıdaki çıktıyı elde ederiz.

Bu sorguda **Imagery/Map/AerialWithLabels** ile şehrin coğrafik haritasının başlık bilgileri kullanılarak gösterileceği belirtilmektedir. istanbul kelimesini takip eden kısımlarda ise **mapSize** anahtar kelimesi kullanılmış ve üretilecek olan haritanın **400,300** boyutlarında olması sağlanmıştır.
http://dev.virtualearth.net/REST/v1/Imagery/Map/Road/Routes?wp.0=istanbul&wp.1=kocaeli&format=png&mapSize=800,600&key={developer key}
Yukarıdaki sorguda ise, yol haritası istenmektedir. **Map/Road** adresine gidilmesinin sebebi budur. Diğer taraftan **Routes** anahtar kelimesine atanan iki **Way Point** değeri ile **İstanbul** ile **Kocaeli** arası yol haritasının gösterilmesi talep edilmiştir. Söz konusu harita **800X600** pixel boyutlarında olacaktır ve **png** formatında üretilecektir. İşte sonuç,

Imagery servisindeki diğer anahtar kelimeler için yine **BING developer center'** daki web sayfasını ziyaret etmenizi öneririm.

Buraya kadar kı kısımda, söz konusu **REST** tabanlı servislerin **HTTP GET** talepleri ile nasıl sorgulanabileceğini anlamaya çalıştık. Görüldüğü gibi sorgu cümlesi içerisinde kullanılabilecek çok fazla sayıda **key=value** çifti bulunmaktadır. Sorguların sonuçlarını **XML** veya **JSON** formatında alabiliyor olmamız tüm platformların desteklenmesi açısından da oldukça önemlidir. Tabi **Imagery** servisi kullandığımızda çıktılar **GIF, JPG** veya **PNG** formatında olmaktadır.

Ancak çıktılar hangi format olurlarsa olsunlar, sonuç itibariyle bu üretimin kod tarafında anlamlı ve işe yarar hale getirilmesi gerektiğinden eminim ki hepimiz hem fikirizdir. Öyleyse dilerseniz basit bir kod parçası yardımıyla, **Locations REST** arayüzüne nasıl talepte bulunabileceğimize ve sonuçları nasıl değerlendirebileceğimize bakalım. Bu amaçla aşağıdaki kod parças ile işe başlayalım.

```
using System;
using System.Configuration;
using System.Linq;
using System.Net;
using System.Xml.Linq;
```

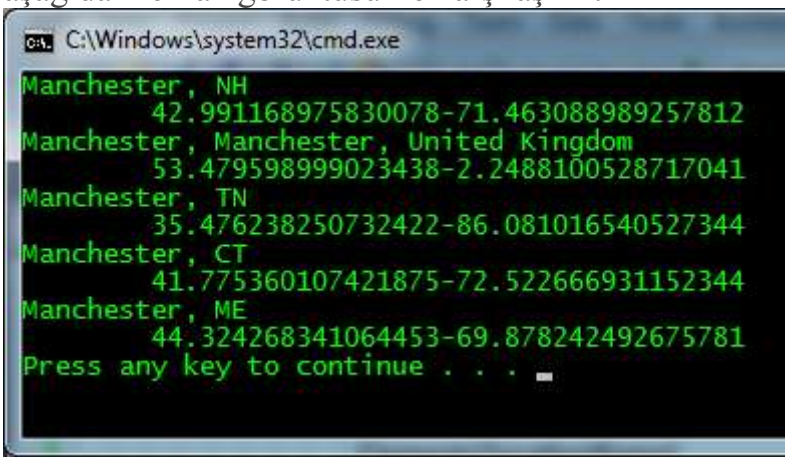
```
namespace BingRestClient
{
   class Program
   {
      static void Main(string[] args)
      {
         // BING Maps servisleri için kullanacağımız Developer Key değerini config dosyasından okuyoruz
         string apiDeveloperKey = ConfigurationManager.AppSettings["DeveloperKey"].ToString();
         // Basit bir sorgu giriyoruz. Birden fazla Location' ı ele almak istediğimizden sadece manchester değerini verdik
         string query = "manchester";
         // Tüm elemenler xmlNamespace değişkeni ile tanımlı Xml Namespace' i kullanmakta
         string xmlNamespace = "http://schemas.microsoft.com/search/local/ws/rest/v1";
         // Buna bağlı olarak gerekli XmlNamespace tanımlamalarını içeren XName değişkenleri oluşturuluyor. Bunlar XLINQ sorgusunda kullanılıyor olacak.
         XName resourceSetsName = XName.Get("ResourceSets", xmlNamespace);
         XName resourceSetName = XName.Get("ResourceSet", xmlNamespace);
         XName resourcesName = XName.Get("Resources", xmlNamespace);
         XName locationName = XName.Get("Location", xmlNamespace);
         XName name = XName.Get("Name", xmlNamespace);
         XName point = XName.Get("Point", xmlNamespace);
         // HTTP GET ile gidecek olan sorgu oluşturuluyor
         string url = String.Format("http://dev.virtualearth.net/REST/v1/Locations/{0}/?o=xml&key={1}", query, apiDeveloperKey);
         XElement locationElement = null;
         try
         {
            // XElement tipinin static Load metodu yardımıyla ilgili URL' e talepte bulunuluyor.
            locationElement = XElement.Load(url);
            // Basit bir XLINQ sorgusu kullanılarak ResourceSet elementlerine kadar iniliyor
            var resourceSet = from n in locationElement
                              .Elements(resourceSetsName)
                              .Elements(resourceSetName)
                              .Elements(resourcesName)
```

```
                    .Elements(locationName)
                     select n;
            foreach (var r in resourceSet) //Her bir ResourceSet elementi dolaşılıyor
            {
               Console.WriteLine(r.Element(name).Value); // önce Name elementinin değeri
               foreach (var p in r.Elements(point)) // ardından her bir Point elementi
```

içerisindeki her bir elementin değeri (ki bunlar longtitude ve latitude değerleri) dolaşılıp ekrana yazdırılıyor.

```
               {
                  Console.WriteLine("\t{0}\t", p.Value);
               }
            }
         }
         catch (WebException excp)
         {
            Console.WriteLine(excp.Message);
         }
      }
   }
}
```

Kodda dikkat edilmesi gereken noktalardan bir tanesi, **XLINQ** sorgularını gönderirken **XmlNamespace** kullanılmasıdır. Eğer bu **Xml Namespace** bilgisini ele almazsak söz konusu elementlerin hiç birisine ulaşamayız. Diğer yandan, **URL** bazlı olarak okuma işlemini gerçekleştirmek için, **XElement** tipinin **static Load** metodundan yararlanılmıştır. Eğer URL geçerli ise dönen sonuç **XML** formatında, **XElement** nesne örneğine yüklenecektir. Biz de bu içerik üzerinde dolaşarak sadece lokasyonun adını ve enlem ile boylam değerlerini ekrana yazdırmaya çalıştık. Örneğimizi çalıştırdığımızda aşağıdaki ekran görüntüsü ile karşılaşırız.



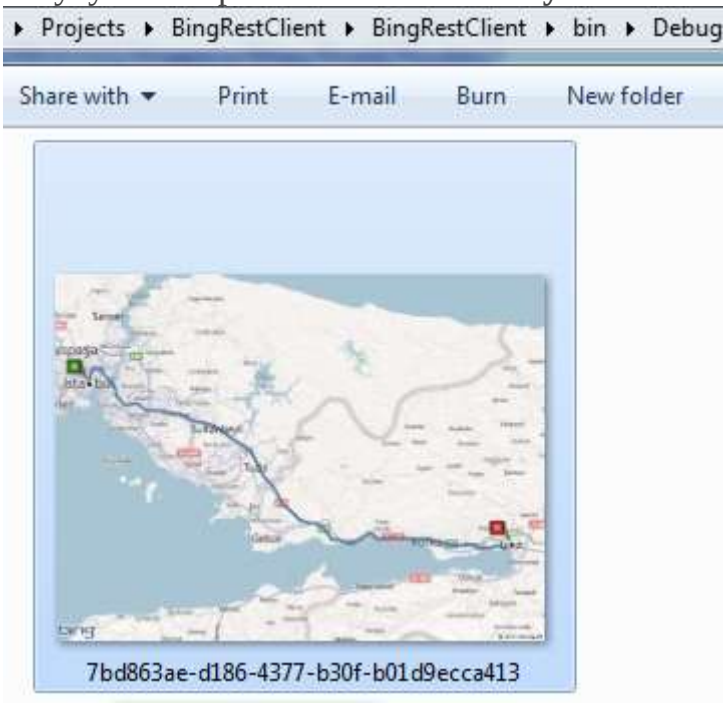
Görüldüğü üzere **manchester** kelimesi için **BING** servisi dünya üzerinde 5 adet lokasyon bilgisi önermiştir. Çok doğal olarak örneğimiz **XML** formatını değerlendirecek şekilde geliştirilmiştir. Eğer **JSON** formatında bir içerik okumak istersek bu durumda JSON ile

ilişkili serileştirme tipinden**(DataContractJsonSerializer)** yararlanabiliriz. *(Bu konuda MSDN üzerinden yayınlanan şu makaleyi incelemenizi öneririm.)*
Yazımızı sonlandırmadan önce son olarak Imagery servisine ait basit bir kod parçası geliştirelim. **Imagery** servisi bildiğiniz üzere **XML** veya **JSON** formatı yerine, **JPEG, PNG** veya **GIF** formatında resim çıktısı sunabilmektedir. Dolayısıyla bu servise yapacağımız talepleri farklı bir şekilde ele almamız gerekmektedir. İşte örnek kod parçamız.

```
string imagerUrl =
String.Format("http://dev.virtualearth.net/REST/v1/Imagery/Map/Road/Routes?wp.0=istan
bul&wp.1= kocaeli&format=png&mapSize=800,600&key={0}", apiDeveloperKey);
        try
        {
           HttpWebRequest request =
(HttpWebRequest)WebRequest.Create(imagerUrl);
           using (HttpWebResponse response =
(HttpWebResponse)request.GetResponse())
           {
              List<byte> content = new List<byte>();
              using (Stream stream = response.GetResponseStream())
              {
                 int currentByte;
                 while ((currentByte = stream.ReadByte()) != -1)
                 {
                    content.Add((byte)currentByte);
                 }
              }
              string filePath = String.Format("{0}.png",
Path.Combine(Environment.CurrentDirectory, Guid.NewGuid().ToString()));
              File.WriteAllBytes(filePath, content.ToArray());
           }
        }
        catch (WebException excp)
        {
           Console.WriteLine(excp.Message);
        }
```

Bu sefer bir **Stream** kullanmak durumundayız. Nitekim **URL** olarak talep ettiğimiz sorgu sonucunda bize okunabilir bir grafik içeriği gelmekte. Talebi oluşturmak ve**response** içeriğini almak için **HttpWebRequest** ve **HttpWebResponse** tiplerinden yararlandık. **HttpWebResponse** ile elde ettiğimiz nesne örneği üzerinden hareket ederek bir **Stream** oluşturduk ve bu **Stream** nesne örneği üzerinden resim dosyasına ait her bir **byte** içeriğini okuyarak generic bir **List<byte>** koleksiyonunda topladık. Çok doğal

olarak elde edilen **byte[]** içeriği resim dosyasının kendisini ifade etmektedir. Bunu da sembolik olarak üretilen ve **GUID** ile isimlendirdiğimiz örnek bir dosyaya **File** tipinin**static WriteAllBytes** metodu ile kaydettik. İşte sonuç;



Artık çorabı yırttık diye düşünüyorum. Bir başka deyişle gerisinin çorap söküğü gibi gelmesi lazım. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim 🙂

BingRestClient.rar (1,20 mb)

Tek Fotoluk İpucu 49–Daha Hızlı Count

Çarşamba, 7 Mart 2012 09:50

Sql, Sql Server 2008, T-Sql, Transact Sql, Row Count, Count, Select

Merhaba Arkadaşlar,
Çok yüksek rakamlarda satır içeren(Milonylarca Satır) tablolar söz konusu olduğunda, bunların satır sayılarını, Count Aggregate fonksiyonu ile bulurken süre kaybı yaşıyorsak ve sonuçları geç alıyorsak daha hızlı bir yola başvurabiliriz. Nasıl mı?

```
Select distinct
    S.row_count [Row Count]
    ,O.Name [Table Name]
from
    sys.dm_db_partition_stats S
    join sys.objects O
    on S.object_id=O.object_id
    where O.type='U'
order by S.row_count desc
```

Results | Messages

	Row Count	Table Name
1	121317	SalesOrderDetail
2	113443	TransactionHistory
3	89253	TransactionHistoryArchive
4	72591	WorkOrder
5	67131	WorkOrderRouting
6	31465	SalesOrderHeader
7	27647	SalesOrderHeaderSalesReason
8	19972	Contact
9	19614	Address
10	19220	CustomerAddress
11	19185	Customer
12	19118	ContactCreditCard
13	19118	CreditCard
14	18484	Individual
15	13532	CurrencyRate
16	8845	PurchaseOrderDetail

Query executed successfully.

Tek Fotoluk İpucu-48(Uri Extensions for RSS)

Cuma, 16 Mart 2012 09:00

Uri, Extension Methods, Rss, Html

Siz de benim gibi aklınıza geldikçe ve vaktiniz oldukça **Extension Method(Genişletme Metodu)**yazmaya çalışanlardan mısınız? 🙂 Geçtiğimiz gün **Uri** tipi için **RSS Feed** kaynağını okuyan ve gelen içeriği basit **HTML** formatı ile geriye döndüren bir genişletme metodu yazmaya çalıştım. Aynen aşağıda görüldüğü gibi 😉 Size kalan ise bunun **Atom** formatı ile çalışan versiyonnu yazmak 🙂

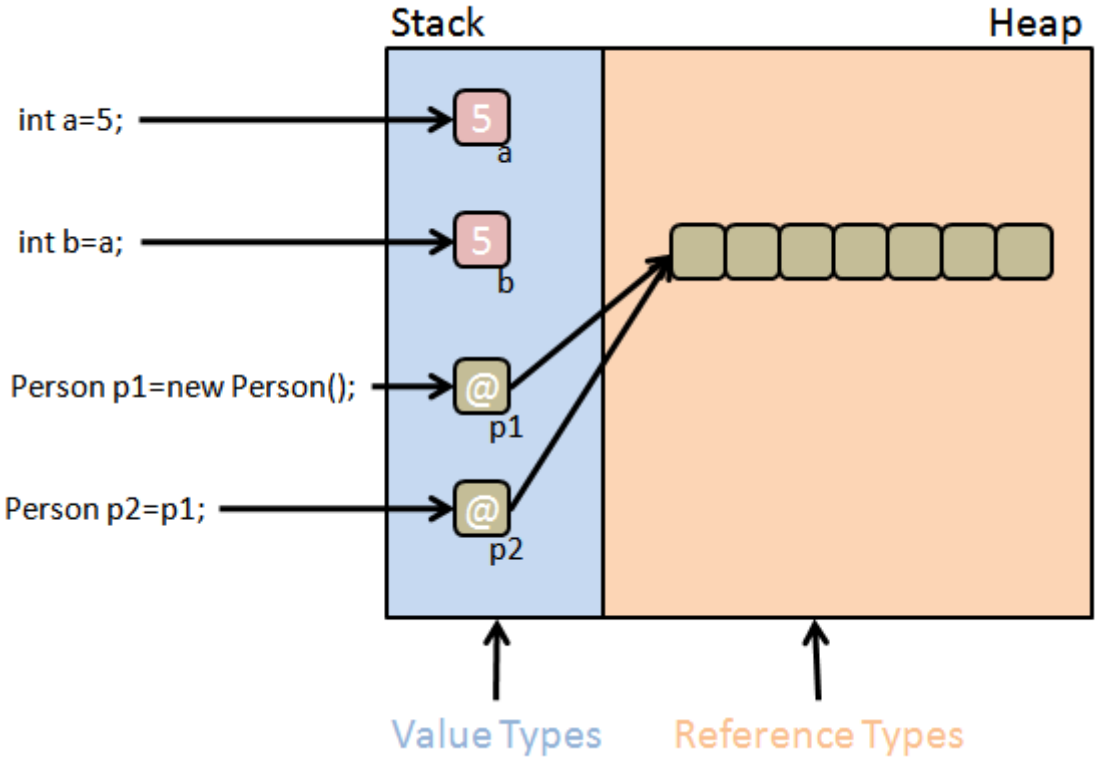
```
namespace UrlExtensions{
    class Program{
        static void Main(string[] args){
            try{
                Uri uri = new Uri("http://feeds.feedburner.com/TalesFromTheEvilEmpire");
                string response = uri.GetRssFeed();
            }
            catch (Exception excp){
                Console.WriteLine(excp.Message);
            }
        }
    }

    public static class UriExtensions{
        /// <summary> ...
        public static string GetRssFeed(this Uri FeedAddress){
            StringBuilder builder = new StringBuilder();
            XDocument rssFeed = null;

            try{
                rssFeed = XDocument.Load(FeedAddress.ToString());
            }
            catch (WebException excp){
                throw excp;
            }

            builder.AppendLine(string.Format("<b><a href='{0}'>{1}</a></b><br/><br/>"
                , rssFeed.Root.Element("channel").Element("link").Value
                , rssFeed.Root.Element("channel").Element("description").Value));

            var posts = (from item in rssFeed.Descendants("item")
                         select new
                         {
                             Title = item.Element("title").Value,
                             Link = item.Element("link").Value,
                             PublishedDate = item.Element("pubDate").Value
                         }).OrderByDescending(f=>f.PublishedDate);

            foreach (var post in posts)
                builder.AppendLine(string.Format("<a href='{0}'>{1}</a><br/>",post.Link,post.Title));

            return builder.ToString();
        }
    }
}
```

.Net Memory Management' i Kavramak

Cuma, 2 Mart 2012 15:54

Garbage Collection, Garbage Collector, .Net Memory Management, Gc

Merhaba Arkadaşlar,
Matix! Ne filmdi ama değil mi? Özellikle yazılım tarafına hakim olan bizler için, filmin içerisindeki pek çok gönderi anlamlı birer mesaj haline gelmişti. İlk bölüm zaten efsanenin başlangıcı olma niteliğindeydi. İkinci bölümde işler daha da bir farklılaştı tabi. Örneğin, silinmeyen ve Matrix içerisinde kendini geliştirip küçük bir krallık yaratan **Merovingian**karakteri vardı. Bu sistem içerisinde yer alan ve süresi dolduktan sonra silinmesi gereken bir program iken, kaynağa (Source) geri dönmemişti.
Sanki C++ ile geliştirilmiş bir değişken tipiydi de, Release edilmesi unutulmuş ve bellek üzerinde bir şekilde ayakta kalmış bir programcıktı 🙂 Şimdi nereden çıktı bu **Matrix, Merovingian** diyeceksiniz**.** Konumuz**.Net bellek yönetimi**. Ama bu kez biraz daha farklı ve detaylı.
.Net Framework' ün bilindiği üzere en önemli özelliklerinden birisi de **Managed Code** adını verdiğimiz yaklaşımı destekliyor olmasıdır. Hatta özellikle bu yaklaşım üzerine kurulmuş bir alt yapı mimarisi sunduğunu ifade edebiliriz. **Managed Code** denildiğinde, üretilen kodun çalışma zamanında bir ortam tarafından kontrol atlında tutulduğu sonucuna varmamız yeterlidir. **.Net Framework**' ün içsel yapılarını göz önüne aldığımızda, çalışma zamanı yani **Common Language Runtime**, yürütülmekte olan **Assembly**' lar ve ilişkili **Module**' ler ile ilgili olarak bir çok yönetsel kontrol mekanizmasını devreye sokar. Örneğin **Exception Handling, Code Access Security, Type Safety** ve en önemlilerinden birisi olan **Memory Management.**
Özellikle **C++** gibi programlama dilleri, sistemler üzerinde özgürce kod geliştirebilmemize olanak sağlamaktadır. Örneğin bellek üzerinde **pointer** gibi temel tipler yardımıyla her noktaya erişebilir ve nesnelerin yaşam döngülerini çok daha esnek bir biçimde ele alabiliriz. Tabi bu özgürlük,**development'** ı biraz daha zorlaştırır. Pointer aritmetiği ile uğraşılmak zorunda kalınır ve bellek yönetimi güçleşir. Sonuç pek tabi, kaynağa geri dönmesi unutulan bir Merovingian olup çıkar ki bunun doğal yansıması da genellikle **Memory Leak** ve kötü performans olur. Bu ve buna benzer bazı nedenlerden dolayı, **.Net Framework** daha ilk sürümünden itibaren, kodu kontrol altında tutmuş ve belleği yönetimini ağırlıklı olarak üzerine almıştır. Bize de belirli ölçülerde esnetmeler sunmuştur.
.Net ile geliştirme yapmaya veya onu öğrenmeye başlayan hemen her programcı aşağıdakine benzer bir şekil ile de mutlaka karşılaşır.

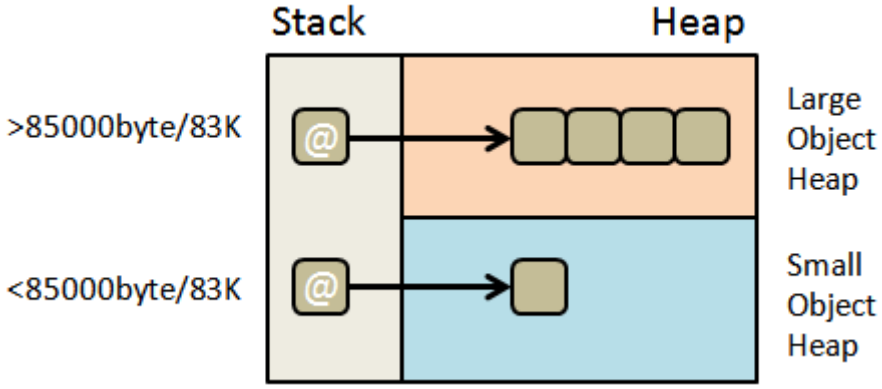


Bize öğretilen, bizim öğrendiğimiz ve hatta öğrettiğimiz haliyle, **.Net Framework** içerisinde veri türleri iki ana dala ayrılır. Belleğin **Stack** bölgesinde tutulan **değer türleri(Value Types)** ve belleğin **Heap** bölgesinde tutulan **referans türleri(Reference Types). int, double, Point, DateTime** gibi aslında **Common Type System** içerisinde birer **struct** ile ifade edilebilen tüm tipler değer türü iken, **class** gibi tipler de referans türleridir. Özellikle bunların kendi aralarındaki atamalarında bellek üzerindeki işleyiş şekilleri de çoğunlukla farklıdır. Aksi belirtilmediği ve müdahale edilmediği sürece referans türleri arası yapılan atamalar, aslında **stack** bölgesindeki adres işaretçilerinin çoğullanması ama heap üzerindeki aynı adres bölgesinin ifade edilmesidir. Değer türlerinde ise bu durum tam tersidir. Değerler **stack** bölgesinde atamalar sonrası kopyalanırlar.

Aslında bu bilgiler bizim için temel niteliği taşımaktadır. Dedik ya, **CLR** aslında çalışma zamanındaki bellek yönetimini de üstlenmektedir. O yüzden çoğumuz, **"nasıl olsa belleği birileri yönetiyor, nesneleri de zamanı gelince temizliyor, ortalığı toparlıyor"** diyerek temel olan başka bir konuyu da atlarız. Gerçekten de **.Net Memory Management** acaba nasıl çalışmaktadır? Eğer bunu merak ediyorsanız, yaptığım araştırmalar ve kendimce edindiğim fikirler ile konuyu sizlere aktarmaya çalışıyor olacağım. Dolayısıyla bundan sonrasını merak ediyorsanız okumaya devam edin 😉

Uygulamalarımızın çalışma zamanında ürettiği referans tiplerinin **Garbage Collector** tarafından ele alındığını biliyoruz aslında. Hatta **GC**, **GCSettings** gibi tipler yardımıyla ona bir ölçüde müdahale etme şansımız da bulunmakta. Teorik olarak **Heap** bellek bölgesindeki nesne örneklerinin yaşam döngüsünden, onların bellek üzerindeki fragmantasyonlarından ve elbetteki serbest

bırakılmalarından sorumlu olduğunu özetleyebiliriz. **Garbage Collector** ilke olarak iki tip nesne ile ilgilenir.



Aslında bir **.Net** uygulaması **process** olarak belleğe açıldığında, **Managed Heap** üzerinde o process'e ait olacak şekilde iki farklı alan göz önüna alınır. Bunlardan birisi uygulamanın**83Kilobyte** ve daha az büyüklükteki nesneleri içindir ki **Small Object Heap(SOH)** olarak adlandırılır. Boyutu **83Kb** üzerinde olan nesneler içinse **Large Object Heap(LOH)** olarak adlandırılan başka bir **heap** bloğu göz önüne alınır.

Tekil bir .net Process' i Win32 platformunda çalıştırıldığında bellek üzerinden maksimum 2Gb' a kadar yer kullanabilir.

Çok doğal olarak **Garbage Collector** söz konusu bu nesnelerin bellek üzerinde yerleştirilmeleri(allocate), fragmante edilmeleri(re-allocate) ve geri çağırılmaları(reclaim) vb işlemler sırasında kritik bir rol üstlenir. Şimdi ilk olarak **SOH** tarafına bir bakalım. **SOH** temel olarak zaman içerisinde nesne ömürlerini 3 farklı jenerasyonda oluşacak şekilde ele almaktadır*(Generational yaklaşım)*.

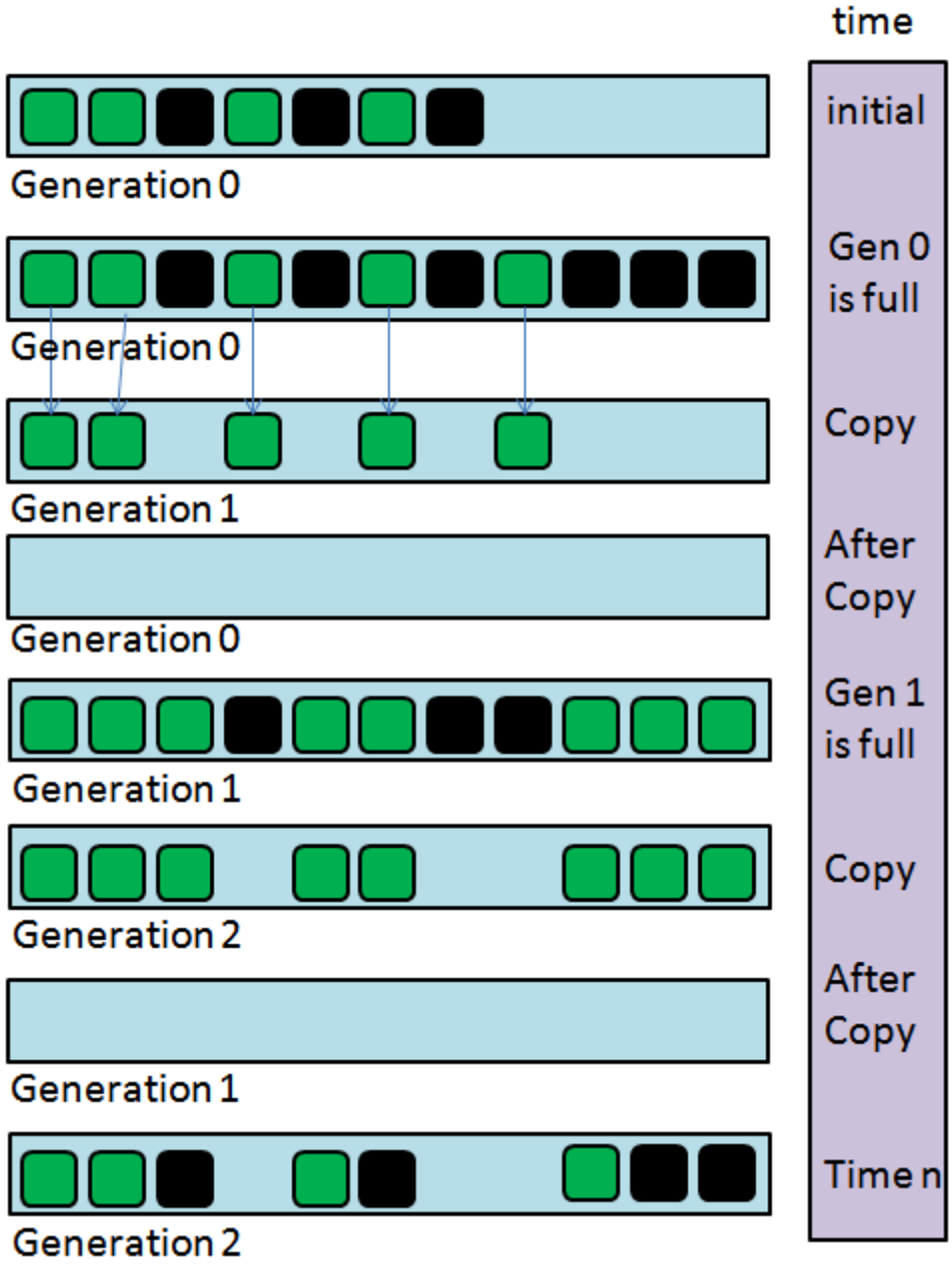
Generation 0, Generation 1 ve Generation 2.

SOH içerisinde yer alması gereken nesne örnekleri oluşturuldukça **Generation 0** adı verilen süreçte sırasıyla yerleştirilmeye başlarlar*(Burada C++ taki Linked List tarzındaki bellek açılımından farklı bir durum söz konusu)*. Çok doğal olarak zaman içerisinde bu bölgede yer alan bazı nesneler **Dispose**edilme aşamasına gelir. **GC** varsayılan davranış stratejisine göre **Generation 0** dolana kadar bir aksiyonda bulunmaz**. Generation 0** dolduğunda, **Dispose** sürecine girmesi gereken atıl nesneler toplanmaya ve halen yaşamakta olan canlı nesnelerde **Generation 1** bölgesine kopyalanarak taşınmaya başlanırlar.

Generation 1 bölgesinin de çok doğal olarak bir kapasitesi vardır ve zaman içerisinde buradaki canlı nesnelerden bazıları yine **Dispose** sürecine girecektir.

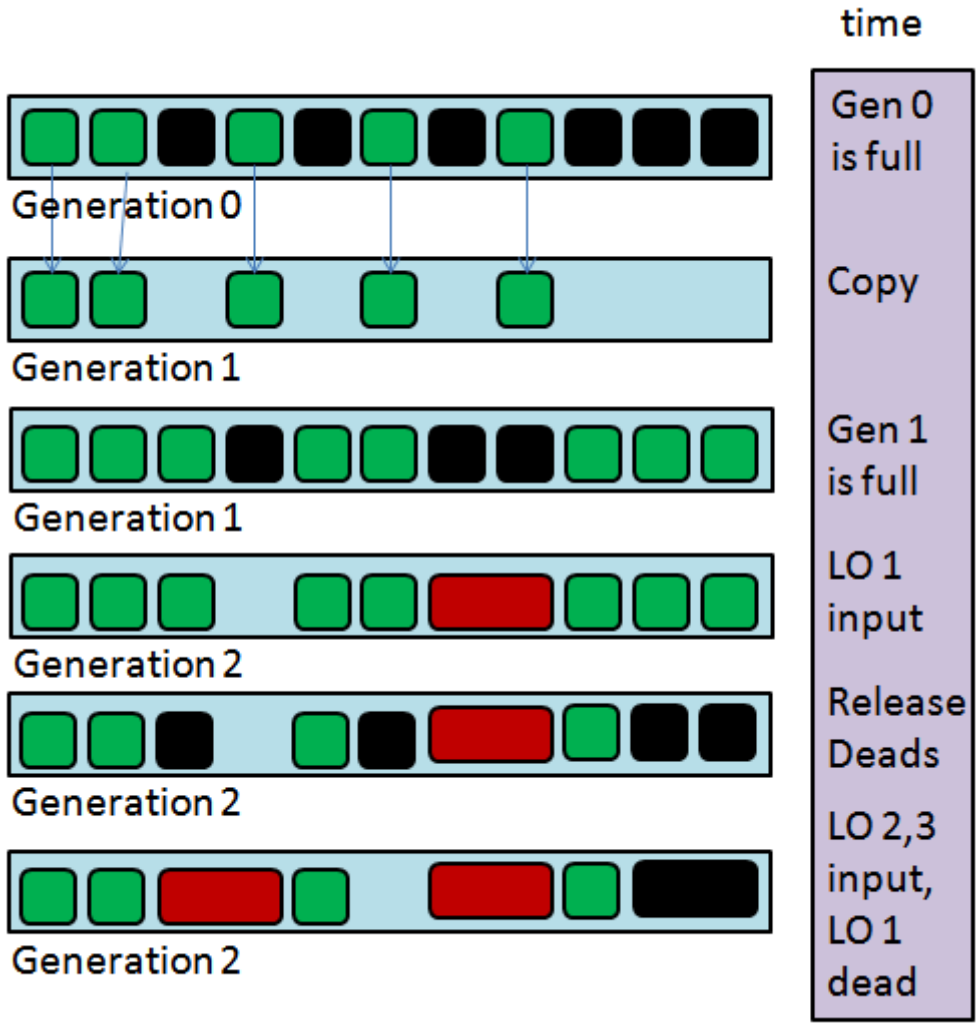
Dolayısıyla **Generation 1** bölgesinin de dolması sonrası yine canlı nesnelerin bu kez **Generation 2** bölgesine kopyalanması ve atıl nesnelerin **Generation 1** den atılması söz konusu olacaktır.

Durumu kabaca bu şekilde düşünecek olursak aşağıdaki gibi bir zaman diagramını göz önüne almamız mümkün olabilir. Kabaca tabi



Ancak, olay bu kadar da basit değildir. Aslında **GC** mekanizması ana uygulama **Thread'** inden bağımsız olarak çalışan farklı bir **Thread** olarak düşünüldüğünde, söz konusu işlemleri **concurrent**olarak gerçekleştirmektedir. Özellikle **Generation 0,1** ve **2** bölgeleri üzerinde her zaman şekilde olduğu gibi sıralı ve düzgün bir dizilim söz konusu olmayacaktır. Dolayısıyla kopyalama metoduna göre yapılan taşıma işlemleri sırasında, nesneler boş bulunan bellek bölgelerine atılırlar.
Diğer yandan kopyalama işlemleri sırasında oluşabilecek bir sorun da vardır. **GC** ayrı bir **Thread**üzerinden, bir alt **Generation'** daki canlı nesneleri tespit ettikten sonra, bunları bir üst generation' a kopyalar. Lakin alt **Generation'** daki nesneler bu taşıma sırasında veya öncesinde halen daha ana veya farklı bir **Thread** tarafından kullanılıyor olabilirler. Hımmm

İşte bu noktada **GC** şöyle bir yol izler. **Thread'** ler arası güvenli bir nokta oluşturur*(Safe Point)* ve taşıma sırasında ilgili uygulama **Thread'** lerinin tamamı durdurulur. Sonrasında ise kopyalanan tüm içeriği orjinal referansları ile eşleştirerek düzeni korur. Güzel bir trick öyle değil mi? 😀 Bir o kadar da karışık aslına bakarsanız. *(Ben hala konu ile ilişkili kaynakları ve CLR via C#' ın ilgili bölümlerini okuyarak pekiştirmeye çalışıyorum)*
Generation 2 bölgesi aslında performans ölçümlerinde de ip ucu veren bir alan olarak düşünülmektedir. Bu bölgenin çok sık ve fazla şişerek dolması ileride programın bellek ile ilişkili sıkıntılar üretebileceğinin de bir işaretedir.
ANTS Memory Profiler gibi araçlar yardımıyla, uygulamalarımızın bellek üzerindeki ölçümlerini detaylamasına yapabiliriz. Tabi daha ucuz çözümler de var. CLR Performance Counter' lar 😉
Gelelim **LOH(Large Object Heap)** bölgesine. **83KB** üzeri olarak belirtilen **Large Object'** lerin taşıma/kopylama maliyetleri tahmin edileceği üzere yüksektir. Bu sebepten dolayı **SOH** için uygulanan **Generations** tekniği yerine farklı bir yaklaşım kullanılır. **Generation 2** parçasında**Large Object** nesnelerinin, ölen nesnelerden boşalan yerlere iliştirilmesi söz konusudur. Aslında aşağıdaki şekil ile durumu biraz olsun ifade edebiliriz.



Bir **LO** eklenmek istendiğinde **Generation 2** kısmındaki ilk boş bölgeye açılması söz konusudur. Sonrasında sisteme dahil olacak diğer **LO'** ler de **Generation 2'** de boş olan yerlere serpiştirilirler. Tabi **Generation 1** den gelen nesne örnekleri **83Kb**' den küçük olduklarından, yeni gelen **83Kb'** den büyük nesnelerin sığabilecekleri uygun yerlerinde **Generation 2** üzerinde var olması gerekir.
Peki yoksa? 😬 Bu durumda uygulama daha fazla bellek alanının allocate edilmesi için işletim sisteminde bir talepte bulunacaktır. Hatta **Heap** alanının yetmediği durumlarda, fiziki disk bölgelerinden sanal olarak bu alanların karşılanması istenecektir. Eğer işletim sisteminden olumlu bir cevap alamazsa bu durumda **GC'** nin **Generation 2** içerisinde yapacağı **de-allocate** işlemlerinin yeteri kadar yer ayırması beklenecektir.
LO' ler için uygulanan strateji, generations sistematiğine göre daha performanslıdır. Nitekim bellek üzerinde **copy/move/re-refereance** işlemleri yoktur. Ancak belleğin fragmente edilme noktalarında da bir dezavantaj oluşturacaktır. Tabi LOH ile SOH' un bir arada çalıştığı da unutulmamalıdır. GC her iki tür için gerekli yönetsel işlemleri üstlenmektedir.
İşte tüm bunlar göz önüne alındığında karmaşık olan ve bize aslına bakarsanız çok fazla sorun çıkartıp baş ağrısı yapmayan en basit uygulamalarımızın bile, şöyle bir bellek testinden geçirilerek ne yaptıklarının incelenmesinde yarar olduğu söylenebilir.
.Net Framework içerisinde sisteme gelen ve **CLR'** ın bellek yönetimi odaklı olarak kullanılan pek çok performans ölçüm kriteri bulunmaktadır. **Bytes in all heaps, time spent in GC, allocated bytes per sec** vb…Bu tip kriterlere bakılarak geliştirdiğimiz **.Net** uygulamalarının bellek yönetimi açısından daha performanslı hale getirilmesi, istatistiki bilgilerinin çıkartılması, farklı ürün stratejilerinin belirlenmesi de söz konusu olabilir. Diğer yandan **Garbage Collector'** un kod tarafından da ele alınması ve bazı kurallarının değiştirilmesi söz konusu olabilir. Özellikle performans ve heap' in etkin kullanımı arasında fedakarlık yapılması gerektiği durumlarda*(kısaca performance ve heap etkinliği arası trade-off diyelim)* uygun modun seçilmesi sağlanabilir. Bu anlamda **GC** iki temel modu desteklemektedir**. Workstation** ve **Server.**
Workstation modu, kullanıcıya maksimum cevap verilebilirlik için tercih edilmekte olup**Concurrent** ve **non-Concurrent** çalışacak şekilde ele alınabilir. Varsayılan olarak **Concurrent**çalışma prensibi uygulanır. Buna göre **GC** mekanizması uygulama ile birlikte ayrı bir **Thread**üzerinden işlemlerini gerçekleştirir.
Server mode ağırlıklı olarak **performans(Performance)**, **ölçeklenebilirlik(Scalability)** ve**verimliliğin(through put)** ön plana çıktığı **sunucu ortamlarında(Server Environment)** göz önüne alınmaktadır. Bu modda, generation eşik değerleri ile bellekteki segment boyutları,**Workstation Mode'** a göre çok daha yüksektir. Bu son derece doğaldır nitekim sunucuların bellek kapasiteleri, workstation' lara göre daha fazldır 🙂 **Server Mode** ile çalışmanın en önemli artısı ise**paralel** veya **multi-thread** olarak çalışabilmesidir. Buna göre **SOH** ve **LOH** bölgeleri n sayıda fiziki işlemci tarafından ele alınabilir*(Tabi birbirlerini kesmeyecek şekilde)*

Bu söylenenlere göre Workstation' ların da Server Mode' da çalıştırılması düşünülebilir. Ancak bu modda kullanıcı cevap verilebilirliği ikinci plandadır. Çünkü tüm uygulama Thread' leri, GC' nin çalıştığı sürelerde Suspend moda geçecektir ki bu kullanıcının direk uygulama ile olan etkileşiminde eksi puandır.
Bir de GC' nin tekrardan toplamasına gerek duyulmayacak şekilde kullanılabilen **Weak Referance**tipleri bulunmaktadır. **WeakReference** sınıfı bu noktada devreye girmektedir. Bu konuyu ilerleyen zamanlarda incelemeyi ve sizlerle paylaşmayı planlıyorum.
Yukarıda bahsettiklerimiz anlamında **GC'** i kod tarafında ve **config** dosyası üzerinde ayarlayabilir ve hangi modda ne şekilde çalışacağına karar verebiliriz. Örneğin uygulamaya ait **config** dosyasına yapılacak aşağıdaki bildirim ile **Server Mod** üzerinde çalışılacağı ifade edilir.

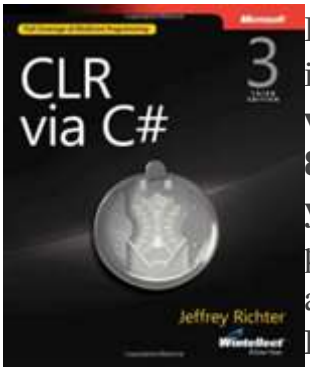
```
<configuration>
  <runtime>
    <gcServer enabled="true" />
  </runtime>
</configuration>
```

veya örneğin

```
<configuration>
  <runtime>
    <gcServer enabled="false"/>
    <gcConcurrent enabled="false"/>
  </runtime>
</configuration>
```

bildirimi ile **Workstation** modda ve **non-concurrent** olarak çalışılacağı ifade edilebilir. Bunlara ek olarak **GCLatencyMode** adında önemli bir ayar daha sunulmaktadır. Bu ayara göre daha fazla nesnenin bellekten toplanması bir başka deyişle çok daha fazla alanın açılması sağlanabilir. Bilindiği üzere **GC**, çöpleri topladığı sırada çalışmakta olan diğer tüm **Thread'** leri geçici olarak durdurur. Bu nedenle **Latency(gecikme)** ' nin kontrol altına alınması gereken uygulamalar söz konusu olabilir.
GCLatencyMode özelliği **Batch, Interactive** ve **LowLatency** olmak üzere 3 sabit değerden birisini alabilir. **Batch** mode genellikle bir arayüzü veya sunucu taraflı operasyonu olmayan uygulamalarda tercih edilmektedir. Arayüzü(UI) bulunan uygulamalarda **Interactive mode** seçilebilir. Bunun dışında bazı uygulamalar bilindiği üzere bellek üzerinde çok daha fazla harekette bulunurlar. Özellikle çok kısa sürelerde işlemlerin yapılması gerekmektedir. Örneğin animasyon işlemleri ile uğraşan uygulamalar buna örnek olarak verilebilir. Bu tip uygulamalarda zaman oldukça önemlidir. O nedenle**LowLatency** modda çalıştırılmaları sağlanabilir. Latency Mode ile ilişkili daha detaylı bilgiyi MSDN adresinden bulabilirsiniz.
Devam eden yazımızda **LOH** ve **SOH** kullanımları sırasında uygulamalarımıza ait bellek değerlerini nasıl ölçümlendirebileceğimizi aktarmaya çalışıyor olacağım. Şimdilik teoriyi

pekiştirmemizde ve neyin ne olduğunu ayrıştırmamızda yarar var. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.



Bu arada **CLR'** ın çalışma şeklini daha iyi ve derinlemesine öğrenmek isterseniz sizlere tavsiyem **MS Press'** in **Jeffrey Ritcher** imzalı **CLR via C#** isimli kitabı olacaktır.

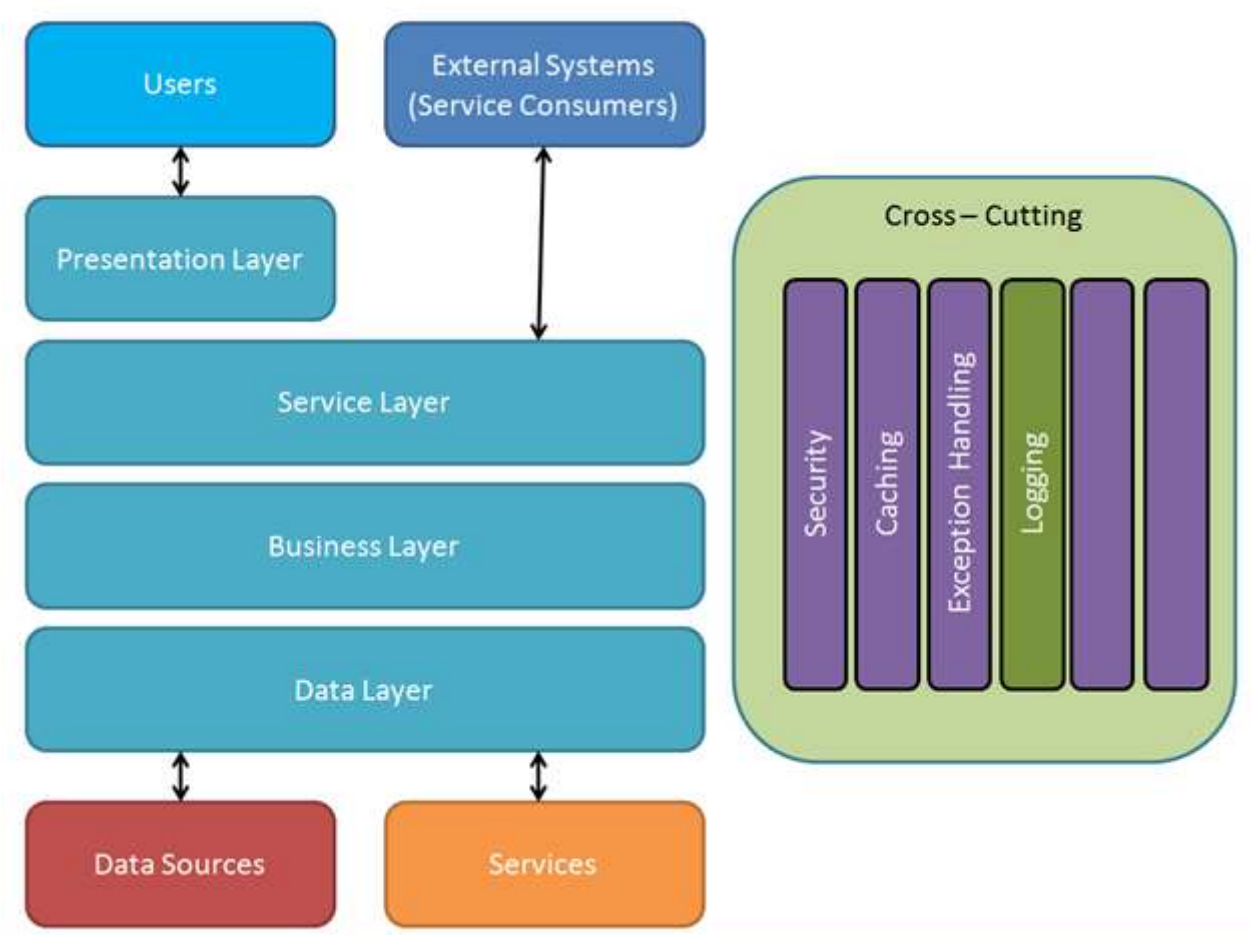
896 sayfalık bu kitap içerisinde elbetteki bulacağınız tek şek **bellek yönetimi(Memory Managemet)** değil. Ama yazdığımız temel C# kod parçalarının **CLR(Common Language Runtime)** tarafından nasıl ele alındığını görmek, **CIL(Common Intermediate Language)** seviyesine kadar inebilmek mümkün. Fiyatı biraz yüksek görünebilir ama bence elinizin altında olması gereken bir kaynaktır diye düşünüyor ve hatta bu konuda ısrar ediyorum.

Log4Net' i Tanıyalım

Perşembe, 1 Mart 2012 22:22

.Net, C#, Cross Cutting, Logging, Concern, Log4net, Nlog, Logging Application Block

Merhaba Arkadaşlar,
Herkesin kendine has bir parmak izi vardır. DNA gibi benzersizdir. Her ne kadar bazı ajanlı filmelerinde bu izler silinebilse de*(belki de silinebilyordur)* :) Krimonoloji labaratuvarlarından tutunda, şirketlerdeki giriş kapılarına kadar pek çok noktada parmak izlerimiz devreye girer. Hatta günümüzde kullandığımız bilgisayarların çoğunun açılması için parmak izi kullanılabilmektedir.
Parmak izinin sahibi, sistem içerisinde yaptığı hareketliliklere ait pek çok bilgi bırakır geriye. Ne zaman gelmiş, nerede durmuş, hangi eşyayı tutmuş, kaç saat çalışmış, bilgisayarını ne zaman açmış vs…Bu tip bilgiler bazı senaryolarda çok kritiktir ve önemli anlamlar ifade etmektedir.
Ancak izlenen sadece parmak izi sahip insanlar değildir. Zaman zaman sistemlerin ve onun parçası olan aktörlerin de*(cihazlar veritabanları, uygulamalar örneğin)* takip edilmesi ve toplanan bilgilerden yola çıkılarak, ya oluşan hataların giderilmesi ya da gelecek için gerekli yönün nasıl olacağına karar verilmesi aşamalarında da bilgi toplamak önem kazanır. Gelelim bu geniş evrenden bizim dünyamıza. Yani yazılıma.
Geliştirdiğimiz uygulama çözümlerde, programın herhangibir zaman diliminde neler yaptığının bilgisini tutmak, geriye dönük yapılan araştırmalarda, performans ölçümlerinde, bug' ların ayıklanmasında veya bir sonraki iterasyon için gerekli backlog' ların oluşturulmasında önemli bir unsurdur.
Benzer şekilde sistem içerisine dahil olan kullanıcıların veya sistemin parçası olan diğer aktörlerin, zaman dilimi içerisindeki hareketliliklerini de kayıt altına almamız önemlidir. Nitekim bu şekilde aktörlerin geriye dönük izlerine ulaşabilir, bazı yasal süreçlerdeki ispatlardan tutunda, ürünün sonraki versiyonları için gerekli kullanıcı deneyimi raporlarının çıkartılmasına kadar pek çok noktada faydalı verilere ulaşabiliriz. Burada loglama kavramı devreye girmekte olup bilginin toplanması noktasında önemli bir Concern olarak karşımıza çıkmaktadır.
Dolayısıyla loglama uygulamalar için hayati bir anlama sahiptir. Üstelik mimari açıdan baktığımızda, katmanlar arasında bir **kıstas(Concern)** olarak da göz çarpmaktadır. Söz gelimi çok basit anlamda**SOA(Service Oriented Architecture)** tabanlı bir mimari modeli düşündüğümüzde, pek çok katman arasında dikine ilişki taşıyan kıstaslardan birisi de Loglama parçasıdır ve Cross-Cutting düzleminde yer alan enstrümanlardan birisidir. Aşağıdaki şekilde bu durum özetlenmektedir.



Dikkat edileceği üzere **Sunum(Presentation), servis(Service), iş(Business)** ve **veri(Data)**katmanlarının tamamı, **Cross-Cutting** içerisinde yer alan bloklara erişebilmektedir. **Logging** bloğu dışında diğer **Concern'** lerde söz konusudur elbette. Örneğin güvenlik, performans odaklı Caching, hata yönetimi vb…
Gelelim Loglama konusuna.
Yazılım evrenimizde Loglama amacıyla kullanılan pek çok yardımcı araç da bulunmaktadır. Bunların çoğu ücretsiz birer kütüphanedir ve kullanımları da son derece basittir. Söz gelimi Microsoft**Enterprise Library** ile birlikte gelen **Caching Application Block, Log4Net, NLog** ve diğerleri. Loglama işlemleri için kullanılan bloklar, **Cross Cutting'** de yer alan diğerleri gibi **Inversion Of Control** prensibini başarılı bir şekilde uygulayan ve dolayısıyla **Dependency Injection** kavramını içeren yapılardır. Daha doğrusu bu şekilde tasarlanmaları çok daha doğrudur.
Genellikle loglama amacıyla kullanılan araçlar, loglama stratejilerini, uygulama yeniden derlenmeye gerek kalmadan çalışma zamanında değişiklikler yapılabilmesine olanak tanımaktadır. Burada çok doğal olarak uygulama harici **konfigurasyon** ayarlarının kullanımı söz konusudur. Özellikle **XML**tabanlı olarak tutulan ve bu sayede basit bir editör ile dahi düzenlenebilecek konfigurasyon dosyaları bulunmaktadır.
.Net Framwork tarafındaki masaüstü uygulamalar*(Console, Windows, WPF, Windows Service vb)* için**app.config** dosyası söz konusu iken, **Web** tabanlı uygulamalarda **Web.config** dosyası ön plana çıkmaktadır. Ancak çoğu loglama aracı, başka

bir **XML** konfigurasyon dosyasının da **App.config**veya **Web.config** içerisinden işaret edilmesine izin vermektedir.
Loglama stratejisinin konfigurasyon dosyaları yardımıyla kolayca belirlenmesi bu işin ilk önemli adımlarından birisidir. Genellikle bu noktadan sonra, loglamanın hangi kaynağa doğru yapılacağına dair bir takım kararlar verilmeli ve uygun sınıf oluşumları sağlanmalıdır. Loglama popüler olarak metin tabanlı dosyalara **düz yazı(Plaint Text – ki en az yer kaplayan ve performanslı olan çözümlerdendir)** veya **XML** gibi formatlarda aktarılabilir. İstenirse bir veritabanı üzerinde ilişkisel anlamda tablolara yazılması da sağlanabilir. Dosya ve veritabanı sistemlerine ek olarak servis veya**Windows Event** sistemi odaklı çözümler de düşünülebilir. Yani log bilgilerinin bir servis aracılığıyla farklı ve çoğunlukla arkasında nasıl bir sistem olduğunu bilmediğimiz ortamlara aktarılması ya da**Windows Event Log'** larda uygulama bazlı olacak şekilde ayrıştırılarak saklanması da söz konusudur.
Çok doğal olarak bu işleri üstlenmesi ve seçilen **provider'** lara göre ilgili ortamlara log bilgilerini çalışma zamanında göndermesi gereken bir takım akıllı tiplerin olması gerekmektedir. Dolayısıyla ilgili loglama araçlarının bu tip desteğini sunduğunu da ifade edebiliriz.
Konfigurasyon ile loglama stratejisini belirlemek, uygun tiplerin devreye girerek bir veya daha fazla ortama log mesajı atılabilmesine olanak sağlamaktadır. Bundan sonraki adımda ise geliştiricinin uygun tiplere ait nesne örneklerini oluşturarak, uygulamanın ilgili ve doğru yerlerinde log atması yeterli olacaktır.
Tabi unutulmaması gereken bir husus da, her yerde her şeyi ve her zaman için loglamamakgerekliliğidir. Nitekim her şeyli loglamak bir süre sonra bir veri çöplüğünün oluşmasına ve bilginin ayrımının zorlaşmasına neden olmaktadır. Ayrıca performans açısından adım başı loglamanın maliyeti düşünülmelidir. Bazen sadece test senaryolarının işletildiği sırada kullanılan logların ürünün**Production** ortamına çıkmasında kaldırılması veya kullanılmaması öngörülebilir. Kısacası loglama stratejimizi belirlerken, neyi, ne zaman, hangi vaka da, ne şekilde loglayacağımıza da karar vermek gerekebilir.
Şimdi lafı fazla uzatmadan makalemizin asıl konusuna gelelim. Bu yazımızda ücretsiz olarak indirip kullanabileceğimiz loglama araçlarından birisi olan **log4Net'** e değiniyor olacağız. Bu adresten ücretsiz olarak indirebileceğiniz ürün basit bir **dll** kütüphanesidir ve 3 önemli unsurun yerine getirilmesini beklemektedir. *(Ben makaleyi yazdığım tarih itibariyle ürünün 1.2.11.0 versiyonunu değerlendiriyor olacağım)*

Log4Net ürününü indirdiğimizde beraberinde oldukça büyük bir **XML** konfigurasyon dosyası da gelmektedir. Bu dosya içerisinde, yapılacak konfigurasyon ayarlarına ait tüm detaylı bilgiler yer alır.**Log4Net'** de diğer pek çok loglama aracında olduğu gibi, loglanacak bilgilnin seviyelendirilmesini önerir. Bu anlamda **Fatal, Error, Warn, Info, Debug, All** ve **Off** gibi çeşitli seviyeler sunar. Konfigurasyon dosyasını oluşturmak son derece kolaydır. Örneğin aşağıdaki konfigurasyon içeriğinde veritabanına log mesajı yazacak şekilde yapılmış ayarlamalar vardır.

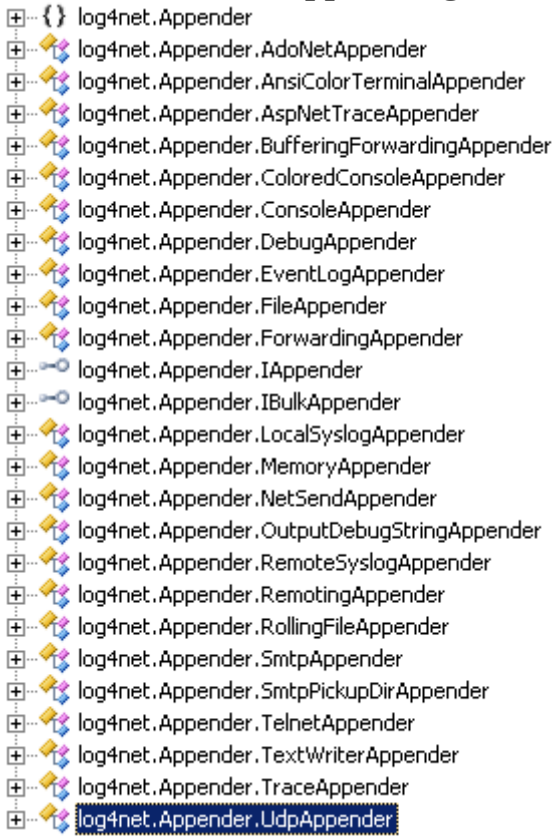
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="log4net"
type="log4net.Config.Log4NetConfigurationSectionHandler,log4net, Version=1.2.10.0,
Culture=neutral, PublicKeyToken=1b44e1d426115821" />
  </configSections>
  <log4net>
  <appender name="AdoNetAppender"
type="log4net.Appender.AdoNetAppender">
  <bufferSize value="100" />
  <connectionType value="System.Data.SqlClient.SqlConnection, System.Data,
Version=1.0.3300.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" />
  <connectionString value="data source=.;initial catalog=SpeedyShop;integrated
security=true;" />
  <commandText value="INSERT INTO ApplicationLog
([Date],[Thread],[Level],[Logger],[Message],[Exception]) VALUES (@log_date,
@thread, @log_level, @logger, @message, @exception)" />
  <parameter>
```

```
  <parameterName value="@log_date" />
  <dbType value="DateTime" />
  <layout type="log4net.Layout.RawTimeStampLayout" />
</parameter>
<parameter>
  <parameterName value="@thread" />
  <dbType value="String" />
  <size value="255" />
  <layout type="log4net.Layout.PatternLayout">
    <conversionPattern value="%thread" />
  </layout>
</parameter>
<parameter>
  <parameterName value="@log_level" />
  <dbType value="String" />
  <size value="50" />
  <layout type="log4net.Layout.PatternLayout">
    <conversionPattern value="%level" />
  </layout>
</parameter>
<parameter>
  <parameterName value="@logger" />
  <dbType value="String" />
  <size value="255" />
  <layout type="log4net.Layout.PatternLayout">
    <conversionPattern value="%logger" />
  </layout>
</parameter>
<parameter>
  <parameterName value="@message" />
  <dbType value="String" />
  <size value="4000" />
  <layout type="log4net.Layout.PatternLayout">
    <conversionPattern value="%message" />
  </layout>
</parameter>
<parameter>
  <parameterName value="@exception" />
  <dbType value="String" />
  <size value="2000" />
  <layout type="log4net.Layout.ExceptionLayout" />
```

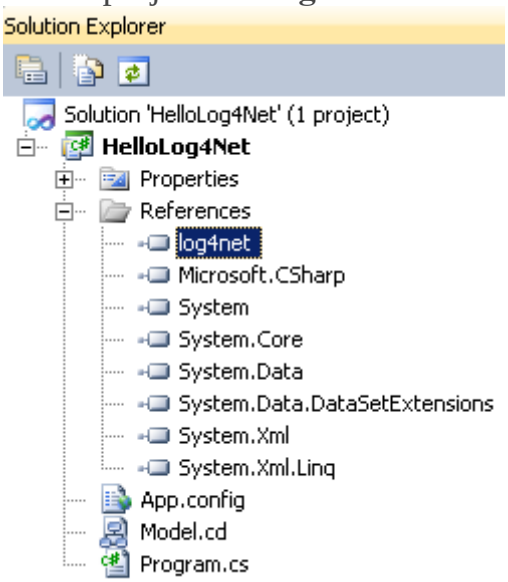
```
  </parameter>
</appender>
  <root>
    <level value="DEBUG"/>
    <appender-ref ref="AdoNetAppender"/>
  </root>
  </log4net>
</configuration>
```

Burada dikkat edileceği üzere **Appender** sekmesinde **AdoNetAdppender** isimli bir kısım tanımlanmıştır. Bu appender, **connectionString** bilgisine göre yerel sunucu üzerinde yer alan**SpeedyShop** isimli veritabanındaki **Log** tablosuna kayıt atacak şekilde tasarlanmıştır. Sorgu cümlesine dikkat edilecek olarak **@** harfi ile başlayan parametrelerinin değerleri **%** ile başlayan özel **keyword'** lerdir. Örneğin, **@log_level** için **%level**, uygulamanın **thread** bilgisine ait değer için**%thread** gibi özel **keyword'** ler kullanılmaktadır. **%** ile başlayan anahtar kelimeler **log4Net'** e özgü yapılmış tanımlamalardır. **%date, %level, %exception, %newline, %identity, %method** ve benzerleri gibi, uygulama ortamında otomatik olarak loglama katına veri taşıyan sabitlerde mevcuttur.

Yukarıdaki **config** dosyasında **Appender** isimli bir kavramdan bahsettik. Veritabanı odaklı çalışan dışında dosyaya yazma amacıyla kullanılabilen **FileAppender** veya özellikle **Console** tabanlı test projelerinde ekrana log bilgisi vermek için kullanılan **ConsoleAppender** gibi versiyonlar da mevcuttur.



Object Browser yardımıyla **Log4Net** kütüphanesine baktığımızda pek çok **Appender** tipinin tanımlanmış olduğunu görürüz. **Udp, SMTP, ASP.Net Trace** bazlı vb gibi pek çok **Appender**bulunmaktadır.
Peki bu tipleri nasıl devreye alacağız? Gelin basit bir **Console** uygulaması üzerinden ilerleyelim ve**Log4Net** için alışılageldiği üzere bir **HelloWorld** örneği geliştirelim. İlk olarak projemize **Log4NetAssembly'** ının referans edilmesi gerekmektedir.



İkinci olarak uygun bir loglama stratejisi belirlemeli ve **App.config** dosyasında buna ait ayarlamalar yapılabilmelidir. Bu örneğimizde loglarımızı dosya bazlı olarak tutuyor olacağız. Bu sebepten dolayı**FileAppender** tipini ele alacağız.
Dolayısıyla **app.config** dosyasının içeriğini aşağıdaki gibi geliştirebiliriz.

```
<?xml version="1.0"?>
<configuration>
  <configSections>
    <section name="log4net"
type="log4net.Config.Log4NetConfigurationSectionHandler,log4net, Version=1.2.10.0,
Culture=neutral, PublicKeyToken=1b44e1d426115821"/>
  </configSections>
  <log4net>
    <appender name="FileAppender" type="log4net.Appender.FileAppender">
      <file value="Logs.txt" />
      <appendToFile value="true" />
      <lockingModel type="log4net.Appender.FileAppender+MinimalLock" />
      <layout type="log4net.Layout.PatternLayout">
        <conversionpattern value="%date [%thread] %-5level – %message%newline"
/>
      </layout>
      <filter type="log4net.Filter.LevelRangeFilter">
        <levelMin value="INFO" />
```

```
      <levelMax value="FATAL" />
     </filter>
   </appender>
   <root>
   <level value="DEBUG"/>
   <appender-ref ref="FileAppender"/>
  </root>
  </log4net>
<startup>
  <supportedRuntime version="v4.0" sku=".NETFramework,Version=v4.0"/>
</startup>
</configuration>
```

Loglanacak metinsel içerik **layout** boğumu içerisinde yer alan **conversionPattern** elementine ait**value** niteliğinde belirtilmektedir. Önce tarih bilgisi ve arkasından sırasıyla **thread**, **log seviyesi**, **mesaj** ve **alt satıra** geçme işlemleri uygulanmaktadır. Loglamayı yaparken minimum seviyemiz**INFO** dur. Maksimum ise **FATAL** olarak set edilmiştir. Bu basit ayarlamalardan sonra uygulamanın kod tarafına geçerek gerekli **Setup** işlemlerini yazabiliriz. İşte kodumuz.

```
using System;
using System.Data;
using System.Data.SqlClient;
using log4net;
namespace HelloLog4Net
{
   class Program
   {
      static ILog log = log4net.LogManager.GetLogger(typeof(Program));
      static void Main(string[] args)
      {
         log4net.Config.BasicConfigurator.Configure();
         SqlConnection connection = null;
         try
         {
            const string connectionString = @"data
source=.\SQLEXPRESS;database=AdventureWorks;user id=sa;pwd=1234.";
            log.Warn(String.Format("Bağlantı açılacak. Connection String
:{0}",connectionString));
            connection=new SqlConnection(connectionString);
            if(connection.State==ConnectionState.Closed)
               connection.Open();
```

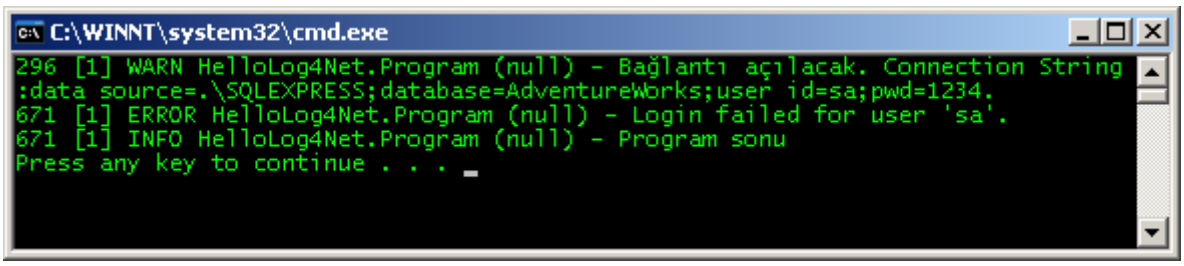
```
            log.Info(String.Format("Bağlantı durumu : {0}",connection.State));
        }
        catch (Exception excp)
        {
            log.Error(excp.Message);
        }
        finally
        {
            if (connection != null && connection.State == ConnectionState.Open)
            {
                connection.Close();
                log.Debug(String.Format("Finally bloğundayız. Bağlantı durumu {0}",
connection.State));
            }
        }
        log.Info("Program sonu");
    }
  }
}
```

Dikkat edileceği üzere loglama nesne örneği
üretilirken **LogManager** tipinin **GetLogger** metodu kullanılmakta ve sonuç **ILog** arayüzü referansına taşınmaktadır. İşte size **IoC örneği** :) Kodun ilerleyen kısımlarında konfigurasyon ayarlarının okunması işlemi gerçekleştirilir ve sonrasında istenilen noktalardan çeşitli seviyelerde log mesajları, konfigurasyon dosyasında belirtildiği üzere ilgili dosyaya yazdırılır. Son olarak **AssemblyInfo** dosyasında yapmamız gereken basit bir nitelik**(Attribute)** bildirimi daha söz konusudur.

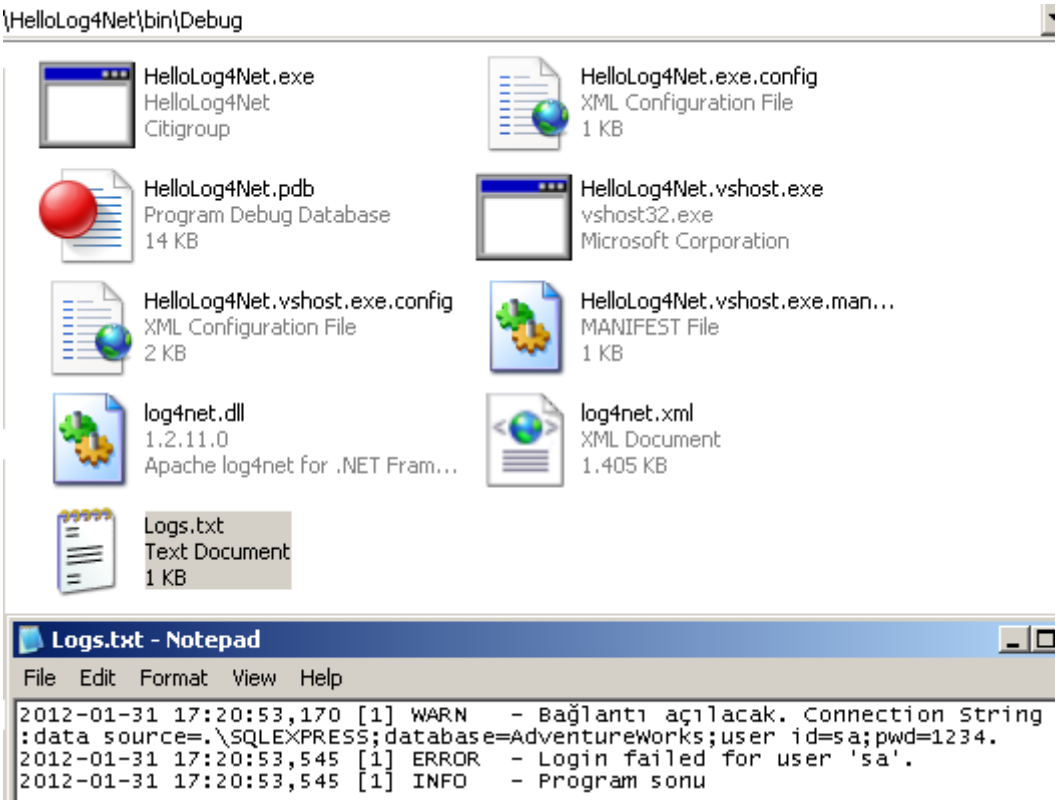
…

```
[assembly: AssemblyVersion("1.0.0.0")]
[assembly: AssemblyFileVersion("1.0.0.0")]
[assembly: log4net.Config.XmlConfigurator(Watch = true)]
```

Uygulamayı çalıştırdığımızda aynı zamanda bir ekran çıktısı aldığımızı da görürüz. Varsayılan olarak**ConsoleAppender'** da devrededir. Tabiki bunu kapatabilirsiniz.



Test dosyasının içeriği de aşağıdaki gibi olacaktır.



Görüldüğü üzere **log4Net** kullanılarak çok basit bir şekilde loglama işlemleri yapılabilmekte ve uygulamanın herhangibir noktasından **feedback'** ler verilebilmektedir. Siz siz olun uygulamalarınızda loglama konusunu ihmal etmeyin. Nitekim hataları ayıklama, kullanıcı ve sistem gibi genel aktörlerin hareketliliklerini izleme ve backlog oluşturma noktasında hayati öneme sahip bir mevzudur. İlerleyen makalelerimizde Log4Net' in farklı kullanımlarına da değinmeye çalışıyor olacağız. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

HelloLog4Net.zip (2,28 mb)

Tek Fotoluk İpucu–47 (Mime Type)

Çarşamba, 15 Şubat 2012 10:11

Mime Type, Response.Contenttype, Asp.Net

Hani olurda web tarafında istemciden gelen talebe göre, döndürmek istediğiniz herhangibir tipteki dosyanın Mime Type değerine göre bir Content Type üretmek istersiniz ya İşte tam bu isteklik bir ipucu. Lazım olmaz demeyin. Bana lazım oldu

ogram.cs

ExtensionMethodSamples.Program

```
using System;
using System.IO;
using Microsoft.Win32;

namespace ExtensionMethodSamples
{
    class Program
    {
        static void Main(string[] args)
        {
            string[] sampleFiles = { "c:\\winter.jpg"
                                    ,"c:\\Muhasebe.xlsx"
                                    ,"c:\\Receteler.txt"
                                    ,"c:\\Proje Dokumani.pdf"
                                    ,"c:\\rehber.xls"
                                   };
            foreach (string sampleFile in sampleFiles)
                Console.WriteLine("{0} : Mime/Content Type : {1}\n"
                    ,sampleFile,sampleFile.GetMimeType());
        }
    }

    public static class FileExtensions
    {
        public static string GetMimeType(this string FilePath)
        {
            if (!File.Exists(FilePath))
                throw new FileNotFoundException();

            string mimeType = "application/octetstream";
            RegistryKey registeryKey =
                Registry.ClassesRoot.OpenSubKey(Path.GetExtension(FilePath).ToLower());
            if (registeryKey != null && registeryKey.GetValue("Content Type") != null)
                mimeType = registeryKey.GetValue("Content Type").ToString();

            return mimeType;
        }
    }
}
```

```
C:\WINNT\system32\cmd.exe
c:\winter.jpg : Mime/Content Type : image/jpeg

c:\Muhasebe.xlsx : Mime/Content Type : application/vnd.openxmlformats-officedocu
ment.spreadsheetml.sheet

c:\Receteler.txt : Mime/Content Type : text/plain

c:\Proje Dokumani.pdf : Mime/Content Type : application/pdf

c:\rehber.xls : Mime/Content Type : application/vnd.ms-excel

Press any key to continue . . .
```

Bing Maps WCF Servisleri

Pazartesi, 13 Şubat 2012 10:05

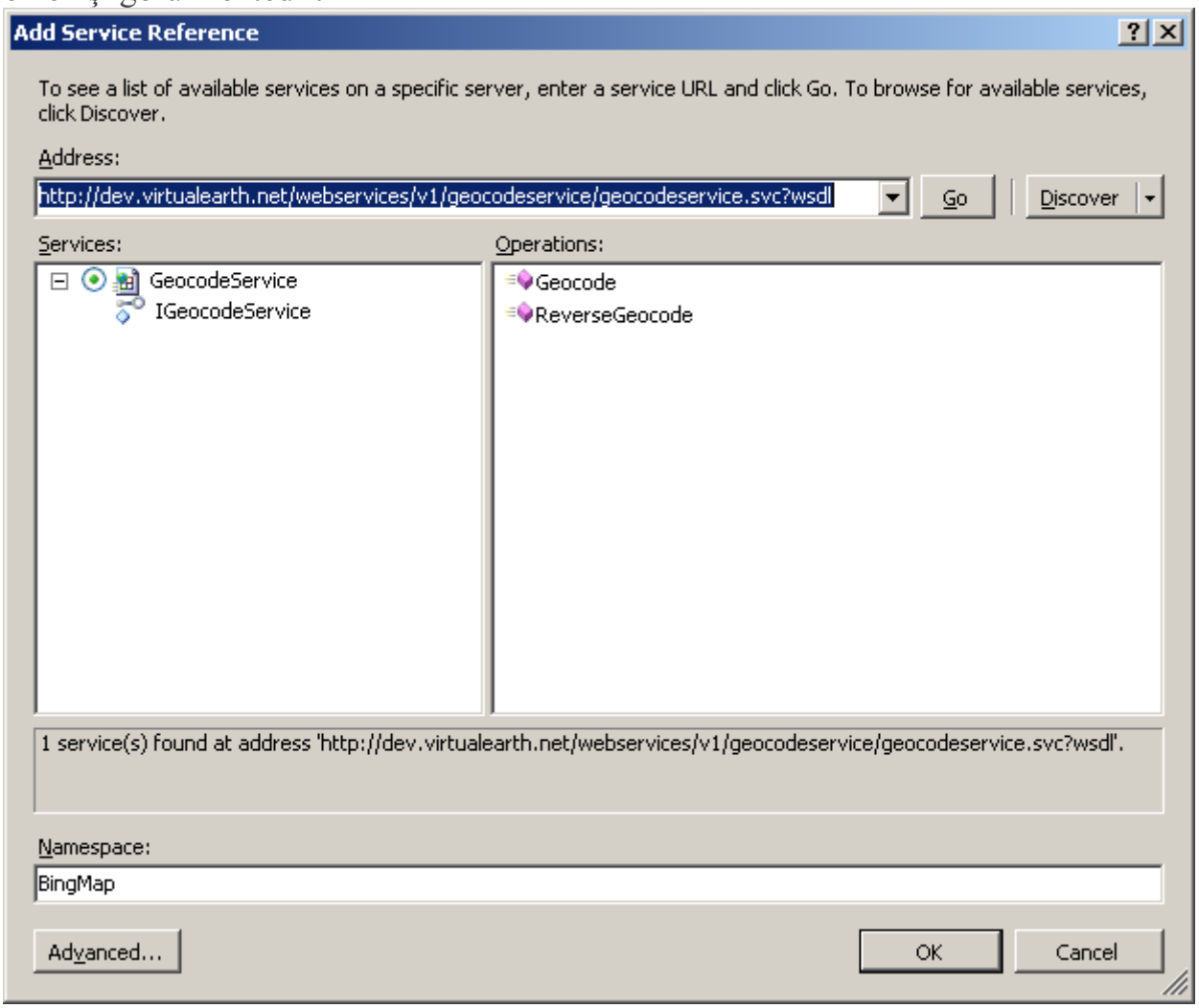
Bings, Bing Maps, C#, Wcf, Msdn, Search, Geolocation, Geocode, Search, Imagery,Route

Merhaba Arkadaşlar,
Eğitimci olarak çalıştığım dönemlerde servis bazlı mimarilerde, **XML Web Service**' leri son derece popüler bir kavramdı. Pek tabi öğrencilerimin çoğu, bu servislerin gerçek hayat örneklerini merak ederdi. Haklı olarak söz konusu yapının, nasıl çalıştığını anlamanın en iyi yolu onu saha da görmekti.
O dönemlerde internet üzerinden yayınlanan bazı ücretsiz servislerden yararlanarak konuyu pekiştirmeye çalışırdım. Klasik hava durumu veya finans servisleri bu anlamda çok işe yaramaktaydı.
Tabi zaman ilerledi ve bildiğiniz üzere **Microsoft,** servis odaklı geliştirme dünyasına yeni bir kavram getirdi. **WCF(Windows Communication Foundation)** Şimdi eskisi kadar çok sık olmasa da arada sırada eğitim veriyorum ve özellikle **WCF** konusuna sıra geldiğinde, öğrencilerime verdiğim gerçek hayat örnekleri arasında **Bing Maps'** in ücretsiz servisleri de yer alıyor. Bu yazımızda söz konusu servislerden bazılarını nasıl kullanacağımızı, basit fonksiyonlar üzerinden görmeye çalışıyor olacağız.
Microsoft Bing Map' in geliştiricilere sunduğu 4 önemli servis bulunmaktadır. Bu servislere ait adresleri aşağıdaki tabloda bulabilirsiniz.

Servis	**Adres**
Geocode	http://dev.virtualearth.net/webservices/v1/geocodeservice/geocodeservice.svc?wsdl
Search	http://dev.virtualearth.net/webservices/v1/searchservice/searchservice.svc?wsdl
Imagery	http://dev.virtualearth.net/webservices/v1/imageryservice/imageryservice.svc?wsdl
Route	http://dev.virtualearth.net/webservices/v1/routeservice/routeservice.svc?wsdl

Dikkat edileceği üzere söz konusu servislerin tamamı **WCF(Windows Communication Foundation)** tabanlı olarak geliştirilmişlerdir*(svc uzantısına dikkat)*

Dilerseniz ilk olarak **Geocode** servisini kullanarak, bir lokasyonun **Latitude** ve **Longtitude**değerlerini elde etmeye çalışarak işe başlayalım. Örnek fonksiyonelliklerimizi bir **Class Library**içerisinde toplayabilir ve her bir metodumuz için birer **Unit Test** geliştirerek ilerleyebiliriz. Servisleri projeye teker teker referans etmemiz gerektiğini hatırlatmak isterim. Aşağıdaki şekilde **Geocode**Servisinin eklenişi görülmektedir.



Diğer servisleri de projeye referans ettiğimizde **Config** dosyasında aşağıdaki içeriğin üretildiğine şahit oluruz.

CommonOperationsTest.cs app.config CommonOperations.cs

```
sion="1.0" encoding="utf-8" ?>
ation>
tem.serviceModel>
bindings>
    <basicHttpBinding>...</basicHttpBinding>
/bindings>
client>
    <endpoint address="http://dev.virtualearth.net/webservices/v1/geocodeservice/GeocodeService.svc"
        binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_IGeocodeService"
        contract="GeoCode.IGeocodeService" name="BasicHttpBinding_IGeocodeService" />
    <endpoint address="http://dev.virtualearth.net/webservices/v1/searchservice/searchservice.svc"
        binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_ISearchService"
        contract="Search.ISearchService" name="BasicHttpBinding_ISearchService" />
    <endpoint address="http://dev.virtualearth.net/webservices/v1/imageryservice/imageryservice.svc"
        binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_IImageryService"
        contract="Imager.IImageryService" name="BasicHttpBinding_IImageryService" />
    <endpoint address="http://dev.virtualearth.net/webservices/v1/routeservice/routeservice.svc"
        binding="basicHttpBinding" bindingConfiguration="BasicHttpBinding_IRouteService"
        contract="Route.IRouteService" name="BasicHttpBinding_IRouteService" />
/client>
stem.serviceModel>
uration>
```

Söz konusu servisler için **BasicHttpBinding** bağlayıcı tipini*(Binding Type)* kullanan birer **EndPoint**teklendiği görülmektedir. Tüm servislere **SOAP(Simple Object Access Protocol)** bazlı erişmek ve iletişime geçmek mümkündür. Dolayısıyla **.Net** dışı uygulamaların da ilgili servislerden yararlanabileceği düşünülebilir.

Şimdi ilk fonksiyonelliğimizi yazalım. Bu operasyon ile girilen bir lokasyonun*(örneğin bir şehrin)*detay bilgisini almaya çalışıyor olacağız. İşte fonksiyonumuz.

```
using BingMaps.Common.GeoCode;
namespace BingMaps.Common
{
   public static class CommonOperations
   {
      private const string appKey = "Sizin Maps Developer Account Key değeriniz
olmalı";
      public static GeocodeResponse GetLocationPoints(string city)
      {
         GeocodeResponse response = null;
         if (string.IsNullOrEmpty(city))
            return response;
         GeocodeRequest geocodeRequest = new GeocodeRequest();
         geocodeRequest.Credentials = new Credentials();
         geocodeRequest.Credentials.ApplicationId = appKey;
         geocodeRequest.Query = city;
```

```
            ConfidenceFilter[] filters = new ConfidenceFilter[1];
            filters[0] = new ConfidenceFilter();
            filters[0].MinimumConfidence = Confidence.High;
            GeocodeOptions geocodeOptions = new GeocodeOptions();
            geocodeOptions.Filters = filters;
            geocodeRequest.Options = geocodeOptions;
            GeocodeServiceClient geocodeService = new GeocodeServiceClient();
            response = geocodeService.Geocode(geocodeRequest);
            return response;
        }
    }
}
```

Geocode servisinden yararlanmak son derece basittir. İlk olarak bir **request** hazırlanır ve aranan içerik **Query** özelliği ile değerlendirilir. Yapılacak olan taleple ilişkili olarak bazı filtreleme seçenekleri ayarlanır ve sonrasında servis üzerinden oluşturulan **request** nesnesi gönderilir. Servise,**Geocode** metodu üzerinden yapılan çağrı **GeocodeResponse** tipinden bir referans örneği döndürecektir. Pek tabi aranan içeriğe göre dünya üzerinde birden fazla lokasyon noktası önerilebilir. Bu sebepten dolayı **GeocodeResponse** referansı kendi içerisinde dizi bazlı olaraktan bir **Result**setini saklar. Bu set içerisinde, aranan lokasyon için olabilecek tüm önerilere yer verilir.

Kodda dikkat edilmesi gereken önemli noktalardan birisi de **Maps Development Account'** umuza ait bir **Application Key** değeri kullanıyor olmasıdır. Bu değer servis ile olan iletişimimiz sırasındaki**Credential** bilgisi için gereklidir.
Key ve Bing Maps Development için tüm bilgilere bu adresten ulaşabilirsiniz. Uygulama anahtarının oluşturulması son derece basittir. SSO(Single Sign On) un bir velinimeti olarak Microsoft' un bu servisinden, var olan Windows Live ID' miz ile hizmet alabiliriz. Söz konusu key değeri diğer BING servisleri için de gereklidir.

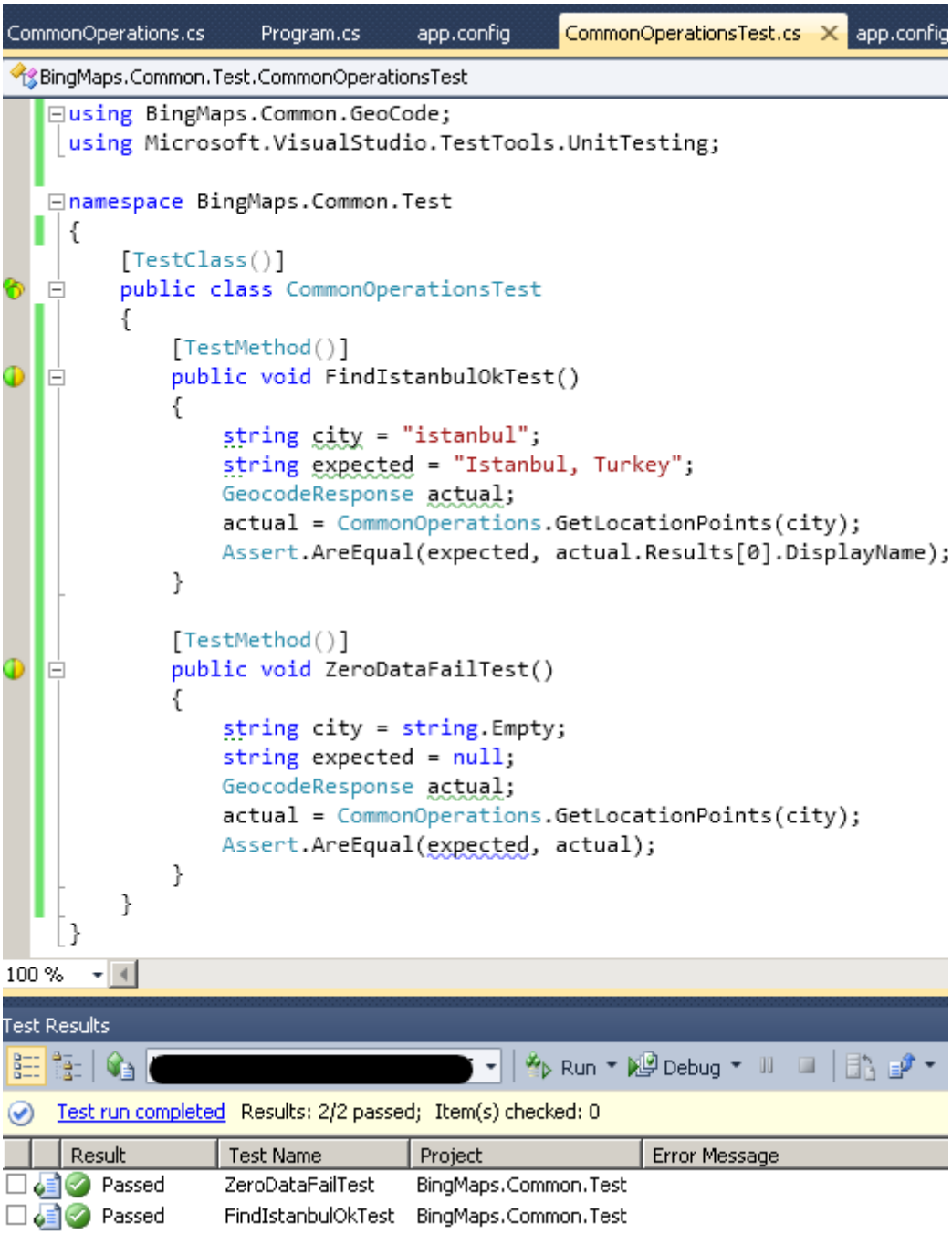
Şimdi yazmış olduğumuz bu metodu bir **Unit Test** fonksiyonu ile test edelim. Bu amaçla**BingMaps.Common.Test** isimli bir test projesini göz önüne alabiliriz. Başlangıçta yapacağımız test**istanbul** şehrini birebir bulmak ile alakalı olacaktır. Buna göre test sınıfı kodlarını aşağıdaki gibi geliştirebiliriz.

```
using BingMaps.Common.GeoCode;
using Microsoft.VisualStudio.TestTools.UnitTesting;
namespace BingMaps.Common.Test
{
    [TestClass()]
    public class CommonOperationsTest
```

```
    {
        [TestMethod()]
        public void FindIstanbulOkTest()
        {
            string city = "istanbul";
            string expected = "Istanbul, Turkey";
            GeocodeResponse actual;
            actual = CommonOperations.GetLocationPoints(city);
            Assert.AreEqual(expected, actual.Results[0].DisplayName);
        }
        [TestMethod()]
        public void ZeroDataFailTest()
        {
            string city = string.Empty;
            string expected = null;
            GeocodeResponse actual;
            actual = CommonOperations.GetLocationPoints(city);
            Assert.AreEqual(expected, actual);
        }
    }
}
```

Testleri çalıştırmadan önce dikkat edilmesi gereken önemli noktalardan birisi de, servisler için**Library** tarafında üretilen **App.config** dosyasına ait **system.serviceModel** içeriğinin, **Runtime**uygulaması içerisinde de yer alması gerekliliğidir. Bir başka deyişle, **BingMaps.Common assembly'** ı için üretilen **App.config** içeriğinin **Test** projesinde de yer alıyor olması gerekmektedir.

FindIstanbulIsOkTest metodumuzda elde edilen **GeocodeResponse** içerisindeki **Results**dizisine gidilmekte ve ilk nesne örneğinin **DisplayName** parametresinin **"Istanbul, Turkey"** olup olmadığına bakılmaktadır. Nitekim istanbul için yapılan arama sonucu bu şekilde dönecektir. Testi Pass edebildiysek, herşeyin yolunda olduğunu düşünebiliriz.
Diğer test metodumuz ise boş veri gönderdiğimiz vakayı ele almaktadır. Normal şartlarda boş içerik gönderdiğimizde servis tarafından bize bir **FaultException** dönmektedir. Bu nedenle, metoda gelen parametre değerinin boş veya null olması durumu tedbir olarak kontrol edilmiş ve ilgili fonksiyonelliğin **GeocodeResponse** değeri olarak **null** döndürmesi garanti edilmiştir.

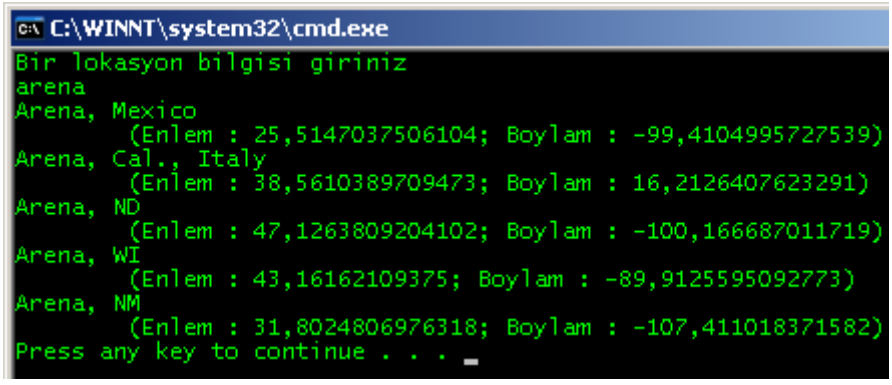


GeocodeResponse nesne örneği üzerinden pek çok değere ulaşılabilmektedir. Söz gelimi aramaya konu olan lokasyon için elde edilecek her önerinin üzerinden ilgili **Latitude** ve **Altitude** değerlerine ulaşılabilir. **GetLocationPoints** metodumuzun çalıştığını gördüğümüz için örnek bir **Console**uygulamasında bahsettiğimiz senaryoyu değerlendirebiliriz. İşte örnek kod içeriğimiz.

```
using System;
using BingMaps.Common;
using BingMaps.Common.GeoCode;
```

```
namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Bir lokasyon bilgisi giriniz");
         GetValue(Console.ReadLine());
      }
      private static void GetValue(string location)
      {
         GeocodeResponse response =
CommonOperations.GetLocationPoints(location);
         foreach (GeocodeResult r in response.Results)
         {
            Console.WriteLine("{0}", r.DisplayName);
            foreach (GeocodeLocation l in r.Locations)
            {
               Console.WriteLine("\t(Enlem : {0}; Boylam : {1})"
                        , l.Latitude
                        , l.Longitude
                  );
            }
         }
      }
   }
}
```

Burada elde edilen **GeocodeResponse** nesne örneğinin **Results** dizisinde dolaşılmakta ve içerisinde önerilen ne kadar **GeocodeLocation** bilgisi varsa, her biri için enlem ve boylam bilgileri ekrana yazdırılmaktadır. Örneğin **arena** yazdığımızda aşağıdaki sonuçları elde ederiz.



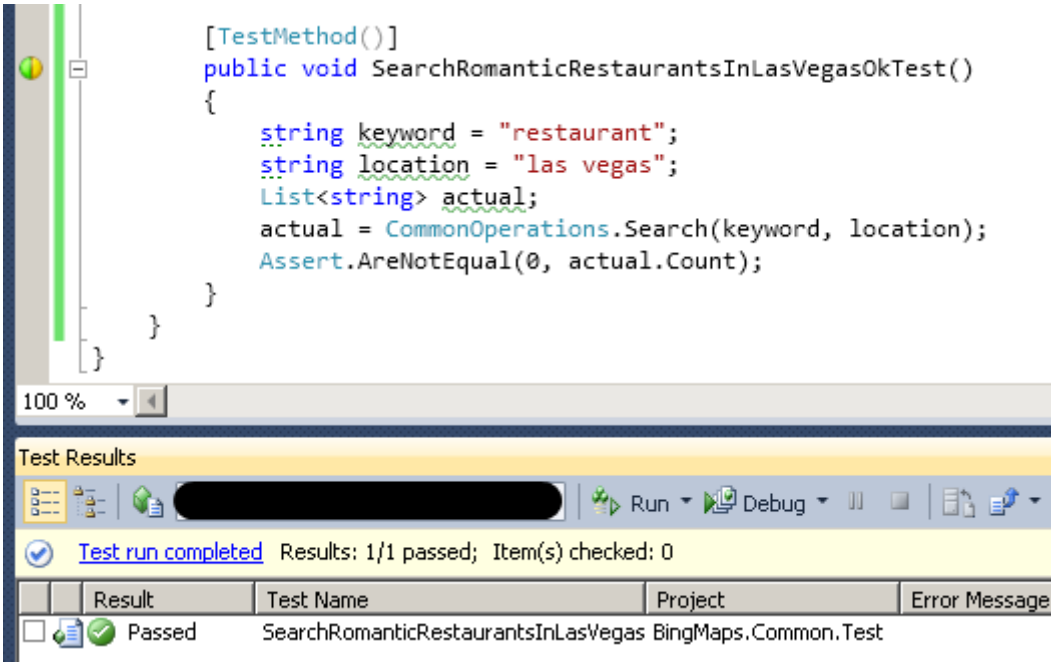
Bunlar **Bing Maps** servisinden bize dönen ve **Arena** ismi ile ilişkili olan lokasyonlar ve onlara ait enlem boylam bilgileridir.

Şimdi diğer servislerden **Search** hizmetine ait bir örnek metodu daha **BingMaps.Common** isimli kütüphanemize ekleyelim.

```
public static List<string> Search(string keyword,string location)
{
   List<String> results = new List<string>();
   if(String.IsNullOrEmpty(keyword)||string.IsNullOrEmpty(location))
      return results;
   Src.SearchRequest request = new Src.SearchRequest();
   request.Culture = "en-US";
   request.Credentials = new Src.Credentials();
   request.Credentials.ApplicationId = appKey;
   Src.StructuredSearchQuery query = new Src.StructuredSearchQuery();
   query.Keyword = keyword;
   query.Location = location;
   request.StructuredQuery = query;
   Src.FilterExpression atmosphereFilterExpression = new Src.FilterExpression();
   atmosphereFilterExpression.PropertyId = 23; // Atmosphere Property ID değeri 23
   atmosphereFilterExpression.FilterValue = 10; // 10 = Romantic bir ortam arıyoruz
   atmosphereFilterExpression.CompareOperator = Src.CompareOperator.Equals;
   Src.FilterExpression ratingExpression=new Src.FilterExpression()
            {
               PropertyId = 3, //Rating property ID
               CompareOperator =
Src.CompareOperator.GreaterThanOrEquals, //Rating değerine göre 3 ve üstünde olan
yerler
               FilterValue = 7 // Kullanıcıların verdiği rating değeri
            };
   Src.FilterExpressionClause combinedFilterExpressionClause = new
Src.FilterExpressionClause();
   combinedFilterExpressionClause.Expressions =
     new Src.FilterExpressionBase[]
     {
        ratingExpression,
        atmosphereFilterExpression
     };
   combinedFilterExpressionClause.LogicalOperator = Src.LogicalOperator.And;
```

```
    request.SearchOptions = new Src.SearchOptions();
    request.SearchOptions.Filters = combinedFilterExpressionClause;
    Src.SearchServiceClient searchService = new Src.SearchServiceClient();
    Src.SearchResponse response = searchService.Search(request);
    if (response.ResultSets[0].Results.Length > 0)
    {
        for (int i = 0; i < response.ResultSets[0].Results.Length; i++)
        {
            results.Add(String.Format("{0}\n({1};{2})\n",
                response.ResultSets[0].Results[i].Name,
                response.ResultSets[0].Results[i].LocationData.Locations[0].Latitude,
                response.ResultSets[0].Results[i].LocationData.Locations[0].Longitude
                )
                );
        }
    }
    return results;
}
```

Search servisi yardımıyla lokasyon bazlı olarak **"ne, nerede?"** tadında sorgulamalar yapabiliriz. Örneğin istanbul da yer alan kullanıcı tarafından 7 ve üzerinde değerlendirilmiş olan romantik restoranların listesi veya berlindeki petrol istasyonlarının bilgileri vb…Biz de örnek metodumuzda kullanıcılar tarafından Rating olarak 7 ve üzerinde not almış, romantik bir atmosfere sahip olan ortamları aratıyor olacağız. Burada büyük ihtimalle restoran, bar, coffee gibi ortamlar söz konusu olacaktır. Aslında bu kadar specific bir metod geliştirmek niyetinde değildim ama iki farklı arama kriterini nasıl kombine edebileceğimizi de göstermek istedim. Metodumuzu test etmek için şöyle bir vaka göz önüne alabiliriz.
Las Vegas kentinde, insanlardan 7 ve üzerinde not almış, romantik ambiyansa sahip restoranları listeyelim.



Dikkat edilmesi gereken noktalardan birisi de şudur. **WCF** tabanlı bir servis söz konusu olduğundan, gelen cevaba ait veri içeriğinin standart **byte** boyutlarını aşması durumu oluşabilir. Bu durum çalışma zamanında bir **Exception** üretilmesine neden olacaktır.

System.ServiceModel.QuotaExceededException: The maximum message size quota for incoming messages (65536) has been exceeded. To increase the quota, use the MaxReceivedMessageSize property on the appropriate binding element.

Dolayısıyla bu duruma karşılık tedbir olarak **MaxReceivedMessageSize** ve **MaxBufferSize**değerlerini, **config** dosyasından eşit olacak şekilde arttırmak gerekebilir ki örneğimizde ben bu sorunu yaşadığım için ilgili Fix işlemini yapmış bulunmaktayım.

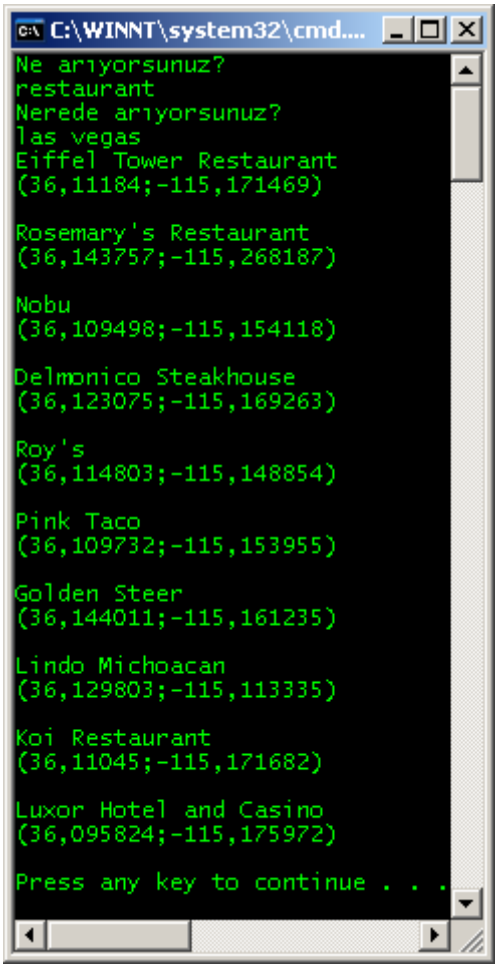
```
<binding name="BasicHttpBinding_ISearchService" closeTimeout="00:01:00"
    openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
    allowCookies="false" bypassProxyOnLocal="false" hostNameComparisonMode="StrongWildcard"
    maxBufferSize="655360" maxBufferPoolSize="524288" maxReceivedMessageSize="655360"
    messageEncoding="Text" textEncoding="utf-8" transferMode="Buffered"
    useDefaultWebProxy="true">
    <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
        maxBytesPerRead="4096" maxNameTableCharCount="16384" />
    <security mode="None">
        <transport clientCredentialType="None" proxyCredentialType="None"
            realm="" />
        <message clientCredentialType="UserName" algorithmSuite="Default" />
    </security>
</binding>
```

Tabi bir de bunu istanbul için denemek lazım. Bu kutsal görevi de siz değerli okurlarıma bırakıyorum 🙂

İlgili metodumuzu dilerseniz bir de Console uygulaması üzerinden kullanmaya çalışalım ve sonuçları görelim.

```
using System;
using BingMaps.Common;
using BingMaps.Common.GeoCode;
using System.Collections.Generic;
namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Ne arıyorsunuz?");
         string keyword = Console.ReadLine();
         Console.WriteLine("Nerede arıyorsunuz?");
         string location = Console.ReadLine();
         Search(keyword,location);
      }
      private static void Search(string keyword,string location)
      {
         List<string> results = CommonOperations.Search(keyword,location);
         foreach (string result in results)
         {
            Console.WriteLine(result);
         }
      }
   }
}
```

ve çalışma zamanı sonuçları



Şahsen **Eiffel Tower Restaurant'** ı oldukça merak ettim. Kısmet, nasip olur mu bilmem ama bu restoran hakkında detaylı bilgiye bu adresten ulaşabilirsiniz.
Search servisi çok fazla detay veri içermektedir. Normal şartlarda **maps.bing.com** adresinden yapabileceğimiz basit sorgulamaları, kod tarafında geliştirici olarak hazır etmek pek de kolay değil. Özellikle filtre nesne örnekleri hazırlanırken bu durum daha da açığa çıkmakta. Servisin etkili kullanımı hakkında daha detaylı bilgiyi MSDN adresinden edinebilirsiniz.

Makalemizde son olarak **Route** servisini kullanarak iki lokasyon arasındaki rota bazlı yol bilgisinin nasıl elde edilebileceğini göstermeye çalışıyor olacağım. Bu amaçla **CommonOperations** sınıfımıza aşağıdaki metodu ilave edelim.

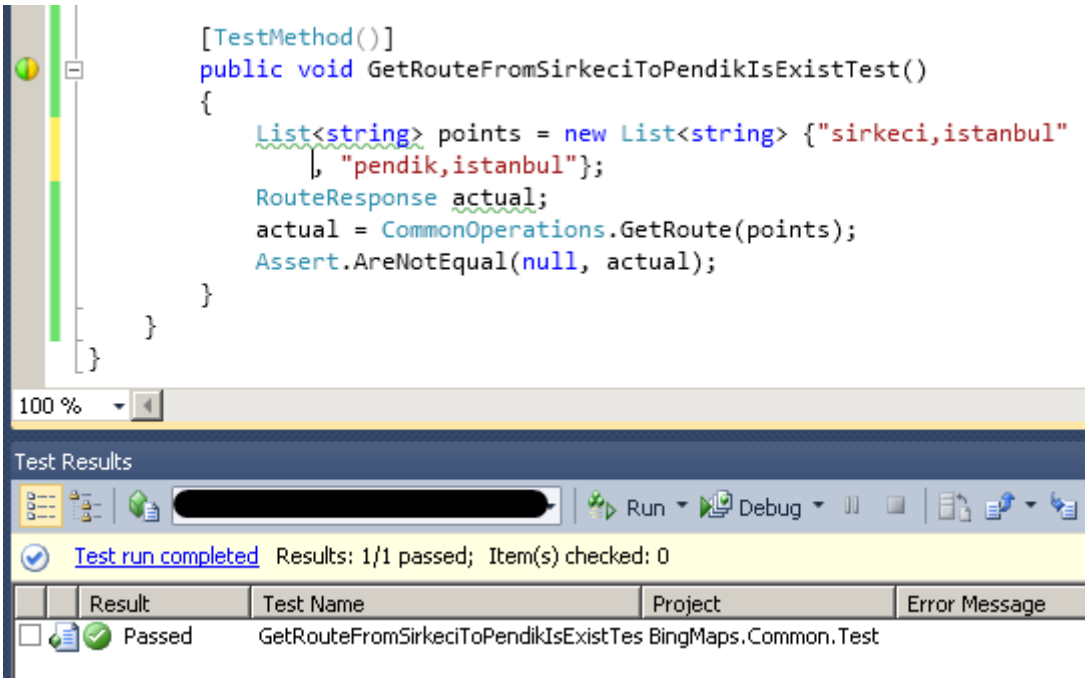
```
public static Route.RouteResponse GetRoute(List<string> points)
{
   if (points.Count < 2)
      return null;
   List<Route.Waypoint> waypoints = new List<Route.Waypoint>();
   Route.RouteRequest routeRequest = new RouteRequest();
```

```
    routeRequest.Credentials = new Route.Credentials();
    routeRequest.Credentials.ApplicationId = appKey;
    foreach (string point in points)
    {
        Geo.GeocodeResponse response=GetLocationPoints(point);
        if (response != null)
        {
            Waypoint wp = new Waypoint();
            wp.Location = new Route.Location();
            wp.Location.Latitude = response.Results[0].Locations[0].Latitude;
            wp.Location.Longitude = response.Results[0].Locations[0].Longitude;
            waypoints.Add(wp);
        }
        else
        {
            return null;
        }
    }
    routeRequest.Waypoints = waypoints.ToArray();
    Route.RouteServiceClient routeService = new Route.RouteServiceClient();
    Route.RouteResponse routeResponse =
routeService.CalculateRoute(routeRequest);
    return routeResponse;
}
```

Metodumuzda iki ve daha fazla nokta girilmesi halinde bunlar arasındaki yol tarifini bulacak şekilde bir geliştirme yapılmıştır. Elbette noktalar arasındaki yolu sağlıklı bir şekilde tespit edebilmek için enlem ve boylam değerlerine ihtiyaç vardır. İlgili lokasyonlara ait enlem ve boylam değerleri yaklaşık olarak **GetLocationPoints** metodu yardımıyla bulunabilir.

Şunu unutmayalım ki**GetLocationPoints** metodu geriye aranan kritere göre birden fazla lokasyon bilgisi döndürebilir. Biz her zaman için ilk değeri alıyoruz. İlaveten hiç bir lokasyon bilgisi dönmeyebilir de. Bu durumda zaten metod bir çalışma zamanı istisnasını **Locations[0].Latitude** değerini eklemeye çalıştığımız sırada verecektir*(Index was outside of the bounds)* Buna dikkat etmemiz gerekiyor. Rota tarifini bulabilmek için **RouteService'** inin **CalculateRoute** metodundan yararlanılmaktadır. Hemen test metodumuzu geliştirerek devam edelim. Örneğin,
Sirkeciden Pendiğe karayolu ile nasıl gidebiliriz? Yol tarifini bulmaya çalışalım.

Bu amaçla test metodunu aşağıdaki gibi geliştirebiliriz. Beklentimiz geriye **null** referans dönmemesi olacaktır.

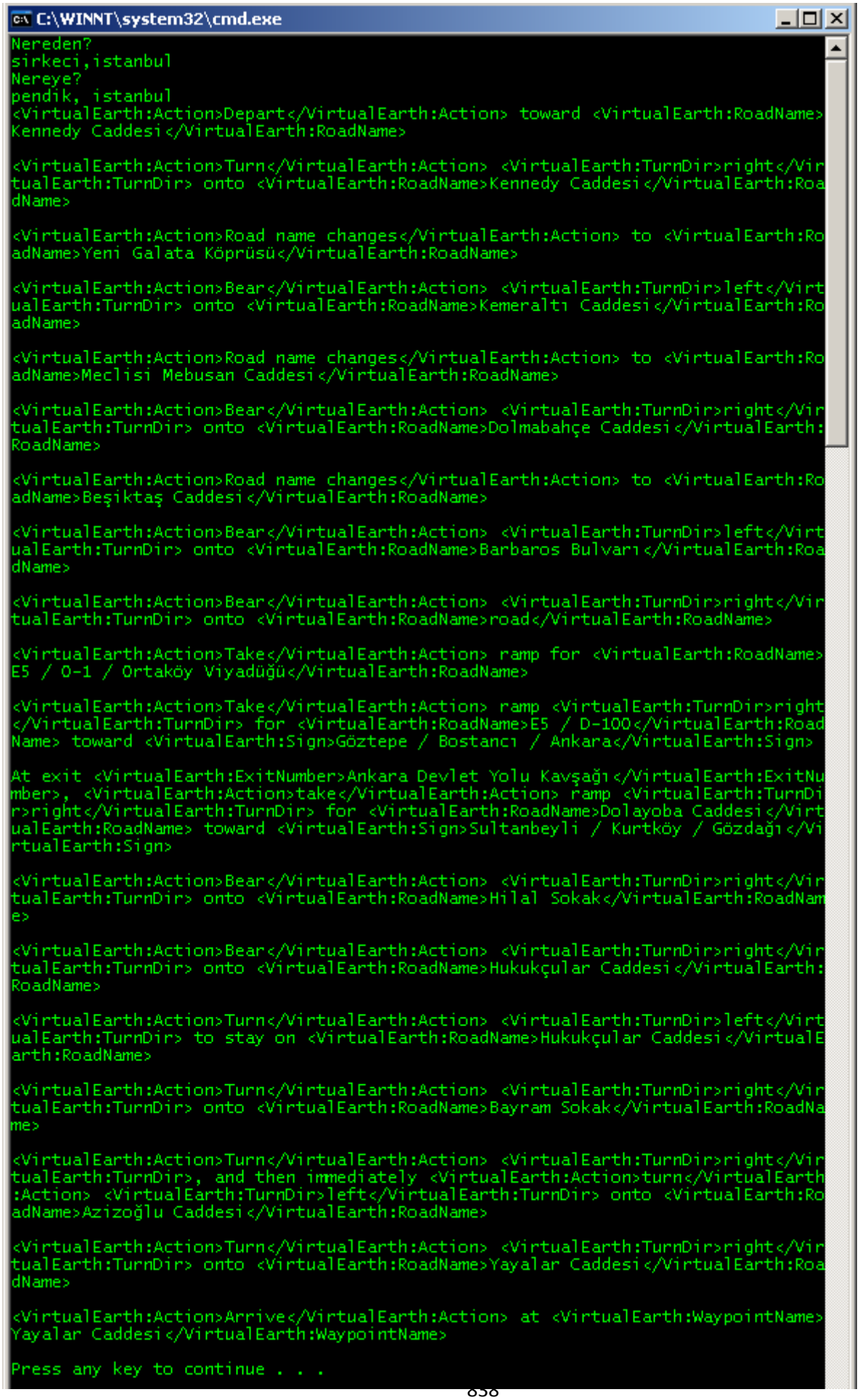


Peki dönüş tipini gerçek bir uygulamada nasıl kullanabiliriz? Bu amaçla Console projemizde aşağıdaki geliştirmeleri yaptığımızı düşünelim.

```
using System;
using BingMaps.Common;
using BingMaps.Common.GeoCode;
using System.Collections.Generic;
namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Nereden?");
         string nereden = Console.ReadLine();
         Console.WriteLine("Nereye?");
         string nereye = Console.ReadLine();
         GetValue(nereye, nereden);
      }
      private static void GetValue(string nereye, string nereden)
      {
```

```
        var result = CommonOperations.GetRoute(new List<string> {nereden,
nereye});
        foreach (var leg in result.Result.Legs)
        {
           foreach (var i in leg.Itinerary)
           {
              Console.WriteLine("{0}\n", i.Text);
           }
        }
     }
  }
}
```

ve sonuç


C:\WINNT\system32\cmd.exe
Nereden?
sirkeci,istanbul
Nereye?
pendik, istanbul
<VirtualEarth:Action>Depart</VirtualEarth:Action> toward <VirtualEarth:RoadName>Kennedy Caddesi</VirtualEarth:RoadName>
<VirtualEarth:Action>Turn</VirtualEarth:Action> <VirtualEarth:TurnDir>right</VirtualEarth:TurnDir> onto <VirtualEarth:RoadName>Kennedy Caddesi</VirtualEarth:RoadName>
<VirtualEarth:Action>Road name changes</VirtualEarth:Action> to <VirtualEarth:RoadName>Yeni Galata Köprüsü</VirtualEarth:RoadName>
<VirtualEarth:Action>Bear</VirtualEarth:Action> <VirtualEarth:TurnDir>left</VirtualEarth:TurnDir> onto <VirtualEarth:RoadName>Kemeraltı Caddesi</VirtualEarth:RoadName>
<VirtualEarth:Action>Road name changes</VirtualEarth:Action> to <VirtualEarth:RoadName>Meclisi Mebusan Caddesi</VirtualEarth:RoadName>
<VirtualEarth:Action>Bear</VirtualEarth:Action> <VirtualEarth:TurnDir>right</VirtualEarth:TurnDir> onto <VirtualEarth:RoadName>Dolmabahçe Caddesi</VirtualEarth:RoadName>
<VirtualEarth:Action>Road name changes</VirtualEarth:Action> to <VirtualEarth:RoadName>Beşiktaş Caddesi</VirtualEarth:RoadName>
<VirtualEarth:Action>Bear</VirtualEarth:Action> <VirtualEarth:TurnDir>left</VirtualEarth:TurnDir> onto <VirtualEarth:RoadName>Barbaros Bulvarı</VirtualEarth:RoadName>
<VirtualEarth:Action>Bear</VirtualEarth:Action> <VirtualEarth:TurnDir>right</VirtualEarth:TurnDir> onto <VirtualEarth:RoadName>road</VirtualEarth:RoadName>
<VirtualEarth:Action>Take</VirtualEarth:Action> ramp for <VirtualEarth:RoadName>E5 / O-1 / Ortaköy Viyadüğü</VirtualEarth:RoadName>
<VirtualEarth:Action>Take</VirtualEarth:Action> ramp <VirtualEarth:TurnDir>right</VirtualEarth:TurnDir> for <VirtualEarth:RoadName>E5 / D-100</VirtualEarth:RoadName> toward <VirtualEarth:Sign>Göztepe / Bostancı / Ankara</VirtualEarth:Sign>
At exit <VirtualEarth:ExitNumber>Ankara Devlet Yolu Kavşağı</VirtualEarth:ExitNumber>, <VirtualEarth:Action>take</VirtualEarth:Action> ramp <VirtualEarth:TurnDir>right</VirtualEarth:TurnDir> for <VirtualEarth:RoadName>Dolayoba Caddesi</VirtualEarth:RoadName> toward <VirtualEarth:Sign>Sultanbeyli / Kurtköy / Gözdağı</VirtualEarth:Sign>
<VirtualEarth:Action>Bear</VirtualEarth:Action> <VirtualEarth:TurnDir>right</VirtualEarth:TurnDir> onto <VirtualEarth:RoadName>Hilal Sokak</VirtualEarth:RoadName>
<VirtualEarth:Action>Bear</VirtualEarth:Action> <VirtualEarth:TurnDir>right</VirtualEarth:TurnDir> onto <VirtualEarth:RoadName>Hukukçular Caddesi</VirtualEarth:RoadName>
<VirtualEarth:Action>Turn</VirtualEarth:Action> <VirtualEarth:TurnDir>left</VirtualEarth:TurnDir> to stay on <VirtualEarth:RoadName>Hukukçular Caddesi</VirtualEarth:RoadName>
<VirtualEarth:Action>Turn</VirtualEarth:Action> <VirtualEarth:TurnDir>right</VirtualEarth:TurnDir> onto <VirtualEarth:RoadName>Bayram Sokak</VirtualEarth:RoadName>
<VirtualEarth:Action>Turn</VirtualEarth:Action> <VirtualEarth:TurnDir>right</VirtualEarth:TurnDir>, and then immediately <VirtualEarth:Action>turn</VirtualEarth:Action> <VirtualEarth:TurnDir>left</VirtualEarth:TurnDir> onto <VirtualEarth:RoadName>Azizoğlu Caddesi</VirtualEarth:RoadName>
<VirtualEarth:Action>Turn</VirtualEarth:Action> <VirtualEarth:TurnDir>right</VirtualEarth:TurnDir> onto <VirtualEarth:RoadName>Yayalar Caddesi</VirtualEarth:RoadName>
<VirtualEarth:Action>Arrive</VirtualEarth:Action> at <VirtualEarth:WaypointName>Yayalar Caddesi</VirtualEarth:WaypointName>
Press any key to continue . . .

Eh! Fena bir tarif sayılmaz. Tabi istanbul' da yaşayan birisi olarak, yol durumuna göre bu tarif hiçe sayılabilir. Kestirme sokaklardan gidilebilir. Hatta bana kalsa ben arabalı vapur ile üsküdara geçer oradan pendiğe gitmek için farklı bir güzergah kullanırım

Dönüş verisi tabi **XML** formatındadır. Bu sebepten bu içeriği parse ederek daha düzgün bir sonuç almak yerinde olacaktır. Bu size ödev olabilir mi acaba? Olur olur Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

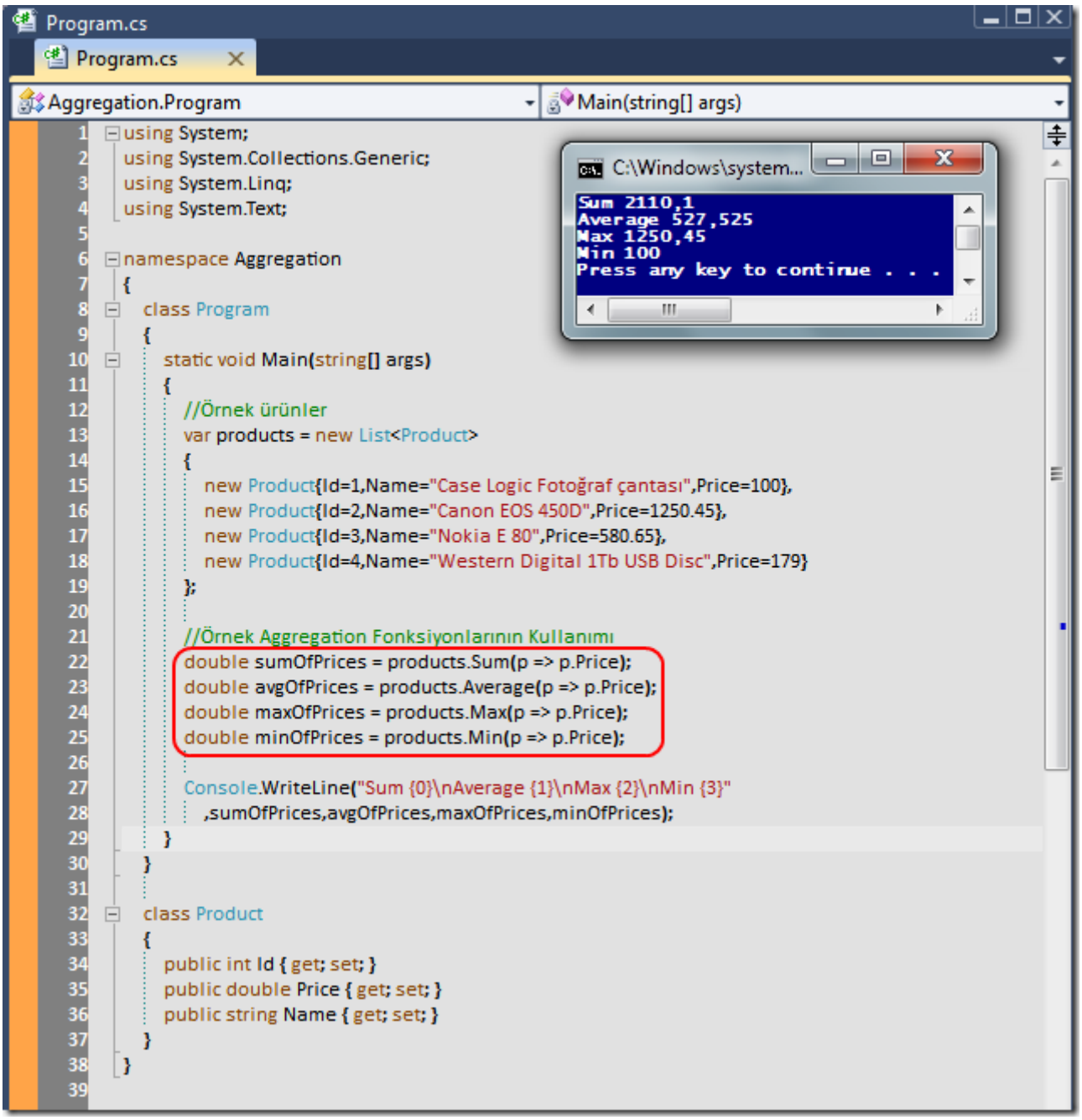
BingMapSamples.zip (734,34 kb)

Tek Fotoluk İpucu-46(LINQ Aggregate Fonksiyonları)

Pazar, 5 Şubat 2012 21:30

Linq, C#, Aggregate Functions

Merhaba Arkadaşlar,
LINQ tarafında Sum,Max,Min,Average gibi bazı hesaplama fonksiyonları vardır. Bunlara ait örnek bir kullanımı aşağıda bulabilirsiniz




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Aggregation
{
    class Program
    {
        static void Main(string[] args)
        {
            //Örnek ürünler
            var products = new List<Product>
            {
                new Product{Id=1,Name="Case Logic Fotoğraf çantası",Price=100},
                new Product{Id=2,Name="Canon EOS 450D",Price=1250.45},
                new Product{Id=3,Name="Nokia E 80",Price=580.65},
                new Product{Id=4,Name="Western Digital 1Tb USB Disc",Price=179}
            };

            //Örnek Aggregation Fonksiyonlarının Kullanımı
            double sumOfPrices = products.Sum(p => p.Price);
            double avgOfPrices = products.Average(p => p.Price);
            double maxOfPrices = products.Max(p => p.Price);
            double minOfPrices = products.Min(p => p.Price);

            Console.WriteLine("Sum {0}\nAverage {1}\nMax {2}\nMin {3}"
                ,sumOfPrices,avgOfPrices,maxOfPrices,minOfPrices);
        }
    }

    class Product
    {
        public int Id { get; set; }
        public double Price { get; set; }
        public string Name { get; set; }
    }
}
```

Aggregation.rar (22,75 kb)

Priority Queue Collection

Çarşamba, 1 Şubat 2012 00:01

Collections, Priortiy Queue, Data Structures, Max Heap, Min Heap, Heap Data Structure, C#

1996 yılıydı. **Efes Pilsen***(bu günkü adıyla Anadolu Efes)*spor klübü altın dönemlerini yaşıyordu. **Tamer Oyguç, Ufuk Sarıca, Peter Naumoski, Hidayet Türkoğlu, Mirsad Türkcan** ve diğerleri. O yıl kulüp **Avrupa Koraç**kupasında finale yükselmiş ve **İtalyan** temsilcisi**Stefanel Milano'** nun rakibi olmuştu. İlk maç **İstanbul Abdi İpekçi** spor salonundaydı. Üniversitedeydim ve arkadaşlarımla günün erken saatlerinde stad kuyruğuna girmiştim.

Biletlerimiz vardı ancak bulunduğumuz tribünde herkes karışık oturacağı için iyi bir yer bulmak adına sabahın köründe ve soğuk bir havada kendimizi orada buluvermiştik. Ne varki sabahın erken saatlerinde sıranın başlarında iken bu zamanla değişmeye başlamıştı. Çünkü alt yola bir basketçi geliyor, hayranları ona bakmak için bulunduğumuz kuyruğa hücum ediyor ve baktıkları yerden tekrar geriye çıkmıyorlarda. 8nci sıradan 148nci sıraya kadar gelmişimdir sanıyorum ki. Bu noktada çok doğal olarak sıranın en başındaki kişi de sıksık değişiyordu. 😬

Aslında **VIP** benzeri bir durum söz konusu idi belkide. Hani pek çok filmde görmüşüzdür. Lüks bir barın veya gece kulübünün önünde içeri girmek için bekleyen pek çok insandan oluşan bir kuyruk söz konusu olur. Ama genelde filmin kahramanı ve hatta arkadaşları

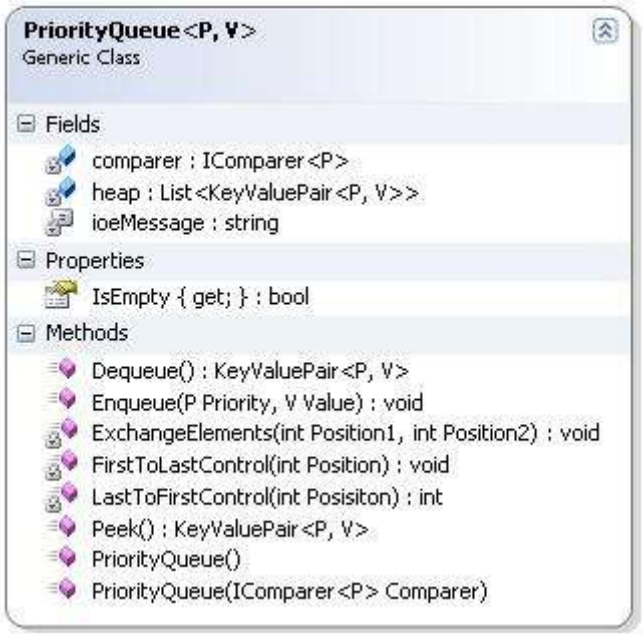
gelir, en önden kulübe pat diye girerlar. Çünkü yüksek öncelikli şahsiyetlerdir. Sanırım hayatımızda yaşadığımız bu ve benzeri tipteki vakalar yazılım dünyasından da nasibini almıştır. Çünkü yazılım tarafında da **Öncelikli Kuyruk Koleksiyonu(Priority Queue Collection)** denen bir **veri yapısı(data structure)** söz konusudur (*Detaylı bilgi için Wiki adresine bakabilirsiniz)*
Temel olarak bu tip bir koleksiyon **Queue*(FIFO - First In First Out ilkesine göre çalışmaktadır)*** ve **Stack*(LIFO - Last in First Out ilkesine göre çalışmaktadır)*** tipinden olanlarına benzer. Bu tip koleksiyonlar*(generic karşılıkları da dahil olmak üzere)*, bildiğiniz gibi tek boyutludur. **Priority** koleksiyonunda ise devreye bir öncelik değeri girmektedir. Bir başka deyişle koleksiyona eklenecek olan elemanın öncelik derecesi söz konusu olup ilk sıraya yerleşmesi söz konusudur. **Priority Queue** koleksiyonunda, koleksiyona dahil olacak eleman yüksek öncelik derecesine göre kuyuruğun ön sırasına alınır. Bunun dışında söz konusu koleksiyon modelinde ilk olarak elde edilmesi gereken veya listeden çıkartılması icap eden bir eleman bulunurken bu yüksek öncelik seviyesi göz önüne alınabilir*(Yine de dilerseniz bu standart veri yapısı kurallarını esnetebilirsiniz)*

Bildiğiniz gibi **.Net Framework** platformu ilk sürümünden itibaren çeşitli tipte koleksiyonlara hizmet etmektedir. İlk sürümde **Stack, Queue, ArrayList, Hashtable** ve **SortedList** gibi farklı şekillerde çalışan koleksiyonlar söz konusudur. **Framework .Net 2.0** sürümüne yükseldiğinde ise koleksiyonların generic olma durumu devreye girmiş ve bu sayede **tür bağımsız** ve **tip güvenli*(Type Safety)*** versiyonlar ortaya çıkmıştır**(List<T>, Dictionary<T,K>, Stack<T>, Queue<T>, HashSet<T>, SortedDictionary<T,K>, SortedList<T> gibi)**.

Tabi **Framework**sürümü **4.0**' a yükseldiğinde ve işin içerisinde **paralel programlama** kabiliyetleri de girdiğinde,**Concurrent** olarak çalışabilen koleksiyonlar ortaya çıkmıştır**(BlockingCollection<T>, ConcurrentBag<T>, ConcurrentDictionary<T,K>, ConcurrentQueue<T>, ConcurrentStack<T>, OrderablePartitioner<T>, Partitioner<T> gibi)**.
Ne yazık ki tüm bu koleksiyon tipleri arasında öncelik seviyelendirmesini kullanan bir **Queue**koleksiyonu mevcut değildir. Bir başka deyişle iş başa düşer ve bu koleksiyonu bizim yazmamız gerekir. Aslında **.Net Framework** alt yapısı içerisindeki koleksiyonların gelişimini düşündüğümüzde, söz konusu yeni koleksiyon tipinin **generic** bir versiyonu dışında **Concurrent**çalışabilen sürümünü de yazmak icap etmektedir Biz bu yazımızda basit ve daha kolay bir adım atıp **generic** olan bir sürümünü yazmaya çalışıyor olacağız.

Tasarlayacağımız yeni koleksiyon tipi içerisinde işimizi kolaylaştırmak adına **Sorted List** kullanımı söz konusu olabilir aslında. Fakat yaptığım araştırmalar sonucunda bu tip bir kullanımın performans açısından olumsuz etkileri olduğunu gördüm. Dolayısıyla bu tip bir veri yapısının literatürde bahsedildiği üzere **Yığın Veri Yapısı(Heap Daha Structure)** ile çalışması tavsiye edilen ve önerilen yoldur.

Heap veri yapısı, **ağaç tabanlı(Tree Based)** bir içeriği model olarak almaktadır. Ağacın en üst dalında her zaman için en yüksek **Key** değerine sahip olan eleman yer alır. Tabi bu durumu tersine doğru çevirebiliriz de. Yani en düşük **Key** değerine sahip elemanın ağacın en üst dalında olması da sağlanabilir. Söz konusu kullanımlara göre **Heap** veri yapısı **Max-Heap** veya**Min-Heap** olarakta adlandırılmaktadır. *(Heap veri yapısı hakkında daha detaylı bilgiye yine Wiki adresi üzerinden ulaşabilirsiniz)*
Dilerseniz öncelikli olarak bu koleksiyon tipini geliştirmeye çalışalım. Bu amaçla basit bir **Console**projesi açıp aşağıdaki sınıfı geliştirdiğimizi düşünebiliriz.




```
using System;
using System.Collections.Generic;

namespace PQueue
{
   public class PriorityQueue<P, V> // P priority derecesini V ise değeri temsil eder
   {
      private List<KeyValuePair<P, V>> heap; // Heap veri yapısına uygun olacak şekilde içeriğimizi tutacağımız koleksiyon
      private IComparer<P> comparer; // Max-Heap veya Min-Heap' e göre bir uyarlamaya hizmet verebilmek için kullanılacak arayüz referansı
      private const string ioeMessage = "Koleksiyonda hiç eleman yok";

      #region Constructors

      // P ile gelen tipin varsayılan karşılaştırma kriterine göre bir yol izlenir
      public PriorityQueue()
```

```
        : this(Comparer<P>.Default)
    {
    }

    // P tipinin karşılaştırma işlevselliğini üstlenen bir IComparer implementasyonu ile bir
Construct işlemi söz konusudur
    public PriorityQueue(IComparer<P> Comparer)
    {
        if (Comparer == null)
            throw new ArgumentNullException();

        heap = new List<KeyValuePair<P, V>>();
        comparer = Comparer;
    }

    #endregion

    #region Temel Fonksiyonlar

    // Koleksiyona bir veri eklemek için kullanıyoruz
    public void Enqueue(P Priority, V Value)
    {
        KeyValuePair<P, V> pair = new KeyValuePair<P, V>(Priority, Value);
        heap.Add(pair);

        // Sondan başa doğru yeniden bir sıralama yaptırılır
        LastToFirstControl(heap.Count - 1);
    }

    public KeyValuePair<P, V> Dequeue()
    {
        if (!IsEmpty) // Eğer koleksiyon boş değilse
        {
            KeyValuePair<P, V> result = heap[0]; // Heap' in Root' undaki elemanı al
            // Dequeue mantıksal olarak ilk elemanı geriye döndürürken aynı zamanda
koleksiyondan çıkartmalıdır
            if (heap.Count <= 1) // 1 veya daha az eleman söz konusu ise zaten temizle
            {
                heap.Clear();
            }
            else // 1 den fazla eleman var ise ilgili elemanı koleksiyondan çıkart
```

```
            {
                heap[0] = heap[heap.Count - 1];
                heap.RemoveAt(heap.Count - 1);
                FirstToLastControl(0); // ve koleksiyonu baştan sona yeniden sırala
            }
            return result;
        }
        else
            throw new InvalidOperationException(ioeMessage);
    }

    // Peek operasyonu varsayılan olarak ilk elemanı geriye döndürür ama koleksiyondan
çıkartmaz(Dequeue gibi değildir yani)
    public KeyValuePair<P, V> Peek()
    {
        if (!IsEmpty)
            return heap[0];
        else
            throw new InvalidOperationException(ioeMessage);
    }

    // Koleksiyonda eleman olup olmadığını belirtir
    public bool IsEmpty
    {
        get { return heap.Count == 0; }
    }

    #endregion

    #region Sıralama Fonksiyonları

    private int LastToFirstControl(int Posisiton)
    {
        if (Posisiton >= heap.Count)
            return -1;

        int parentPos;

        while (Posisiton > 0)
        {
            parentPos = (Posisiton - 1) / 2;
```

```
            if (comparer.Compare(heap[parentPos].Key, heap[Posisiton].Key) > 0)
            {
                ExchangeElements(parentPos, Posisiton);
                Posisiton = parentPos;
            }
            else break;
        }
        return Posisiton;
    }

    private void FirstToLastControl(int Position)
    {
        if (Position >= heap.Count)
            return;

        while (true)
        {
            int smallestPosition = Position;
            int leftPosition = 2 * Position + 1;
            int rightPosition = 2 * Position + 2;
            if (leftPosition < heap.Count &&
                comparer.Compare(heap[smallestPosition].Key, heap[leftPosition].Key) > 0)
                smallestPosition = leftPosition;
            if (rightPosition < heap.Count &&
                comparer.Compare(heap[smallestPosition].Key, heap[rightPosition].Key) > 0)
                smallestPosition = rightPosition;

            if (smallestPosition != Position)
            {
                ExchangeElements(smallestPosition, Position);
                Position = smallestPosition;
            }
            else break;
        }
    }

    private void ExchangeElements(int Position1, int Position2)
    {
        KeyValuePair<P, V> val = heap[Position1];
        heap[Position1] = heap[Position2];
        heap[Position2] = val;
```

```
        }

        #endregion
    }
}
```

PriortiyQueue<P,V> isimli sınıfımız, P tipinin değerine ve seçilen karşılaştırma kriterine göre bir öncelik seviyelendirmesini kullanmaktadır. Söz konusu seviyelendirmeye göre de **V** tipi ile belirtilen değerlerin koleksiyon içerisindeki önceliği değişmektedir. **Enqueue** ve **Dequeue** gibi veri ekleme ve en yüksek önceliğe sahip veriyi elde edip listeden çıkartmak gibi temel **Queue** fonksiyonellikleri dışında **Peek** isimli operasyon desteği de mevcuttur. Şimdi dilerseniz yazdığımız bu yeni koleksiyon tipini test edelim. Bu amaçla aşağıdaki kod parçasını göz önüne alabiliriz.

```
using System;

namespace PQueue
{
    class Program
    {
        static void Main(string[] args)
        {
            PriorityQueue<int, Person> pQueue = new PriorityQueue<int, Person>();

            pQueue.Enqueue(9, new Person { Name = "Bill", PersonId = 10, Salary = 1000 });
            pQueue.Enqueue(10, new Person { Name = "Jeni", PersonId = 5, Salary = 950 });
            pQueue.Enqueue(8, new Person { Name = "Samantha", PersonId = 4, Salary = 750
});
            pQueue.Enqueue(17, new Person { Name = "Richard", PersonId = 3, Salary = 800
});
            pQueue.Enqueue(12, new Person { Name = "Steve", PersonId = 9, Salary = 2000
});

            Console.WriteLine("Şu anda en yüksek öncelik {0} isimli
personeldedir",pQueue.Dequeue().Value.Name);

            // Şu anda en yüksek öncelik seviyesi Dennis' de olduğundan Root' a
yerleştirilecektir
            pQueue.Enqueue(3, new Person { Name = "Dennis", PersonId = 1, Salary = 350
});
            Console.WriteLine("Şu anda en yüksek öncelik {0} isimli
personeldedir",pQueue.Dequeue().Value.Name);
            Console.WriteLine("Şu anda en yüksek öncelik {0} isimli
```

```
personeldedir",pQueue.Dequeue().Value.Name);
        }
    }

    class Person
    {
        public int PersonId { get; set; }
        public string Name { get; set; }
        public decimal Salary { get; set; }
    }
}
```

Örneğimizde kullandığımız **PriorityQueue<P,V>** nesne örneği, seviyelendirme için **int** tipini kullanmakta ve buna göre **Person** tipinden olan nesne örneklerini **Min-Heap** mantığında değerlendirmektedir. İlk olarak **Bill, Jenni, Samantha, Richard** ve **Steve** isimli çalışanlar kuyruğa eklenmiştir. Bu personelin öncelik dereceleri sırasıyla **9,10,8,17** ve **12**' dir.

Geliştirdiğimiz tip **Min-Heap** yapısını ele aldığından yani aslında düşük değer yüksek öncelik anlamına geldiğinden ilk dizilime göre kuyruktan ilk çıkacak kişi **Samantha**' dır. Bu andan itibaren **Dequeue** metodunu kullandığımız noktalarda öncelik seviyesi yüksek olan*(yani en küçük int değerine sahip olan)* çalışan elde edilecek ve aynı zamanda koleksiyondan çıkartılacaktır. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.

```
using System;

namespace PQueue
{
    class Program
    {
        static void Main(string[] args)
        {
            PriorityQueue<int, Person> pQueue = new PriorityQueue<int, Person>();
            pQueue.Enqueue(9, new Person { Name = "Bill", PersonId = 10, Salary = 1000 });
            pQueue.Enqueue(10, new Person { Name = "Jeni", PersonId = 5, Salary = 950 });
            pQueue.Enqueue(8, new Person { Name = "Samantha", PersonId = 4, Salary = 750 });
            pQueue.Enqueue(17, new Person { Name = "Richard", PersonId = 3, Salary = 800 });
            pQueue.Enqueue(12, new Person { Name = "Steve", PersonId = 9, Salary = 2000 });

            Console.WriteLine("Şu anda en yüksek öncelik {0} isimli personeldedir"
                ,pQueue.Dequeue().Value.Name);

            // Şu anda en yüksek öncelik seviyesi Dennis' de olduğundan Root' a yerleştiril
            pQueue.Enqueue(3, new Person { Name = "Dennis", PersonId = 1, Salary = 350 });
            Console.WriteLine("Şu anda en yüksek öncelik {0} isimli personeldedir"
                , pQueue.Dequeue().Value.Name);
            Console.WriteLine("Şu anda en yüksek öncelik {0} isimli personeldedir"
                , pQueue.Dequeue().Value.Name);
        }
    }

    class Person
    {
        public int PersonId { get; set; }
        public string Name { get; set; }
        public decimal Salary { get; set; }
    }
}
```

```
C:\WINNT\system32\cmd.exe
Şu anda en yüksek öncelik Samantha isimli personeldedir
Şu anda en yüksek öncelik Dennis isimli personeldedir
Şu anda en yüksek öncelik Bill isimli personeldedir
Press any key to continue . . .
```

Burada dikkat edilmesi gereken nokta **Dequeue** metodunun her zaman için öncelik seviyesi yüksek olan*(örneğimizde int değeri en düşün olan)* elemanı ilk olarak listeden getiriyor olmasıdır. Yani klasik**Queue** koleksiyonunda olduğu gibi **FIFO** ilkesi değerlendirilirken, **öncelik seviyesi*(Priority Level)*** baz alınmaktadır.

Kodu **Debug** ederek incelediğinizde ancak var olan elemanlardan daha yüksek öncelik seviyesine sahip bir eleman eklendiğinde listenin başına alındığını görürüz. Bu işleyişte koleksiyon üzerinde her hangibir sıralama operasyonunun söz konusu olmadığına özellikle dikkat etmenizi isterim.

Görüldüğü gibi biraz uğraşarak öncelikli derecelendirmeli bir kuyruk koleksiyonu tipini yazabildik. Sizde denemelerinizde IComparer implementasyonu yapan bir kritere göre söz

konusu koleksiyonu**Max-Heap** mantığında çalıştırmay deneyebilirisiniz. Böylece geldik bir makalemizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

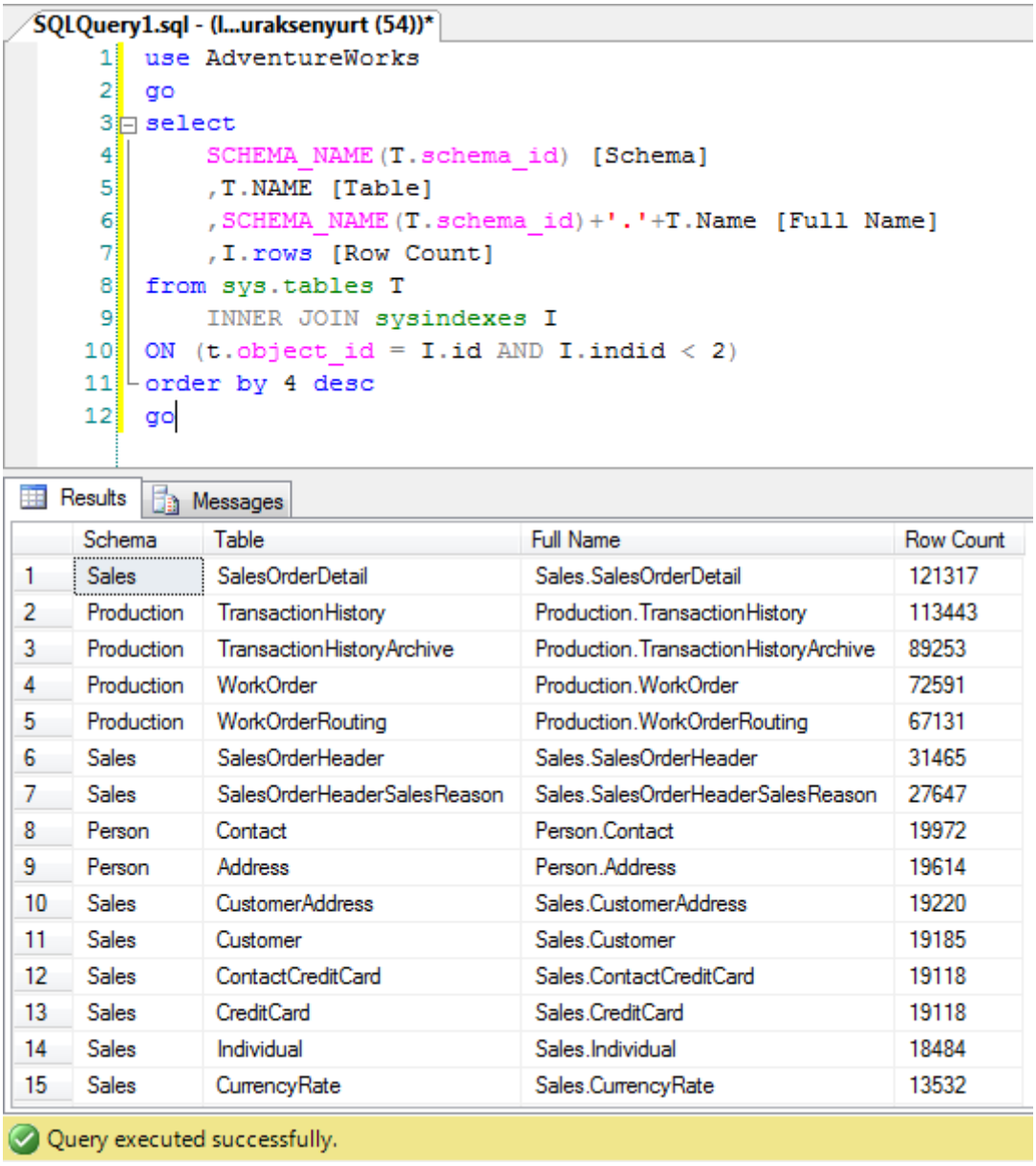
PQueue.rar (31,33 kb)

Tek Fotoluk İpucu 45 - Schema Adı ile birlikte Tablo Satır Sayılarını Elde Etmek (2012-01-12T17:46:00)

sql,table,indexes,row count,schema,

Merhaba Arkadaşlar,

SQL tarafındaki sistemsel nesneleri göz önüne aldığımızda inanılma faydalı ve bilgilendirici sorgular yazabildiğimizi eminim ki hepiniz biliyorsunuzdur. Söz gelimi bir veritabanı içerisinde yer alan tablo adlarını, şema adları ile birlikte ve güncel satır sayılarını da içerek şekilde elde etmek istediğinizi düşünelim. Ne yaparsınız



	Schema	Table	Full Name	Row Count
1	Sales	SalesOrderDetail	Sales.SalesOrderDetail	121317
2	Production	TransactionHistory	Production.TransactionHistory	113443
3	Production	TransactionHistoryArchive	Production.TransactionHistoryArchive	89253
4	Production	WorkOrder	Production.WorkOrder	72591
5	Production	WorkOrderRouting	Production.WorkOrderRouting	67131
6	Sales	SalesOrderHeader	Sales.SalesOrderHeader	31465
7	Sales	SalesOrderHeaderSalesReason	Sales.SalesOrderHeaderSalesReason	27647
8	Person	Contact	Person.Contact	19972
9	Person	Address	Person.Address	19614
10	Sales	CustomerAddress	Sales.CustomerAddress	19220
11	Sales	Customer	Sales.Customer	19185
12	Sales	ContactCreditCard	Sales.ContactCreditCard	19118
13	Sales	CreditCard	Sales.CreditCard	19118
14	Sales	Individual	Sales.Individual	18484
15	Sales	CurrencyRate	Sales.CurrencyRate	13532

Binary Search Tree' yi Anlamak (2012-01-10T00:01:00)

binary tree,binary search tree,tree node,data structures,algorithms,

 Merhaba Arkadaşlar,

İnsan hafızası gizemli çalışan ama çoğu zamanda bizleri şaşırtan bir mekaniğe sahiptir. Doğduğumuz andan itibaren 3 yaşına kadar geçen zaman dilimi içerisinde görsel olarak ne izlersek kaparız. Ancak neredeyse bunların hiç birini hatırlamayız. çocukluğumuz, ergenliğimiz, yetişkinliğimiz, orta yaş halimiz ve yaşlılığımız. Tüm bu zaman dilimlerinde beynimiz sürekli olarak bir şeyleri hafıza da tutma ihtiyacı hisseder. Zaman zaman öğrendiğimiz pek çok bilgiyi kolayca unutur ve ihtiyacımız olduğunda hatırlamakta zorlanırız. Ama kimi bilgilerde bilinç altımıza neredeyse kazınır ve geçerli bir sağlık problemi oluşmadığı sürece hiç unutmayız*(çarpım tablosu, matematik dört işlemin nasıl yapıldığı veya hangi ışıkta durulması gerektiği gibi)*

Peki kolayca unutabileceğimiz bilgiler nasıl oluşurlar? 😕

Genelde öğrendiklerimizi çok sık kullanmadığımız veya tekrar etmediğimiz durumlarda unutmak kaçınılmaz gerçeklerden birisidir. Kullanmama süresi arttığında ise, ihtiyaç duyulduğu anda söz konusu bilgileri tekrardan hatırlamak da giderek zorlaşmaktadır. Aslında çok sık kullanılmayan ama yaşamın herhangibir anında ihtiyaç duyabileceğimiz bilgileri yazarak bir yerlerde saklamak, mücadele etme yollarından birisidir.

örneğin üniversitede okutulan veri yapıları ve algoritmalar derslerini düşünelim. öğrenmek, kavrayabilmek, kodlarını geliştirebilmek için epey kafa patlattığımız zorlayıcı bu içerikler, kolayca unutulabilecek cinstendir 😕 Tabi eğer bunları her dönem anlatan bir akademisyen veya sürekli matematik modeller geliştiren bir uzman değilseniz. Eğer ki, **.Net Framework, Java EE, SAP** gibi ortamları kullanarak ağırlıklı olarak veri odaklı uygulamalar geliştiriyorsanız, zaten bu ders içeriğinin yakınından çok nadir olarak geçersiniz. İşte ben de bu hafıza kaybını yaşadığım şu dönemlerde, eski bilgilerimi hatırlamaya çalışmak istedim. İlk gözüme kestirdiğim konu ise **ikili ağaç yapısı(Binary Tree)** ve bunun üzerinden arama, ekleme gibi temel işlemlerin nasıl yapıldığı oldu?

İkili ağaç yapısı basitliği ve hızlı sonuç üretimi açısından bakıldığında, arama algoritmalarından tutunda oyun programlamaya, ilişkisel veri tabanlarından, karmaşık matematik modellere kadar pek çok alanda kullanılmaktadır. **Binary Tree** veri yapısı ve bu

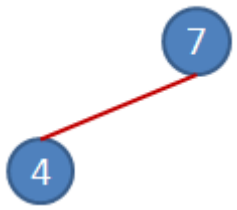
yapı üzerinde gerçekleştirilecek çeşitli operasyonlar*(arama, ekleme, silme gibi)* göz önüne alındığında ilk etapta bu ağacın nasıl oluşturulduğunun kavranması gerekmektedir. Aslında ikili ağaçlar, şirketlerin organizasyon ağacına benzer bir içeriğe sahiptirler. Tabi teknik olarak düşündüğümüzde bu ağacı oluşturmanın bazı kuralları vardır. Şimdi gelin teknik terim karmaşası içerisine girmeden konuyu bir örnek üzerinde kavramaya çalışalım.

Elimizde şöyle bir sayı dizisi olduğunu düşünelim.

7,4,9,1,3,10,12,8,5,6,9,11

Şimdi bu rakamsal dizinin **Binary Tree** grafiğini oluşturmaya çalışalım.

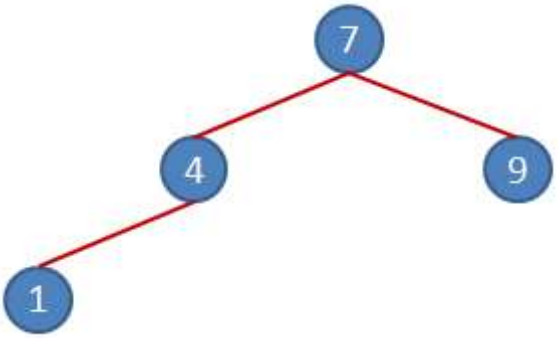
İlk olarak 7 rakamından başlayıp, bunun **Kök Boğum(Root Node)** olduğunu düşünerekten en başa yerleştirelim. Ardınan ikinci rakama geçelim. İkinci rakam, ilk rakam olan 7' ye bağlı olmak durumda. Bir başka deyişle onun **alt boğumu/çocuk boğumu(Child Node)** olacak. 4, 7' den küçük bir rakam. Bu sebepten onu sol alt tarafa alalım.



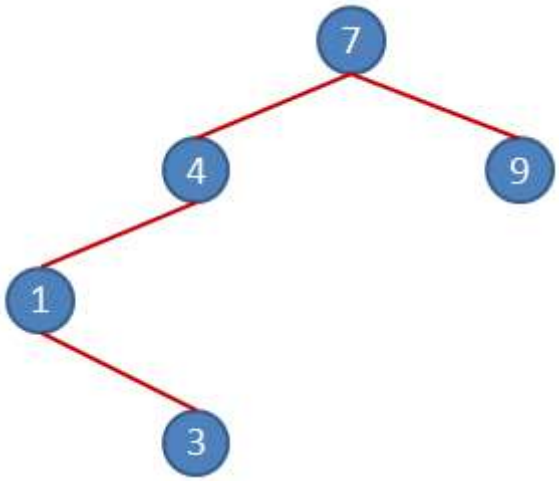
3ncü rakamımız ise 9. Root Node ile karşılaştırdığımızda ondan büyük olduğunu görüyoruz. Bu yüzden onu 7nin sağ alt node' u olacak şekilde grafiğimize yerleştirelim.

Hımmm... Demek ki bir kuralımız var. **"Bir node değeri eğer bağlı olduğu node' un içindeki değerden küçükse onun solunda, değilse sağında yer alıyor olmalı."**

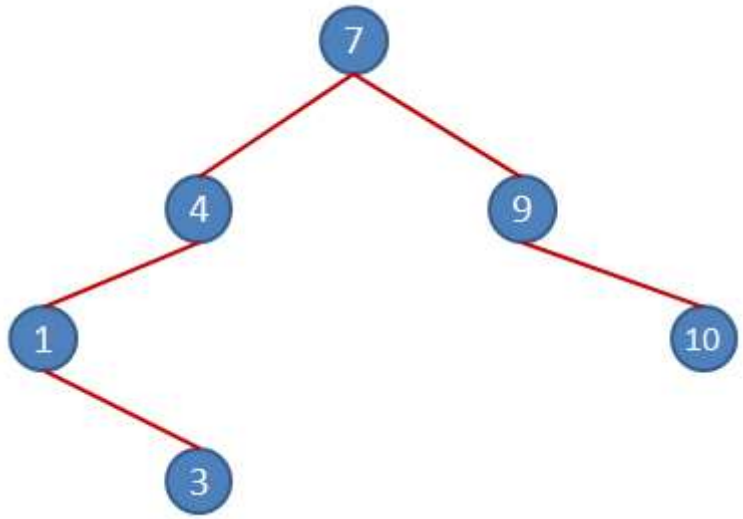
4ncü rakamdan devam edelim.1, root node olan 7den küçük. Dolayısıyla sol dalda yer almalı. Ancak sol dalda 4 değeri de var. 1, 4ten küçük olduğundan ve az önce bahsettiğimiz kuraldan dolayı sol alt node olarak grafiğe eklenmeli.



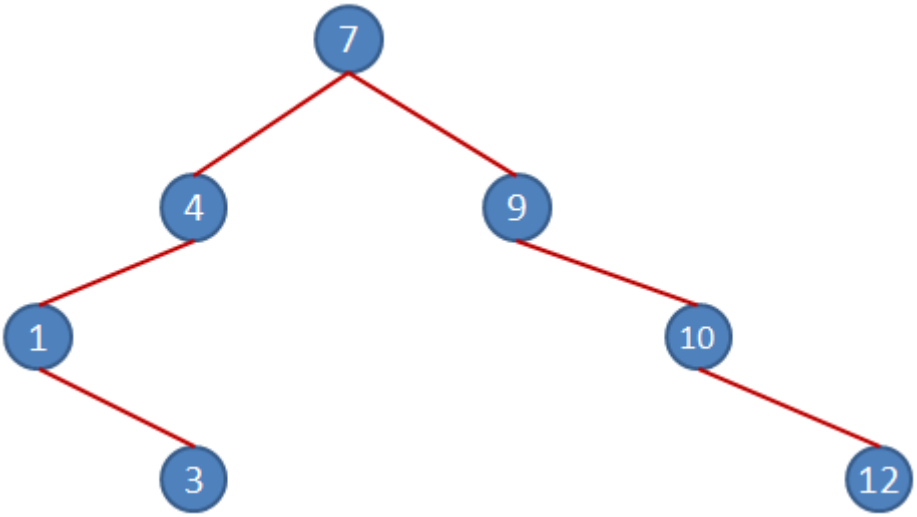
5nci rakamımız ise 3. Yine Root Node' dan aşağıya doğru inmeye başlıyoruz. Kuralımıza göre 3, 7den küçük ve onun sol dalında yer almalı. Bir alt seviyeye indiğimizde 4 ile karşılaşılıyor. 3, 4ten küçük olduğu için kuralımıza göre sol alt node olmalı ama yol devam ediyor. çünkü sol alt node' da 1 var. 3, 1den büyük olduğu için kuralımıza göre sağ alt node' olmalı.



6ncı rakamımız 10. 10, root node olan 7den büyük olduğu için kuralımıza göre sağ dalda yer almalı. Sağ daldan aşağıya doğru indiğimizde 1nci seviyede 9 olduğunu görüyoruz. 10, 9dan büyük olduğu için sağ alt node' unda yer almalı.



7nci rakamımız 12. Yine Root Node ile kıyaslayarak başladığımızda kuralımıza göre sağ dalda olması gerekiyor. 12, 9 dan büyük olduğu için sağ daldan inmeye devam ediyor ve yine 10dan büyük olduğu için sağ alt node olarak 3ncü seviyedeki yerini alıyor.



8nci elemanımız ise 8. Root node olan 7den büyük olduğu için sağ daldan aşağıya doğru akması gerekiyor. Yol üstünde ilk olarak karşılaştığı 9dan küçük olduğu içinse sol alt node olarak 2nci seviyedeki yerini alıyor.

9ncu elemanımız 5. Root Node olan 7den küçük olduğu için kuralımıza göre sol daldan aşağıya doğru kaydırılması gerekiyor. 1nci seviyede karşılaştığımız 4ten büyük bir değer olduğu içinse sağ alt node olarak 2nci seviyedeki yerini alıyor.

10ncu elemanımız 9. Ancak 9 zaten rakam dizimizde bir kez kullanıldı ve bir kere daha kullanılmaması gerekiyor. Hımm...öyleyse bir kural daha ortaya çıkıyor.

"Zaten dizide var olan bir elemanı ağaça tekrardan ekleyemeyiz."

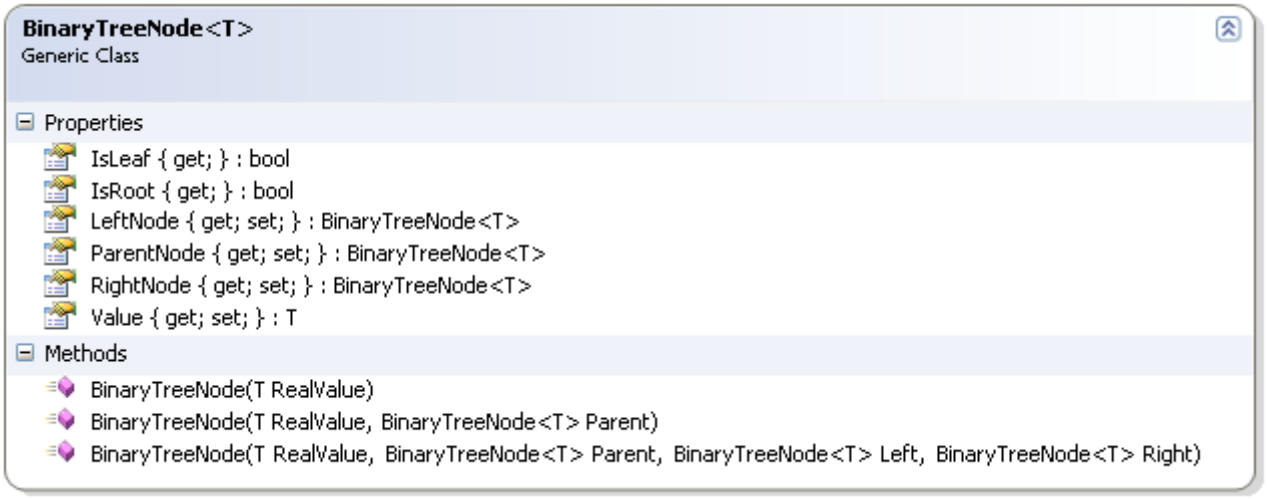
örneğimizde yer alan son elemanımız ise 11. Yine root node ile başladığımızda 7den büyük olduğu için kuralımıza göre sağ dalda yer alması gerekiyor. Rakamı sağ daldan aşağıya doğru kaydırdığımızda, 1nci seviyedeki 9dan büyük olduğunu görüyoruz. Buna göre sağ daldan devam edilmeli. Sıradaki 10 rakamından da büyük olduğundan yine sağ dalda kalmalı. Sonradan gelen 12 rakamından küçük olduğu içinse sol alt dal olarak son seviyede yerini almalı.

Daha başka sayımız kalmadığı için Binary Tree grafiğini tamamlamış bulunuyoruz. Görüldüğü gibi Root Node' dan aşağıda doğru inen ağaç yapısında **her boğuma en fazla iki boğum bağlanabilmektedir***(Zaten Binary denmesinin sebebi de budur)*. Bunun temel sebebi ise büyük ve küçük olma durumuna göre ilgili boğumun sağ veya sol alt boğum olarak grafiğe ekleniyor olmasıdır. Tabi bunun dışında zaten ağaç içerisinde yer almış bir elemanın tekrardan yer almaması gerekliliği ortaya konulmuştur. Dallar haricinde aslında boğumlar belirli seviyelere denk gelirler. Tüm bu bilgiler ışığında sayı dizisinin **Binary Tree** grafiği aşağıdaki nihai halini alır.

Peki bu tip bir ağaç yapısını programatik ortamda tasarlamak istersek?

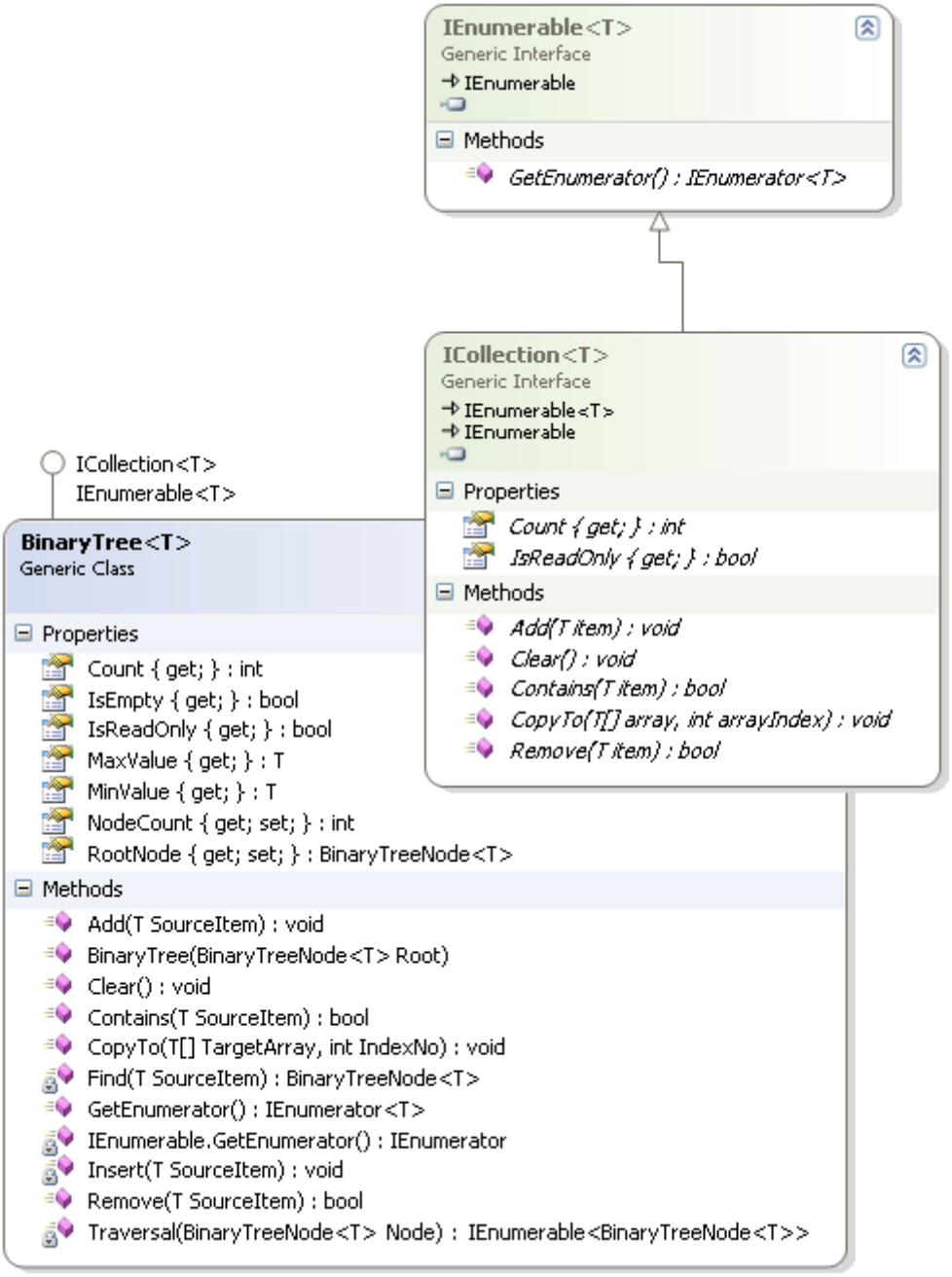
Aslında bu ağaç yapısı içerisinde dolaşmak bile başlı başına bir iş. Kağıt üzerinde her ne kadar kolay olsa da teorik olarak bunun bilinen 3 yöntemi bulunmaktadır. **Traverse** adı verilen bu dolaşım teknikleri InOrder*(Sol dalı dolaş, root' a uğra, sağ dalı dolaş)*, PreOrder*(Root' dan başla, sol dalı yukarıdan aşağı gez, sol dalı yukarıdan aşağı gez)* ve PostOrder*(Sol dalı aşağıdan yukarı tara, sağ dalı aşağıdan yukarı tara, root' a uğra)* olarak geçmektedir.

Ne yazık ki **.Net Framework** tarafında **Binary Tree** yapısını destekleyen hazır bir tip ve özellikle koleksiyon bulunmamaktadır. Dolayısıyla bunu kendimiz oluşturmak durumundayız. Biraz uğraştırıcı olsa da buna hizmet edecek bir geliştirme yapabiliriz. Bize iki temel tip gerekmektedir. Birincisi ağaç yapısındaki her bir boğumu temsil edecek olan tiptir. Bu tip bir Node' un değerini, kendinden sonraki Sol ve Sağ Node' ları ve bağlı olduğu Parent Node' u bilecek şekilde tasarlanmalıdır. Yani aşağıdaki gibi.




```
public class BinaryTreeNode<T>
{
   public T Value { get; set; }
   public BinaryTreeNode<T> ParentNode { get; set; }
   public BinaryTreeNode<T> LeftNode { get; set; }
   public BinaryTreeNode<T> RightNode { get; set; }
   public bool IsRoot { get { return ParentNode == null; } }
   public bool IsLeaf { get { return LeftNode == null && RightNode == null; } }

   public BinaryTreeNode(T RealValue)
   {
      Value = RealValue;
   }

   public BinaryTreeNode(T RealValue, BinaryTreeNode<T> Parent)
   {
      Value = RealValue;
      ParentNode = Parent;
   }

   public BinaryTreeNode(T RealValue, BinaryTreeNode<T> Parent,
BinaryTreeNode<T> Left, BinaryTreeNode<T> Right)
   {
      Value = RealValue;
      RightNode = Right;
      LeftNode = Left;
      ParentNode = Parent;
   }
}
```

Bu basit bir tip. Asıl önemli olan ağacın grafiğini kodsal olarak sembolize edecek, Add, Remove ve Traverse gibi işlemlerini yapacak olan tipi geliştirmek. Onu da biraz uğraştıktan sonra aşağıdaki gibi yazabiliriz.



Dikkat edileceği üzere **BinaryTree** sınıfı **generic** olarak tasarlanmış ve **ICollection<T>** ile **IEnumerable<T> arayüzlerini(Interface)** uygulamıştır. Bu nedenle zorunlu olarak ezmesi gereken bazı üyeleri vardır. Ayrıca **BinaryTree** içerisinde yer alacak olan **BinaryTreeNode** nesne örneklerinin en azından değer bazında karşılaştırma yapılabilir olması gerekmektedir. Nitekim değerlerin bibirlerinden büyük ve küçük olma durumlarına göre bir yerleştirme ve arama söz konusudur. Bu yüzden bir de

generic **Constraint** konularak **T** tipinin**IComparable<T>** arayüzünü uygulama zorunluluğu konulmuştur. Sınıfımıza ait kodlar ise aşağıdaki gibidir.

```
public class BinaryTree<T>
   : ICollection<T>, IEnumerable<T>
   where T : IComparable<T>
{
   public BinaryTreeNode<T> RootNode { get; set; }
   public int NodeCount { get; set; }
   public bool IsEmpty { get { return RootNode == null; } }

   // Ağaç içerisindeki en küçük değerli elemanı döndürür
   public T MinValue
   {
      get
      {
         if (IsEmpty)
            throw new Exception("Ağaç içerisinde hiç bir eleman yok");
         BinaryTreeNode<T> tempNode = RootNode;

         while (tempNode.LeftNode != null) // Sol dallarda değer olduğu sürece dolaş
            tempNode = tempNode.LeftNode;

         return tempNode.Value;
      }
   }

   // Ağaç içerisindeki en büyük değerli elemanı döndürür
   public T MaxValue
   {
      get
      {
         if (IsEmpty)
            throw new Exception("Ağaç içerisinde hiç bir eleman yok");

         BinaryTreeNode<T> tempNode = RootNode;
         while (tempNode.RightNode != null) // Sağ dallarda değer olduğu sürece dolaş
            tempNode = tempNode.RightNode;

         return tempNode.Value;
      }
   }

   // Kaç eleman olduğunu döndürür
   public int Count
```

```
{
   get { return NodeCount+1; }
}

public BinaryTree(BinaryTreeNode<T> Root)
{
   RootNode = Root;
   NodeCount = 0;
}

public IEnumerator<T> GetEnumerator()
{
   foreach (BinaryTreeNode<T> tempNode in Traversal(RootNode))
      yield return tempNode.Value; // çok şükürki 2.0 ile gelen yield keyword' ü var :)
}

IEnumerator IEnumerable.GetEnumerator()
{
   foreach (BinaryTreeNode<T> tempNode in Traversal(RootNode))
      yield return tempNode.Value;
}

// Koleksiyona eleman ekleyebilmek için kullanılır
public void Add(T SourceItem)
{
   if (RootNode == null)
   {
      RootNode = new BinaryTreeNode<T>(SourceItem);
      NodeCount++;
   }
   else if(Contains(SourceItem))
      return;
   else
      Insert(SourceItem);
}

public void Clear()
{
   RootNode = null;
}

// Koleksiyonda T tipinden olan eleman olup olmadığını araştırır
public bool Contains(T SourceItem)
{
   if (IsEmpty)
```

```
            return false;

        BinaryTreeNode<T> tempNode = RootNode;
        while (tempNode != null)
        {
            int comparedValue = tempNode.Value.CompareTo(SourceItem);

            if (comparedValue == 0)
                return true;
            else if (comparedValue < 0)
                tempNode = tempNode.LeftNode;
            else
                tempNode = tempNode.RightNode;
        }

        return false;
    }

    // Koleksiyon içeriğinin aynı tipten bir Array' e kopyalar
    public void CopyTo(T[] TargetArray, int IndexNo)
    {
        T[] tempArray = new T[NodeCount+1];
        int Counter = 0;
        foreach (T value in this)
        {
            tempArray[Counter] = value;
            Counter++;
        }
        Array.Copy(tempArray, 0, TargetArray, IndexNo, this.NodeCount);
    }

    // Koleksiyondan eleman çıkartmak için kullanılır
    public bool Remove(T SourceItem)
    {
        BinaryTreeNode<T> item = Find(SourceItem);
        if (item == null)
            return false;

        List<T> values = new List<T>();
        foreach (BinaryTreeNode<T> tempNode in Traversal(item.LeftNode))
            values.Add(tempNode.Value);

        foreach (BinaryTreeNode<T> tempNode in Traversal(item.RightNode))
            values.Add(tempNode.Value);
```

```
    if (item.ParentNode.LeftNode == item)
        item.ParentNode.LeftNode = null;
    else
        item.ParentNode.RightNode = null;

    item.ParentNode = null;
    foreach (T value in values)
        Add(value);

    return true;
}

public bool IsReadOnly
{
    get { return false; }
}

BinaryTreeNode<T> Find(T SourceItem)
{
    foreach (BinaryTreeNode<T> item in Traversal(RootNode))
        if (item.Value.Equals(SourceItem))
            return item;

    return null;
}

// InOrder modeline göre elemanlar dolaşılır.
IEnumerable<BinaryTreeNode<T>> Traversal(BinaryTreeNode<T> Node)
{
    if (Node.LeftNode != null)
        foreach (BinaryTreeNode<T> leftNode in Traversal(Node.LeftNode))
            yield return leftNode;

    yield return Node;

    if (Node.RightNode != null)
        foreach (BinaryTreeNode<T> rightNode in Traversal(Node.RightNode))
            yield return rightNode;
}

void Insert(T SourceItem)
{
    BinaryTreeNode<T> tempNode = RootNode;
    bool found = false;
    while (!found)
```

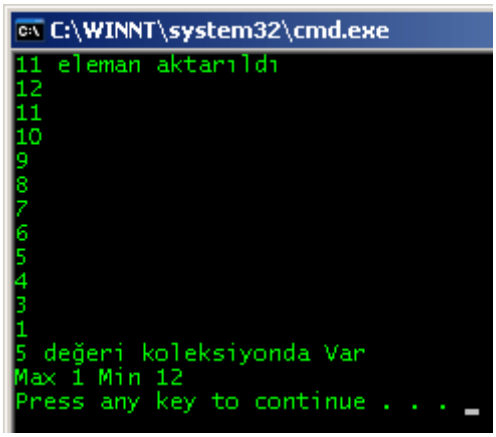
```
        {
            int comparedValue = tempNode.Value.CompareTo(SourceItem);
            if (comparedValue < 0)
            {
                if (tempNode.LeftNode == null)
                {
                    tempNode.LeftNode = new BinaryTreeNode<T>(SourceItem, tempNode);
                    NodeCount++;
                    return;
                }
                else
                {
                    tempNode = tempNode.LeftNode;
                }
            }
            else if (comparedValue > 0)
            {
                if (tempNode.RightNode == null)
                {
                    tempNode.RightNode = new BinaryTreeNode<T>(SourceItem, tempNode);
                    NodeCount++;
                    return;
                }
                else
                {
                    tempNode = tempNode.RightNode;
                }
            }
            else
            {
                return;
            }
        }
    }
}
```

Artık yeni tiplerimizi denemeye çıkabiliriz. Bu amaçla program kodunda aşağıdaki test kodları göz önüne alınabilir.

```
class Program
{
    static void Main(string[] args)
    {
        BinaryTree<int> numbers = new BinaryTree<int>(new BinaryTreeNode<int>(7));
```

```
            numbers.Add(4);
            numbers.Add(9);
            numbers.Add(1);
            numbers.Add(3);
            numbers.Add(10);
            numbers.Add(12);
            numbers.Add(8);
            numbers.Add(5);
            numbers.Add(6);
            numbers.Add(9);
            numbers.Add(11);

            Console.WriteLine("{0} eleman aktarıldı", numbers.Count);

            foreach (int number in numbers)
                Console.WriteLine(number);

            Console.WriteLine("{0} değeri koleksiyonda
{1}",5,numbers.Contains(5)?"Var":"Yok");

            Console.WriteLine("Max {0} Min {1}",numbers.MaxValue,numbers.MinValue);

            int[] array = new int[numbers.Count];
            numbers.CopyTo(array, 0);
        }
    }
```

Temel olarak koleksiyonumuza eklediğimiz temel operasyonlardan bazılarını kullanmaya çalıştık. Uygulamanın **Runtime** çıktısı aşağıdakine benzer olacaktır.



Tabi kodda gözden kaçırdığım bazı noktalar olabilir. Ben elimden geldiğince test etsem de farklı gözlerin bakmasında yarar var. Ancak temel olarak **Binary Tree** veri yapısını, C# ile nasıl kodlanabileceğini kavradığınızı düşünüyorum. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim

BinarySearchTree.zip (42,36 kb)

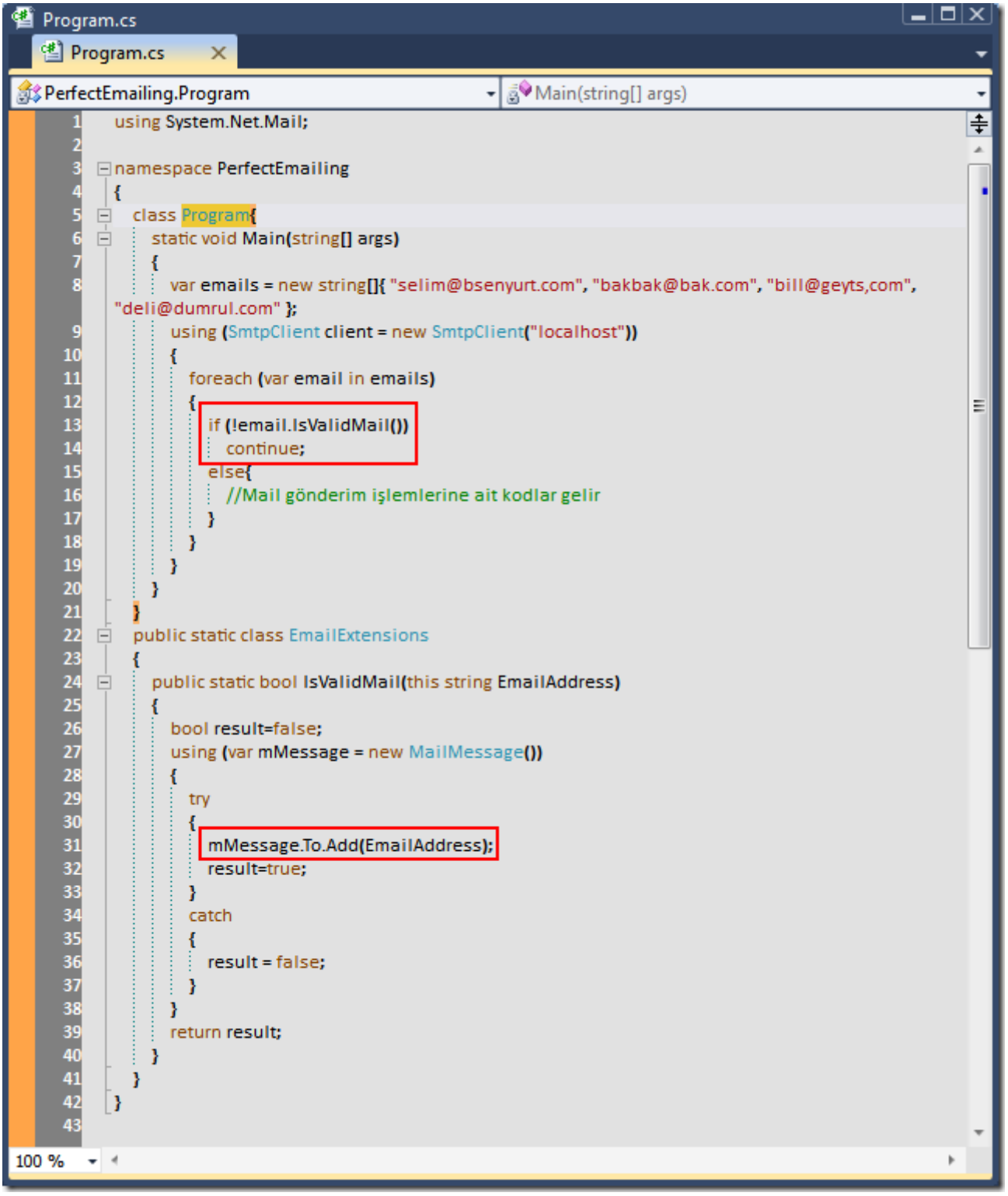
Tek Fotoluk İpucu-44 (Mail Adresi Doğru mu?) (2012-01-02T23:40:00)

c#,smtp,mail,.net,extension methods,

Merhaba Arkadaşlar,

Aslında bu soruya cevap vermek özellikle web developer' lar için son derece kolay. RegularExpressionValidator kontrolünde uygun deseni seçip kontole hatalı mail adresi girilmesi engellenebilir. Ama yine de bazen tedbiri elden bırakmamakta yarar vardır. Söz gelimi bir mail adres listesine toplu mail atacağımız bir senaryoyu göz önüne alalım. Geliştirdiğimiz kodlarda mail adreslerinin doğru olup olmadığını çok basit bir hile ile kontrol edebiliriz. Nasıl mı? 

(Tabi bir yol da RegEx kullanmaktır bildiğiniz üzere. O yolun uygulanış biçimini de size bırakıyorum)




```
using System.Net.Mail;

namespace PerfectEmailing
{
    class Program{
        static void Main(string[] args)
        {
            var emails = new string[]{ "selim@bsenyurt.com", "bakbak@bak.com", "bill@geyts,com",
"deli@dumrul.com" };
            using (SmtpClient client = new SmtpClient("localhost"))
            {
                foreach (var email in emails)
                {
                    if (!email.IsValidMail())
                        continue;
                    else{
                        //Mail gönderim işlemlerine ait kodlar gelir
                    }
                }
            }
        }
    }
    public static class EmailExtensions
    {
        public static bool IsValidMail(this string EmailAddress)
        {
            bool result=false;
            using (var mMessage = new MailMessage())
            {
                try
                {
                    mMessage.To.Add(EmailAddress);
                    result=true;
                }
                catch
                {
                    result = false;
                }
            }
            return result;
        }
    }
}
```

PerfectEmailing.rar (21,55 kb)

ve isteyenler için VS Schema Settings dosyası :) BurakSenyurtVsColorSchema.vssettings (280,59 kb)

T-SQL ile Eğlenmeye Devam(İkinci Devre) (2012-01-01T23:06:00)

t-sql,transact sql,sql server,schema,function,temp table,output,inserted,sp_msforeachdb,sp_executesql,sql functions,

Merhaba Arkadaşlar,

Hatırlayacağınız üzere geçtiğimiz günlerde kafayı **T-SQL** ile bozmuş ve can sıkıntısından eğlenceli ifadeler yazmaya çalışmıştım. Sanırım söz konusu bu eğlence sonraki günlere de sirayet etti ve yine bir kaç eğlenceli **T-SQL** sorgusu ile karşınızdayım *(İnsan ne oldum dememeli ne olacağım demeli belki de…Ben ki SQL' den nefret eden bir birey olarak bu hale geldiysem…*  *)*

Aslında hiç vakit kaybetmeden sorgularımızı incelemeye başlayalım dilerseniz. Elbetteki yine merak ettiğim ve aklıma gelen bazı ihtiyaçlar dahilinde bu sorgular ortaya çıkmakta. örneğin sakin sakin otururken ilk aklıma gelen **T-SQL** tarafında bizim söyleyeceğimiz bazı kriterlere göre rastgele şifre üretecek bir fonksiyon yazmak oldu 

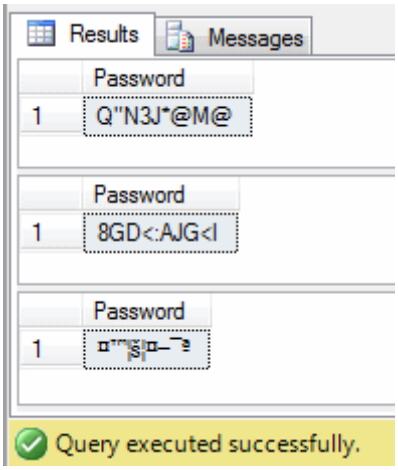
Bunun için aşağıdaki **T-SQL** betiğini yazdım.

```
Use AdventureWorks
Go
-- Function içerisinde Rand() fonksiyonunu kullanamayız(Invalid use of side-effecting or time-dependent operator in 'rand' within a function.) hatası alırız. Bu yüzden bir hile yapacağız ve rastgele sayıyı bir view içerisinden alacağız ;) Steve Kass' ın güzide çözümlerinden birisidir.

create view ViewRandomNumbers
as
  select rand( ) as Number
go

create Function ufnGeneratePassword(
  @PasswordLength int -- Kaç karakterlik password oluşturacağız
  ,@StartChar tinyint -- başlangıç karakterinin ascii karşılığı sayısal değeri
  ,@CharRange tinyint -- Son karakterin ascii karşılığı sayısal değeri
  ,@ExcludedChars varchar(50) -- şifre içerisinde bulunmaması istenen karakterler
  )
returns varchar(50)
as
```

```
begin
  Declare @Password varchar(50)=''
  Declare @char char -- Belirtilen aralıkta üretilen karakteri tutan değişken

  while @PasswordLength> 0 begin
    -- önce @StartChar' dan itibaren @CharRange mesafesine kadarlık bir alan içerisinde rastgele bir char üretilir
    select @char = char(round((Select Number from dbo.ViewRandomNumbers) * @StartChar + @CharRange, 0))
    -- şifrede bulunması istenmeyen karakter olup olmama durumuna göre şifre üretilir ve sayac 1 azaltılır
    if charindex(@char, @ExcludedChars) = 0 begin
      set @Password =@Password+ @char
      set @PasswordLength = @PasswordLength - 1
   end
  end

  return(@Password)

end
Go
```

Burada tanımlamış olduğumuz **ufnGeneratePassword** isimli fonksiyon parametre olarak üretilecek şifre uzunluğunu, bu şifrenin **ASCII** tablosundaki hangi değer aralığında olacağını ve şifre içerisinde olmasını istemediğimiz karakterleri almaktadır. Fonksiyon kendi içerisinde söz konusu **ASCII** değer aralığında bir üretim gerçekleştirmek için de, rastgele sayı üretme işini üstlenen bir **View'** dan yararlanmaktadır. Fonksiyonu aşağıdaki **T-SQL** ifadesinde olduğu gibi kullanıp bir kaç kez test ettiğimizde başarılı sonuçlar elde ettiğimizi görebiliriz.

```
Declare @Password1 nvarchar(10)
Set @Password1= dbo.ufnGeneratePassword(10,65,29,'abcdefg')
Select @Password1 [Password]

Declare @Password2 nvarchar(10)
Set @Password2= dbo.ufnGeneratePassword(10,30,50,'/.+-|@')
Select @Password2 [Password]

Declare @Password3 nvarchar(10)
Set @Password3= dbo.ufnGeneratePassword(10,30,150,'0?*/&^#>é!')
Select @Password3 [Password]
```



Bu ilginç ama bana göre oldukça işe yarayacak **T-SQL** ifadesinden sonra bir başkası ile devam edelim. Söz gelimi veritabanınızda yer alan belirli bir **Şemaya(schema)** ait tablolarınızı yeni bir schema adına taşımak istiyorsunuz*(İstemem demeyin)* Bu durumda ne yaparsınız? Aslında geçtiğimiz yazımızdaki örneklerimizde kullandığımız **system view** nesnelerini göz önüne alırsak; öncelikli olarak ilgili şemadaki tabloları bulmamız ve her biri için dinamik bir **T-SQL** ifadesini yürütmemiz gerektiği ortadadır. Temel olarak şema transferi için aşağıdaki gibi bir **T-SQL** ifadesi kullanılabilir.

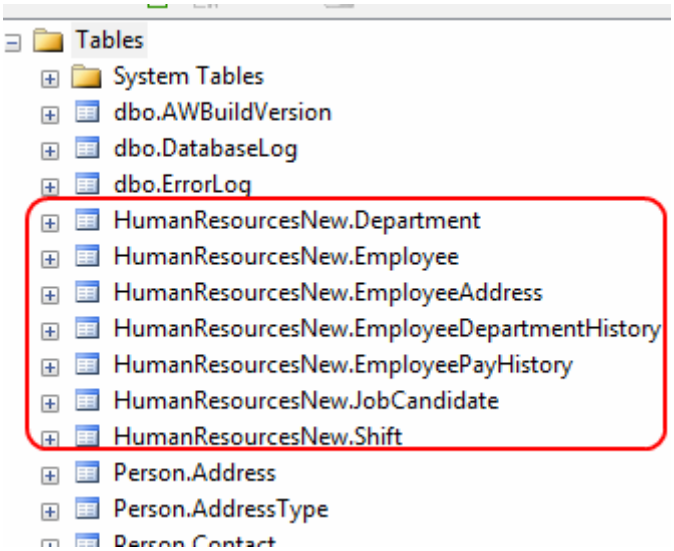
alter schema YeniSchemaAdi transfer [HumanResources].[Employee]

Söz gelimi bu ifade
ile **Employee** tablosunun **HumanResources** şemasından **YeniSchameAdi** şemasına transfer edilmesi sağlanmaktadır. Ancak işi zorlaştıran kısım bu **T-SQL** ifadesinin dinamik olarak oluşturulması ve yürütülmesi sırasında ortaya çıkmaktadır. Dolayısıyla
bir **cursor** kullanımı ve söz konusu şemaya ait tablolar üzerinden dolaşılması, diğer yandan her bir tablo için ilgili şema transfer işini üstlenen **T-SQL** ifadesinin dinamik olarak oluşturulması ve bu ifadeninde dinamik olarak çalıştırılması gerekmektedir. Aynen aşağıdaki T-SQL betik bloğunda görüldüğü gibi

```
-- önce yeni bir schema üretelim
create schema HumanResourcesNew
go

declare @NewSchemaName sysname
declare @CursorObject sysname
declare @SqlExpression nvarchar(1000)
set @NewSchemaName = quotename('HumanResourcesNew')

-- sys.objects içerisinde dolaşıp HumanResources şemasına ait tüm kullanıcı tanımlı tabloları dolaşacak bir Cursor açıyoruz
declare crsr cursor for select quotename([name])from sys.objects where schema_id = schema_id('HumanResources') and type in ('U')
```

```
open crsr
fetch from crsr into @CursorObject

   while @@fetch_status=0 begin
      --her bir tablo için gerekli şema transfer etme T-SQL ifadesini üretiyoruz
      set @SqlExpression = 'alter schema '+@NewSchemaName+'
transfer [HumanResources].'+@CursorObject
      print @SqlExpression
      -- üretilen T-SQL ifadesini sp_executeSQL Stored Procedure' ü yardımıyla
çalıştırıyoruz
      exec sp_executeSQL @SqlExpression
   fetch next from crsr into @CursorObject
end

close crsr
deallocate crsr
```

bu T-SQL ifadesini yürüttüğümüzde HumanResources şemasındaki kullanıcı tanımlı tabloların, HumanResourcesNew şemasına taşındığını görürüz.



Sizi bilmem ama ben çok eğleniyorum. Hız kesmeden farklı bir **T-SQL** ifadesi ve ihtiyacı ile devam edelim. Bu kez merak ettiğim şuydu. Acaba sistemimde ki veritabanlarında yer alan toplam kullanıcı tanımlı tablo sayıları ne kadardı?

Tüm veritabanlarını gezmek için yine sistem **SP**' lerinden birisi olan sp_**MSforeachdb'** den yararlanabilirdim. Hatta daha önceden yaptığımız gibi bir **temp** tablo kullanıp tüm sonuçları buraya da aktarabilirdim. Hımm…Beni biraz uğraştıran bir sorgu oldu aslında. Nitekim toplam tablo sayısını bulmak için öncelikli olarak her bir veritabanı bağlantısı altında çalıştırılacak **T-SQL** ifadeleri gerekiyordu. Bir başka deyişle yine dinamik olarak üretilecek ve her bir veritabanı için çalıştırılacak bir **T-**

SQL ifadesi söz konusuydu Ancak biraz uğraştıktan ve epey bir hata aldıktan sonra aşağıdaki **T-SQL** sorgusunu yazmayı başarabilmiştim.

-- önce veritabanı adı ve buradaki toplam tablo sayısını tutacak olan Temp tabloyu üretelim
Create Table #AllTables
(
  DbName varchar(50)
  ,TableCount int
)

--sp_Msforeachdb SP' sinden yararlanarak tüm veritabanlarını dolaşalım
EXEC sp_MSforeachdb '
USE
?

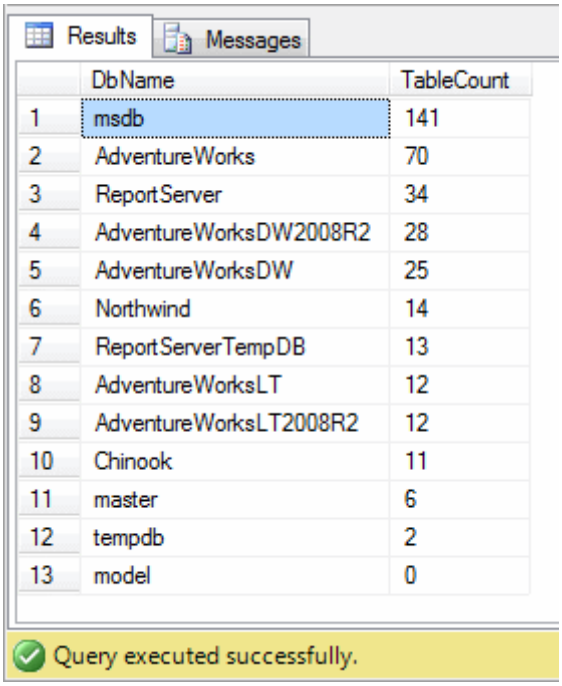
Declare @TableCount int
Set @TableCount=(Select Count(name) from sys.objects where type=''U'')

Insert into #AllTables Values (''?'',@TableCount)
' -- Her bir veritabanı için USE ile o veritabanı alanına geçiyor ve sys.objects' den yararlanarak toplam tablo sayılarını bulup @TableCount isimli değişkende tuttuğumuz bu sayıları ve güncel veritabanı adını insert sorgusu ile temp tabloya alıyoruz

Select * from #AllTables order by TableCount desc

Drop Table #AllTables

ve işte sonuçlar



Results | Messages

	DbName	TableCount
1	msdb	141
2	AdventureWorks	70
3	ReportServer	34
4	AdventureWorksDW2008R2	28
5	AdventureWorksDW	25
6	Northwind	14
7	ReportServerTempDB	13
8	AdventureWorksLT	12
9	AdventureWorksLT2008R2	12
10	Chinook	11
11	master	6
12	tempdb	2
13	model	0

Query executed successfully.

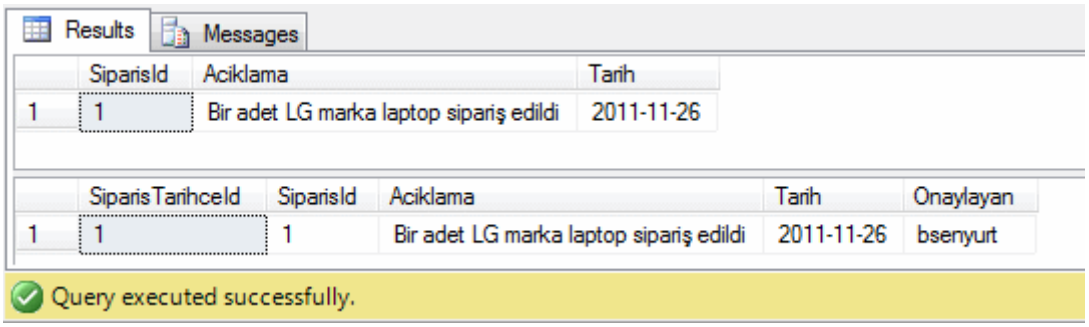
Merak ettiğim bir diğer konu ise **Insert** işlemlerine ilişkindi. Bazı hallerde bir **Insert** işlemi gerçekleştirildiğinde, insert edilen verilerin başka bir tabloya da aktarılması istenebilir. Söz gelimi bir tablo için gerçekleştirilen **Insert** işlemi sırasında, **History** bilgisini tutan başka bir tabloya da veri aktarımı yapılması sırasında... Burada aslında **output** anahtar kelimesi ve **Inserted**elemanının kullanıldığı bir ifade dizimi söz konusudur. çoğumuz Insert işlemini bu tip bir şekilde çok fazla kullanmamışızdır eminim ki Senaryo gereği **OziRestoran** isimli bir veritabanı oluşturup üzerine **Siparis** ve **SiparisTarihce** isimli iki tablo ekledim.

```
Create database OziRestoran
go

Create table Siparis
(
   SiparisId int identity(1,1) primary key
   ,Aciklama nvarchar(250)
   ,Tarih date
)
Create table SiparisTarihce
(
   SiparisTarihceId int identity(1,1) primary key
   ,SiparisId int
   ,Aciklama nvarchar(250)
   ,Tarih date
   ,Onaylayan nvarchar(20)
)
Go
```

Insert işlemimizde şunu yapmak istediğimizi düşünelim ; **Siparis** tablosuna bir satır eklenirken, üretilen otomatik **SiparisId**, **Aciklama** ve **Tarih** alanları değerlerinin, siparisi onaylayan kişi bilgisi ile birlikte tarihçe tablosuna yazdırılmasını istiyoruz İşte **Insert** sorgumuz.

```
Use OziRestoran
Go
insert into Siparis( Aciklama, Tarih )
output inserted.SiparisId, inserted.Aciklama,inserted.Tarih,'bsenyurt'
   into SiparisTarihce
  (
      SiparisId,
      Aciklama,
      Tarih,
      Onaylayan
   )
values ( 'Bir adet LG marka laptop sipariş edildi',GETDATE())
Go

Select * from Siparis
Select * from SiparisTarihce
```

Görüldüğü üzere **Insert** ifadesi yazılırken **output** anahtar kelimesinden itibaren **SiparisTarihce** içerisine de veri aktarımının yapılacağı belirtilmektedir. Sonrasında **values** anahtar kelimesini takip eden kısımda, asıl **Siparis** tablosu için eklenecek içerik set edilmektedir. Sonuçlar aşağıdaki gibi olacaktır.

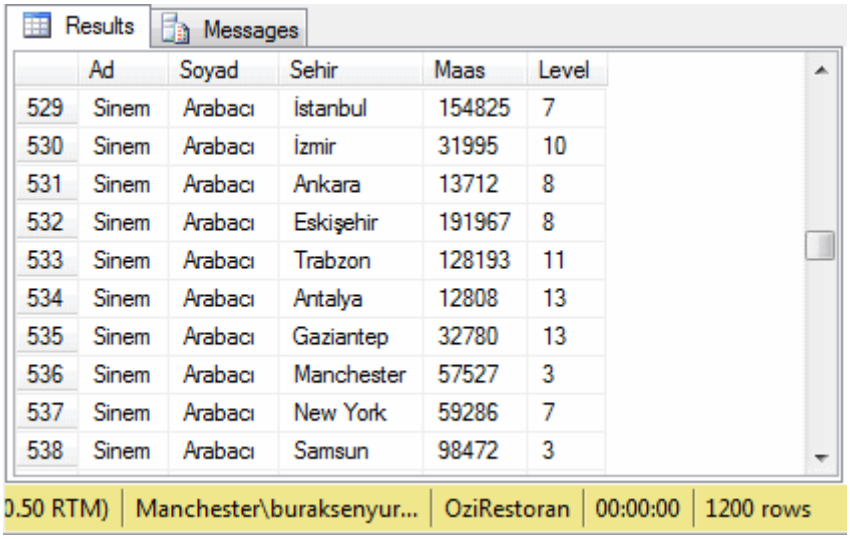


Hazır **Insert** işlemlerinden konu açılmışken acaba içeriğini rastgele test verisi ile dolduracağımız devasa boyutlu tabloları nasıl oluşturabiliriz sorusu aklıma geldi Aslında bu konuda bir önceki çalıştığım firmada **Database Developer** arkadaşlarımın yaptığı önemli çalışmalar vardı. Milyonlarca anlamlı veri yığını oluşturuyorlardı. Onların eline su dökemem belki ama en azından kendi çapımda bir şeyler yapabilirim diye düşündüm. İşe basit bir senaryo ile başladım. örneğin rastgele **Ad,Sodad,Şehir,Maaş** ve**Seviye** bilgilerinden oluşacak bir veri tablosunu üretmeye çalıştım. Bu amaçla aşağıdaki gibi bir sorgu oluşturdum.

```
Create Table Adlar
(
   Ad nvarchar(50)
)
Create Table Soyadlar
(
   Soyad nvarchar(50)
)
Create Table Sehirler
(
   Sehir nvarchar(50)
)

Go
Insert into Adlar values ('Burak')
Insert into Adlar values ('Kamil')
Insert into Adlar values ('Burcu')
Insert into Adlar values ('Elif')
Insert into Adlar values ('Sinem')
Insert into Adlar values ('Hakan')
Insert into Adlar values ('Bill')
Insert into Adlar values ('Murat')
Insert into Adlar values ('Nazım')
Insert into Adlar values ('Cansu')

Insert into Soyadlar values ('Şenyurt')
Insert into Soyadlar values ('Kırmızı')
Insert into Soyadlar values ('Sucu')
Insert into Soyadlar values ('Salimoğlu')
Insert into Soyadlar values ('Arabacı')
Insert into Soyadlar values ('Kısakol')
Insert into Soyadlar values ('Odabaşı')
Insert into Soyadlar values ('Şamil')
Insert into Soyadlar values ('Limoncu')
Insert into Soyadlar values ('Kurtaran')

Insert into Sehirler values ('İstanbul')
Insert into Sehirler values ('İzmir')
Insert into Sehirler values ('Ankara')
Insert into Sehirler values ('Eskişehir')
Insert into Sehirler values ('Trabzon')
Insert into Sehirler values ('Antalya')
Insert into Sehirler values ('Gaziantep')
Insert into Sehirler values ('Manchester')
Insert into Sehirler values ('New York')
```

```
Insert into Sehirler values ('Samsun')
Insert into Sehirler values ('Aydın')
Insert into Sehirler values ('Moskova')

select
   Ad
   ,Soyad
   ,Sehir
   ,Maas=ROUND(ABS(CHECKSUM(NEWID()))/10000,0)
   ,Level=ABS(CHECKSUM(NewId())) % 14
   into PersonelTestTable FROM Adlar
      cross join Soyadlar
      cross join Sehirler

Select * From PersonelTestTable
```

Şimdi burada işin püf noktası **Adlar, Soyadlar** ve **Sehirler** tablolarının **CROSS JOIN** ile birleştirilmesi ve **PersonelTestTable** içerisine atılması işlemidir. çok doğal olarak ortaya **1200**satırlık **(10 Ad X 10 Soyad X 12 Şehir)** veri kümesi çıkacaktır



Eğer kombinasyon sayısını arttırırsanız kısa sürede milyonlarca satırdan oluşabilecek devasa test verileri üretebilirsiniz. örneği geliştirmek sizin elinizde

Böylece geldik bir yazımızın daha sonuna. Bu yazımızda sadece **5** çeşit **T-SQL** ifadesine değindik ancak inanıyorum ki ilerleyen zamanlarda bunlara yenilerini ekliyor olacağım. çünkü bu iş çok eğlenceli olmaya başladı. Merak işte Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

EglencelikSQL_2.sql (11,65 kb)

Entity Framework ile Gerçek Hayat Örnekleri 2 Webineri (2011-12-22T17:30:00)

entity framework,nedirtv?com,wcf,windows communication foundation,ef 4.0,

Merhaba Arkadaşlar,

Hatırlayacağınız üzere bir önceki bölümümüzde, **Entity Framework** tabanlı **Data-Centric** çözümümüzü(Solution) geliştirmeye başlamış ve çok basit olarak çatıyı kurduktan sonra, iş katmanına(Business Layer) ait bir operasyonu örneklemiştik. Ardından bu operasyonu bir **Unit Test** metodu ile denemeye tabi tutarak Webinerimizi sonlandırmıştık.

Serimizin ikinci bölümünde ise projemizi kalan parçaları ile birlikte tamamlıyor ve servis tarafından istemcilere nasıl açabileceğimizi irdelemeye çalışıyoruz. İlk olarak Business Layer tarafında gerekli fonksiyonellikleri geliştirip her biri için ilgili Unit Test işlemlerini aştıktan sonra,**WCF(Windows Communication Foundation)** tarafında ilgili operasyonları tanımlıyoruz. Son olarakta servisimizi basit bir istemci*(Console projesi kullandık)* üzerinden deniyoruz.

Webinerimiz sonunda Camtasia ile gerçekleştirdiğimiz ekran kayıtları teknik bir arıza nedeniyle patladı Ancak takipçilerimizden sevgili arkadaşım **İsmail Burak Yücel** Webineri kendi bilgisayarı üzerinde yine Camtasia ile başarılı bir şekilde kaydetmiş. Onun elindeki kayıtları kullanarak gerekli render işlemlerini yapabildim. Kendisine çok teşekkür ediyorum İkinci Webinerimize her zamanki gibi **Nedirtv?com**üzerinden erişebilirsiniz.

Entity Framework ile Gerçek Hayat örnekleri 2 (1 Saat, 85.6Mb)

örnek Solution Dosyamız RealEF.rar (337,57 kb)

T-SQL ile Dinlenmece Eğlenmece (2011-12-15T01:25:00)

sql,sys.objects,sys.tables,sys.columns,stored procedures,newid,sys.servers,transact sql,cursor,database,adventure works,sql server 2008 r2,temp table,

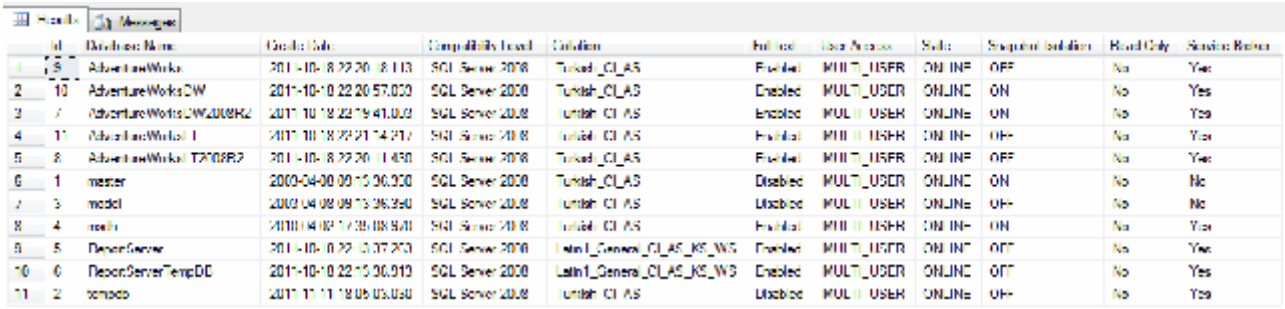
Merhaba Arkadaşlar,

Hiç canınızın sıkıldığı ve böyle bir buhran anına girdiğinizde **SQL Server Management Studio**' yu açıp **T-SQL** ile eğlenceli bir şeyler yapmaya çalıştığınız oldu mu? 

Açıkçası geçtiğimiz günlerde böyle sıkkın ve bıkkın bir ruh halindeyken ve konuşmak istediğim tüm arkadaşlarım yoğunken, ekranımda duran **Management Studio**' daki bembeyaz ve bomboş **Query** penceresi ile muhabbet etmeye karar verdim. Aslında amacım basitti. Daha önceki tecrübelerime dayanarak ihtiyaçlar dahilinde kullandığım **T-SQL** ifadelerini şöyle bir tekrar etmeye çalışacak ve siz değerli okurlarıma bir blog girdisi olarak sunacaktım. Aklıma geldikçe ihtiyaçlarımın **T-SQL** karşılıklarını yazmaya başladım. Düşündüğüm ilk gereksinim, sistemimde yüklü olan kaç veritabanı olduğunu ve bunlara ait bazı temel bilgileri edinmekti...İşte serüvenimiz bu ilk sorgumuz ile başlıyor.

```
select
   database_id [Id]
   ,name [Database Name]
   ,create_date [Create Date]
   ,Case [compatibility_level]
      when '60' then 'SQL Server 6.0'
      when '65' then 'SQL Server 6.5'
      when '70' then 'SQL Server 7.0'
      when '80' then 'SQL Server 2000'
      when '90' then 'SQL Server 2005'
      when '100' then 'SQL Server 2008'
      else 'unknown'
   end as [Compatibility Level]
   ,collation_name [Collation]
   ,Case is_fulltext_enabled
      when 1 then 'Enabled'
      else 'Disabled'
   end as [FullText]
   ,user_access_desc [User Access]
   ,state_desc [State]
   ,snapshot_isolation_state_desc [Snapshot Isolation]
   ,Case is_read_only
      when 1 then 'Yes'
      else 'No'
   end as [Read Only]
   ,Case is_broker_enabled
      when 1 then 'Yes'
      else 'No'
   end as [Service Broker]
from sys.databases
order by [Database Name]
```

Yukarıdaki SQL sorgusunu kullanarak sistemde var olan veritabanlarına ait bazı temel bilgileri elde edebiliriz. Söz konusu ifadenin çalışma zamanındaki çıktısı aşağıdakine benzer olabilir. Bu sonuçlada pek tabi benim sistemimde yer alan veritabanları ve onlara ait bilgileri bulunmaktadır. *(Ekran çıktısının orjinal halini görmek içim fotoğrafa tıklayın)*

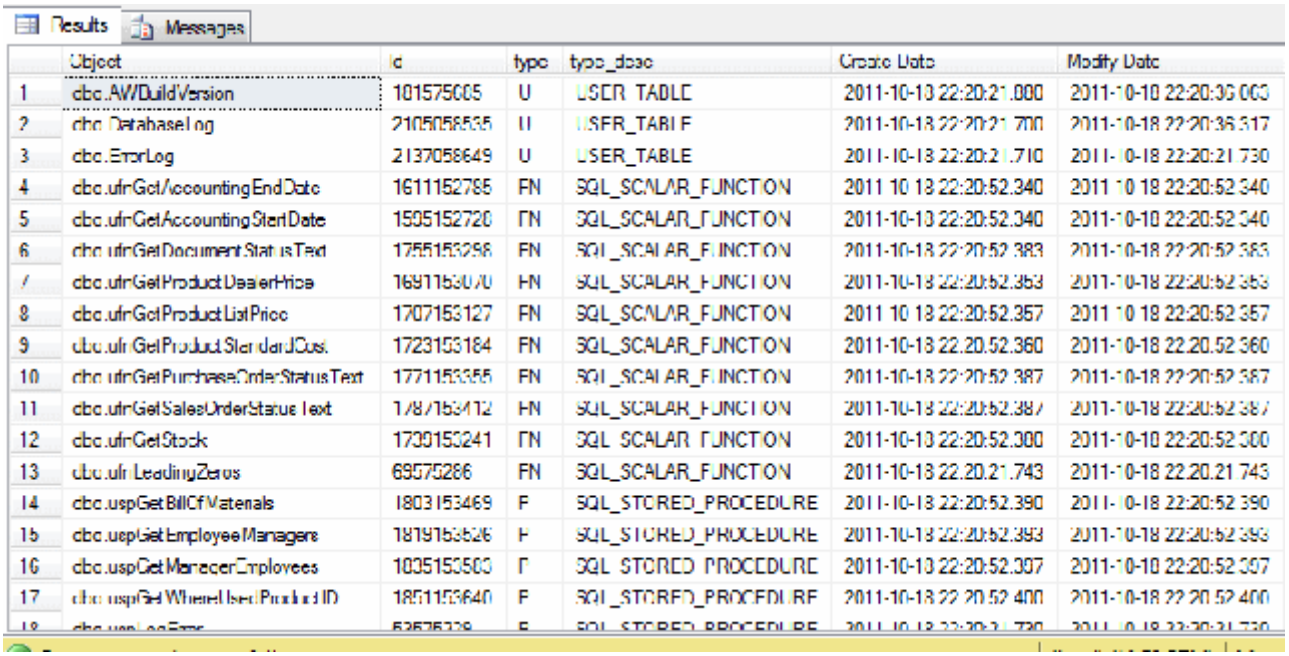


Görüldüğü üzere sistemimde var olan veritabanlarının adlarını, oluşturulma zamanlarını, SQL Server uyumluluk sürümlerini, hangi Collation' ı kullandıklarını ve bunlara benzer bilgilerini elde etmiş bulunuyoruz*(Doğruyu söylemek gerekirse SQL tarafında sys ön eki ile başlayan View' lar içerisinde inanılmaz sürprizler bulumaktadır. İncelemediyseniz bile araştırmanızı şiddetle öneririm)*

Gelelim sıradaki sorgumuza. Bu sefer sistemimde kullanıcı tanımlı ne kadar **table, stored procedure, function, view** ve **trigger** varsa, **şema adları(Schema Name)** ile birlikte elde etmek istedim. Bunun içinde yine sys ön ekli view' lardan yararlanabiliriz. İşte sorgumuz;

```
select
   S.name+'.'+O.name [Object]
   ,object_id [Id]
   ,type
   ,type_desc
   ,create_date [Create Date]
   ,modify_date [Modify Date]
from sys.all_objects O
join sys.schemas S on O.schema_id=S.schema_id
where type in ('U','V','TR','FN','P')
order by [Object]
```

Dikkat edileceği üzere **sys.schema** ve **sys.all_objects** isimli sistem görünümlerinden yararlanmaktayız. **sys.all_objects** tüm veritabanı nesnelerini tutan bir görünüm sunmaktadır. Aslında sadece belirli bir veritabanı bağlantısı ile ilişkili olan nesnelere gitmek istersek **sys.objects** görünümünden de yararlanabiliriz. Söz konusu **T-SQL** ifademizin benim sistemimde ürettiği sonuçlar ise aşağıdaki gibidir.

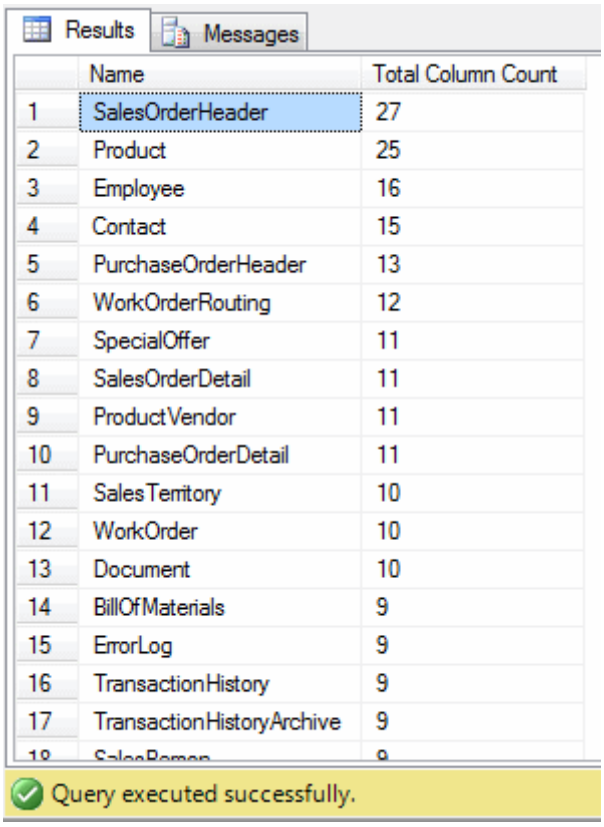


	Object	id	type	type_desc	Create Date	Modify Date
1	dbo.AWBuildVersion	101575005	U	USER_TABLE	2011-10-13 22:20:21.000	2011-10-13 22:20:36.003
2	dbo.DatabaseLog	2105058535	U	USER_TABLE	2011-10-13 22:20:21.700	2011-10-13 22:20:36.317
3	dbo.ErrorLog	2137058649	U	USER_TABLE	2011-10-13 22:20:21.710	2011-10-13 22:20:21.730
4	dbo.ufnGetAccountingEndDate	1611152785	FN	SQL_SCALAR_FUNCTION	2011-10-13 22:20:52.340	2011-10-13 22:20:52.340
5	dbo.ufnGetAccountingStartDate	1595152728	FN	SQL_SCALAR_FUNCTION	2011-10-13 22:20:52.340	2011-10-13 22:20:52.340
6	dbo.ufnGetDocumentStatusText	1755153298	FN	SQL_SCALAR_FUNCTION	2011-10-13 22:20:52.383	2011-10-13 22:20:52.383
7	dbo.ufnGetProductDealerPrice	1691153070	FN	SQL_SCALAR_FUNCTION	2011-10-13 22:20:52.353	2011-10-13 22:20:52.353
8	dbo.ufnGetProductListPrice	1707153127	FN	SQL_SCALAR_FUNCTION	2011-10-13 22:20:52.357	2011-10-13 22:20:52.357
9	dbo.ufnGetProductStandardCost	1723153184	FN	SQL_SCALAR_FUNCTION	2011-10-13 22:20:52.360	2011-10-13 22:20:52.360
10	dbo.ufnGetPurchaseOrderStatusText	1771153355	FN	SQL_SCALAR_FUNCTION	2011-10-13 22:20:52.387	2011-10-13 22:20:52.387
11	dbo.ufnGetSalesOrderStatusText	1787153412	FN	SQL_SCALAR_FUNCTION	2011-10-13 22:20:52.387	2011-10-13 22:20:52.387
12	dbo.ufnGetStock	1739153241	FN	SQL_SCALAR_FUNCTION	2011-10-13 22:20:52.300	2011-10-13 22:20:52.300
13	dbo.ufnLeadingZeros	69575286	FN	SQL_SCALAR_FUNCTION	2011-10-13 22:20:21.743	2011-10-13 22:20:21.743
14	dbo.uspGetBillOfMaterials	1803153469	P	SQL_STORED_PROCEDURE	2011-10-13 22:20:52.390	2011-10-13 22:20:52.390
15	dbo.uspGetEmployeeManagers	1819153526	P	SQL_STORED_PROCEDURE	2011-10-13 22:20:52.393	2011-10-13 22:20:52.393
16	dbo.uspGetManagerEmployees	1835153583	P	SQL_STORED_PROCEDURE	2011-10-13 22:20:52.397	2011-10-13 22:20:52.397
17	dbo.uspGetWhereUsedProductID	1851153640	P	SQL_STORED_PROCEDURE	2011-10-13 22:20:52.400	2011-10-13 22:20:52.400
18	dbo.uspLogError	53575229	P	SQL_STORED_PROCEDURE	2011-10-13 22:20:21.730	2011-10-13 22:20:21.730

Eğlenceli değil mi? O zaman hız kesmeden devam edelim. çalıştığım sırada aklıma gelen ve merak ettiğim sorgulardan birisi de şuydu : Acaba sistemimde yer alan tablolarda kullanılan**alanların(Columns)** tablo bazlı toplam sayıları neydi? çok doğal olarak hangi tabloda kaç alan kullanıldığını bilmek istiyordum. Bunun için basit bir SQL ifadesi yeterli olacaktı. Aynen aşağıdaki gibi.

```
select
   T.Name
   ,Count(C.column_id) [Total Column Count]
   from sys.tables T
   join sys.columns C on T.object_id=C.object_id
   where T.type='U'
group by T.Name
order by Count(C.Column_id) desc
```

Bu kez **sys.tables** ve **sys.columns** view nesnelerini ele alıp ve kullanıcı tanımlı tabloları adlarına göre gruplandırarak sonuca ulaşmaya çalıştım. Tabi kendi sistemimde bu tabloyu yürüttüğümde aşağıdaki ekran çıktısında yer alan sonuçları elde ettim.



Results | Messages

	Name	Total Column Count
1	SalesOrderHeader	27
2	Product	25
3	Employee	16
4	Contact	15
5	PurchaseOrderHeader	13
6	WorkOrderRouting	12
7	SpecialOffer	11
8	SalesOrderDetail	11
9	ProductVendor	11
10	PurchaseOrderDetail	11
11	SalesTerritory	10
12	WorkOrder	10
13	Document	10
14	BillOfMaterials	9
15	ErrorLog	9
16	TransactionHistory	9
17	TransactionHistoryArchive	9

Query executed successfully.

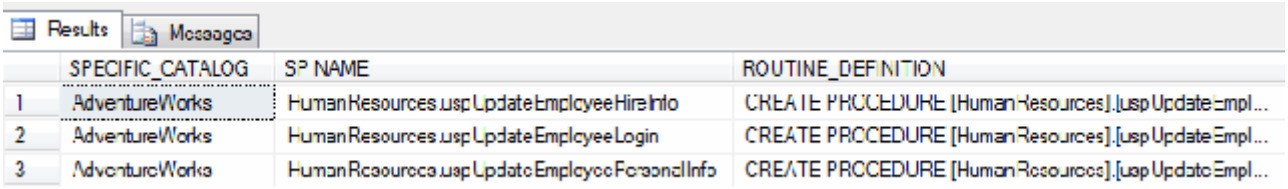
Tabi kendi sistemimde gayet makul seviyelerde rakamlara ulaştığımı ifade edebilirim. Nasıl ki kod yazarken bazı metrikleri uyguluyor ve örneğin satır sayısı 25i geçen metodları tespit edip kod standartları açısından denetlemeler yapıyoruz, benzer şekilde SQL tarafında da bu tip metrikleri uygulayabiliriz. Bu sorgu söz konusu metriklerden birisi olarak düşünülebilir. Tabi çalışmakta olduğum bankada aynı sorguyu denediğim de piuvvvvv Ehem…Ehem…Tekrar sistemime döneyim.

Bu kez aklımda şöyle bir soru vardı : Acaba sistemimde yer alan **AdventureWorks** veritabanında, hangi **Stored Procedure**' ler içerisinde **Update** anahtar kelimesi(Keyword) kullanılmaktaydı Bir başka deyişle hangi SP' ler içerisinde güncelleme ile ilişkili işlemler yapıldığını görmek istiyordum. Bu tip bir ihtiyaç pek çok durumda gerekebilir. özellikle SQL tarafına yıkılmış iş süreçlerinde değişiklikler yapmanız gerektiği durumlarda kullanabileceğiniz bir tespit yöntemidir. Söz gelimi bir tablonun adının değişmesi sonucu ilgili SP' lerde de geçtiği yerlerde de pansumanlar yapmak gerekecektir*(SQL Tarafında Visual Studio' da olduğu gibi Refactor-Rename özelliği olsaydı fena olmazı aslında)*Bunu öğrenmek için aşağıdaki sorguyu kullandım.

```
Use AdventureWorks
Go
select
   SPECIFIC_CATALOG
   ,SPECIFIC_SCHEMA+'.'+SPECIFIC_NAME [SP NAME]
```

```
    ,ROUTINE_DEFINITION
from INFORMATION_SCHEMA.ROUTINES
where ROUTINE_TYPE='PROCEDURE' and ROUTINE_DEFINITION like
'%UPDATE%'
```

Görüldüğü üzere farklı bir **View** içerisinde **Stored Procedure**' lerin **T-SQL** içerikleri de tutulmaktadır. Sorgunun sonucu olarak aşağıdaki ekran görüntüsünde yer alan çıktıları elde ettim. *(Size bir antrenman önerebilirim. Eğer çok sayıda veritabanı ve çok sayıda SP ile karmaşık iş süreçlerini barındıran bir sistemde görev alıyorsanız, örneğin içerisinde **@@IDENTITY, BEGIN TRANSACTION** gibi kritik terimleri içeren SP' leri araştırmayı deneyebilirisiniz)*

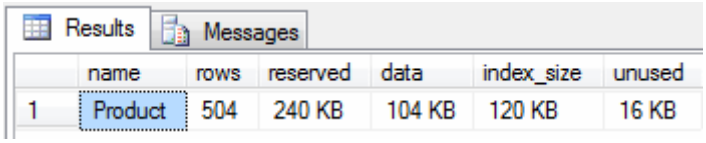


	SPECIFIC_CATALOG	SP NAME	ROUTINE_DEFINITION
1	AdventureWorks	HumanResources.uspUpdateEmployeeHireInfo	CREATE PROCEDURE [HumanResources].[uspUpdateEmpl...
2	AdventureWorks	HumanResources.uspUpdateEmployeeLogin	CREATE PROCEDURE [HumanResources].[uspUpdateEmpl...
3	AdventureWorks	HumanResources.uspUpdateEmployeePersonalInfo	CREATE PROCEDURE [HumanResources].[uspUpdateEmpl...

Aklıma sorgu geldikçe geliyordu. Karşımdaki **SQL Query** pencersi iyi bir arkadaştım. Ne sorsam cevap veriyordu*(Yani sayılır)*. Gerçi bazen ırım kırım ediyor naz yapıyordu ama olsun Şimdi merak ettiğim sorgu ise şuydu : Acaba bir veritabanında veya daha da iyisi tüm sistemde yer alan tabloların kapladıkları alanların boyut bilgileri nelerdi? Burada çözüme giderken biraz sıkıntı çektiğimi ifade etmek isterim. **Query** pencersi ile bir türlü mutabakat sağlayamadık. önce tek ve herhangibir tablo için bunu öğrenmeye çalıştım. Bunun için tasarlanmış özel bir sistem SP' si mevcuttu nitekim*(Sanırım Tüme varım yöntemi ile hareket edeceğim)*

```
Use AdventureWorks
Go
sp_spaceused 'Production.Product'
```

Sonuç ise şöyleydi.

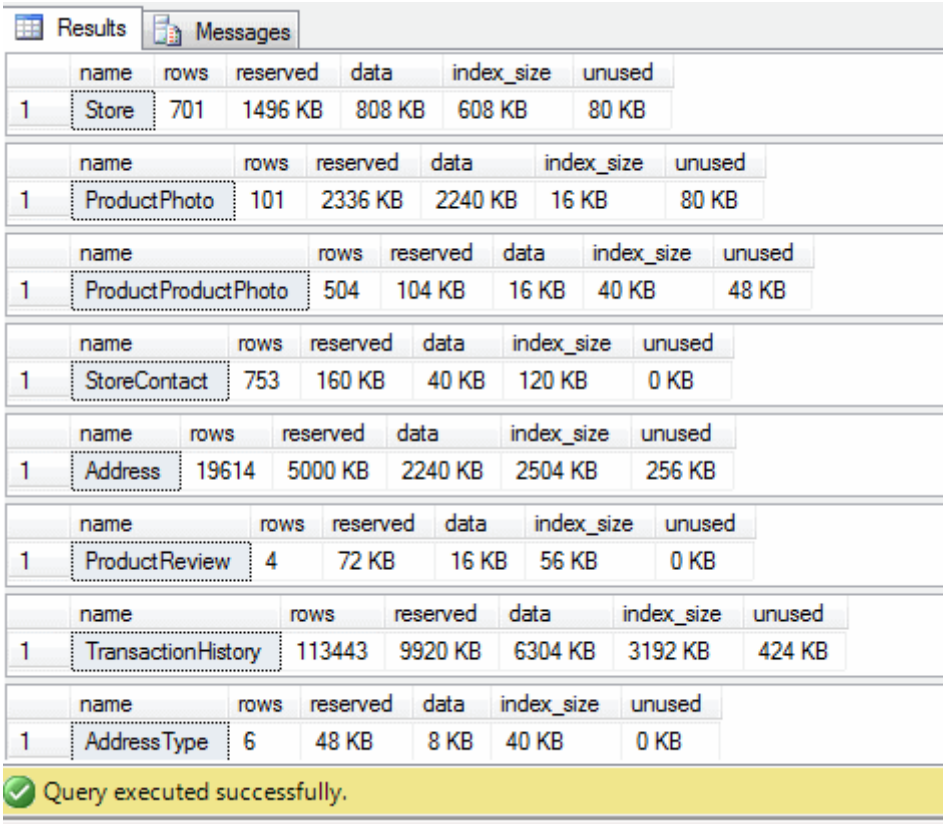


	name	rows	reserved	data	index_size	unused
1	Product	504	240 KB	104 KB	120 KB	16 KB

Şimdi işi bir adım daha ileri götürmeliydim. çünkü asıl amacım sistemde ne kadar tablo varsa her birinin boyutsal özelliklerini öğrenmekti*(Yani ne kadar alanı reserve ettikleri, bu alanın ne kadarını kullandıkları vb)* Bunun içinde aslında bir **for each** ifadesini çalıştırmam gerekiyordu. Yani her bir tabloyu gezmeli ve her biri için **sp_spaceused** SP' sini çalıştırmalıydım. Bu **foreach** içinde aslında sistem de yer alan güzel bir SP bulunmaktadır.

```
EXEC sp_MSforeachtable @command1="EXEC sp_spaceused '?'"
```

ve işte sonuç ;



Results | Messages

	name	rows	reserved	data	index_size	unused
1	Store	701	1496 KB	808 KB	608 KB	80 KB

	name	rows	reserved	data	index_size	unused
1	ProductPhoto	101	2336 KB	2240 KB	16 KB	80 KB

	name	rows	reserved	data	index_size	unused
1	ProductProductPhoto	504	104 KB	16 KB	40 KB	48 KB

	name	rows	reserved	data	index_size	unused
1	StoreContact	753	160 KB	40 KB	120 KB	0 KB

	name	rows	reserved	data	index_size	unused
1	Address	19614	5000 KB	2240 KB	2504 KB	256 KB

	name	rows	reserved	data	index_size	unused
1	ProductReview	4	72 KB	16 KB	56 KB	0 KB

	name	rows	reserved	data	index_size	unused
1	TransactionHistory	113443	9920 KB	6304 KB	3192 KB	424 KB

	name	rows	reserved	data	index_size	unused
1	AddressType	6	48 KB	8 KB	40 KB	0 KB

Query executed successfully.

Aslına bakarsanız istediğim bilgileri elde etmiştim. Ancak görüntü pek hoş değildi. Keşke tablo bazlı bir **ızgara çıktısı(Grid View)** elde edebilseydim Ama çaresiz değildim. Biraz **Cursor,** biraz **Temp** tablo işimi görebilirdi pekala. Kolları sıvadım ve uzun bir uğraştan sonra aşağıdaki SQL ifadesini yazmayı başardım.

```
Use AdventureWorks
Go
declare @TableName nvarchar(100)
create table #TempTable
(
   [Table Name] nvarchar(100),
   [Row Count] varchar(100),
   [Reserved Size] varchar(50),
   [Data Size] varchar(50),
   [Index Size] varchar(50),
   [Unused Size] varchar(50)
)

declare tableCursor cursor forward_only
for
   select S.name+'.'+T.[name]
   from sys.tables T
```

```
    join sys.schemas S on T.Schema_id=S.Schema_id
    where T.type='U'
for read only

open tableCursor
    while (1=1)
    begin
        fetch next from tableCursor into @TableName
            if(@@FETCH_STATUS<>0)
                break;
            insert #TempTable exec sp_spaceused @TableName
    end

close tableCursor
deallocate tableCursor

select * from #TempTable Order by [Table Name]
drop table #TempTable
```

Aslında teori basitti. Tablo ve Şema adlarını elde ettikten sonra her birisi için **sp_spaceused** SP' ini çalıştıracak ama sonuçlarını bir **Temp** tabloya ekleyecektim. Şimdi sonuçlar ve elde edilen görüntü çok daha güzeldi.

Results | Messages

	Table Name	Row Count	Reserved Size	Data Size	Index Size	Unused Size
1	Address	19614	5000 KB	2240 KB	2504 KB	256 KB
2	AddressType	6	48 KB	8 KB	40 KB	0 KB
3	AWBuildVersion	1	16 KB	8 KB	8 KB	0 KB
4	BillOfMaterials	2679	472 KB	160 KB	200 KB	112 KB
5	Contact	19972	6888 KB	4576 KB	2000 KB	312 KB
6	ContactCreditCard	19118	584 KB	480 KB	16 KB	88 KB
7	ContactType	20	32 KB	8 KB	24 KB	0 KB
8	CountryRegion	238	48 KB	16 KB	32 KB	0 KB
9	CountryRegionCurrency	109	32 KB	8 KB	24 KB	0 KB
10	CreditCard	19118	2448 KB	1496 KB	776 KB	176 KB
11	Culture	8	32 KB	8 KB	24 KB	0 KB
12	Currency	105	32 KB	8 KB	24 KB	0 KB
13	CurrencyRate	13532	1240 KB	768 KB	400 KB	72 KB
14	Customer	19185	2336 KB	824 KB	1176 KB	336 KB
15	CustomerAddress	19220	1616 KB	864 KB	552 KB	200 KB
16	DatabaseLog	1566	6136 KB	6056 KB	48 KB	32 KB
17	Department	16	32 KB	8 KB	24 KB	0 KB
18	Document	9	368 KB	344 KB	24 KB	0 KB

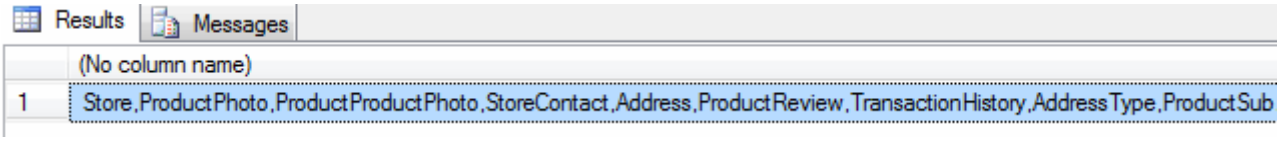
Query executed successfully.

Tam bu sorguyu bitirmiştim ki aklıma başka bir ihtiyaç geldi. Acaba sistemde yer alan tablo adlarının tamamını, aralarına virgül koyarak tek bir hücreye indirgiyebilir miydim?

Hımm…Eğer kod tarafında olsaydık bu benim çocuk oyuncağı sayılırdı. Ama SQL özürlü birisi olarak biraz araştırma yapmam gerekecekti. Sonuçta **COALESCE** fonksiyonundan yararlanarak bu isteği karşılayabileceğimi gördüm. Nasıl mı?

```
DECLARE @Names VARCHAR(8000)
SELECT @Names = COALESCE(COALESCE(@Names + ',', '') + Name, @Names)
   FROM sys.tables
   where type='U'
select @Names
```

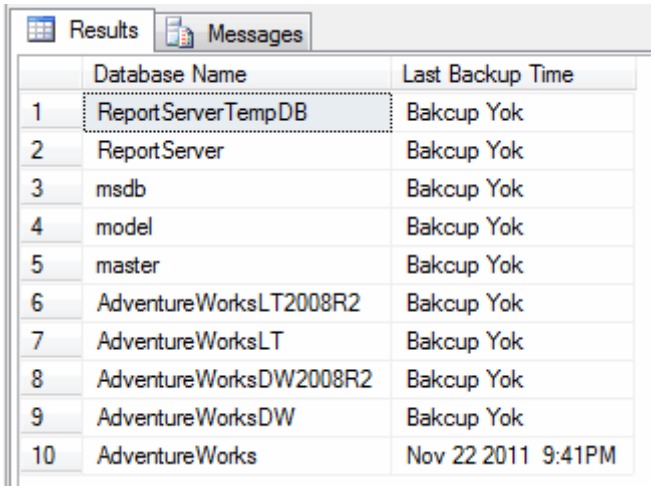
ve sonuç



Query Explorer ile olan sohbetim harika ilerliyordu. Bu kez ondan bana sistem de yer alan veritabanlarının ne zaman yedeklendiğini*(ve hatta yedeklenmediğini)* söylemesini istiyordum. Aslına bakarsanız bu önemli bir sorguydu. çünkü ilk çalıştırdığımda **AdventureWorks** için hiç bir zaman **Backup** almadığımı fark etmiştim

```
SELECT
   D.name [Database Name]
   ,case when MAX(b.backup_finish_date) is NULL
   then 'Bakcup Yok'
   else Convert(varchar(100), MAX(b.backup_finish_date))
   end AS [Last Backup Time]
FROM sys.databases D
LEFT JOIN msdb.dbo.backupset B ON D.name = B.database_name AND B.type = 'D'
WHERE D.database_id NOT IN (2)
GROUP BY D.name
ORDER BY [Database Name] DESC
```

Tabi burada anahtar nokta **backupset** içeriğinden yararlanılmasıydı. İlk sonuçlarda hiç backup almamış olduğumu görünce, hemen bir tane ürettirdim ve yeni sonuçlara baktığımda aşağıdaki ekran görüntüsünde yer alan çıktıyı elde ettim.



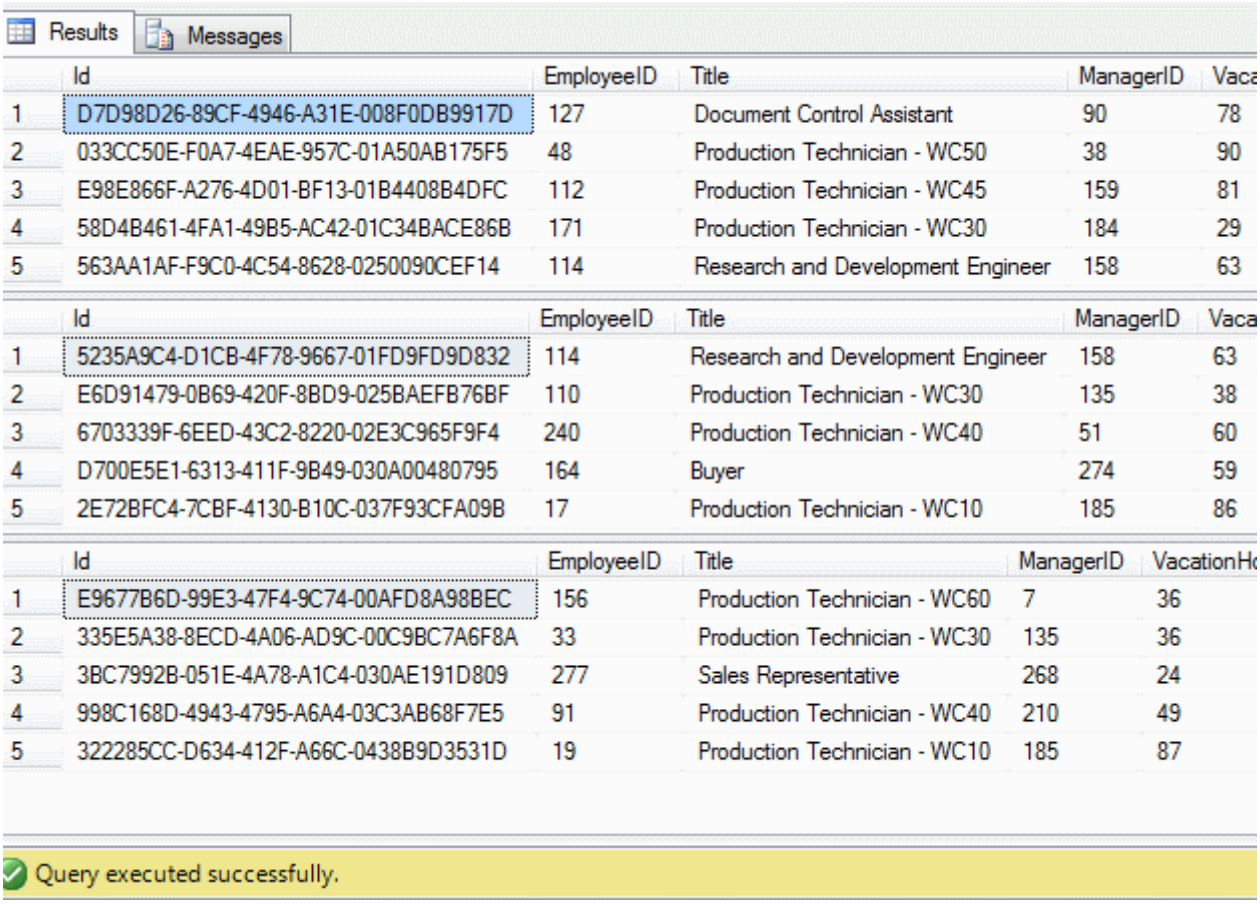
Results | Messages

	Database Name	Last Backup Time
1	ReportServerTempDB	Bakcup Yok
2	ReportServer	Bakcup Yok
3	msdb	Bakcup Yok
4	model	Bakcup Yok
5	master	Bakcup Yok
6	AdventureWorksLT2008R2	Bakcup Yok
7	AdventureWorksLT	Bakcup Yok
8	AdventureWorksDW2008R2	Bakcup Yok
9	AdventureWorksDW	Bakcup Yok
10	AdventureWorks	Nov 22 2011 9:41PM

Derken aklıma biraz daha eğlenceli bir sorgu geldi. Söz gelimi çalışanlarımızdan*(Hani o anda koca bir fabrikanın sahibi olduğunu düşündüm de)* rastgele 5 farklı kişiyi getirip onlara hediye dağıtmak istediğimi düşündüm. *(Bu o gün indirimli olarak satılacak rastgele 10 ürün de olabilirdi)*. Eğlenceli bir sorguydu. **NewId()** fonksiyonu burada işi eğlenceli hale getiren kişiydi. örnek olarak **Employee** tablosu için şanslı 5 kişiyi bulmaya çalıştım.

```
Select
   Top 5 NewId() Id
   ,EmployeeID
   ,Title
   ,BirthDate
   ,ManagerID
   ,VacationHours
from HumanResources.Employee
order by 1
```

ve aşağıdaki ekran görüntüsünde yer alan sonuçları elde ettim. Tabi ki her çalıştırılmada farklı sonuçlar elde edilmesi garantiydi.



Results | Messages

	Id	EmployeeID	Title	ManagerID	Vaca
1	D7D98D26-89CF-4946-A31E-008F0DB9917D	127	Document Control Assistant	90	78
2	033CC50E-F0A7-4EAE-957C-01A50AB175F5	48	Production Technician - WC50	38	90
3	E98E866F-A276-4D01-BF13-01B4408B4DFC	112	Production Technician - WC45	159	81
4	58D4B461-4FA1-49B5-AC42-01C34BACE86B	171	Production Technician - WC30	184	29
5	563AA1AF-F9C0-4C54-8628-0250090CEF14	114	Research and Development Engineer	158	63

	Id	EmployeeID	Title	ManagerID	Vaca
1	5235A9C4-D1CB-4F78-9667-01FD9FD9D832	114	Research and Development Engineer	158	63
2	E6D91479-0B69-420F-8BD9-025BAEFB76BF	110	Production Technician - WC30	135	38
3	6703339F-6EED-43C2-8220-02E3C965F9F4	240	Production Technician - WC40	51	60
4	D700E5E1-6313-411F-9B49-030A00480795	164	Buyer	274	59
5	2E72BFC4-7CBF-4130-B10C-037F93CFA09B	17	Production Technician - WC10	185	86

	Id	EmployeeID	Title	ManagerID	VacationH
1	E9677B6D-99E3-47F4-9C74-00AFD8A98BEC	156	Production Technician - WC60	7	36
2	335E5A38-8ECD-4A06-AD9C-00C9BC7A6F8A	33	Production Technician - WC30	135	36
3	3BC7992B-051E-4A78-A1C4-030AE191D809	277	Sales Representative	268	24
4	998C168D-4943-4795-A6A4-03C3AB68F7E5	91	Production Technician - WC40	210	49
5	322285CC-D634-412F-A66C-0438B9D3531D	19	Production Technician - WC10	185	87

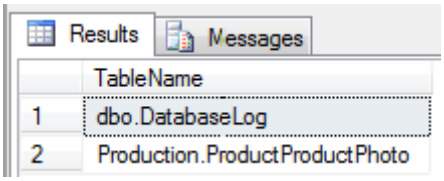
Query executed successfully.

Gerçi şimdi fark ettim ki **114** numaralı çalışan oldukça şanslıymış. çünkü ilk iki sorguda tesadüfen çıkmış Buna tabi bir tedbir almak gerektiği kanısındayım. Aslında bu tedbiri size bırakıyorum. En azından hediye çıkmış işçileri bir flag ile işaretlemeye veya farklı bir tabloda belirli süreliğine saklayarak tekrardan sorgu sonuçlarında çıkmalarını engellemeyi düşünebilirsiniz.

SQL sorgularını denediğim sırada arka planda çalışmakta olan diğer SQL penceresine gözüm ilişmişti. Aslında arada sırada oraya bakmak zorundaydım. Nitekim **Test** ortamında yer alan bir veritabanı üzerinde bazı işlemler yapılması gerekiyordu. Ne varki ilgili sistemde yer alan tablo **32 milyon satırlık** veri içerdiğinden ve test makinesi nuhnebiden kalma bir **Pentium III**olduğundan miniminnacık sıkıntılar vardı. O anda aklıma acaba **index** kullanılmayan*(örneğin Clustered Index)* tablolar var mıdır acaba sorusu geldi? Hemen local sistemimde bunu araştırmak için aşağıdaki sorgu ifadesini hazırladım.

```
select
   S.name+'.'+T.name AS [TableName]
from sys.tables T
inner join sys.schemas S
on S.schema_id = T.schema_id
where OBJECTPROPERTY(OBJECT_ID,'TableHasClustIndex') =0 and T.Type='U'
order by[TableName] ASC
```

Bir de ne göreyim



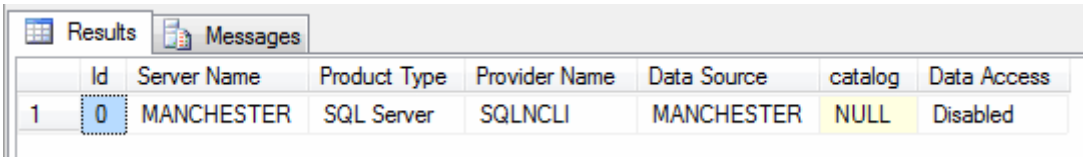
	TableName
1	dbo.DatabaseLog
2	Production.ProductProductPhoto

AdventureWorks veritabanındaki **ProductProductPhoto** tablosunda **ClusteredIndex** yok …Bak bak baaakk Tabi bu işin şaka tarafı ama performans araştırmaları yaparken belki de işe yarayacak bir sorgu olarak düşünülebilir.

İşler gayet eğlenceli gidiyordu ama enerjim de bitmek üzereydi. Son olarak basit bir sorgu yardımıyla **Query** penceresi ile olan muhabbetime son vereyim istemiştim. Bu sefer merak ettiğim, çevrede var olan SQL sunucularının hangileri olduğuydu. Aşağıdaki sorgu bunu karşılıyordu.

```
Select
   server_id Id
   ,name [Server Name]
   ,product [Product Type]
   ,provider [Provider Name]
   ,data_source [Data Source]
   ,catalog
   ,case is_data_access_enabled
   when 1 then 'Enabled'
   else 'Disabled'
   end as [Data Access]
from sys.servers
```

İşte sonuçlar,



	Id	Server Name	Product Type	Provider Name	Data Source	catalog	Data Access
1	0	MANCHESTER	SQL Server	SQLNCLI	MANCHESTER	NULL	Disabled

Elbette ben yerel makinemden sadece tek bir veri sunucusuna bağlandığımdan cılız bir sonuç çıkmıştı. Ancak aynı sorguyu arka planda çalışmakta olduğum test makinesinde yürüttüğümde piuvvvvv!!!

Buraya kadar yazılmış olan sorguları eğlence amaçlı olarak veya ciddi manada göz önüne alarak çalışmakta olduğunuz gerçek hayat SQL sunucuları üzerinde de deneyebilirsiniz. çok ilginç sonuçlar elde edeceğinizi ama oldukça faydalı bilgiler alabileceğinizi belirtmek isterim.

Peki ya bundan sonrasında ne olacak? Aslında bakarsanız burada yazılmış olan pek çok **SQL** ifadesi birer **View** haline dönüştürülüp sunucu üzerindeki farklı bir veritabanında saklanabilirler. Hatta bu veritabanının karşılığı olan bir **Entity Framework** kütüphanesi üretilip ilgili raporların örneğin bir **WCF Data Service** yardımıyla dış ortama sunulması da sağlanabilir. Tabi hassas veriler söz konusu olduğundan bu pek de iyi bir fikir değildir. Ama dilerseniz basit bir **WCF(Windows Communication Foundation) Servisini** güvenli hale getirerek ilgili içerikleri dış dünyaya servis bazlı olarak sunabilirsiniz. Sanırım bir sonraki makalemde hangi konuyu/senaryoyu ele alacağımı anlamışsınızdır Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

EglencelikSQL.sql (5,39 kb)

Entity Framework Gerçek Hayat Örnekleri Bölüm 1 (2011-12-14T16:00:00)

entity framework,wcf,surrogate types,c#,

Merhaba Arkadaşlar,

12 aralık 2011 Pazartesi günü Nedirtv?com ve Zenith Bilişim sponsorluğunda gerçekleştirdiğimiz **Entity Framework Gerçek Hayat örnekleri Bölüm 1** isimli webinerimizi aşağıdaki adresten izleyebilir veya isterseniz bilgisayarınıza indirebilirsiniz.

İlk bölümümüzde **Entity Framework** ve **Surrogate Library** projelerimizi oluşturup örnek bir iş fonksiyonelliğini diğer bir kütüphane içerisinde ele aldık ve buna ait basit bir **Unit Test** metodu geliştirdik.

İkinci webinerimizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Entity Framework ile Gerçek Hayat örnekleri Bölüm 1
{Süre : 56 Dakika 54 Saniye Boyut : 65.7 Mb}

NedirTv?Com Söyleşileri - Yazılım Eğitimleri Üzerine (2011-12-06T22:20:00)

nedirtv?com,söyleşiler,yazılım,yazılım eğitimleri,

Merhaba Arkadaşlar,

Bu aralar **Nedirtv?com** adına sizlere elimizden geldiğince çok paylaşım yapma uğraşısı içerisindeyiz. Her zaman teknik konulardan bahsetmeyeceğimizi daha önceden de ifade etmiştik. İşte bu amaçla gerçekleştirdiğimiz ikinci söyleşimiz ile karşınızdayız.

Bu kez aslında önemli bir mevzuya değiniyoruz. İster öğrenci olun ister çalışan, ister genç ister yaşlı, hayatınızın belirli bir noktasında yazılım eğitimi almaya karar verdiğinizi düşünelim. Fakat araştırıyorsunuz, kıyaslıyorsunuz, çabalıyorsunuz bir türlü nereden nasıl bir eğitim almanız gerektiğine karar veremiyorsunuz. **Acaba doğru kurum nasıl seçilir? Nelere dikkat edilmelidir? Sınıf ortamı ne kadar önemlidir? Peki ya eğitmen?**

İşte bu söyleşimizide eğitim tecrübesi olan değerli meslektaşlarım **Uğur Umutluoğlu, Ercan Bozkurt** ve **Cenk özdemir** ile bu sorulara cevap bulmaya ve sizlere yol göstermeye çalışıyoruz. Umarım yararlı bir çalışma üretmişizdir. Söyleşimizi Nedirtv?com üzerinden izleyebilir ve indirebilirsiniz. Görüşmek dileğiyle.

Entity Framework ile Gerçek Hayat Örnekleri Webinerlerim (2011-12-06T17:51:00)

entity framework,nedir?tv,surrogate types,business logic layer,wcf,

Merhaba Arkadaşlar,

Uzun zamandır **NedirTv?com** bünyesinde webiner gerçekleştirmiyordum Bununla birlikte geçtiğimiz ay **Zenith** adına verdiğimiz **Entity Framework 4.0** eğitiminde **WCF(Windows Communication Foundation)** tabanlı güzel bir gerçek hayat örneği geliştirdik. Aslına bakarsanız geliştirdiğimiz bu örneği sizler ile de paylaşmak istedik. Bu sebepten önümüzdeki **12** ve **13 Aralık**tarihlerinde iki bölüm halinde yayınlanacak olan webinerlerimizde söz konusu uygulamayı canlı olarak geliştirmek

istiyoruz. Değerli katılımlarınızı bekliyoruz İki bölümden oluşacak olan webinerlerimizde, **Entity Framework, Windows Communication Foundation, Business Logic Layer, Surrogate Types** gibi kavramlara değiniyor olacağız.

Konu : Entity Framework ile Gerçek Hayat örnekleri-1
Konuşmacı : Burak Selim ŞENYURT
Zaman : 12.12.2011 21:00:00
Katılmak için tıklayın

Konu : Entity Framework ile Gerçek Hayat örnekleri-2
Konuşmacı : Burak Selim ŞENYURT

Zaman : 13.12.2011 21:00:00
Katılmak için tıklayın

Tek Fotoluk İpucu-43(Active Directory Connection String Bilgisini Almak) (2011-12-05T23:10:00)

c#,active directory,ldap,.net framework,system.directoryservices,directory services,

Merhaba Arkadaşlar,

Oldu da domain üzerinde çalışırken Active Directory' nin bağlantı bilgisine ihtiyaç duydunuz? Bu özellikle AD ile .Net tarafında çalışırken size gerekli olan önemli bir bilgidir. Nasıl mı elde edebiliriz? Aslında basit bir teknik var. Garantisi yok ama en azından ben şirket makinemde başarılı bir şekilde test edebildiğimi söyleyebilirim

Program.cs

ADConStr.Helper

```
using System;
using System.DirectoryServices;

namespace ADConStr
{
   class Program
   {
      static void Main(string[] args)
      {
         string adConStr=Helper.GetActiveDirectoryConnectionString();
         Console.WriteLine(adConStr);
         Console.ReadLine();
      }
   }
   static class Helper
   {
      public static string GetActiveDirectoryConnectionString()
      {
         string result = String.Empty;

         using (DirectoryEntry de = new DirectoryEntry("LDAP://RootDSE"))
         {
            string defaultNamingContext = de.Properties["defaultNamingContext"][0].ToString();
            string dnsHostName = de.Properties["dnsHostName"][0].ToString();
            result= String.Format("LDAP://{0}/{1}",dnsHostName,defaultNamingContext);
         }

         return result;
      }
   }
}
```

ADConStr.rar (23,80 kb)

Parallel Programming-Reduction (2011-12-02T22:51:00)

tpl,parallel programming,reduction,mapreduce pattern,map reduce,parallel.for,parallel.foreach,

Merhaba Arkadaşlar,

Neredeyse son bir kaç saattir yoğun bir şekilde **Wolfenstein** isimli bilgisayar oyununu oynamaktaydım. Aslında çok fazla bilgisayar oyunu oynayan birisi değilimdir. Hatta bu oyunun ilk versiyonunu çok çok uzun zaman önce oynadığımı ve arada çok az oyunla haşır neşir olduğumu itiraf edebilirim

Lakin bazen oyun perisi gelip beni bir dürtmekte ve saatlerce bilgisayar başından kalkmadan oyun oynamamı istemekte. Bu gece kendisini kıramadım işte

Aslında gece boyunca Wolfenstein her ne kadar beni aşırı derece de sürüklemiş olsa da aklımın bir köşesinde beni kemiren paralel programlama konulu düşüncelerimin önüne de geçemedim. Doğruyu söylemek gerekirse şu anda çok geç bir saat de olsa konuyu açıklığa kavuşturmanın ve bununla ilişkili bir blog girdisi üretmenin tam zamanıdır diye düşündüm ve işte karşınızdayım. Oyun perisinin ensesinde biten ilham perisinin isteğiydi bu sanırım. Lafı fazla uzatmadan konuya giriş yapalım dilerseniz.(*Bunu belirtmek istedim çünkü bu kısa girişleri sevmeyen bir sürü geliştirici de tanıyorum*)

Olay paralel programlamada veriyi paralelize etmek ile alakalı. Aslında çok basit ama gözden kaçtığı takdirde önemli hatalara neden olabilecek bir durum söz konusu. Bunu izah etmenin en iyi yolu belkide basit bir örnek üzerinden ilerlemekle olacaktır. Şimdi şöyle düşünelim; elimizde yüsek boyutlu sayısal bir dizi veya koleksiyon olsun ve biz bu sayı kümesi üzerinde örneğin 7 ile tam bölünebilenlerin sayısını bulmak istediğimizi var sayalım. Standart bir **for** döngüsü ile bu işlemi yapabileceğimiz gibi, çok yüksek boyutlu bir sayı olması halinde **Parallel.For** veya**Parallel.ForEach** metodlarını da söz konusu hesaplama için kullanabiliriz pekala Elimizde çok çekirdekli veya çok işlemcili bir sistem var ise, paralel döngüleri kullanmak pek çok açıdan avantajlı olabilir nitekim. Şimdi az önce bahsetmiş olduğumuz senaryoyu aşağıdaki **Console** uygulamasına ait kod satırlarında simüle ettiğimizi var sayalım.

```
using System;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace ReductionSample
{
   class Program
   {
      static void Main(string[] args)
      {
         List<int> numbers = Helper.GetRandomNumberList(9000000);

         #region Klasik Yol

         int count=0;
         for (int i = 0; i < numbers.Count; i++)
         {
            if (numbers[i] %7==0)
               count++;

         }
```

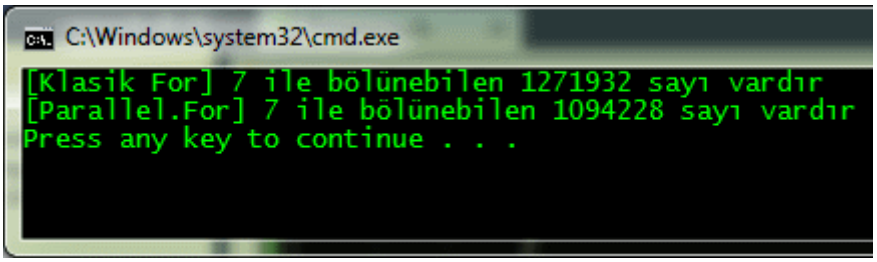
```
            Console.WriteLine("[Klasik For] 7 ile bölünebilen {0} sayı
vardır",count.ToString());

            #endregion

            #region Parallel For

            int parallelCount=0;

            Parallel.For(0, numbers.Count, (i) =>
            {
               if (numbers[i] % 7 == 0)
                  parallelCount++;
            }
            );

            Console.WriteLine("[Parallel.For] 7 ile bölünebilen {0} sayı
vardır",parallelCount.ToString());

            #endregion

        }
    }

    static class Helper
    {
        public static List<int> GetRandomNumberList(int NumberCount)
        {
            List<int> result = new List<int>();

            Random rnd = new Random();
            for (int i = 0; i < NumberCount; i++)
            {
                result.Add(rnd.Next(1, 100));
            }

            return result;
        }

    }
}
```

static olarak tanımlanmış olan **Helper** sınıfı belirli sayıda rastgele tam sayı üretmek üzere yazılmış **GetRandomNumberList** isimli bir metod içermektedir. Uygulamaya ait **Main**metodu içerisinde ise önce standart bir **for** döngüsü ile ardından da

bunun **paralel** versiyonu ile birer iterasyon gerçekleştirilmekte ve **7 ile bölünebilen sayıların toplam sayısı** hesap edilerek ekrana yazdırılmaktadır.

Aslında sayı aralığı ne kadar yüksek olursa standart **for** döngüsünün hesaplama için daha fazla zaman harcayacağı ve **Parallel.For** döngüsünün aynı işi daha kısa sürede bitireceği aşikardır. Ne varki burada öngöremediğimiz ve belki de tahmin etmediğimiz bir durum daha söz konusudur. Bunu anlamak için kodun çalışma zamanındaki bir kaç çıktısına bakabiliriz

Birinci çalıştırmanın sonuçları;

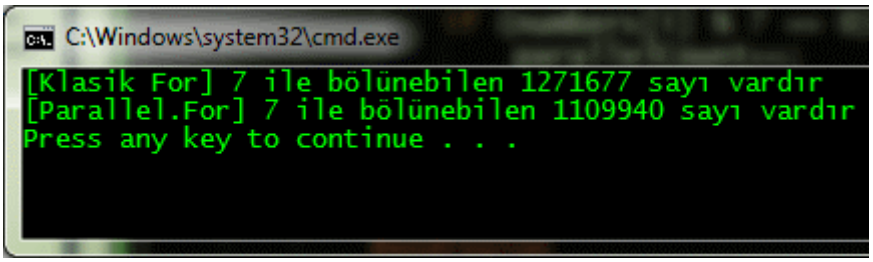


O oooo!!!

İkinci çalıştırmanın sonuçları;

O ooooo too!!!

üçüncü çalıştırmanın sonuçları;



Şaka mı bu ?

Dikkat edilecek olursa **Parallel.For** döngüsü 7 ile bölünebilen sayıların miktarını her defasında standart for döngüsüne göre farklı hesaplamış ve aslında hiç birisinde de doğru sonucu tutturamamıştır. Bu aslına bakılacak olursa son derece doğal bir sonuçtur.

Nitekim **Parallel.For** döngüsü çalışmaya başladıktan sonra birden fazla **Task** oluşturmakta ve bunları bir veya daha fazla **Thread'** in yönetimine sunmaktadır. Gözden kaçan nokta, bu **Task'** ların her birinin aslında aynı değişken üzerinde hesaplama yapmaya çalışıyor olmalarıdır. Yani, örneğimizde açılan **Task** bloklarının her biri aslında aynı **parallelCount** değişkeni üzerinde bir artım gerçekleştirmeye çalışmaktadır. Bu da çok doğal olarak doğru sonucun hesaplanamamasına neden olmaktadır. Peki ya öyleyse çözüm nedir?

Standart bir **for** döngüsünün kullanılması tercih edilebilir Lakin bu durumda **Parallel** olmanın avantajları kaybedilecektir. Buradaki gibi bir sayı aralığında bu çok önemli gözükmese de, bilimsel veya finansal hesaplama yapan bir uygulamada bu ayrım, performans açısında çok kritik bir fark doğurabilir. Bu nedenle **Parallel** döngüler ile devam edecek isek **Reduction** olarak tanımlanan ve aslında ilerleyen zamanlarda işleyeceğimiz **MapReduce** deseninin temelini oluşturan bir konuyu göz önüne alarak ilerlememiz gerekmektedir. Aslında teorik olarak çok fazla sıkmak istemiyorum sizi, ancak özet olarak işleyişi ifade etmek isterim. **Parallel.For** döngüsünü öyle bir çalıştırmalıyız ki, açılacak olan **Thread'** ler ve bunların içerisinde yer alacak olan**Task'** lar, **parallelCount** sayısının aslında aralarında paylaştıkları bir veri olduğunu bilmelidirdler üstelik her bir Thread kendi içerisinde bir toplam sayı hesabı yapmalı ve işleyişini bitirdiğinde de herkes için ortak olan bir değişkene bunu eklemelidir. Bu ekleme işinin ise senkronize edilerek yapılması şarttır.

Neyseki **Parallel.For** ve **Parallel.ForEach** döngüleri bu ortak paylaşımlı veri değişkenlerine izin veren **aşırı yüklenmiş(Overload)** versiyonlara sahiptirler. çözüm olarak kodumuzu aşağıdaki gibi değiştirmemiz yeterli olacaktır.

```
#region Reduction

int reductionCount = 0;

Parallel.For(0, numbers.Count,
   () => 0,
   (i, state, currentTotal) =>
   {
      if (numbers[i] % 7 == 0)
         currentTotal++;

      return currentTotal;
   },
   (currentTotal) =>
   {
      Console.WriteLine("{0} Current Total
{1}",Thread.CurrentThread.ManagedThreadId,currentTotal.ToString());
      Interlocked.Add(ref reductionCount, currentTotal);
```

```
    }
    );

Console.WriteLine("[Reduction] 7 ile bölünebilen {0} sayı
vardır",reductionCount.ToString());

#endregion
```

Aslında **Parallel.For** için biraz karmaşık bir yazım söz konusu. İlk iki parametre tanıdık. üçüncü parametrede herhangibir işlem yapmadan geçiyoruz. önemli olan ise metodun aldığı son iki parametre. Bunlardan
ilki **Func<int,ParallelLoopState,TLocal,TLocal>** tipinden. Diğeri
ise **Action<TLocal>** türündendir. Teorik olarak yapılmak istenen her bir **Thread**' in kendi içerisinde değerlendireceği **özel bir yerel değişken oluşturmak(Private Thread-Local Variable)** ve tüm paralel çalışma tamamlandığında bu özel yerel değişkenlerin bir toplamını hesaplayarak sonuca ulaşmaktır. Bir başka deyişle açılan **Thread**' ler kendi özel toplam değişkenlerini arttıracaklardır. Bu işlemi,

```
(i, state, currentTotal) =>
{
    if (numbers[i] % 7 == 0)
        currentTotal++;

    return currentTotal;
}
```

kısmı gerçekleştirmektedir.

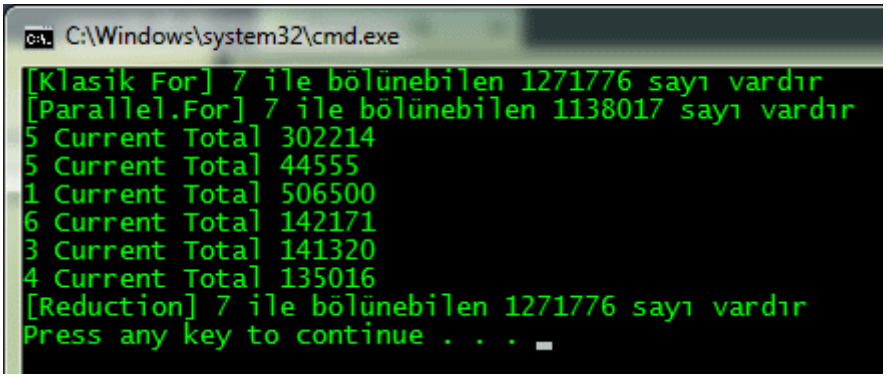
Paralel çalışmaya dahil olan **Thread'** ler işlemlerini bitirdikten sonra da,

```
(currentTotal) =>
{
      Console.WriteLine("{0} Current Total
{1}",Thread.CurrentThread.ManagedThreadId,currentTotal.ToString());
      Interlocked.Add(ref reductionCount, currentTotal);
}
```

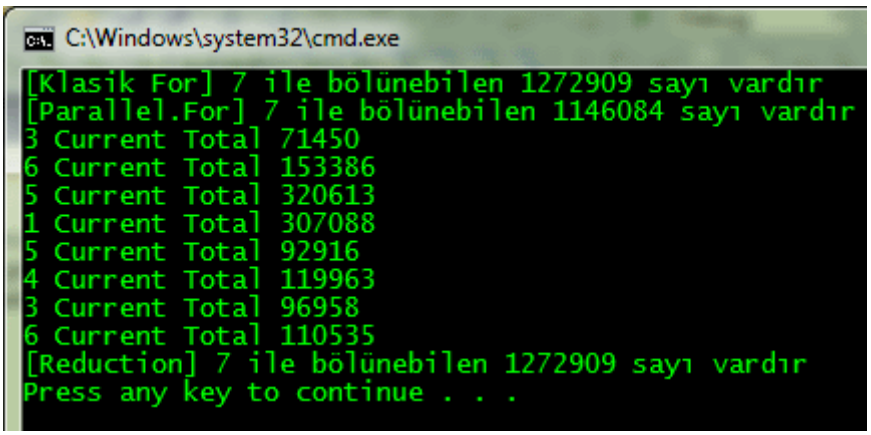
kodu devreye girmekte ve **Thread'** ler için
hesaplanan **currentTotal** değerlerini **ref** ile **reductionCount** değerine eklemektedir. Böylece tüm **Thread'** lerin kendi yerel alanlarındaki veriler üzerinde yaptığı 7 ile bölünebilen sayıların miktarı, birleştirilmektedir. **Interlocked** burada devreye giren önemli bir fonksiyonellik sunmakta ve senkronize bir şekilde currentTotal değerlerinin reductionTotal değişkenine eklenmesine olanak sağlamaktadır. İşte bir koleksiyon içeriğinin bu şekilde tekil bir değere indirgenmesine **Reduction** adı verilmektedir.

örneğimizi az önceki testte olduğu gibi yine arka arkaya 3 defa çalıştırırsak aşağıdaki ekran görüntüsünde yer alanlara benzer sonuçları elde ettiğimizi görebiliriz.

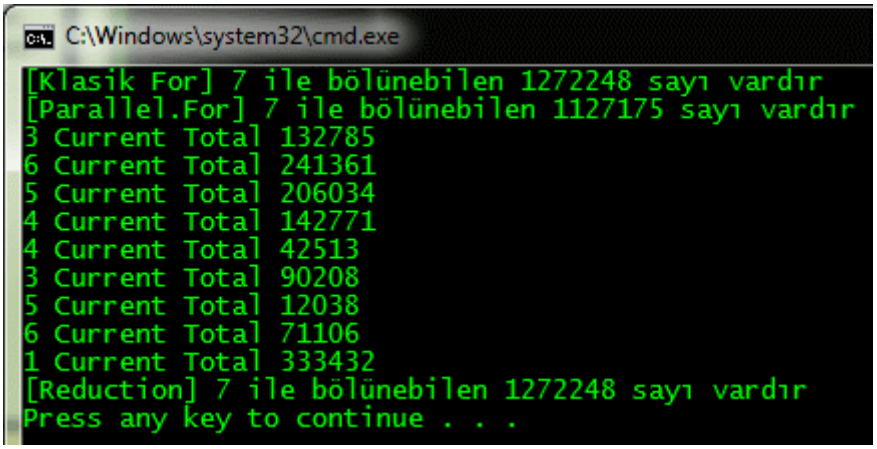
İlk çalışma sonucu;



ikinci çalışma sonucu;



üçüncü çalışma sonucu



Görüldüğü gibi **Reduction** tekniğine göre yapılan hesaplama sonuçları ile klasik **for** döngüsü ile yapılan hesaplama sonuçları bire bir örtüşmektedir. Elbetteki her

çalışma sonrasında **Parallel.For** döngüsünün başlatacağı **Thread'** ler farklı olacaktır. Bu yüzden son **Action** temsilcisinin icrası sonucu üretilen **currentTotal** değerleri farklılıklar gösterecektir. Bu yüzden Current Total değerleri hep farklı sonuçlar vermiştir.

özetle **Reduction** tekniğini kullandığımızda, paralel olarak işletilen **Thread'** lerin kendi yerel değişkenleri ile çalışmaları sağlanabilmekte ve bunlar sonuç olarak tek bir değişkene indirgenerek bu tip senaryolarda göz önüne alınabilmektedir. Yazımızın başında belirttiğimiz üzere **Reduction** tekniği aslında **MapReduce** deseninin temelini oluşturan önemli bir kavramdır. Bu deseni ilerleyen yazılarımızda sizlere örnek bir senaryo üzerinden aktarmaya çalışıyor olacağım. Eğer bu ve bunun gibi diğer paralel programlama desenlerini merak ediyorsanız **Microsoft'** un ücretsiz olarak indirebileceğiniz Patterns for Parallel Programming : Understanding and Applying Parallel Patterns with .Net Framework 4.0 dökümanını okuyabilirsiniz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

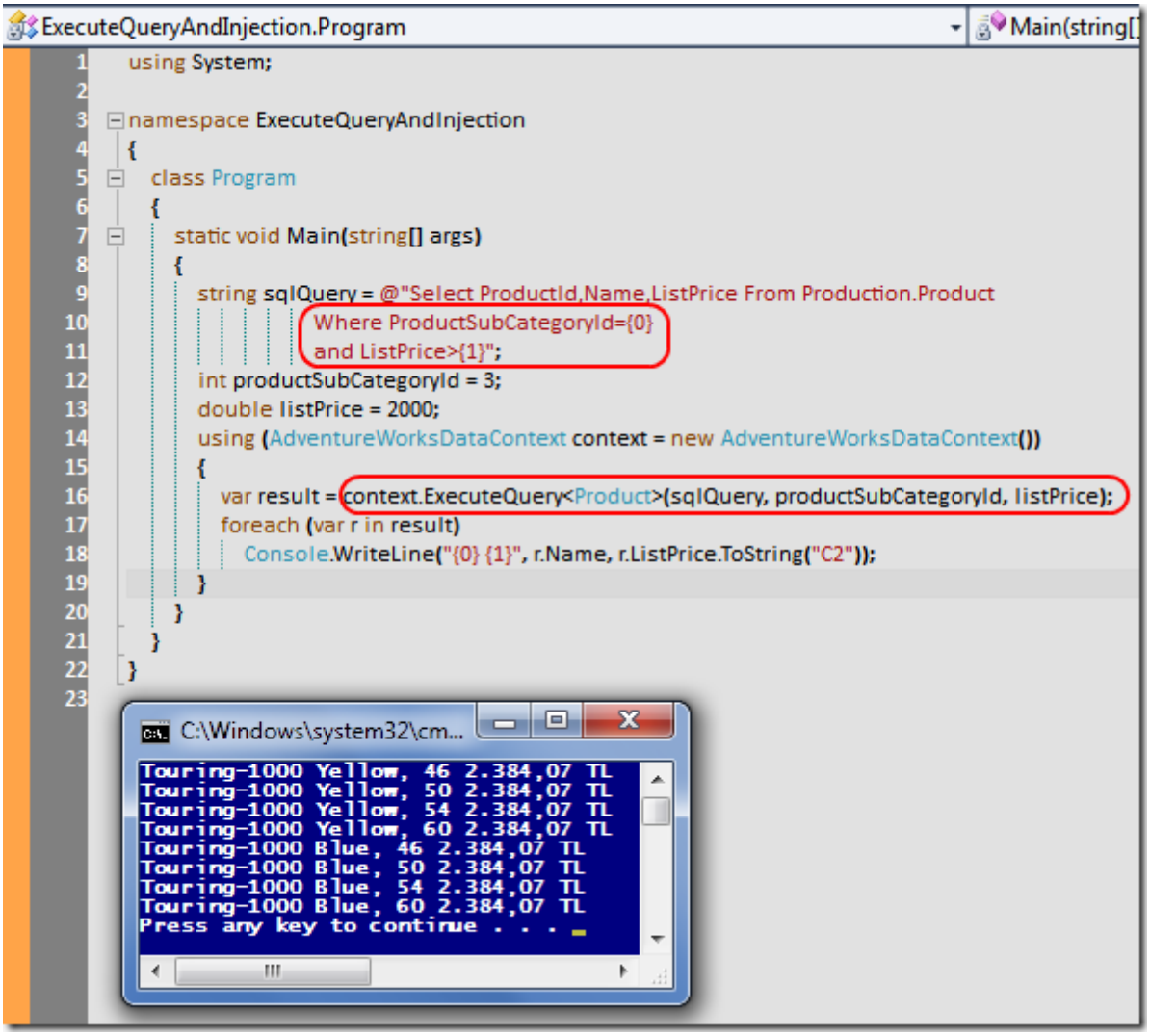
ReductionSample.rar (24,88 kb)

Tek Fotoluk İpucu-42(ExecuteQuery ile Injection' dan Korunmak) (2011-11-26T14:15:00)

c#,linq to sql,executequery,sql injection,sql,

Merhaba Arkadaşlar,

LINQ to SQL kullandığımız durumlarda bildiğiniz gibi dışarıdan SQL sorgularını da icra ettirebilmekteyiz. Bu amaçla DataContext tipinin ExecuteQuery metodu kullanılmakta. Ancak özellikle SQL Injection saldırılarına karşı dikkatli olmamız gerekiyor. Bu nedenle söz konusu metodun placeholder kullanımına izin veren versiyonunu ele almamızda yarar olduğu kanısındayım. Nasıl mı? 



ExecuteQueryAndInjection.rar (52,04 kb)

NedirTv?Com Söyleşileri - Yazılımcıların Sosyal Hayatı (2011-11-24T09:20:00)

nedirtv?com,nedirtv?com söyleşileri,yazılım,sosyalleşme,

Merhaba Arkadaşlar,

Uzun süredir **NedirTv?Com** bünyesinde bir çalışmada bulunamamıştım. Açıkçası teknik paylaşımlarımıza uzun bir süre ara vermiştik Geçtiğimiz haftalarda bu amaçla **NedirTv?Com** kurucusu **Uğur Umutluoğlu** ve **Ercan Bozkurt** hocalarımız ile

toplanarak yeni bir seriye başlamaya karar verdik. Serimiz içerisinde oldukça güzel, eğlenceli ve önemli konular yer almakta. Sadece teknik değil ama yaşam, eğlence, sosyallik, spor ve benzeri pek çok konu hakkında da söyleşilerimiz olacak. Bu serinin ilk bölümünde, sektörde yaygınlaşmış bir algıyı(kanıyı) tartışmaya açıyoruz. Acaba Yazılımcıların A-Sosyal oldukları doğrumudur? Günlerini sadece monitore bakarak mı geçirirler? Hayata küskünler midir? Bir başka deyişle, yazılımcıların sahip oldukları meslek ve yoğun uğraşıları gereği Sosyallik adına hangi yerde olduklarını konumlandırmaya çalıştık. Umarım sizleri Yazılımcının Sosyalliği adına biraz da olsa bilgilendirebiliriz.

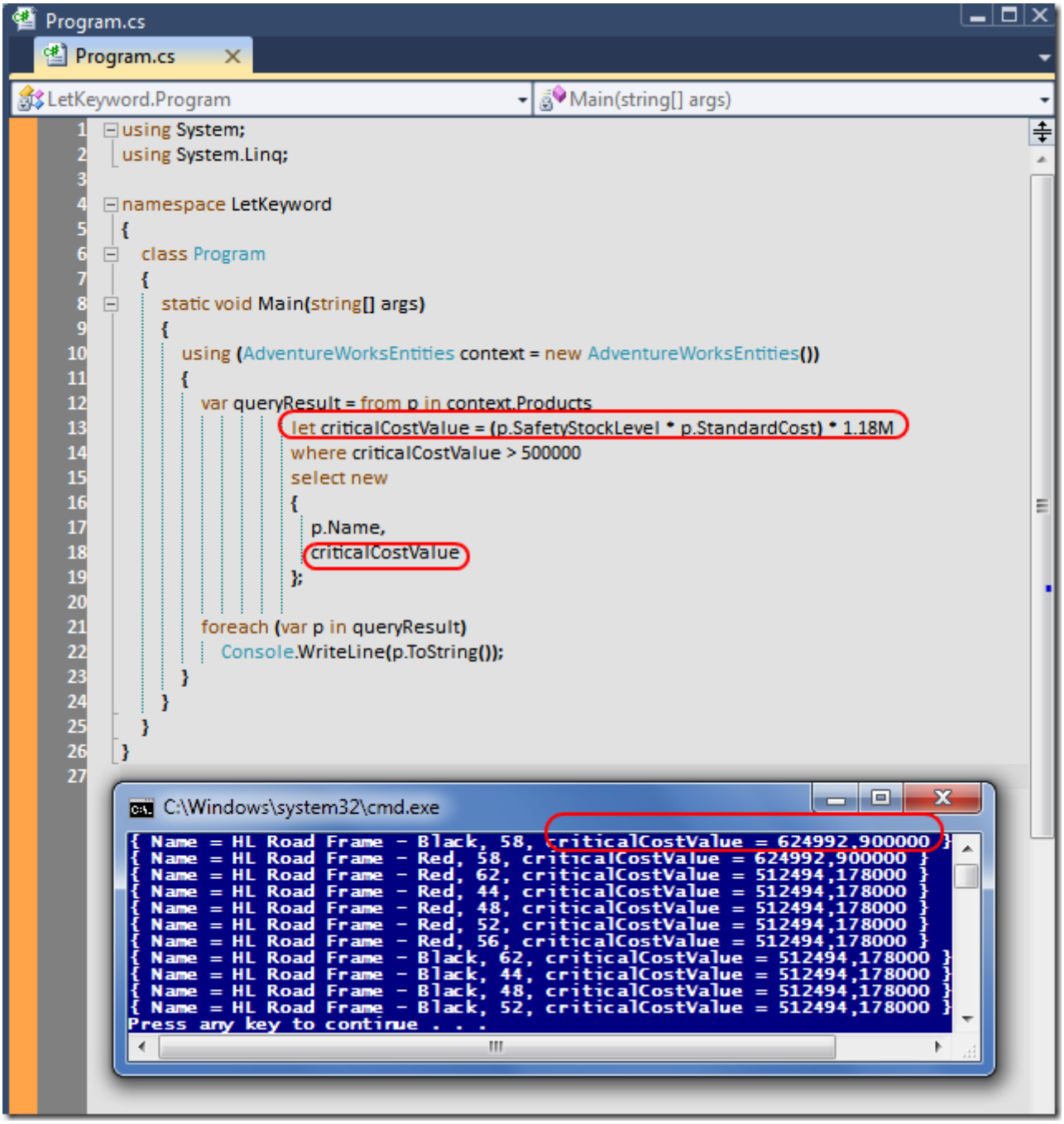
Şu anda **NedirTv?Com** bünyesinde yayınlanan söyleşimize bu adres üzerinde ulaşabilirsiniz. Keyifli seyirler dilerim

Tek Fotoluk İpucu-41(Let Keyword) (2011-11-21T22:35:00)

c#,c# temelleri,linq,let,.net framework,

Merhaba Arkadaşlar,

LINQ sorgularını pek çoğumuz etkin bir şekilde kullanıyoruzdur. Ama belki aralarda atladığımız keyword' ler de vardır. Mesela Let. çık sık kullanmasakta oldukça işimize yarayan bir anahtar kelimedir. Söz gelimi onu bir ifadeye eşitleyip LINQ sorgusunun hatırlamasını sağlayabilir, koşul olarak değerlendirebilir hatta anonymoust tip içerisine bile dahil edebiliriz. Tipik olarak sorgu içinde bir değişken mantığında ele almış oluruz. Nasıl mı?



LetKeyword.rar (194,69 kb)

[Girişimci Ruh ve BizSpark (2011-11-21T05:14:00)]

bizspark,microsoft,visual studio 2010 ultimate,sql server,biztalk,windows server,dynamics,azure,expression,team system,

Merhaba Arkadaşlar,

Üniversiteye 1993 yılında girdim. O zamanlarda Matematik Mühendisliği bölümünde **GwBasic, C++, Cobol** gibi programlama dilleri okutuluyordu. Yazılıma ilgili ve meraklı bir öğrenciydim. Gerçi o zamanlar bu meslek şimdiki gibi dallara ayrılmıyordu*(Yani Yazılım Mühendisi, Database Developer, Database Admin, Scrum Master, Team Leader, Software Architect, Senior Developer, Web Designer vsâ€¦)*Genel olarak ortada dolaşan Bilgisayar Programcısı gibi bir kavram söz konusu idi. Zaman ilerledikçe **Delphi** ile*(ve üstad Anders Heijslberg ile)* tanıştım ve sonrasında **Microsoft** ürünlerine geçiş yaptım ve bu yoldan**1999** yılından beri hiç sapmadım

Aslında Üniversitenin ilk yıllarından itibaren hayalim bir şirket kurabilmek ve yazılım ürünleri geliştirebilmekti. **Freelance** bir kaç iş yapmıştım ama sonunda maaşlı çalışan bir eleman olup çıktım ve bu yıllardır bu şekilde devam etmekte Bir başka deyişle hayallerimi tam olarak gerçekleştiremedim.

Doğruyu söylemek gerekirse bir yazılım şirketi kurmak hiç ama hiç kolay değil. Her şeyden önce elinizden güçlü bir veya daha çok ürün bulunması şart. Üstelik bu ürünleri satabilmeli ve daha da önemlisi pazarlayabilmelisiniz. Bu anladam piyasadaki ihtiyaçarı fark edebilmek son kullanıcıya hitap edecek ruhunu okşayacak hamleleri yapabilmek çok önemli. Sermaye, şirket binası, demirbaşlar, bilgisayarlar gibi konulara ve gerekliliklere hiç değinmiyorum zaten Ancak işin önemli bir kısmı da varki o da Lisanslı ürün kullanımı.

Şöyle bir düşündüğümüzde **Microsoft** ürün grubu içerisinde lisans ücretleri oldukça yüksek olanların varlığında aslında hem fikiriz. Ama olmassa olmaz ürünler olduğunu da kabul etmekteyiz. Özellikle bir yazılım ekibine sahipsek ve **Agile** süreçleri uygulatıyorsak **Visual Studio Team System** şart. Veri ambalarımız için **SQL Server** şart. Uygulama geliştirme için **Visual Studio** şart. Hatta bir testçimiz veya mimarımız var ise onlar için özel olan Visual Studio sürümlerini kullanmamız da şart. Daha sayılabilecek pek çok temel ve gerekli ürün söz konusu.

Hal böyle olunca lisans ücretlerinin yüksek olmasının karşısında özellikle yeni çıkan versiyonların kullandırılması adına **Microsoft** tarafında önemli çalışmalar da söz konusu. **BizSpark** programı bu anlamda oldukça önemli bir yere sahip.
Ben **MVP** olduğum dönemlerde*(yani emekli olmadan önce)* pek çok seminerimin girişinde **Bizspark** programının tanıtılmasına yer vermeye çalışırdım.

Aslında program bana göre son derece başarılı ve hızla gelişiyor. 3 senelik bir yazılım şirketiyseniz, elinizde ürün veya ürünleriniz var ve yıllık cironuz 250bin doların altında ise yukarıda bahsettiğim ürünler ve daha fazlasını **BizSpark** programından yararlanarak ücretsiz olarak kullanabiliyorsunuz. Ürün ailesinde neler yok ki

1. Visual Studio Ultimate with MSDN
2. Visual Studio Team Foundation Server 2010
3. ExpressionÂ® Studio Version 4
4. Microsoft Windows ServerÂ® (Enterprise'a kadar tüm versionlar)

5. Microsoft SQL ServerÂ® (tüm versionlar)
6. Microsoft Office SharePointÂ® Portal Server
7. Microsoft System Center
8. Microsoft BizTalkÂ® Server
9. Microsoft DynamicsÂ® CRM
10. Ek olarak BizSpark startups üyelerine Microsoft Azureâ„¢ Services Platformun da kullanım hakkı

Süper öyle değil mi? Program bence çok önemli. Daha başka avantajları da mevcut. Özellikle daha fazla kitleye ulaşabilmek adına BizSpark veritabanına kayıtlo olmak bunlardan birisi. Bildiğiniz gibi buradaki pek çok ürünün *(Visual Studio, SQL Server, Biz Talk)* yeni versiyonları önümüzdeki yıl içerisinde çıkıyor olacak. Dolayısıyla 3 senelik bir şirket değilseniz bile, geleceğe yatırım yapmak açısından programı ciddi anlamda takip etmeniz ve ileride katılmanız faydalı olacaktır.

Program hakkında daha detaylı bilgiye bu adresten de ulaşabilirsiniz. Özellikle referans olarak yazılmış firmaların isimlerine dikkatinizi çekmek isterim.

Task Relations-Continuation Metodları (2011-11-18T23:00:00)

tpl,task parallel library,task relations,parallel programming,

Merhaba Arkadaşlar,

Böylesine yağmurlu ve sabah trafiğinin tavan yaptığı bir günde size ne Radyo Eksen' deki güzel melodiler, ne de okuduğunuz mizah dergisindeki karikatürler iyi gelmiyorsa, başka bir şeyle uğraşmanın yeridir diyebilirim. Ben bu sıkıntıyı aşmak ve kendimi daha iyi hissetmek adına bir makale daha yazmaya karar verdim ve hemen **Windows Live Writer** programını açtım Yanında da **Paint.Net'** i. Bakalım bu gün menümüzde neler var?*(Gerçi şöyle sahil kenarına gidip yürüyüşte yapabilirdim ama bu seferlik böyle olsun)*

Microsoft, paralel programlama ile ilişkili olarak olabildiğince çok alternatifi düşünmüş ve kullanıma sunmuştur. özellikle senaryo bazında düşündüğümüzde paralel çalışma algoritmalarından senkronizasyona kadar pek çok noktada bu durumu görmekteyiz. Söz gelimi **Task'ler arası ilişkiler(Relations)** ve veri transferleri konusunda olabilecek tüm kombinasyonlar değerlendirilmeye çalışılmıştır. **Task'** ler arası ilişkiler oldukça ilginç ve enteresan bir konudur. Gerçek hayat senaryolarında doğru ve uygun karşılıklarını bulmak her ne kadar zor olsa da en azından teorik olarak paralel programlamacıların konuyu

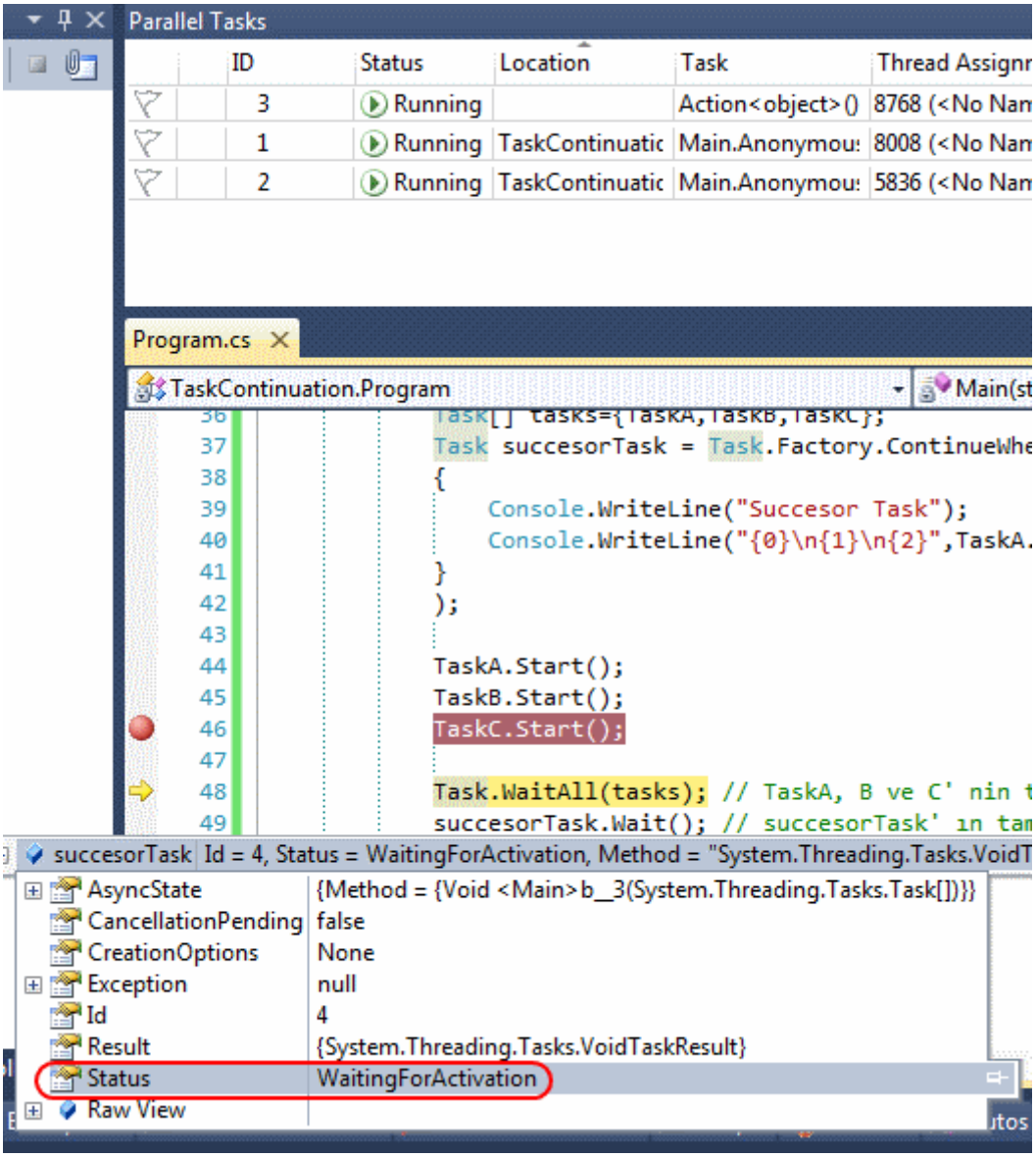
bilmesi önemlidir. İşte bu yazımızda **Task** örnekleri arası ilişkiler konusuna değinmeye başlayacak ve ilk olarak **Continuations** seçeneklerini irdelemeye çalışacağız.

Continuations tekniklerinde, bir **Task'** ın çalıştırılması veya icra edilmesi, atası olan ya da öncesinden tanımlanıp kendisine bağlanan **Task** örneklerine bağlıdır. Normal şartlar altında size tek bir **Task** örneği ve bu **Task** çalışmasını bitirdikten sonra devreye girmesi gereken bir **Task** örneğinin ele alındığı senaryoyu aktarmam gerekiyor. Ancak bana göre konunun daha iyi anlaşılabilmesi için aşağıdaki kod parçasında yer alan örneği göz önüne alarak başlamamız daha doğru olacaktır

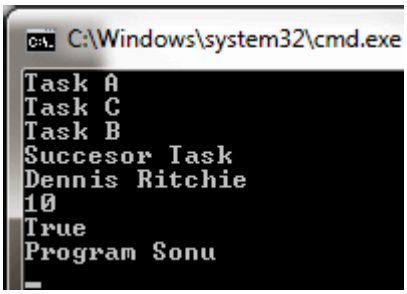
```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace TaskContinuation
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Task örnekleri

         Task<string> TaskA = new Task<string>(() =>
            {
               Thread.SpinWait(Int32.MaxValue-4000000);
               Console.WriteLine("Task A");
               return "Dennis Ritchie";
            }
         );

         Task<int> TaskB = new Task<int>(() =>
            {
               Thread.SpinWait(Int32.MaxValue - 5000000);
               Console.WriteLine("Task B");
               return 10;
            }
         );
         Task<bool> TaskC = new Task<bool>(() =>
            {
               Thread.SpinWait(Int32.MaxValue - 10000000);
               Console.WriteLine("Task C");
               return true;
            }
         );
```

```
        #endregion

        Task[] tasks={TaskA,TaskB,TaskC};
        // Succesor Task, TaskA,TaskB ve TaskC tamamlanıncaya kadar bekleyecektir
        Task succesorTask = Task.Factory.ContinueWhenAll(tasks, (antecedentTasks)
=>
        {
           Console.WriteLine("Succesor Task");
           Console.WriteLine("{0}\n{1}\n{2}",TaskA.Result,TaskB.Result,TaskC.Result);
        }
        );

        TaskA.Start();
        TaskB.Start();
        TaskC.Start();

        Task.WaitAll(tasks); // TaskA, B ve C' nin tamamlanmasını bekle
        succesorTask.Wait(); // succesorTask' ın tamamlanmasını bekle

        Console.WriteLine("Program Sonu");
        Console.ReadLine();
      }
   }
}
```

öncelikli olarak kod parçamızı kısaca inceleyelim. Senaryomuzda 4 adet **Task** örneği bulunmaktadır. **TaskA, TaskB** ve **TaskC** isimli **Task** nesne örnekleri geriye farklı tiplerde değerler döndürmektedir. Dikkat edilmesi gereken nokta **succesorTask** değişkeni ile tanımlanmış olan **Task** örneğinin oluşturulma şeklidir. Dikkat edileceği üzere **Task.Factory.ContinueWhenAll**metodu kullanılmıştır. Buna göre, **ContinueWhenAll** metodunun ilk parametresine verilen **Task** dizisine ait **Task** örnekleri tamamlanmadığı sürece, ikinci parametre ardından gelen **Anonymous Metod** içeriği çalıştırılmayacaktır  Bir başka deyişle **Succesor Task** devreye girmeyecektir. Bu durumu uygulamayı çalışma zamanında **Debug** ederken daha net bir şekilde görebiliriz.



Şekilden de görüleceği üzere **Task** örneklerinin her üçü de **Start** edilmiş ancak **succesorTask'** ın o anki **Status** durumu **WaitingForActivation** olarak kalmıştır. Bunun sebebi, önceki**Task** örneklerinin tamamının işleyişini henüz bitirmemiş olmasıdır. örnek uygulamamızın çalışma zamanındaki görüntüsü ise aşağıdaki gibi olacaktır.



Görüldüğü üzere bir **Task** örneğinin, kendisinden önceki başka **Task** örneklerinin tamamının işleyişini bitirmesinden sonra devreye girmesi bekleniyorsa, **ContinueWhenAll** metodu kullanılabilir. Aslında bakarsanız daha gerçekçi

senaryolara gitmek için **Continue**… metodlarının
aldığı **TaskContinuationOptions enum** sabitinin değerlerine bakmakta yarar vardır. çünkü bu **Enum** sabitinin değerleri, **Succesor Task** örneğinin hangi durumlarda devreye girmesi konusunda daha farklı bakış açılarının değerlendirilebilmesini sağlamaktadır. Enum sabitinin alabileceği değerler ise şunlardır.

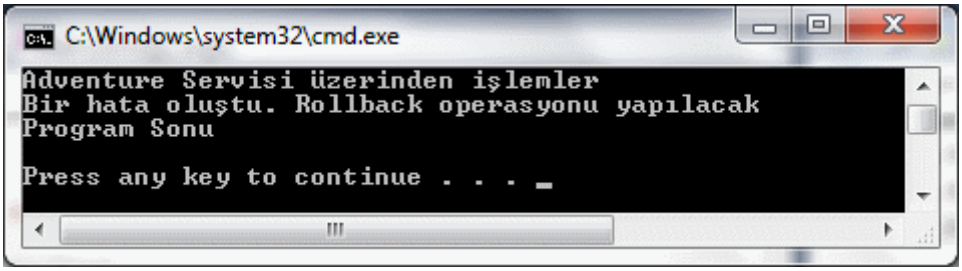
- None
- AttachedToParent
- ExecuteSynchronously
- LongRunning
- NotOnCanceled
- NotOnFaulted
- NotOnRanToCompletion
- OnlyOnCanceled
- OnlyOnFaulted
- OnlyOnRanToCompletion
- PreferFairness

Konuyu daha net kavramak adına örnek uygulamamızdaki kodlarımızı biraz değiştirelim.

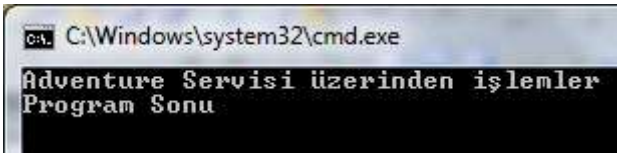
```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.IO;

namespace TaskContinuation
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Senaryo #2

         Task TaskA = new Task(() =>
            {
               Console.WriteLine("Adventure Servisi üzerinden işlemler");
               throw new FileNotFoundException();
            }
         );

         Task succesorTask = TaskA.ContinueWith(
            (antecedentTasks) =>
            {
               Console.WriteLine("Bir hata oluştu. Rollback operasyonu yapılacak");
            }
```

```
            , TaskContinuationOptions.OnlyOnFaulted
            );

        TaskA.Start();
        try
        {
            TaskA.Wait();
        }
        catch(AggregateException excp)
        {
            succesorTask.Wait();
        }

        #endregion

        Console.WriteLine("Program Sonu");
        Console.ReadLine();
    }
  }
}
```

Şimdi bu senaryoda daha farklı bir durum söz konusudur. **TaskA** içerisinde bilinçli olarak bir **Exception** üretildiği görülmektedir. Tabiki gerçek hayat senaryosunda böyle bir olasık olma ihtimali olduğu göz önüne alınmalıdır. Diğer yandan **TaskA** üzerinden **ContinueWith** metodunu kullanarak **succesorTask** örneği oluşturulmakta ve **OnlyOnFaulted enum sabiti** değeri verilmektedir. Buna göre, **succesorTask** nesne örneğinin devreye girme durumu, bir önceki **Antecedent Task** örneği içerisinde bir **Exception** oluşması ve **Faulted** durumuna düşmesi halidir. Dolayısıyla örneğimizi çalıştırdığımızda aşağıdaki ekran görüntüsüne benzer bir sonuç ile karşılaşmamız son derece doğaldır.



Diğer yandan **throw Exception** satırı yorum haline getirilir veya kaldırılırsa bu kez çalışma zamanı görüntüsü aşağıdaki gibi olacaktır.

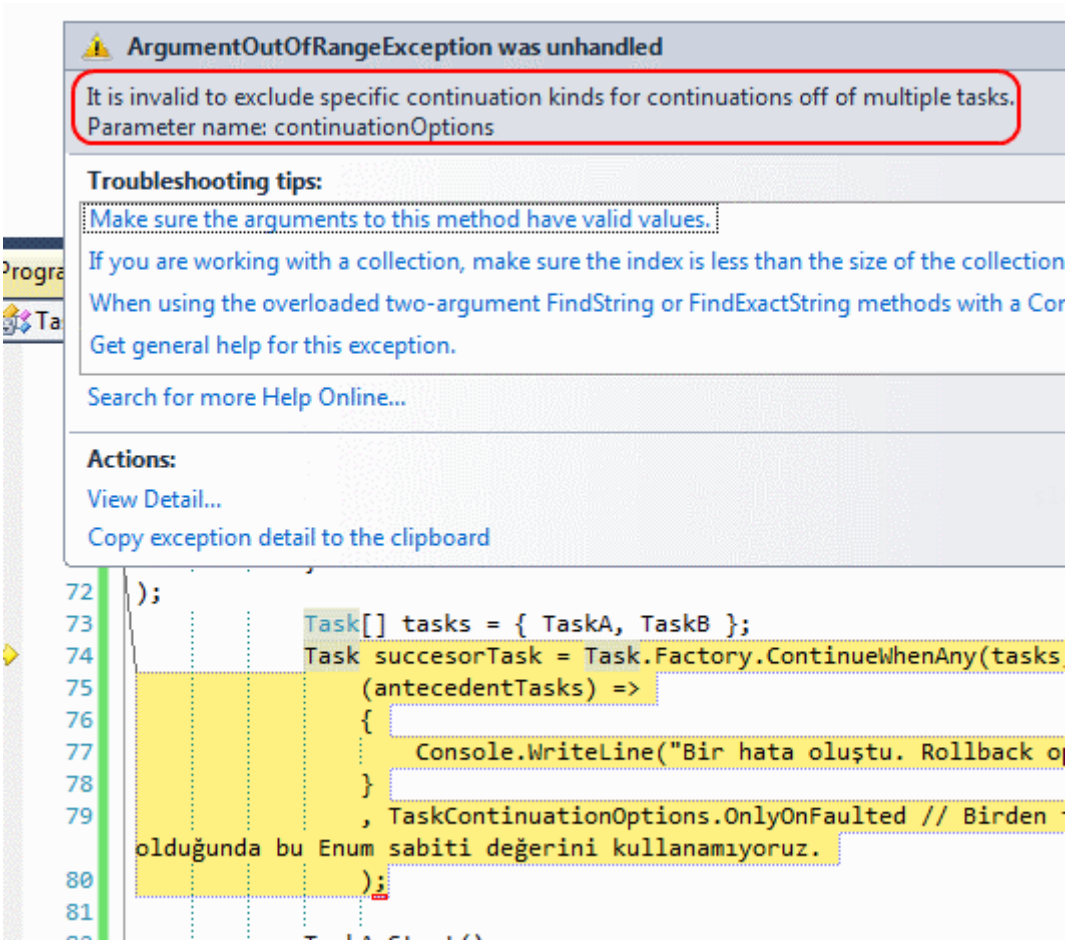


Görüldüğü üzere bir önceki **Task** örneğinde herhangibir **Exception** durumu söz konusu olmadığından, **succesorTask** örneğine ait metod icra edilmemiştir. Tabi burada akıllı bir geliştirici hemen şunu soracaktır; **Birden fazla Task örneğinden herhangibirinde bir hata meydana geldiğinde ilgili Succesor Task devreye girse olmaz mı?** Güzel soru…Bu vakayı test etmek için aşağıdaki kod parçasını deneyebiliriz.

```
using System;
using System.IO;
using System.Threading.Tasks;

namespace TaskContinuation
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Senaryo #2

         Task TaskA = new Task(() =>
            {
               Console.WriteLine("Adventure Servisi üzerinden işlemler");
               //throw new FileNotFoundException();
            }
         );

         Task TaskB = new Task(() =>
         {
            Console.WriteLine("Northwind Servisi üzerinden işlemler");
            //throw new FileNotFoundException();
         }
);
         Task[] tasks = { TaskA, TaskB };
         Task succesorTask = Task.Factory.ContinueWhenAny(tasks,
            (antecedentTasks) =>
            {
               Console.WriteLine("Bir hata oluştu. Rollback operasyonu yapılacak");
            }
            , TaskContinuationOptions.OnlyOnFaulted
            );

         TaskA.Start();
         TaskB.Start();
         try
         {
            Task.WaitAll(tasks);
```

```
        }
        catch(AggregateException excp)
        {
           succesorTask.Wait();
        }

        #endregion

        Console.WriteLine("Program Sonu");
        Console.ReadLine();
     }
  }
}
```

Ne yazık ki uygulamayı çalıştırdığımızda aşağıdaki ekran görüntüsünde yer alan **Exception** mesajını alırız



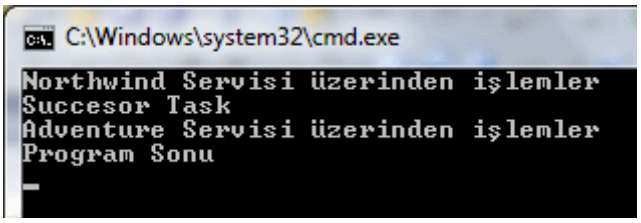
Bu aslında **TaskContinuationOptions enum sabitine** verdiğimiz **OnlyOnFaulted** değeri için söz konusu bir durumdur. *(Aslına bakarsanız ben bu senaryonun çalışmasını beklerdim)*Diğer **enum** sabiti değerlerinde bu tip bir sorun ile karşılaşmasanız da **OnlyOnFaulted** hakkatten bir **Fault** vermektedir

Yazımızın buraya kadarki kısmında kısaca **Task** örnekleri arasındaki ilişkileri sağlamak adına **Continous** tekniklerini ve özellikle **ContinueWhenAll ve ContinueWith** metodlarını irdeledik. Bu iki metoda ek olarak **ContinueWhenAny** isimli bir metodun daha olduğunu belirtmek isterim. Bu metod aslında bir **Task** dizisi içerisindeki **Task'** lerden herhangibiri tamamlandıktan sonra **SuccesorTask** örneğinin devreye girmesi amacıyla tasarlanmıştır. Bu durumu analiz etmek için aşağıdaki kod parçasını göz önüne alabiliriz.

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace TaskContinuation
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Senaryo #3

         Task TaskA = new Task(() =>
         {
            Thread.Sleep(4000);
            Console.WriteLine("Adventure Servisi üzerinden işlemler");
            //throw new FileNotFoundException();
         }
         );

         Task TaskB = new Task(() =>
         {
            Thread.Sleep(2000);
            Console.WriteLine("Northwind Servisi üzerinden işlemler");
            //throw new FileNotFoundException();
         }
);
         Task[] tasks = { TaskA, TaskB };
         Task succesorTask = Task.Factory.ContinueWhenAny(tasks,
           (antecedentTasks) =>
           {
             Console.WriteLine("Succesor Task");
           }
           );

         TaskA.Start();
         TaskB.Start();
```

```
            try
            {
               Task.WaitAll(tasks);
            }
            catch (AggregateException excp)
            {

            }
            succesorTask.Wait();

            #endregion

            Console.WriteLine("Program Sonu");
            Console.ReadLine();
        }
    }
}
```

Uygulama kodunda yer alan **TaskA** ve **TaskB** nesne örneklerine ait kod bloklarından ilk olarak **TaskB** tamamlanacaktır*(Verilen Thread durdurma süreleri gereği)* Buna göre de çalışma zamanında TaskB tamamlanır tamamlanmaz **Succesor Task** bloğu yürütülecektir. Aşağıda görüldüğü gibi.



Böylece geldik bir yazımızın daha sonra Bir sonraki yazıda büyük bir olasılıkla **Task'** ler arası ilişkiler konusuna devam ediyor olacağım. Ama araya başka bir konu da girebilir. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

TaskContinuation.rar (22,47 kb)

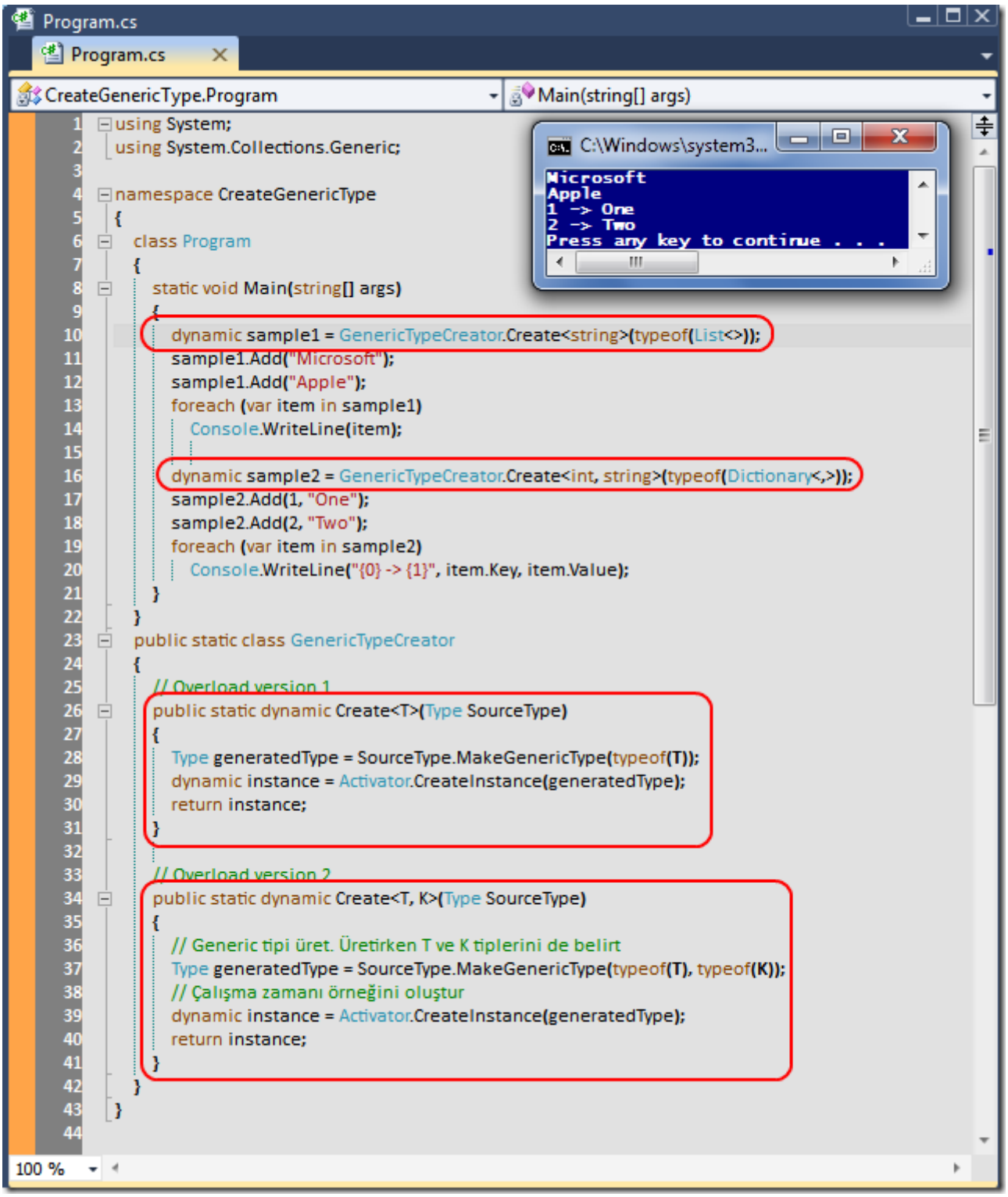
Tek Fotoluk İpucu-40(Sebze Çorbası) (2011-11-17T00:20:00)

c#,c# temelleri,generics,dynamic,dynamic language runtime,reflection,

Merhaba Arkadaşlar,

Hani böyle annemiz zamanında içinde yok yok dedirtecek türden çorbalar yapmıştır. Her çeşit sebzenin konulduğu Hah işte bu fotoğrafta ona benziyor. İçinde generic mimari var, reflection var, dynamic tip kullanımı var Olay gayet basit. çalışma zamanında

generic tipleri dinamik olarak üretip kullanmak istediğinizi düşünün. Bunu nasıl sağlarsınız? İşte basit bir örnek




```
using System;
using System.Collections.Generic;

namespace CreateGenericType
{
    class Program
    {
        static void Main(string[] args)
        {
            dynamic sample1 = GenericTypeCreator.Create<string>(typeof(List<>));
            sample1.Add("Microsoft");
            sample1.Add("Apple");
            foreach (var item in sample1)
                Console.WriteLine(item);

            dynamic sample2 = GenericTypeCreator.Create<int, string>(typeof(Dictionary<,>));
            sample2.Add(1, "One");
            sample2.Add(2, "Two");
            foreach (var item in sample2)
                Console.WriteLine("{0} -> {1}", item.Key, item.Value);
        }
    }
    public static class GenericTypeCreator
    {
        // Overload version 1
        public static dynamic Create<T>(Type SourceType)
        {
            Type generatedType = SourceType.MakeGenericType(typeof(T));
            dynamic instance = Activator.CreateInstance(generatedType);
            return instance;
        }

        // Overload version 2
        public static dynamic Create<T, K>(Type SourceType)
        {
            // Generic tipi üret. Üretirken T ve K tiplerini de belirt
            Type generatedType = SourceType.MakeGenericType(typeof(T), typeof(K));
            // Çalışma zamanı örneğini oluştur
            dynamic instance = Activator.CreateInstance(generatedType);
            return instance;
        }
    }
}
```

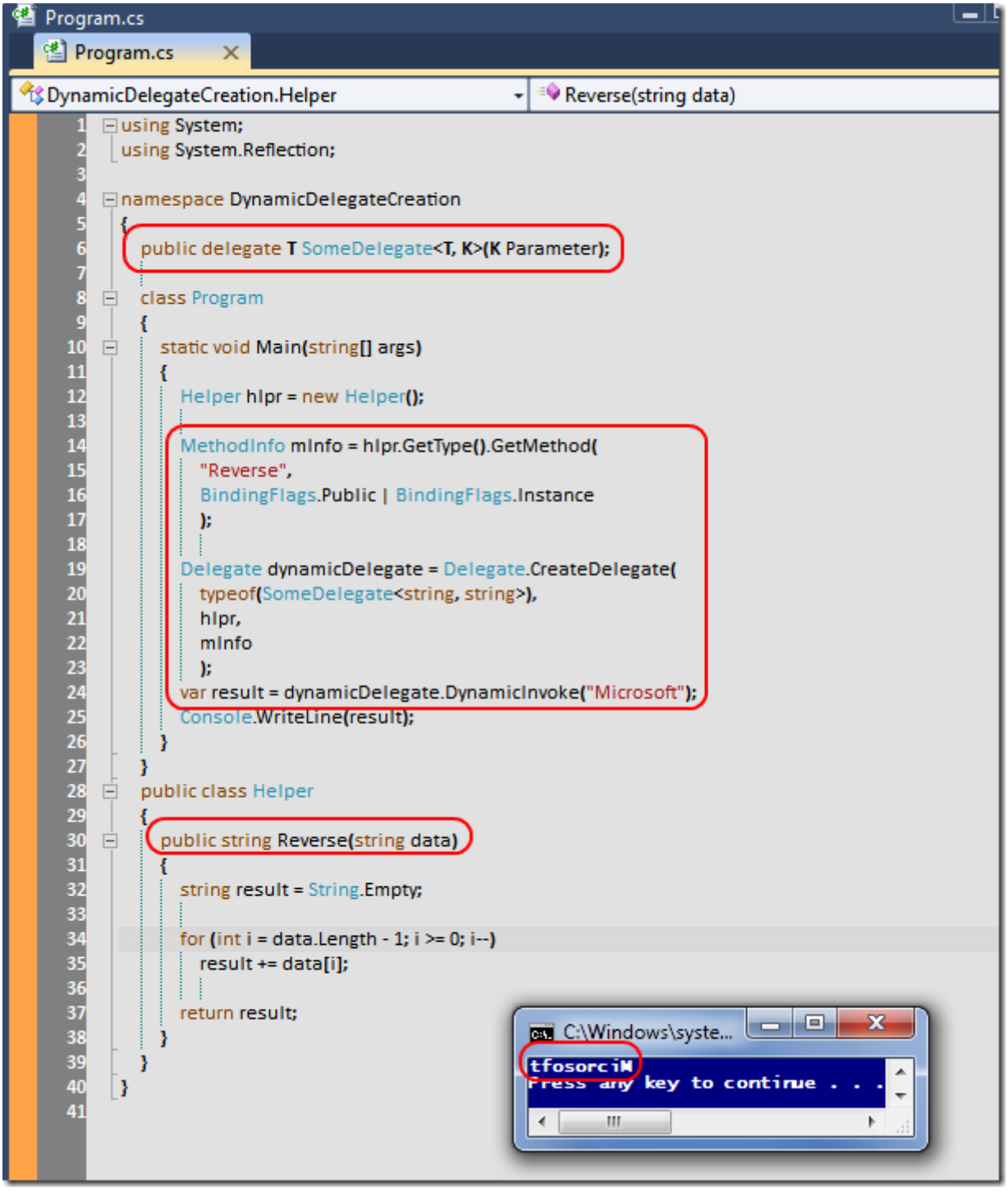
CreateGenericType.rar (24,98 kb)

Tek Fotoluk İpucu-39(Dynamic Delegate Üretmek) (2011-11-14T23:45:00)

c#,c# temelleri,delegate,reflection,method info,

Merhaba Arkadaşlar,

Bazen çalışma zamanına ilişkin yapmamız gereken atraksiyonlar olur. Söz gelimi çalışma zamanında bir delegate tipinin dinamik olarak üretilmesini ve yürütülmesini isteyebiliriz? Peki bu nasıl olacak? İşin içerisine birazcık Reflection katarak tabiki de



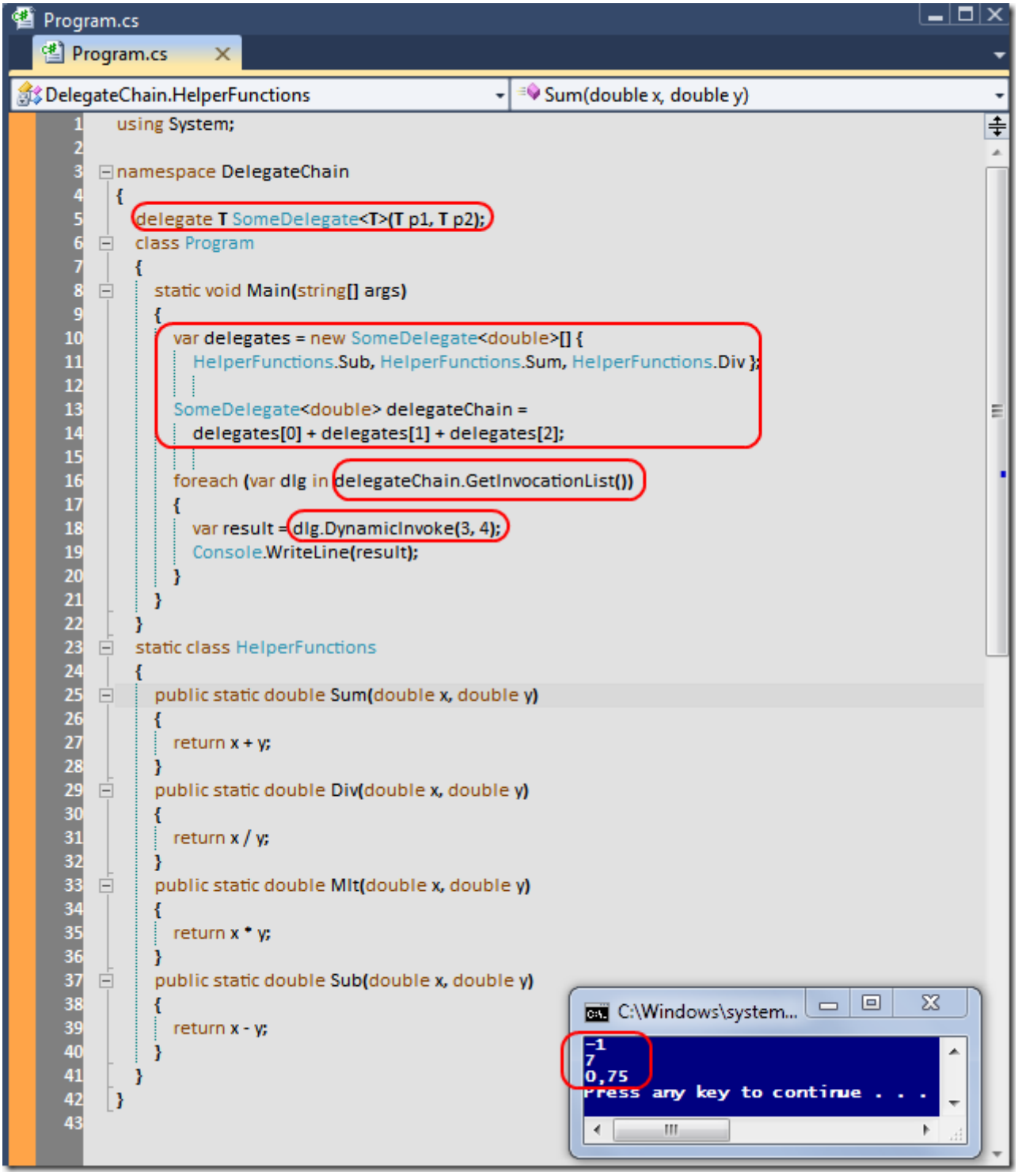
DynamicDelegateCreation.rar (23,25 kb)

Tek Fotoluk İpucu-38(Delegate Chain) (2011-11-13T17:45:00)

c#,c# temelleri,delegate,delegate chain,

Merhaba Arkadaşlar,

Arada sırada temelleri de hatırlamak gerekir değil mi? Söz gelimi bir delegate zincirini nasıl kurar ve aynı parametreler için nasıl çalıştırırsınız? İşte size örnek



DelegateChain.rar (22,77 kb)

SSIS - Programatik Olarak Variable Değeri Set Etmek (2011-11-11T23:59:00)

c#,sql server integration services,ssis,dts,dts package,ssis variables,manageddts,

Merhaba Arkadaşlar,

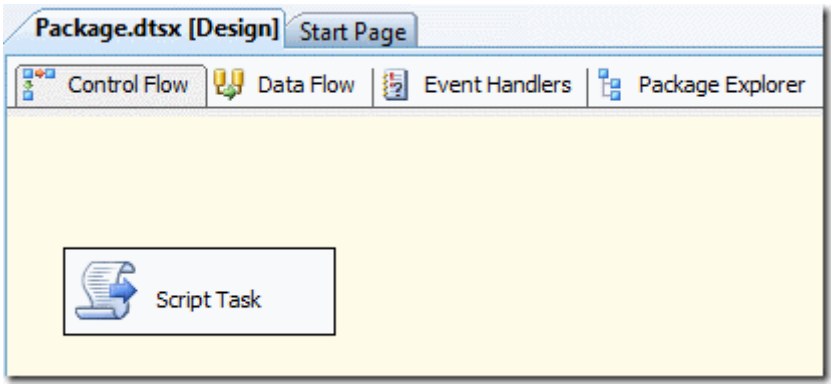
Beni tanıyanlar **SQL** ailesini pek sevmediğimi ve biraz uzak durmaya çalışmak istediğimi bilirler. Ne varki bazen iş hayatının gerçekleri ile karşı karşıya kalırız ve mecburen **SQL** ailesinin bazı fertleri ile yakın ilişkiler içerisine gireriz örneğin ben çalışmakta olduğum bankanın önemli bazı operasyonlarında **SSIS(Sql Server Integration Services)** paketleri ile çalışmak durumundayım. özellikle bankaların metin tabanlı dosya formatlarını sıklıkla tercih ettiklerini biliyoruz.

Ne varki bu ham veri içeriklerinin operasyonel düzeyde ele alınabilmeleri için ilişkisel hale getirilmeleri, bir başka deyişle **SQL Server** gibi ilişkisel veritabanı ortamlarına aktarılmarı gerekmektedir. Pek tabi tersi bir durumda çoğu zaman söz konusu olmaktadır. Bu gibi ihtiyaçlar dahilinde **SSIS(Sql Server Integration Services)** paketleri oldukça kullanışlıdır. özellikle bir tasarım aracının söz konusu olması, zengin kontrol seti ve akış bazlı çalışma modeli önemli avantajlar olarak karşımıza çıkmaktadır. Tabi böyle bir senaryo ve işin içerisinde benim gibi bir .Net geliştiricisi olunca, ister istemez bir **SSIS** paketini programatik olarak değerlendirmek söz konusu olabilmektedir Ben de bu düşünceden yola çıkarak programatik anlamda bir **SSIS** paketinin**değişkenlerine(Variables)** dış ortamdan nasıl erişebileceğimizi ve değiştirebileceğimizi incelemeye çalıştım.

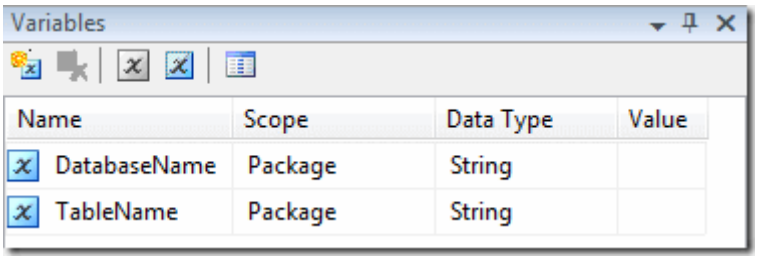
Normal şartlarda bir **SSIS** paketinin çalıştırılması için kullanılabilecek pek çok yol bulunmaktadır. Doğrudan designer aracı ile, komut satırından **Dtcexec.exe** programı ile, bir **SQL Server Job** şeklinde vb...Ancak hangi çalıştırma modeli seçilirse seçilsin bazen söz konusu paketin dış ortamdan parametrik değerler alması ve bunları içerisinde kullanması gerekmektedir. Bunun için de bir kaç yol mevcuttur aslında.

Söz gelimi paket seviyesindeki değişkenleri, pakete ait **XML** bazlı bir konfigurasyon dosyası içerisinde tutabiliriz. Bu sayede basit bir metin editörü yardımıyla söz konusu **XML** dosyası içeriğini değiştirebilir ve paketin yeni paremetre değerleri ile yürütülmesini sağlayabiliriz. Hatta bu **XML** dosyasını **yönetimli kod(Managed Code)** tarafında geliştireceğimiz bir kod parçasıda **XML API'** sini kullanaraktan da ele alabiliriz. Ancak ben bu yazımızda **SSIS** paketinin yönetimli kod tarafında nesnel olarak ele alınması ve bu tekniğe göre ilgili **Variable** değerlerinin değiştirilmesi üzerinde durmaya çalışacağım. Bu amaçla basit bir senaryo üzerinden ilerliyor olacağız.

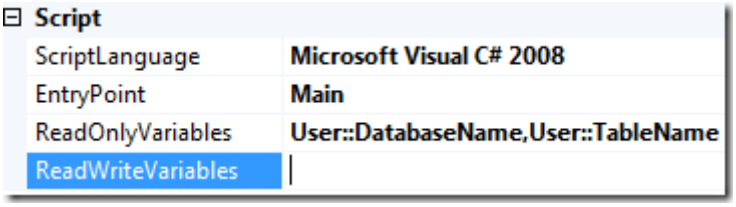
Senaryomuza göre, çok basit ve sembolik bir **SSIS** paketinin kendi içerisinde kullandığı variable' lara başka bir Console uygulaması üzerinden müdahale etmeyi ve paketin yeni ortam değerlerine göre yürütülmesini sağlamayı bekliyor olacağız. İlk olarak senaryomuz gereği aşağıdaki ekran görüntüsünde yer alan **SSIS** paketini geliştirdiğimizi düşünelim.



Paket içerisinde kullanılan **değişkenlerimiz(Variables)** ise şunlardır.



DatabaseName ve **TableName** isimli değişkenler **string** tipindendir ve paket seviyesinde tanımlanmışlardır. Paket içerisinde yer alan **Script Task** tipinden olan **Task** bileşeni ise kendi içerisinde söz konusu değişkenleri **ReadOnly** seviyede kullanmaktadır.



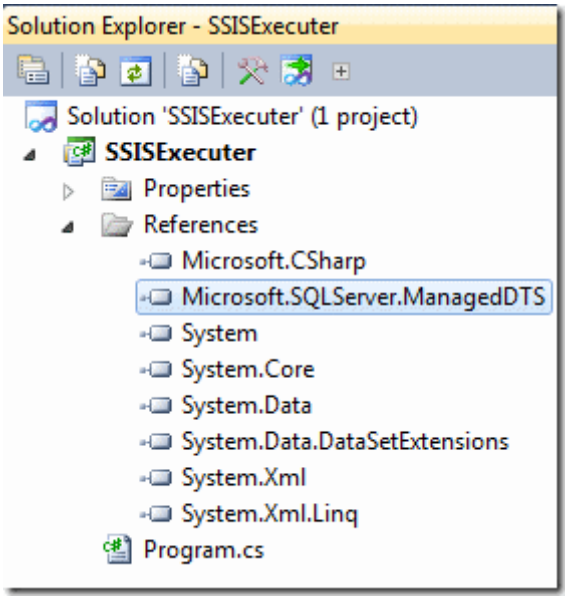
Script kodu ise aşağıdaki gibidir.

```
using System;
using System.IO;

namespace ST_7bc89561acee425798facb4212b1828a.csproj
{
   [System.AddIn.AddIn("ScriptMain", Version = "1.0", Publisher = "", Description = "")]
   public partial class ScriptMain :
Microsoft.SqlServer.Dts.Tasks.ScriptTask.VSTARTScriptObjectModelBase
   {
      #region VSTA generated code

      enum ScriptResults
      {
         Success = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Success,
```

```
        Failure = Microsoft.SqlServer.Dts.Runtime.DTSExecResult.Failure
    };

    #endregion

    public void Main()
    {
        string dbName = Dts.Variables["DatabaseName"].Value.ToString();
        string tbName = Dts.Variables["TableName"].Value.ToString();
        File.WriteAllText(Path.Combine(Environment.CurrentDirectory, "Results.txt"),
String.Format("{0}.{1} için işlemler yapılacak.", dbName, tbName));
        Dts.TaskResult = (int)ScriptResults.Success;
    }
  }
}
```

Bu tabi kobay bir **SSIS** paketi olduğundan ne yaptığının çok fazla önemi yoktur. Ancak senaryomuz gereği **Script** olarak çalıştırılan kod bloğu içerisindeki kullanım şekli önemlidir. **Main** metodunda paket seviyesindeki **DatabaseName** ve **TableName** değişkenleri kullanılmakta olup değerleri bir **Text** dosyaya yazdırılmaktadır. Aslına bakarsanız senaryomuzu test etmek için yeterli bir kod parçasıdır  Bildiğiniz üzere amacımız başka bir .Net uygulamasını kullanarak söz konusu değişkenlerin dış ortamdan set edilmesini sağlamaktı. Şimdi dilerseniz basit bir **Console** uygulaması oluşturarak akışımıza devam edelim.

İlk olarak **SSIS** paketleri üzerinde yönetimli kod tarafını kullanabilmek için ilgili **Assembly'** ın projeye referans edilmesi gerekmektedir. Bu yüzden **C:\Program Files\Microsoft SQL Server\100\SDK\Assemblies** klasörü içerisinde yer alan **Microsoft.SQLServer.ManagedDTS.dll**' ini projeye referans etmemiz yeterlidir. *(SSIS aslında eskiden DTS olarak anılan bir alt yapıdır. Bu yüzden assembly adlarına şaşırmayın)*



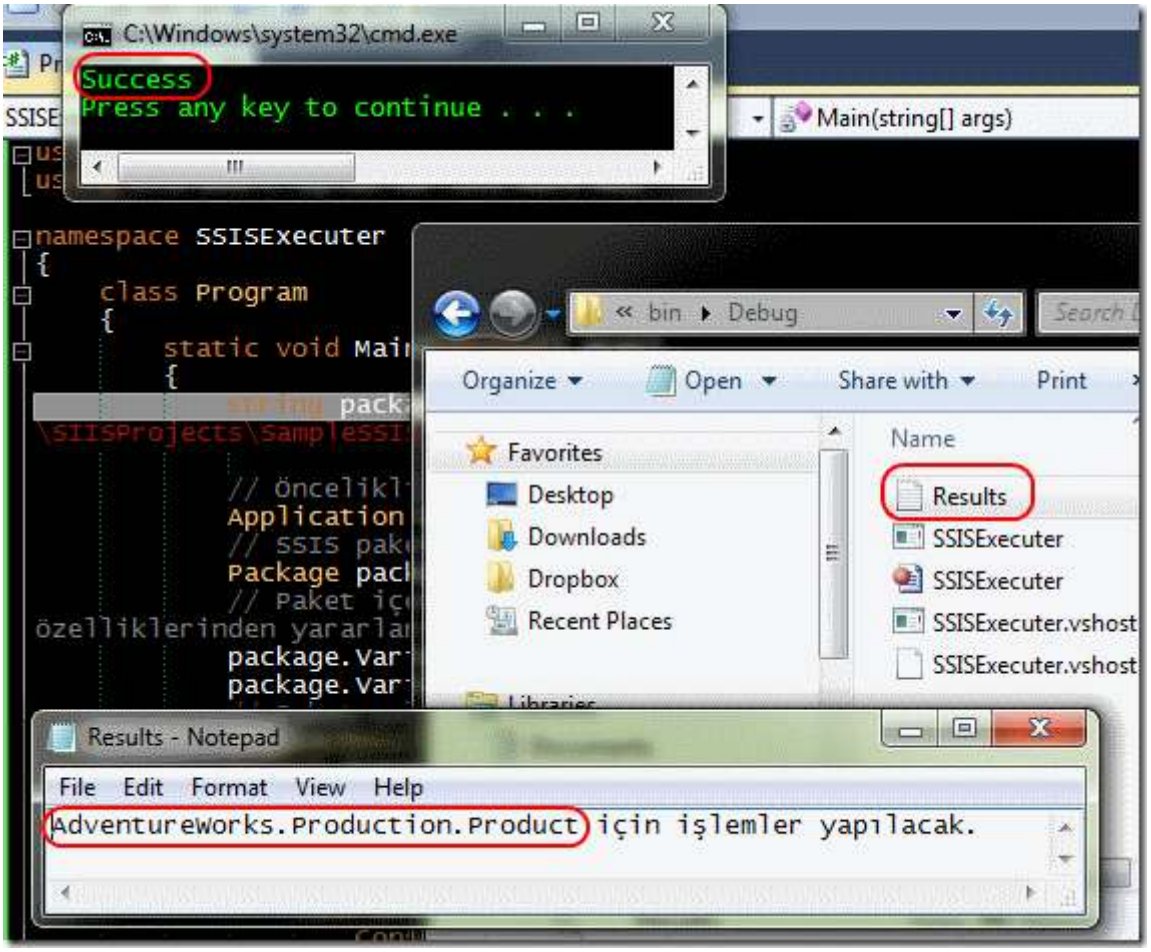
Bundan sonrası ise oldukça basit. Hatta zevkli bir oyun gibi diyebilirim

İşte örnek kod parçamız.

```
using System;
using Microsoft.SqlServer.Dts.Runtime;

namespace SSISExecuter
{
   class Program
   {
      static void Main(string[] args)
      {
         string packagePath = @"C:\Users\buraksenyurt\Documents\Visual Studio
2008\Projects\SIISProjects\SampleSSIS\SampleSSIS\bin\Package.dtsx";

         // öncelikli olarak paketi yüklemek için kullanılacak uygulama nesnesi oluşturulur
         Application app = new Application();
         // SSIS paketi yüklenir
         Package package = app.LoadPackage(packagePath, null);
         // Paket içerisinde tanımlanmış olan değişkenlere erişilir ve Value özelliklerinden
yararlanılarak ilgili değerleri set edilir
         package.Variables["DatabaseName"].Value = "AdventureWorks";
         package.Variables["TableName"].Value = "Production.Product";
         // Paket çalıştırılı ve sonucu alınarak değerlendirilir
         DTSExecResult result = package.Execute();
         switch (result)
         {
            case DTSExecResult.Canceled:
               Console.WriteLine("Canceled");
```

```
                break;
            case DTSExecResult.Completion:
                Console.WriteLine("Completion");
                break;
            case DTSExecResult.Failure:
                Console.WriteLine("Failure");
                break;
            case DTSExecResult.Success:
                Console.WriteLine("Success");
                break;
            default:
                break;
        }
    }
  }
}
```

Görüldüğü üzere ilk olarak paketin ilgili adresten yüklenmesi sağlanmış ve arından **Variables** özelliği üzerinden **DatabaseName** ve **TableName** parametrelerine yeni değerleri aktarılmıştır. Son olarakta ilgili paket çalıştırılmıştır. Paketin çalıştırıcısı bu **Console** uygulaması olduğu için text dosyasının çıktısı da **exe**' nin bulunduğu dosya adresi olacaktır. İçeriği ise tam istediğimiz gibidir 



Sonuç olarak bir **SSIS** paketinin iç değişkenlerine dış ortamdan değer atamanın farklı bir yolunu görmüş olduk. **Managed** tarafta SSIS paketlerini daha da etkin yönetebilmemiz de mümkündür. Hatta bu kütüphaneyi kullanarak özellikle görsel SSIS yürütücüleri geliştirebilirsiniz. Bir düşünün Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

SampleSSIS.rar (25,47 kb)

SSISExecuter.rar (25,64 kb)

Nedirtv.com Webinerleri Yeniden Başlıyor (2011-11-11T16:46:00)

nedirtvcom,webiner,nedirtv,

Nedirtv webinerlerine yeniden başlıyor.

Kasım ayı içerisinde gerçekleştirilecek olan üç webinere ait bilgilere aşağıdan ulaşabilirsiniz.

Not: Webinerlere katılmak için sadece bir defaya mahsus bilgisayarınıza Office Live Meeting'i kurmanız gerekmektedir. Webiner saatinden 15 dakika önce webiner linkine tıkladığınızda, gerekli kurulum için yönlendirmeler yapılacaktır.

Linkedin etkinliğine kayıt olun

Konu: NoSQL Nedir? - MongoDB ile .NET Kardeşliği
Tarih: 14 Kasım Pazartesi 21:00
Konuşmacı: İbrahim ATAY
Link: https://www.livemeeting.com/cc/mvp/join?id=N7732S&role=attend

Konu: SQL Server - Stored Procedure ve Function
Tarih: 21 Kasım Pazartesi 21:00
Konuşmacı: Görkem SEZGİN
Link: https://www.livemeeting.com/cc/mvp/join?id=Q24KBF&role=attend

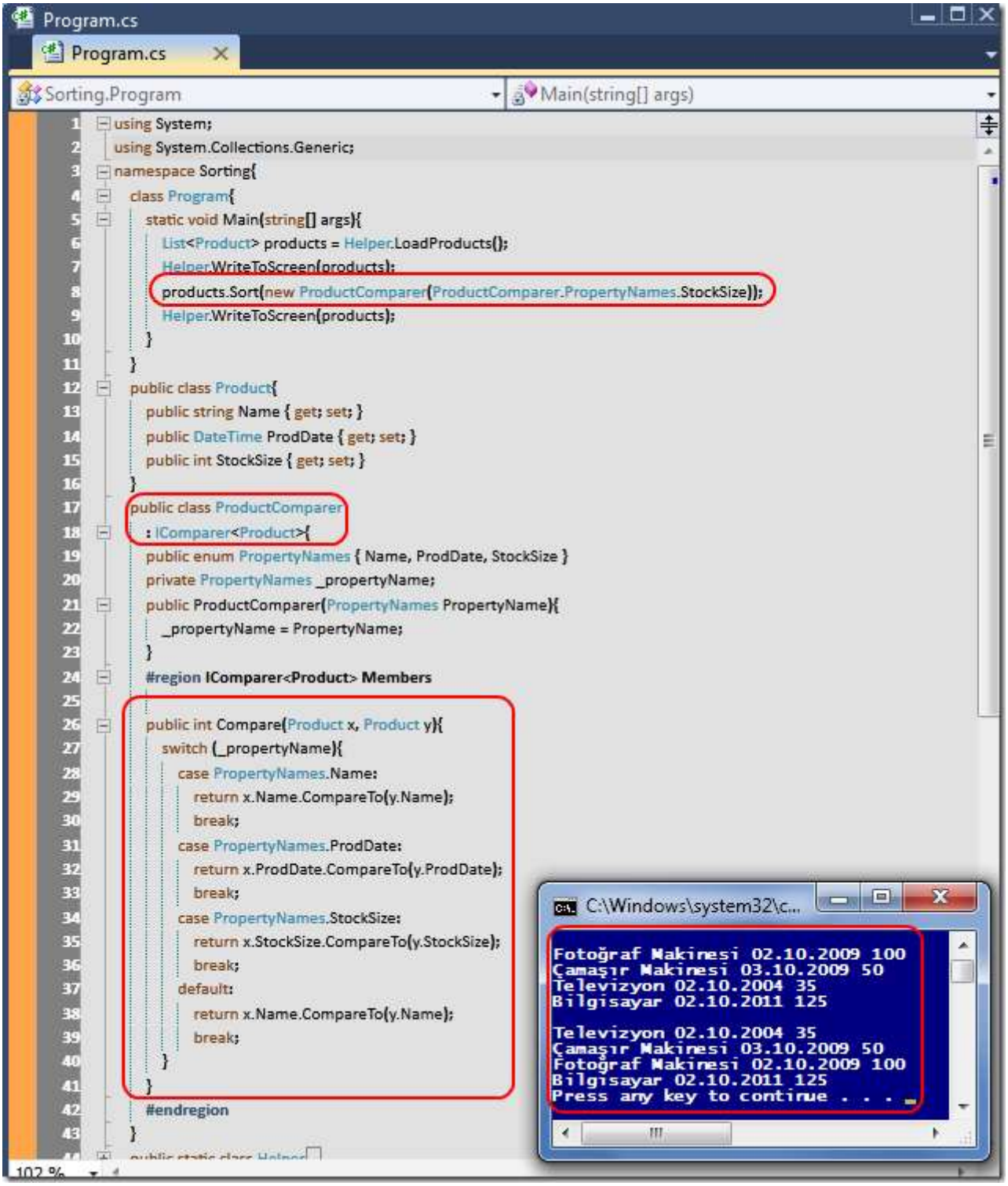
Konu: HTML 5
Tarih: 28 Kasım Pazartesi 21:00
Konuşmacı: Cemil UZUN
Link: https://www.livemeeting.com/cc/mvp/join?id=H7GTWK&role=attend

Tek Fotoluk İpucu-37(Faydalı Interface Tiplerinden IComparer) (2011-11-05T18:39:00)

c#,c# temelleri,icomparer,list,sorting,interface,

Merhaba Arkadaşlar,

.Net içerisinde pek çok faydalı Interface tipi bulunmaktadır. örneğin kendi tiplerinizin sıralama işlemlerini öğrenebilmesi için kullanabileceğimiz IComparer<T>. Nasıl kullanıldığını merak ediyor musunuz? İşte size basit bir fotoğraf



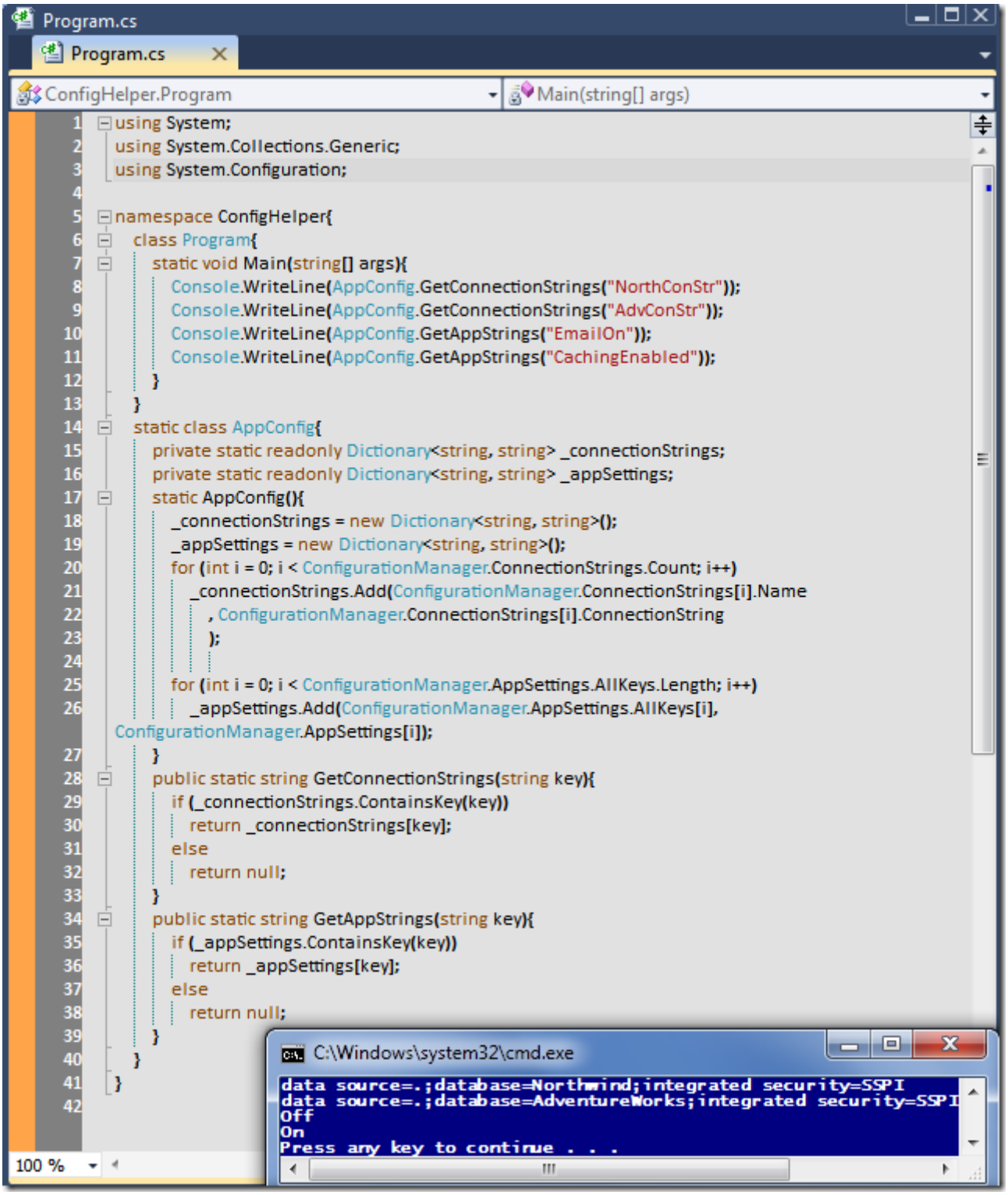
Sorting.rar (24,13 kb)

Tek Fotoluk İpucu-36(Config Dosyasına Kolay Ulaşım) (2011-10-24T23:30:00)

c#,c# temelleri,configuration manager,configuration api,app.config,.net framework,

Merhaba Arkadaşlar,

.Net Framework 2.0 ile birlikte gelen Configuration API' sini hepimiz biliyoruzdur. Bu API sayesinde config dosya içeriklerinin Managed karşılıkları olan tiplere ulaşmamız son derece kolay. Aslına bakarsanız pek çok uygulamada config dosyası içerisinde **ConnectionStrings** ve **AppSettings** kısımlarını sıklıkla kullandığımızı görmekteyiz. Bu içeriklere daha efektif ve performanslı erişim için belki bir Wrapper tip işimizi görebilir. Nasıl mı? 



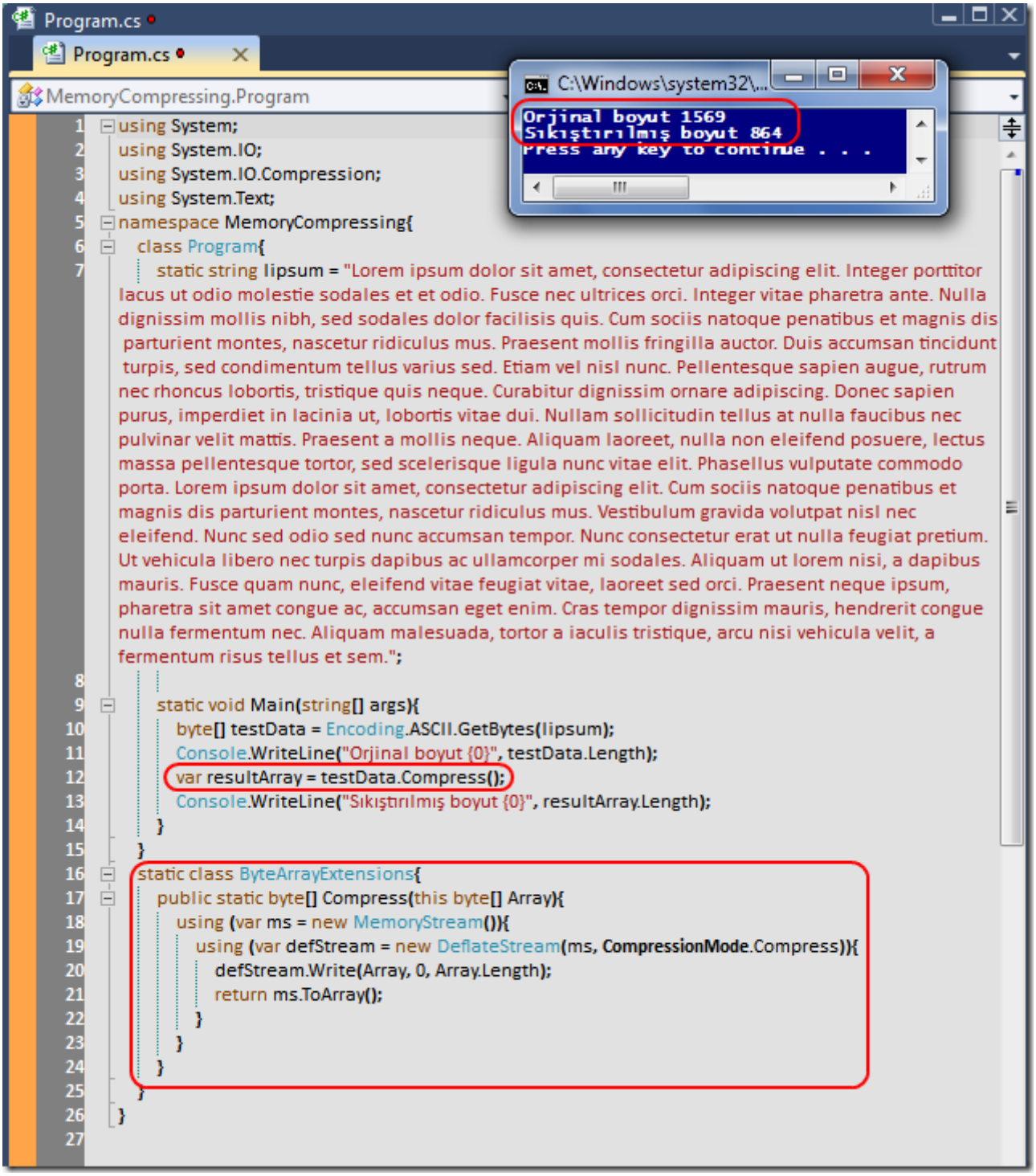
ConfigHelper.rar (24,02 kb)

Tek Fotoluk İpucu-35(DeflateStream ile Sıkıştırmak) (2011-10-21T17:13:00)

c#,c# temelleri,.net framework,deflatestream,io,compress,decompress,extension methods,

Merhaba Arkadaşlar,

Diyelim ki uygulama içerisinde kullandığınız büyük boyutlu bir byte dizisi var. Aslında bu diziyi bellek üzerinde sıkıştırarak daha az yer tutacak şekilde de kullanma şansınız olabilir. DelfateStream tipi bu anlmada işinize yarayacak Compress ve Decompress metodlarını içermektedir. İşte size örnek bir kullanım. Lorem Ipsum' u byte seviyesinde sıkıştırıyoruz. E decompress kısmı da size kaldı.



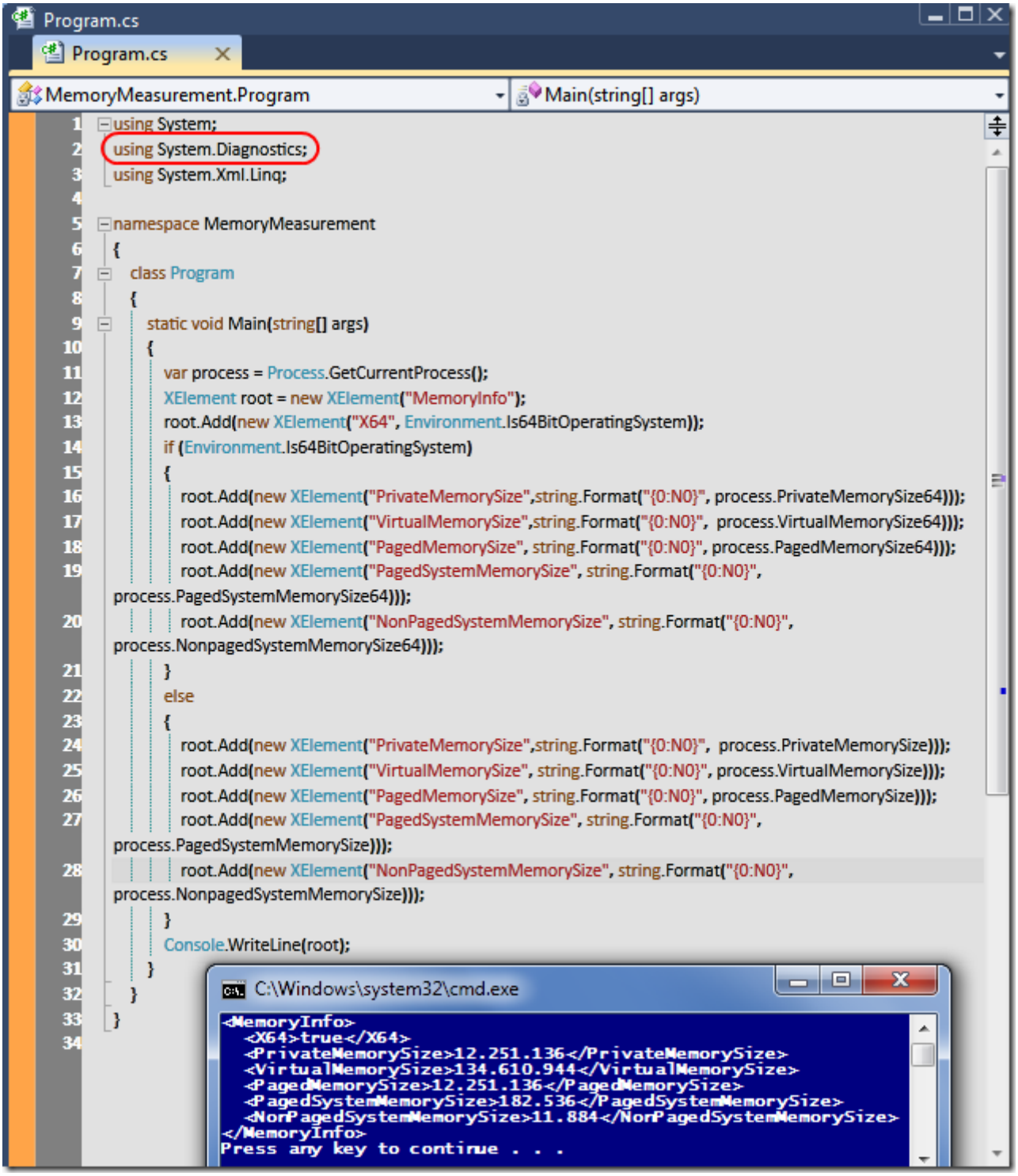
MemoryCompressing.rar (25,22 kb)

Tek Fotoluk İpucu-34(Güncel Process için Bellek Bilgileri) (2011-10-19T00:11:00)

c#,c# temelleri,.net framework,process,xlinq,xml,

Merhaba Arkadaşlar,

çalıştırdığımız .Net tabanlı uygulamaların anlık bellek tüketimlerini kod içerisinden ölçümlemek ve hatta loglamak iyi bir fikir olabilir. Hatta bu çıktıyı XML formatında dış dünyaya da sunabiliriz. Basit anlamda aşağıdaki fotoğraf size ipucu verecektir kanaatindeyim.



MemoryMeasurement.rar (22,15 kb)

Barrier Class, Sıralama Algoritmaları ve At Yarışı (2011-10-17T21:47:00)

parallel programming,barrier class,task parallel library,sort algoritmaları,

Merhaba Arkadaşlar,

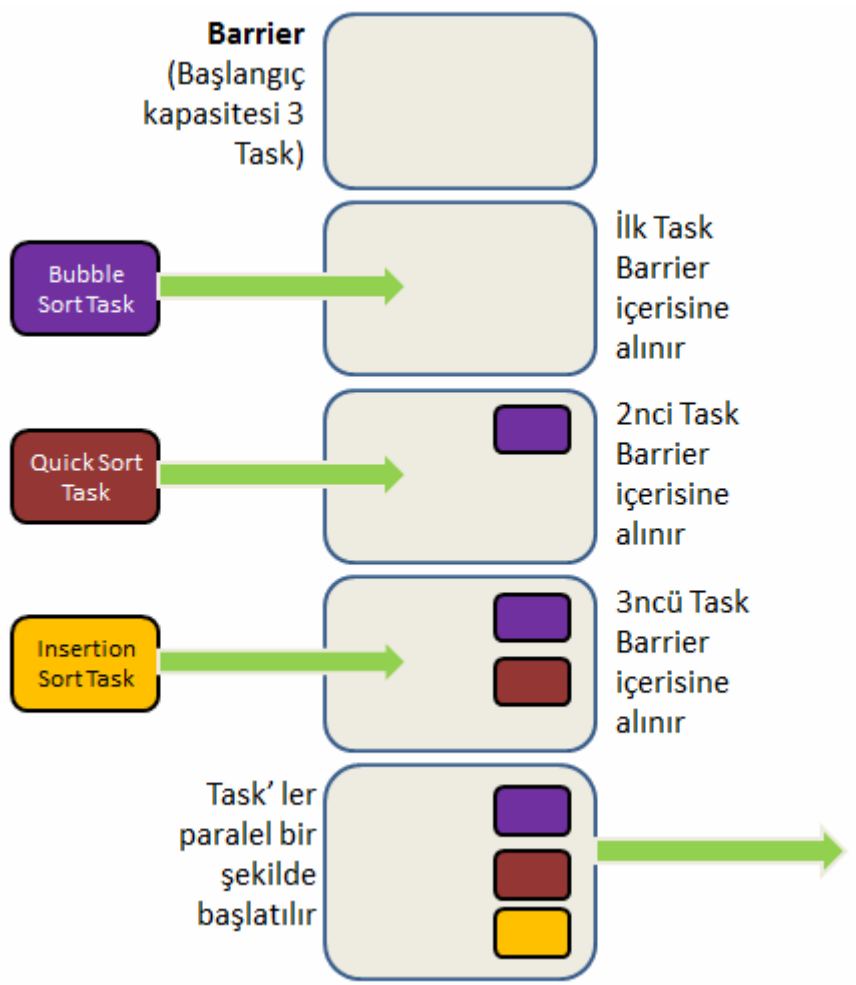
At yarışlarına pek ilgim yoktur aslında ama tam da bu günlerde okuduğum kitap nedeniyle, paralel programlama ile aralarında sıkı bir ilişki olduğunu ifade edebilirim Bildiğiniz üzere bu yarışların pek çok meraklısı bulunmaktadır. özellikle yarışları stadyumdan seyredenler oldukça heyecanlıdır. Gerçi yarışın başlamasından önceki tahminler tam başlangıç anında yerini endişeye bırakır. Bizim gibi yazılımcılar için önemli olan ise başlangıç anıdır. Neden mi? Bazen bir yarışa başlanırken, yarışa iştirak eden katılımcıların start düzlüğünde bir arada yer almalarını bekleriz. At yarışlarındaki başlangıç kapıları bu işe yaramaktadır. Tahmin edeceğiniz gibi bu kapılar atların başlangıç işareti gelmeden hareket etmelerini engellemek üzere tasarlanmışlardır. çok doğal olarak ve pek tabi ki aynı durum program ortamı içerisinde yer alan ve paralel çalıştırılması düşünülen metodlar içinde geçerli olabilir Bir başka deyişle bazı senaryolarda paralel olarak yürütülmesi istenen metodların, çalışmaya başlamadan önce bir kutu içerisinde de paralel olarak dizilmeleri ve kutu beklenen limiti aştığında başlatılmaları istenebilir. Yani paralel çalıştırılması istenen fonksiyonellikler hazırlanıp belirli bir başlangıç noktasına doğru senkronize edilir ve sonrasında belirli bir kurala göre*(örneğin kutunun limitini doldurması gibi)* başlatılırlar.

Durumu daha iyi analiz edebilmek için dilerseniz basit bir senaryo üzerinden ilerlemeye çalışalım. Malumunuz günümüz üniversitelerinde özellikle algoritma derslerinde en çok sorulan, okutulan, ezberletilen konuların başında sıralama teknikleri gelmektedir. **Bubble Sort, Quick Sort, Insertion Sort** vb…Eminim çoğumuz bunları ezberlemek ve sınavlarda çıkacak diye uzun geceler boyu hazırlanmak zorunda kalmışızdır Peki ya bu sıralama algoritmalarını bir yarışa sokmak ister miydiniz? Aslında her birini bir at olarak düşünüp aynı anda yarışı başlatmak istemez miydiniz dersem çok daha uygun olacaktır Bunu gerçekleştirirken aslında paralel programlama tekniklerinden yararlanma şansımız bulunmaktadır. Her bir sıralama algoritması için birer **Task** nesne örneği üretip bunları aynı anda çalıştırmamız yeterli olacaktır…Aynı anda

İşte **Barrier** sınıfını kullanmak için güzel bir fırsat. Yazımızın odak noktası hangi sıralama algoritmasının hızlı çalıştığını tespit etmekten ziyade, **Barrier** sınıfını kullanarak ilgili sıralama algoritma metodlarını aynı anda başlatmaktır. Bu amaçla ben örnek olarak 3 adet sıralama algoritmasını baz almayı uygun gördüm. **Bubble Sort, Quick Sort** ve **Insertion Sort…**

Hedefimiz bu algoritmaları içeren metodları birer **Task** olarak tanımlamak ve **start** düzlüğünde aynı kapı içerisine yerleştirerek aynı anda çalıştırılmalarını

sağlamaktır. Kabaca **Barrier** sınıfını kullanarak gerçekleştireceğimiz örneğimizin çalışma zamanındaki akışı belkide aşağıdaki grafik ile ifade edilebilir.



Barrier sınıfı öncellikli olarak bir kapasite bildirimi ile çalışmaktadır. Bu kapasite aslında kutu içerisine dahil edilecek olan **Task** sayısını bildirmektedir. Söz konusu **Task** sayısına ulaşıldığında ilgili metodların aynı anda başlatılması söz konusu olacaktır*(Tabi arkadan gelen başka bir Task daha olursa, kapıdaki örnekler bir anda çıkana kadar beklemek zorunda kalacaktır)* Durumu elbetteki örnek kod parçası üzerinden anlamamız daha uygundur. Bu amaçla basit bir **Console** uygulaması açıp aşağıdaki kod içeriğini yazarak ilerleyebiliriz.

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;

namespace SortingTest
{
   class Program
   {
```

```
static string insertionSortStartTime=String.Empty;
static string quickSortStartTime = String.Empty;
static string bubbleSortStartTime = String.Empty;

static void Main(string[] args)
{
   // önce rastgele sayı koleksiyonumuz oluşturuluyor
   List<int> testCollection = CreateRandomNumbers(100000,1, 100000);

   #region Barrier Kullanımı

   // 3 metodluk bir başlangıç kutusu tanımlıyoruz.
   Barrier _barrier = new Barrier(3);

   // Bubble Sort metodu için bir Task üretiliyor ve Task ilk olarak Barrier içerisine
ilave ediliyor
   Task bubbleSortingTask = new Task(
      () =>
      {
         _barrier.SignalAndWait();
         BubbleSorting(testCollection);
      }
   );
   bubbleSortingTask.Start();

   // Quick Sort metodu için bir Task üretiliyor ve Task içerisinde yine Barrier' e bir
ekleme işlemi gerçekleştiriliyor
   Task quickSortingTask = new Task(
      () =>
      {
         _barrier.SignalAndWait();
         QuickSorting(testCollection, 0, testCollection.Count - 1);
      }
   );
   quickSortingTask.Start();

   // Insertion Sort metodu için bir Task üretiliyor ve Barrier içerisine ekleniyor
   Task insertionSortingTask = new Task(
      () =>
      {
        _barrier.SignalAndWait();
         InsertionSorting(testCollection);
      }
   );
   insertionSortingTask.Start();
```

```
        // Task' lerin tamamlanması uzun sürebilir o yüzden bekleyelim
        Task[] tasks = { bubbleSortingTask, quickSortingTask, insertionSortingTask
};
        Task.WaitAll(tasks);

        Console.WriteLine("Bubble : {0}\nQuick : {1}\nInsertion :
{2}",bubbleSortStartTime,quickSortStartTime,insertionSortStartTime);

        #endregion

        #region Sequential çalıştırma

        bubbleSortStartTime = String.Empty;
        quickSortStartTime = String.Empty;
        insertionSortStartTime = String.Empty;

        Console.WriteLine("\nSequential çalıştırma\n");
        // Bubble algoritmasına göre sıralama
        var bubbleOrdered = BubbleSorting(testCollection);
        // Quick sort algoritmasına göre sıralama
        var quickOrdered = QuickSorting(testCollection, 0, testCollection.Count - 1);
        // Insertion sort algoritmasına göre sıralama
        var insertionOrdered = InsertionSorting(testCollection);

        Console.WriteLine("Bubble : {0}\nQuick : {1}\nInsertion : {2}",
bubbleSortStartTime, quickSortStartTime, insertionSortStartTime);

        #endregion
    }

    /// <summary>
    /// Bubble sıralama algoritmasına göre sayı dizisini sıralar
    /// </summary>
    /// <param name="Numbers">Sıralanacak olan sayı koleksiyonu</param>
    /// <returns>Parametre olarak gelen Numbers koleksiyonunun sıralanmış
halidir</returns>
    static List<int> BubbleSorting(List<int> Numbers)
    {
        if (String.IsNullOrEmpty(bubbleSortStartTime))
            bubbleSortStartTime = DateTime.Now.ToLongTimeString();

        for (int i = 0; i < Numbers.Count - 1; i++)
        {
            for (int j = 1; j < Numbers.Count - i; j++)
            {
```

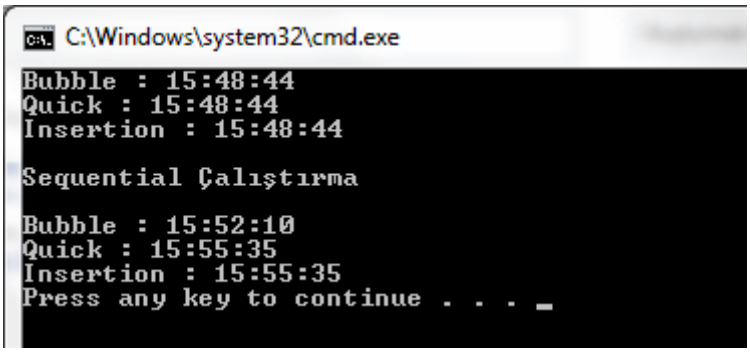
```
            if (Numbers[j] < Numbers[j - 1])
            {
               int temporary = Numbers[j - 1];
               Numbers[j - 1] = Numbers[j];
               Numbers[j] = temporary;
            }
         }
      }

      return Numbers;
   }

   /// <summary>
   /// QuickSort veya Pivot Sort algoritmasına göre sıralama işlemini yapan metoddur.
   /// </summary>
   /// <param name="Numbers">Sıralanacak olan sayı koleksiyonu</param>
   /// <param name="LeftValue">Sol değer</param>
   /// <param name="RightValue">Sağ değer</param>
   /// <returns>Sayı koleksiyonunun sıralanmış hali</returns>
   static List<int> QuickSorting(List<int> Numbers, int LeftValue, int RightValue)
   {
      if (String.IsNullOrEmpty(quickSortStartTime))
         quickSortStartTime = DateTime.Now.ToLongTimeString();

      int i = LeftValue;
      int j = RightValue;
      double pivotValue = ((LeftValue + RightValue) / 2);
      int x = Numbers[Convert.ToInt32(pivotValue)];
      int w = 0;
      while (i <= j)
      {
         while (Numbers[i] < x)
         {
            i++;
         }
         while (x < Numbers[j])
         {
            j--;
         }
         if (i <= j)
         {
            w = Numbers[i];
            Numbers[i++] = Numbers[j];
            Numbers[j--] = w;
         }
```

```
    }
    if (LeftValue < j)
    {
        QuickSorting(Numbers, LeftValue, j);
    }
    if (i < RightValue)
    {
        QuickSorting(Numbers, i, RightValue);
    }

    return Numbers;
}

/// <summary>
/// InsertionSort algoritmasına göre sayı koleksiyonunu sıralar
/// </summary>
/// <param name="Numbers">Sıralanacak olan sayı koleksiyonu</param>
/// <returns>Sayı koleksiyonunun sıralanmış hali</returns>
static List<int> InsertionSorting(List<int> Numbers)
{
    if (String.IsNullOrEmpty(insertionSortStartTime))
        insertionSortStartTime = DateTime.Now.ToLongTimeString();

    for (int j = 1; j < Numbers.Count; j++)
    {
        int keyValue = Numbers[j];
        int i = j - 1;
        while (i >= 0 && Numbers[i] > keyValue)
        {
            Numbers[i + 1] = Numbers[i];
            i = i - 1;
        }
        Numbers[i + 1] = keyValue;
    }

    return Numbers;
}

/// <summary>
/// Sıralama testlerine tabi tutulacak sayı koleksiyonunu üretir
/// </summary>
/// <param name="NumberCount">Kaç adet sayıdan oluşacak</param>
/// <param name="StartValue">Rastgele sayı üretici için minimum değer</param>
/// <param name="EndValue">Rastgele sayı üretici için maksimum değer</param>
/// <returns>Oluşan sayı koleksiyonu</returns>
```

```
    static List<int> CreateRandomNumbers(int NumberCount,int StartValue, int
EndValue)
    {
        List<int> collection = new List<int>();
        Random rnd = new Random();
        for (int i = 0; i < NumberCount; i++)
        {
            collection.Add(rnd.Next(StartValue,EndValue));
        }
        return collection;
    }
  }
}
```

Aslında uygulamamız çok karmaşık gözükse de oldukça basit temellere dayanıyor. Sıralama algoritmalarının üçü için ve bir de rastgele sayı üretimi için tasarlanmış metodlarımız bulunmaktadır. Asıl önemli olan kısım ise **Main** metodu içerisinde yaptıklarımızdır. Dikkat edileceği üzere ilk olarak **Barrier** nesne örneği kullanılarak bir akış gerçekleştirilmektedir.

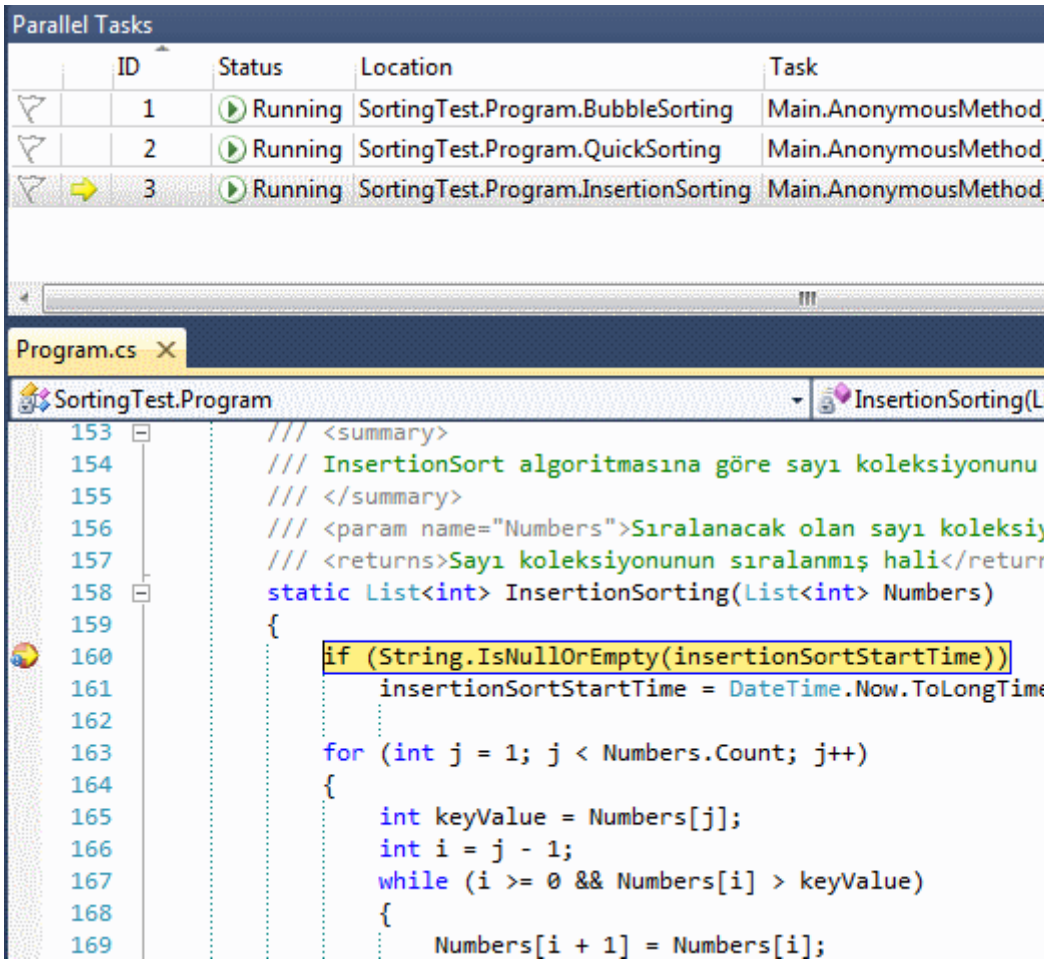
Barrier sınıfına ait nesne örneği üretildikten sonra, ilgili **Task** örneklerinin buraya dail olmaları için, **anonymous** metodlarda **SignalAndWait** fonksiyonuna birer çağrıda bulunulduğuna dikkat edelim Sonrasında **Task** nesne örneklerinin **Start** metodları çağırılıyor. Ne var ki **Barrier** bloğunun kapasitesi dolana kadar ilgili metodlar yürütülmeyecektir. Zaten söz konusu **Task**örneklerinin **Barrier** bloğuna dahil edilme sebebi de kapasite dolumundan sonra aynı anda çalışmaya başlamalarının istenmesidir. Uygulamayı çalıştırdığımızda aşağıdaki ekran görüntüsündekine benzer sonuçlar aldığımızı görebiliriz.

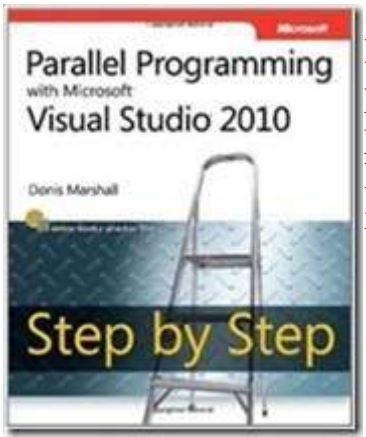


Görüldüğü üzere **Barrier** kullanımı nedeni ile ilk üç **Task** örneğinin tam olarak aynı anda başlatılmaları söz konusudur*(Aslında bakarsanız milisaniye cinsinden de kontrol edilmeleri gerekmektedir)* Diğer yandan **Sequential** çalışma da elbetteki metodlar başlatıldıkları sıra ile yürütülmektedirler. Burada **Quick Sort** algoritmasının gerçekten çok hızlı olması nedeni ile işini çok kısa sürede bitirdiğini ve hemen **Insertion Sort** metoduna geçilebildiğini vurgulamak isterim.

Sanırım **Barrier** tipinin kullanımını biraz olsun kavrayabilmişizdir. Konuyu daha net kavrayabilmek için uygulamayı **Debug** modda çalıştırmanızı öneririm. özellikle **Sort** metodlarının başlangıç noktalarına **Breakpoint** koyarsanız, **Main** metodu içerisinde ilgili **Task** nesne örnekleri üzerinden **Start** metodları çağırılsa bile **Barrier** bloğunun kapasitesi dolana kadar bu **breakpoint** noktalarına gelinemediğini görüyor olacaksınız. Burada eğer **Debug Mod**' da **Parallel Tasks** penceresine bakarsanız **Task** örneklerinin **Start** metodlarına yapılan çağrılardan sonra **Status** olarak **Running'** e geçtikleri görülecektir ki bu sizi yanıltmamalıdır. Bu sebepten örneği **Debug** modda izlemeniz son derece önemlidir

Bakın ben çalışma zamanında durumu analiz ettiğimde **Barrier** bölgesine son eklenen **Insertion Sort** algoritma metodundan sonra, bu son eklenen ilk olmak üzere işleyişin başlayabildiğin gördüm. Bir başka deyişle **Barrier** için belirtilen 3 kapasite değeri **InsertionSort** metodu eklenene kadar dolmadığından hiç bir metod başlatılmamıştır.





Parallel programlama ile ilişkili olarak bir kaç yazı yazmayı daha planlamaktayım. İçeride gözden kaçan oldukça önemli ve hayati pek çok konu var. özellikle de gelecek nesil .Net platformunda bu konunun ne kadar önemli olduğunu düşünürsek geliştirici olarak iyi hazırlanmamız gerektiği kanısındayım. Aslında ben şu

sıralarda Parallel Programming Step by Step isimli kitabı takip ediyorum. İnce bir kitap ve sizlere de şiddetle tavsiye ederim. Gerçi ince olduğuna aldanmayın lütfen. Yeteri kadar doyurucu bilgi ve kod örneğini barındırmakta. Eğer paralel programlamaya ilgi duyuyorsanız tabi Buraya kadar sabırla okuduğunuz için teşekkür ederken tekrardan görüşünceye dek hepinize mutlu günler dilerim.

SortingTest.rar (29,66 kb)

WCF Data Services- Annotations Builder (2011-10-14T23:00:00)

wcf data services,astoria,wcf,annotations,wcf data services october 2011 ctp,

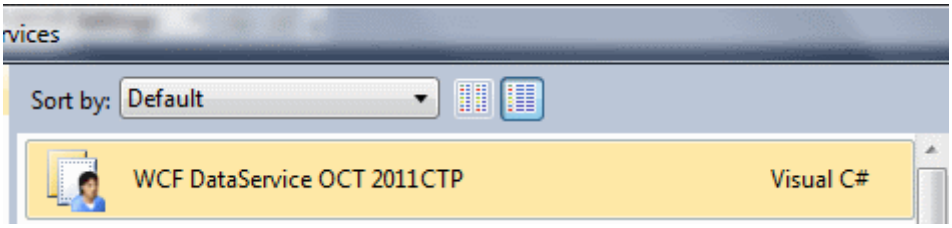
Merhaba Arkadaşlar,

Yağmurlu bu sonbahar günlerinde eminim ki kimse hüzünlenmek, kara kara düşünmek istemez. Ama bazen o kadar garip sorunlar ile karşı karşıya kalırız ki…Ne yapacağımızı bilemeyiz ve kara kara düşünürüz. Mesela ben bu girişi yazdığım sırada, aslında tamamlamış olduğum makalenin sonucunda bir yere varamamış ve beklentilerimi karşılayamamış birisi modundayım. Neden mi? Gelin anlatayım.

Gün geçmiyor ki **Microsoft** ürünlerinde yeni bir sürüm, yeni bir güncelleme görmeyelim Şurada daha iki gün oldu **Entity Framework 4.2 RC** ile ilişkili bir şeyler yazmaya çalışmıştım. Bu gün de ne duyduk dersiniz? WCF Data Services October 2011 CTP yayınlanmış Biraz sitemkar bir giriş oldu ama boşverin…

çok doğal olarak hemen arkasında söz konusu yeni **CTP** ile ilişkili ilk blog girişleri de **Team Blog** üzerinden yayınlanmaya başladı. Ben de bunun üzerine gelen yeniliklerden birisi olan**Vocabularies** kavramını özellikle getirildiği Annotations yapısı ile birlikte incelemeye ve anlamaya çalıştım. Dilerseniz yine tümden gelimden hareket edelim ve önce senaryomuzu geliştirip sonunda elde ettiğimiz*(ve hatta elde edemediğimiz)* bulguları değerlendirelim. *(Bundan sonraki kısımda daha önceden **WCF Data Service** ile geliştirme yaptığınızı ve biraz aşina olduğunuzu varsayarak ilerleyeceğim)*

Tabi ilk olarak WCF Data Services October 2011 CTP adresinden son sürümü indirmeniz gerekiyor. Bunu yaptığımız takdirde **Visual Studio 2010** ortamına aşağıdaki ekran görüntüsünde yer alan yeni bir proje öğesinin eklendiğini görebiliriz.



Bildiğiniz üzere **WCF Data Service'** ler veri odaklı çalışan, **HTTP** protokolünün **GET,POST,PUT,DELETE** metodlarına cevap verebilen ve günümüzde **OData** formatına da destek sağlayan özelleştirilmiş **WCF(Windows Communication Foundation)** Servisleridir. **Data Centric** özellikleri nedeniyle de genellikle **Ado.Net Entity Framework** tabanlı ORM modelleri üzerinden sunulmaktadırlar. Bu nedenle örnek uygulamamızda çok basit olarak **AdventureWorks** veritabanındaki **Product** tablosunu modelleyen bir **Entity Data Model** olduğunu düşünerek ilerleyeceğiz. Söz konusu ekleme işlemi sonrasında Web uygulamamıza dahil edeceğimiz **WCF DataService OCT 2011 CTP** öğesinin kod içeriğini de aşağıdaki gibi geliştirebiliriz.

```
using System.Collections.Generic;
using System.Data.Services;
using System.Data.Services.Common;
using System.Xml;
using Microsoft.Data.Edm;
using Microsoft.Data.Edm.Csdl;
using Microsoft.Data.Edm.Validation;

namespace AdventureWorksDataServices
{
   public class ProductDataService
: DataService<AdventureWorksEntities>
   {
      public static void InitializeService(DataServiceConfiguration config)
      {
         config.SetEntitySetAccessRule("Products", EntitySetRights.All);
         config.DataServiceBehavior.MaxProtocolVersion =
DataServiceProtocolVersion.V3;

         config.AnnotationsBuilder = (model) =>
            {
               IEdmModel edmModel;
               IEnumerable<EdmError> errors;

               XmlReader[] readers =new XmlReader[] {
                  XmlReader.Create("f:\\ProductsAnnotations.xml")
               };
```

```
                bool parsed = CsdlReader.TryParse(readers, out edmModel, out errors,
model);
                return parsed ? new IEdmModel[] { edmModel } : null;
            };
        }
    }
}
```

Piuvvvv Burada neler oldu böyle? Daha önceden yaptığımız **Ado.Net Data Service** geliştirmelerine hiç ama hiç benzemiyor gibi. Daha önceden sadece ilgili **Entity** tiplerini belirli erişim kuralları çerçevesinde*(örneğin sadece okuma amaçı açılsınlar)* dış dünyaya açan bir **InitializeService** metodu bulunurdu aslında Aslında olayı kısaca açıklayayım.

DataServiceConfiguration tipine eklenen yeni **AnnotationsBuilder** özelliği aslında

public Func<Microsoft.Data.Edm.IEdmModel, IEnumerable<Microsoft.Data.Edm.IEdmModel>> AnnotationsBuilder

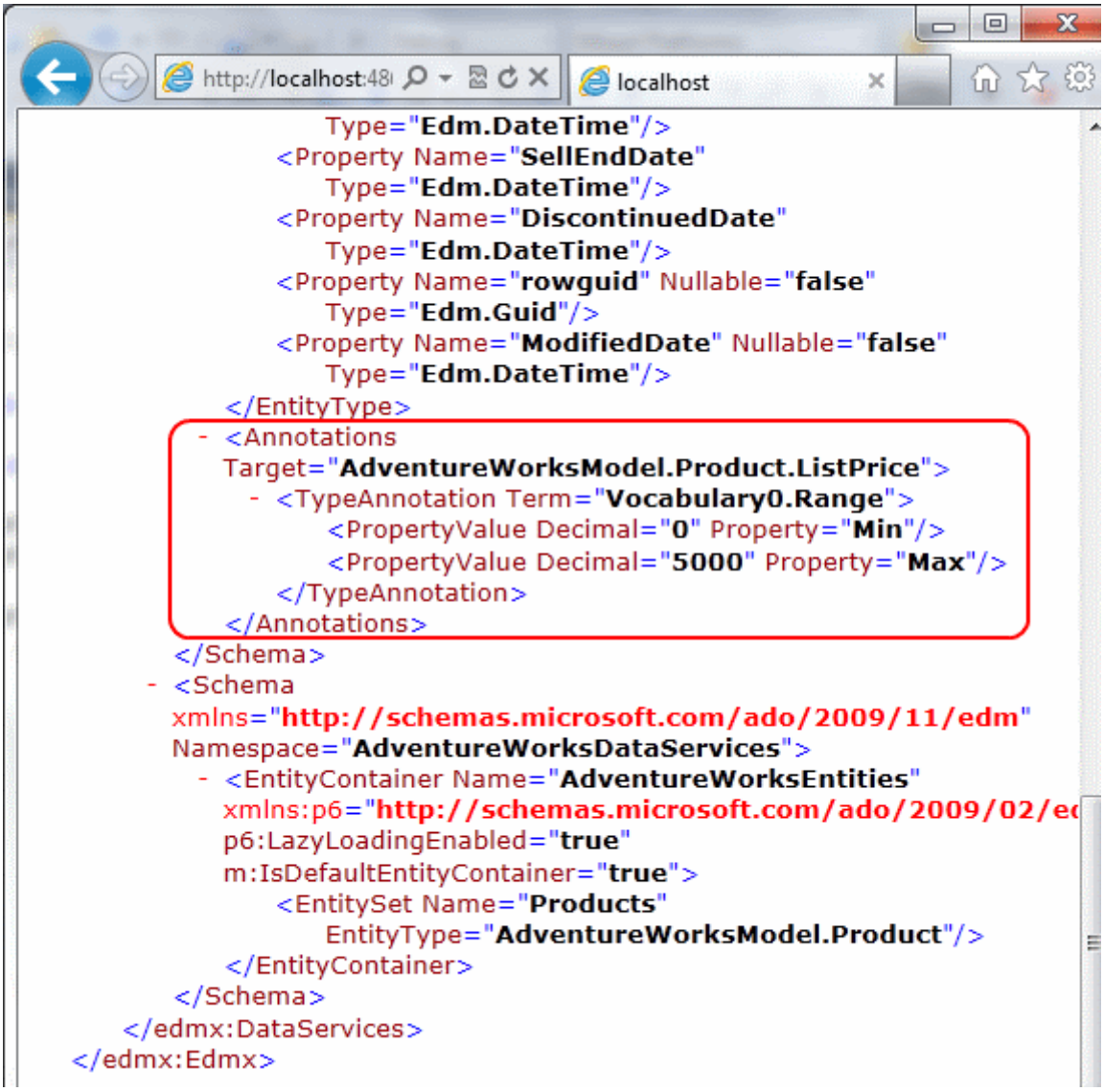
ile işaret edilen bir metodu göstermektedir. Bu metod içerisinde biz, **Entity Data Model'** in **CSDL(Conceptual Schema Definition Language)** içeriğine müdahale ederek örnek bazı değişikliklerde bulunuyoruz. Bu değişiklikler koda göre XmlReader yardımıyla okunan **ProductsAnnotations.xml** isimli dosyada yer almaktadır. Dikkat edecek olursak eğer **CsdlReader** tipinin**TryParse** metodu bir den fazla **XmlReader** referansı alabilmektedir. Bir başka deyişle bir **CSDL** içeriğine **n adet Annotation** kuralının enjekte edilmesi de mümkündür. örneğimizde kullandığımız dosya içeriği ise aşağıdaki gibidir.

```
<Schema Namespace="AdventureWorksModel" Alias="AdventureWorksModel"
xmlns="http://schemas.microsoft.com/ado/2009/11/edm">
  <Using
NamespaceUri="http://vocabularies.odata.org/Validation" Alias="Validation"/>
  <Annotations Target="AdventureWorksModel.Product.ListPrice">
    <TypeAnnotation Term="Validation.Range">
      <PropertyValue Property="Min" Decimal="0" />
      <PropertyValue Property="Max" Decimal="5000" />
    </TypeAnnotation>
  </Annotations>
</Schema>
```

Burada son **CTP** ile gelen önemli bir nokta **Using** elementi ve içerisindeki **NamespaceUri** niteliğinin değeridir. Bu detaylar bir kenara dursun aslında bu dosyanın ne söylediğini anlamak şu anda bizim için çok daha önemlidir

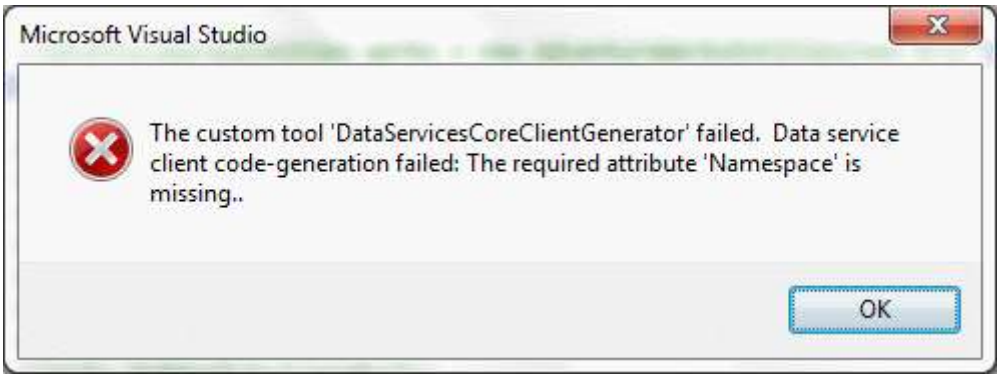
Dikkat edileceği üzere **Annotations** elementine ait **Target** niteliğinde bir tanımlama yapılmıştır. Buna göre **AdventureWorksModel** şemasında yer alan **Product Entity tipinin ListPrice**özelliği için **Validation.Range** formatında bir **TypeAnnotation** bildirimi yapılmaktadır. Bir başka deyişle **ListPrice** özelliği için çalışma zamanında üretilen **metadata** içeriğine enjekte edilecek bir doğrulama kalıbı sunulmaktadır.

Bu kalıba göre **ListPrice** özelliğinin **Decimal** veri tipinden olan **minimum** ve **maximum** alan değerleri belirtilmektedir. Senaryomuza göre **ListPrice** özelliği **0 ile 5000 birim arasında** olmalıdır. İşte kod tarafında kullanılan **TryParse** metodu eğer **XML** dosyasını okuyabilme işlemi başarılı olursa, ilgili validasyonu çalışma zamanındaki metadata içeriğine dahil edecektir. Eğer **Data Service** örneği herhangibir tarayıcıda açılır ve **metadata** içeriğine ulaşılırsa aşağıdaki çıktı ile karşılaşıldığı görülecektir. *(Benim örneğimde söz konusu Data Service adresi http://localhost:4860/ProductDataService.svc/$metadata şeklindedir)*

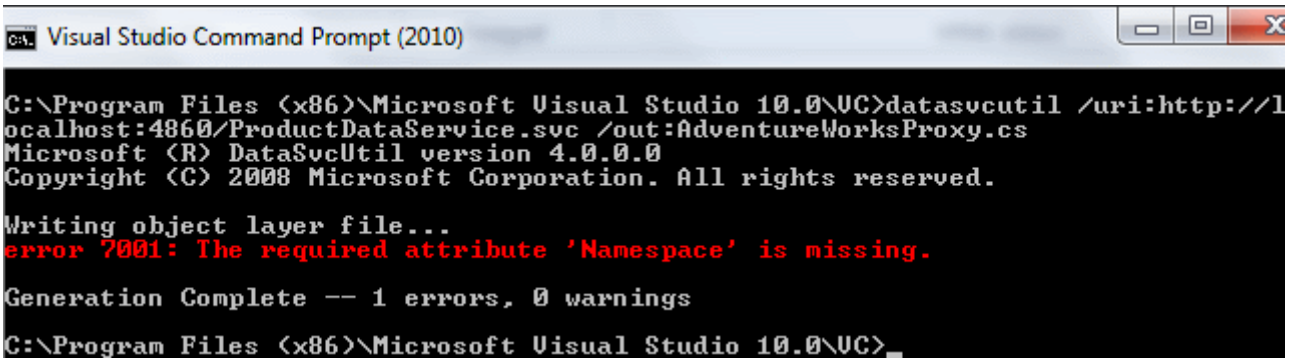


Bakın burası çok önemli. Bir **WCF Data Service'** in çalışma zamanındaki **metadata** çıktısına bir doğrulama kuralı enjekte edilmektedir. *(Bu tip bir çalışma zamanı alt yapısını sıfırdan nasıl yazabileceğinizi, hangi tasarım kalıplarına veya yazılım prensiplerine ihtiyacınız olacağını bir düşünün)*

Peki ama bu ne işimize yarayacak? Bunun için öncelikli olarak basit bir Client uygulaması geliştirmeyi düşünebiliriz. **Console Application** şeklinde tasarlayabileceğimiz bu uygulama çok basit olarak ilgili **WCF Data Service'** i referans etmelidir. Sonrasında test amacıyla herhangibir **Product** nesne örneğinin **ListPrice** özelliğinin değeri **0-5000** aralığı dışında **set** edilerek nasıl bir sonuç elde edildiğine bakılabilir. Ben bu amaçla istemci tarafına ilgili servis referansını eklemeye çalışarak işe başladım. İşe başladım diyorum çünkü **Visual Studio 2010'** a ait **Add Service Reference** ekleme aracı bir **Namespace**' i bulamadığı için söz konusu **proxy** tipini üretmeyi başaramadı



Ben de bunun üstüne söz konusu servis referansını komut satırından **DataSvcUtil** aracı ile üretmeyi denedim. İşte sonuç



Hal böyle olunca tabi heyecanlı bir şekilde makaleyi yazmaya çalışan bir yazılım sevdalısının nasıl da hüsrana uğradığını zannediyorum ki anlayabilirsiniz Eğer **Proxy**üretimi başarılı bir şekilde gerçekleşseydi bu durumda aşağıdaki ilüsturasyon da yer alan bir kod parçasını denemeye tabi tutacaktık.

```
using System;
using System.Linq;
using ConsoleApp.AdventureWorks;
```

```
namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         AdventureWorksEntities works = new AdventureWorksEntities(new
Uri("http://localhost:4860/ProductDataService.svc"));

         var product = (from p in works.Products
                        where p.ProductID == 1
                        select p).FirstOrDefault();
         if (product != null)
         {
            product.ListPrice = 5001M;
         }
         works.UpdateObject(product);

         var response = works.SaveChanges();
      }
   }
}
```

ve özellikle örnek bir **Product** nesne örneğinin **ListPrice** özelliğinin değerini, **Annotations'** da belirttiğimiz aralığın dışına çıkartmaya çalışacak ve **SaveChanges** çağrısı sonucu servis tarafından gelen **Response'** a bakacaktık. Bu noktada tahmin edeceğiniz üzere beklentimiz validasyon ihlali nedeniyle **Update** işleminin yapılmaması ve buna uygun bir Response kodunun istemci tarafında döndürülmesi olacaktı. En azından benim beklentim bu yöndeydi.

Yine de buraya kadar yazdıklarımız ile aslında **WCF Data Service** tarafındaki **Annotations** bildirimlerinin, yeni **CTP** sürümü ile gelen **Vocabularies namespace'** i ile ilişkilendirildiğini düşünebiliriz. Amaç ise **Entity, Property, Navigation Property** gibi bazı yapılarda doğrulama kurallarını uygulatabilmektir. Bu konu ile ilişkili olarak ben dehttp://blogs.msdn.com/b/astoriateam/archive/2011/10/13/vocabularies-in-wcf-data-services.aspx adresindeki girdiye ait yorumları takip etmeye çalışıyor olacağım. Malum istemci tarafı için gerekli bir testi gerçekleştiremedik/gerçekleştiremedim. Dolayısıyla şimdilik yazımızı sonlandırmak durumundayız. Buruk bir son oldu ama ne yapalım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

EF 4.2 ile Code-First Development (2011-10-13T11:45:00)

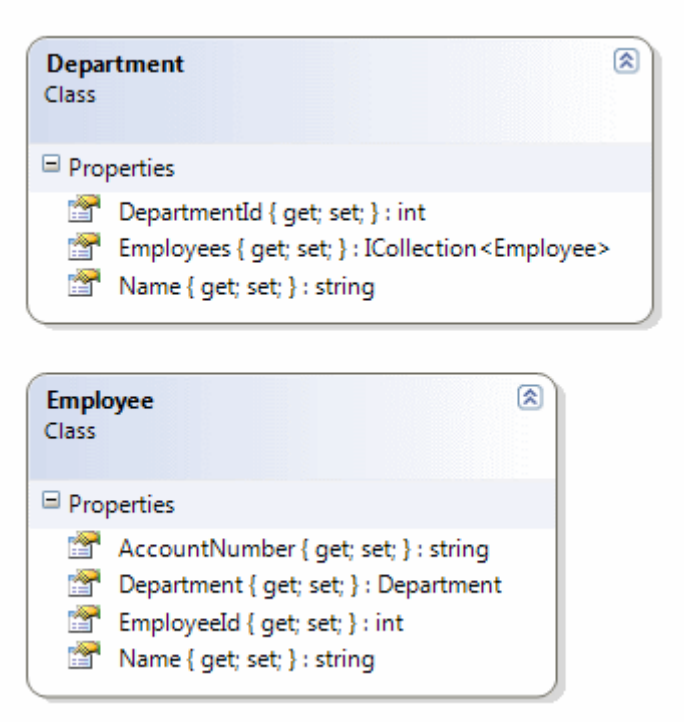
entity framework 4.2,code first development,entity framework,

Merhaba Arkadaşlar,

Teknolojinin hızına inanın ki yetişilemiyor. Her gün yeni bir bilim ve teknoloji haberi var dünyada. Ancak bir geçek daha var ki o da **Microsoft, Google, Apple** gibi devlerin de hızına yetişilememsi. örneğin daha henüz **2011 Mix'** te **RTM** olarak duyurulan **Entity Framework 4.1** sürümü üzerine geçtiğimiz günlerde **4.2 RC(Release Candidate)** duyuruldu. Ben de bunun üzerine yeni sürümde **Code-First Development'** ın nasıl uygulandığını anlamak ve görmek istedim. Haydi gelin keşfetmeye başlayalım.

İlk olarak **Entity Framework 4.2** sürümü ile ilişkili detaylı bilgilere Ado.Net Team Blog' un ilgili adresi üzerinden ulaşabileceğinizi belirtmek isterim. Peki herşey güzel de nedir bu **Code-First Development? Ado.Net Entity Framework** tarafında en başından beri var olduğunu bildiğimiz **Model-First** ve **Database-First Development** yaklaşımları mevcut. Ancak **Code-First development** ne ola ki Doğruyu söylemek gerekirse bu tanımı en sonda yapmanın ve aradan geçen zaman dilimi içerisinde basit bir örnek üzerinden adım adım ilerlemenin daha doğru olacağı kanısındayım.

Bu amaçla ilk olarak basit bir **Console Application** projesi açtığımızı ve içerisine aşağıdaki sınıf diagramında görülen tipleri eklediğimizi var sayalım.



Department.cs

```
using System.Collections.Generic;
```

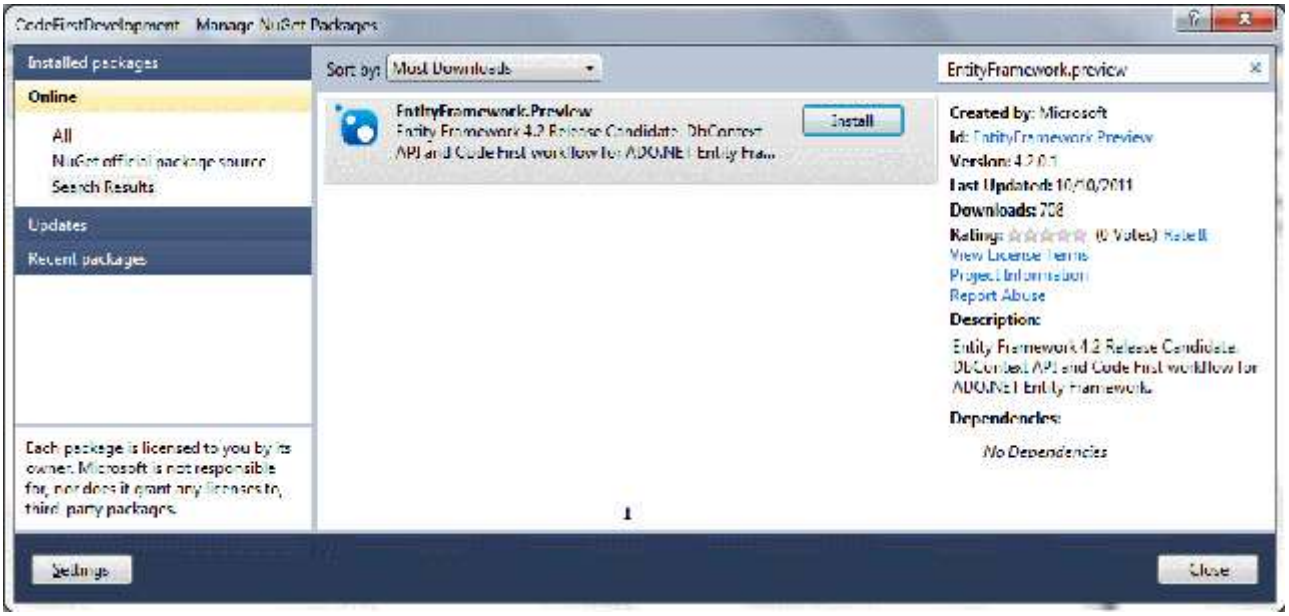
```
namespace CodeFirstDevelopment
{
   public class Department
   {
      public int DepartmentId { get; set; }
      public string Name { get; set; }
      public virtual ICollection<Employee> Employees { get; set; }
   }
}
```

Employee.sc

```
namespace CodeFirstDevelopment
{
   public class Employee
   {
      public int EmployeeId { get; set; }
      public string Name { get; set; }
      public string AccountNumber { get; set; }
      public virtual Department Department { get; set; }
   }
}
```

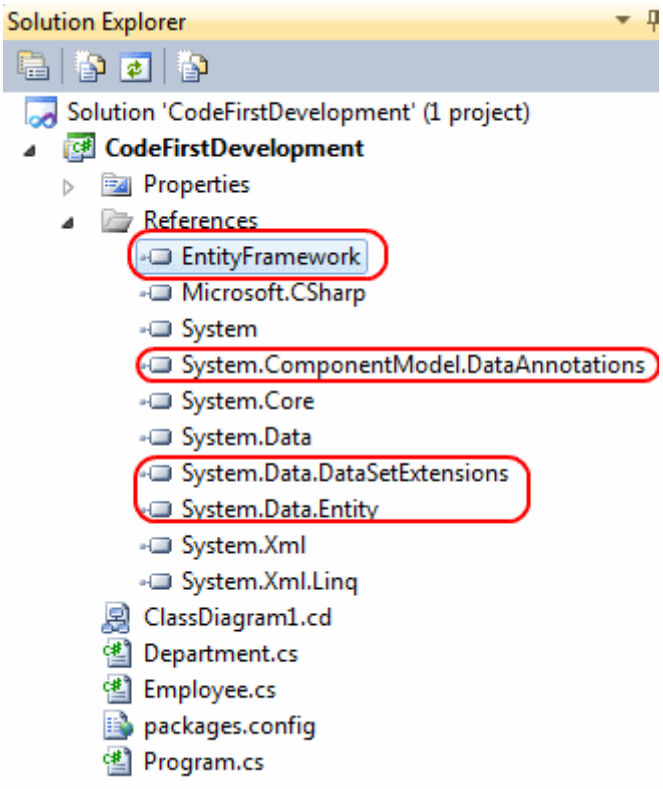
Aslında bir departman ve bu departmana bağlı işçileri temsil etmeye çalıştığımız basit iki **POCO(Plain Old CLR Object)** tipi yazdığımızı görmekteyiz. Bir başka deyişle çok basit tipleri kullanarak aralarında ilişkiler tanımladığımız basit bir **Model'** i kurgulamaktayız. Peki bu iki tipe ait nesne koleksiyonlarını nasıl kullanacağız? Normal şartlarda**Ado.Net Entity Framework** içerisinde, basit tipleri sarmallayan ve onlara ait koleksiyonları birer özellik gibi sunan, ve ayrıca yükleme, ekleme, silme gibi temel operasyonları üstlenen bir tipin daha olduğunu biliyoruz.

Code-First Development senaryosunda bu tipin yazılması da geliştiriciye ait. Ancak bu tip sanıldığı kadar karmaşık değil. Tek dikkat edilmesi gereken nokta var olan **EF** kabiliyetlerini sunabilmesi için **.Net'** in aşina olduğu bir tipten türemesi gerekliliği. İşte bu noktada **NuGet** ile uygulamamıza **install** edebileceğimiz bir kütüphane meydana çıkıyor.**EntityFramework.Preview** 



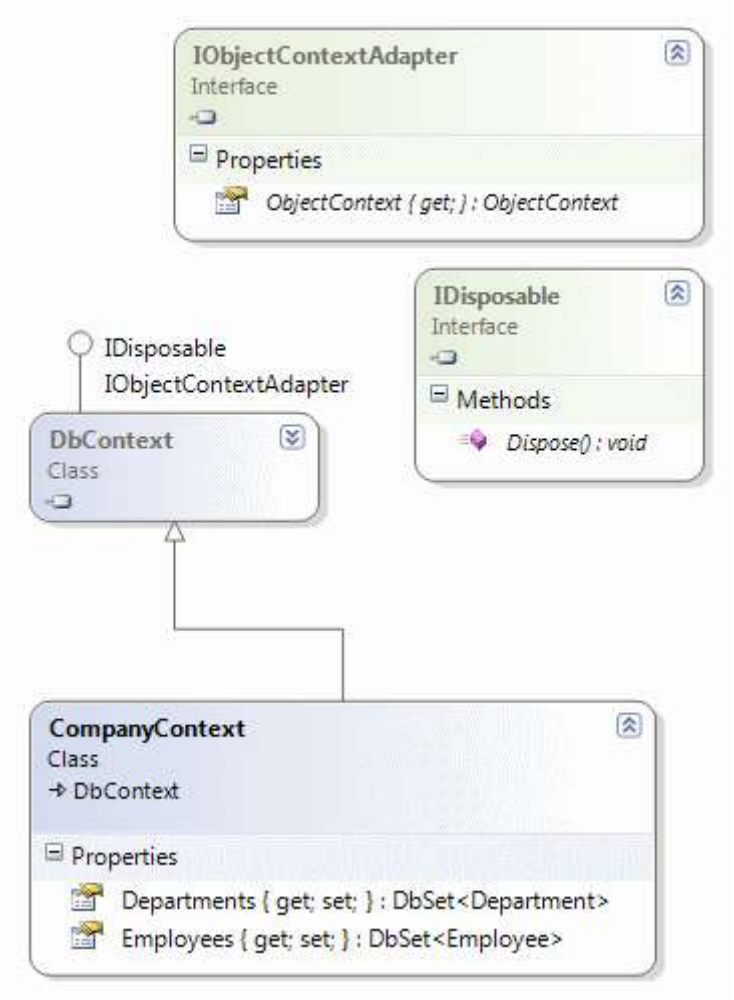
Teşekkürler NuGet

Güncel proje üzerinden kısa sürede gerçekleştirilen **install** işlemi sonrasında bazı ek library dosyalarının referans edildiğini görürüz. Aşağıdaki şekilde bu kütüphaneler işaretlenmiştir.



Şimdi projemize yeni bir sınıf daha ekliyor olacağız. Aslında **Context** diye tabir edilen bu sınıfın görevi temel olarak **Model** içerisindeki **POCO** tiplerine ait koleksiyon bazlı nesneleri sunmak olacaktır. Tabi **EF** kabiliyetlerinin kendisine kazandırılması

gerektiğinden indirilen paket içerisinde yer alan bir tipten de **türetilmesi(Inheritance)** söz konusudur. İşte **Context** tipimiz;



ve kod içeriğimiz;

```
using System.Data.Entity;

namespace CodeFirstDevelopment
{
   public class CompanyContext
      :DbContext
   {
      public DbSet<Department> Departments { get; set; }
      public DbSet<Employee> Employees { get; set; }
   }
}
```

Dikkat edileceği üzere **CompanyContext** tipi **Department** ve **Employee** tiplerine ait nesnel koleksiyonları sunmak için **DbSet<T>** tipinden olan özellikler sunmaktadır. Buna ek olarak **DbContext** tipinin

de**IObjectContextAdapter** ve **IDisposable** arayüzlerini(Interface) uyguladığını görmekteyiz. Bu **interface** yardımıyla aslında **Object Context'** e erişilmektedir. Diğer yandan ilgili nesnesinin**Dispose** edilebilir olması için **IDisposable** arayüzünün de uygulandığı görülmektedir. Eğer **DbContext** tipinin üyelerine bakılacak olursa **SaveChanges** gibi bir metod olduğunu fark edebiliriz ki bunun EF tarafında ne kadar önemli bir metod olduğunu tahmin edebilirsiniz Aslında **DbContext**, **Entity** verileri ile birer nesne gibi çalışılabilmesini ve sorgulama işlemlerinin yapılabilmesini sağlamaktadır. Bu yüzden Entity nesneleri üzerinde yapılan veri değişimlerinin bir kaynağa doğru gönderilebilmesi için **SaveChanges** gibi metod sunmaktadır.

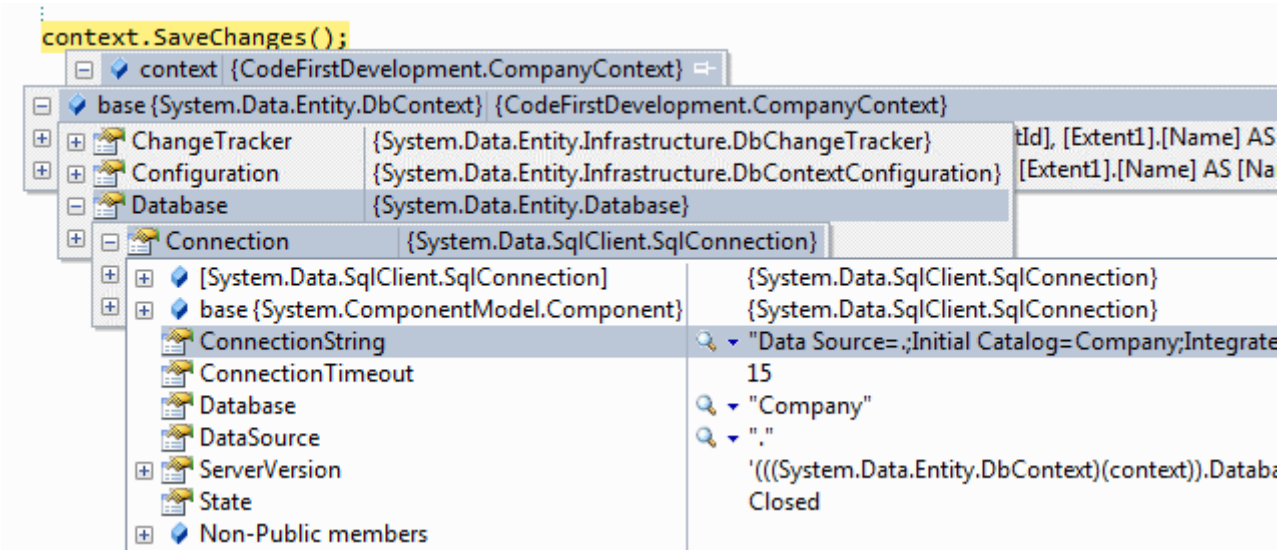
Artık Console uygulamamızda söz konusu tipleri kullanarak bir test gerçekleştirebiliriz. Bu amaçla aşağıdaki örnek kod parçasını göz önüne alalım.

```
using System.Collections.Generic;

namespace CodeFirstDevelopment
{
   class Program
   {
      static void Main(string[] args)
      {
         using (CompanyContext context = new CompanyContext())
         {
            Department IT = new Department { DepartmentId = 1, Name = "IT" };

            List<Employee> employees = new List<Employee>
            {
               new Employee{ EmployeeId=1000, Name="Burak Senyurt", Department=IT,
AccountNumber="IT10235"},
               new Employee{ EmployeeId=1001, Name="Steve Jobs", Department=IT,
AccountNumber="RIP1234"},
               new Employee{ EmployeeId=1250, Name="Bill Gates", Department=IT,
AccountNumber="BIL1222"}
            };
            IT.Employees = employees;

            context.Departments.Add(IT);

            context.SaveChanges();
         }

      }
   }
}
```

Dikkat edileceği üzere **Context** nesnesi örneklendikten sonra önce bir **Department** üretilmiş ardından bu departmana bağlı bir kaç örnek **Employee** eklenmiştir. Yani bir departman ve bu departmana bağlı veri içerikleri bellek üzerinde gerçekleştirilmiştir. Son olarakta **Context** nesne örneği üzerinden **SaveChanges** metodu çağırılmıştır **SaveChanges**? Hımmm…Şimdi burada bir saniye durup düşünelim. **SaveChanges** metodu bellek üzerinde gerçekleşen ekleme, silme, güncelleme vb işlemler sonucu ilgili varlık içeriklerini nereye kayıt edecektir. Sanki bir yerlerde bir şeyleri söylememiz gerekiyordu. Aslında söylemesekte olur Nitekim **DbContext** nedeniyle ilgili şema yapıları için standart bir SQL bağlantısı söz konusudur.**SQLExpress** sürümü kullanılaraktan bir veritabanı otomatik olarak oluşturulacak ve söz konusu tablolar üretilerek ilgili veri içerikleri buraya yüklenecektir. Ancak biz bu davranışı değiştirmek istediğimizi düşünelim. Söz gelimi **SQL Server** üzerinde bir veritabanına doğru yazılsın. Bu amaçla tek yapmamız gereken uygulamanın **App.Config** dosyasına aşağıdaki **Connection String** bilgisini eklemektir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="CompanyContext" connectionString="data
source=.;database=Company;integrated security=SSPI"
providerName="System.Data.SqlClient"/>
  </connectionStrings>
</configuration>
```

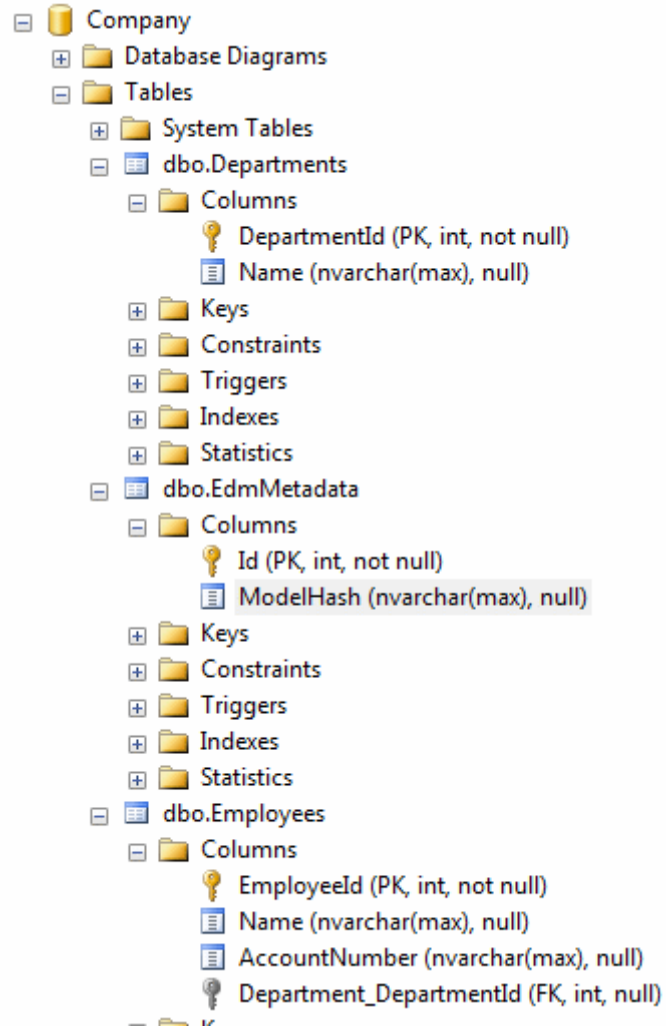
Buraya dikkat! Biz şu ana kadar **Company** isimli bir veritabanı(database) oluşturmadık Dolayısıyla ilk çalışma esnasında söz konusu veritabanı ve içeriği üretilse çok iyi olur değil mi? Haydi gelin uygulamamızı **debug** modda çalıştıralım. Eğer çalışma zamanında **Context** nesne örneği üzerinde biraz dolaşırsak aşağıdaki gibi bir**Connection String** bilgisinin uygulanmaya çalışıldığını görürüz.



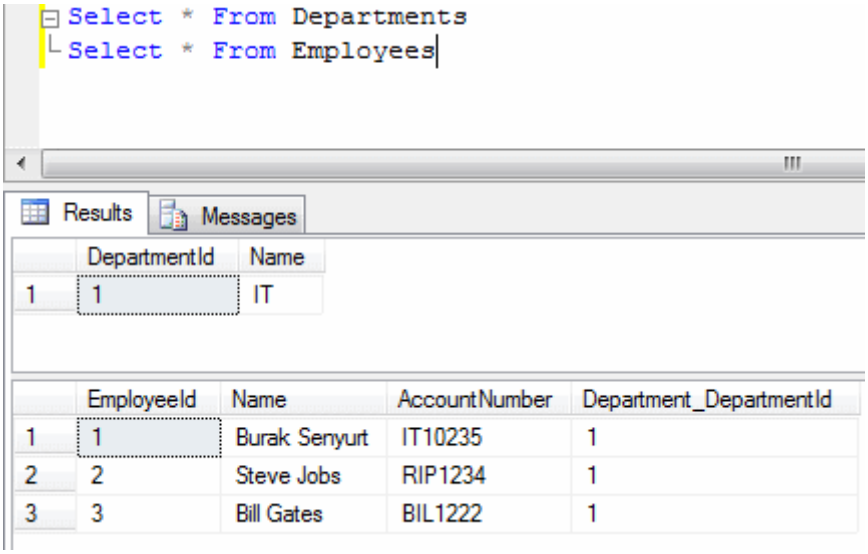
Burada ,

Data Source=.\SQLEXPRESS;Initial Catalog=CodeFirstDevelopment.CompanyContext;Integrated Security=True;MultipleActiveResultSets=True;Application Name=EntityFrameworkMUE

şeklinde bir bağlantı bilgisi üretilmiştir. **SaveChanges** metodunu atladığımızda ise ilk yapmamız gereken aslında yerel sunucuya bakmak olmalıdır. Ve sonuç ;



Volaaa!!! Sizce de süper değil mi? çok basit tipler tanımladık ve code modelimize bakılarak ilgili tablolar ve aralarındaki ilişkiler otomatik olarak oluşturuldu. Hatta verilerin yüklendiğini de SQL sorgularımızı atarak görebiliriz.



Bu durumda geldiğimiz aşamaya baktığımızda, basit **POCO** nesnelerini ve **DbContext** tipini kullanaraktan veritabanı modelimizi, önce kodu düşünerek geliştirmiş olduğumuzu ifade edebiliriz. Elbette yapılabilecek daha pek çok işlem var. **Code-First Development** ile ilişkili diğer özellikleri ben de zaman içerisinde incelemeyi düşünüyorum. Şimdilik bu giriş niteliğindeki Hello World yazısı ile idare etmeye çalışacağım. Eğer **Entity Framework** konusunda geçerli ve güncel bir kaynaktan yararlanmak isterseniz **Julie Lerman'** ın Programming Entity Framework: Building Data Centric Apps with the ADO.NET Entity Framework adlı kitabını tavsiye edebilirim. Her ne kadar **2010 Ağustos** baskısı olsada kalan kısımları **Offical Ado.Net Team Blog** ile eksik kısımları ve yenilikleri tamamlayabilirsiniz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

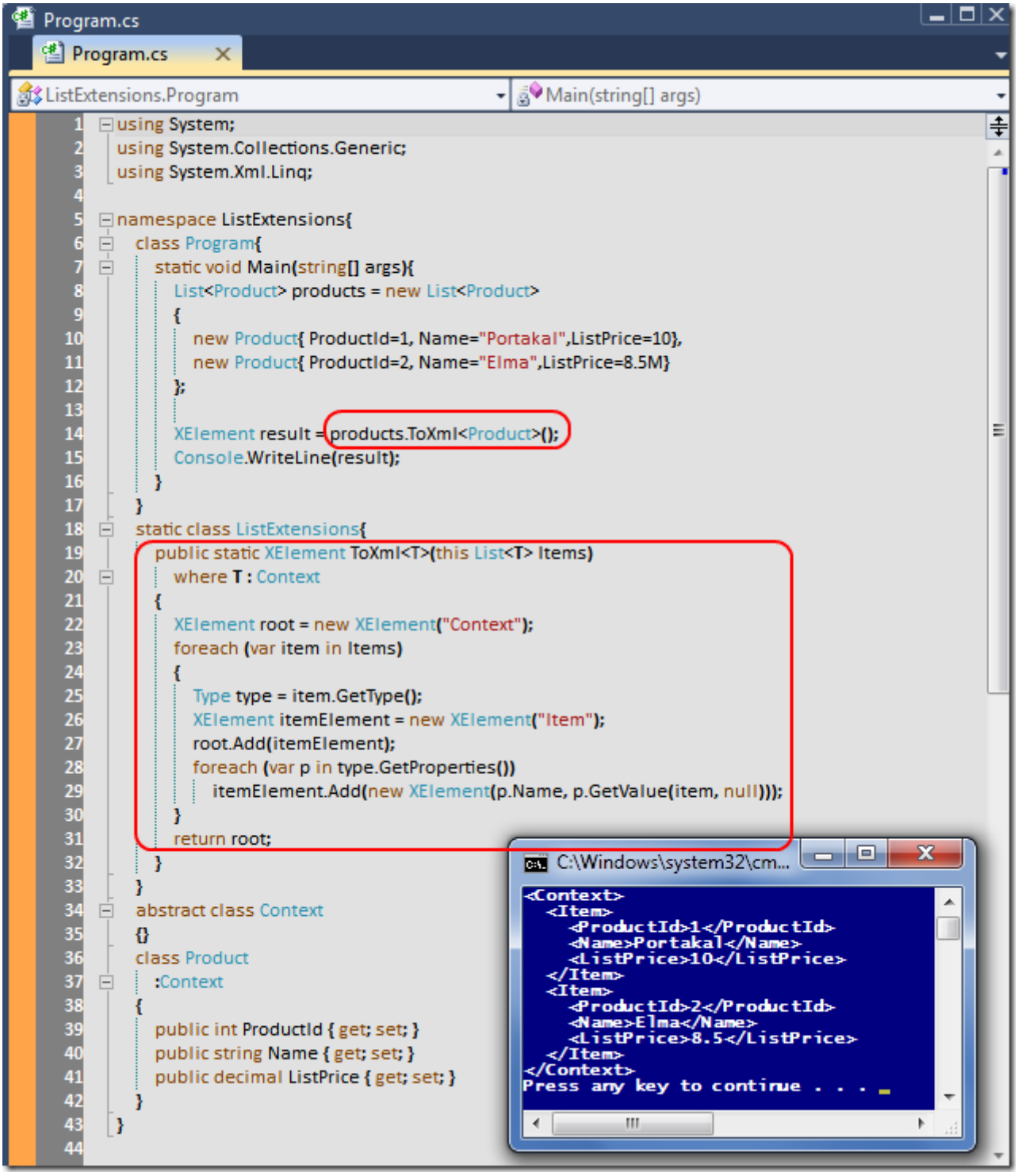
CodeFirstDevelopment.rar (1,01 mb)

Tek Fotoluk İpucu-33(Xml Cast) (2011-10-10T16:00:00)

c#,c# temelleri,xlinq,xelement,list,generic constraints,extension methods,reflection,

Merhaba Arkadaşlar,

Varsayalım ki elimizde kendi geliştirdiğimiz tipler ve kullandığımız List koleksiyonları var. Ve olduda bir yerde bu koleksiyonların içeriklerinin XML çıktılarına ihtiyaç duyduk. Basit bir Extension method geliştirebilir miyiz acaba?



ListExtensions.rar (23,69 kb)

Birlikte Geliştirdik (2011-10-03T11:48:00)

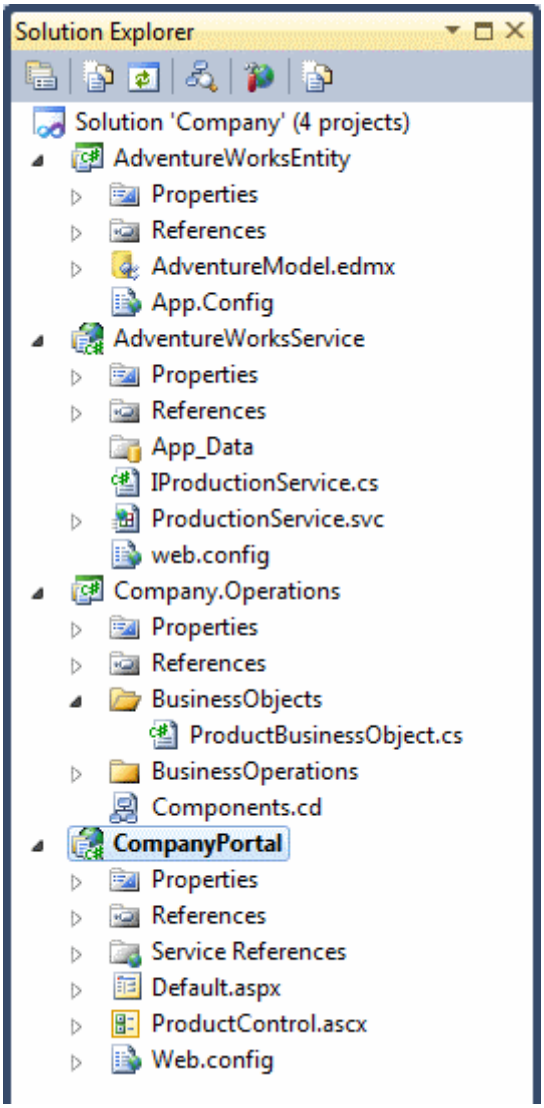
web user control,dynamic control loading,entity framework,linq,wcf,service,entity model,

Merhaba Arkadaşlar,

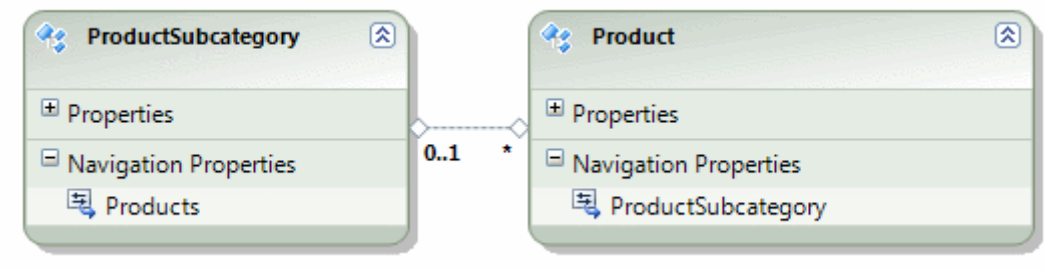
Eğer benim gibi yaz kış kolayca grip oluyorsanız eminim ki bol limonlu çorbalara aşinasınızdır. Hele ki şanslıysanız ve eşinizin ya da annenizin yanındaysanız şöyle evde var olan tüm sebzelerden oluşan karma bir sebze çorbası süper rahatlatıcı olacaktır. E bazen .Net Framework tarafında da eldeki materyalleri bir araya getirip güzel bir çorba yapmak gerekir Güzel bir çorba hazırlamaya ne dersiniz?

Hazırlayacağımız çorbamızda çok kıymetli yardımcılarımız da var. Son katılımcılarımız ile gerçekleştirmekte olduğumuz **Asp.Net**eğitiminden çok güzel fikirler ve örnekler çıkmaya devam ediyor. Geliştireceğimiz örnek **Solution** içerisinde **Entity Framework, WCF Service, LINQ, Asp.Net Web Application, Web User Control, LINQ** gibi pek çok kavram yer almakta. Temel olarak başlangıçtaki senaryomuz ise şu : *"Asp.Net web uygulamamızda yer alan bir Web User Control' ümüz, AdventureWorks veritabanında yer alan herhangibir Product' satırına ait bazı alan bilgilerini gösterecek"*

Pekala bu iş için hemen bir **Web User Control** geliştirebilir ve **Load** metodunda ilgili veri çekme işlemlerini gerçekleştirerek basitçe sonuca gitmeyi düşünebilirsiniz. Ama biz bu şekilde button arkası programlama yapmamayı tercih ediyoruz Bunun yerine aşağıdaki grafikte yer alan Solution içeriğini üreterek ilerleyeceğiz. *(Dilerseniz yazının son satırından Solution' ı indirip inceleyin)*

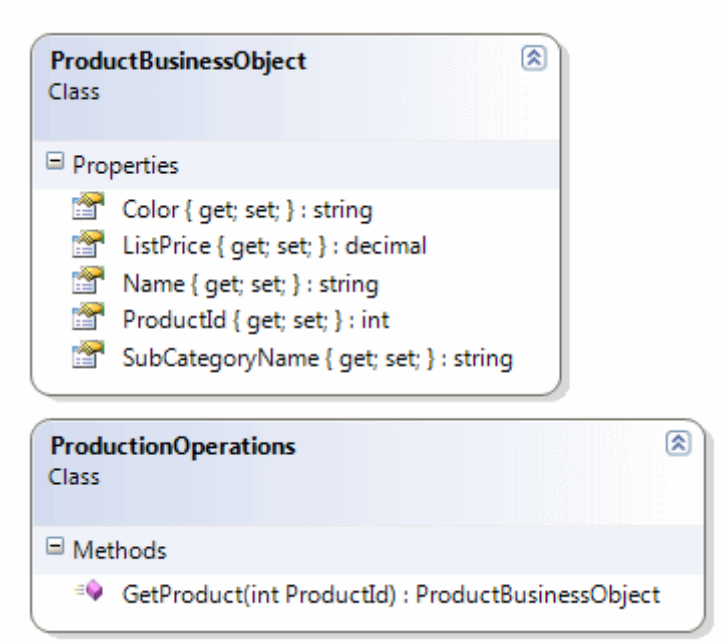


İlk olarak **AdventureWorks** veritabanındaki bazı tabloları içerecek olan **Entity Model'** imizi tasarlayarak işe başlayalım. Söz konusu **Class Library** projesi içerisinde kullanacağımız**Entity Model** diagramının içeriğini ise başlangıçta aşağıdaki gibi oluşturabiliriz.



Şimdilik **Product** ve **ProductSubCategory** tablolarının karşılığı olan Entity içeriklerinin yer aldığı bir diagram ile karşı karşıyayız. Diğer yandan **Web User Control'**ümüzü geliştirene kadar kat etmemiz gereken daha çok yolumuz olacak. öncelikle bir **Product** içeriğini dış ortama verecek olan iş fonksiyonelliklerini içeren bir katman daha

yazacağız. Söz konusu katman tahmin edeceğiniz üzere bir **Class Library** olacak. Bu katmanda ekstradan bir **Business Object** daha kullanıyor olacağız. *çünkü **Product** tipinin tüm içeriğini dış ortama sunmak gibi bir niyetimiz olmadığını düşünmekteyiz.* Bu sebepten **Product** tipi yerine geçecek olan bir **Surrogate Type** kullanımını ele alacağız. Dolayısıyla**Company.Operations** isimli **Class Library** içeriğini aşağıdaki sınıf çizelgesinde olduğu gibi tasarlayarak devam ediyoruz.



Burada görülen **ProductBusinessObject** aslında dış ortama sunulacak olan **Product** içeriğini taşıyan bir **POCO** nesnesidir. Aslında bu tip **Business Object** türlerini başka bir katman içerisinde tutmayı da düşünebiliriz

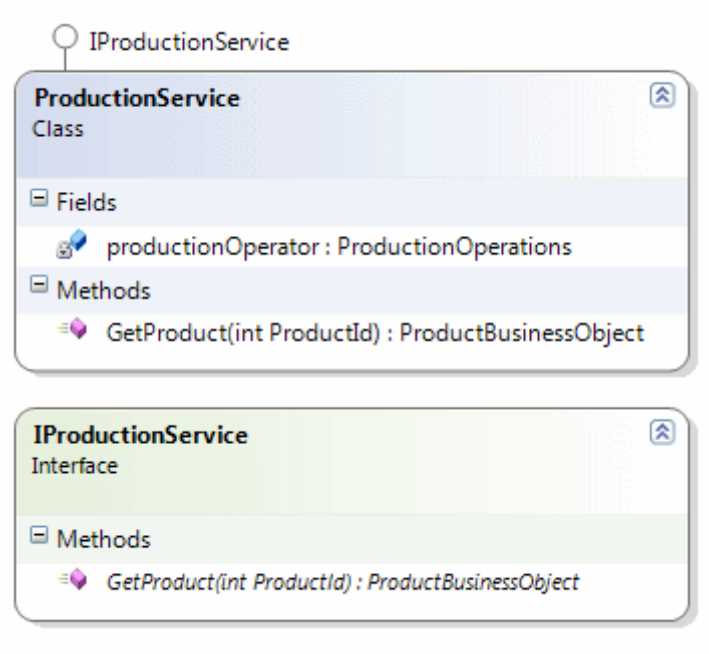
```
namespace Company.Operations
{
   public class ProductBusinessObject
   {
      public int ProductId { get; set; }
      public string Name { get; set; }
      public decimal ListPrice { get; set; }
      public string SubCategoryName { get; set; }
      public string Color { get; set; }
   }
}
```

ProductionOperations isim sınıfımız ise şu an için tek bir operasyon sunmaktadır ve **ProductId** değerine göre **ProductBusinessObject** tipinden bir referans döndürmek üzere tasarlanmıştır.

```
using System.Linq;
using AdventureWorksEntity;

namespace Company.Operations
{
   public class ProductionOperations
   {
      public ProductBusinessObject GetProduct(int ProductId)
      {
         ProductBusinessObject result = null;

         using (AdventureWorksEntities context = new AdventureWorksEntities())
         {
            result = (from p in context.Products
                      where p.ProductID == ProductId
                      select new ProductBusinessObject
                      {
                          ProductId=p.ProductID,
                           Name=p.Name,
                            ListPrice=p.ListPrice,
                             Color=p.Color,
                              SubCategoryName=p.ProductSubcategory.Name
                      }).FirstOrDefault<ProductBusinessObject>();
         }

         return result;
      }
   }
}
```

örneğimizi **EF 4.0** versiyonu ile geliştirdiğimiz için bir **Lazy Loading** durumunun da söz konusu olduğunu ifade edelim. **SubCategoryName** özelliğine değer atanan satıra dikkat edin 

Artık söz konusu iş operasyonunu dış dünyaya sunacak bir uygulamayı da geliştirebiliriz. Bu uygulama favorimiz olan **WCF** tabanlı bir servis olarak düşünülmektedir.
Böylece **Entity**modelimiz, **Business Object'** lerimiz ve **Business Operasyonlarımızın** tamamı söz konusu servis uygulamasının arkasında kalacaktır. Yani asıl End User' lar bu işlevselliklere erişebilmek için servis katmanı üzerinden geçmek zorunda kalacaklardır. WCF Service uygulamamızın içeriği aşağıdaki diagramda görüldüğü gibidir.



IProductionService isimli **servis sözleşmemiz(Service Contract)** şu şekildedir;

```
using System.ServiceModel;
using Company.Operations;

namespace AdventureWorksService
{
   [ServiceContract]
   public interface IProductionService
   {
      [OperationContract]
      ProductBusinessObject GetProduct(int ProductId);
   }
}
```

Sözleşmemiz senaryomuza göre bir ürünün elde edilmesi
ve **ProductBusinessObject** tipinden geriye döndürülmesi işlemini üstlenen basit bir operasyon sunmaktadır. Bu operasyonun uyarlaması ise **ProductionService** sınıfı içerisinde gerçekleştirilmektedir.

```
using Company.Operations;

namespace AdventureWorksService
{
   public class ProductionService
      : IProductionService
   {
      ProductionOperations productionOperator = new ProductionOperations();
```

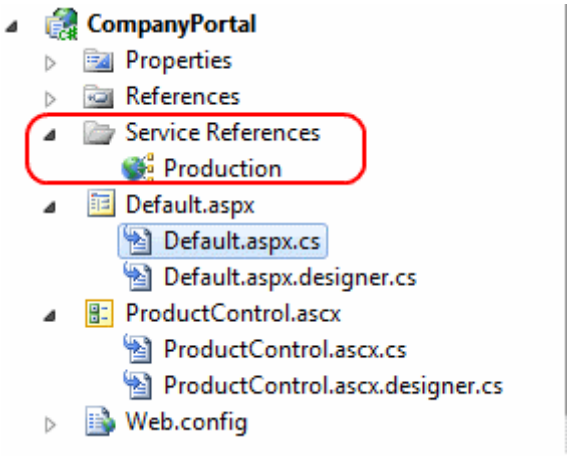
```
        public ProductBusinessObject GetProduct(int ProductId)
        {
            return productionOperator.GetProduct(ProductId);
        }
    }
}
```

Servis uygulamamızda dikkat edilmesi gereken noktalardan birisi de, **AdventureWorks** entity modelini kullanabilmesi için gerekli **Connection String** bilgisine sahip olma zorunluluğudur. Bildiğiniz üzere Entity Modelimizi bir **Class Library** içerisinde tutumaktayız ve çalışma zamanında ilgili mapping işlemlerini üstlenecek olan yürütücü uygulamanın söz konusu mapping için bir bağlantı bilgisine sahip olması gerekiyor. Dolayısıyla **AdventureWorksEntity** kütüphanesinde otomatik olarak üretilen **connection string** bilgisinin servis uygulamasındaki **web.config** dosyasında yer alması şart.

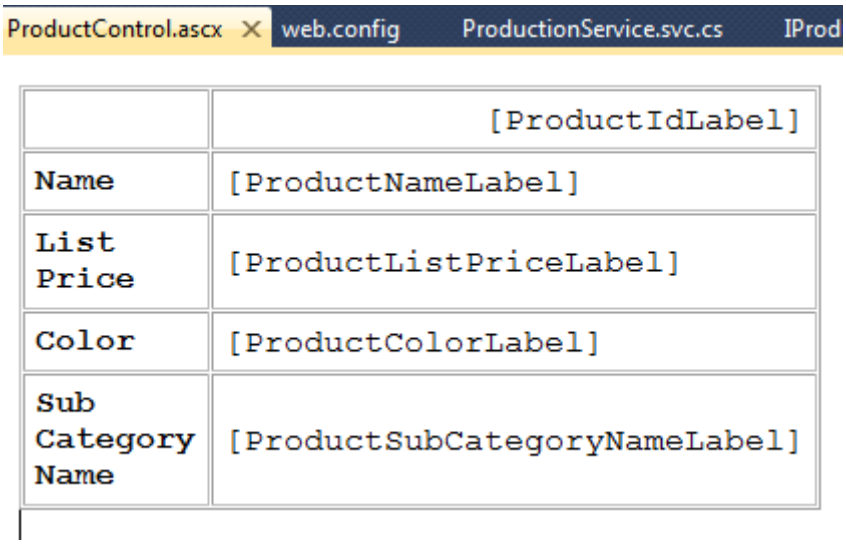
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="AdventureWorksEntities"
connectionString="metadata=res://*/AdventureModel.csdl|res:
//*/AdventureModel.ssdl|res://*/AdventureModel.msl;
provider=System.Data.SqlClient;provider connection string=&quot;Data
Source=.;Initial Catalog=AdventureWorks;Integrated
Security=True;MultipleActiveResultSets=True&quot;"
providerName="System.Data.EntityClient" />
  </connectionStrings>
    <system.serviceModel>
        <behaviors>
            <serviceBehaviors>
                <behavior name="">
                    <serviceMetadata httpGetEnabled="true" />
                    <serviceDebug includeExceptionDetailInFaults="false" />
                </behavior>
            </serviceBehaviors>
        </behaviors>
        <serviceHostingEnvironment multipleSiteBindingsEnabled="true" />
    </system.serviceModel>
</configuration>
```

çorbamıza yeni baharatlar katarak devam edelim

Artık servis uygulamamız da hazır olduğuna göre bunu kullanacak olan **Asp.Net Web Uygulamasını** geliştirerek yol alabiliriz. Tahmin edileceği üzere Asp.Net Web uygulamamız,**AdventureWorksService** isimli **WCF Servisini** referans etmelidir.



Yazımızın başında belirttiğimiz senaryomuzdan hatırlayacağınız üzere bir **Web User Control'** den bahsediyorduk. Bu **Web User Control'** ün **Business Object'** imize göre tasarlanması gerekiyor. Bu amaçla aşağıdaki gibi bir tasarım gerçekleştirebiliriz.




```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="ProductControl.ascx.cs" Inherits="CompanyPortal.ProductControl" %>
<style type="text/css">
   .style1
   {
      width: 400px;
      font-family: "Courier New", Courier, monospace;
   }
   .style2
   {
      width: 176px;
   }
   .style3
   {
      width: 176px;
```

```
        font-weight: bold;
    }
</style>

<table border="1" cellpadding="5" cellspacing="1" class="style1">
    <tr>
        <td class="style2">
             </td>
        <td style="text-align: right">
            <asp:Label ID="ProductIdLabel" runat="server"></asp:Label>
        </td>
    </tr>
    <tr>
        <td class="style3">
            Name</td>
        <td>
            <asp:Label ID="ProductNameLabel" runat="server"></asp:Label>
        </td>
    </tr>
    <tr>
        <td class="style3">
            List Price</td>
        <td>
            <asp:Label ID="ProductListPriceLabel" runat="server"></asp:Label>
        </td>
    </tr>
    <tr>
        <td class="style3">
            Color</td>
        <td>
            <asp:Label ID="ProductColorLabel" runat="server"></asp:Label>
        </td>
    </tr>
    <tr>
        <td class="style3">
            Sub Category Name</td>
        <td>
            <asp:Label ID="ProductSubCategoryNameLabel" runat="server"></asp:Label>
        </td>
    </tr>
</table>
```

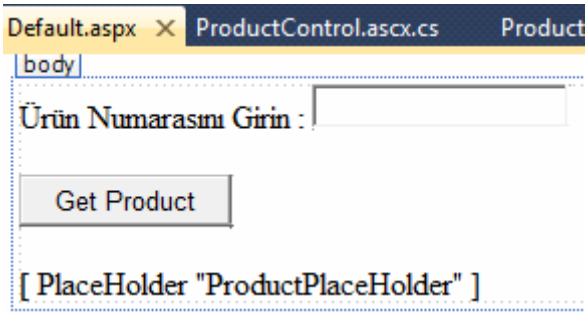
Kod tarafında ise aşağıdaki işlemleri gerçekleştirebiliriz.

```
using System;
using CompanyPortal.Production;

namespace CompanyPortal
{
   public partial class ProductControl
      : System.Web.UI.UserControl
   {
      ProductionServiceClient proxy = new ProductionServiceClient();
      int productId;

      public int ProductId
      {
         get { return productId; }
         set { productId = value; }
      }
      public string ProductName
      {
         get { return ProductNameLabel.Text; }
      }
      public decimal ProductListPrice
      {
         get { return Decimal.Parse(ProductListPriceLabel.Text); }
      }
      public string ProductColor
      {
         get { return ProductColorLabel.Text; }
      }
      public string ProductSubCategoryName
      {
         get { return ProductSubCategoryNameLabel.Text; }
      }

      public void LoadContent()
      {
         ProductBusinessObject bO = proxy.GetProduct(ProductId);
         if (bO != null)
         {
            ProductIdLabel.Text = bO.ProductId.ToString();
            ProductNameLabel.Text = bO.Name;
            ProductListPriceLabel.Text = bO.ListPrice.ToString();
            ProductColorLabel.Text = bO.Color;
            ProductSubCategoryNameLabel.Text = bO.SubCategoryName;
         }
      }
```

```
    }
}
```

Dikkat edileceği üzere **Web User Control** herhangibir ürün numarasına bağlı bir **ProductBusinessObject** içeriğini dış ortama **özellikler(Properties)** yardımıyla da sunmaktadır. Ayrıca içerisinde yer alan **Label** kontrollerinin çalışma zamanında yüklenmesi için **LoadContent** isimli bir metoddan yararlanılmaktadır.

Artık **Web User Control** bileşenimizi örnek bir **aspx** sayfasında kullanmaya çalışarak ilk testimizi gerçekleştirebiliriz. Bunun için aşağıdaki tasarıma ve kod içeriğine sahip bir sayfa eklediğimizi düşünelim.



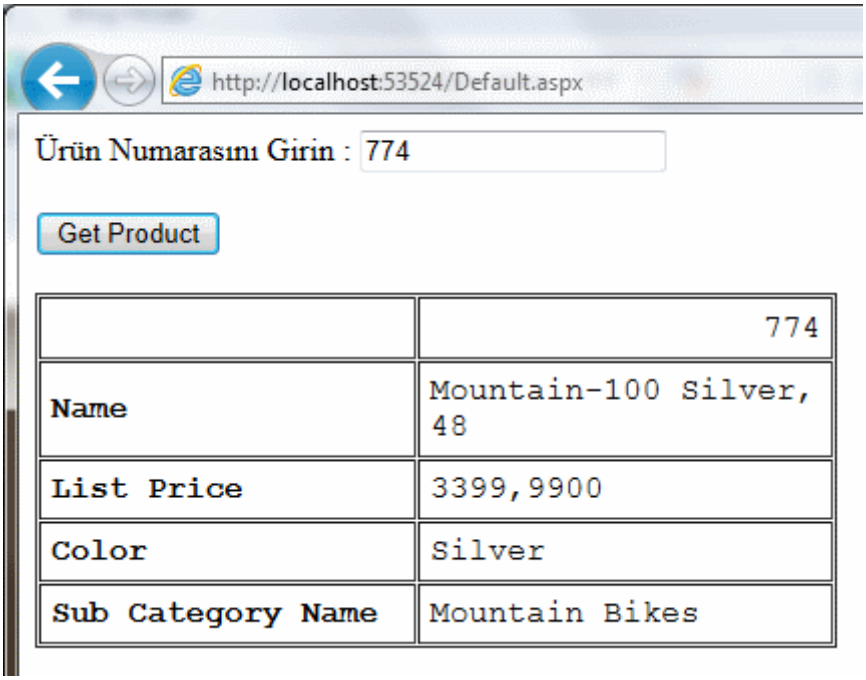

```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="CompanyPortal.Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>

        ürün Numarasını Girin :
        <asp:TextBox ID="ProductIdTextBox" runat="server"></asp:TextBox>
        <br />
        <br />
        <asp:Button ID="GetProductButton" runat="server"
            onclick="GetProductButton_Click" Text="Get Product" />
        <br />
        <br />
        <asp:PlaceHolder ID="ProductPlaceHolder"
runat="server"></asp:PlaceHolder>
```

```
   </div>
   </form>
</body>
</html>
```

ve kod içeriği;

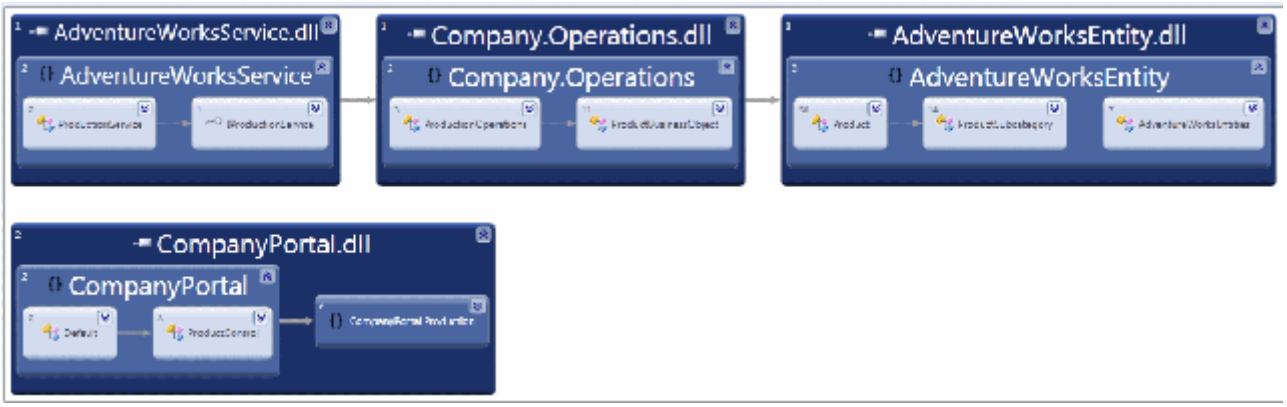
```
using System;

namespace CompanyPortal
{
   public partial class Default : System.Web.UI.Page
   {
      protected void GetProductButton_Click(object sender, EventArgs e)
      {
         int productId;
         if (Int32.TryParse(ProductIdTextBox.Text, out productId))
         {
            ProductControl prdControl = LoadControl("ProductControl.ascx") as
ProductControl;
            if (prdControl != null)
            {
               prdControl.ProductId = productId;
               prdControl.LoadContent();
               ProductPlaceHolder.Controls.Add(prdControl);
            }
         }
      }
   }
}
```

Sayfamızda sembolik olarak **TextBox** kontrolüne girilen sayısal değere göre **Product** bilgisinin çekilmesi ve bulunan ürüne ait bazı bilgilerin dinamik olarak yüklenen **Web User Control** içerisinde gösterilmesi sağlanmaktadır. Söz gelimi çalışma zamanında **774** numaralı ürüne ait bilgileri çekmek istersek aşağıdaki ekran görüntüsünde yer alan sonucu elde ederiz



Eğer örneği buraya kadar başarılı bir şekilde getirdiyseniz şöyle bir arkanıza yaslanın ve neler yaptığımızı bir düşünün

Son olarak **Solution** içerisinde yer alan **Assembly'** lar arasındaki ilişkiyi göstererek yazımızı yavaş yavaş sonlandıralım.



örneği geliştirmek, farklı testler uygulayarak olası hataları düzeltmek, kodu tekrardan gözden geçirip iyileştirmek de sizin göreviniz olsun. Söz gelimi bir alt kategoriye bağlı ürünleri**ProductBusinessObject** tipinden çekip her biri için dinamik olarak **ProductControl** bileşeni üreten ve web sayfasına ekleyen bir senaryoyu çözümümüze ilave edebilirsiniz. Hatta alt kategori bilgilerinin Web uygulaması arayüzüne dahil edilmesi için gerekli geliştirmeleri de yapabilirsiniz. Eğer buraya kadar yaptıklarımızla kafanız çok karıştıysa aşağıdaki parçayı dinleyip kendinize gelmeyi de deneyebilirsiniz

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Company.rar (153,21 kb)

Karmaşık Değil Son Derece Basit (2011-09-29T18:50:00)

generic,generic methods,interface,domain,binary serialization,

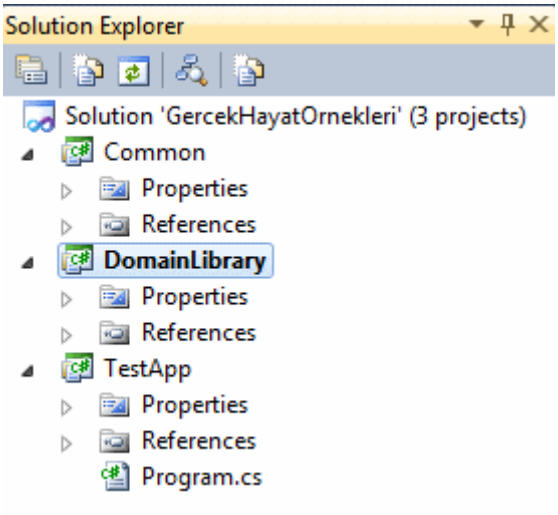
Merhaba Arkadaşlar,

Kurumsal eğitim vermenin en güzel yanlarından birisi de, gelenlerin istekleri ve talepleri doğrultusunda gerçek hayat örneklerini daha kolay bir şekilde kodlayabilmeniz ve gösterebilmenizdir.

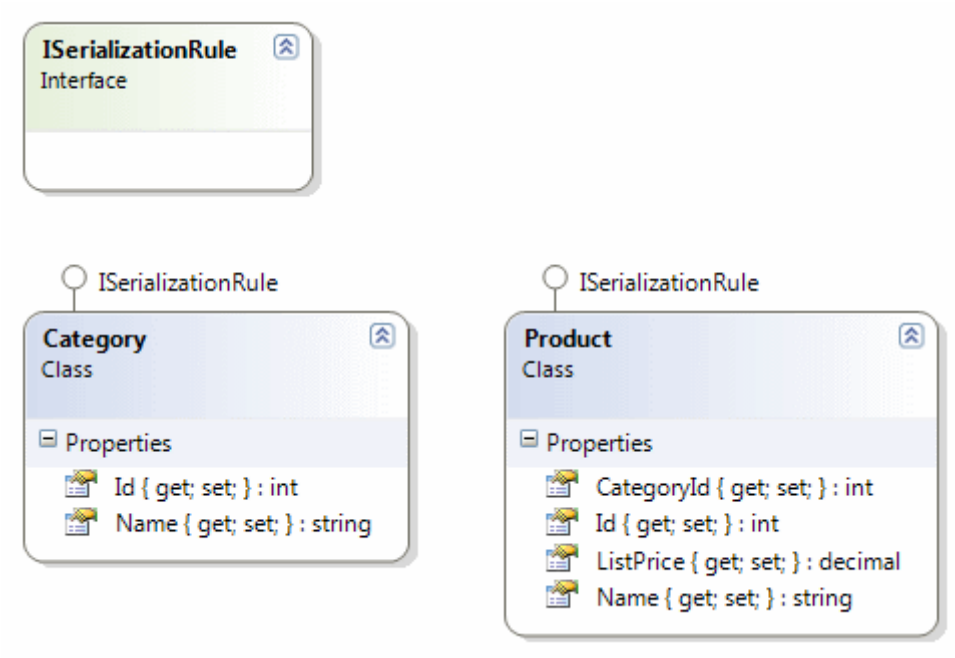
Söz gelimi geçtiğimiz hafta içerisinde vermeye başladığım ve makaleyi yazdığım tarih itibariyle devam etmekte olan bir eğitim sırasında, **Binary** ve**XML Serileştirme** konularını anlatırken, sahip olduğumuz dil ve framework materyallerinden bazılarını iç içe ve ne kadar etkili kullanabildiğimizi gördük

Bu durumdan esinlenerek sizlere de bir gerçek hayat örneği aktarmaya çalışmak isterim.

İlk önce ne yapacağımızı belirtmem gerekiyor ama bunu en sona bırakmak ve ne yapmış olduğumuzu o zaman göstermek*(aslında sizin anladığınızı görmek)* arzusundayım. öncelikli olarak aşağıdaki şekilde görülen **Solution** yapısını oluşturarak işe başlayabiliriz. Tabi ki bu yapı bizim test çözümümüz olarak tasarlanmıştır.



Common ve **DomainLibrary** isimli projelerimiz birer **Class Library** iken **TestApp** tahmin edileceği üzere bir **Console** uygulamasıdır. Şimdi de **DomainLibrary** içeriğini aşağıdaki **Class Diagram**' da olduğu gibi yapılandıralım.



ISerializationRule.cs;

```
namespace DomainLibrary
{
   public interface ISerializationRule
   {
   }
}
```

Product.cs;

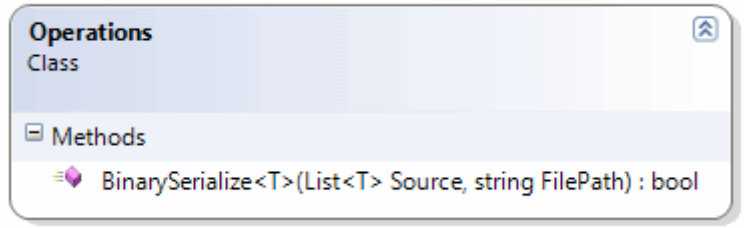
```
using System;

namespace DomainLibrary
{
   [Serializable]
   public class Product
      :ISerializationRule
   {
      public int Id { get; set; }
      public string Name { get; set; }
      public int CategoryId { get; set; }
      public decimal ListPrice { get; set; }
   }
}
```

Category.cs;

```
using System;
namespace DomainLibrary
{
   [Serializable]
   public class Category
     :ISerializationRule
   {
      public int Id { get; set; }
      public string Name { get; set; }
   }
}
```

ISerializationRule interface tipini uygulamakta olan **Category** ve **Product** isimli iki sınıfımız bulunduğunu görmekteyiz. **ISerializationRule** arayüzü herhangibir kural bildirimi yapmasa bile ilerleyen bölümlerde çok kritik bir görevi üstlenecektir 

Şimdi de **Common** isimli sınıf kütüphanemiz içerisine aşağıdaki **Operations** sınıfını ve içeriğini eklediğimizi düşünelim.




```
using System;
using System.Collections.Generic;
using System.IO;
using System.Runtime.Serialization.Formatters.Binary;
using DomainLibrary;

namespace Common
{
   public class Operations
   {
      public bool BinarySerialize<T>(List<T> Source, string FilePath)
         where T : ISerializationRule
      {
         bool result = false;

         try
         {
            using (FileStream fs = new FileStream(FilePath, FileMode.OpenOrCreate,
FileAccess.Write))
```

```
            {
                BinaryFormatter formatter = new BinaryFormatter();
                formatter.Serialize(fs, Source);
                result = true;
            }
        }
        catch (Exception excp)
        {
            throw excp;
        }
        return result;
    }
  }
}
```

Operations sınıfı içerisinde **Binary** serileştirme işlemi olan bir metod olduğunu görmektesiniz. Söz konusu metod **generic** olarak tasarlanmıştır. Generic olmakla kalmayıp bir de **kısıtlama(Constraints)** getirmiştir. Bu kısıtlamaya göre **T** tipinin **ISerializationRule** isimli arayüz tarafından taşınabilen bir referans olması şartı konulmaktadır Bir başka deyişle az önce tasarlamış olduğumuz **Domain** yapısı içerisinde yer alan ve **ISerializationRule** arayüzünü uygulayan tipler için bu metodun kullanılabilmesi mümkündür. Dolayısıyla bu koşulun dışında kalan tipler için söz konusu metod kullanılamayacaktır. Sanırım bu gerçek hayat örneğinin en can alıcı noktası da burasıdır. Bizim sahip olduğumu bir **Domain**ile çalışabilecek **generic** bir serileştirme metodu geliştirmiş bulunmaktayız.

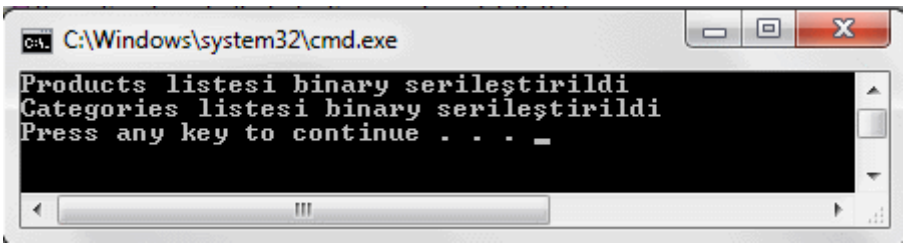
Metodun içeriği son derece basittir. **Exception** yönetimi metodu kullanan bir üst katmana bırakılmıştır*(catch bloğu içerisinde yaptığımız throw hareketine dikkat edin)* Diğer yandan **BinaryFormatter** tipinden yararlanılmış ve metoda parametre olarak gelen **FilePath** ile işaret edilen ve **FileStream** ile yazmak üzere açılan dosyaya doğru bir serileştirme işlemi gerçekleştirilmektedir.

Şimdi söz konusu operasyonu test edeceğimiz kod içeriğini de **Console** uygulamamızda aşağıdaki gibi geliştirelim.

```
using System;
using System.Collections.Generic;
using System.IO;
using Common;
using DomainLibrary;

namespace TestApp
{
```

```
class Program
{
   static void Main(string[] args)
   {
      List<Category> categories = new List<Category>()
      {
         new Category{ Id=1,Name="Book"},
         new Category{ Id=2,Name="Music"}
      };

      List<Product> products = new List<Product>()
      {
         new Product{ Id=1,CategoryId=1, Name="Kitap 1", ListPrice=10},
         new Product{ Id=2,CategoryId=1, Name="Kitap 2", ListPrice=5},
         new Product{ Id=3,CategoryId=2, Name="Muzik 1", ListPrice=15},
         new Product{ Id=4,CategoryId=1, Name="Kitap 3", ListPrice=3},
         new Product{ Id=5,CategoryId=2, Name="Muzik 2", ListPrice=9},
         new Product{ Id=6,CategoryId=1, Name="Kitap 4", ListPrice=8},
      };

      Operations opt = new Operations();
      bool result1=opt.BinarySerialize<Product>(products,
Path.Combine(Environment.CurrentDirectory, "Products.bin"));
      bool result2 = opt.BinarySerialize<Category>(categories,
Path.Combine(Environment.CurrentDirectory, "Categories.bin"));

      if(result1)
         Console.WriteLine("Products listesi binary serileştirildi");
      if(result2)
         Console.WriteLine("Categories listesi binary serileştirildi");
   }
}
```

örneğimizde sembolik olarak **Product** ve **Category** tipinden birer koleksiyon üretilmekte ve bunlar için Binary serileştirme işlemi icra edilmektedir. Uygulamayı çalıştırdığımızda aşağıdakine benzer bir ekran çıktısı ile karşılaşırız



Tabi çıktı olarak üretilen **Binary** dosyaların içeriği de aşağıdaki gibi oluşturulacaktır.

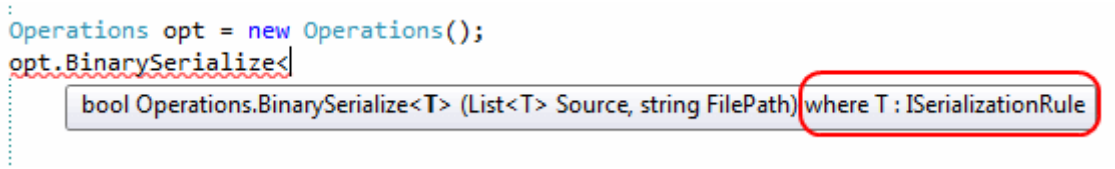
Categories.bin içeriği;

ÿÿÿÿ DDomainLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null Â□System.Collections.Generic.List`1[[DomainLibrary.Category, DomainLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null]] _items_size_version DomainLibrary.Category[] DomainLibrary.Category
DomainLibrary.Category <Id>k__BackingField<Name>k__BackingField Book Music

Products.bin içeriği;

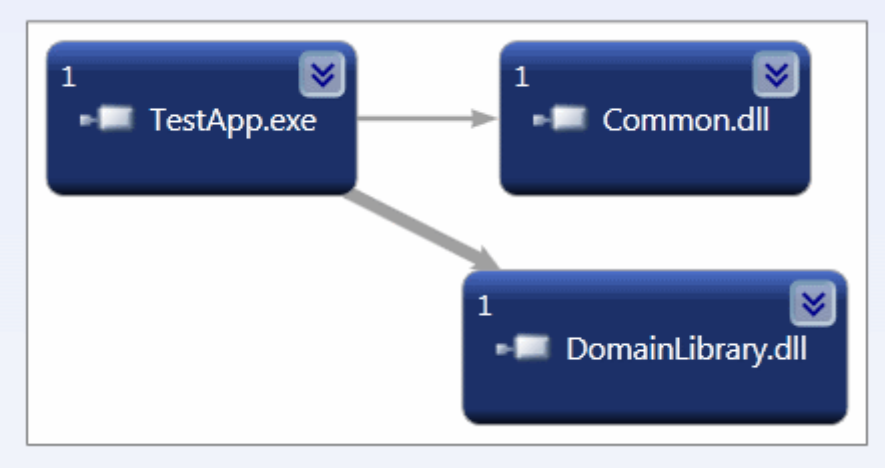
ÿÿÿÿ DDomainLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null €System.Collections.Generic.List`1[[DomainLibrary.Product, DomainLibrary, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null]] _items_size_version DomainLibrary.Product[] D omainLibrary.Product
DomainLibrary.Product <Id>k__BackingField<Name>k__BackingField<CategoryId>k __BackingField<ListPrice>k__BackingField
Kitap 1 10 Kitap 2 5 Muzik 1 15
Kitap 3 3 Muzik 2 9 Kitap 4 8

Aslında bu geliştirdiğimiz örnek ile kazandığımız bir takım avantajlar olduğunu vurgulamalıyız. öncelikli olarak **development** safhasındayken **Binary** serileştirme işini üstlenen metoda atayabileceğimiz tipler için **Business** anlamda bir **Domain** kuralı getirmiş bulunmaktayız. Bunu metodu çağırdığımız sırada da zaten net bir şekilde görebiliriz.



Dikkat edileceği üzere kırmızı kutucuk içerisinde almış olduğumuz kısım ile **T** tipinin **ISerializationRule** arayüzü tarafından taşınabilecek bir tip olması zorunluluğu geliştiriciye bildirilmiş oluyor Sanırım şu anda ne demek istediğimi daha net anlatabilmişimdir.

Geliştirdiğimiz örnek **Solution** içerisinde yer alan **Assembly'** lar arası bağları ele alarak makalemizi yavaş yavaş sonlandırmaya başlayalım. Tam olarak **Assembly'** larımız arası ilişki aşağıdaki şekilde görüldüğü gibidir.



Generate Dependency Graph' ı seviyorummm

Peki bu çözümde neleri kullandık?

- **Generic** bir **metod** geliştirdik.
- **Generic** metodda kullandığımız **T** tipi için **generic kısıtlama(Constraint)** kullandık.
- Kendi **Domain** yapımızı düşündük ve bir **arayüz(ISerializationRule)** ile generic kısıtlama için imkan sağladık.
- **BinaryFormatter** tipi ile serileştirme işlemini icra ettik.
- **Serileştirme** hedefi olarak fiziki bir dosya bağlantısını kullandık*(FileStream)*.
- **Class Diagram'** ları kullanarak **Domain** yapımızı daha net görebildik.
- **Assembly** bazında **Dependency Graph** üreterek **Assembly'** larımız arası referans bağımlılıklarını da daha net görebildik.

Görüldüğü üzere sahip olduğumuz .Net bilgi ve materyallerini bazı durumlarda bir araya getirip gerçek hayat senaryoları için icra ettirebiliyoruz. Tabi söz konusu senaryoya eklenebilecek daha pek çok fonksiyonellik söz konusu olabilir. İlerleyen zamanlarda başka gerçek hayat örneklerini de sizlerle paylaşmaya çalışıyor olacağım. özellikle bu yazıda katılımcı arkadaşlarımızın oldukça büyük emeği var. Kendilerine de çok teşekkür ediyorum. Tekrardan görüşünceye dek hepinize mutlu günler dilerim

GercekHayatOrnekleri.rar (58,66 kb)

Haydi Bir Captcha Kontrolü Yazalım (2011-09-27T17:11:00)

asp.net,captcha,handler,

Merhaba Arkadaşlar,

Şu sıralarda kurumsal bir Asp.Net eğitimi vermekteyim. Eğitim içeriği oldukça geniş ve güzel konuları içermekte. Bunlardan bir tanesi de Asp.Net uygulamalarında **Captcha** doğrulamasının kullanımı. Bildiğiniz üzere web ortamı üzerinde özellikle Form veri girişlerinin yapıldığı senaryolarda akıllı robotların gereksiz yere post atma işlemlerini engellemek çok önemlidir.

örneğin şu an kullanmakta olduğum **BlogEngine** sürümünde, sisteme monte edilmiş bir **Captcha** kontrolü veya modülü bulunmamakta *(Biliyorum son sürüme geçmeliydim)* . Bu nedenle özellikle yorum kısımlarında dünyanın çeşitli bölgelerindeki robot programlarının tacizlerine fazlasıyla maruz kalmaktayım. Anlamsız pek çok bilgiden oluşan spam yorumlar söz konusu.

E tabi diyebilirsiniz ki, "Ya Hocam sen de amma yaptın...Eklesene şu Captcha kontrolünü bloğa...Hayret bişi" E ama ne demişler bilirsiniz... Terzi kendi söküğünü dikemezmiş *(Aslında ben söküğümü nedense dikmek istemiyorum sanırım. üşengeçlik bu olsa gerek)*

Neyse sözü fazla uzatmadan konumuza devam edelim. Asp.Net uygulamalarında form veri girişlerinin yapılabildiği ve post işlemlerinin gerçekleştirilebildiği her ortamda robot saldırılarına maruz kalmamız olasıdır. Bu sebepten formu dolduran kişinin gerçek insan gözüne sahip olduğunu bir şekilde anlamamız ve bunu doğrulatmamız gerekmektedir. İşte Captcha kontrolleri bu noktada devreye girmektedir. üretilen resim formatlı metinsel içeriklerin o anki HTTP Context verisinden okunması mümkün değildir. En azından şu an için. Dolayısıyla bu içeriği gören birisinin elle giriş yapması gerekmektedir. Bize kalan ise sadece ve sadece Captcha içeriği ile kullanıcının girdiği verinin eşit olup olmadığını kontrol etmektir.

Olayı daha net bir şekilde kavrayabilmek adına örnek bir uygulama üzerinden gitmemizde yarar vardır. Bu amaçla dilerseniz **Visual Studio 2010** ortamı üzerinde basit bir **Web Site** şablonu açarak işe başlayalım. Söz konusu **Captcha** kontrolü esas itibariyle **Drawing** tipleri tarafından çizilen bir resim ve üzerine yerleştirilen bir takım sayısal veya karakter bazlı verilerden oluşmaktadır. Burada bahsedilen çizim işini ise genellikle bir **Handler** tipi üstlenmektedir. İşte Handler içeriğimiz.

<%@ WebHandler Language="C#" Class="CaptchaHandler" %>

```
using System;
using System.Web;
using System.Drawing;
using System.IO;
using System.Drawing.Imaging;
using System.Web.SessionState;

public class CaptchaHandler
    : IHttpHandler,IReadOnlySessionState
```

```
{
   public void ProcessRequest (HttpContext context) {

      using (Bitmap bmp = new Bitmap(220, 80)) //120X40 uzunluğunda bir Bitmap tanımlıyoruz.
      {
         using (Graphics painter = Graphics.FromImage(bmp)) // Grafik nesnesi söz konusu alan üzerinde çizim yapabilmek için bmp isimli nesneden üretiliyor
         {
            painter.Clear(Color.LightGray);

            // Font nesnesi üretiliyor.
            using (Font writer = new Font("Helvetica", 10, System.Drawing.FontStyle.Bold))
            {
               string capcthaContent = string.Empty;
               // Session nesnesinden Captcha kontrolünün içeriğini oluşturan bilgi alınır

                  if (context.Session["CaptchaContent"].ToString() != null)
                     capcthaContent = context.Session["CaptchaContent"].ToString();

               // ve bu içerik belirlenen font, fırça rengi bilgileri ile Graphics tipinden olan painter üzerine çizdirilir
               painter.DrawString(capcthaContent, writer, Brushes.Black, 3, 3);

               using (MemoryStream mStream = new MemoryStream())
               {
                  bmp.Save(mStream, ImageFormat.Gif);

                  // üretilen captcha resmini HttpContext tipinin Response özelliği üzerinden Binary formatta yazdırarak görünmesini sağlıyoruz.
                  byte[] bmpBytes = mStream.GetBuffer();
                  context.Response.ContentType = "image/gif";
                  context.Response.BinaryWrite(bmpBytes);
               }
            }
         }
      }
      context.Response.End();
   }

   public bool IsReusable {
      get {
         return false;
      }
```

```
    }
}
```

Aslında Handler tipinin temel görevi **Captcha** resmini içeriği ile birlikte çizmektir. çizim işlemi sırasında bir Asp.Net sunucu kontrolünden yararlanılması gerekmektedir. Bu sunucu kontrolü yukarıda yazılmış olan Handler tipinden yararlanacaktır. Söz konusu kontrol içeriğini aşağıdaki gibi oluşturabiliriz.

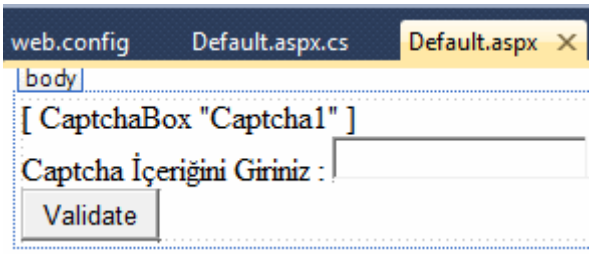
```
using System;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Collections.Generic;

namespace CustomControls
{
   public class CaptchaBox
      : Control
   {
      Image capcthaImage;

      public string Text
      {
         get
         {
            if (HttpContext.Current.Session["CaptchaContent"] != null)
               return HttpContext.Current.Session["CaptchaContent"].ToString();
            else
               return null;
         }
      }

      protected override void OnLoad(EventArgs e)
      {
         base.OnLoad(e);

         List<string> strArray=new List<string> { "A", "B", "C", "ç","D", "E", "F",
"G","Äž", "H", "I","İ", "J", "K", "L", "M", "N", "O","ö", "P", "R", "S", "Ş","T", "U","ü",
"V","Y", "Z", "a", "b", "c", "ç","d", "e", "f", "g", "h","ı", "i", "j", "k", "l", "m", "n", "o","ö",
"p", "r", "s","ş", "t", "u","ü" ,"v", "y", "z", "0", "1", "2", "3", "4", "5", "6", "7", "8", "9" };

         Random rnd = new Random();
         string strCaptcha = string.Empty;
         // 10 haneli rastgele bir Captcha değeri üretmek için aşağıdaki döngüden
yararlanıyoruz.
```

```
            for (int i = 0; i < 10; i++)
            {
               int j = Convert.ToInt32(rnd.Next(0, strArray.Count-1));
               strCaptcha += strArray[j].ToString();
            }

            HttpContext.Current.Session.Add("CaptchaContent", strCaptcha);

            capcthaImage = new Image();
            capcthaImage.ImageUrl = "~/CaptchaHandler.ashx";
            // captcha resmini üretecek olan Handler' ımızı ImageUrl özelliğine set ettikten
sonra CaptchaBox kontrolünün Controls koleksiyonuna ekliyoruz.
            this.Controls.Add(capcthaImage);
         }
      }
}
```

Oldukça basit bir içeriğe sahip olan bu kontrol
aslında **strArray** isimli **string** tipindeki **List** koleksiyonu içerisinden **rastgele 10** değeri alarak Captcha handler tipine vermektedir. Bunun karşılığında Captcha Handler tipinin ürettiği resim içeriği sayfaya basılmaktadır. CustomControls isim alanında yer alan kontrolümüzü bir ön takı(Tag Prefix) ile kullanabilmek adına **web.config**dosyasında aşağıdaki bildirimi yapmamız yeterlidir.

```
<?xml version="1.0"?>
<configuration>
   <system.web>
      <compilation debug="true" targetFramework="4.0"/>
      <pages>
         <controls>
            <add namespace="CustomControls" tagPrefix="ccntrl"/>
         </controls>
      </pages>
   </system.web>
</configuration>
```

Artık örnek bir web sayfasında söz konusu kontrolümüzü test edecek içeriği ve kod parçasını geliştirebiliriz. Dilerseniz aspx sayfamızın ön yüz içeriğini aşağıdaki gibi geliştirelim.




```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
   <ccntrl:CaptchaBox ID="Captcha1" runat="server" />
   <br />
   Captcha İçeriğini Giriniz :
   <asp:TextBox ID="txtContent" runat="server" />
   <br />
   <asp:Button ID="btnSubmit" runat="server" Text="Validate" />
   </div>
   </form>
</body>
</html>
```

Burada dikkat edilmesi gereken bir nokta vardır. O da düğmeye bastığımızda **PostBack** işlemi sırasında Handler tipinin çok doğal olarak tekrar devreye girmesi ve yeniden taze bir **Captcha**içeriği üretecek olmasıdır. Dolayısıyla **TextBox** içeriği ile **Capctha** kontrolünü karşılaştırırken **PreInit** olayından yararlanabiliriz.
İşte **aspx.cs** tarafındaki kodlarımız.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Web.UI;
using System.Web.UI.WebControls;
```

```
public partial class _Default : System.Web.UI.Page
{
   // Postback olması durumunda Captcha içeriği yeniden üretileceğinden PreInit olay metodunu değerlendiriyoruz.
   protected override void OnPreInit(EventArgs e)
   {
      if (IsPostBack)
         if (Session["CaptchaContent"].ToString() == Request["txtContent"]) // Session' da tutulmakta olan Captcha içeriği ile kullanıcının girmiş olduğu bilgiyi karşılaştırıyoruz.
            Response.Write("Sen bir robot değilsin :)");
         else
            Response.Write("Seni gidi seni :)");

      base.OnPreInit(e);
   }
}
```

Artık uygulamamızı test edebiliriz ne dersiniz. Hatta edebilirsiniz örneğin ben Captcha içeriğini doğru girdikten sonra aşağıdaki çıktıyı elde ettim. Tabi burada post işleminden sonra yeni bir Captcha değeri üretildiğini unutmayalım. Bu yüzden TextBox kontrol içeriği ile şu anki Capctha içeriği farklı görünüyor. En iyisi siz uygulamayı bir çalıştırında ne demek istediğimi kendiniz görün

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

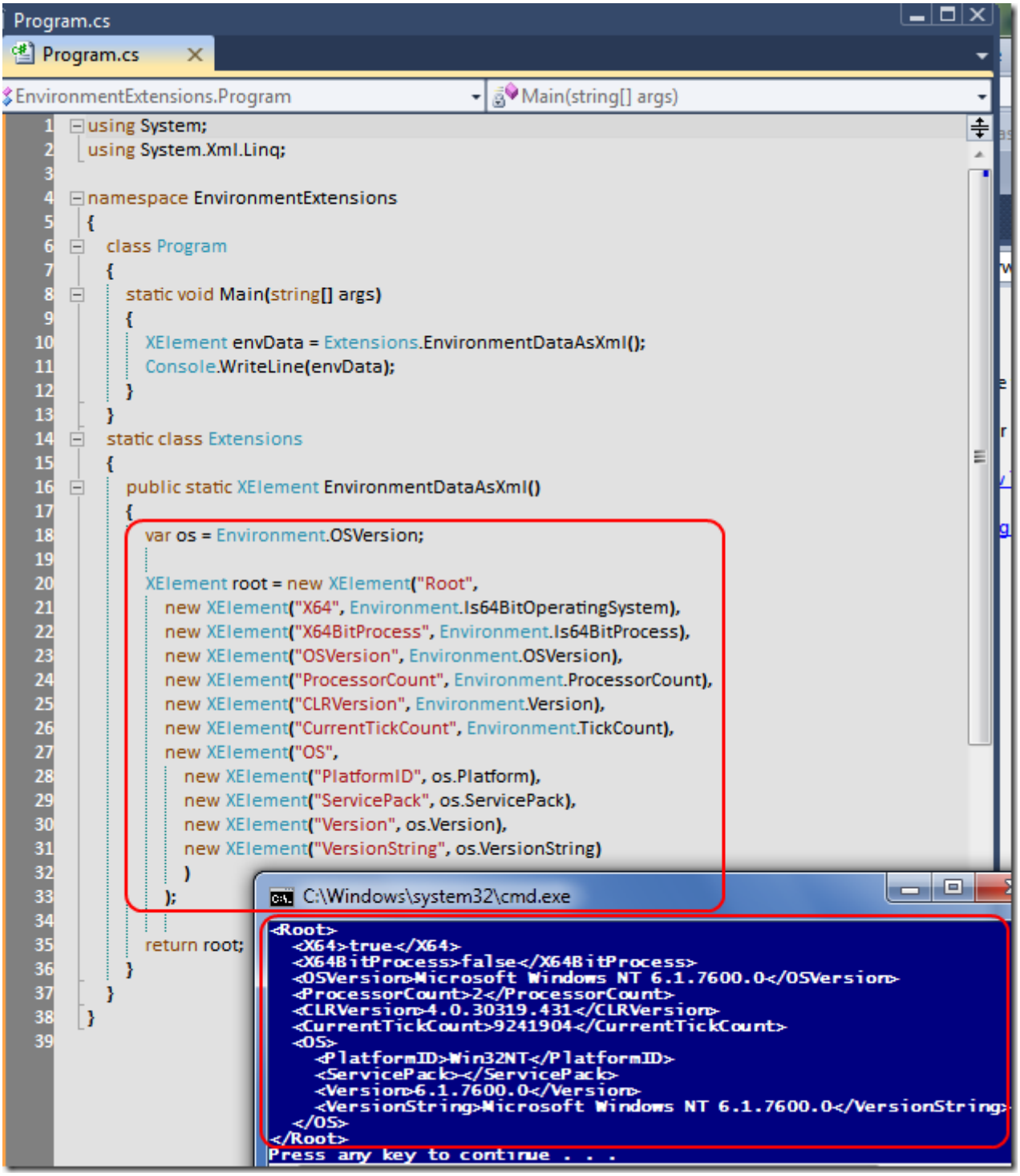
CaptchaKullanimi.rar (3,23 kb)

Tek Fotoluk İpucu-32(Environment Verisini XML Olarak Sunmak) (2011-09-22T16:05:00)

c#,c# 3.0,c# temelleri,system.environment,xlinq,xelement,

Merhaba Arkadaşlar,

System.Environment tipi içerisinde son derece yararlı ortam bilgileri bulunmaktadır. Bu bilgileri elde etmek son derece kolaydır. Hatta dilerseniz bunları XML formatında dış dünyaya sunabilirsinizde. Nasıl mı? 😉



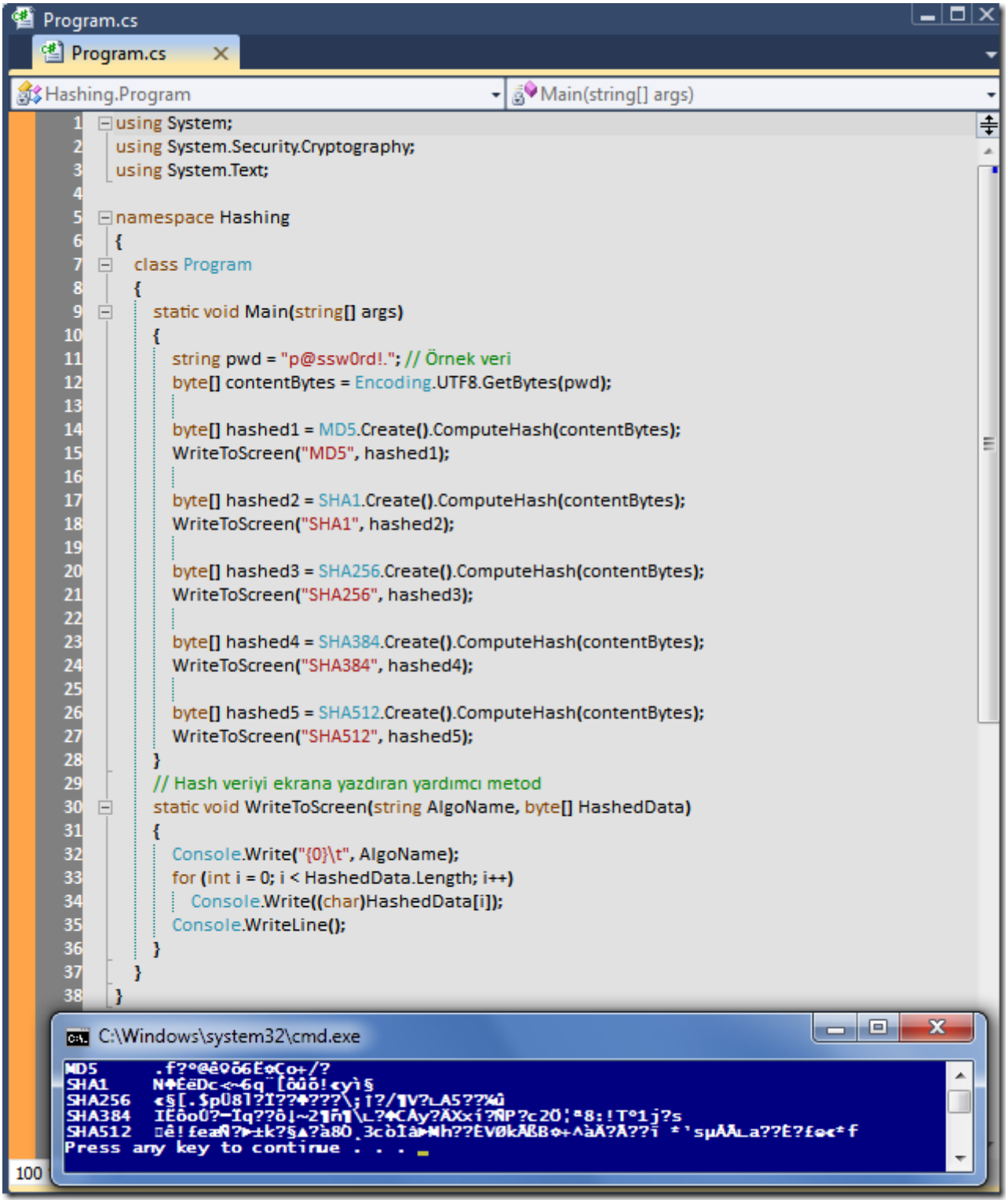
EnvironmentExtensions.rar (22,66 kb)

Tek Fotoluk İpucu-31(Hashing) (2011-09-18T11:50:00)

c#,c# temelleri,security,hashing,md5,sha1,sha256,sha384,sha512,

Merhaba Arkadaşlar,

Hiç bir zaman kullanıcılarımıza ait şifreleri açık formatta saklamamamız gerekir. En basit anlamda söz konusu verileri Hash' leyerek tutmak en doğrusudur. Bu anlamda .Net tarafında kullanımı son derece basit olan Hash algoritma tipleri mevcuttur. Nasıl kullanıldığını merak ediyor musunuz? 



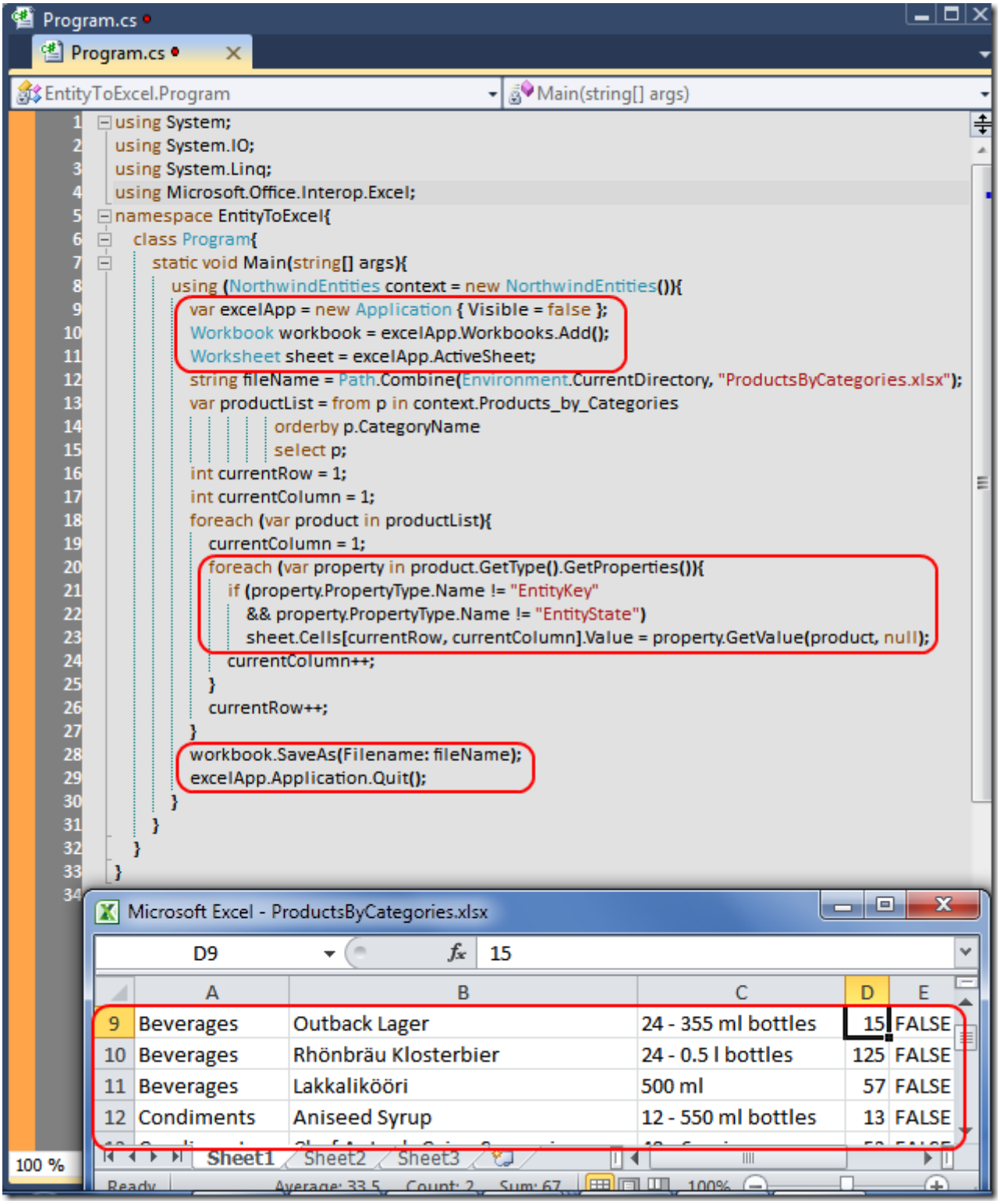
Hashing.rar (21,76 kb)

[Tek Fotoluk İpucu-30 (Entity Sorgusundan Excel Dosyasına) (2011-09-13T22:30:00)]

.net framework 4.0,c#,office interop,linq,entity framework,excel,

Merhaba Arkadaşlar,

.Net Framework 4.0' ın getirdiği pek çok yenilik sayesinde Office gibi API' leri kullanmamız çok daha fazla kolaylaştı. örneğin bir Entity sorgusunun sonucunu Excel dosyasına aktarmak için daha basit kodlamalar yapabiliyoruz. Nasıl mı?



EntityToExcel.rar (57,44 kb)

Zenith Entity Framework Eğitimi (2011-09-07T16:39:00)

entity framework,ado.net entity framework,eğitim,zenith bilişim,cenk özdemir,

Merhaba Arkadaşlar,

Malum çok uzun bir süredir eli ayağı teknik işlerden çekmiş bulunmaktayım. Kendimi çok fazla yormadan ama sizlere yine de yardımcı olduğunu düşündüğüm Tek Fotoluk İpucu serisi dışında giriş seviyesindeki bir eğitimle karşınızdayım. Konumuz **Ado.Net Entity Framework**.

Sevgili hocam **Cenk özdemir** ve eğitim kurumları **Zenith Bilişim** bu konuda bana destek oldular. Aşağıda eğitim içeriğini ve iletişim bilgilerini bulabilirsiniz. Daha başka eğitimlerimizin de olacağını şimdiden belirtmek isterim

Eğitim İçeriği

- EF Yokken...
- ORM Nedir?
- LINQ' i tanıyalım.
- EF Nedir?
- Database First yaklaşımı.
- Model First yaklaşımı.
- Code First yaklaşımı.
- EF ile temel CRUD işlemleri.
- Lazy ve Eager yüklemeleri.
- ComplexType özelliği.
- EF ile Stored Procedure'lerin kullanılması.
- Entity Splitting – Table Splitting.
- EF ile Transaction yönetimi.
- EF ve Servisler(Ado.Net Data Services).

Zenith Bilişim İletişim Bilgileri

Adres : SARAYARDI CADDESİ NO: 96 / 8 HASANPAŞA - KADIKöY

İletişim :
Cenk özdemir
0533 392 31 34

EMail : info@zenithbilisim.com

Kroki

Eager Loading, Lazy Loading, Explicit Loading (2011-09-07T11:30:00)

entity framework,ado.net entity framework 4.0,eager loading,lazy loading,explicit loading,

Merhaba Arkadaşlar,

Şöyle basit tek bir Main metodu içerisinde, Entity Framework' teki Loading çeşitlerini görmek ister miydiniz? öyleyse aşağıdaki kod bloğu oldukça işinize yarayacaktır diye düşünüyorum. Senaryolar oldukça basit. Meşhur Chinook veritabanında yer alan Artist ve buna bağlı Album tablolarını ele alıyoruz.

Eager Loading Senaryosunda Artistler ve bunlara bağlı olan tüm Albumlerin Sub Select içeren bir Select sorgusunda tek seferde yüklendiğine şahit olmaktayız.

Varsayılan olarak açık olan Lazy Loading senaryosunda ise 1 numaralı Artist ve buna bağlı Albümler çekiliyor. Dikkat edilmesi gereken artiste bağlı albümlerin sorgulandığı anda arka planda bir SQL sorgusu ile bağlı veri kümesinin çekiliyor olması. Yani lazım olduğunda.

Son olarak Explicit Loading senaryosunda ise Lazy Loading' deki davranış biçiminin kodlamacı tarafında açık bir şekilde yapılması durumuna şahit oluyoruz. Biz Load metodunu çağırmadığımız sürece bağlı veri kümesi yüklenmiyor.

İşte kod parçamız.

```
using System;
using System.Linq;

namespace ConsoleApplication5
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Eager Loading

         using (ChinookEntities context = new ChinookEntities())
         {
            // Eager Loading için LazyLoading özelliği kapatılmış olmalıdır
            context.ContextOptions.LazyLoadingEnabled = false;

            // Tek bir Select sorgusu içerisinde Sub Select kullanılarak Hem Artist hem de
bağlı Album kümeleri yüklenir
            var resultSet = from a in context.Artists.Include("Albums")
                            select a;

            //foreach (var r in resultSet)
            //{
            //    Console.WriteLine(r.Name);
            //    foreach (var album in r.Albums)
            //    {
            //        Console.WriteLine("\t{0}",album.Title);
            //    }
            //}
         }

         #region SQL Sorgusu

         // Eager Loading için arka planda çalışan SQL Sorgusu şu şekildedir
         //           SELECT
         //[Project1].[ArtistId] AS [ArtistId],
         //[Project1].[Name] AS [Name],
         //[Project1].[C1] AS [C1],
         //[Project1].[AlbumId] AS [AlbumId],
         //[Project1].[Title] AS [Title],
         //[Project1].[ArtistId1] AS [ArtistId1]
         //FROM ( SELECT
         //    [Extent1].[ArtistId] AS [ArtistId],
         //    [Extent1].[Name] AS [Name],
         //    [Extent2].[AlbumId] AS [AlbumId],
```

```
        //    [Extent2].[Title] AS [Title],
        //    [Extent2].[ArtistId] AS [ArtistId1],
        //    CASE WHEN ([Extent2].[AlbumId] IS NULL) THEN CAST(NULL AS int)
ELSE 1 END AS [C1]
        //    FROM  [dbo].[Artist] AS [Extent1]
        //    LEFT OUTER JOIN [dbo].[Album] AS [Extent2] ON [Extent1].[ArtistId] =
[Extent2].[ArtistId]
        //)  AS [Project1]
        //ORDER BY [Project1].[ArtistId] ASC, [Project1].[C1] ASC

        #endregion

        #endregion

        #region Lazy Loading

        // Varsayılan olarak Lazy Loading özelliği açıktır

        using (ChinookEntities context = new ChinookEntities())
        {
           var resultSet = from a in context.Artists
                           where a.ArtistId==1
                           select a;

           //foreach (var r in resultSet)
           //{
           //    Console.WriteLine(r.Name);
           //    foreach (var album in r.Albums) // Bağlı Album kümesi talep edildiğinde
ikinci Select sorgusu otomatik olarak çalışır
           //    {
           //        Console.WriteLine("\t{0}", album.Title);
           //    }
           //}

           #region SQL Sorguları

           //            SELECT
           //[Extent1].[ArtistId] AS [ArtistId],
           //[Extent1].[Name] AS [Name]
           //FROM [dbo].[Artist] AS [Extent1]
           //WHERE 1 = [Extent1].[ArtistId]


           //            exec sp_executesql N'SELECT
           //[Extent1].[AlbumId] AS [AlbumId],
```

```
            //[Extent1].[Title] AS [Title],
            //[Extent1].[ArtistId] AS [ArtistId]
            //FROM [dbo].[Album] AS [Extent1]
            //WHERE [Extent1].[ArtistId] = @EntityKeyValue1',N'@EntityKeyValue1
int',@EntityKeyValue1=1

            #endregion

        }

        #endregion

        #region Explicit Loading

        // Bağlı veri kümesi bilinçli olarak Load metodu ile yüklenir

        using (ChinookEntities context = new ChinookEntities())
        {
            // Explicit Loading için LazyLoading özelliği kapatılmış olmalıdır
            context.ContextOptions.LazyLoadingEnabled = false;

            var resultSet = from a in context.Artists
                            where a.ArtistId==1
                            select a;

            foreach (var r in resultSet)
            {
                Console.WriteLine(r.Name);

                if (!r.Albums.IsLoaded) //Albums seti yüklenmediyse
                    r.Albums.Load();  //yükle (Burada ikinci Select sorgusu çalışır)

                foreach (var a in r.Albums)
                {
                    Console.WriteLine("\t{0}",a.Title);
                }
            }
        }

        #region SQL Sorguları

        //          SELECT
        //[Extent1].[ArtistId] AS [ArtistId],
        //[Extent1].[Name] AS [Name]
        //FROM [dbo].[Artist] AS [Extent1]
        //WHERE 1 = [Extent1].[ArtistId]
```

```
            //    exec sp_executesql N'SELECT
            //[Extent1].[AlbumId] AS [AlbumId],
            //[Extent1].[Title] AS [Title],
            //[Extent1].[ArtistId] AS [ArtistId]
            //FROM [dbo].[Album] AS [Extent1]
            //WHERE [Extent1].[ArtistId] = @EntityKeyValue1',N'@EntityKeyValue1
int',@EntityKeyValue1=1

            #endregion

            #endregion
        }
    }
}
```

ConsoleApplication5.rar (98,75 kb)

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

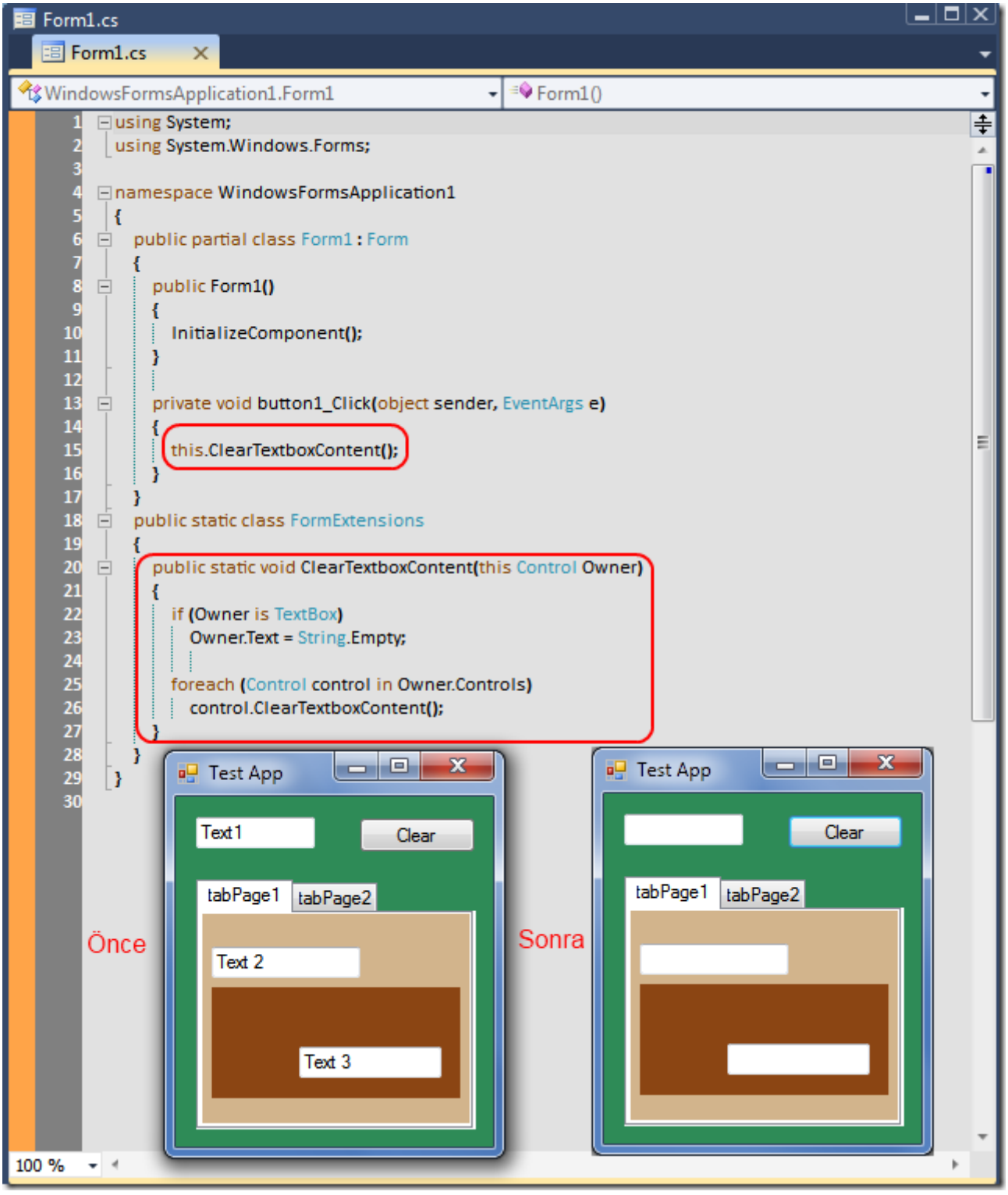
Chinook veritabanını bu adresten indirebilirsiniz.

Tek Fotoluk İpucu-29 (Ne Kadar TextBox Varsa) (2011-08-26T15:17:00)

c#,c# temelleri,windows forms,extension methods,textbox,

Merhaba Arkadaşlar,

Kaliteli kod yazmak için aslında biraz titiz düşünmek gerekir. Söz gelimi bir Windows programlamada bir Container kontrol içerisindeki tüm TextBox' ların içeriğini temizlemek istediğiniz bir durumda nasıl kodlama yaparsınız? İşin içerisine Recursive metod formatını katabilirsiniz. Hatta bunu bir Extension Method haline de getirebilirsiniz. Nasıl mı? 



WindowsFormsApplication1.rar (40,37 kb)

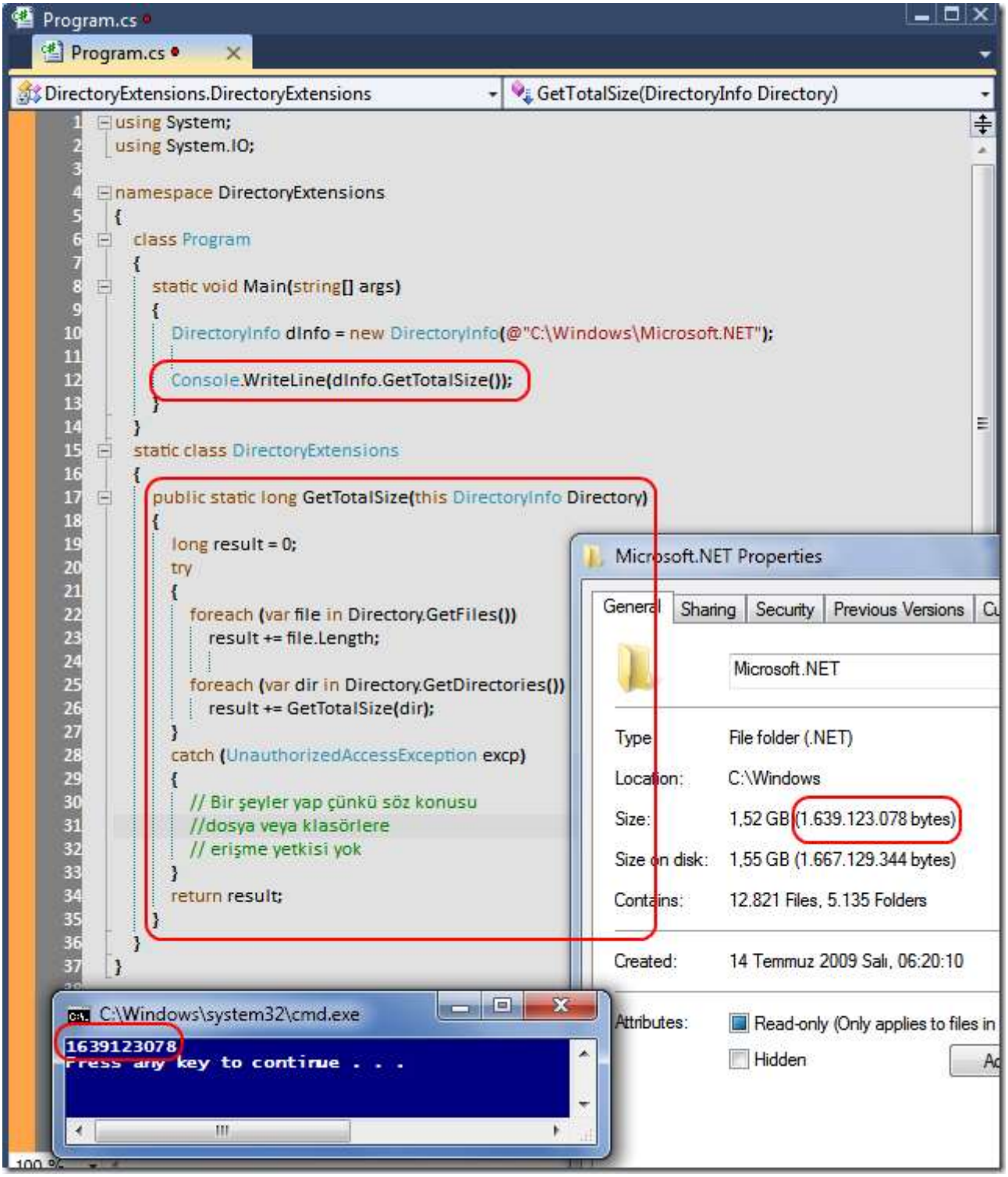
Tek Fotoluk İpucu-28(Bir Klasörün Yaklaşık Toplam Boyutunu Bulmak) (2011-08-24T09:09:00)

c#,c# temelleri,io,directoryinfo,extension methods,

Merhaba Arkadaşlar,

Bir klasörün tüm içeriğinin toplam boyutunu öğrenmek isteyebiliriz. Bunun için DirectoryInfo tipine bir ExtensionMethod eklersek de güzel olur. Hatta bu metodun alt klasörleri de gezebilmesi için Recursive olarak yazılması da gerekir. Nasıl mı?

Not : Yanlız erişim yetkisi olmayan klasörler söz konusu olduğunda boyut bilgisi eksik çıkacaktır. Bunun çözümünü de size bırakıyorum. Biraz araştırın bakalım



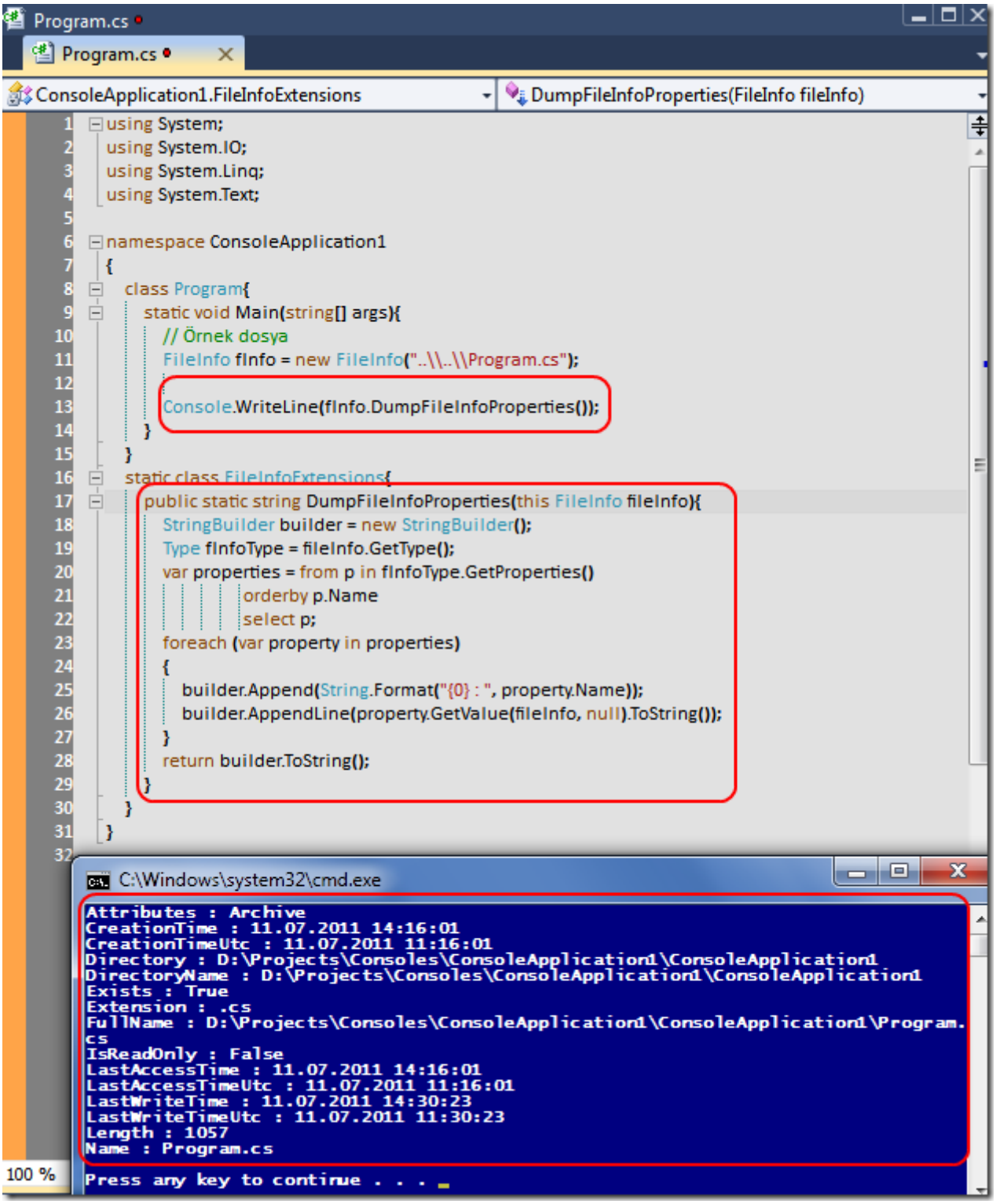
DirectoryExtensions.rar (23,28 kb)

Tek Fotoluk İpucu-27(FileInfo Bilgisinin Tamamını İndirmek) (2011-08-22T13:51:00)

c#,c# temelleri,io,fileinfo,extension methods,

Merhaba Arkadaşlar,

FileInfo tipi yardımıyla bir dosyanın pek çok özelliğine erişebiliriz bildiğiniz üzere. Peki tüm bu bilgileri tek bir String içerisinde toplamak ister misiniz? Söz gelimi loglamalarda bu oldukça işe yarayabilir. Hatta bunu bir Extension method olarak da yazabiliriz. Nasıl mı?



ConsoleApplication1.rar (23,97 kb)

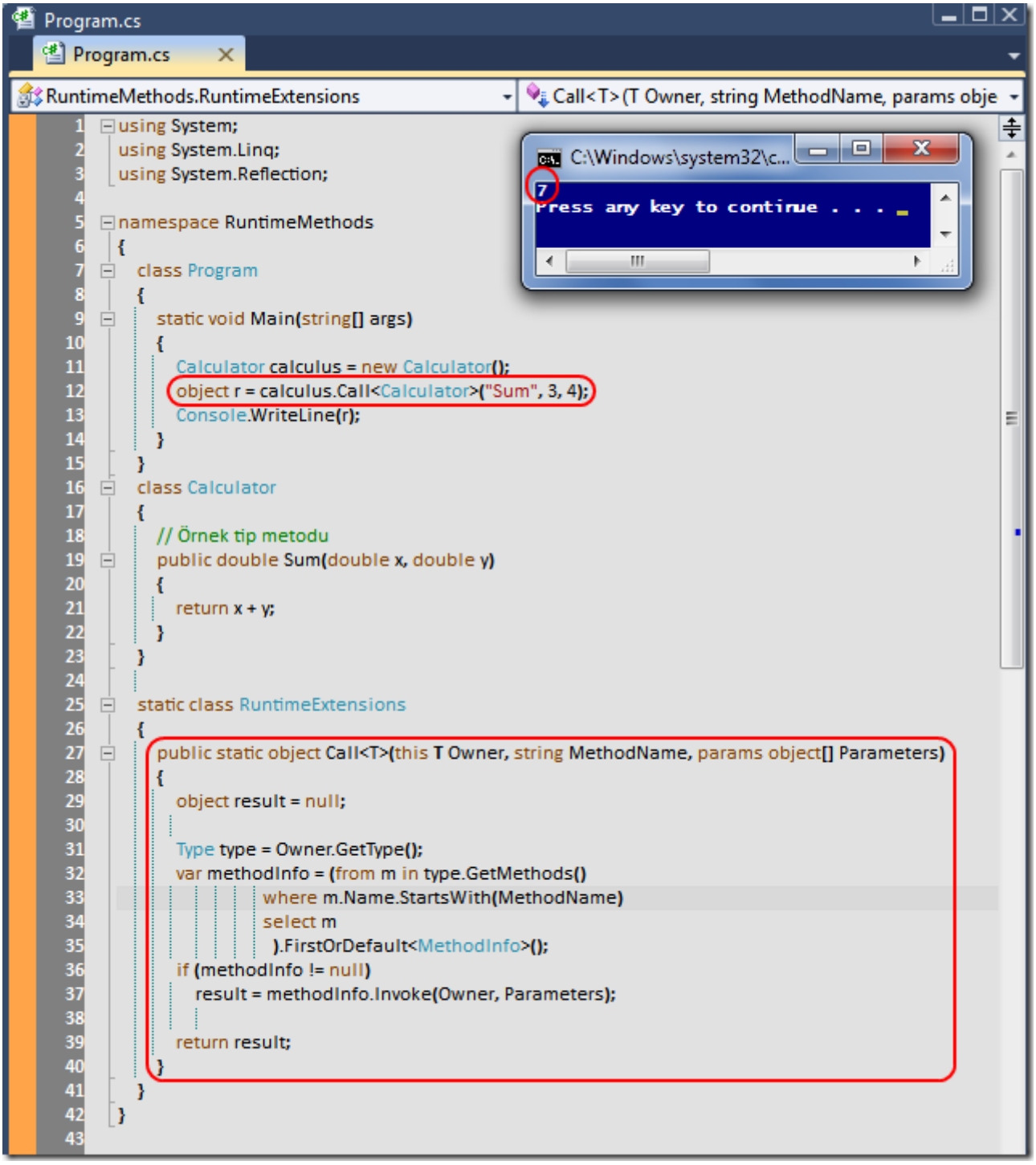
BurakSenyurtVsColorSchema.vssettings (280,59 kb)

Tek Fotoluk İpucu-26 (Runtime Method Çağrımı) (2011-08-19T15:30:00)

c#,c# temelleri,extension methods,reflection,linq,

Merhaba Arkadaşlar,

Sanırım bir önceki tek fotoluk ipucunda çalışma zamanındaki bir nesne özelliğinin değerinin nasıl alınabileceğini görmüştük. Elbette reflection konulu işlerde bir nesne örneğinin bir metodunun çağırılması da söz konusu olabilir. Nasıl mı?



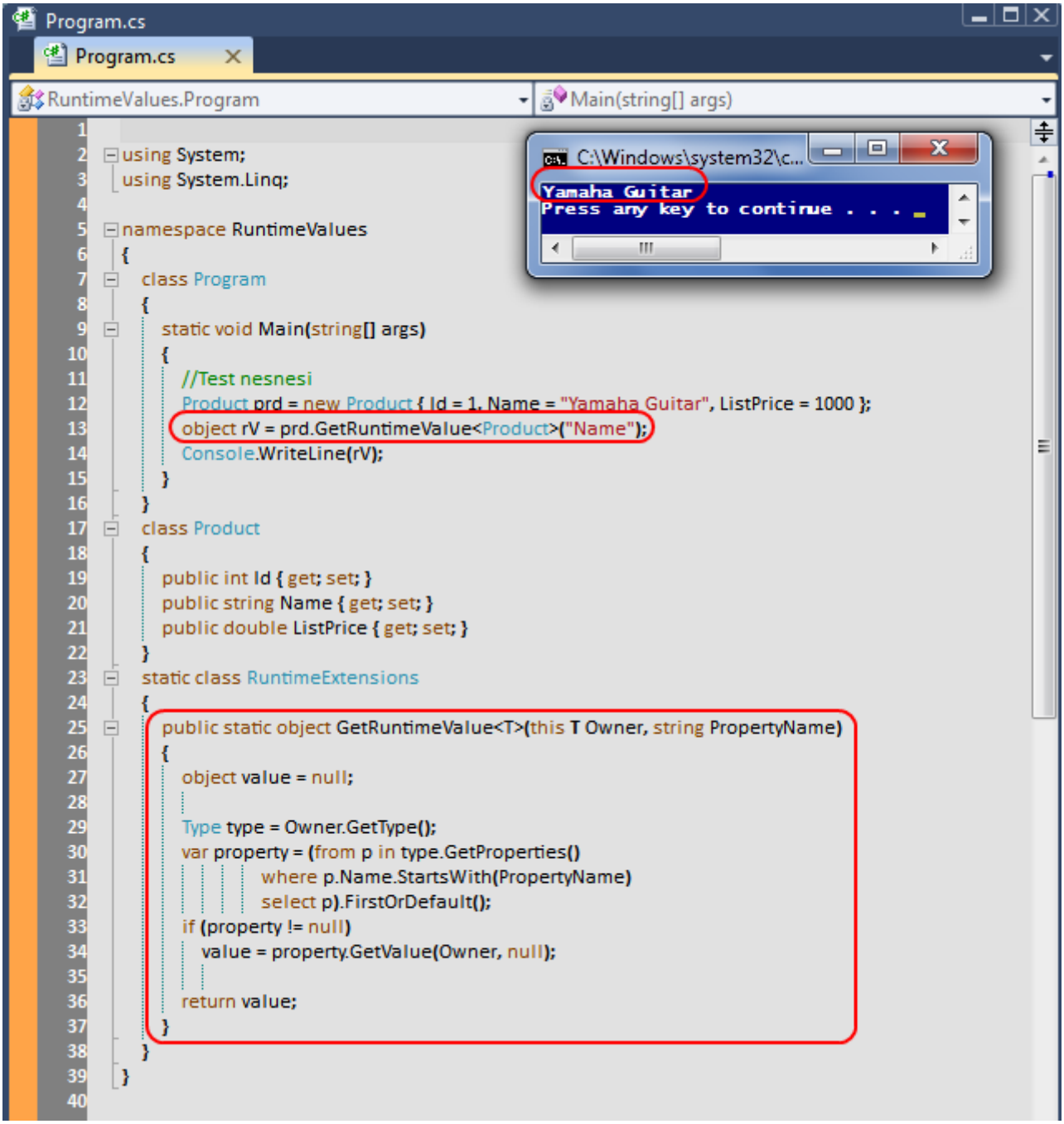
RuntimeMethods.rar (24,17 kb)

Tek Fotoluk İpucu-25 (Runtime Value ve Extension Method) (2011-08-18T15:08:00)

c#,c# temelleri,extension methods,reflection,linq,

Merhaba Arkadaşlar,

özellikle Reflection kullandığımız bazı çalışma zamanı senaryolarında, nesnelerin özellik değerlerini elde etmek istediğimiz durumlar da söz konusu olabilir. çok basit bir senaryo göz önüne alındığında bunun için bir Extension method dahi geliştirebiliriz. Nasıl mı?



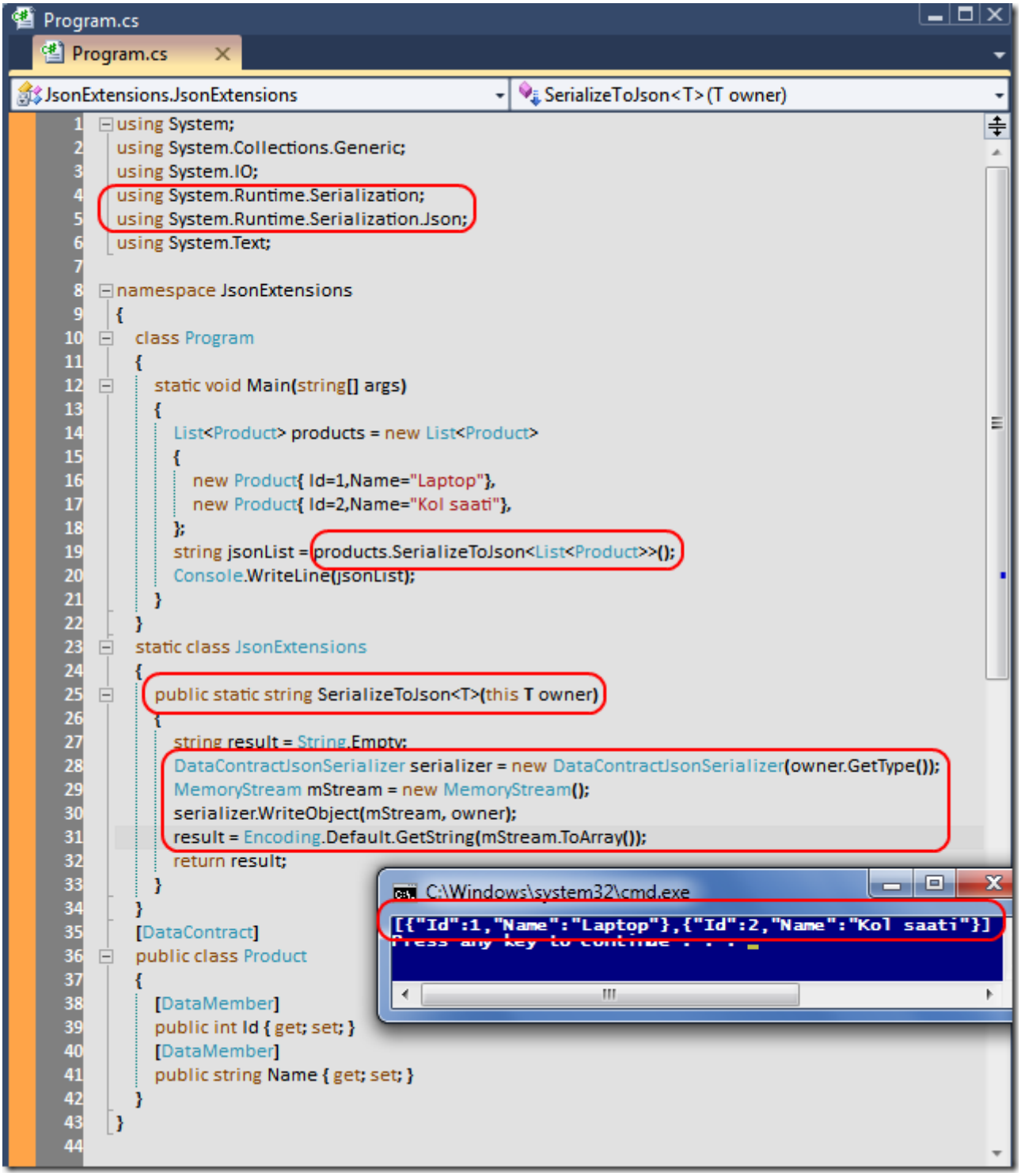
RuntimeValues.rar (23,48 kb)

Tek Fotoluk İpucu-24(DataContractJsonSerializer ve Extension Method) (2011-08-15T09:21:00)

c#,extension methods,json,serialization,

Merhaba Arkadaşlar,

Extension metodlar çok ama çok işimize yarayabiliyor. örneğin serileştirilebilir herhangibir tipin Json formatındaki çıktısının string tipinden döndüren bir extension metodu geliştirebilirsiniz. Nasıl mı? 😉



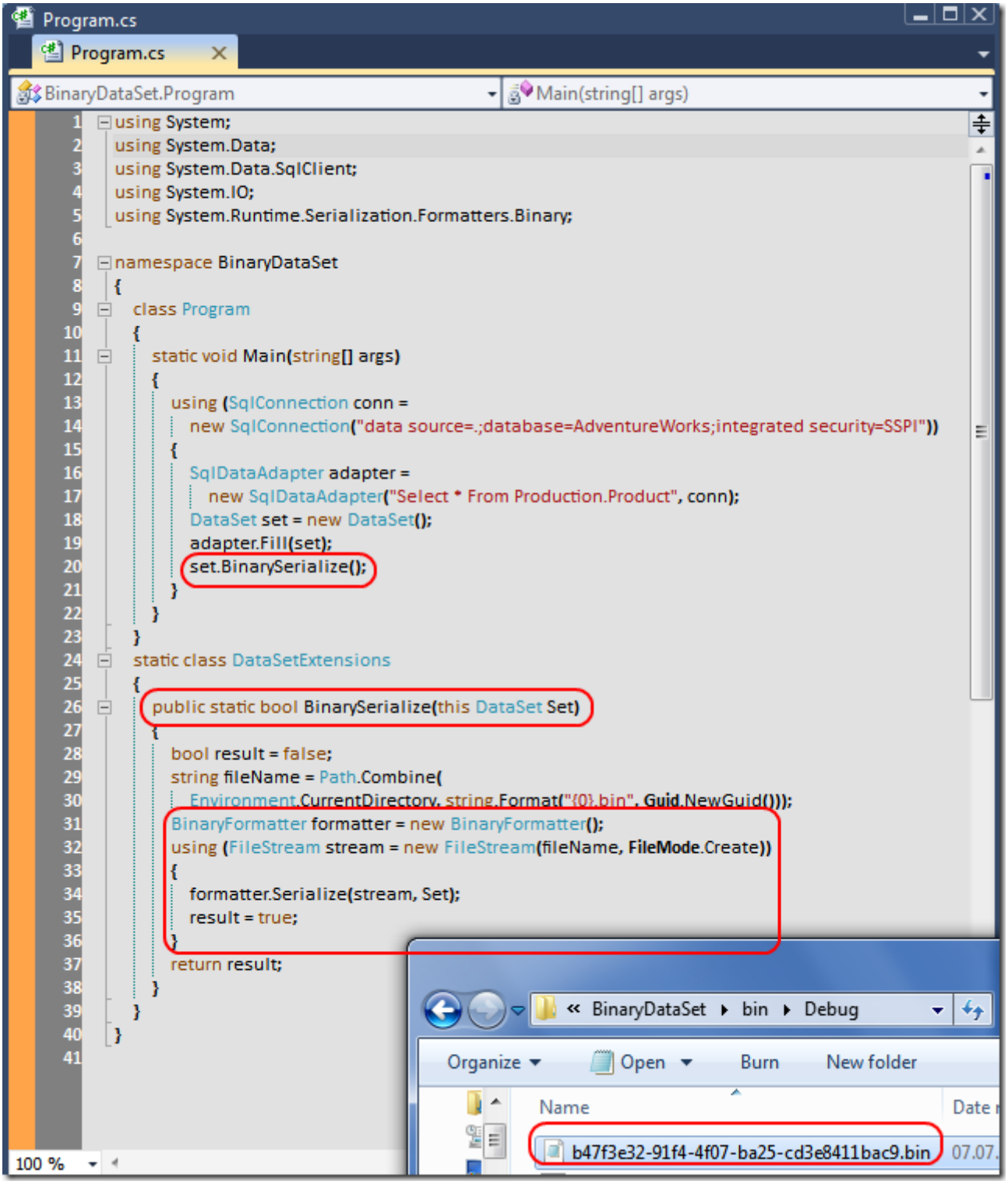
JsonExtensions.rar (23,64 kb)

Tek Fotoluk İpucu-23 (BinaryFormatter, DataSet, Extension Methods) (2011-08-05T09:38:00)

c#,c# temelleri,extension methods,binary serialize,binaryformatter,dataset,

Merhaba Arkadaşlar,

Bu kez elimde bir DataSet, Binary serileştirme için BinaryFormatter ve tabiki Extension Method kabiliyeti var. Ne yapabiliriz? Belki de bir DataSet' in Binary formatta Serialize, DeSerialize işlemlerini üstlenen genişletme metodlarını yazabiliriz. Ben Serialize kısmını yazdım. Gerisi size kalmış 



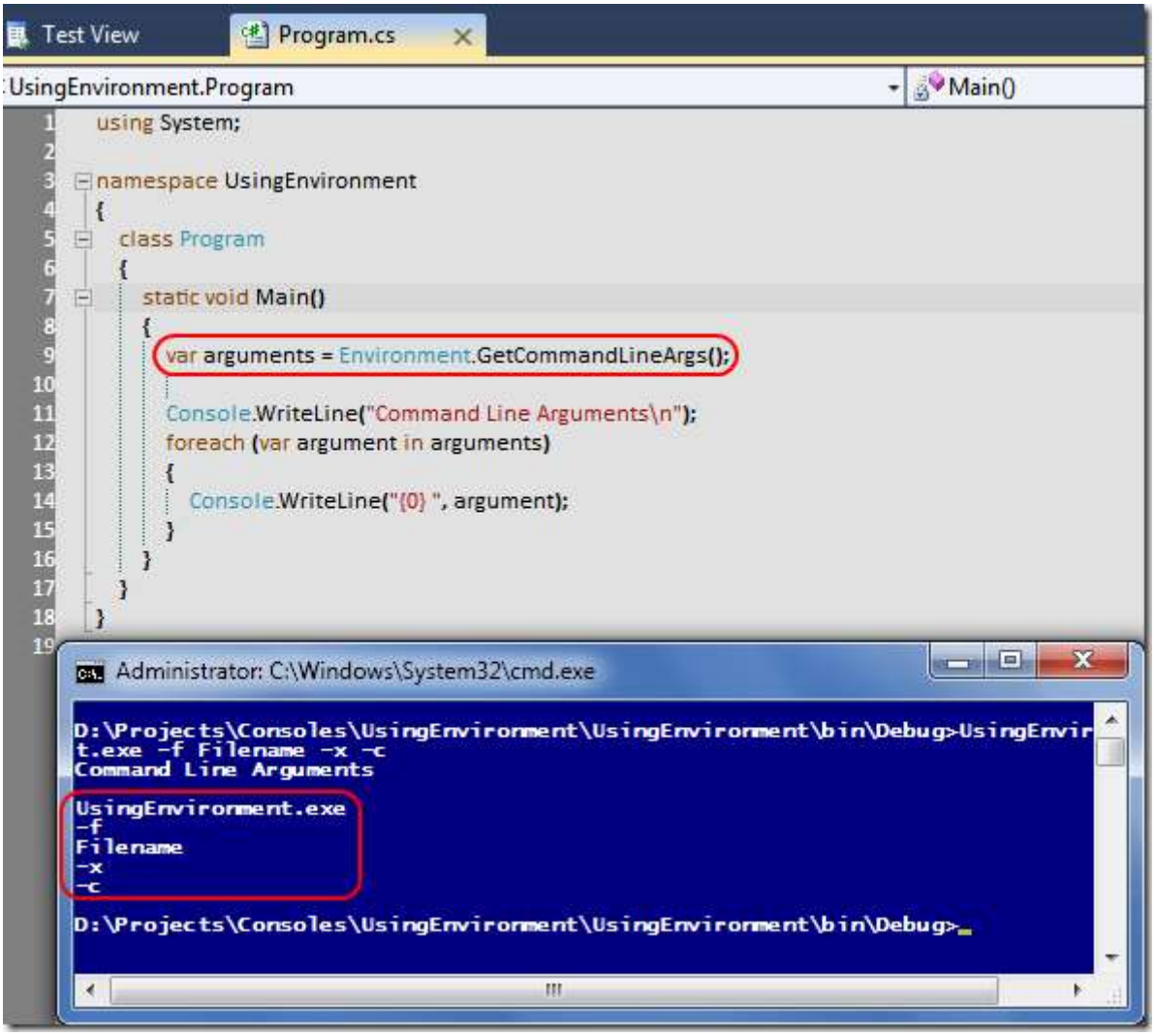
BinaryDataSet.rar (49,42 kb)

Tek Fotoluk İpucu-22 (GetCommandLineArgs) (2011-08-01T09:31:00)

c#,c# temelleri,console,console arguments,environment class,

Merhaba Arkadaşlar,

Zaman zaman komut satırından çalışan Console uygulamaları geliştiririz ve bu programlar genellikle komut satırı parametreleri alarak çalışırlar. çoğunlukla Main metodunun string[] tipinden parametresini kullanırız. Peki Environment tipinin de komut satırı argümanlarını alabilmemiz için bir metod sunduğunu biliyor muydunuz? Environment tipinde neler var neler zaten



UsingEnvironment.rar (20,82 kb)

Bu Aralar Fransızım (2011-07-27T11:43:00)

müzik,fransız,fransızca,axelle red,

Merhaba Arkadaşlar,

Bu aralar fransızca şarkılara merak salmış durumdayım. Sanırım bunda ZAZ grubunun büyük etkisi var. Şu anda

müzik marketlerde en çok satanlar listesinde ilk kez bir fransızın bu kadar iddialı olarak durduğunu görüyorum.

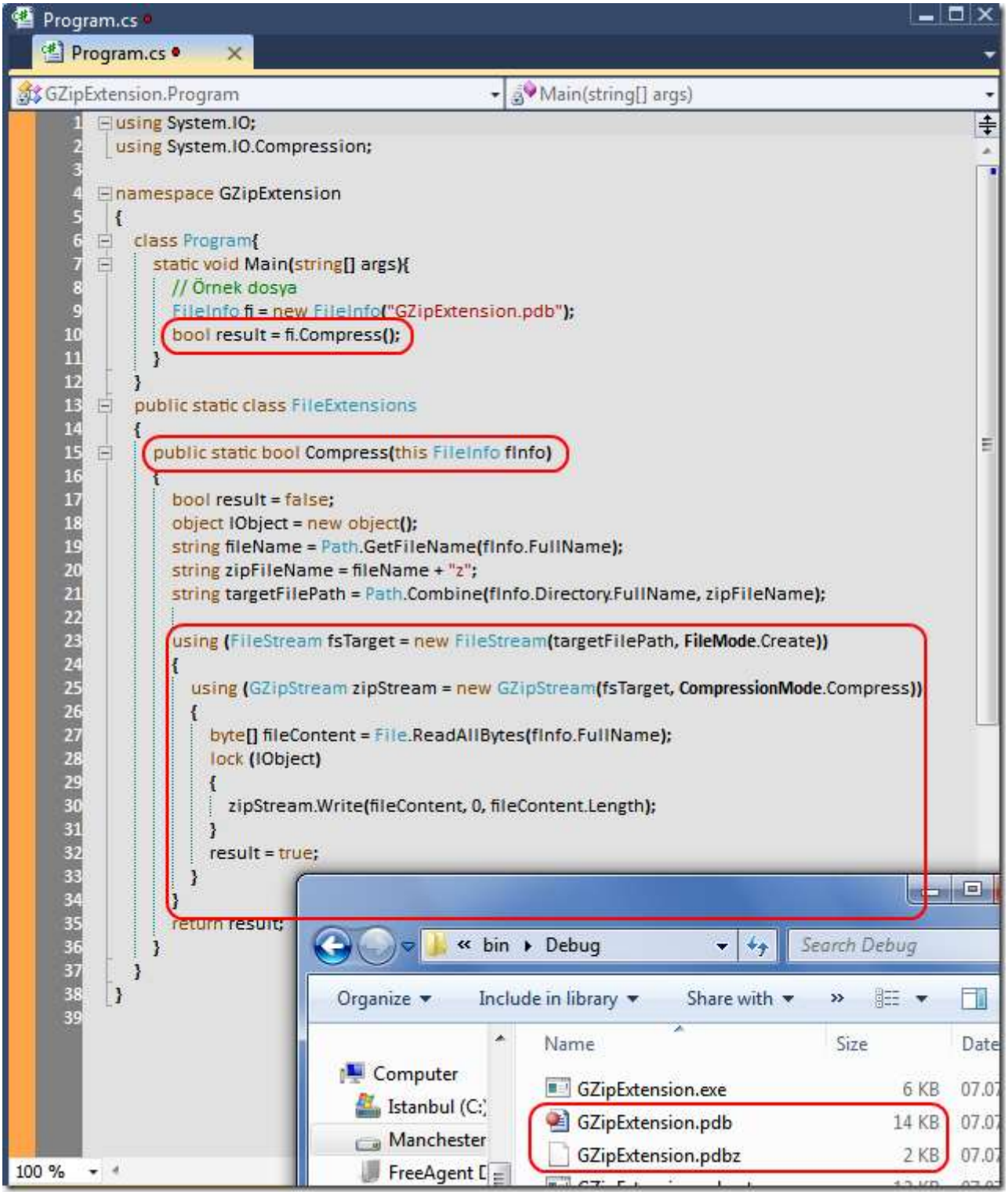
Tabi memleketimin müzik marketlerinden bahsetmekteyim. ZAZ demişken yeni favorim de Axelle Red. Hatta onun grooveshark' tan dinlediğim **Rester Femme** isimli şarkısı gerçekten mükemmel. O ne güzel bir ses tonudur öyle. Diğer parçaları ve albümleri de oldukça güzel. Sanki kadife bir ses tonu var. Dilerseniz aşağıdaki widget üzerinden Rester Femme şarkısını dinleyebilirsiniz.

Tek Fotoluk İpucu-21(FileInfo,GZipStream ve Extension) (2011-07-20T17:28:00)

c#,c# temelleri,.net framework,gzipstream,compress,extension methods,fileinfo,io,

Merhaba Arkadaşlar,

Elimde FileInfo, GZipStream tipleri ve Extension Method kabiliyeti var. Sizce ne yapılabilir? Yoksa FileInfo tipinde Compress ve DeCompress işlemleri için birer Extension Method' mu yazılabilir? Ben Compress' I yazdım. Kalanı da size ait olsun



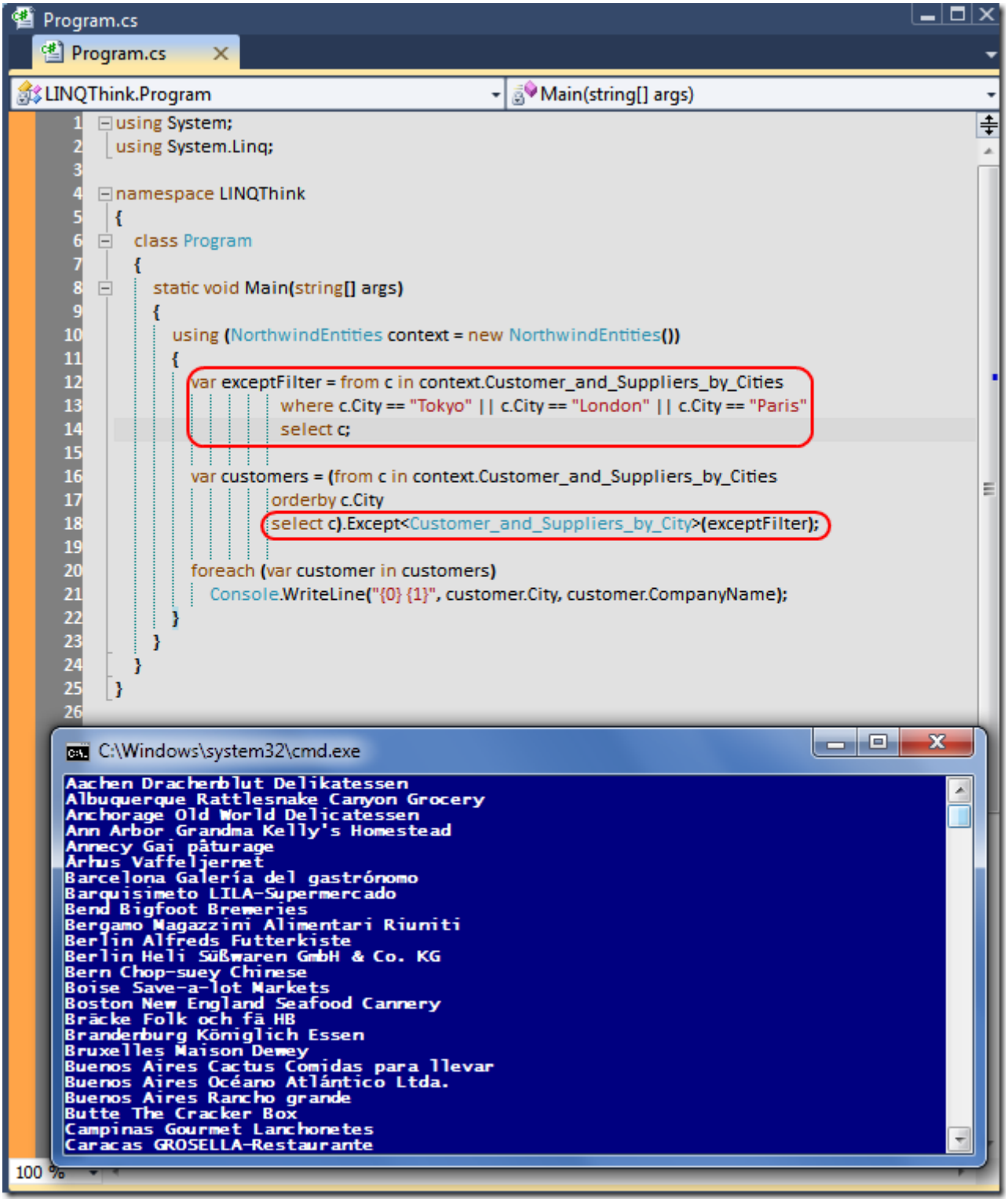
GZipExtension.rar (24,43 kb)

Tek Fotoluk İpucu-20 (Except Sorgusu) (2011-07-19T12:17:00)

c#,linq,c# temelleri,

Merhaba Arkadaşlar,

Hemen hepimiz LINQ sorgularını kullanıyoruz(Tabi aramızda halen .Net 2.0 ve altı ile çalışan zavallılar da yok değil) Lakin LINQ içerisinde çok enteresan extension method' lar olduğunu da biliyor muyuz? örneğin, şehir bazındaki müşteri listesini veren bir View' un LINQ sorgusunda, belirli şehirlerin dışında kalanların kümesini almak isteyebiliriz. Aşağıdaki örnekte olduğu gibi



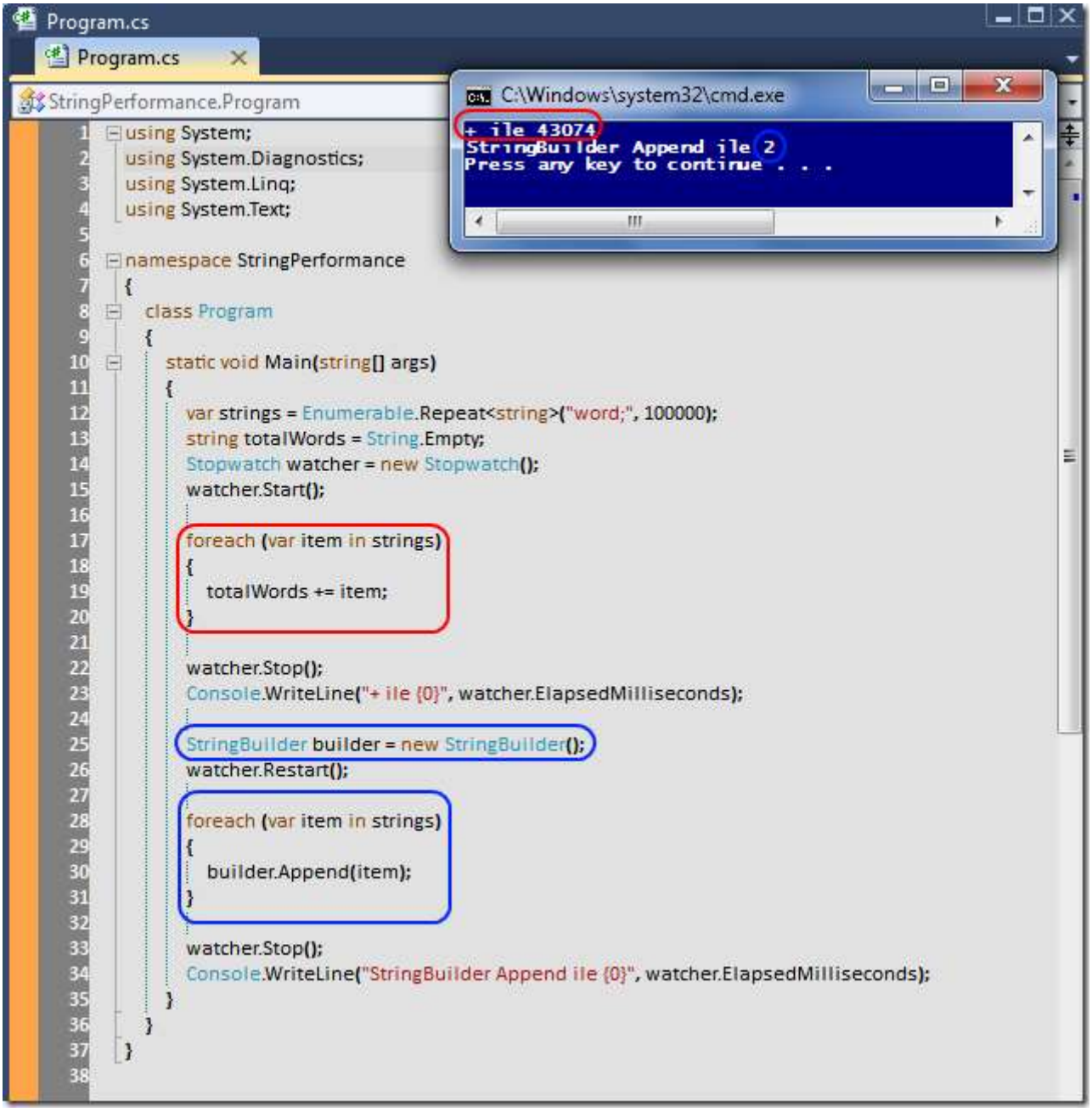
LINQThink.rar (38,50 kb)

Tek Fotoluk İpucu-19 (StringBuilder deyip geçme) (2011-07-18T09:51:00)

c#,c# temelleri,stringbuilder,string,

Merhaba Arkadaşlar,

String tipleri çok garip tiplerdir. Onları + operatörü ile birleştirmek bazen akıl karı değildir. çok fazla performans kaybettirir. Bir de uzlaşma yoluna gidebileceğini StringBuilder var. örneğin



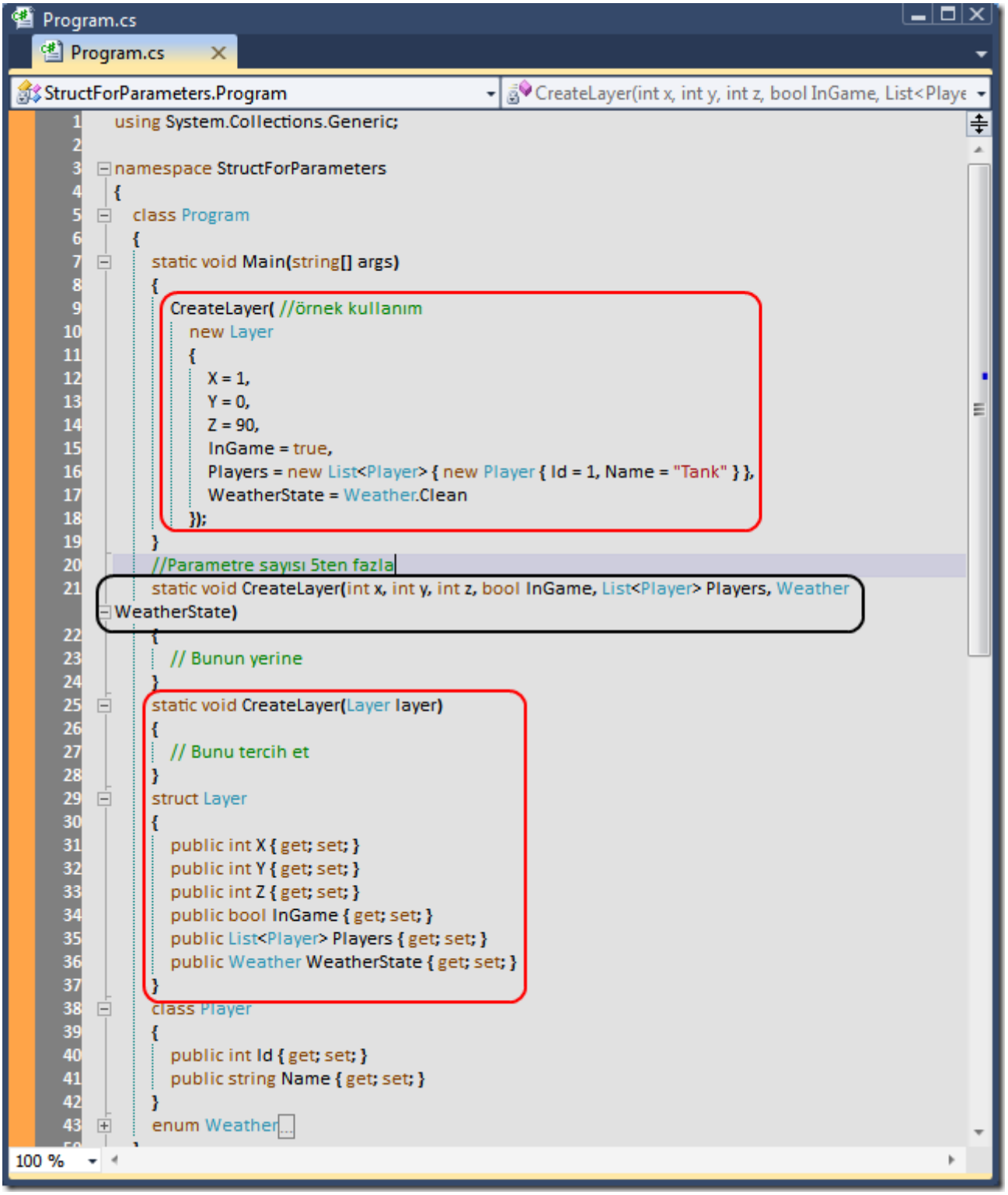
StringPerformance.rar (22,11 kb)

Tek Fotoluk İpucu-18 (5 Parametreden Fazlası için Struct) (2011-07-13T12:48:00)

c# temelleri,struct,parameters,juval lowy,code standarts,

Merhaba Arkadaşlar,

çok sevgili Juval Löwy der ki : "Bir metod 5den fazla parametre alıyorsa, verileri Struct tipini kullanarak aktarın". Meşhur kod standartlarından birisi olan bu kurala kaçımız ne kadar uyuyoruz acaba? Oysaki kullanımı çok basit. İşte basit bir örnek



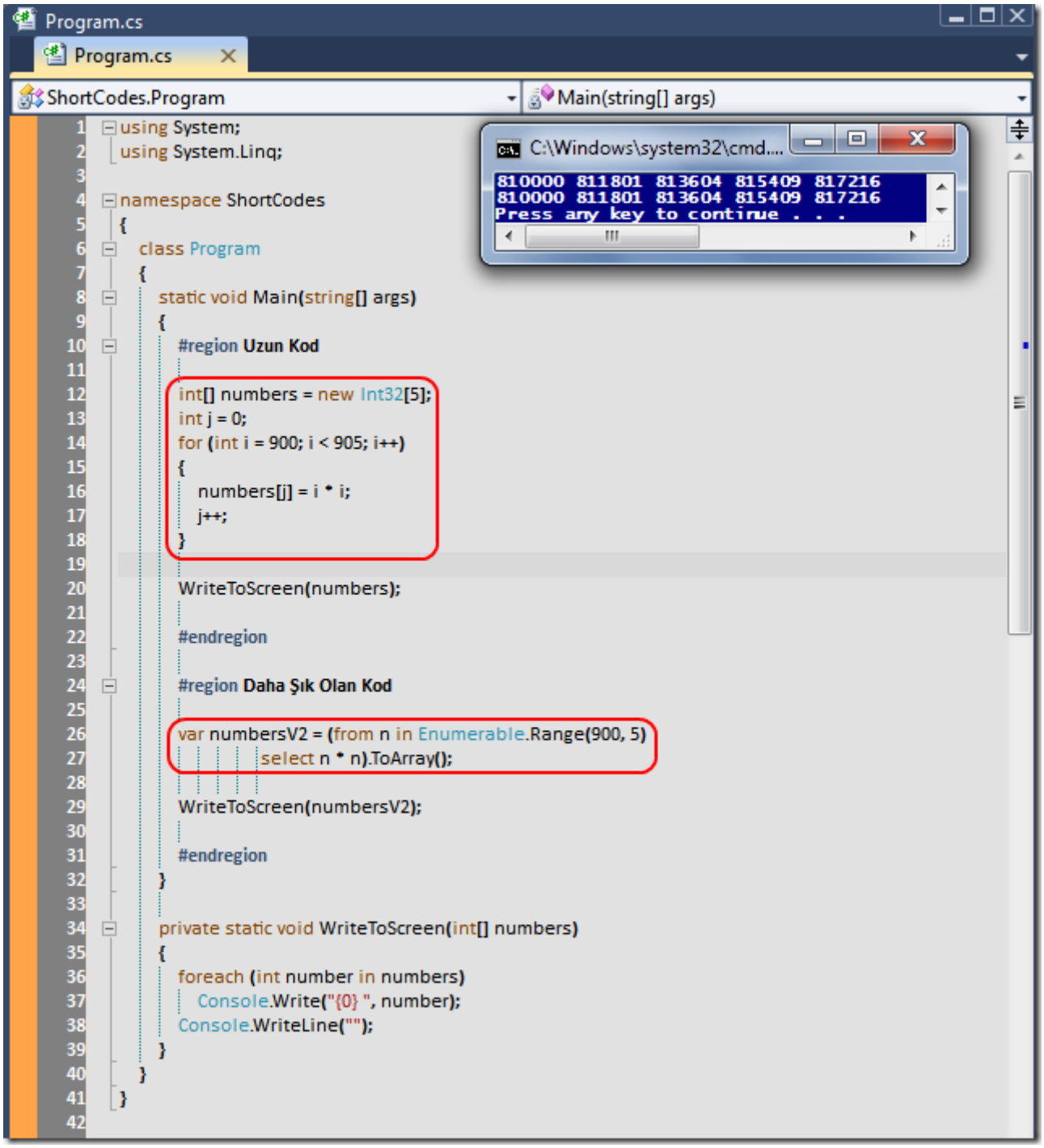
StructForParameters.rar (22,91 kb)

Tek Fotoluk İpucu-17 (Query ile Daha Şık Kodlama) (2011-07-12T09:54:00)

c#,linq,enumerable,

Merhaba Arkadaşlar,

LINQ sorgularını sadece sorgulamak için kullandığımızı da nereden çıkartıyorsunuz Aslında onları kodlarımızı daha şık hale getirmek için de kullanabiliriz? Nasıl mı? İşte küçük bir örnek



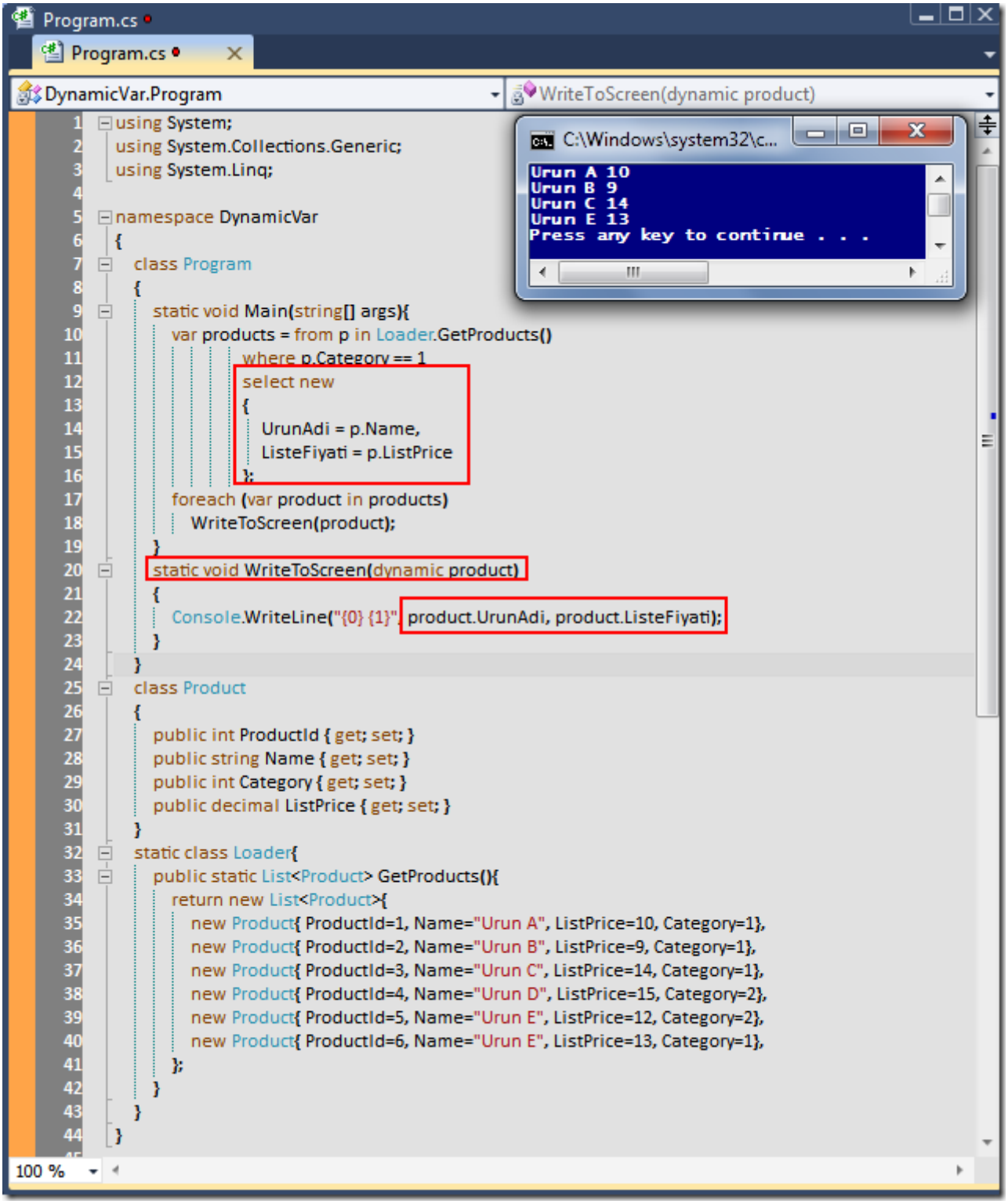
ShortCodes.rar (22,69 kb)

Tek Fotoluk İpucu-16 (Dynamic Var) (2011-07-11T09:41:00)

c#,c# 3.0,c# 4.0,c# temelleri,dynamic,var,dynamic language runtime,

Merhaba Arkadaşlar,

LINQ tarafında isimsiz tipleri(Anonymous Types) oldukça sık kullanmaktayız. Ancak isimsiz tiplerin metodlara parametre olarak geçirilemediğini de biliyoruz çünkü bu tipler derleyici tarafından üretiliyorlar. Ama üzülmeyin. çünkü elimizde 4.0 ile gelen dynamic anahtar kelimesi var. Peki nasıl kullanırız?




```
using System;
using System.Collections.Generic;
using System.Linq;

namespace DynamicVar
{
    class Program
    {
        static void Main(string[] args){
            var products = from p in Loader.GetProducts()
                           where p.Category == 1
                           select new
                           {
                               UrunAdi = p.Name,
                               ListeFiyati = p.ListPrice
                           };
            foreach (var product in products)
                WriteToScreen(product);
        }
        static void WriteToScreen(dynamic product)
        {
            Console.WriteLine("{0} {1}", product.UrunAdi, product.ListeFiyati);
        }
    }
    class Product
    {
        public int ProductId { get; set; }
        public string Name { get; set; }
        public int Category { get; set; }
        public decimal ListPrice { get; set; }
    }
    static class Loader{
        public static List<Product> GetProducts(){
            return new List<Product>{
                new Product{ ProductId=1, Name="Urun A", ListPrice=10, Category=1},
                new Product{ ProductId=2, Name="Urun B", ListPrice=9, Category=1},
                new Product{ ProductId=3, Name="Urun C", ListPrice=14, Category=1},
                new Product{ ProductId=4, Name="Urun D", ListPrice=15, Category=2},
                new Product{ ProductId=5, Name="Urun E", ListPrice=12, Category=2},
                new Product{ ProductId=6, Name="Urun E", ListPrice=13, Category=1},
            };
        }
    }
}
```

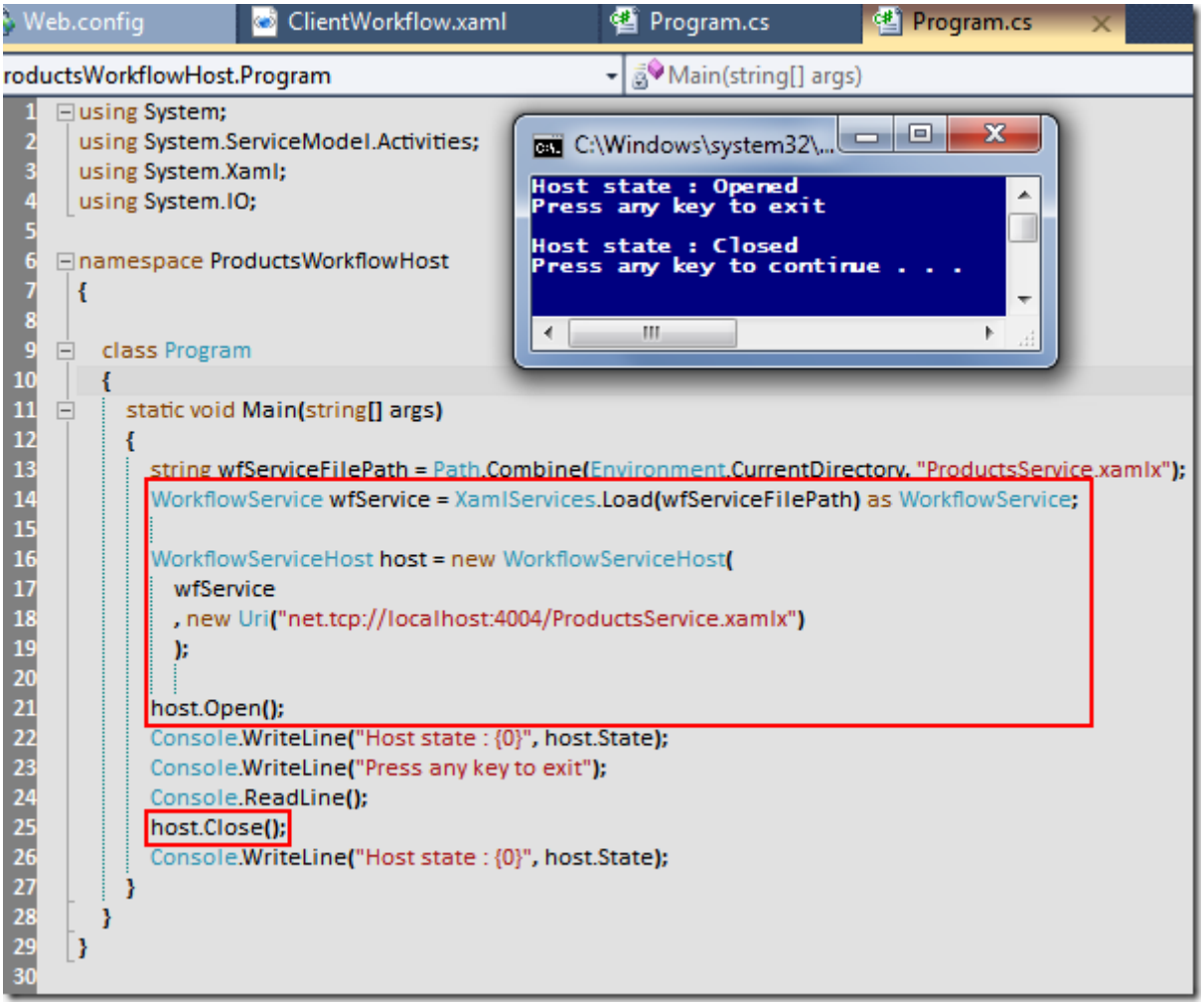
DynamicVar.rar (26,49 kb)

Tek Fotoluk İpucu - 15(Self Hosted Workflow Service) (2011-07-07T17:05:00)

wcf,wf,workflow services,windows workflow foundation,windows communication foundation,

Merhaba Arkadaşlar,

Elinizde bir Workflow Service kütüphanesi ve XAMLX uzantılı Workflow Service dosyaları var. Bu dosyalardan yararlanarak kendi Workflow Service Host uygulamanızı yazmak niyetindesiniz. Diyelim ki bu uygulama bir Console projesi olacak. Nasıl yaparsınız? İşte böyle



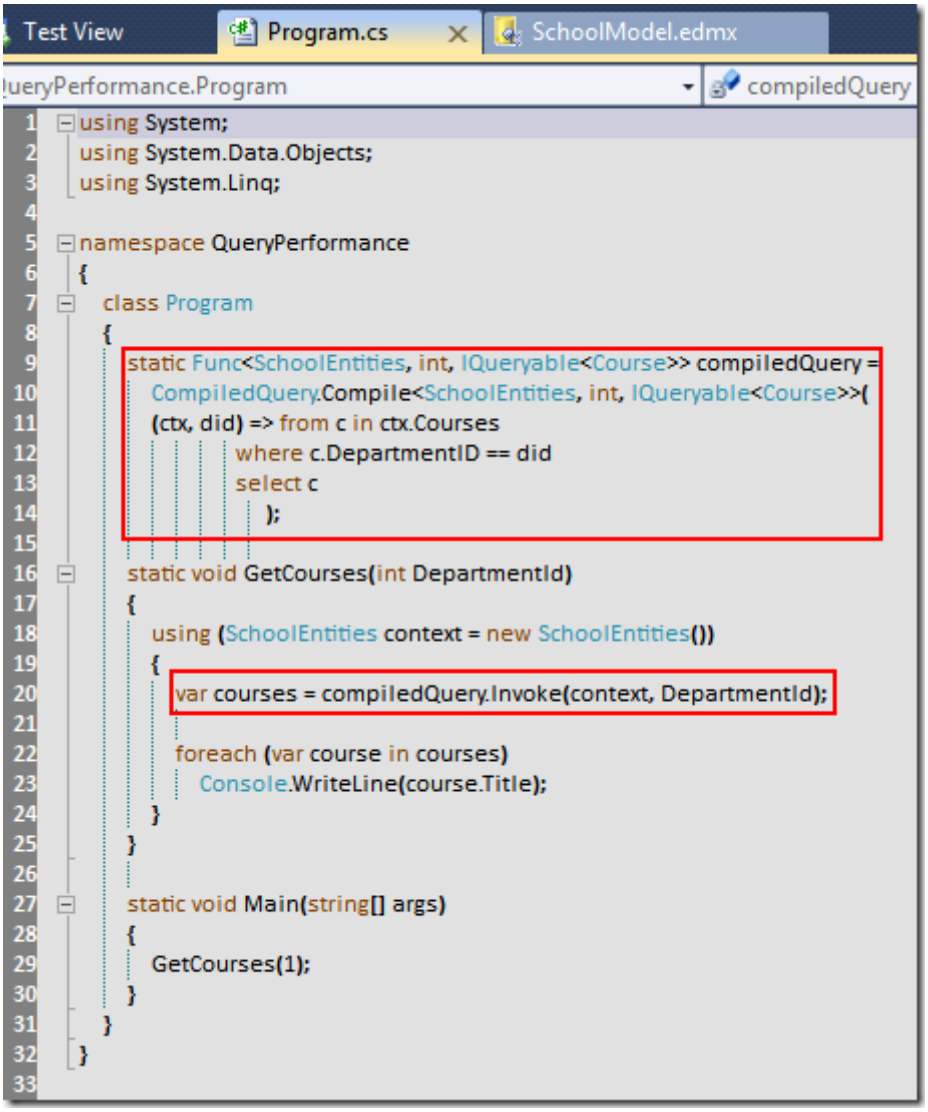
ProductsWorkflowHost.rar (38,48 kb)

Tek Fotoluk İpucu-14 (Compiled Query) (2011-07-06T15:30:00)

entity framework,linq,compiled query,c#,

Merhaba Arkadaşlar,

Entity Framework ile Compiled Query' ler hazırlayabileceğinizi ve daha performanslı sorgulamalar yaptırabileceğinizi biliyor muydunuz? İşte basit bir örnek



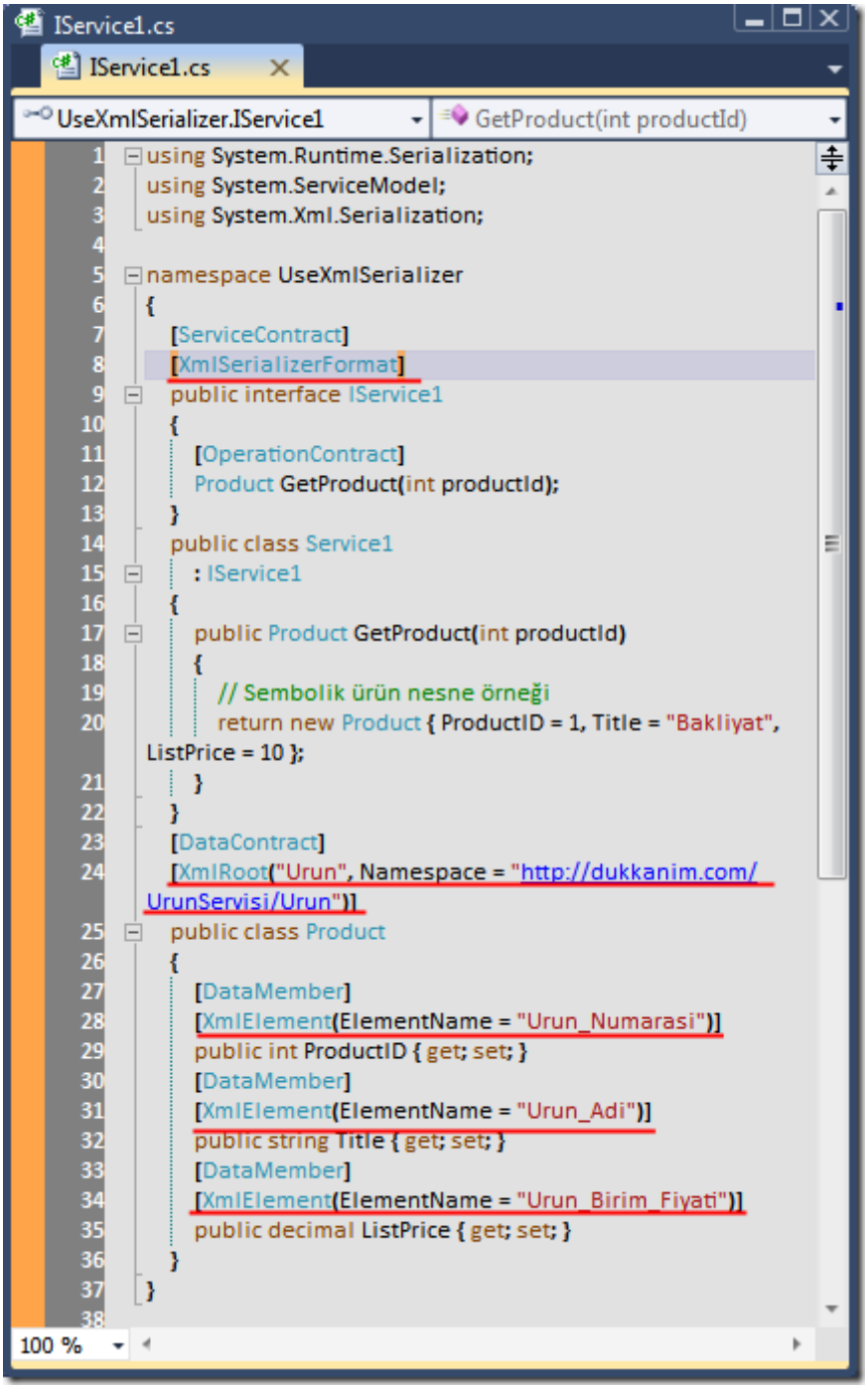
QueryPerformance.rar (75,31 kb)

[Tek Fotoluk İpucu-13(XmlSerializer ile Daha Fazla Kontrol) (2011-07-05T11:02:00)]

wcf,wcf 4.0,xml,soap,xml serialization,xml serializer,windows communication foundation,

Merhaba Arkadaşlar,

Bazen SOAP Bazlı WCF servisimizdeki veri türlerinin, .Net tabanlı olmayan platformlarda yer alan istemci veya servislerle daha kolay anlaşabilmesini sağlamak isteyebiliriz. özellikle bu noktada XmlSerializer işimizi kolaylaştırabilir. Nasıl mı?



UseXmlSerializer.rar (18,30 kb)

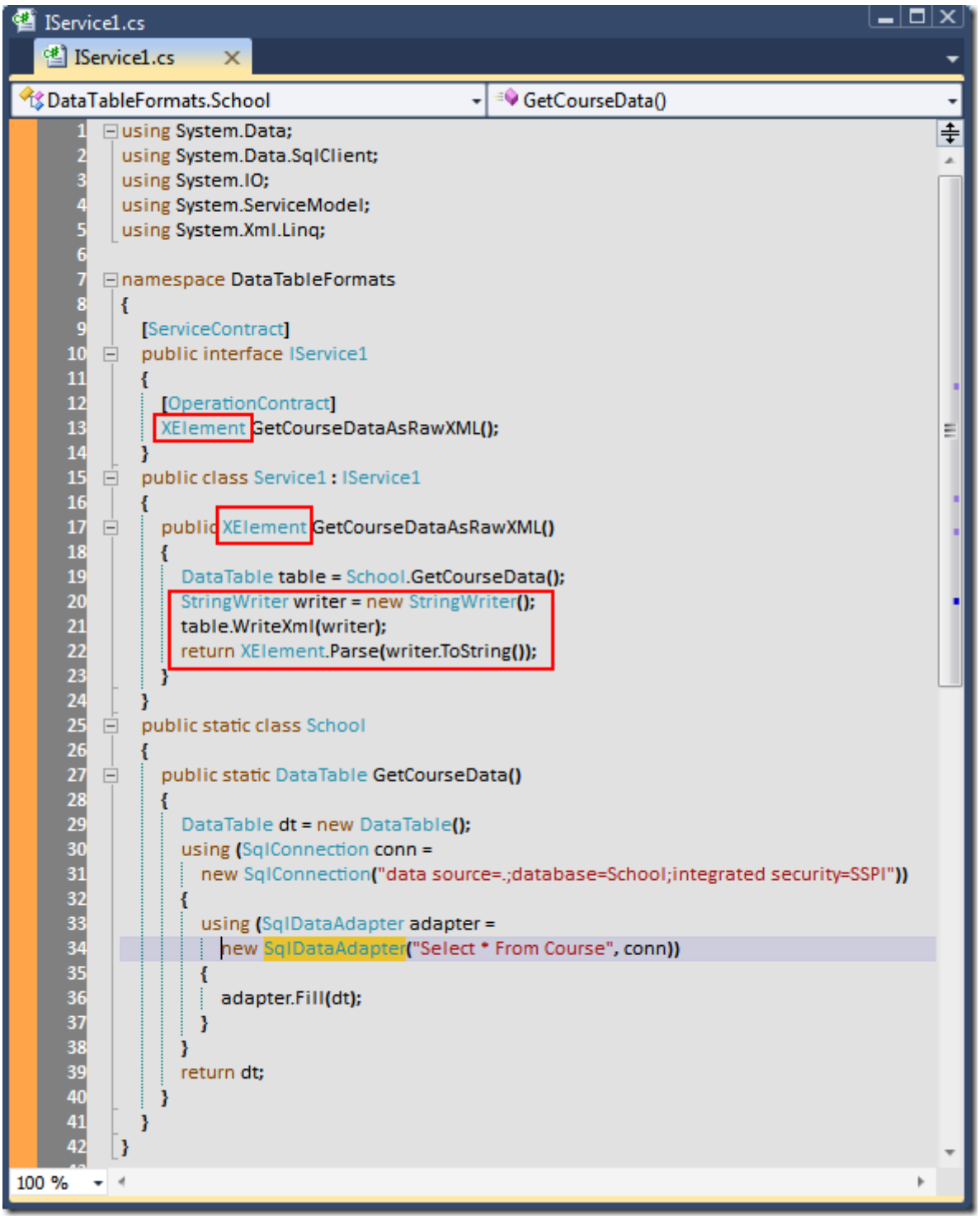
Tek Fotoluk İpucu-12 (DataTable için Raw XML Formatı) (2011-07-04T09:45:00)

wcf,datatable,ado.net,windows communication foundation,

Merhaba Arkadaşlar,

Peki elinizde bir DataTable var ve siz bunun Raw XML formatındaki çıktısını istemcilere vermek istiyorsunuz. Ne yaparsınız?

Not : WcfTestClient istemcisine güvenmeyin. XElement tipinin geriye döndürelemeyeceğini söyleyerek örneği test etmenize izin vermez. Bu sizi yanıtlmasın.



DataTableFormats.rar (18,32 kb)

Tek Fotoluk İpucu - 11 (Ham XML ve XElement) (2011-07-01T23:30:00)

wcf,xlinq,xelement,windows communication foundation,

Merhaba Arkadaşlar,

Hani olurda yazdığınız WCF servislerini .Net istemcilerine açarken XML olarak döndürdüğünüz içeriklari RAW formatlarında sunmak istersiniz. Bu durumda yapacağınız iş çok basittir. İşte örnek

Not : WcfTestClient istemcisine güvenmeyin. XElement tipinin geriye döndürelemeyeceğini söyleyerek örneği test etmenize izin vermez. Bu sizi yanıtlmasın.

```
using System.ServiceModel;
using System.Xml.Linq;

namespace WcfServiceLibrary1
{
    [ServiceContract]
    public interface IService1
    {
        [OperationContract]
        XElement GetLibrary();
    }

    public class Service1
        : IService1
    {
        public XElement GetLibrary()
        {
            string xmlData = @"<Library>
                                 <Book>
                                   <Name>Cloud Computing</Name>
                                   <Category>Computer</Category>
                                 </Book>
                                 <Book>
                                   <Name>Red Octeber</Name>
                                   <Category>Novel</Category>
                                 </Book>
                               </Library>";
            return XElement.Parse(xmlData);
        }
    }
}
```

Örnek XML içeriğidir. Benze XML içerğini istediğiniz kaynaktan yükleyebilirsiniz.

WcfServiceLibrary1.rar (50,01 kb)

Tek Fotoluk İpucu - 10 (MessageContract yardımıyla SoapHeader' a Bilgi Eklemek) (2011-06-30T09:39:00)

wcf,message contract,soap,windows communication foundation,

Merhaba Arkadaşlar,

Bu kez de WCF ile ilişkili bir fotoğraf paylaşalım istedim. Aslına bakarsanız iki fotoğrafçık oldu ama idare edin artık. Varsayalım ki SOAP paketlerinizin Header kısmında kendi

tanımladığınız tip içeriklerinin yer almasını istiyorsunuz. İşte bunun için aşağıdaki fotoğrafta görülen yolu izleyebilirsiniz

IService1.cs

IService1.cs

WcfService2.ResponseMessage | Content

```
1  using System.Runtime.Serialization;
2  using System.ServiceModel;
3
4  namespace WcfService2
5  {
6      [ServiceContract]
7      public interface IService1
8      {
9          [OperationContract]
10         ResponseMessage Execute(RequestMessage Message);
11     }
12     public class Service1 : IService1
13     {
14         public ResponseMessage Execute(RequestMessage Message)
15         {
16             return new ResponseMessage { Content = string.Format("{0} dan talep geldi ",
   Message.UserInformation.EMailAddress) };
17         }
18     }
19     [MessageContract]
20     public class RequestMessage
21     {
22         [MessageHeader]
23         public UserInfo UserInformation { get; set; }
24         [MessageBodyMember]
25         public string Content { get; set; }
26     }
27     [MessageContract]
28     public class ResponseMessage
29     {
30         [MessageBodyMember]
31         public string Content { get; set; }
32     }
33     [DataContract]
34     public class UserInfo
35     {
36         [DataMember]
37         public string EMailAddress { get; set; }
38         [DataMember]
39         public string UserLevel { get; set; }
40     }
41 }
42
```

100 %

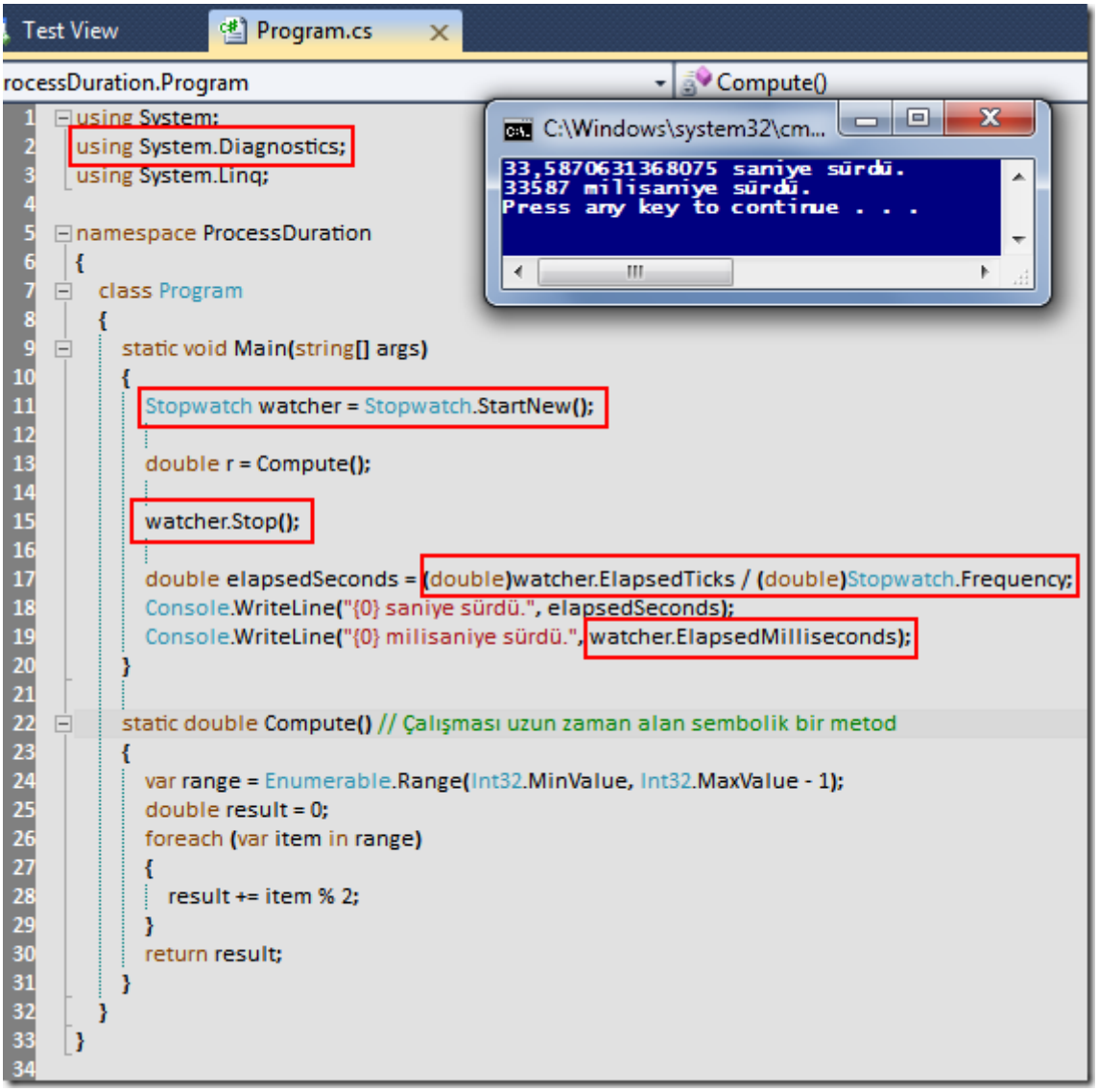
WcfService2.rar (16,58 kb)

Tek Fotoluk İpucu - 9 (Stopwatch ile süre ölçümü) (2011-06-29T09:58:00)

c#,.net framework,.net framework 4.0,stopwatch,

Merhaba Arkadaşlar,

Bazen yazdığımız kod parçalarının işlem sürelerini hesaplama ihtiyacı duyarız. Bu anlamda en çok kullanılan yöntemlerden birisi DateTime ve TimeSpan tiplerini ele almakta iken gerçekte en efektif olanı Stopwatch sınıfını değerlendirmektir. Nasıl mı?



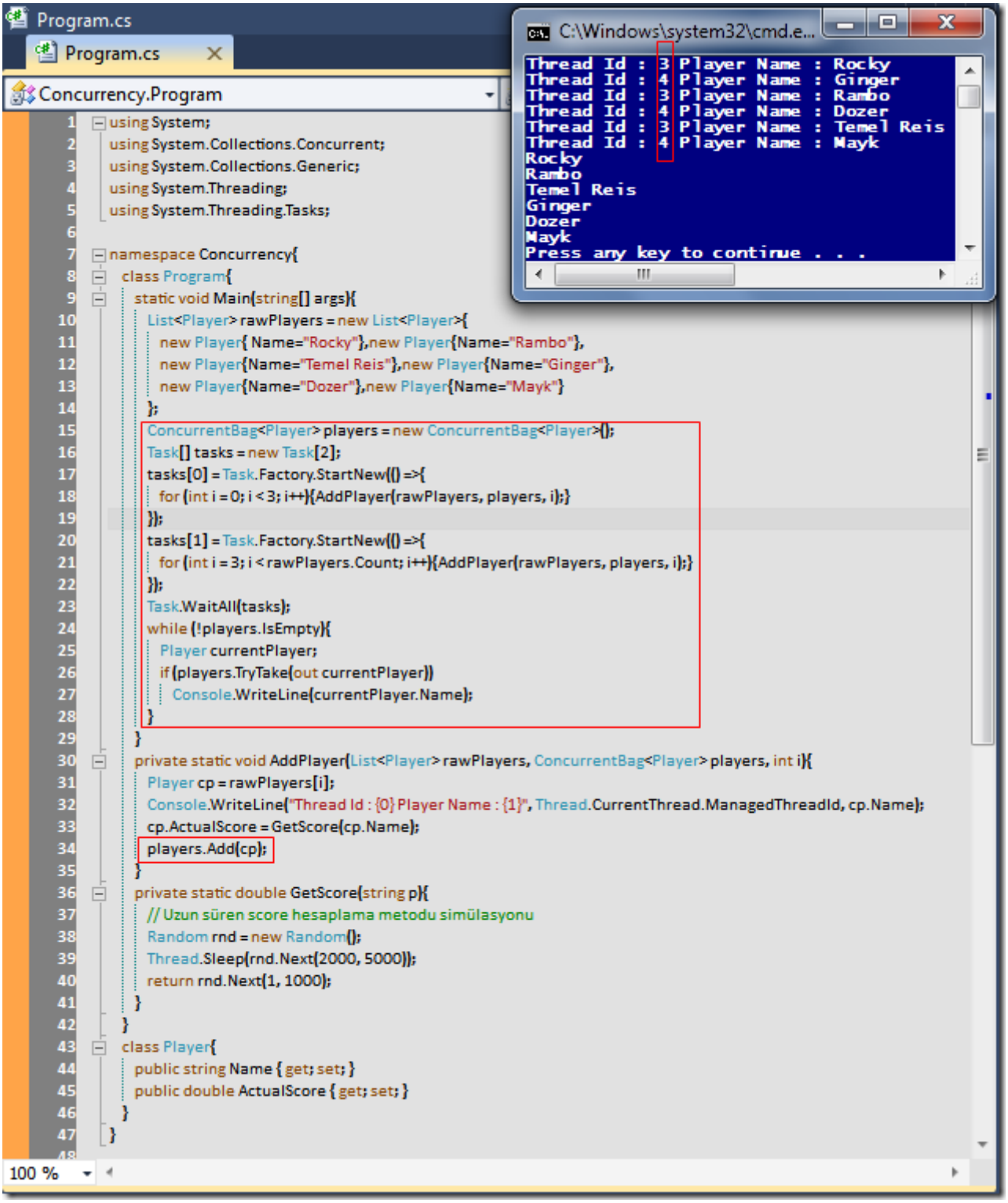
ProcessDuration.rar (22,01 kb)

Tek Fotoluk İpucu - 8 (Parallel ConcurrentBag) (2011-06-28T08:00:00)

c#,parallel programming,concurrent collections,concurrentbag,thread safe,

Merhaba Arkadaşlar,

Concurrent Collections deyince aklımıza Thread-Safe koleksiyon tipleri gelmelidir. Söz gelimi bir ConcurrentBag koleksiyonunun basit kullanımına bir örneği aşağıdaki gibi verebiliriz.



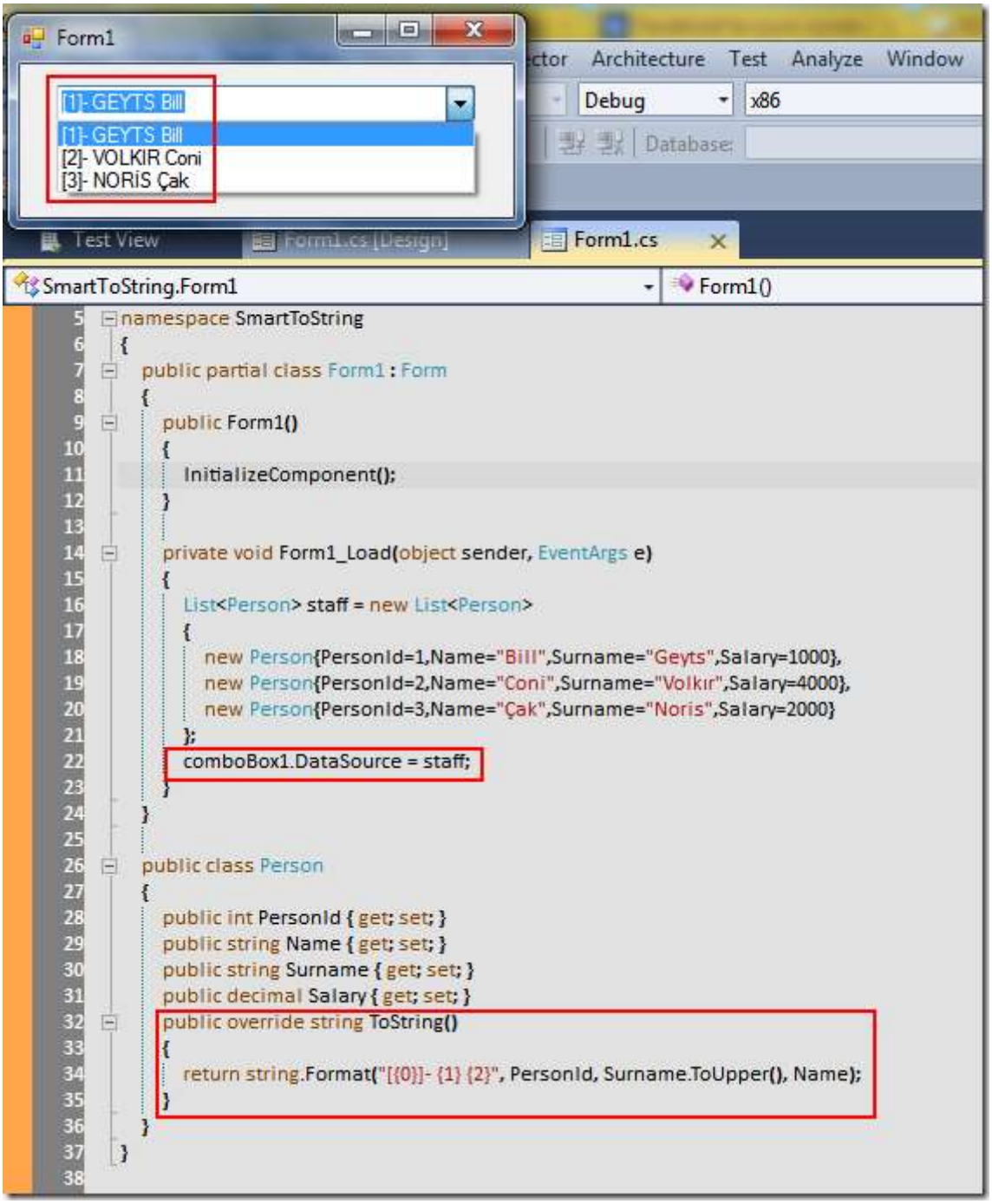
Concurrency.rar (23,94 kb)

Tek Fotoluk İpucu - 7 (Windows Liste Bazlı Kontrolleri ve ToString Metodu) (2011-06-27T12:48:00)

c#,c# temelleri,window,winforms,list,generic list,overriding,

Merhaba Arkadaşlar,

WinForms programcılığında sık rastlanan sorunlardan birisi de, kendi özel tiplerimizi liste bazlı kontrollere bağladığımız durumlarda ortaya çıkmaktadır. Acaba liste bazlı kontrolün içeriğini kendimiz nasıl belirleyebiliriz?



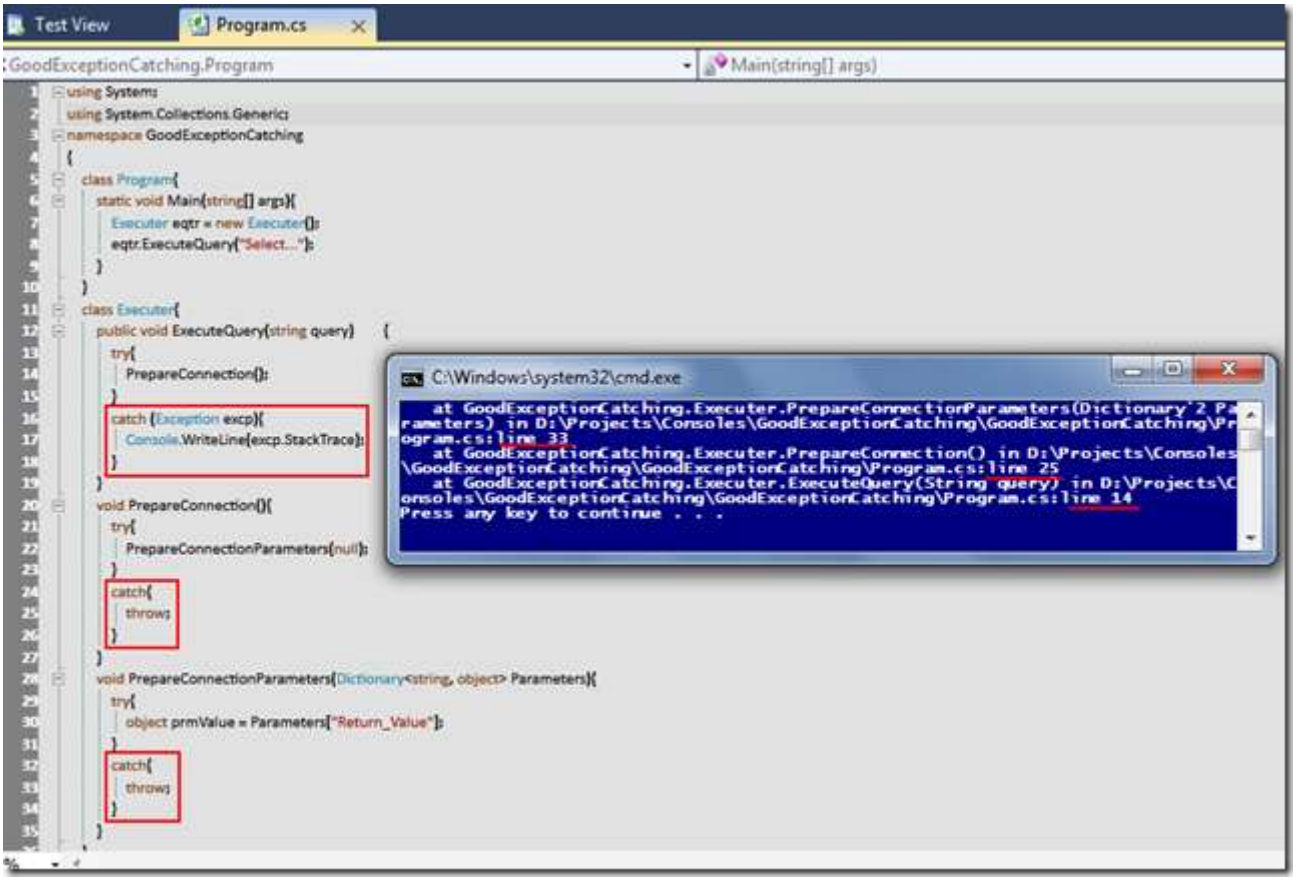
SmartToString.rar (36,38 kb)

Tek Fotoluk İpucu - 6 (Fluent Exception Handling) (2011-06-24T18:22:00)

c#,.net framework,exception handling,

Merhaba Arkadaşlar,

Bazen iç içe çağrılarda bulunan metod zincirlerinde herhangibir seviyede meydana gelen Exception durumunu, en üst noktada yakalamak isteriz. Bu durumda balon köpüğü misali bir aşağıdan yukarı yükselen bir mekanizmayı kullanabiliriz. Nasıl mı?



(Büyük hali için fotoğrafa tıklayın)

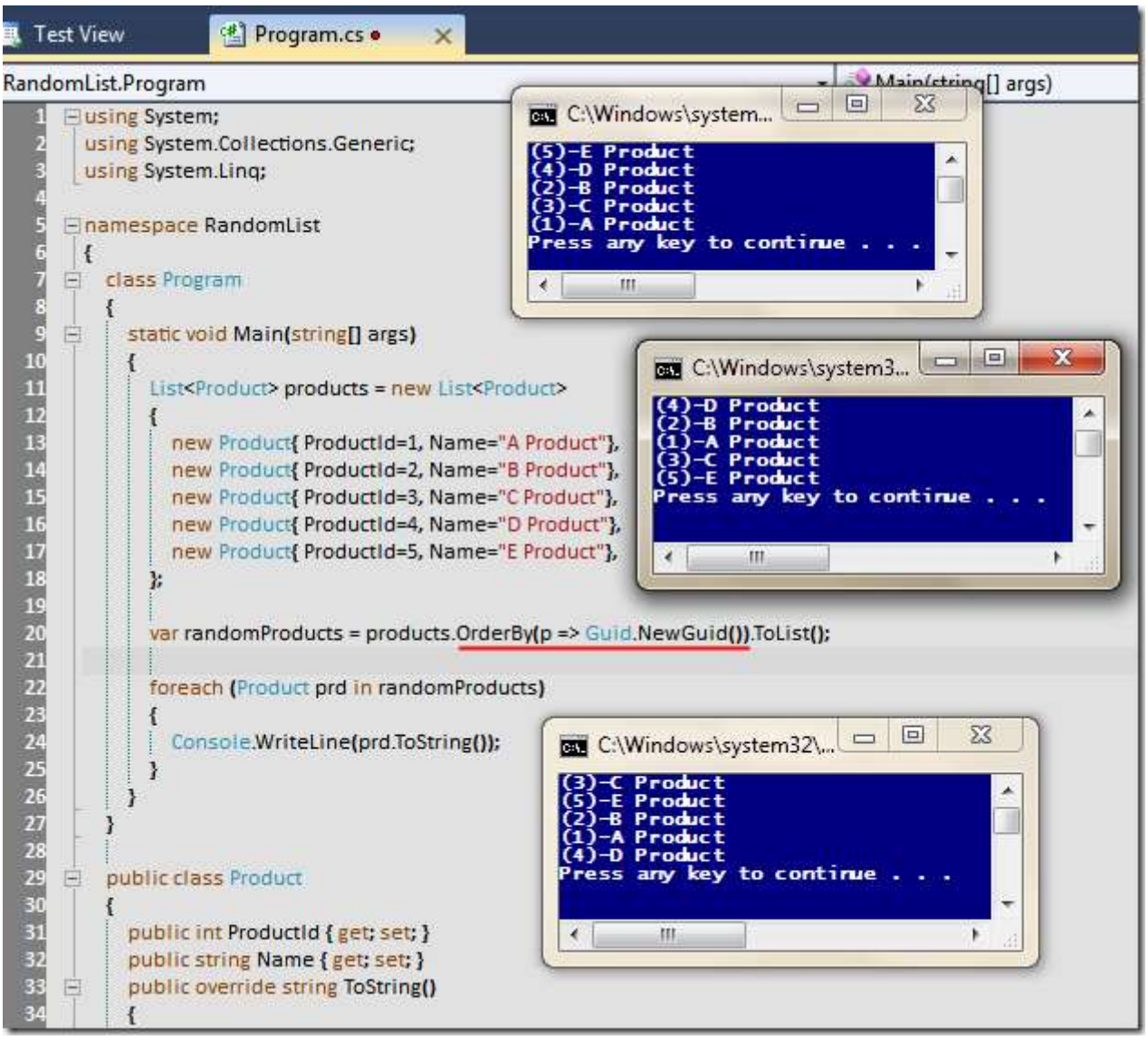
GoodExceptionCatching.rar (22,12 kb)

Tek Fotoluk İpucu - 5 (Rastgele Sıralı Generic List Koleksiyonu) (2011-06-24T09:15:00)

c#,c# 3.0,c# 4.0,c# temelleri,linq,list,generic list,collections,

Merhaba Arkadaşlar,

Elinizde List tipinden bir koleksiyon var ve içerisindeki nesnelerden rastgele sırada yeni bir liste kullanmak istiyorsunuz. Ne yaparsınız? İşte cevabı



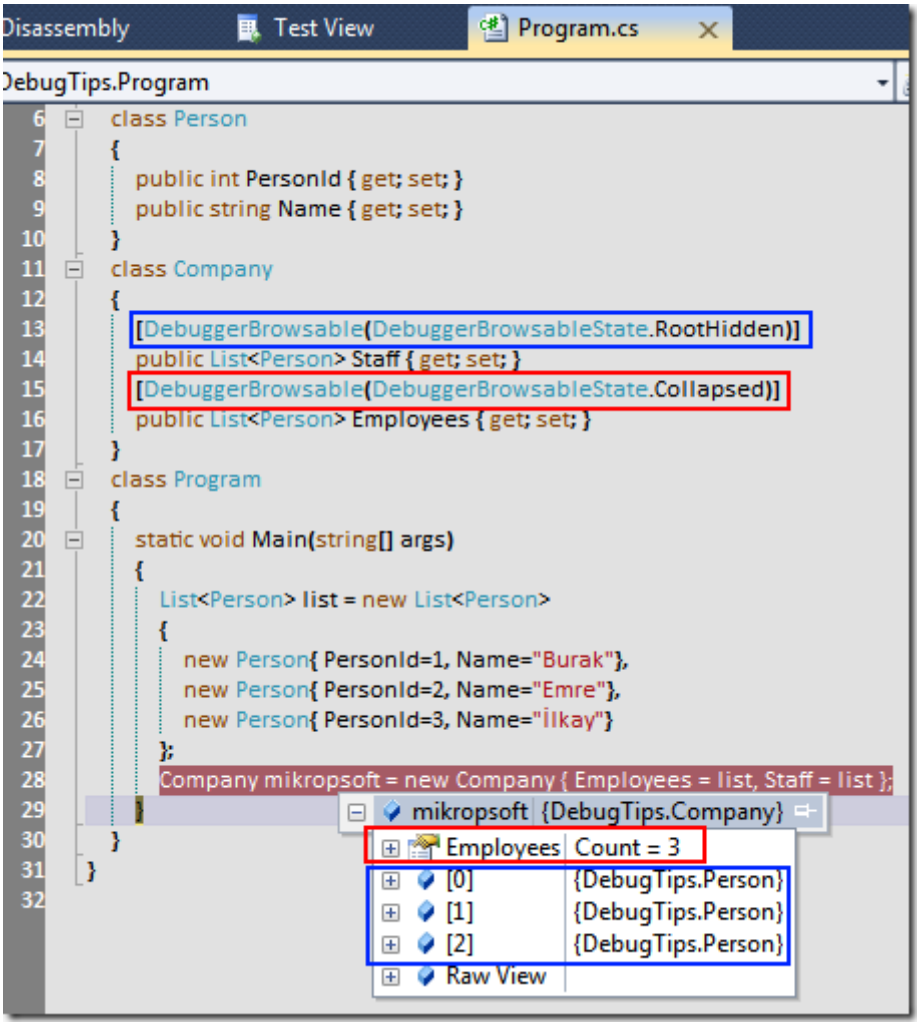
RandomList.rar (22,53 kb)

Tek Fotoluk İpucu - 4 (DebuggerBrowsable Niteliği) (2011-06-23T21:40:00)

c#,.net framework,visual studio,visual studio 2010,

Merhaba Arkadaşlar,

Attribute diyip geçmeyin. Bazıları çalışma zamanında o kadar çok işe yarıyor ki. örneğin DebuggerBrowsable niteliği. İşte kullanım şekli. Farkı görebiliyor musunuz?



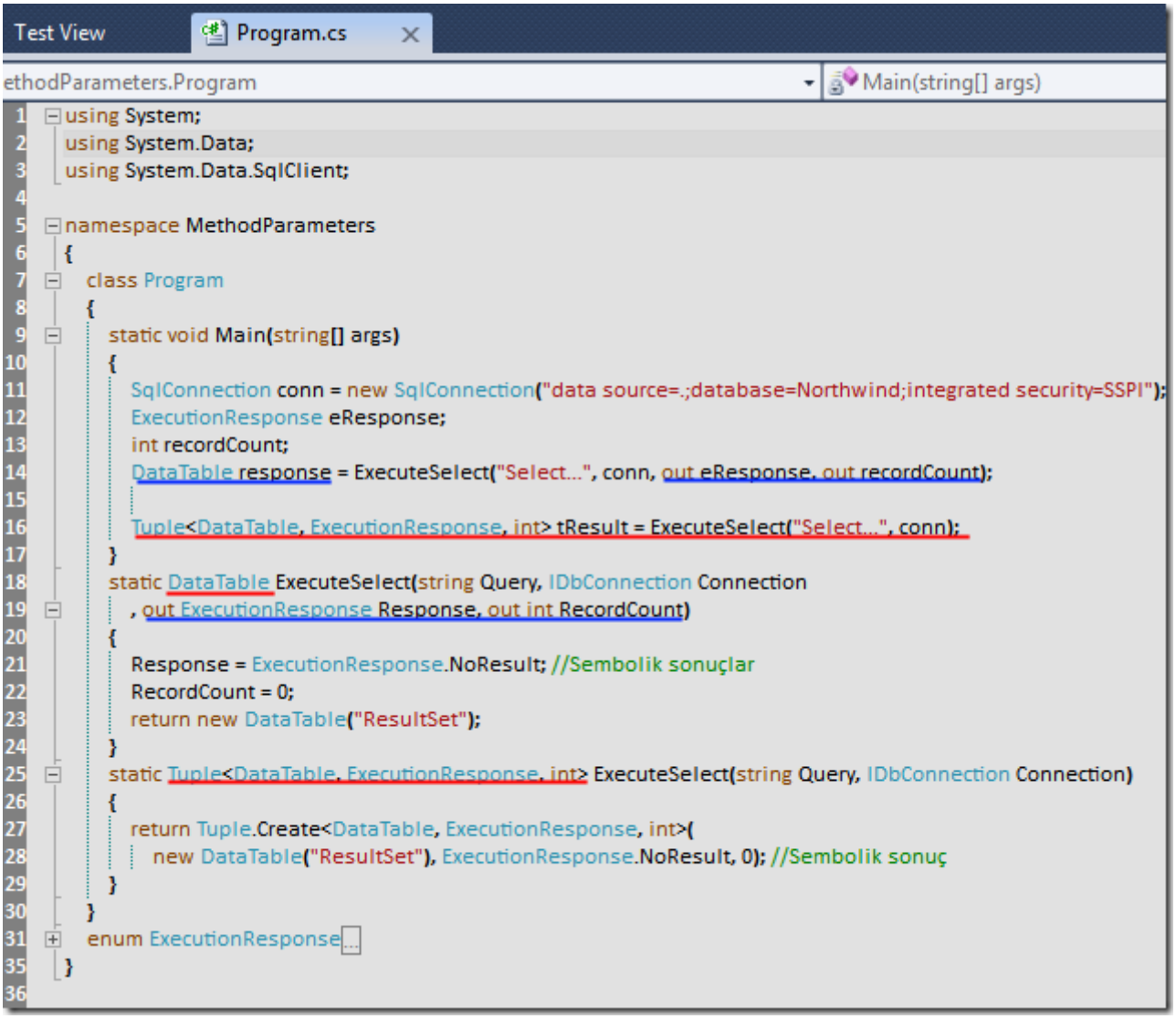
DebugTips.rar (21,35 kb)

Tek Fotoluk İpucu - 3 (Tuple) (2011-06-23T10:24:00)

c#,tuple,tuple<>,out parameters,.net framework 4.0,

Merhaba Arkadaşlar,

.Net Framework 4.0 ile gelen Tuple tipini duymayan kalmamıştır. Duymak bir yana en büyük sorun ne zaman ve hangi amaçlar ile kullanılabileceğidir. İşte tek fotoluk ipucu serisinin bu günkü konusu. örnek bir Tuple kullanımı. Metodlardan birden fazla değeri out veya ref ile döndürmek yerine, Tuple tipiyle döndürebiliriz.



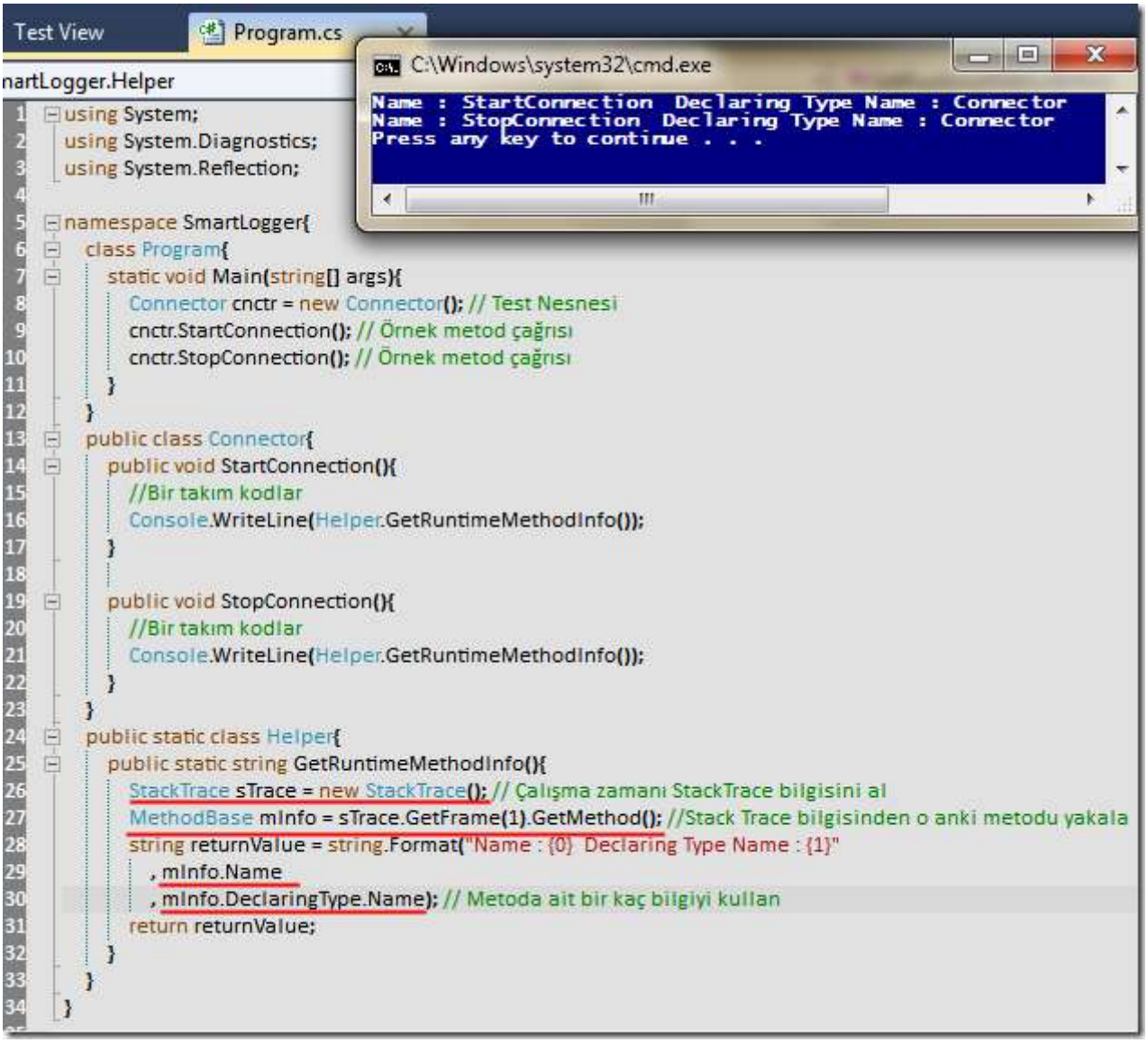
MethodParameters.rar (21,90 kb)

Tek Fotoluk İpucu - 2 (StackTrace ve Çalışma Zamanı Metod Bilgisi) (2011-06-22T16:48:00)

c#,system.diagnostics,stacktrace,run time,reflection,.net framework,

Merhaba Arkadaşlar,

Hani olurda çalışma zamanında(Runtime) o anda yürütülmekte olan metodun bilgilerine kolayca ulaşmak istersiniz. özellikle loglama sistemlerinde. İşte bu durumda **StackTrace** tipinden yararlanabilirsiniz. Nasıl mı? Aşağıdaki fotoğrafta*(ya da Ercan Hocamızın belirttiği üzere Screen Capture' da)* görüldüğü gibi



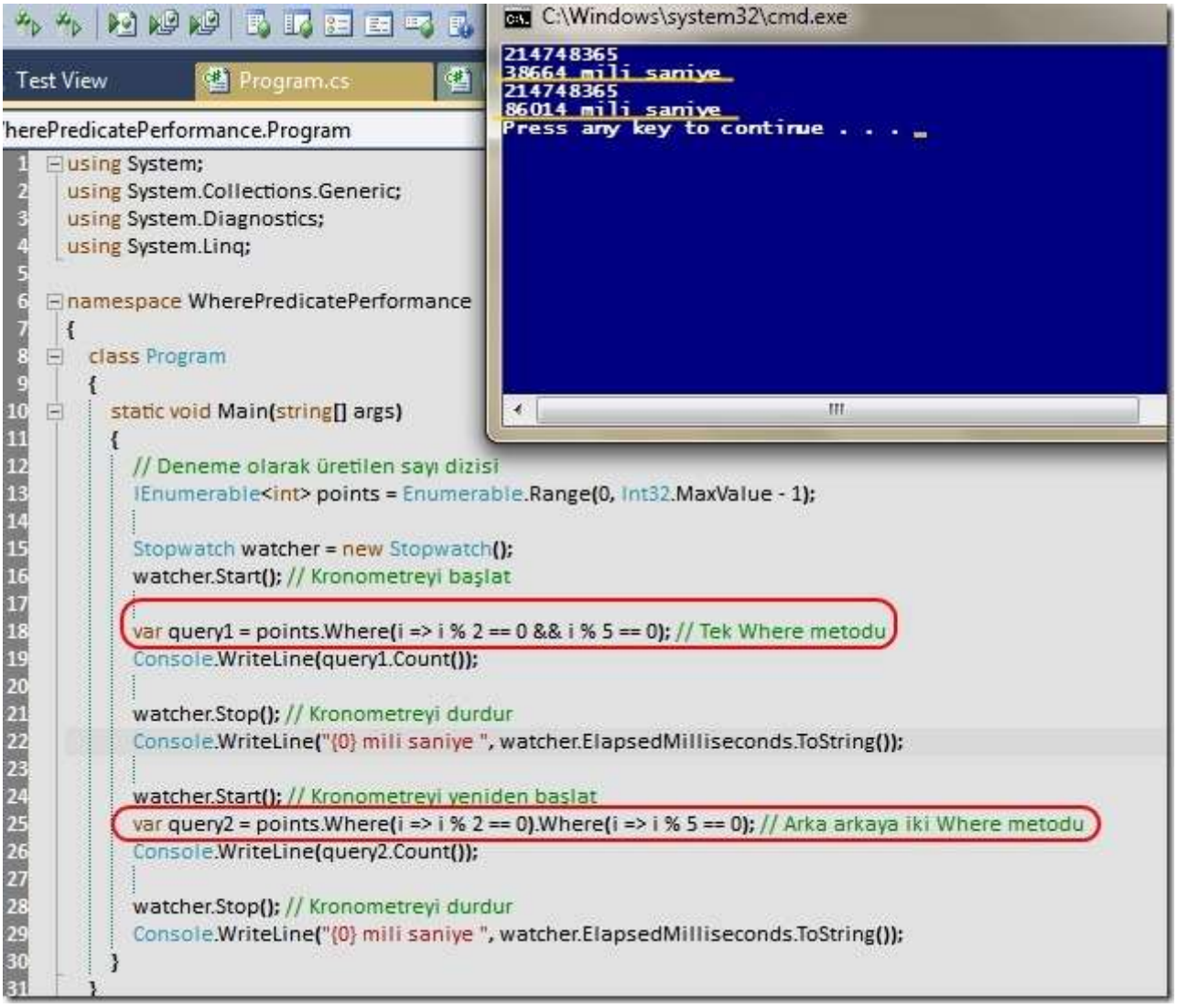
SmartLogger.rar (21,41 kb)

Tek Fotoluk İpucu - 1 (Tek Where ya da n adet Where) (2011-06-20T11:47:00)

c#,linq,where,

Merhaba Arkadaşlar,

Bazen bir fotoğraf bin kelimeye bedeldir derler. Bin kelime konusunda şüpheliyim ama bir fotoğrafın anlatım gücü açısından çok önemli katma değerlere sahip olduğuna inanıyorum. İşte size LINQ konusunda performans ipucu verecek bir fotoğraf. Bakalım ben de yarattığı etkiyi siz de yaratacak mı? 😉



Composite Cancellations (2010-12-20T00:15:00)

c# 4.0,parallel programming,cancellationtokensource,cancellationtoken,visual studio 2010 ultimate,task parallel library,task cancellation,monitoring cancellation,composite cancellationtokensource,

Merhaba Arkadaşlar,

Hatırlayacağınız üzere bir önceki yazımızda (Task İptal İşlemlerinin İzlenmesi(Monitoring Cancellation)) **Task Cancellation** işlemlerinin izlenmesi ile ilişkili teknikleri ve konuları irdelemeye başlamıştık. Bu yazımızda da iptal işlemleri ile ilgili farklı bir konuya değinmeye çalışıyor olacağız. Bu gün kü konumuz **Composite Cancellation** vakası.

Bildiğiniz üzere **Task** iptal taleplerinde, **CancellationTokenSource** örneğine ait **Cancel** metodunun çağırılması gerekmektedir.**CancellationTokenSource** örneği üzerinden yapılan iptal taleplerinin hangi **Task** işleyişini keseceğinin belirlenmesinde ise **CancellationToken**örneklerinden yararlanılmaktadır. **CancellationToken** örnekleri hatırlayacağını üzere **CancellationTokenSource** örnekleri tarafından üretilmekte ve **Task**' ler ile ilişkilendirilmektedir. Bu sebepten **Cancel** metodunun hangi **Task** ile ilişkili olduğu bellidir. Bir önceki yazımızda geliştirdiğimiz son örnekte aynı **Token** örneğini kullanan birden fazla **Task'** in tek bir iptal çağrısı ile nasıl kesilebileceğini incelemiştik. Bir diğer durum da şudur;

"Birden fazla CancellationTokenSource birbirlerine bağlanarak bir zincir oluşturulabilir ve bunlardan herhangibiri üzerinden Cancel işleminin yapılması, zincirdeki tüm source' lar için de aynı talebin gerçekleştirilmesi anlamına gelmektedir."

Konuyu daha net kavrayabilmek için aşağıdaki kod parçasını göz önüne alabiliriz.

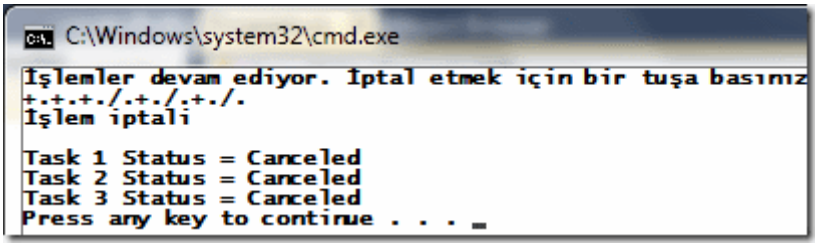
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace CancellationScenarios2
{
   class Program
   {
      static void Main(string[] args)
      {
         IEnumerable<int> numbers = Enumerable.Range(0, 100000000);

         CancellationTokenSource tokenSource1 = new CancellationTokenSource();
         CancellationTokenSource tokenSource2 = new CancellationTokenSource();
         CancellationTokenSource tokenSource3 = new CancellationTokenSource();

         CancellationTokenSource compositeSource =
CancellationTokenSource.CreateLinkedTokenSource(tokenSource1.Token,
tokenSource2.Token,tokenSource3.Token);

         Task startedTask = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers.Count(); i++)
            {
               compositeSource.Token.ThrowIfCancellationRequested();
               i++;
```

```
        i--;
        i *= 2;
        Console.Write(".");
    }
}
, compositeSource.Token);

Task startedTask2 = Task.Factory.StartNew(() =>
{
    for (int i = 0; i < numbers.Count(); i++)
    {
        compositeSource.Token.ThrowIfCancellationRequested();
        i++;
        i--;
        i++;
        i *= 2;
        Console.Write("+");
    }
}
, compositeSource.Token);

Task startedTask3 = Task.Factory.StartNew(() =>
{
    for (int i = 0; i < numbers.Count(); i++)
    {
       compositeSource.Token.ThrowIfCancellationRequested();
        i++;
        i *= 2;
        Console.Write("/");
    }
}
, compositeSource.Token);

compositeSource.Token.Register(() =>
{
    Console.WriteLine("İşlem iptali");
}
);

Console.WriteLine("İşlemler devam ediyor. İptal etmek için bir tuşa basınız");
Console.ReadLine();

tokenSource3.Cancel();
```

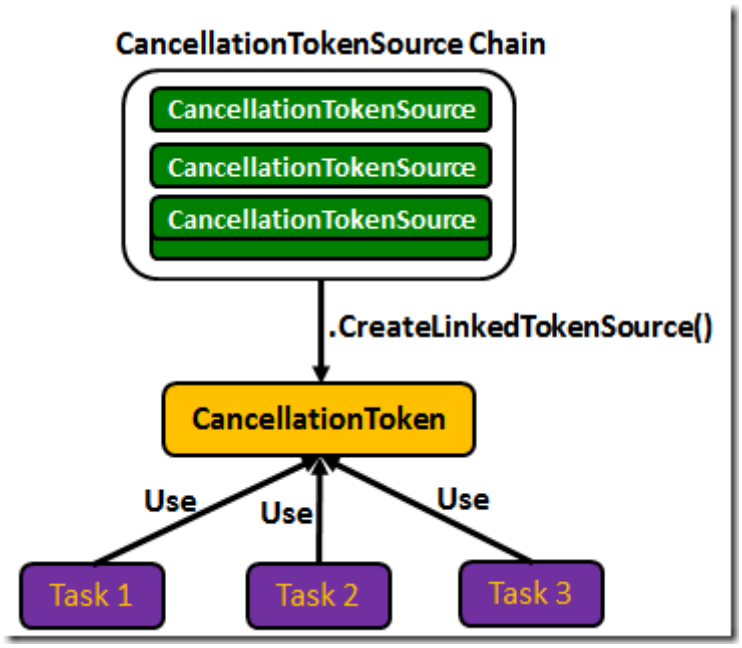
```
            Console.ReadLine();
            Console.WriteLine("Task 1 Status = {0}\nTask 2 Status = {1}\nTask 3 Status =
{2}", startedTask.Status, startedTask2.Status,startedTask3.Status);
        }
    }
}
```

örnek uygulamamızda 3 adet **CancellationTokenSource** nesnesi örneklendiği görülmektedir. Sonrasında ise bu örnekler **CancellationTokenSource** sınıfının **static CreateLinkedTokenSource** metodu yardımıyla birbirlerine bağlanmıştır. 3 Task örneği oluşturulduğu sırada yapılan **Token** bildirimlerinde, **CreateLinkedTokenSource** metodu ile üretilen **CancellationTokenSource** nesne örneğine ait **Token** özelliğinden yararlanılmıştır.

Kodun ilerleyen kısımlarında **Task** fonksiyonellikleri icra ettikleri işleri tamamlanmadan önce kullanıcı bir tuşa basarsa **tokenSource3** isimli değişken üzerinden yapılan **Cancel** çağırısı devreye girmektedir. Buna göre zincir içerisinde yer alan **CancellationTokenSource** örneklerinin her biri için bir iptal talebi söz konusu olacaktır. Dolayısıyla uygulamanın çalışması sonrasında aşağıdaki ekran görüntüsüne benzer sonuçlar üretilecektir.



Dikkat edileceği üzere 3 Task içinde **Status** değeri **Canceled** olmuştur. **CancellationTokenSource** örneklerinin birden fazla olduğu ve herhangibiri üzerinde yapılan iptal işleminin diğerleri içinde söz konusu olması gerektiği durumlarda bu teknikten yararlanılabilir. İlk örneğimizdeki durumu kafamızda daha kolay canlandırmak için aşağıdaki tasviri göz önüne alabiliriz.



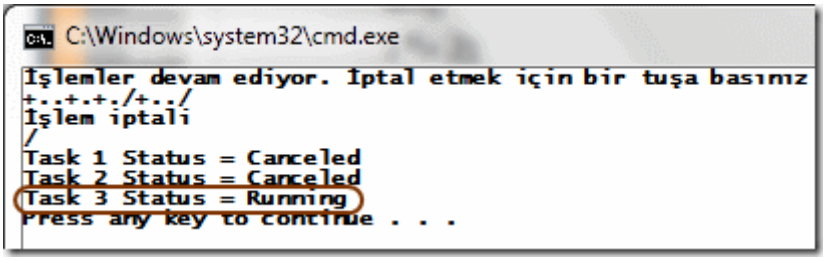
Şimdi konuyu farklı bir bakış açısından değerlendirmek için kodu biraz değiştirip aşağıdaki hale getirelim.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace CancellationScenarios2
{
   class Program
   {
      static void Main(string[] args)
      {
         IEnumerable<int> numbers = Enumerable.Range(0, 100000000);

         CancellationTokenSource tokenSource1 = new CancellationTokenSource();
         CancellationTokenSource tokenSource2 = new CancellationTokenSource();
         CancellationTokenSource tokenSource3 = new CancellationTokenSource();

         CancellationTokenSource compositeSource =
CancellationTokenSource.CreateLinkedTokenSource(tokenSource1.Token,
tokenSource2.Token);

         Task startedTask = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers.Count(); i++)
            {
```

```
        compositeSource.Token.ThrowIfCancellationRequested();
        i++;
        i--;
        i *= 2;
        Console.Write(".");
    }
}
, compositeSource.Token);

Task startedTask2 = Task.Factory.StartNew(() =>
{
    for (int i = 0; i < numbers.Count(); i++)
    {
        compositeSource.Token.ThrowIfCancellationRequested();
        i++;
        i--;
        i++;
        i *= 2;
        Console.Write("+");
    }
}
, compositeSource.Token);

Task startedTask3 = Task.Factory.StartNew(() =>
{
    for (int i = 0; i < numbers.Count(); i++)
    {
        tokenSource3.Token.ThrowIfCancellationRequested();
        i++;
        i *= 2;
        Console.Write("/");
    }
}
, tokenSource3.Token);

compositeSource.Token.Register(() =>
{
    Console.WriteLine("İşlem iptali");
}
);

Console.WriteLine("İşlemler devam ediyor. İptal etmek için bir tuşa basınız");
Console.ReadLine();

tokenSource1.Cancel();
```

```
            Console.ReadLine();
            Console.WriteLine("Task 1 Status = {0}\nTask 2 Status = {1}\nTask 3 Status =
{2}", startedTask.Status, startedTask2.Status,startedTask3.Status);

        }
    }
}
```

Bu örnekte sadece tokenSource1 ve tokenSource2 örnekleri birbirlerine bağlanmış ancak tokenSource3 bu zincirin dışında tutulmuştur. Buna göre tokenSource1.Cancel çağrısının gerçekleştirilmesi sonucunda sadece zincir içerisinde dahil edilmiş Task' lerin iptal işlemi gerçekleşecektir. çalışma zamanı çıktısında bu durum net bir şekilde görülebilir.



Görüldüğü gibi **Task 1** ve **2** için **Status** değerleri **Canceled** olarak set edilmişken, **Task 3** çalışmaya devam ettiğinden **Running** değerini almıştır.

Böylece geldik bir yazımızın daha sonuna. Paralel programlama ile ilişkili konuları incelemeye devam ediyor olacağız. Bundan sonraki yazımızda **Task'** lerin belirli süreler boyunca bekletilmesi amacıyla kullanabileceğimiz teknikleri araştırıyor ve örnekliyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

CancellationScenarios2.rar (21,79 kb) **[örnek Visual Studio 2010 Ultimate sürümünde geliştirilmiş ve test edilmiştir]**

Task İptal İşlemlerinin İzlenmesi(Monitoring Cancellation) (2010-12-20T00:10:00)

parallel programming,task parallel library,cancellationtokensource,task cancellation,cancellationtoken,c# 4.0,visual studio 2010 ultimate,monitoring cancellation,

Merhaba Arkadaşlar,

Bu yazımızda daha önceden **.Net Framework Beta 1** ve **Beta 2** sürümlerinde incelediğimiz **Task** iptal işlemlerini son sürümde ele alıp toparlamaya çalışıyor olacağız.

Task iptal işlemleri oldukça önemli ve üzerinde titizlikle durulması gereken bir konudur. Nitekim bazı hallerde

çalıştırılmakta olan **Task** işlevlerinin iptal edilmesi gerekebilir. Bu iptal işlemi, sistem tarafından her hangibir koşulun gerçeklenmesi sonucu talep edilebileceği gibi, kullanıcı tarafından da uygulatılmak istenebilir.

Task örnekleri işaret ettikleri ve planladıkları fonksiyonellikleri çalıştırdıklarında ve herhangibir sebeple iptal işlemi gerçekleştirilmek istendiğinde**CancellationTokenSource** ve **CancellationToken** tiplerinden yararlanılması gerekmektedir.

Bu amaçla İptal işlemlerinde **CancellationTokenSource** nesne örnekleri üzerinden **Cancel** metodunun çağırılması yeterlidir. Ancak burada da dikkat edilmesi gereken bir husus vardır. çalışma zamanı **Cancel** çağrısı sonucu ilgili **Task** örneklerinin işleyişlerini otomatik olarak sonlandırmaz. Bir başka deyişle**Task** gövdeleri veya paralel yürütülen metod blokları içerisinde **Cancel** işleminin talep edilip edilmediğinin takibi gerekmektedir(**Monitoring Cancellation**). Peki bu takip işlemleri nasıl gerçekleştirilebilir. Yazımızın ilerleyen kısımlarında bu takip işlemlerinde hangi konuların ele alındığını irdelemeye çalışıyor olacağız.

Polling Tekniğini ile İptal Taleplerinin İzlenmesi

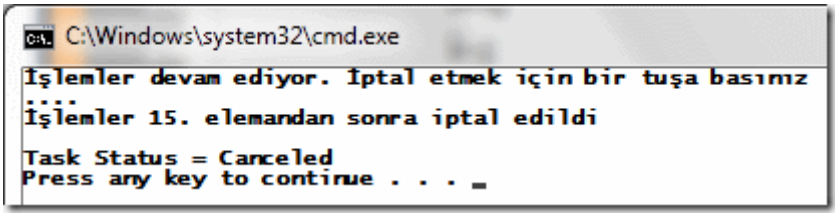
Bu teknik adından da anlaşılacağı üzere bir iptal talebinin yapılıp yapılmadığını sürekli olarak denetlemeyi gerektirir. Bu durumu analiz etmek için **Visual Studio 2010 Ultimate**ortamında geliştirilmiş aşağıdaki kod parçasını göz önüne alabiliriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace CancellationScenarios
{
   class Program
   {
      static void Main(string[] args)
      {
         CancellationTokenSource tokenSource = new CancellationTokenSource();
         CancellationToken token = tokenSource.Token;
         IEnumerable<int> numbers =Enumerable.Range(0, 100000000);

         Task startedTask=Task.Factory.StartNew(() =>
            {
              for(int i=0;i<numbers.Count();i++)
               {
                  if (token.IsCancellationRequested)
```

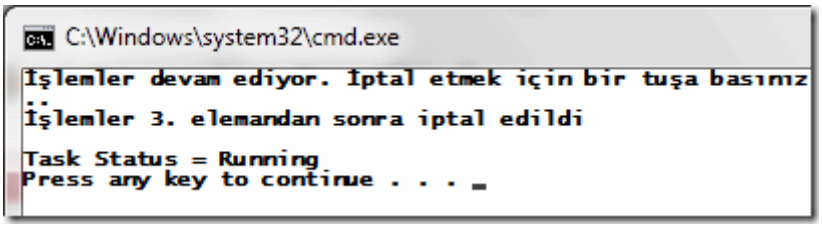
```
                {
                   Console.WriteLine("İşlemler {0}. elemandan sonra iptal
edildi",i.ToString());
                  throw new OperationCanceledException(token);
                }
                else
                {
                   i++;
                   i--;
                   i *= 2;
                   Console.Write(".");
                }
             }
          }
        , token);

        Console.WriteLine("İşlemler devam ediyor. İptal etmek için bir tuşa basınız");
        Console.ReadLine();

        tokenSource.Cancel();

        Console.ReadLine();
        Console.WriteLine("Task Status = {0}",startedTask.Status);
     }
  }
}
```

İlk olarak **CancellationTokenSource** nesnesi örneklenmiş ve sonrasında bu örneğin **Token** özelliğinden yararlanılarak bir **CancellationToken** üretilmiştir. **CancellationToken** nesne örneği**Task** tipinin örneklenmesi sırasında da parametre olarak aktarılmaktadır.

Dikkat edileceği üzere **for** döngüsü içerisinde parametre olarak gelen **token** değişkeni üzerinden **IsCancellationRequested** özelliği kontrol edilmektedir. Kontrol işlemi döngünün her bir iterasyonunda söz konusudur. **IsCancellationRequested** özelliğinin **true** değerini alması için **CancellationToken** ile ilişkilendirilmiş olan **CancellationTokenSource** nesne örneği üzerinden**Cancel** metodunun çağırılması gerekmektedir. örneği bu haliyle çalıştırdığımızda ve işlemler devam ederken herhangibir noktada herhangibir tuşa bastığımızda ise aşağıdaki ekran görüntüsüne benzer sonuçlar ile karşılaşırız.



önemli olan noktalardan birisi de başlatılan **Task'** in **Status** değeridir. Dikkat edileceği üzere bu değer **Canceled** olarak set edilmiştir. Ancak bunun böyle olmasının sebebi iptal işleminin kontrol edildiği noktadan **OperationCanceledException** tipinden istisna nesnesinin fırlatılmasıdır. Bu istisnanın üretilmemesi sonucu ise aşağıdaki gibi olacaktır ki bu da **Task** durumunu değerlendiren kod parçaları için tam anlamıyla bir handikaptır.

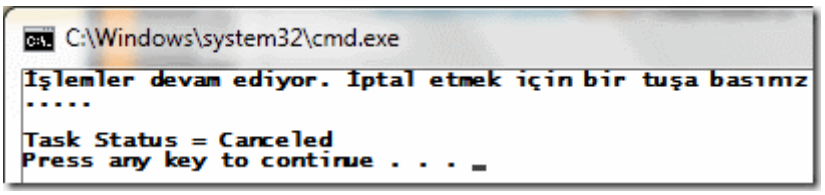


Görüldüğü gibi **Exception** fırlatılmadığı için **Task** durumu **Running** olarak kalmıştır. Oysaki **Task** iptal edilmiştir.

Polling modelinde uygulanan genel kod deseni yukarıdaki gibi olmasına rağmen ikinci bir teknik daha söz konusudur. Bu teknikte kontrol etme ve Exception fırlatma işlemleri zaten var olan bir metoda yüklenmiştir. Buna göre for döngüsü içeriğini aşağıdaki gibi değiştirdiğimizi düşünelim.

```
for(int i=0;i<numbers.Count();i++)
            {
               token.ThrowIfCancellationRequested();
               i++;
               i--;
               i *= 2;
               Console.Write(".");
            }
```

Bu durumda da çalışma zamanında aşağıdaki sonuçları alırız.



Polling tekniği özellikle **Task** gövdelerinin döngüsel işlemler yaptığı senaryolar için uygundur. Nitekim döngünün her iterasyonu sırasında, bir iptal isteği olup olmadığı kontrol edilmekte ve eğer varsa, döngü dışına çıkılması, ortama uygun **İstisna(Exception)** nesnesi

fırlatılması, gerekiyorsa ilgili **kaynakların(Resources)** serbest bırakılması işlemleri gerçekleştirilmektedir.

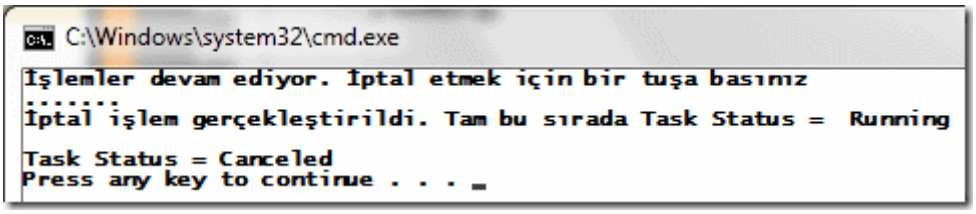
Delegate Kullanımı

Polling modeli kullanışlı olmasına rağmen
özellikle **CancellationToken** üzerinden **ThrowIfCancellationRequested** metodu kullanıldığında, sanki eksik olan bir şey var hissi uyandırmaktadır. **6ncı** hissimiz bize bu noktada şunu söylemektedir; **User Interface'** e sahip uygulamalarda kullanıcıların işlemin iptal edildiğine dair bilgilendirilmesi nasıl sağlanacaktır? Bu his biraz daha genişletilebilir ama sonuç itibariyle iptal işlemi sonrası ana uygulamanın bilgilendirilmesi önemlidir. özellikle işlemlerin loglandığı, IO süreçlerinin söz konusu olduğu durumlarda. Bu tip bir vaka da delegasyon kullanımı ile iptal işleminin ele alınması tercih edilebilir. Aşağıdaki örnek kod parçasında bu durum irdelenmektedir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace CancellationScenarios
{
   class Program
   {
      static void Main(string[] args)
      {
         CancellationTokenSource tokenSource = new CancellationTokenSource();
         CancellationToken token = tokenSource.Token;
         IEnumerable<int> numbers = Enumerable.Range(0, 100000000);

         Task startedTask = Task.Factory.StartNew(() =>
            {
               for (int i = 0; i < numbers.Count(); i++)
               {
                  token.ThrowIfCancellationRequested();
                  i++;
                  i--;
                  i *= 2;
                  Console.Write(".");
               }
            }
         , token);
```

```
        token.Register(() =>
        {
           Console.WriteLine("İptal işlem gerçekleştirildi. Tam bu sırada Task Status
=  {0}",startedTask.Status);
        }
        );
        Console.WriteLine("İşlemler devam ediyor. İptal etmek için bir tuşa basınız");
        Console.ReadLine();

        tokenSource.Cancel();

        Console.ReadLine();
        Console.WriteLine("Task Status = {0}", startedTask.Status);
      }
   }
}
```

Dikkat edileceği üzere token nesnesi üzerinden **Register** metodu kullanılarak bir delegasyon yapılmaktadır. Buna göre **tokenSource.Cancel** çağrısı gerçekleştirildiğinde, **Register**metodu ile işaret edilen blok otomatik olarak devreye girecektir. örneğin çalışma zamanı çıktısı aşağıdaki ekran görüntüsündekine benzer olacaktır.



Dikkat edilmesi gereken notkalardan birisi de, delegasyon metodu içerisine girildiğinde iptal edilen Task örneğinin durumunun halen Running olarak görünmesidir. Ancak işlem tamamlandıktan sonra Status değeri Canceled olmaktadır.

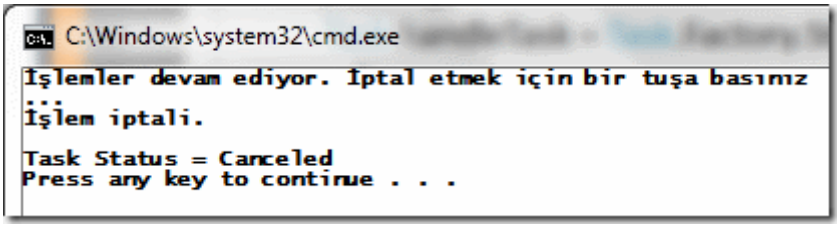
***Bloğun sorusu; Register** ile işaret edilen delegasyon metodu içerisinden, **for** döngüsünün hangi iterasyonunda iptal işleminin gerçekleştirildiği çalışma ortamına nasıl bildirilebilir?*

Wait Handle Kullanımı

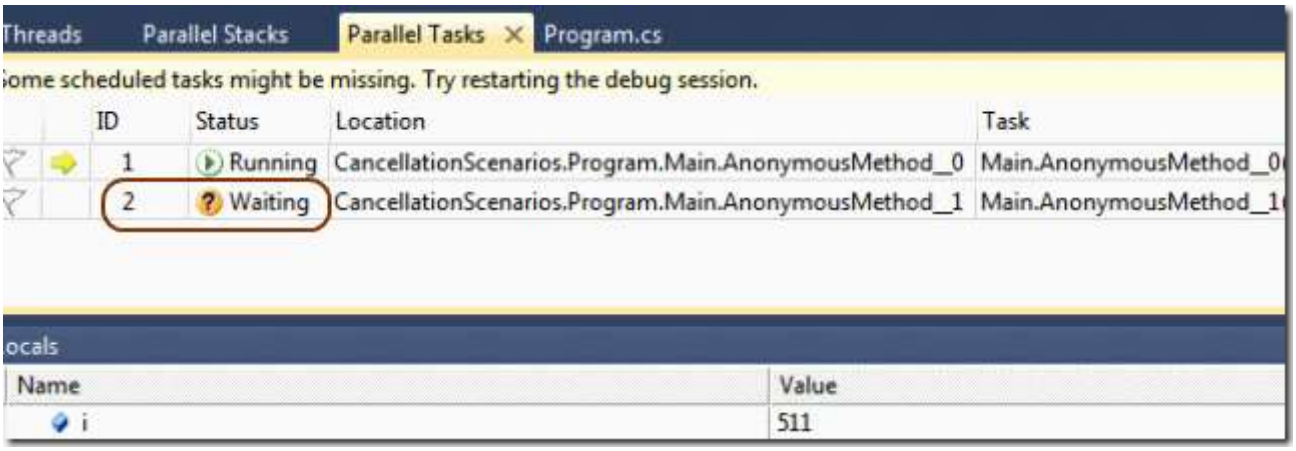
Task İptal işlemlerinde delegasyon kullanımına alternatif bir yol olarak **Wait Handle** tekniğinden de yararlanılabilir. Bu kullanım şeklini daha net bir kavrayabilmek adına aşağıdaki kod parçasını göz önüne alalım.

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Threading;
using System.Threading.Tasks;

namespace CancellationScenarios
{
   class Program
   {
      static void Main(string[] args)
      {
         CancellationTokenSource tokenSource = new CancellationTokenSource();
         CancellationToken token = tokenSource.Token;

         IEnumerable<int> numbers = Enumerable.Range(0, 100000000);

         Task startedTask = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers.Count(); i++)
            {
               token.ThrowIfCancellationRequested();
               i++;
               i--;
               i *= 2;
               Console.Write(".");
            }
         }
         , token);
         Task handleTask = Task.Factory.StartNew(() =>
            {
               token.WaitHandle.WaitOne();
               Console.WriteLine("İşlem iptali.");
            }
         );
         Console.WriteLine("İşlemler devam ediyor. İptal etmek için bir tuşa basınız");
         Console.ReadLine();

         tokenSource.Cancel();

         Console.ReadLine();
         Console.WriteLine("Task Status = {0}", startedTask.Status);
      }
   }
}
```



Bu kullanım şeklinde ikinci bir **Task** bloğunun rol alması gözden kaçmamalıdır ;) **handleTask** isimli **Task**' e ait kod bloğu içerisinde, **CancellationToken** nesnesi üzerinden**WaitHandle.WaitOne** çağrısının gerçekleştirildiği görülmektedir. Aslında çalışma zamanında her hangibir anda **Task** durumlarına bakıldığında aşağıdaki ekran görüntüsündekine benzer sonuçlarla karşılaşıldığı görülecektir.



Dikkat edileceği üzere **handleTask** örneğinin durumu **Waiting** olarak görülmektedir. Bu son derece doğaldır nitekim **Task** örneğinin başlatılması sonrasında devreye giren fonksiyon bloğunun daha ilk satırında **WaitOne** ile **Thread** ' in bekletilmesi söz konusudur.

CancellationTokenSource nesne örneği üzerinden **Cancel** metodunun çağırılması sonrasında ise **WaitOne** metod çağrısını takip eden kod satırına girildiği ve **startedTask**örneğinin icra etmekte olduğu işlemlerinde iptal edildiği görülebilir.

Buraya kadar geliştirdiğimiz örnek kod parçaları ve iptal deseneleri göz önüne alındığında her zaman için **CancellationTokenSource** üzerinden üretilen bir **CancellationToken** nesneleri olduğu görülmektedir. Ayrıca iptal işlemi de her zaman için **CancellationTokenSource** örneği üzerinden yapılmaktadır. **CancellationToken** örnekleri ise **Task** üretimi sırasında ve içeride kullanılmaktadır. Peki bu bizim ne işimize yarayabilir?

Birden Fazla Task İşleminin Tek Noktadan İptal Edilmesi

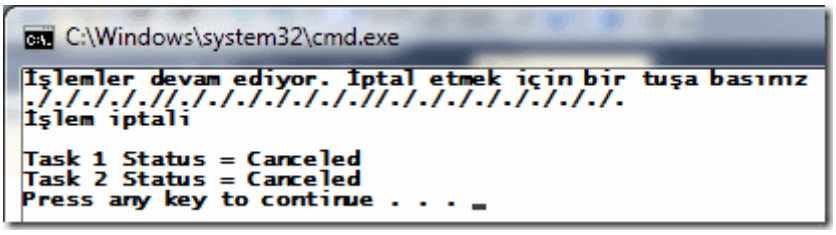
İşte az önceki sorunun cevabı. Aynı **CancellationToken** nesne örneğinin birden fazla **Task** ile ilişkilendirilmesi, bu token örnekleri ile ilişkili olan **CancellationTokenSource** örneği üzerinden yapılan iptal talebinin, tüm

ilişkili **Task'** lere iletilmesi anlamına gelmektedir. Gelin bu cümleyi aşağıdaki kod parçası ile anlamaya çalışalım.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace CancellationScenarios
{
   class Program
   {
      static void Main(string[] args)
      {
         CancellationTokenSource tokenSource = new CancellationTokenSource();
         CancellationToken token = tokenSource.Token;

         IEnumerable<int> numbers = Enumerable.Range(0, 100000000);
         IEnumerable<int> numbers2 = Enumerable.Range(0, 100000000);

         Task startedTask = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers.Count(); i++)
            {
               token.ThrowIfCancellationRequested();
               i++;
               i--;
               i *= 2;
               Console.Write(".");
            }
         }
         , token);

         Task startedTask2 = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers2.Count(); i++)
            {
               token.ThrowIfCancellationRequested();
               i++;
               i--;
               i *= 2;
               Console.Write("/");
            }
```

```
        }
        , token);

        token.Register(() =>
           {
              Console.WriteLine("İşlem iptali");
           }
        );

        Console.WriteLine("İşlemler devam ediyor. İptal etmek için bir tuşa basınız");
        Console.ReadLine();

        tokenSource.Cancel();

        Console.ReadLine();
        Console.WriteLine("Task 1 Status = {0}\nTask 2 Status = {1}",
startedTask.Status,startedTask2.Status);
      }
   }
}
```

Kod parçasında iki farklı **Task** içeriği söz konusudur. Ancak her iki **Task** örneklenirken aynı **CancellationToken** ile ilişkilendirilmiştir. Buna göre **CancellationTokenSource** üzerinden yapılacak olan iptal talebi, her iki task işlemi içinde istenmiş olmaktadır. örnek çalışma zamanı çıktısı aşağıdaki şekilde görüldüğü gibidir.



Görüldüğü gibi her iki Task örneğinin durumu Canceled olarak set edilmiştir. Task iptal işlemleri ile ilişkili olarak ele almamız gereken farklı konular da mevcuttur. Bunları bir sonraki yazımızda ele almaya çalışıyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

CancellationScenarios.rar (23,87 kb) **[örnekler Visual Studio 2010 Ultimate sürümü üzerinde geliştirişmiş ve test edilmiştir]**

Big Big Big Integer ve Faktöryel Hesaplarken Yüzümde Oluşan Tebessüm (2010-12-20T00:05:00)

biginteger,system.numerics,.net framework 4.0,c# 4.0,visual studio 2010 ultimate,

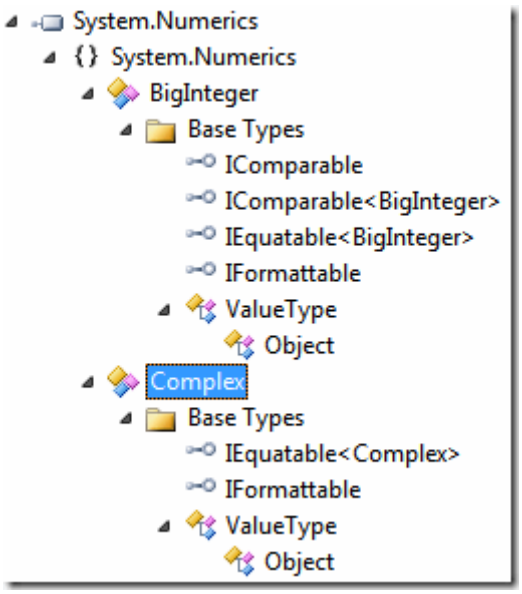
Merhaba Arkadaşlar,

Monster Truck yarışlarını izleyen var mıdır bilemiyorum. Bir zamanlar Eurosport kanalında sık sık izler ve bu devasa, kocaman araçların, önlerinde ufacık kalan*(ki o araçların çoğu avrupada kullanılan binek otoların çoğundan en ve boyca büyüktür)* araçların üstünden atlarken onları nasıl ezdiklerine ağzım açık bakardım.

Amerikalıların gerçekten garip müsabaka anlayışları ve sportif aktiviteleri var. Monster Truck araçlarının kullanıldığı bu tip yarışmalarda bile binlerce seyirciyi toplayabiliyorlar. üstelik bu seyirciler çılgınlar gibi bağırıp duruyor ve keyif alıyorlar. (Biz daha basketbol maçlarına seyirci toplayamazken üstelik :( )

Bu hüzünlü girişten sonra bu kocaman araçların konumuzla ne alakası olduğunu düşünebilirsiniz. Aslında bu gün sizlere yine **.Net Framework 4.0** ile birlikte gelen yeniliklerden birisinde bahsediyor olacağım. Aslında kocaman, iri, büyük bir yenilik. **BigInteger** ;)

.Net Framework 4.0 ile birlikte **System.Numerics.dll** isimli bir **assembly** daha gelmektedir. Bu yeni assembly içerisinde ise aşağıdaki şekilde görülen iki **Değer Türü(Value Type)** yer almaktadır.

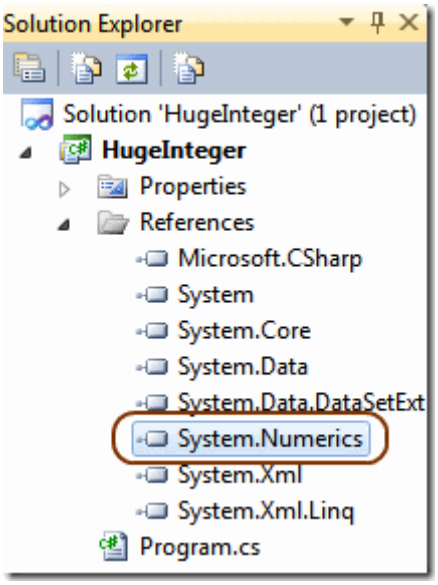


Hey gidi günler :D Bir zamanlar **C#Nedir?** adına düzenlenen **C# Akademi** eğitimlerinde, **operatörlerin aşırı yüklenmesi(Operator Overloading)** konusunu anlatırken genellikle kompleks sayılardan(-3i+2j gibi) yararlanırdık. öncelikle kompleks sayıları ifade edebileceğimiz bir tip tanımlar ve bu tipe toplama, çıkarma gibi matematiksel işlemleri öğreterek **Operator Overloading**konusunu irdelerdik. Nihayet **.Net Framework 4.0** sürümü ile birlikte **Complex** isimli yeni bir değer türüne daha sahip olduk. Tahmin edeceğiniz üzere bu tip ile kompleks sayıları ifade edebilmekteyiz.

System.Numerics içerisine dahil edilen ve bu yazımıza konu olan diğer bir tip ise **BigInteger** isimli tam sayı türüdür. Bu tip ile **gerçekten çok büyük** sayıları ifade edebilmemiz mümkündür. Bu önemli bir gelişmedir. Nitekim **BigInteger** dışında değerlendirebileceğimiz büyük sayısal değerleri düşündüğümüzde, değer aralıklarının aşağıdaki tabloda ifade edildiği gibi olduklarını görebiliriz.

Tip	Minimum Değer	Maksimum Değer
Int64(long)	-9223372036854775808	9223372036854775807
Unsigned Int64	0	18446744073709551615
Decimal	-79228162514264337593543950335	79228162514264337593543950335
Double	-1,79769313486232E+308	1,79769313486232E+308

Her ne kadar değer aralıkları büyük görünse de bazı durumlarda asla yeterli gelmeyeceklerdir. Bu durumu daha net bir şekilde anlayabilmek için aşağıdaki **Console** uygulaması kodlarını göz önüne alalım. Bu arada **BigInteger** tipini kullanabilmemiz için **System.Numerics.dll assembly'** ının projeye referans edilmesi gerektiğini de unutmayalım.



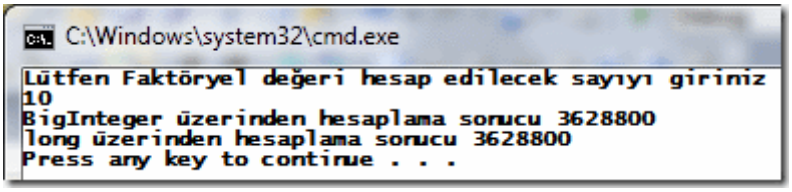
```
using System;
using System.Numerics;

namespace HugeInteger
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Lütfen Faktöryel değeri hesap edilecek sayıyı giriniz");
         int number = 0;
         if (Int32.TryParse(Console.ReadLine(), out number))
         {
            Console.WriteLine("BigInteger üzerinden hesaplama sonucu {0}
",Factorial(number));
            Console.WriteLine("long üzerinden hesaplama sonucu {0}
",FactorialOld(number));
         }
         else
            Console.WriteLine("Geçersiz sayısal değer");
      }
      static BigInteger Factorial(int value)
      {
         if (value == 0 || value == 1)
            return 1;
         else
            return value*Factorial(value-1);
      }
      static long FactorialOld(int value)
      {
         if (value == 0 || value == 1)
            return 1;
         else
            return value * FactorialOld(value - 1);
      }
   }
}
```

Yine hey gidi günler diyeceğim :) özellikle **C#** programlama dilinin temellerinden **Recursive** metodların anlatılmasında en çok kullandığımız fonksiyonellikler arasında, **Faktöryel** ve**Fibonacci** sayı dizisi hesaplamaları gelmekteydi.

Yukarıdaki kod parçasında da kullanıcının girdiği sayısal değerin faktöryel hesaplamalarının yapıldığı iki **yinelemeli(Recursive)** metod görülmektedir. Bu metodlar arasındaki tek fark ise **Factorial** metodunun **BigInteger** tipinden bir değer döndürmesi

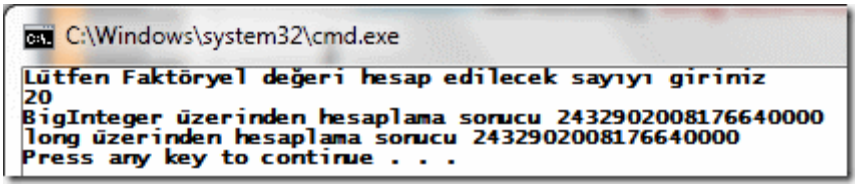
diğerinin ise **long** tipini kullanmasıdır. İyi de ne olmuş ki? Gelin bir kaç sayısal değer için deneme yapalım.

10! (10 Faktöryel)



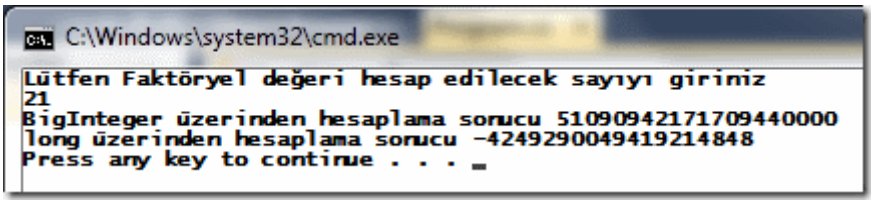
10 sayısının faktöryeli için bulunun sonuçlar güzel.

20!



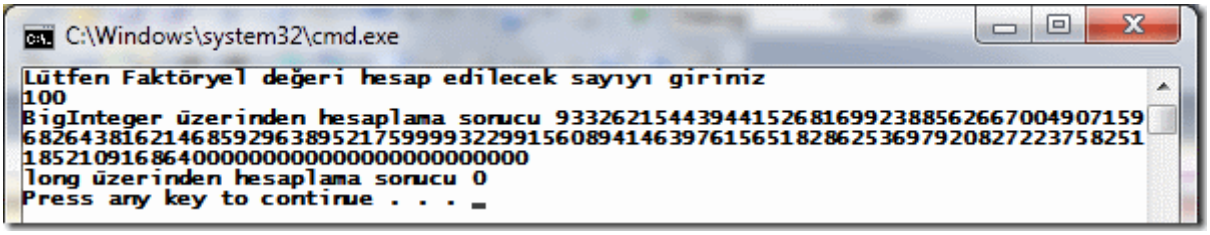
Herşey yolunda görünüyor.

21!



Uppsss!!! :S Bir terslik var sanki. 21 sayısının faktöryel değeri için **Int64** tipinden olan hesaplama negatif değer döndürdü. Oysaki **BigInteger** ile çalışan metodumuz olması gereken değeri döndürdü. Sanıyorum ki ne demek istediğimi gayet iyi anladınız ;) Peki olayı biraz daha büyütelim mi? örneğin **100** sayısının faktöryel değerini hesap etmek istediğimizi düşünelim.

100!



Volaaaaaa!!!! :D Oldukça büyük, kocaman bir rakam ile karşı karşıyayız. Ama doğru sonuç olduğunu ifade edebiliriz. Faktöryel hesaplamalarını kontrol etmek

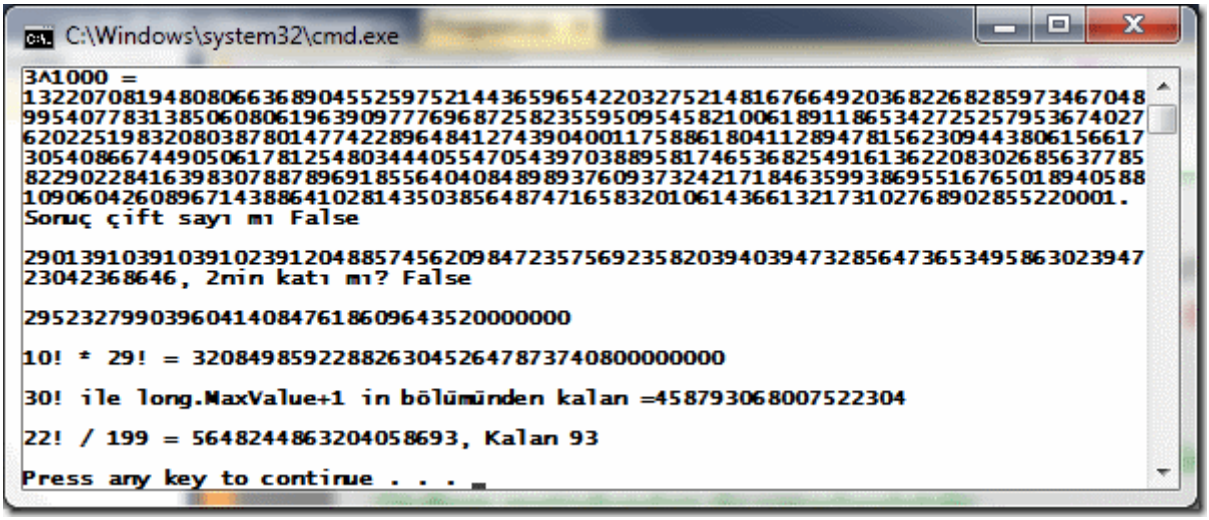
içinhttp://www.cs.uml.edu/~ytran/factorial.html adresindeki web tabanlı hesap makinesinden de yararlanabilirsiniz. Hatta zamanında elimizde **BigInteger** gibi bir kavram olmadığından, bu web sayfasına kod içinden sayısal değerleri **request** olarak gönderip sonuçlarını program ortamına aktarmayı bile denemiştim. Artık bu kadar kolaya kaçmanın gereği yok ;)

BigInteger tipi yukarıdaki kullanımı dışında sahip olduğu **static** metodlar sayesinde çok yüksek haneli sayılar ile kolayca çalışılabilmesine olanak sağlamaktadır. Aşağıdaki kod parçasında bir kaç örnek kullanıma yer verilmektedir.

```
using System;
using System.Numerics;

namespace HugeInteger
{
   class Program
   {
      static void Main(string[] args)
      {
         #region BigInteger diğer kullanım çeşitleri

         // 3ün 1000 üssü hesap edilmektedir.
         // Ayrca sonucun çift sayı olup olmadığı IsEven özelliği ile kontrol edilir.
         BigInteger number1=BigInteger.Pow(3, 1000);
         Console.WriteLine("3^1000 = \n{0}. Sonuç çift sayı mı
{1}\n",number1.ToString(),number1.IsEven);
         // String olarak girilen büyük bir sayısal değerin Parse edilmesi işlemi
gerçekleştirilir.
         // Ayrıca girilen sayının 2nin katı olup olmadığına IsPowerOfTwo özelliği ile
bakılmaktadır.
         BigInteger
number2=BigInteger.Parse("290139103910391023912048857456209847235756923582039403947328564736534958630239472304236864 6");
         Console.WriteLine("290139103910391023912048857456209847235756923582039403947328564736534958630239472304236864 6, 2nin katı mı?
{0}\n", number2.IsPowerOfTwo);

         // İki BigInteger değerinden büyük olanı Max metodu yardımıyla bulunabilir.
         BigInteger number3 = BigInteger.Max(Factorial(34), Factorial(33));
         Console.WriteLine("{0}\n",number3);

         // İki BigInteger sayının çarpılması için * operatörü haricinde Multiply metodundan
da yararlanılabilir.
         BigInteger number4 = BigInteger.Multiply(Factorial(10), Factorial(29));
         Console.WriteLine("10! * 29! = {0}\n",number4.ToString());
```

```
        // çok büyük iki sayının bölümünde kalan değerin hesaplanması için Remainder
fonksiyonundan yararlanılabilir
        BigInteger
number5=BigInteger.Remainder(Factorial(30),((BigInteger)(long.MaxValue))+1);
        Console.WriteLine("30! ile long.MaxValue+1 in bölümünden kalan
={0}\n",number5.ToString());

        // İki büyük sayının bölümünün sonucu ve bölümden kalan değerin elde edilmesi
için DivRem metodundan da yararlanılabilir
        BigInteger number6,remainderNumber;
        number6=BigInteger.DivRem(Factorial(22), 199, out remainderNumber);
        Console.WriteLine("22! / 199 = {0}, Kalan
{1}\n",number6.ToString(),remainderNumber.ToString());

        #endregion
    }
    static BigInteger Factorial(int value)
    {
        if (value == 0 || value == 1)
            return 1;
        else
            return value*Factorial(value-1);
    }
  }
}
```

Kod parçasında BigInterger türü ile ilişkili çeşitli örnek kullanımlar yer almaktadır. Uygulamanın çalışmasının sonucu aşağıdaki gibi olacaktır.


C:\Windows\system32\cmd.exe
3^1000 =
1322070819480806636890455259752144365965422032752148167664920368226828597346704899540778313850608061963909777696872582355950954582100618911865342725257953674027620225198320803878014774228964841274390400117588618041128947815623094438061566173054086674490506178125480344405547054397038895817465368254916136220830268563778582290228416398307887896918556404084898937609373242171846359938695516765018940588109060426089671438864102814350385648747165832010614366132173102768902855220001.
Sonuç çift sayı mı False

290139103910391023912048857456209847235756923582039403947328564736534958630239472304236864 6, 2nin katı mı? False

29523279903960414084761860964352000000 0

10! * 29! = 32084985922882630452647873740800000000

30! ile long.MaxValue+1 in bölümünden kalan =458793068007522304

22! / 199 = 5648244863204058693, Kalan 93

Press any key to continue . . .
```

**3ün 1000nci üssünün** sonucunun dahi ele alınabildiği görülmektedir. üstelik **string** tabanlı olarak gelen çok yüksek haneli bir değer, kolay bir şekilde sayısal olarak ele alınabilmiştir. Diğer yandan **çok büyük sayılar üzerinden bölme, bölmeden kalan sonucun hesap**
```

edilmesi veya **primitive** bir sayısalın **maksimum değerinin 1 fazlasının** ele alınması mümkün hale getirilmiştir.

BigInteger türü özellikle yüksek değerli sayıların ele alındığı matematiksel hesaplamaların olduğu senaryolarda, görüntü işleme programlarında, finansal analiz yapan uygulamalarda vb… bizlere önemli kolaylıklar ve avantajlar sunmaktadır. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HugeInteger.rar (22,79 kb) **[örnek Visual Studio 2010 Ultimate sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Non-Persisted Memory Mapped Files (2010-12-20T00:00:00)

.net framework 4.0,c# 4.0,memory-mapped files,visual studio 2010 ultimate,

Merhaba Arkadaşlar,

Akşam saatleriydi. Sıcak ama güneşin fazla görünmediği bir yaz gününün sonları yaklaşmaktaydı. Gün boyu güneşi İstanbul' dan saklayan bulutlar yavaş yavaş parçalanmaya başlamıştı. Aralardan güneşin ışınları hafif kızılımsı bir renk ile göğü süslemekteydi. Ancak yine de sert esen rüzgar ve hızla hareket eden bulutlar adeta fırtına öncesi sessizliğin bir habercisiydi. Aile efradı tatildeydi ve çalışma odamda tek başıma dışarıya doğru bakerken geceyi nasıl devam ettirmem gerektiğini planlıyordum.

Zaman ilerledi ve gecenin zifiri karanlığında güçlü bir gök gürültüsü duyuldu. Belli ki fırtınalı, zor bir gece olacaktı. Kimsenin açık denizde veya havada olmak istemeyeceği bir gece.

Kısa bir süre sonra yağmaya başlayan yağmur ve cama vuran damlaların sesi eşliğinde kendimi bilgisayarımın başında buldum. Odayı kaplayan loş lamba ışığı konstanrasyonu en üst seviye çıkartırken, bilgisayardan gelen hafif müzik nameleri, bloğumun girişi için gerekli cümleleri hazırlamama yardımcı oluyordu. Rota çizilmişti. Geceyi güzel bir blog yazısı ile tamamlamak.

Hatırlayacağınız üzere bir önceki yazımızda **.Net Framework 4.0** sürümüne dahil edilen Memory-Mapped Files kavramını incelemeye başlamıştık. İncelememizde ele aldığımız örnekte ise **Persisted** modeli göz önüne almıştık. Bu modelde bellek üzerine açılan içerikler, fiziki disk üzerinde yer alan dosyalar ile doğrudan ilişkilidir. Yani sanal belleğe açılan görünümler(Views), fiziki disk üzerindeki dosyanın belirli bir bölümü veya tamamıdır.

Bir de **Non-Persisted** modeli söz konusudur. Bu modelde **Sanal Bellek(Virtual Memory)** üzerinde oluşturulan bir dosya içeriği söz konusudur. **Non-Persisted** modelinde de, aynen **Persisted** modelinde olduğu gibi, bellek içeriklerinin ve buna bağlı olarak oluşturulan görünümlerinin birden fazla **Process** tarafından ele alınması mümkündür. İşte bu yazımızda söz konusu modeli ele alan basit bir örnek geliştirmeye çalışıyor olacağız. **MemoryMappedFile** tipinin static **CreateNew** veya **CreateOrOpen** metodları, **Non-Persisted** modeline göre dosya oluşturulmasını sağlamaktadırlar. Elbette birden fazla **Process'** in bellek bölgesi üzerindeki aynı içeriğin tamamını veya belirli parçalarını kullanabilmeleri için **MemoryMappedViewStream** nesne örneklerinden yararlanmaları gerekmektedir. Bir başka deyişle içeriğe ait **görünümler(View)** ele alınmalıdır.

örnek çözümümüzde, **Visual Studio 2010 Ultimate** ortamında geliştirilmiş üç **Console** uygulaması bulunmaktadır. Bu uygulamalardan ilki bellek üzerinde bir **Mapped-File**oluşturmakta ve içeriğine örnek veri yazmaktadır. İkinci uygulama da benzer şekilde, aynı bellek bölgesindeki dosyaya farklı bir içerik yazdırmaktadır. üçüncü uygulama ise **Mapped-File** içeriğini okumak üzere tasarlanmıştır. çözümün amacı **Virtual Memory** üzerine açılmış olan bir **Mapped-File** ile çalışan üç farklı **Process**' in okuma ve yazma işlemlerini icra etmesidir. İlk olarak **Application1** isimli örnek uygulama kodlarına bakalım.

```
using System;
using System.IO.MemoryMappedFiles;
using CommonLib;

namespace Application1
{
   class Program1
   {
      static void Main(string[] args)
      {
         Console.Title = "Program 1";
         string content = String.Empty;
         Console.WriteLine("Günün mesajını giriniz");
         content = Console.ReadLine();

         using (MemoryMappedFile mappedFile =
MemoryMappedFile.CreateNew("mappedImage", 1024))
         {
            byte[] array = null;
            using (MemoryMappedViewStream view =
mappedFile.CreateViewStream())
            {
               array = Helper.CreateByteArray(content);
               view.Write(array, 0, array.Length);
```

```
            }
            Console.WriteLine("Program 1 {0} byte uzunluğunda içeriği Non Persisted
Memory-Mapped dosyaya yazdı.\nProgram 2' yi çalıştırın", array.Length);
            Console.ReadLine();
        }
    }
  }
}
```

örnek, kullanıcıdan günün anlam ve önemine dair bir mesaj istemektedir.
Sonrasında **1024 byte** uzunluğunda bir **Memory-Mapped File** oluşturulmaktadır. Bu oluşturulma işleminin**CreateNew** metodu ile yapıldığına dikkat edelim. **Virutal Memory** üzerinde bu Process için açılan bölgeye veri yazmak içinse **MemoryMappedViewStream** tipine ait bir nesne örneğinden yararlanılmaktadır. Söz konusu nesnenin **Write** metodu kullanılarak **array** isimli değişkenin **byte** içeriği, söz konusu alana yazılmaktadır. Burada **CommonLib** isimli **Class Library** içerisinde yer alan **Helper static** sınıfına ait ve aşağıdaki kod içeriğine sahip **CreateByteArray** metodundan yararlandığımızı da belirtelim.

```
namespace CommonLib
{
  public static class Helper
  {
    public static byte[] CreateByteArray(string content)
    {
      char[] charArray=content.ToCharArray();
      byte[] byteArray=new byte[charArray.Length];

      for (int i = 0; i < charArray.Length; i++)
      {
        byteArray[i] = (byte)charArray[i];
      }
      return byteArray;
    }
  }
}
```

Bu metod **string** olarak alınan içeriğin her bir karakterinin **byte** karşılığını topladığı bir diziyi geriye döndürmektedir. Tekrar çözümümüze dönelim. İkinci uygulamanın kod içeriği birincisine oldukça benzerdir.

```
using System;
using System.IO.MemoryMappedFiles;
using CommonLib;
```

```
namespace Application2
{
   class Program2
   {
      static void Main(string[] args)
      {
         Console.Title = "Program 2";
         string content = String.Empty;
         Console.WriteLine("Doğum yerinizi giriniz");
         content = Console.ReadLine();

         using (MemoryMappedFile mappedFile
= MemoryMappedFile.OpenExisting("mappedImage"))
         {
            byte[] array=null;
            using (MemoryMappedViewStream view
= mappedFile.CreateViewStream(50,100))
            {
               array = Helper.CreateByteArray(content);
               view.Write(array, 0, array.Length);
            }
            Console.WriteLine("Program 2 {0} byte uzunluğunda içeriği Non Persisted
Memory-Mapped dosyaya yazdı.\nProgram 3' ü çalıştırın", array.Length);
            Console.ReadLine();
         }
      }
   }
}
```

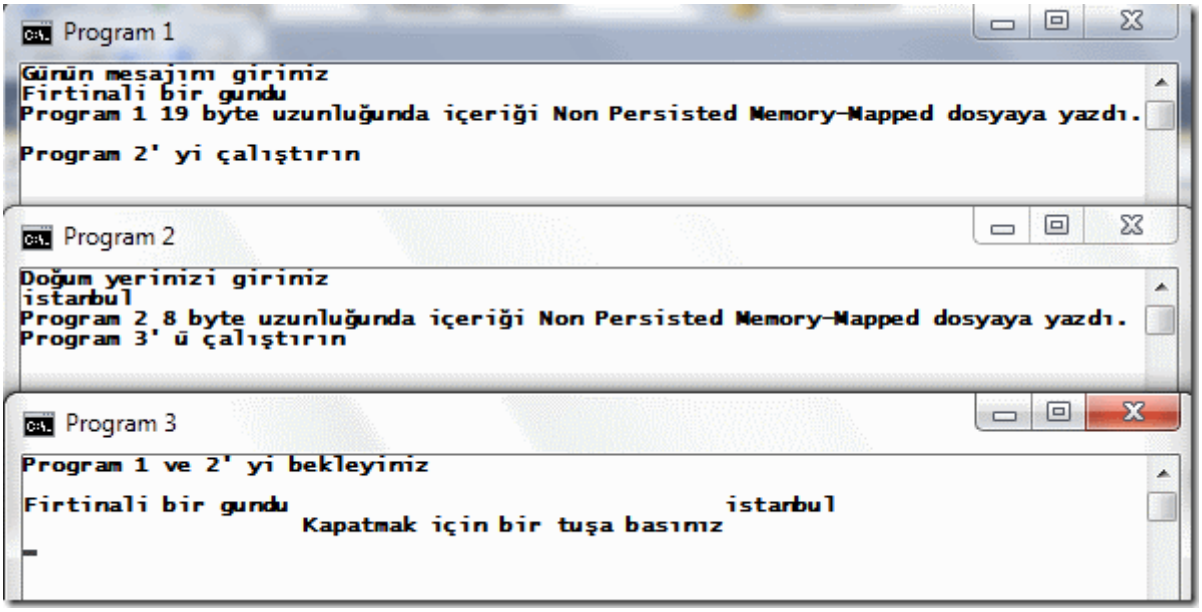
Bu sefer kullanıcıdan doğduğu yer bilgisi istenmektedir. Ancak bir önceki uygulamadan farklı olarak bu kez, **MemoryMappedFile** tipinin **static OpenExisting** metodu kullanılmaktadır. Eğer**mappedImage** ismiyle işaret edilebilen bir **Mapped-File** söz konusu ise*(ki işaret edilemediği durumda bir Exception söz konusu olacaktır)* yine bir **MemoryMappedViewStream** nesnesi oluşturulmakta ve söz konusu bellek bölgesinin **50nci byte'** ından itibaren **100 byte'** lık olan bölgeye veri yazabilmek için bir kanal oluşturulmaktadır. Yardımcı **CreateByteArray** metodu ile buraya doğum yeri bilgisi yazdırılmaktadır.

Gelelim 3ncü programın kod içeriğine.

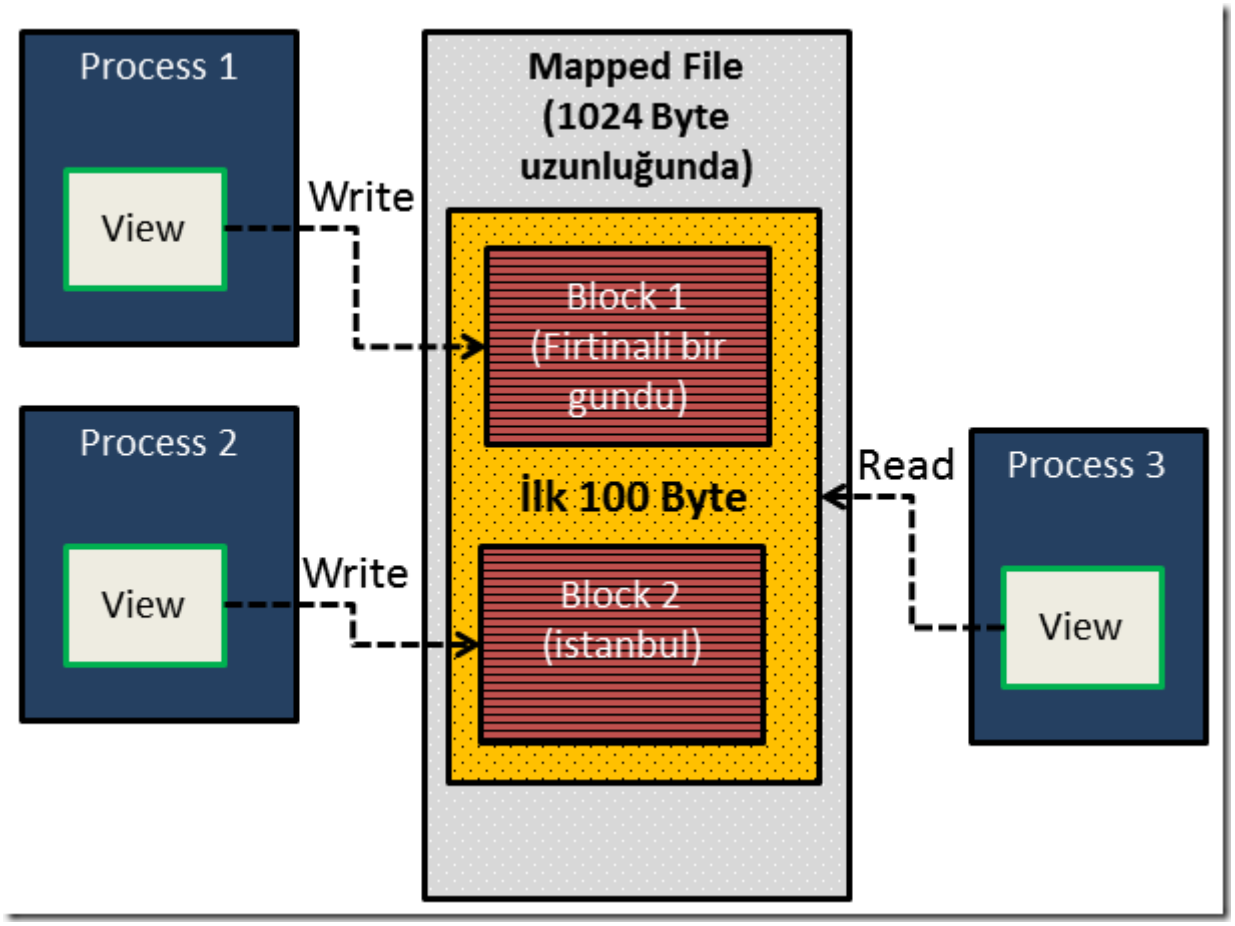
```
using System;
using System.IO.MemoryMappedFiles;

namespace Application3
{
```

```
   class Program3
   {
      static void Main(string[] args)
      {
         Console.Title = "Program 3";
         Console.WriteLine("Program 1 ve 2' yi bekleyiniz");
         Console.ReadLine();

         using (MemoryMappedFile mappedFile =
MemoryMappedFile.OpenExisting("mappedImage"))
         {
            using (MemoryMappedViewStream view = mappedFile.CreateViewStream())
            {
               byte[] array=new byte[100];
               view.Read(array, 0, 100);
               for (int i = 0; i < 100; i++)
               {
                  Console.Write((char)array[i]);
               }
            }
            Console.WriteLine("Kapatmak için bir tuşa basınız");
            Console.ReadLine();
         }
      }
   }
}
```

üçüncü uygulamanın tek bir amacı vardır. O da **mappedImage** ismi ile eşleşen **Mapped-File** içeriğinin ilk **100 byte'** lık kısmını okumak. Bu kod içeriklerini geliştirdikten sonra sırasıyla 1nci, 2nci ve 3ncü uygulamaları çalıştırdığımızda, aşağıdaki ekran görüntüsündekine benzer sonuçları elde ettiğimizi görebiliriz.



Görüldüğü üzere ilk iki uygulama söz konusu **Memory-Mapped File** üzerine bir takım **byte** içeriklerini yazmaktadır. 3ncü program ise ilk iki programın yazdığı verileri başarılı bir şekilde okumaktadır. çözümümüzde yer alan uygulamaların ne yaptığı aşağıdaki grafik ile temsil edilmeye çalışılmaktadır.



Non-Persisted dosyalar fiziki disk üzerinde bir dosyayı işaret etmediklerinden, örnekte üretilen **Memory Mapped-File** içerikleri, programların kapanması sonrasında doğal olarak yok olacaktır. Bu yok olma işleminde **Garbage Collector** devreye girmekte ve gerekli çok toplaması işlemlerini üstlenmektedir. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

NonPersistedMemoryMappedFiles.rar (80,65 kb) **[örnek Visual Studio 2010 Ultimate sürümünde geliştirilmiş ve test edilmiştir]**

Netspecter Takipte - Object Initializer Deyip Geçmemek Lazım (2010-12-19T23:34:00)

c#,c# 3.0,c# temelleri,

Merhaba Arkadaşlar,

Puslu bir sonbahar akşamında detektif **Netspecter** odasında sessiz sakin oturmaktadır. Loş bir ortama neden olan gece lambasının yeşil cam aksamı altından oda içerisindeki tozların sessiz ve sakin akışı bir yana, **Netspecter**' ın kafasında masasına yeni gelen dosya ile ilişkili soru işaretleri koşup durmaktadır.

Sıkıntılı geçen bir kaç saat sonrasında aniden telefon çalar. ölüm sessizliği içerisinde olan odanın neredeyse canlanmasına neden olan bir çalıştır bu. Ahizeyi ancak bir kaç seferden sonra fark edip kulağına götüren **Netspecter**, karşısında acı çektiği belli olan bir inleme ile irkilir.

Diğer ses : Objjj….eeeccttt!!!

Netspector : Kimsiniz

Diğer ses : Obb…ect!!!

Netspector : Oba makarnası mı? Nalo, nalooo…Anlamıyorum. Etecer mi?

Diğer ses : Aghhh!!!

Netspector : Ha ağrı kesici?

Diğer ses : Object Iniiittt….rrrr!!!

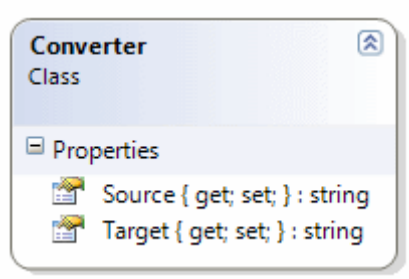
Diğer taraf : Dıt dıt dıt dıtttt!!!

Derken telefon sesi aniden kesilir. **Netspector** hemen sandalyesinde çabucak doğrulur, fotör şapkasını takar ve kapıdan hızla çıkar. Sevgili kedisi **CAD** ise bu telaşı umursamadan yemek kabındaki sütünü içmeye devam etmekte ve her içişten sonra patilerini temizlemektedir. Aslında **Netspector'** ın kafasındaki güzergah bellidir. Şehir merkezinde ki büyük **MSDN**kütüpahnesine uğrayacak ve **Object Initializer** ile ilgili bir kaç soru sorup olayı çözmeye çalışacaktır.

Asıl Mevzu

C# 3.0 ile birlikte gelen önemli yeniliklerden birisi de **Object Initializers** kullanımı idi. Bu kullanım sayesinde özellikle **LINQ(Language Integrated Query)** sorgularında **Anonymous Type** üretiminin mümkün hale gelmesi de sağlanmaktaydı. Dolayısıyla her zaman ifade ettiğimiz gibi bu yenilik, başka bir yeniliğin yapılabilmesi için getirilmiş bir yenilikti 😉

Tabi o zamandan beri **Object Initializers** kavramı üzerinde çok fazla yazıp çizdim ama yine de pek fazla derinlerine girmediğimi ya da olaya farklı gözeler ile bakmadığımı farkettim. Bu nedenle konuyu farklı bir yaklaşım ile ele almanın daha doğru olacağına karar verdim. Dilerseniz kavramın derinliklerine inerken örnek bir kod parçası ile küçük bir başlangıç yapmaya çalışalım. Bu amaçla **Visual Studio 2010** ortamında aşağıdaki **Converter** isimli sınıfı kullanan bir **Console** uygulamasını göz önüne alıyor olacağız.



örnek program kodunu ise aşağıdaki gibi geliştirebiliriz.

```
namespace CaseObjectInitializers
{
   class Program
   {
      static void Main(string[] args)
      {
         //Object Initializer kullanarak nesneyi başlatmak
         Converter cvrtr = new Converter { Source = "c:\\source.xls", Target =
"c:\\target.txt" };

         // Klasik yöntem. önce varsayılan yapıcı metod çağrısı, ardından özelliklere değer
atama yolu ile başlatmak
```

```
        Converter cvrt2 = new Converter();
        cvrt2.Source = "c:\\source.xls";
        cvrt2.Target = ":\\target.text";
      }
   }

   class Converter
   {
      public string Source { get; set; }
      public string Target { get; set; }
   }
}
```

Bu kod parçasında dikkat edileceği üzere **Converter** tipinden iki farklı nesne örneklenmesi yapılmaktadır. **cvrtr** isimli ilk değişkenin üretimi sırasında **object initializer** tekniği kullanılmış ve süslü parantezler içerisinde **public** olan söz konusu tipe ait **Source** ve **Target** özelliklerine bazı değerler verilmiştir. **cvrt2** isimli değişkenin oluşturulma şekli ise tam bir klasiktir İlk olarak **varsayılan yapıcı(Default Constructor)** metoddan yararlanılmış ve sonrasında örnek nesne üzerinden **Source** ve **Target** özelliklerine değerleri atanmıştır. çok güzel. Güzel ama ilk örnekleme işleminde kullanılan **Object Initializer** tekniği aslında bir sihir mi uygulamıştır?

Detektifin konuyu çözmesi için ipuçlarını daha iyi görebileceği bir aygıta ve örneğe ihtiyacı vardır. Karanlık odayı loş bir şekilde aydınlatan ışığın altında düşünürken, aklına parlak bir fikir gelir. Mikroskop olarak daha bir kaç gün öncesinde kermesten aldığı eski pentium işlemcili makinede yer alan ILDASM aracını, örnek olarakta makdulun salonda bıraktığı Console uygulamasına ait exe çıktısını kullanacaktır.

ILDASM(Intermediate Language DisASeMbler Tool) aracı yardımıyla gerekli **IL** çıktısına baktığımızda aşağıdaki sonuçlar ile karşılaşırız.

```
.method private hidebysig static void  Main(string[] args) cil managed
{
 .entrypoint
 // Code size       64 (0x40)
 .maxstack  2
 .locals init ([0] class CaseObjectInitializers.Converter cvrtr,
          [1] class CaseObjectInitializers.Converter cvrt2,
          [2] class CaseObjectInitializers.Converter '<>g__initLocal0')
 IL_0000:  nop
 IL_0001:  newobj     instance void CaseObjectInitializers.Converter::.ctor()
 IL_0006:  stloc.2
 IL_0007:  ldloc.2
 IL_0008:  ldstr      "c:\\source.xls"
```

```
  IL_000d:  callvirt   instance void
CaseObjectInitializers.Converter::set_Source(string)
  IL_0012:  nop
  IL_0013:  ldloc.2
  IL_0014:  ldstr      "c:\\target.txt"
  IL_0019:  callvirt   instance void
CaseObjectInitializers.Converter::set_Target(string)
  IL_001e:  nop
  IL_001f:  ldloc.2
  IL_0020:  stloc.0
  IL_0021:  newobj     instance void CaseObjectInitializers.Converter::.ctor()
  IL_0026:  stloc.1
  IL_0027:  ldloc.1
  IL_0028:  ldstr      "c:\\source.xls"
  IL_002d:  callvirt   instance void CaseObjectInitializers.Converter::set_Source(string)
  IL_0032:  nop
  IL_0033:  ldloc.1
  IL_0034:  ldstr      ":\\target.text"
  IL_0039:  callvirt   instance void CaseObjectInitializers.Converter::set_Target(string)
  IL_003e:  nop
  IL_003f:  ret
} // end of method Program::Main
```

Dikkat edileceği üzere **IL_0001** satırından **Converter** tipine ait **varsayılan yapıcı metod(Default Constructor)** çağırılmıştır. **IL_000d** numaralı satırda ise **Source** özelliği için **set** metodunun çağırıldığı görülmektedir. Diğer yandan benzer işlem **IL_0019** numaralı satırda da yapılmaktadır. özetlemek gerekirse **Object Initializer** kullandığımızda aslında klasik yöntem de uygulanan önce yapıcı metodun çağırılması ve sonrasında özelliklere değer atanması işlemi **IL** tarafında aynen yapılmaktadır. Hatta koleksiyonları **initializer** ile başlatttığımız durumlarda da benzer bir üretim modeli söz konusu olacaktır. Bu durumu da analiz edersek yerinde olur mu? Olur Bu amaçla uygulamamıza aşağıdaki kod satırlarını eklediğimizi düşünelim.

```
List<Converter> converters = new List<Converter>
{
   new Converter{Source="c:\\source1.xls",Target="C:\\target1.txt"},
   new Converter{Source="c:\\source2.xls",Target="C:\\target2.txt"},
   new Converter{Source="c:\\source3.xls",Target="C:\\target3.txt"},
};
```

Bu sefer hem **List<T>** tipi için hem de içerisine eklenen her bir **Converter** tipi için yapılan örnekleme işlemlerinde **Object Initializer** tekniği kullanılmıştır. Bildiğiniz üzere **IEnumerable**arayüzünü uygulayan ve **Add** metodunu içeren koleksiyon tipleri için de **Object Initializer** tekniğinden yararlanılabilmektedir. Peki ya **IL** çıktısı?

```
.method private hidebysig static void  Main(string[] args) cil managed
{
  .entrypoint
  // Code size       203 (0xcb)
  .maxstack  3
  .locals init ([0] class CaseObjectInitializers.Converter cvrtr,
           [1] class CaseObjectInitializers.Converter cvrt2,
           [2] class [mscorlib]System.Collections.Generic.List`1<class
CaseObjectInitializers.Converter> converters,
           [3] class CaseObjectInitializers.Converter '<>g__initLocal0',
           [4] class [mscorlib]System.Collections.Generic.List`1<class
CaseObjectInitializers.Converter> '<>g__initLocal1',
           [5] class CaseObjectInitializers.Converter '<>g__initLocal2',
           [6] class CaseObjectInitializers.Converter '<>g__initLocal3',
           [7] class CaseObjectInitializers.Converter '<>g__initLocal4')
  IL_0000:  nop
  IL_0001:  newobj     instance void CaseObjectInitializers.Converter::.ctor()
  IL_0006:  stloc.3
  IL_0007:  ldloc.3
  IL_0008:  ldstr      "c:\\source.xls"
  IL_000d:  callvirt   instance void CaseObjectInitializers.Converter::set_Source(string)
  IL_0012:  nop
  IL_0013:  ldloc.3
  IL_0014:  ldstr      "c:\\target.txt"
  IL_0019:  callvirt   instance void CaseObjectInitializers.Converter::set_Target(string)
  IL_001e:  nop
  IL_001f:  ldloc.3
  IL_0020:  stloc.0
  IL_0021:  newobj     instance void CaseObjectInitializers.Converter::.ctor()
  IL_0026:  stloc.1
  IL_0027:  ldloc.1
  IL_0028:  ldstr      "c:\\source.xls"
  IL_002d:  callvirt   instance void CaseObjectInitializers.Converter::set_Source(string)
  IL_0032:  nop
  IL_0033:  ldloc.1
  IL_0034:  ldstr      ":\\target.text"
  IL_0039:  callvirt   instance void CaseObjectInitializers.Converter::set_Target(string)
  IL_003e:  nop
  IL_003f:  newobj     instance void class
[mscorlib]System.Collections.Generic.List`1<class
CaseObjectInitializers.Converter>::.ctor()
  IL_0044:  stloc.s    '<>g__initLocal1'
  IL_0046:  ldloc.s    '<>g__initLocal1'
  IL_0048:  newobj     instance void CaseObjectInitializers.Converter::.ctor()
  IL_004d:  stloc.s    '<>g__initLocal2'
```

```
  IL_004f:  ldloc.s    '<>g__initLocal2'
  IL_0051:  ldstr      "c:\\source1.xls"
  IL_0056:  callvirt   instance void
CaseObjectInitializers.Converter::set_Source(string)
  IL_005b:  nop
  IL_005c:  ldloc.s    '<>g__initLocal2'
  IL_005e:  ldstr      "C:\\target1.txt"
  IL_0063:  callvirt   instance void
CaseObjectInitializers.Converter::set_Target(string)
  IL_0068:  nop
  IL_0069:  ldloc.s    '<>g__initLocal2'
  IL_006b:  callvirt   instance void class
[mscorlib]System.Collections.Generic.List`1<class
CaseObjectInitializers.Converter>::Add(!0)
  IL_0070:  nop
  IL_0071:  ldloc.s    '<>g__initLocal1'
  IL_0073:  newobj     instance void CaseObjectInitializers.Converter::.ctor()
  IL_0078:  stloc.s    '<>g__initLocal3'
  IL_007a:  ldloc.s    '<>g__initLocal3'
  IL_007c:  ldstr      "c:\\source2.xls"
  IL_0081:  callvirt   instance void
CaseObjectInitializers.Converter::set_Source(string)
  IL_0086:  nop
  IL_0087:  ldloc.s    '<>g__initLocal3'
  IL_0089:  ldstr      "C:\\target2.txt"
  IL_008e:  callvirt   instance void
CaseObjectInitializers.Converter::set_Target(string)
  IL_0093:  nop
  IL_0094:  ldloc.s    '<>g__initLocal3'
  IL_0096:  callvirt   instance void class
[mscorlib]System.Collections.Generic.List`1<class
CaseObjectInitializers.Converter>::Add(!0)
  IL_009b:  nop
  IL_009c:  ldloc.s    '<>g__initLocal1'
  IL_009e:  newobj     instance void CaseObjectInitializers.Converter::.ctor()
  IL_00a3:  stloc.s    '<>g__initLocal4'
  IL_00a5:  ldloc.s    '<>g__initLocal4'
  IL_00a7:  ldstr      "c:\\source3.xls"
  IL_00ac:  callvirt   instance void
CaseObjectInitializers.Converter::set_Source(string)
  IL_00b1:  nop
  IL_00b2:  ldloc.s    '<>g__initLocal4'
  IL_00b4:  ldstr      "C:\\target3.txt"
  IL_00b9:  callvirt   instance void
CaseObjectInitializers.Converter::set_Target(string)
```

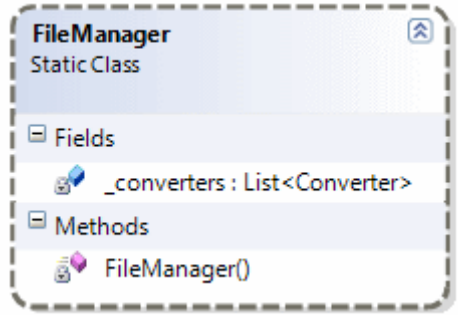
```
  IL_00be:  nop
  IL_00bf:  ldloc.s    '<>g__initLocal4'
  IL_00c1:  callvirt   instance void class
[mscorlib]System.Collections.Generic.List`1<class
CaseObjectInitializers.Converter>::Add(!0)
  IL_00c6:  nop
  IL_00c7:  ldloc.s    '<>g__initLocal1'
  IL_00c9:  stloc.2
  IL_00ca:  ret
} // end of method Program::Main
```

Haydi gelin bir çılgınlık yapalım ve bu **IL** kodunu okumaya çalışalım Dikkat edileceği üzere **IL_003f** satırında **List<Converter>** tipi için **varsayılan yapıcı metod** çağrısı gerçekleştirilmektedir. Dolayısıyla bu satırda ilgili koleksiyona ait bir nesne örneğinin ürettirildiğini düşünebiliriz. Diğer yandan **IL_0048** numaralı satırda bir **Converter** tipi örneklemesi için varsayılan yapıcı metod çağrısı söz konusudur. **IL_0056** ve **IL_0063** numaralı satırlarda ise, **IL_0048'** de üretilen **Converter** nesnesine ait özelliklerin(Source ve Target) değerleri atanmaktadır. **IL_006b** satırında ise **IL_0048**' de üretilen ve **IL_0056** ile **IL_0063** satırlarında sırasıyla **Source** ve **Target** özelliklerine değer atanan **Converter** nesne örneğinin**IL_003f** satırında örneklenen **List<Converter>** tipli koleksiyon örneğine **Add** metodu ile eklendiği gözlemlenmektedir. Bu **IL** çağrı akışı diğer iki **Converter** nesne örneği için de geçerli olacaktır.

Sanıyorum ki **Object Initializer** tekniği hakkında biraz daha derin fikir sahibi olmaya başladık. Yeter mi? Yetmez. Bakın daha neler var?

beforefieldinit

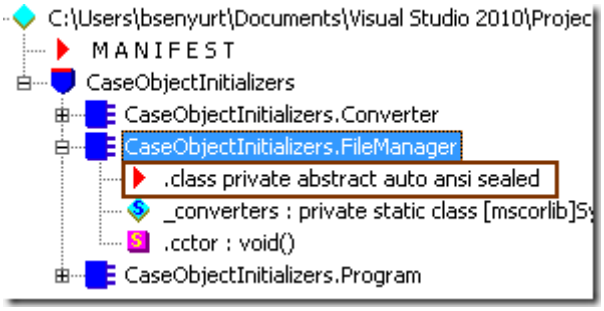
örneğimize aşağıdaki sınıf örneğini eklediğimi düşünelim.




```
static class FileManager
{
   static List<Converter> _converters = new List<Converter>();
```

```
    static FileManager()
    {
        Converter c1 = new Converter();
        c1.Source = "c:\\sourcex.xls";
        c1.Target = "c:\\targetx.txt";
        _converters.Add(c1);

        Converter c2 = new Converter();
        c2.Source = "c:\\sourcez.xls";
        c2.Target = "c:\\targetz.txt";
        _converters.Add(c2);
    }
}
```

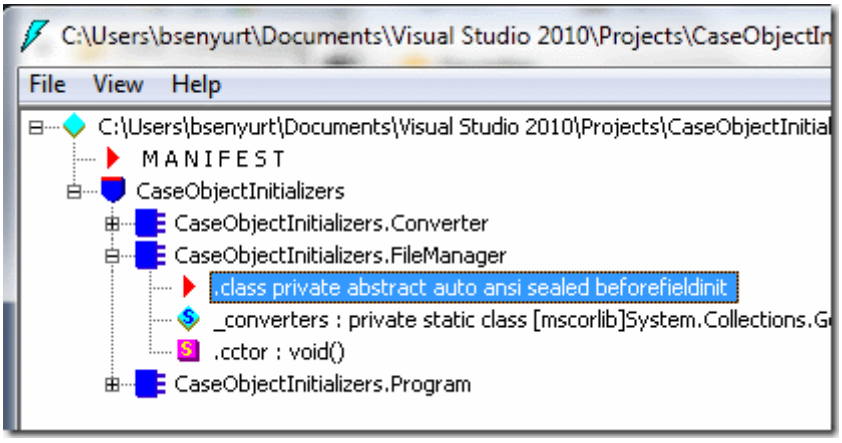
FileManager static tipli bir sınıf olmakla birlikte **_converters** isimli **List<Converter>** koleksiyonu türünden bir alan içermekedir. Söz konusu alan tanımlandığı yerde **new** operatörü ile örneklenmiş ve ilk değerlerinin atanması için **FileManager static** yapıcı metodu kullanılmıştır. Söz konusu sınıfın **IL** çıktısına baktığımızda özellikle sınıf tanımının aşağıdaki şekilde olduğu gibi yapıldığı görülecektir.



Şimdi sınıf kodunu biraz daha değiştirelim.

```
static class FileManager
{
    static List<Converter> _converters = new List<Converter>
    {
        new Converter{Source="c:\\sourcesx.xls",Target="c:\\targetx.txt"},
        new Converter{Source="c:\\sourcesz.xls",Target="c:\\targetz.txt"}
    };
}
```

FileManager tipinin yeni versiyonunda **_converters** isimli koleksiyon örneğini oluşturmak için **static yapıcı metod** yerine **object initializer** kullandığımızı görmekteyiz. Bu durumda uygulamanın **IL** çıktısına baktığımızda **FileManager** tipi için aşağıdaki tanımlamanın yapıldığına şahit oluruz.



İki örneğe ait tanımlamalara baktığımızda **beforefieldinit** isimli bir değerin kullanıldığı fark edilmektedir. Aslında bu değerin uygulama performansını arttırıcı bir etkisi olduğunu ifade edebiliriz. Normalde static yapıcı metod kullandığımızda, tip içerisinde yer alan tüm static değişkenlerin ilk erişimlerinden önce örneklenip örneklenmediklerine dair bir kontrol işlemi uygulanmaktadır. Ancak son kod parçamızda olduğu gibi **Object Initializer** kullanır ve static yapıcı metodu dışarıda bırakırsak, bu durumda sınıf söz konusu**beforrefiledinit** özelliği ile işaretlenerek ilgili kontrol işleminin atlanması sağlanır. Bu da azıcık bile olsa performans kazanımı anlamına gelmektedir

Görüldüğü üzere **Object Initializer** karvramını sadece kodun yazımını kısaltan bir yenilik şeklinde düşünmemek gerekir. Ancak yazım ve kod okunurluğunu kolaylaştırdığı da bir gerçektir. Bunun için aşağıdaki tabloya bakmanız sanırım yeterli olacaktır

Klasik Yaklaşım	**Object Initializer ile Yaklaşım**

using System.Collections.Generic;
 	
namespace CaseObjectInitializers
{
  class Program
  {
  static void Main(string[] args)
  {
  List<Book> books = new List<Book>();
 	
  Book newBook = new Book();
  newBook.Id = 1;
  newBook.Name = "Dick Tracy Maceraları 1";
  newBook.Summary = "İlk bölüm maceraları";
  newBook.ListPrice = 10;
  newBook.Authors = new
	using System.Collections.Generic;
	
	namespace CaseObjectInitializers
	{
	  class Program
	  {
	  static void Main(string[] args)
	  {
	  List<Book> books = new List<Book>
	  {
	  new Book{Id=1, Name="Dick Tracy Maceraları 1", Summary="İlk bölüm maceraları", ListPrice=10
	  , Authors=new List<Author>{
	  new Author{Id=1, Name="Dick", Surname="Tracy"}
	  }
	  },
	  new Book{Id=2,Name="Uygulamalı

```
List<Author>();

            Author newAuthor = new
Author();
            newAuthor.Id = 1;
            newAuthor.Name = "Dick";
            newAuthor.Surname = "Tracy";
            newBook.Authors.Add(newAutho
r);

            books.Add(newBook);
            newBook = new Book();
            newBook.Id = 2;
            newBook.Name = "Uygulamalı
WCF";
            newBook.Summary = "İlk
denemeler";
            newBook.ListPrice = 34.49M;
            newBook.Authors = new
List<Author>();

            newAuthor = new Author();
            newAuthor.Id = 3;
            newAuthor.Name = "Burak S.";
            newAuthor.Surname = "Şenyurt";
            newBook.Authors.Add(newAutho
r);

            newAuthor = new Author();
            newAuthor.Id = 4;
            newAuthor.Name = "Ingo";
            newAuthor.Surname =
"Rammer";
            newBook.Authors.Add(newAutho
r);

            books.Add(newBook);
        }
    }

    class Book
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public List<Author> Authors{ get;
```

```
WCF", Summary="İlk denemeler",
ListPrice=34.59M
                ,Authors=new
List<Author>{
                    new Author{Id=3,Name="Burak
S",Surname="Şenyurt"},
                    new
Author{Id=4,Name="Ingo",Surname="Rammer
"}
                }
            }
        };
    }
  }

  class Book
  {
    public int Id { get; set; }
    public string Name { get; set; }
    public List<Author> Authors{ get; set; }
    public string Summary { get; set; }
    public decimal ListPrice { get; set; }
  }

  class Author
  {
    public int Id { get; set; }
    public string Name { get; set; }
    public string Surname { get; set; }
  }
}
```

set; }
public string Summary { get; set; }
public decimal ListPrice { get; set; }
}
 	
class Author
{
public int Id { get; set; }
public string Name { get; set; }
public string Surname { get; set; }
}
}	

book koleksiyonunun oluşturulması için **36** satır	book koleksiyonunun oluşturulması için **13** satır

Her iki örnek kod parçasında da **Book** tipinden nesne örnekleri taşıyan birer **List<T>** koleksiyonu oluşturulmakta ve doldurulmaktadır. **Book** tipi içerisinde **Author** tipinden de bir koleksiyon da yer almaktadır.
Tabi **Book** ve **Author** sınıfları için **aşırı yüklenmiş yapıcı metodlar(Overloaded Constructors)** olmadığını varsayıyoruz bu senaryo da. Kodun daha da kısalması, performans yönündeki minik fark göz önüne alındığında biraz daha önemsiz gibi duruyor. Yine de çok fazla sayıda kod parçası içeren projelerde kodun okunurluğunun da geliştiricinin psikolojisi üzerinde doğrudan etkili olduğunu ifade etmek isterim. Böylece geldik bir yazımızın daha sonuna. **Netspecter**' ın bir sonraki macerasında görüşünceye dek hepinize mutlu günler dilerim.

CaseObjectInitializers.rar (23,71 kb)

Debug Edilebilir Windows Service Geliştirmek (2010-12-19T22:34:00)

windows services,debug,

Merhaba Arkadaşlar,

Uzun süre önce dış kaynak(Outsource) olarak görev aldığım bir bankacılık uygulamasında **Windows Service** tabanlı entegrasyon işlemleri için görevlendirilmiştim. Herşeyden önce bu servislerin bankacılık uygulaması olması nedeniyle, farklı ve yabancı sistemleri de ilgilendiren iş adımları bulunmaktaydı. Bu sebepten söz

konusu **Windows Service** uygulamalarının hem kod içerikleri hem de iş kuralları oldukça karışık olabiliyordu. İlgili **Windows Service** örneklerinin geliştirilmesi bir yana, bunların test ortamına atılması ve sonuçlarının takip edilmesi ise başlı başına bir dertti

Genellikle servisin çalışma durumunu izlemek adına özellikle **Exception** bloklarında veya metod başlangıç ile bitiş noktalarında*(örneğin OnStart başında ve sonunda)* işletim sisteminin uygulamaya özel **Event Log'** larına bilgi atmaktaydım. Bu bilgileri atarken de durumun kritikliğine göre**Warning, Exclamation, Error** gibi hazır sistem ikonlarından yararlanıyor ve çalışma zamanındaki durumu analiz etmeye çalışıyordum.

Ancak bir developer için, kodun çalışma zamanındaki durumunu incelemenin sayısız yolu olduğu da bir gerçek. öyleki, **Debug** etmek bence en güzel yollardan birisi. Lakin bir **Windows Service** uygulamasının **Debug** edilmesi de sanıldığı kadar kolay değil İşte bu yazımızda internet üzerinden yaptığım araştırmalar sonucu öğrendiğim ve bir **Windows Service** uygulamasının nasıl **debug** edilebeceğine dair uygulanabilen yöntemlerden birisini ele alıyor olacağız. Olabildiğince basit bir şekilde anlatmaya gayret edeceğim bu vaka çalışmamızda, adım adım ilerliyor olacağız öyleyse gelin şu metal entegre üzerindeki böcekleri ayıklamaya çalışalım.

İlk olarak **Windows Service Application** tipinden bir uygulama oluşturmamız gerekiyor. Bu uygulama içerisinde yer alan **ControllerService** isimli **Windows Service** tipinin içeriği başlangıçta aşağıdaki gibidir. Söz konusu servis içerisindeki **Timer** örneğinin **10** saniyede bir çalıştırdığı olay metoduna göre, belirli bir klasördeki*(ki path bilgisi **App.config** dosyasından çekilmektedir)* dosyaların şifrelenmesi için bir akış çalıştırmaktadır ki aslında bunun konumuz için çok da büyük bir önemi yoktur

```
using System.Configuration;
using System.IO;
using System.ServiceProcess;
using System.Timers;

namespace FileControllerService
{
   public partial class ControllerService
      : ServiceBase
   {
      private string _path = null;
      private Timer _timer = null;

      public ControllerService()
      {
         InitializeComponent();
         _path = ConfigurationManager.AppSettings["Path"];
         _timer = new Timer(10000);
```

```
        _timer.Elapsed += new ElapsedEventHandler(_timer_Elapsed);
    }

    void _timer_Elapsed(object sender, ElapsedEventArgs e)
    {
        string[] files=Directory.GetFiles(_path);
        foreach (string file in files)
        {
            File.Encrypt(file);
        }
    }
    protected override void OnStart(string[] args)
    {
        _timer.Start();
    }

    protected override void OnStop()
    {
        _timer.Stop();
    }
  }
}
```

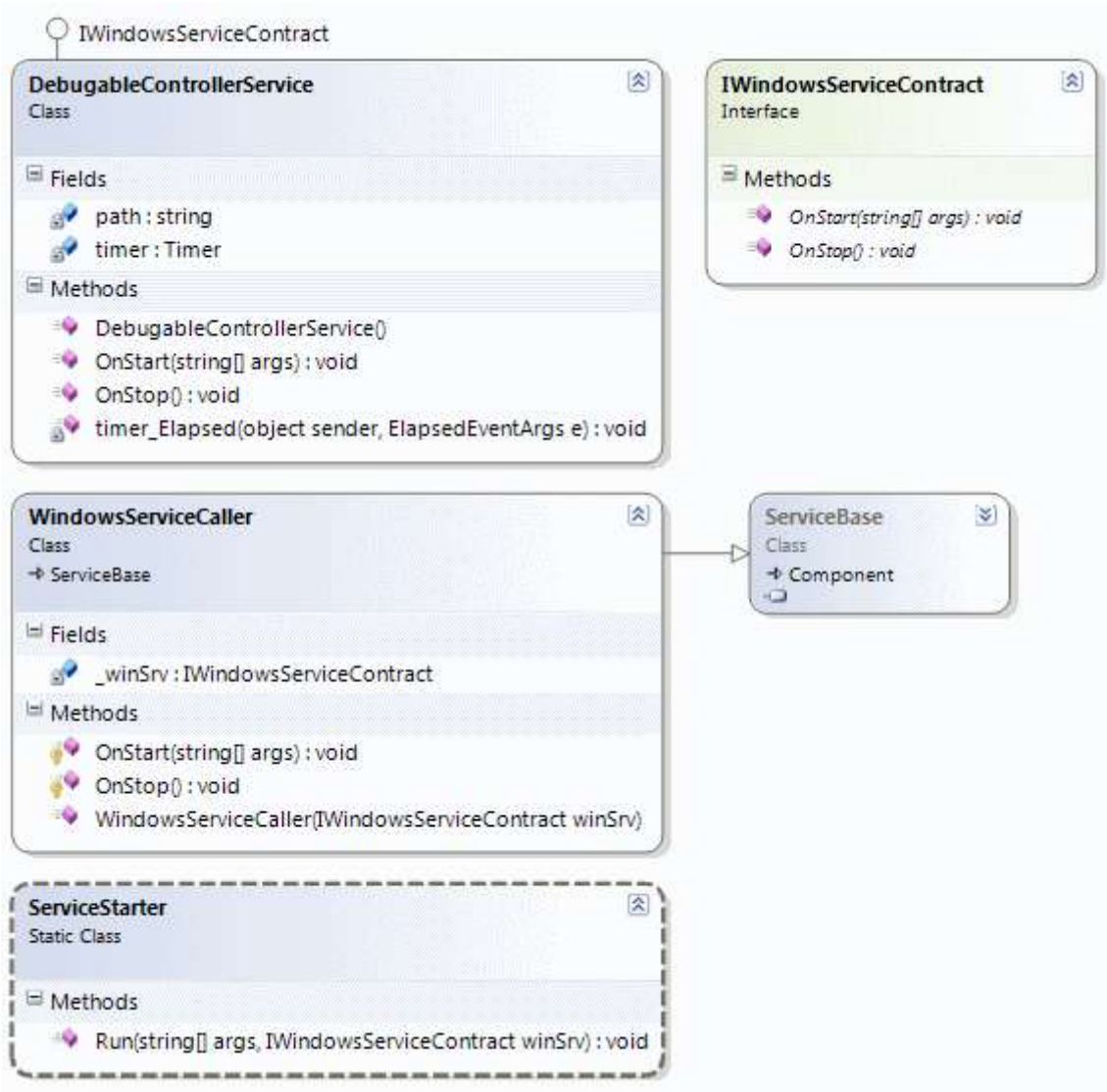
Bir **Windows Service** tipi temel olarak **ServiceBase** tipinden türemektedir. **Windows Service Application** proje şablonunda yer alan bu hazır uyarlamaya göre **Program** sınıfı içerisindeki **Main**metodunun kod yapısı da aşağıdaki gibi olacaktır.

```
using System;
using System.Configuration;
using System.ServiceProcess;
using InvestigationLib;

namespace FileControllerService
{
  static class Program
  {
    static void Main()
    {
      ServiceBase[] ServicesToRun;
      ServicesToRun = new ServiceBase[]
      {
        new ControllerService()
      };
      ServiceBase.Run(ServicesToRun);
    }
```

```
    }
}
```

Dikkat edileceği üzere Servisin çalıştırılması işini **ServiceBase** tipinin **static Run** metodu üstlenmektedir. Ne varki bu tip bir servisin varsayılan olarak **Debug** edilmesi mümkün değildir. Bir başka deyişle servise ait **OnStart** ve **OnStop** gibi metodların içerisine girilememektedir. Bu, özellikle karmaşık iş kuralları ve modelleri içeren **Windows Service** örnekleri düşünüldüğünde, geliştirme sürecini zorlaştıran ve uzatan bir durumun oluşması anlamına gelmektedir. Elbette servisin pek çok noktasından örneğin işletim sistemi **Event Log'**larına bilgi gönderilebilir veya bir **Text** dosyası içerisine servisin çalışması sırasındaki sürecin önemli adımlarına ait bilgiler*(mesela Exceptipon mesajları)* yazdırılabilir. Ancak değişken, parametre ve metod sayılarının iyiden iyiye fazlalaştığı, referans edilen kütüphanelerin çok olduğu bir modelde bu içerikleri izlemekte, üretmekte zor olmaktadır.

Aslında bir geliştirici için kodun yazılması ve test ortamına alınması esnasında **debug** edilebilyor olması son derece önemlidir. Şimdi dilerseniz yazımıza konu olan bu basit **Windows Service** örneğinin **debug** edilebilir bir versiyonunu nasıl geliştirebileceğimize bir bakalım Durumu anlamanın en kolay yolu bitmiş olan örneğin sınıf diagramına bakmak olacaktır*(Yani çözüme biraz tersten bakmamız gerekmektedir)* İşte çözümsel yaklaşıma ait sınıf diagramı görüntüsü.



Şimdi bu sınıf diagramını adım adım üretmeye başlayalım. İlk olarak bir **arayüz sözleşmesi(Interface Contract)** oluşturarak işe başlayacağız. Buradaki en büyük amacımız birden fazla **Windows Service** tipinin **Debug** edilebilir versiyonları için ortak bir sözleşme sunmaktır.

```
namespace InvestigationLib
{
   public interface IWindowsServiceContract
   {
      void OnStart(string[] args);
      void OnStop();
   }
}
```

IWindowsServiceContract arayüzünün içinde iki basit metod bildirimi olduğu görülmektedir. örneğimizi çok basit tutmak istediğimizden **OnPause, OnContinue, OnShutdown** gibi metod bildirimlerini göz önüne almadık. Ancak siz kendi

denemelerinizi yaparken veya bir ürün geliştirmesi sırasında bu tekniği uygularken mutlaka söz konusu diğer metodları da düşünmelisiniz. Aslında yaptığımız bir anlamda **ServiceBase** ile **override** edilen metodlara ait bir sözleşme bildiriminde bulunmaktan ibarettir.

Bu sözleşme bildiriminin ardından **ServiceBase** türevli olan ve **IWindowsServiceContract** arayüzünü implemente eden tiplerin fonksiyonelliklerini çağırabilen başka bir sınıf daha üretilir.**WindowsServiceCaller**…

```
using System.ServiceProcess;

namespace InvestigationLib
{
   public class WindowsServiceCaller
      :ServiceBase
   {
      IWindowsServiceContract _winSrv;

      public WindowsServiceCaller(IWindowsServiceContract winSrv)
      {
         _winSrv = winSrv;
      }
      protected override void OnStart(string[] args)
      {
         _winSrv.OnStart(args);
      }
      protected override void OnStop()
      {
         _winSrv.OnStop();
      }
   }
}
```

WindowsServiceCallar tipi için söylenebilecek iki önemli nokta vardır. Bunlardan ilki **ServiceBase** tipinden türemiş olması ve senaryomuza göre **OnStart** ile **OnStop** metodlarını **override**etmesidir. Diğer yandan ezilen **OnStart** ve **OnStop** metodları içerisinden yapılan çağrılar, tipin **yapıcı metodu(Constructor)** içerisinden alınan **IWindowsServiceCaller** arayüzünü implemente eden herhangibir nesne örneğine aittir. Bir başka deyişle bu tip, **OnStart** ve **OnStop** metod bildirimlerini sözleşme olarak sunan **IWindowsServiceContract** arayüzünü implemente eden bir tipin, çalışma zamanındaki asıl **OnStart** ve **OnStop** fonksiyonelliklerini kullanmaktadır. *(Oh oh ohhh!!! **Including** var, **inheritance** var, **Polimorphysm** var, **overriding** var ... **Nesne Yönelimli Programlama-Object Oriented Programming** temellerini hatırlamanın zamanı)*

Dolayısıyla bu tanımlamanın ardınan **debug** edilmesi gereken kod içeriğini taşıyan asıl servis tipi geliştirilir ki tesadüf bu olsa gerek bu
tipte **IWindowsServiceContract** arayüzünü implemente etmektedir

```
using System;
using System.IO;
using System.Timers;

namespace InvestigationLib
{
   public class DebugableControllerService
      :IWindowsServiceContract
   {
      string path=String.Empty;
      Timer timer = null;

      public DebugableControllerService()
      {
         timer = new Timer(10000);
         timer.Elapsed += new ElapsedEventHandler(timer_Elapsed);
      }

      void timer_Elapsed(object sender, ElapsedEventArgs e)
      {
         try
         {
            string[] files = Directory.GetFiles(path);
            foreach (string file in files)
            {
               File.Encrypt(file);
            }
         }
         catch (Exception excp)
         {
            //TODO:Handle Exception
            // Eğer Windows Service projesinin özelliklerinden Output Type -> Console
Application olarak seçilirse buradan Console penceresine de bilgi yazdırılabilir
         }
      }

      #region IWindowsServiceContract Members

      public void OnStart(string[] args)
      {
         path = args[0];
```

```
            timer.Start();
        }

        public void OnStop()
        {
            timer.Stop();
        }

        #endregion
    }
}
```

DebugableControllerService sınıfı aslında ilk başta **debug** edemediğimiz **ControllerService** sınıfının asıl fonksiyonelliklerini içermektedir. Tabi **ControllerService** tipinin yaptığı gibi **ServiceBase'** den türemek yerine **IWindowsServiceContract** arayüzünü uygulamaktadır.
Böylece **WindowsServiceCaller** tipinin kullanabileceği bir sınıf oluşturulmuştur. Ancak hazırlıklar bu tipleri yazmakla bitmez. Birde söz konusu **debug** edilecek tip örneğini **başlatacak/yürütüecek** bir başka sınıfın olmasında yarar vardır  İşte başlatıcı tipimiz.

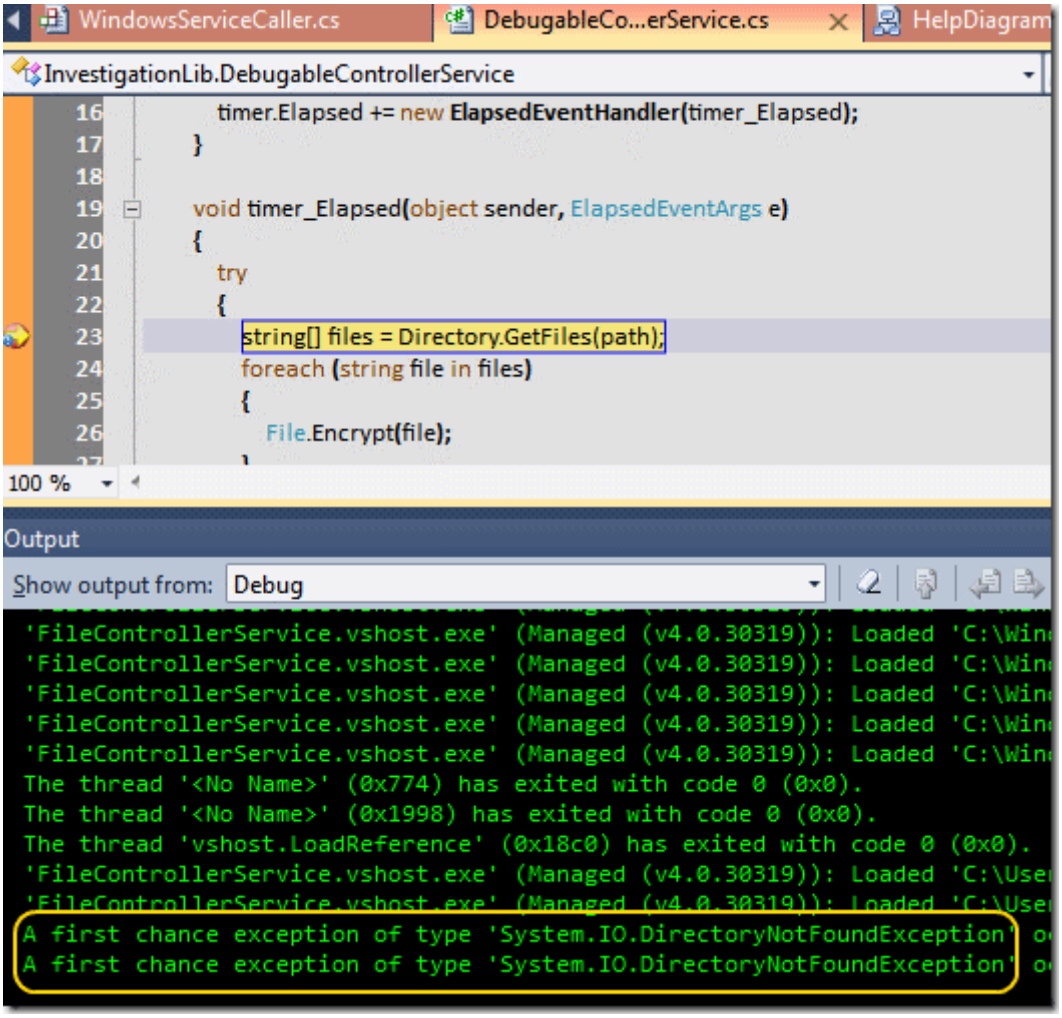
```
using System.Threading;

namespace InvestigationLib
{
    public static class ServiceStarter
    {
        public static void Run(string[] args,IWindowsServiceContract winSrv)
        {
            winSrv.OnStart(args);
            Thread.Sleep(45000);
            winSrv.OnStop();
        }
    }
}
```

Bu **static** sınıf içerisindeki **Run** metodu **IWindowsServiceContract** arayüzünü implemente eden her hangibir tip üzerinden **OnStart** ve **OnStop** metodlarını çağırabilir ve böylece simülasyon işlemini başlatabilir. Geriye yapılması gereken tek bir işlem kalmaktadır. **Windows Service** projesindeki **Main** metodu içeriğini aşağıdaki kod parçasında görüldüğü gibi değiştirmek.

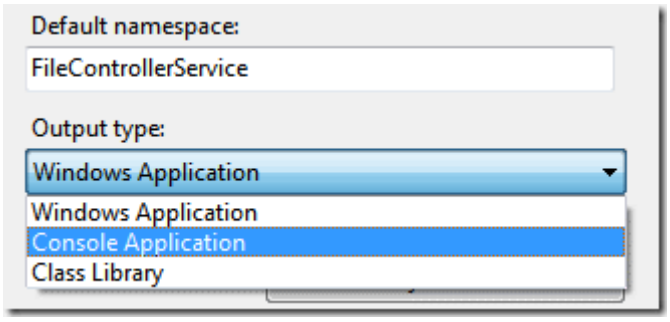
```
using System;
using System.Configuration;
```

```
using System.ServiceProcess;
using InvestigationLib;

namespace FileControllerService
{
   static class Program
   {
      static void Main()
      {
         #region Debug Edilebilir Versiyon

         DebugableControllerService implementation = new
DebugableControllerService();
         if (Environment.UserInteractive)
            ServiceStarter.Run(
               new string[]{
                  ConfigurationManager.AppSettings["Path"]
               }
               , implementation);
         else
            ServiceBase.Run(new WindowsServiceCaller(implementation));

         #endregion
      }
   }
}
```

Debug edilmek istenen kod içeriğini taşıyan **DebugableControllerService** örneğinin oluşturulmasından sonra **ServiceStarter** tipinin **Run** metodundan yararlanarak sürecin başlatılması sağlanır. Şu aşamada kod içerisinde **breakpoint'** ler yardımıyla ilerlenmesi mümkündür Hatta örneğin olmayan bir klasör içeriğinin **Directory** tipinin **GetFiles** metodu yardımıyla okunmaya çalışılması sırasında oluşan **Exception** mesajları da, **Visual Studio** arabiriminin **Output** ekranına düşmektedir. Aynen aşağıdaki ekran çıktısında görüldüğü gibi.



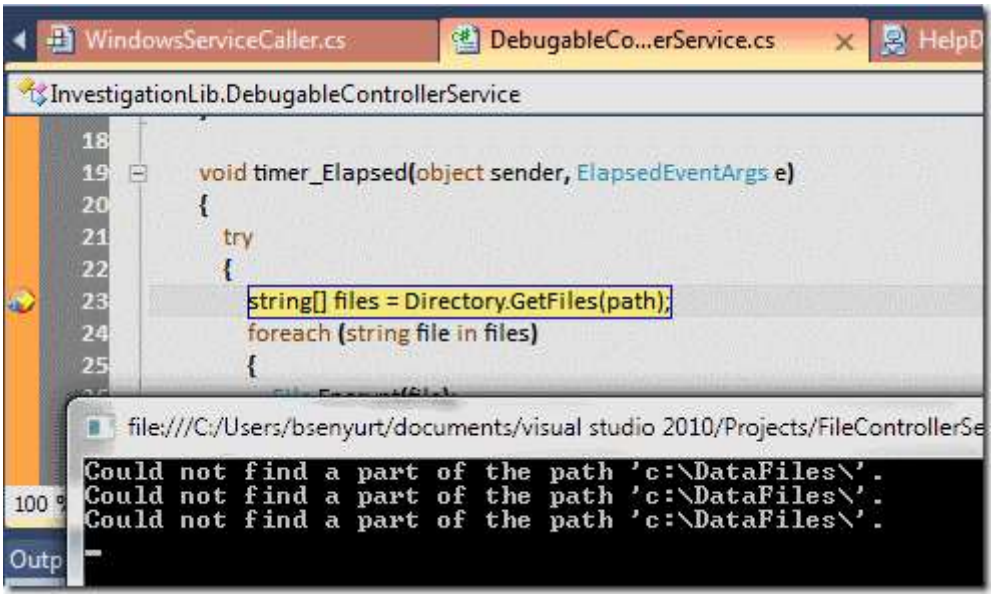
Ancak istenirse işlemlerin daha kolay takip edilmesi adına **Console** penceresine bilgi yazdırılması da mümkün olabilir. Bunun için **Windows Service** projesinin **Output** tipinin **Console Application**olarak değiştirilmesi yeterlidir.



Buna göre **Exception** bloğunu aşağıdaki gibi değişitirsek **debug** işlemleri sırasında **Console** ekranına çıktı üretebilmeyi de sağlayabiliriz.

```
void timer_Elapsed(object sender, ElapsedEventArgs e)
{
   try
   {
```

```
        string[] files = Directory.GetFiles(path);
        foreach (string file in files)
        {
           File.Encrypt(file);
        }
    }
    catch (Exception excp)
    {
        Console.WriteLine(excp.Message);
    }
}
```



Elbette **Debug** işlemleri sonrasında hataları bertaraf edilen servis kodunun son hali asıl **Windows Service** kodu ile değiştirilmeli ve bu şekilde **install** edilmelidir. örneğimiz bu haliyle artık **Console**penceresine bilgilendirme de bulunabilir. Ne **Text** dosyaya, ne işletim sistemindeki **Event Log'** lara ne veritabanı üzerindeki ilgili tablolara loglama işlemleri yapmak için gerekli atraksiyonlar ile uğraşmamıza gerek kalmamaktadır. Uygulamanın dışına çıkmadan, olduğumuz **Visual Studio ortamı(Environment)** içerisinden gerekli izleme ve **Debug** işlemleri yapılabilir. Doğrudan **Console**penceresinden gerekli izlemeler de yapılabilecektir. Bunu da unutmayalım Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

FileControllerService.rar (51,95 kb)

Netspecter Abstract Class Peşinde (2010-12-19T21:18:00)

c#,c# temelleri,interface,abstract class,

Yağmur şiddetini giderek arttırıyordu. Karanlık ara sokakta gizemli bir pardesü ise ağır ağır ilerlemekteydi. Etraftaki pis kokunun hemen sokağın başındaki çöp konteynerlerinden geldiği ap açık ortadaydı. Ancak gizemli pardesü bu kokuyu umursamıyordu bile. Bir an durdu ve şüpheli bir şekilde arkasına baktı.

Karanlık içerisinde sadece gözleri belli oluyordu. Kaşlarını çattı ve bir anda irkilerek koşmaya başladı. O kadar paniklemişti ki, koşarken teneke çöp kutularını fark etmedi bile. önce yere düştü, bir süre yuvarlandı. çevredeki kediler sağ sola kaçışırken, kalkmaya çalıştı ama önündeki metal iskeleyi hesaplayamadı. Kafasını sert bir şekilde demire çarptı.

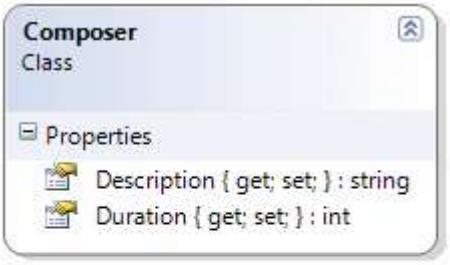
Bir kaç saniye sonra gökyüzünden düşen yağmur damlalarını görebiliyor ama seslerini duyamıyordu. Onun yerine kafasında bir uğultu vardı. Etrafında dans eden kod parçları görüyordu. Gözleri yavaş yavaş kararmaya başlamıştı. Derken başında beliren kişiyi gördü…

- Gizemli Pardesü : Netspecter!!! Sen haaa …

- Netspecter : Evet yaaa. Ben…Fazla uzağa kaçamadın eski dostum **Abstract Class**

Merhaba Arkadaşlar,

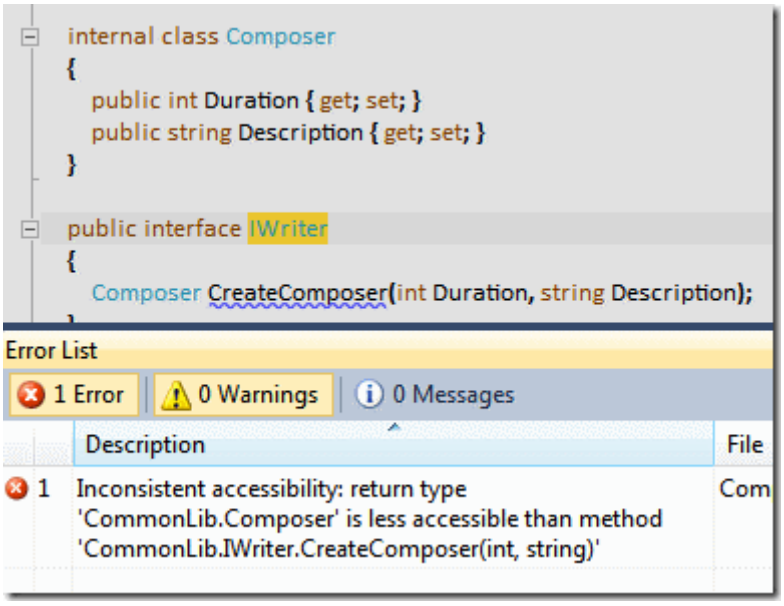
Netspecter bu kez bir abstract sınıfın peşinde. Sizin içinde eğlenceli bir deneyim olacağına inandığım enteresan bir vakayı analiz etmeye çalışıyor olacağız. çoğunlukla kod geliştirirken pek fark etmediğimiz bir hata ama hemen çözüm üretebiliyorz. Lakin bu çözümü üretirken istediğimizin dışında bir sonuca da neden olabiliyoruz. Dilerseniz hiç vakit kaybetmeden konumuza geçelim. İlk olarak aşağıdaki kod içeriğini göz önüne alarak başlamamızda yarar olacağı kanısındayım.



```
internal class Composer
{
   public int Duration { get; set; }
   public string Description { get; set; }
}

public interface IWriter
{
   Composer CreateComposer(int Duration, string Description);
}
```

örnek kod parçasında **IWriter** isimli bir **arayüz(interface)** tipi olduğunu görmekteyiz. Bildiğiniz üzere arayüzler **Plug-In** tabanlı programlamada, servis yönelimli mimarilerde sıklıkla kullanılmaktadır. Arayüzlerin tipik özelliklerinden birisi uygulandıkları tipler için zorlayıcı kuralları bildiriyor olmalarıdır. Ayrıca çok biçimli davranışta gösterebilirler. Bir başka deyişle kendisinde türeyen tipleri taşıyabilir ve çalışma zamanında onlara ait olan fonksiyonellikleri yürüterek kılık değşitirebilirler.

örneğimizde yer alan **IWriter** arayüzü ise özel bir duruma neden olmaktadır. Dikkat edileceği üzere **CreateComposer** metodu geriye **Composer** tipinden bir nesne örneği döndürmektedir. Söz konusu nesne tipinin erişim belirleyici **Internal'** dır. Bu koşullar altında kodu derlediğimizde aşağıdaki ekran görüntüsünde yer alan hatanın oluştuğunu görebiliriz.



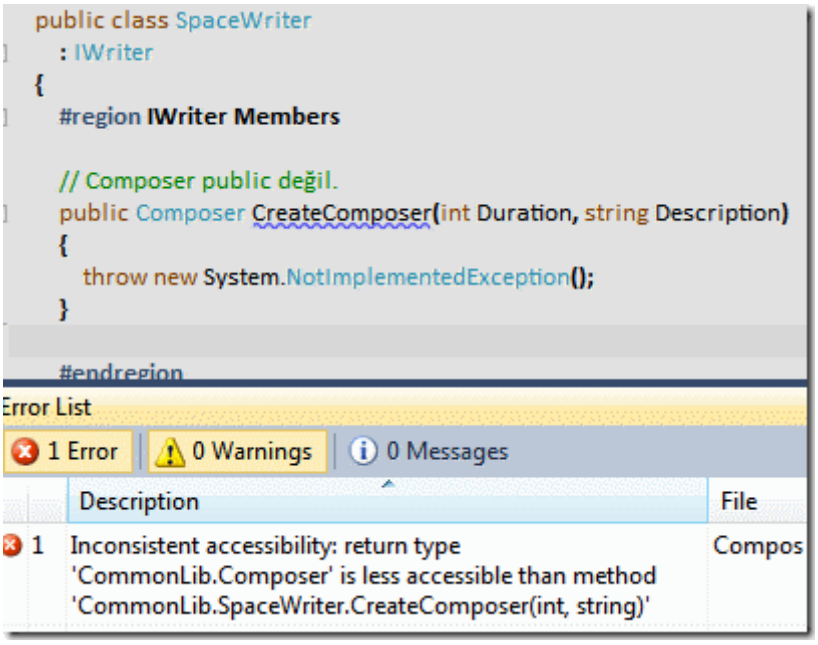
Dikkat edileceği üzere **IWriter** arayüzünün **internal** olan **Composer** tipini kullanan bir üyeye sahip olması mümkün değildir. Sorunun çözümü aslında basittir. Hatta her developer hemen bunu yapacaktır. **IWriter** arayüzünün internal olarak tanımlanması halinde herhangibir derleme hatası ile karşılaşılmayacaktır.

```
internal class Composer
{
   public int Duration { get; set; }
   public string Description { get; set; }
}

internal interface IWriter
{
   Composer CreateComposer(int Duration, string Description);
   // ...kurallara göre bu zaten Public bir üyedir. Ama şu noktada Internal olan bir tipi geriye döndürmektedir. Şüpheli bir durum mudur acaba?
}
```

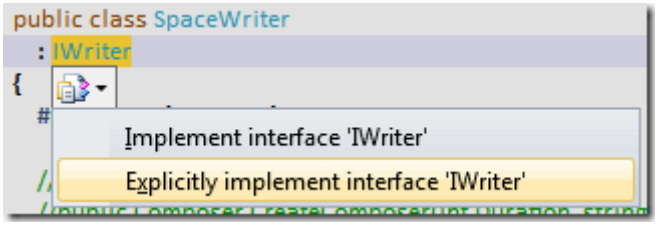
Burada yine de şüphe uyandırıcı bir gelişme vardır. Bilindiği üzere **Interface** üyeleri doğal olarak **public'** tir. Buna rağmen doğal olarak public olan **CreateComposer** metodu **Internal**erişim belirleyicisine sahip bir örneği döndürmektedir. Oysaki public olan IWriter arayüzünün internal olan Composer tipini kullanmasına izin verilmemiştir Şimdilik bunu düşünerek kafamızı karıştırmayalım. çünkü asıl amacımız **internal'** a çekmek zorunda kaldığımız **IWriter** arayüz tipinin **public** olarak kullanılmasının istenmesidir. Açık olmak gerekirse şu an için bunu da bir kenara bırakalım ve **IWriter** arayüzünü bir sınıfa bu haliyle uygulamak istediğimizi düşünelim. örneğin aşağıdaki kod parçasında görüldüğü gibi.

```
public class SpaceWriter
      : IWriter
   {
      #region IWriter Members

      public Composer CreateComposer(int Duration, string Description)
      {
         throw new System.NotImplementedException();
      }

      #endregion
   }
```

Görüldüğü üzere **SpaceWriter** isimli sınıf, **IWriter interface** tipini implemente etmektedir. Buna göre de **CreateComposer** metodunu **override** etmiştir. Herşey yolunda görünmesine rağmen kodu derlediğimizde aşağıdaki hata mesajını almamız kaçınılmazdır.



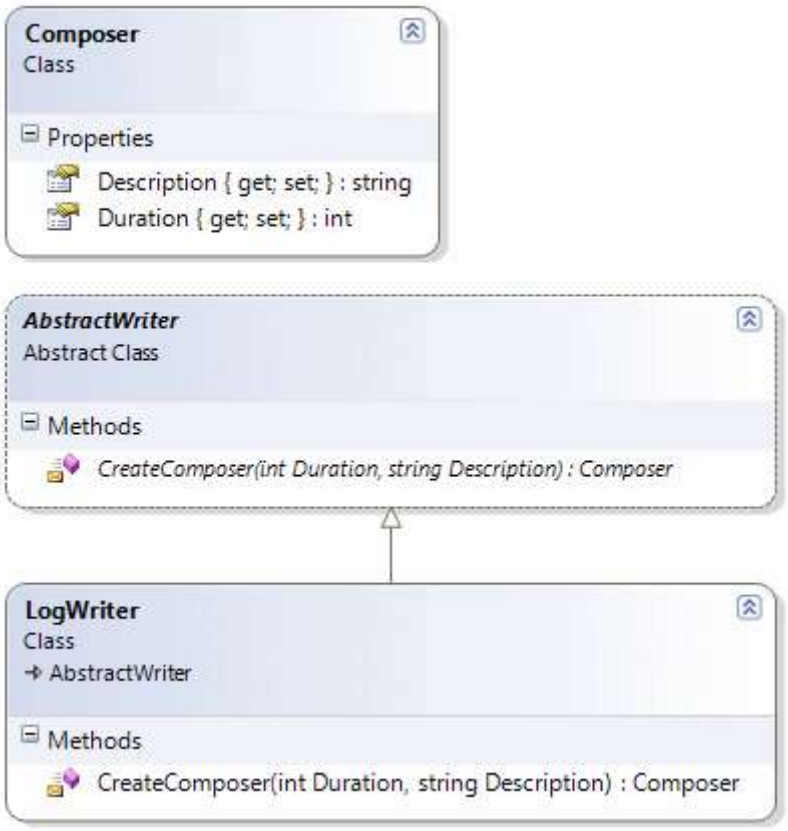
Yine yine yine…**Inconsistent Accessibility** hatası Burada arayüz tipinin **bilinçsiz(Implicitly)** olarak uygulandığı görülmektedir. İşte hani bazen **interface** tiplerini implemente ederken**Ctrl+Shift+F10** tuşlarına bastığımızda çıkan seçenekler arasında bir de **Explicit** olanı vardır ya İşte o açık bildirim burada bir çözüm olmaktadır.



Yani arayüz implementasyonunu aşağıdaki hale getirirsek kod sorunsuz bir şekilde derleniyor olacaktır.

```
public class SpaceWriter
   : IWriter
{
   #region IWriter Members

   Composer IWriter.CreateComposer(int Duration, string Description)
   {
      throw new System.NotImplementedException();
   }

   #endregion
}
```

Dikkat edileceği üzere metod adının bildirimi sırasında **IWriter.CreateComposer(InterfaceName.InterfaceMemberName)** isimlendirme notasyonu devreye girmiştir ve derleme hatası ortadan kalkmıştır. Yine de sorun devam etmektedir Mecburen **IWriter** arayüzü internal erişim belirleyicisini kullanmak zorunda kalmıştır. İşte sevgili kahramanımız **Netspecter'** ın peşinden koştuğu **abstract class** bize bu tip bir vaka için çözüm getirebilir. Nasıl mı? İşte **sınıf çizelgemiz(Class Diagramı)** ve örnek kod parçamız.




```
internal class Composer
{
   public int Duration { get; set; }
   public string Description { get; set; }
}

public abstract class AbstractWriter
{
   internal abstract Composer CreateComposer(int Duration, string Description);
}

public class LogWriter
   : AbstractWriter
{
   internal override Composer CreateComposer(int Duration, string Description)
```

```
    {
        throw new System.NotImplementedException();
    }
}
```

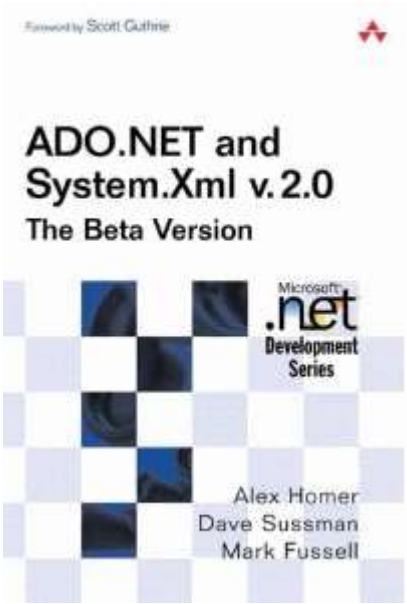
Volaaaa… Evet evet biliyorum **interface** tipi kullanımından vazgeçtik ve **abstract class** kullanımına geçtik. Ancak bir açıdan baktığımızda aynı amaca hizmet ettiklerini ifade ebiliriz. Söz gelimi gerek **interface** gerek **abstract sınıflar** örneklenemezler. Yani **new** operatörü ile **initialize** edilemezler üstelik her ikiside çok biçimli davranış gösterebilirler. Türetme amacıyla kullanılabilirler ve kendisinden türeyen tiplerin mutlaka uyması gereken kuralları belirtebilirler. Sadece nasıl belirttikleri farklıdır. Tabi arada başka farklarda bulunmaktadır. Söz gelimi arayüz tiplerinde iş yapan bloklara sahip üye bildirimleri yapılamaz veya bu üyeler için erişim belirleyicisi kullanılamaz. Ancak **abstract** tiplerde durum tam tersidir.

örnek vakamızın son halinde **AbstractWriter abstract** sınıfı içerisinde **internal** olarak işaretlenen **CreateComposer** metodu olduğu görülmetkedir. üstelik bu metodun geriye dönüş tipi de **internal** olan **Composer** sınıfıdır. Diğer yandan en önemli sonuç **AbstractWriter** tipinin **public** olarak işaretlenebilmiş olmasıdır. Yani çok biçimlilik gösteren tipimiz dış ortama public olarak sunulabilmektedir.

Bu vaka çalışmasında her zaman karşımıza çıkmayacak ama hangi durumlarda **abstract** sınıf kullanımını tercih etmemizi belirleyebilecek bir konuyu ele almaya çalıştık. Sanıyorum artık bir **interface** tipini **implemente** ederken **Explicitly** olan versiyonuna daha bir anlamlı bakıyor olacağız Tekrardan görüşünceye ve Netspecter' ın farklı bir macerasında buluşuncaya dek hepinize mutlu günler dilerim.

Entity Framework Üzerinde TransactionScope Kullanımı (2010-12-19T20:52:00)

entity framework,ef 4.0,transaction,transaction scope,



Günlerden Salı, sonbahar. Neredeyse hiç uyumadan geçen bir gecenin ardından sabaha karşı yorgunluktan sızan **Netspecter**, **CAD**' in baş ucunda dakikalarca miyavlaması sonucu ancak kendine gelebilir. Eksi ve köhne divanı, o doğrulurken olabildiğince haykırarak gıcırdamaktadır. önce terliklerini arar. Oda darma dağınıktır. Tüm gece kütüphanedeki sayısız kitabı indirmiş ve bir sonuç bulmak için saatlerce araştırma yapmıştır. Sonuçta terliğin tekinin dahi bulunamadığı bir kalabalık kitap yığınıdır.

Kafada korkunç bir baş ağrısı, dışarıdan gelen metronun raylarda bıraktığı ses ve CAD' in cılız miyavlamaları…Netspecter divandan kalkarken şöyle bir

belinden geriye doğru esner. Derken tavan lambasının biraz üstüne vuran dikdörtgen biçimli gölgeyi fark eder. Nasıl olur? Bu kitap gözünden nasıl kaçmıştır. Gece karanlığında fark edemediği kaynağı gün ışığı açığa çıkarmıştır. İşte oradadır. Gölgenin kaynağına doğru gider. **Ado.Net and System.Xml v2.0 The Beta Version** Yaşasın diyerek haykırır.

Merhaba Arkadaşlar,

Yıllar yıllar önce **.Net Framework 2.0** ile gelen yenilikleri takip etmeye çalıştığım dönemlerde, **Amazon** üzerinden getirttiğim kitaplardan birisi de yandaki resimde görülen **Ado.Net and System.Xml v2.0** kitabı idi. Şu an halen kitaplığımda durmakta. O sıralar **CSharpNedir?** bünyesinde **Ado.Net** bölüm editörlüğü yaptığımdan, bu kitabı tedarik etmiş ve çalışmıştım. **.Net Framework 2.0** ile gelen yeni **Xml** alt yapısının beta hali ile alması bir yana **Ado.Net**' in yeni 2.0 sürümü için planlanan bazı kabiliyetlerde anlatılmaktaydı ki bunlardan bekli de en önemlisi **System.Transaction.dll assembly** içerisinde yer alan ve özellikle **Distributed Transaction** yönetimini daha etkili ve kolay bir şekilde ele almamızı sağlayan **TransactionScope** tipiydi.

Bilindiği üzere bu tip sayesinde özellikle blok içerisine alınan birden fazla ve farklı **bağlantı(Connection)** üzerinden gerçekleştirilecek veritabanı işlemlerinin, aynı **Transaction** alanı içerisinde ele alınması ve toplu olarak **Commit/Rollback** edilmeleri mümkün olmaktadır. çok doğal olarak ardışıl olarak gerçekleştirilen ve belirli iş kuralları içerisinde sıralanan bazı veritabanı işlemlerinin, tamamen başarılı oldukları takdirde onaylanmasının istendiği durumlar son derece yaygındır.*(Lütfen Transaction kavramı ve ACID ilkelerini hatırlayınız)*

Bu durum **Entity Framework** içinde geçerlidir. çok doğal olarak basit bir **Entity** nesne örneğinin veri içeriğinin veritabanına eklenmesi işleminden tutunda da, farklı **Context** örnekleri içerisinde gerçekleştirilen veri işlemlerinin de bir **Transaction** alanı içerisinde gerçekeştirilmesi istenebilir İşte bu yazımızda **Transaction** yapısının **Entity Framework** tarafındaki kullanımını incelemeye çalışıyor olacağız. İşe ilk olarak aşağıdaki basit kod parçası ile başlayabiliriz.

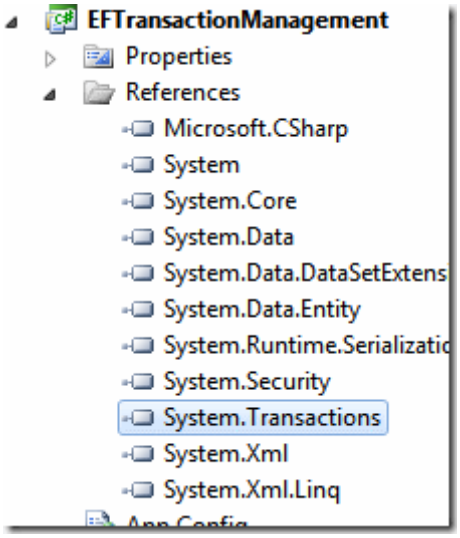
```
namespace EFTransactionManagement
{
   class Program
   {
      static void Main(string[] args)
      {
         using (ChinookEntities entities = new ChinookEntities())
         {
            Artist newArtist = new Artist
            {
               Name="Sertap Erener"
```

```
            };
            entities.AddToArtists(newArtist);

            Album newAlbum = new Album()
            {
                Title="Rengarenk"
            };
            newArtist.Albums.Add(newAlbum);
            entities.AddToAlbums(newAlbum);

            entities.SaveChanges();
        }
    }
  }
}
```

Chinook model veritabanını kullanan bu örnekte **Context** nesnesine ait **SaveChanges** metodunun çağırılmasından önce iki **Insert** işlemi gerçekleştirildiği görülmektedir. öncelikli olarak**Artist** tipinden bir örnek oluşturulmuş ve **Context** tipine ait koleksiyona eklenmiştir. Hemen arından da bir **Album** örneği oluşturulmuş ve yine ilgili koleksiyona ilave edilmiştir. Tabi bu örnekte**Album** ile **Artist** örnekleri arasında bir ilişki de söz konusudur. Yani üretilen yeni **Album** nesne örneği, üretilen yeni **Artist** örneğinin **Albums** özelliği ile işaret edilen koleksiyonuna da eklenmiştir. Eğer **SaveChanges** metodunun çağırılması sonrasındaki **SQL Server Profiler** görüntüsüne bakarsak aşağıdaki çıktı ile karşılaştığımızı görebiliriz.

TM: Begin Tran starting	BEGIN TRANSACTION
TM: Begin Tran completed	BEGIN TRANSACTION
RPC:Completed	exec sp_executesql N'insert [dbo].[Artist]([Name]) values (
RPC:Completed	exec sp_executesql N'insert [dbo].[Album]([Title], [ArtistId
TM: Commit Tran starting	COMMIT TRANSACTION
TM: Commit Tran completed	COMMIT TRANSACTION

Dikkat edileceği üzere iki **insert** işleminden önce bir **Transaction** başlatılmış ve sonrasında ise **Commit** işlemi ile bu girişler onaylanmıştır. Bir başka deyişle **Entity Framework**tarafında, **SaveChanges** metodunun çağırılması sonrasında **bilinçsiz olarak(Implicitly)** bir **Transaction** oluşturulduğu görülmektedir. Elbette bu davranış şekli değiştirilebilir. Bir başka deyişle developer bazlı bir **Transaction** kullanımı da söz konusu olabilir. Bunun için ilk akla gelen bilinçli olarak **TransactionScope** örneğinin kullanılması ve **SaveChanges** çağrısının burada kullanılmasıdır. Şimdi örnek uygulamamıza **System.Transaction.dll assembly'** ını referans ederek vakamızı incelemeye devam edelim.

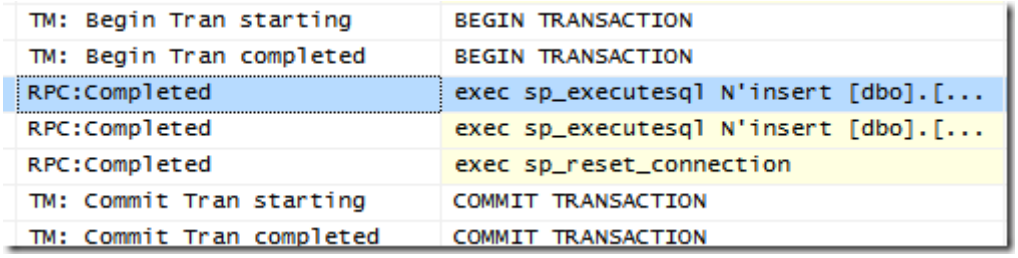


örnek kod içeriğini ise aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.Transactions;

namespace EFTransactionManagement
{
   class Program
   {
      static void Main(string[] args)
      {
         bool acceptStatus = false;

         using (ChinookEntities entities = new ChinookEntities())
         {
               Artist newArtist = new Artist
               {
                  Name = "Sertap Erener"
               };
               entities.AddToArtists(newArtist);

               Album newAlbum = new Album()
               {
                  Title = "Rengarenk"
               };
               newArtist.Albums.Add(newAlbum);
               entities.AddToAlbums(newAlbum);

            using (TransactionScope scope = new TransactionScope())
            {
               try
```

```
                {
                    entities.SaveChanges();
                    scope.Complete();
                    acceptStatus = true;
                }
                catch //Exception kontrolü yapmamızda yarar var
                {
                    Console.WriteLine("Sorun var!");
                }

                if (acceptStatus)
                    entities.AcceptAllChanges();
            }
        }
    }
}
```

Dikkat edileceği üzere **Album** ve **Artist** nesne örneklerinin giriş işlemlerinin onaylandığı **SaveChanges** metod çağrısı bir **try…catch** bloğu içerisinde kontrol altına alınmıştır. Dahası söz konusu **try…catch** bloğu da **TransactionScope** bloğu içerisinde konuşlandırılmıştır. Eğer kod **try** bloğunun son satırına kadar başarılı bir şekilde gelebilirse**TransactionScope** nesne örneğine ait **Complete** metodunun çağırılması ile, yapılan tüm işlemlerin onaylanması sağlanmaktadır. *(Elbette catch bloğuna girilmesi halinde ilgili Transaction içeriğinin Rollback edilmesi süreci de otomatik olarak işletilecektir)* Burada bilinçli olarak bir transaction bloğu açılması söz konusudur. Uygulama kodunu çalıştırdığımızda ve **SQL Server Profiler** aracının çıktısına baktığımızda aşağıdaki ekran görüntüsünde yer alan sonuçların üretildiği görülebilir.



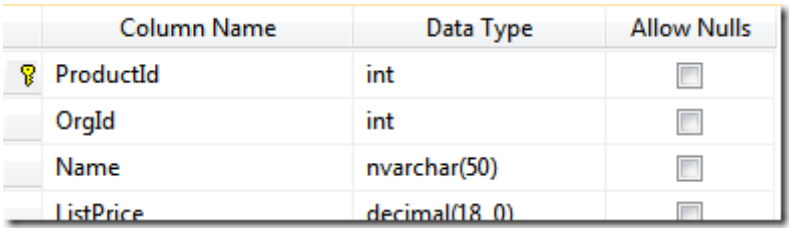
TM: Begin Tran starting	BEGIN TRANSACTION
TM: Begin Tran completed	BEGIN TRANSACTION
RPC:Completed	exec sp_executesql N'insert [dbo].[...
RPC:Completed	exec sp_executesql N'insert [dbo].[...
RPC:Completed	exec sp_reset_connection
TM: Commit Tran starting	COMMIT TRANSACTION
TM: Commit Tran completed	COMMIT TRANSACTION

Dikkat edileceği üzere ilk örneğimizde olduğu gibi bir **transaction(Begin Transaction)** başlatılmış ve **insert** işlemleri sonrasında **commit(Commint Transaction)** edilmiştir.

Peki ya İki farklı Context nesnesi üzeriden bilinçli TransactionScope kullanımı söz konusu olabilir mi?

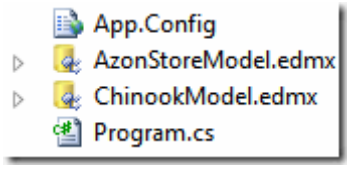
çok doğal olarak uygulama içerisinde farklı veri kaynaklarını temsil eden farklı **Context** nesneleri kullanılıyor olabilir. Buna göre ilgili **Context** nesneleri arasında

akan bazı iş fonksiyonelliklerinin **transaction** kullanımını gerektirmesi de düşünülebilir. Dilerseniz bu durumu incelemek için senaryomuza aşağıdaki **Product** isimli tablo içeriğine sahip **AzonStore** isimli bir veritabanını daha ekleyelim. **AzonStore** tahmin edeceğiniz üzere buradaki **TransactionScope** kullanımını ele almamız için düşünülmüş kobay veritabanıdır ve her hangibir özel tarafı yoktur



Column Name	Data Type	Allow Nulls
ProductId	int	
OrgId	int	
Name	nvarchar(50)	
ListPrice	decimal(18, 0)	

Tabi bu durumda uygulamamızda iki adet **Entity Data Model** olması gerektiğini de ifade etmeliyiz.



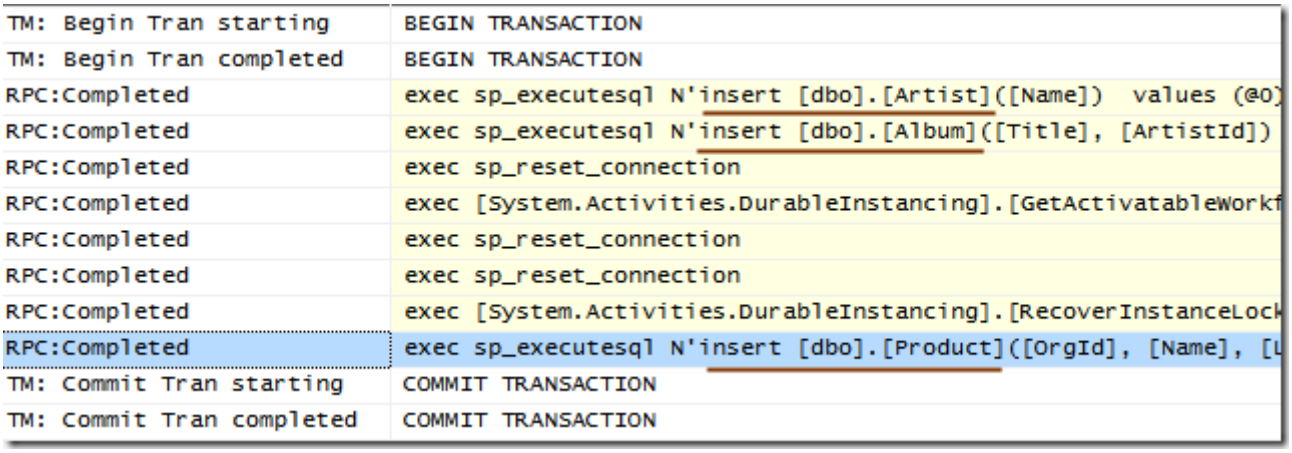
örnek kod parçamızı da aşağıdaki gibi geliştirdiğimizi düşünelim.

```
using System;
using System.Transactions;

namespace EFTransactionManagement
{
   class Program
   {
      static void Main(string[] args)
      {
         bool acceptStatus = false;
         ChinookEntities chinookEntities = null;
         AzonStoreEntities azonEntities = null;

         using (chinookEntities = new ChinookEntities())
         {
            Artist newArtist = new Artist
            {
               Name = "Sertap Erener"
            };
            chinookEntities.AddToArtists(newArtist);

            Album newAlbum = new Album()
            {
```

```
            Title = "Rengarenk"
        };
        newArtist.Albums.Add(newAlbum);
        chinookEntities.AddToAlbums(newAlbum);

        azonEntities = new AzonStoreEntities();
        Product newProduct = new Product
        {
            OrgId = newAlbum.AlbumId,
            Name = newAlbum.Title,
            ListPrice = 10
        };
        azonEntities.AddToProducts(newProduct);

        using (TransactionScope scope = new TransactionScope())
        {
            try
            {
                chinookEntities.SaveChanges();
                azonEntities.SaveChanges();

                scope.Complete();
                acceptStatus = true;
            }
            catch (Exception excp)
            {
                Console.WriteLine(excp.Message);
            }
        }
        if (acceptStatus)
        {
            chinookEntities.AcceptAllChanges();
            azonEntities.AcceptAllChanges();
        }
    }
  }
}
```

Bu kez işin içerisine ikinci bir **Context** örneği daha girmektedir. **Chinook** üzerinde yapılan **Album** ekleme işleminden sonra bu albüme ait bilgilerden bazıları **AzonStore Context'** I içerisindeki**Product** nesnesine de eklenmektedir. Dolayısıyla iki farklı veritabanı üzerinde gerçekleşecek bir işlem söz konusudur. Burada söz konusu **Context'** lerin işaret ettiği/kullandığı veritabanları farklı sunucular üzerinde de olabilir ki bu durumda **TransactionScope, Distirbuted Transaction**

Coordinator(DTC) aracını otomatik olarak devreye alarak bir **dağıtık transaction**alanı başlatacaktır. Dilerseniz bir de **SQL Server Profiler** aracının ürettiği çıktıya bakalım.



TM: Begin Tran starting	BEGIN TRANSACTION
TM: Begin Tran completed	BEGIN TRANSACTION
RPC:Completed	exec sp_executesql N'insert [dbo].[Artist]([Name]) values (@0)
RPC:Completed	exec sp_executesql N'insert [dbo].[Album]([Title], [ArtistId])
RPC:Completed	exec sp_reset_connection
RPC:Completed	exec [System.Activities.DurableInstancing].[GetActivatableWorkf
RPC:Completed	exec sp_reset_connection
RPC:Completed	exec sp_reset_connection
RPC:Completed	exec [System.Activities.DurableInstancing].[RecoverInstanceLock
RPC:Completed	exec sp_executesql N'insert [dbo].[Product]([OrgId], [Name], [U
TM: Commit Tran starting	COMMIT TRANSACTION
TM: Commit Tran completed	COMMIT TRANSACTION

Görüldüğü üzere **Chinook** ve **AzonStore** veritabanları üzerindeki tüm **Insert** işlemleri aynı **Transaction** alanı içerisine dahil edilmişlerdir*(Begin Transaction <–> Commit Transaction)*

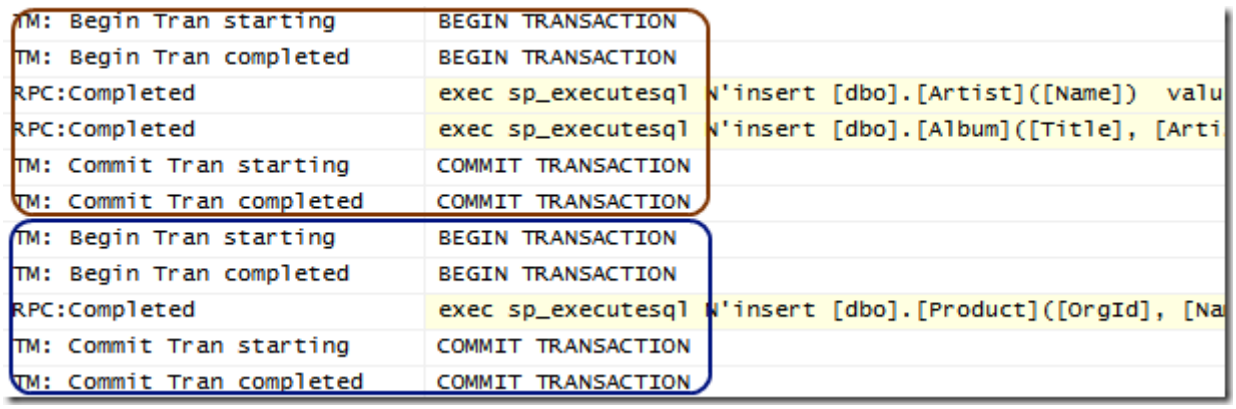
Peki son senaryodaki Context nesne örnekleri üzerinden gerçekleştirilen işlemleri bir TransactionScope bloğu içerisine ele almasaydık?

Nitekim bu son derece doğal bir davranış biçimi olacaktır. özellike **Entity Framework**'ü ilk kez kullanmaya başlayan bir geliştirici için. Bu durumu analiz etmek için kodu aşağıdaki gibi yenilediğimizi düşünebiliriz.

```
using System;

namespace EFTransactionManagement
{
   class Program
   {
      static void Main(string[] args)
      {
         ChinookEntities chinookEntities = null;
         AzonStoreEntities azonEntities = null;

         using (chinookEntities = new ChinookEntities())
         {
            Artist newArtist = new Artist
            {
               Name = "Sertap Erener"
            };
            chinookEntities.AddToArtists(newArtist);
```

```
            Album newAlbum = new Album()
            {
               Title = "Rengarenk"
            };
            newArtist.Albums.Add(newAlbum);
            chinookEntities.AddToAlbums(newAlbum);

            azonEntities = new AzonStoreEntities();
            Product newProduct = new Product
            {
               OrgId = newAlbum.AlbumId,
               Name = newAlbum.Title,
               ListPrice = 10
            };
            azonEntities.AddToProducts(newProduct);

            chinookEntities.SaveChanges();
            azonEntities.SaveChanges();
         }
      }
   }
}
```

Bu kez sadece ilgili Context nesne örneklerine ait **SaveChanges** metodlarının ardışıl olarak çağırılması söz konusudur ki bunun **SQL Server Profiler** aracına baktığımızda üreteceği sonuç aşağıdaki ekran görüntüsünde olduğu gibidir.



Görüldüğü üzere her **SaveChanges** çağrısı için ayrı bir **Transaction** işlemi başlatılmıştır. Burada **AzonStore** için kullanılan **Context** örneğinin, **Chinook** için üretilen **Context** nesnesine ait **using** bloğu içerisinde olmasının dahi bir önemi yoktur. Olay sadece **SaveChanges** metod çağrılarına aittir.

Entity Framework tarafında bu yazıda ele aldığımız gibi basit veritabanı modellerinin söz konusu olduğu durumlarda **Transaction** kullanımına dikkat edilmemesi söz konusu olabilir. Hatta gerekmeyebilir. Nitekim Transaction oluşturmanın da veritabanı kaynağı

üzerinde bir maliyeti vardır. Ancak **Enterprise** seviyedeki uygulamalar ve kullandıkları veritabanı sistemleri ile karmaşık ve bütünlüğü önem arz eden iş kuralları söz konusu olduğunda, **TransactionScope** kullanımına ciddi anlamda dikkat edilmesi gerekmektedir. Tabi bilindiği üzere **TransactionScope**nesne örneği üzerinden yapılabilecek farklı ayarlamalar da söz konusudur. Şu anki örneklerimizde bu tipin son derece yalın ve sade kullanımı değerlendirilmiştir. Size tavsiyem söz konusu örneği **farklı sunucular** veya **SQL Instance**' ları üzerindeki n sayıda veritabanında ele alacak şekilde değiştirmeniz ve gözlemlemenizdir. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Fluent Interface Nedir? (2010-12-19T18:15:00)

.net framework,c#,fluent interface,interface,

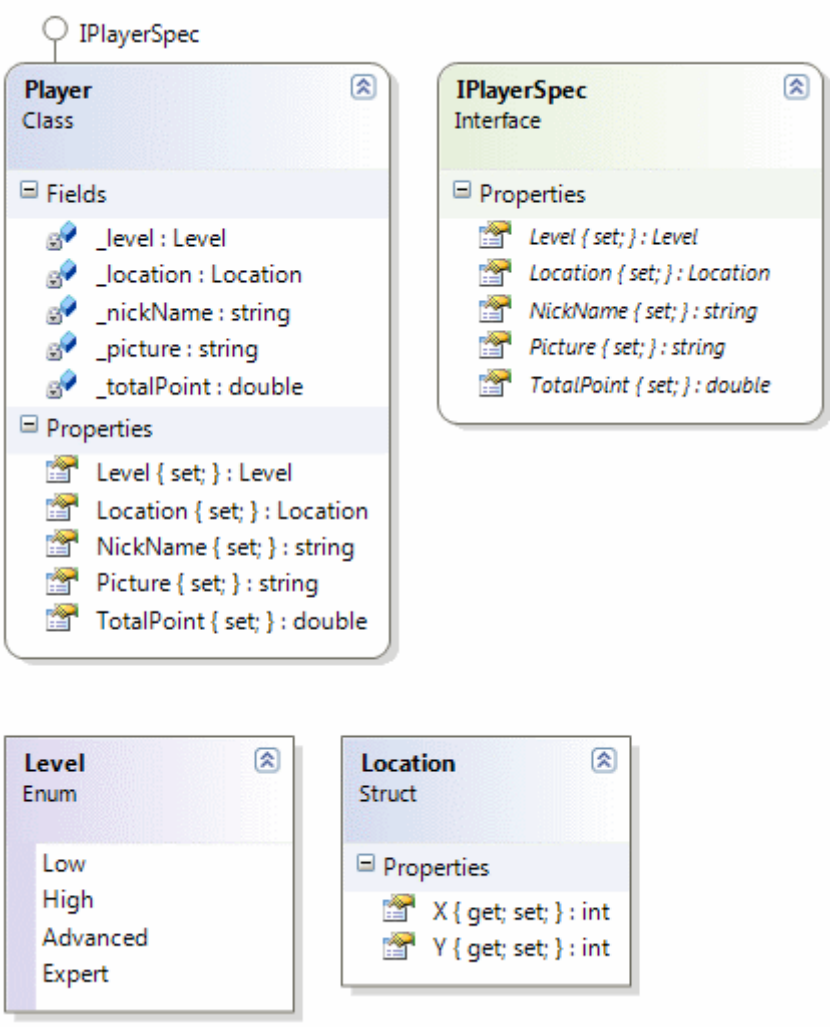
Merhaba Arkadaşlar,

Yazılımcı olarak bizlerin zaman içerisindeki gelişimimiz/ilerleyişimiz açısından takip etmemiz gereken önemli kişiler olduğu aşikardır. Söz gelimi çevik süreç prensiplerine ait manifestoyu hazırlayanlar arasında yer alan **Martin Fowler** gibi. Martin Fowler bana göre yazılım mühendisliğinin uç noktalarında yaşayan bir bilim insanıdır. Bilim insanı diyorum nitekim çalıştığı şirkette **Chief Scientist**pozisyonunda görev almaktadır

Bu güne kadar yayımlamış olduğu çok değerli kitaplar bulunmaktadır. Hatta son çıkarttığı ve merakla bekleyip okumaya başladığım Domain-Specific Languages (Addison-Wesley Signature Series) isimli kitap en büyük favorilerim arasında yer almakta.

Bundan önceki favori kitaplarım ise Clean Code: A Handbook of Agile Software Craftsmanship ve Agile Principles, Patterns, and Practices in C# dir

Pek **Martin Fowler**' ın kulaklarını niye bu kadar çok çınlatıyoruz Bu günkü yazımızda ilk olarak **Martin Fowler** ve **Eric Evans** tarafından tanımlanan **Fluent Interface** konusunu irdelemeye çalışıyor olacağız. Aslında kelime anlamlarından yola çıkarsak okunabilir, açık, net arayüz tiplerinden bahsettiğimizi düşünebiliriz. Ancak bu şekilde söz konusu kavrama biraz haksızlık etmiş oluruz. **Fluent Interface** esas itibariyle daha okunabilir kodlama açısından önem arz eden ve uygulanması sırasında **metod zincirlerinden** yararlananan bir yaklaşım sunmaktadır. Şimdi ne demek istediğimizi ben de ifade edemedim aslında Gelin basit bir örnek ile konuyu didiklemeye başlayalım.Bu amaçla aşağıdaki kod içeriğini göz önüne alabiliriz.



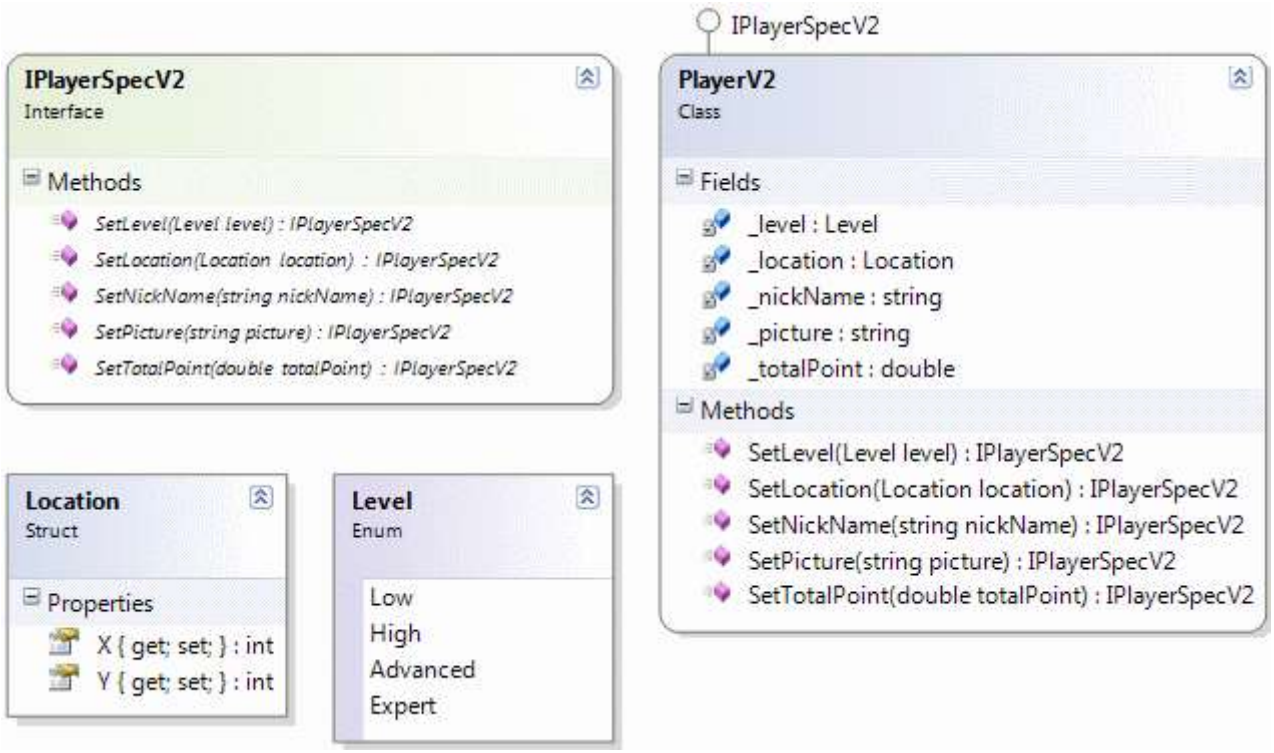

```
namespace UsingFluentInterface
{
   struct Location
   {
      public int X { get; set; }
      public int Y { get; set; }
   }

   enum Level
   {
      Low,
      High,
      Advanced,
      Expert
   }

   interface IPlayerSpec
   {
```

```
    string NickName{  set; }
    Level Level { set; }
    double TotalPoint { set; }
    string Picture { set; }
    Location Location { set; }
}

class Player
    : IPlayerSpec
{
    string _nickName;
    Level _level;
    double _totalPoint;
    string _picture;
    Location _location;

    #region IPlayerSpec Members

    public string NickName
    {
        set { _nickName = value; }
    }

    public Level Level
    {
        set { _level = value; }
    }

    public double TotalPoint
    {
        set { _totalPoint = value; }
    }

    public string Picture
    {
        set { _picture = value; }
    }

    public Location Location
    {
        set { _location = value; }
    }

    #endregion
}
```

```
   class Program
   {
      static void Main(string[] args)
      {
         IPlayerSpec marti = new Player();

         marti.NickName = "Makflay";
         marti.Level = Level.Advanced;
         marti.Location = new Location { X = 12, Y = 19 };
         marti.Picture = "Monster.jpg";
         marti.TotalPoint = 198.45;

      }
   }
}
```

örnek kod parçasında **IPlayerSpec** isimli bir **arayüz(Interface)** tanımının kullanıldığı görülmektedir. Bu tanımlaya göre söz konusu arayüzü implemente eden herhangibir sınıfın uygulaması gereken bazı **özellikler(Properties)** vardır. **Player** isimli sınıf, **IPlayerSpec** isimli arayüzü implemente etmektedir ve buna göre **NickName, Level, Location, Picture** ve**TotalPoint** isimli özelliklerinin her birini ele almalıdır.

Aslında buraya kadar ki kod parçasında olağan dışı bir durum yer almadığını ifade edebiliriz. Standart olarak **interface** tanımlaması ve bunu uygulayan bir sınıf tasarımı söz konusudur. **Main** metodu içerisinde ise dikkat edileceği üzere **interface** tipinin çok biçimli yapısı göz önüne alınarak **marti** isimli değişkene, **Player** tipinden bir nesne örneğinin oluşturulup atandığı görülmektedir. Güzel…Şimdi örneğimizi biraz daha değiştiriyor olacağız. İşte yeni kod içeriğimiz.




```
namespace UsingFluentInterface
{
   struct Location
   {
      public int X { get; set; }
      public int Y { get; set; }
   }

   enum Level
   {
      Low,
      High,
      Advanced,
      Expert
   }

   #region Fluent Sample

   interface IPlayerSpecV2
   {
      IPlayerSpecV2 SetNickName(string nickName);
      IPlayerSpecV2 SetLevel(Level level);
      IPlayerSpecV2 SetTotalPoint(double totalPoint);
      IPlayerSpecV2 SetPicture(string picture);
      IPlayerSpecV2 SetLocation(Location location);
   }
```

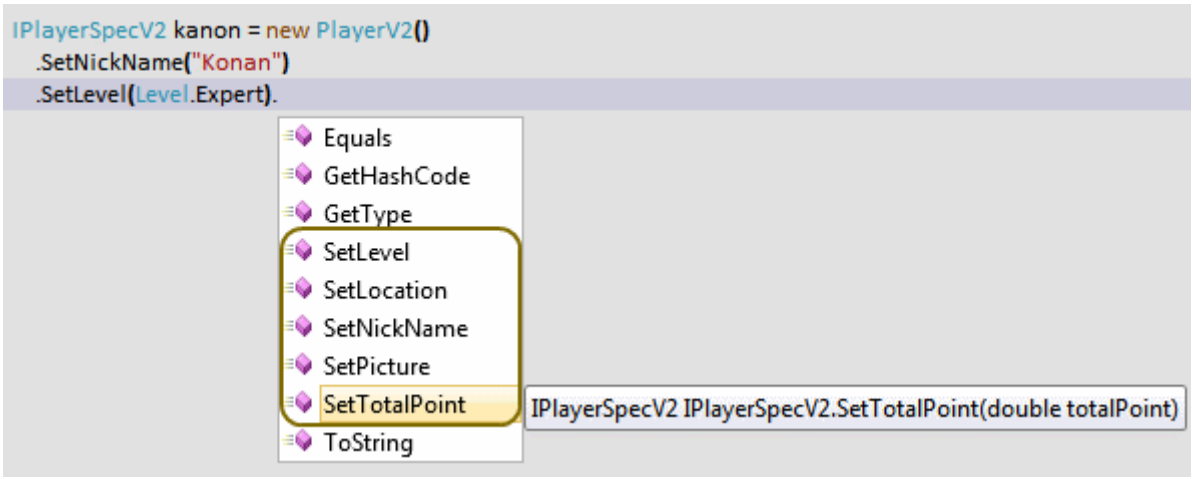
```
class PlayerV2
   :IPlayerSpecV2
{
   string _nickName;
   Level _level;
   double _totalPoint;
   string _picture;
   Location _location;

   #region IPlayerSpecV2 Members

   public IPlayerSpecV2 SetNickName(string nickName)
   {
      this._nickName = nickName;
      return this;
   }

   public IPlayerSpecV2 SetLevel(Level level)
   {
      this._level = level;
      return this;
   }

   public IPlayerSpecV2 SetTotalPoint(double totalPoint)
   {
      this._totalPoint = totalPoint;
      return this;
   }

   public IPlayerSpecV2 SetPicture(string picture)
   {
      this._picture = picture;
      return this;
   }

   public IPlayerSpecV2 SetLocation(Location location)
   {
      this._location = location;
      return this;
   }

   #endregion
}

#endregion
```

```
   class Program
   {
      static void Main(string[] args)
      {
         IPlayerSpecV2 kanon = new PlayerV2()
            .SetNickName("Konan")
            .SetLevel(Level.Expert)
            .SetLocation(new Location { X = 15, Y = 45 })
            .SetPicture("Snoopy.gif")
            .SetTotalPoint(45);
      }
   }
}
```

Vay…Vay…Vay…Vayyyy!!! 

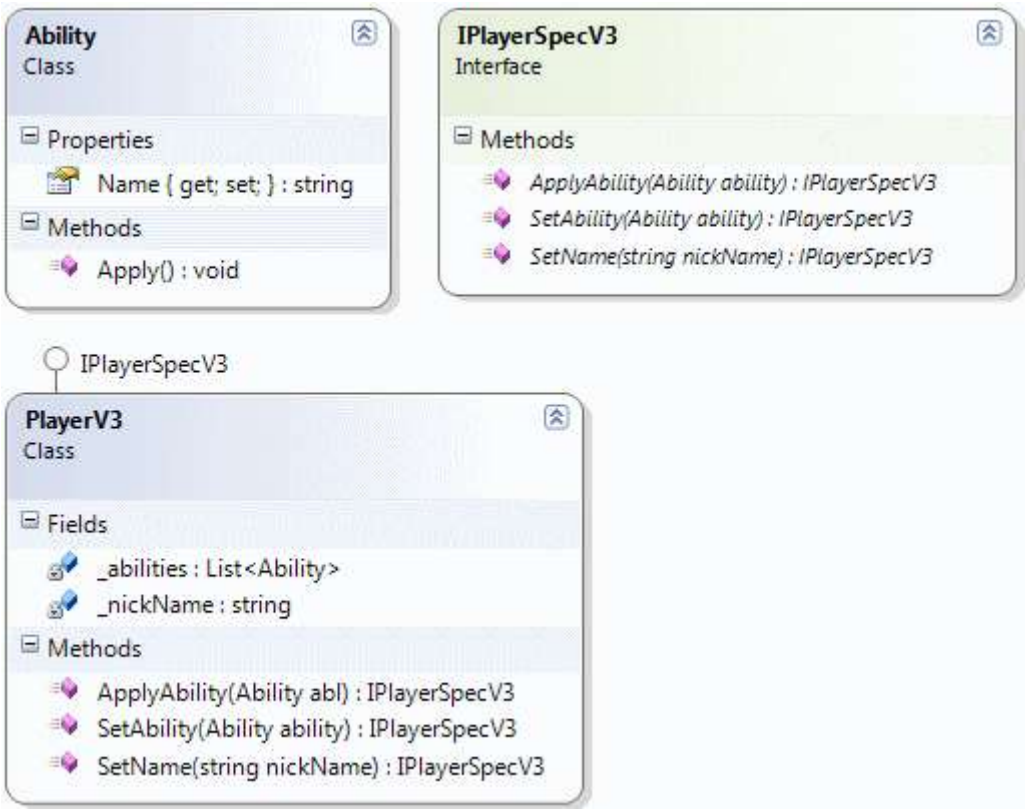
Bu sefer çok daha ilginç bir kod parçası ile karşı karşıyayız. İlk olarak **IPlayerSpecV2** isimli arayüz tipi içerisine bakmamızda yarar olacağı kanısındayım. Görüldüğü üzere burada tanımlı olan özellikler yine **IPlayerSpecV2** arayüz tipinin taşıyabileceği referansları döndürmektedir. Bir başka deyişle, **IPlayerSpecV2**arayüzünü uygulayan sınıfa ait nesne örneklerinin döndürüldüğünü ifade edebiliriz. Bu durumda **PlayerV2** sınıfının içeriği de önem kazanmaktadır.

Nitekim **Interface** implementasyonu sonucu dikkat edileceği üzere **Set** ön eki ile başlayan her metod, iç değer atamalarında **this** anahtar kelimesini kullanmaktadır. Buna göre çalışma zamanında o anki **PlayerV2** nesne örneğinin kullanılması söz konusudur. Ayrıca her **Set…** metodunun sonunda **return this;** ifadesinin kullanıldığına da dikkat edilmelidir. Peki tüm bunlar ne anlama geliyor? Aslında tüm bunların ne anlama geldiğini anlamak için **Main** metodu içerisinde yer alan kod parçasını göz önüne almamız yeterli olacaktır.



Mutlaka dikkatinizi çekmiştir. **Set…** ön eki ile başlayan metodlardan hangisini kullanırsak kullanalım arkasından yine **IPlayerSpecV2** üzerinden tanımlanmış olan metodlardan

birisine erişilebilmektedir İşte size bir metod zinciri. Bu tip bir kullanım son derece doğaldır, nitekim **Set…** metodları geriye **IPlayerSpecV2** arayüzünün taşıyabileceği referansları döndürmek üzere kodlanmıştır. Bu sayede her hangibir **Set…** metod çağrısı ile başlatılan çalışma zamanı **Context**' inin alt metod çağrılarına da taşınabilmesi kolaylaşmaktadır. Ayrıca ilk başlangıçta üretilen ile son kullanılan **Context** içeriğinin aynı ve tek olması da garanti altına alınmaktadır Şimdi dilerseniz başka bir örnek daha ele alalım. İşte kodlarımız.



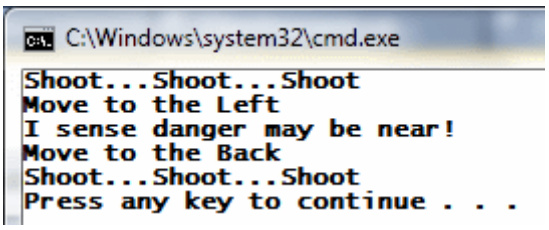

```
using System.Collections.Generic;
using System;

namespace UsingFluentInterface
{
   #region Real Sample

   class Ability
   {
      public string Name { get; set; }
      public void Apply()
      {
         Console.WriteLine("{0}",Name);
      }
   }
```

```
interface IPlayerSpecV3
{
   IPlayerSpecV3 SetName(string nickName);
   IPlayerSpecV3 SetAbility(Ability ability);
   IPlayerSpecV3 ApplyAbility(Ability ability);
}

class PlayerV3
   : IPlayerSpecV3
{
   string _nickName;
   List<Ability> _abilities = new List<Ability>();

   #region IPlayerSpecV3 Members

   public IPlayerSpecV3 SetName(string nickName)
   {
      this._nickName = nickName;
      return this;
   }

   public IPlayerSpecV3 SetAbility(Ability ability)
   {
      if (!this._abilities.Contains(ability))
         this._abilities.Add(ability);
      return this;
   }

   public IPlayerSpecV3 ApplyAbility(Ability abl)
   {
      var ability=this._abilities.Find(t => t == abl);
      ability.Apply();
      return this;
   }

   #endregion
}

#endregion

class Program
{
   static void Main(string[] args)
   {
      Ability shoot=new Ability{Name="Shoot...Shoot...Shoot"};
```

```
            Ability moveLeft=new Ability{Name="Move to the Left"};
            Ability moveBack = new Ability { Name = "Move to the Back" };
            Ability lookBack=new Ability{Name="I sense danger may be near!"};

            IPlayerSpecV3 tank = new PlayerV3()
               .SetAbility(shoot)
               .ApplyAbility(shoot)
               .SetAbility(moveLeft)
               .SetAbility(lookBack)
               .ApplyAbility(moveLeft)
               .ApplyAbility(lookBack)
               .SetAbility(moveBack)
               .ApplyAbility(moveBack)
               .ApplyAbility(shoot);
        }
    }
}
```

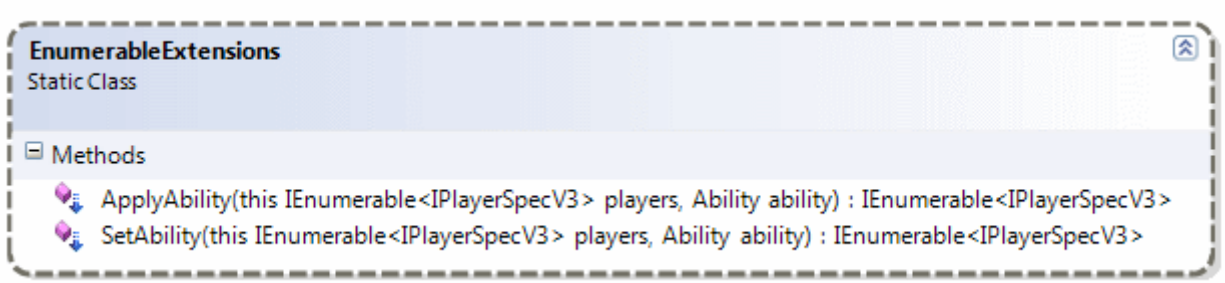
Bu sefer **IPlayerSpecV3** arayüzü içerisinde **SetAbility** ve **ApplyAbility** isimli iki metod bildirimi yapıldığı görülmektedir. **Fluent Interface** modeline göre bu metodlar geriye yine **IPlayerSpecV3**referansı döndürmektedir. örnekte ilginç olan nokta ise şudur; Esas istibariyle **PlayerV3**, bir oyuncu için eklenebilecek n sayıda farklı kabiliyeti ele alabilir ve uygulayabilir yapıda düşünülmüş ve tasarlanmıştır. Söz gelimi çalışma zamanında örneklenen **tank** değişkenine, **ateş etmek, sola dönmek, geriye bakmak, geriye hareket etmek** gibi yetenekler eklenmektedir. Ayrıca bu yeteneklerin istenildiği zaman uygulanması da sağlanmaktadır.

Bu noktada **PlayerV3** nesne örneğini üretirken yapılan metod çağrılarına dikkat etmenizi öneririm. **SetAbility** ve **ApplyAbility** metodlarının aynı ifade içerisinde, karmaşık sırada çağırılabildiği görülmektedir. Bu tipik olarak bir metod zinciridir ve aynı çalışma zamanı **Context**' ini kullanarak tankımza farklı kabiliyetlerin eklenmesi ve yürütülmesi işlemlerinin gerçekleştirilebilmesini sağlamaktadır. örneğimize ait çalışma zamanı görüntüsü aşağıdakine benzer olacaktır.



Sanıyorum ki bu son örnek **Fluent Interface** kullanımı konusunda bizlere daha fazla fikir verebilmiştir. Ancak son olarak işin içerisine bir de **Extension** metodları katarak işin kaymağını da hazırlayabiliriz

Son örneğimizde **PlayerV3** tipinden tek bir nesne örneği için metod zincirini oluşturduğumuzu görüyoruz. Oysaki **n** sayıda **tank** için de birlikte hareket etmelerini sağlayacak biçimde metod zincirleri kurmak ve kullanmak isteyebiliriz. Bu noktada elimizde bir koleksiyon yapısı olduğu düşünülebilir*(IEnumerable<PlayerV3> gibi)* Söz konusu koleksiyon yapısı için eklenecek olan bir**Extension** ile de istediğimiz fonksiyonelliği kazanma şansına sahip olabiliriz. Gelin bu amaçla aşağıdaki gibi bir kod parçası geliştirelim.



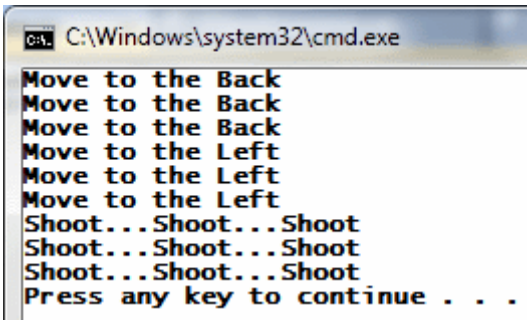

```
using System;
using System.Collections.Generic;

namespace UsingFluentInterface
{
   #region Real Sample

   public class Ability
   {
      public string Name { get; set; }
      public void Apply()
      {
         Console.WriteLine("{0}",Name);
      }
   }

   public interface IPlayerSpecV3
   {
      IPlayerSpecV3 SetName(string nickName);
      IPlayerSpecV3 SetAbility(Ability ability);
      IPlayerSpecV3 ApplyAbility(Ability ability);
   }

   public class PlayerV3
      : IPlayerSpecV3
   {
      public string PlayerName { get; set; }
      string _nickName;
      List<Ability> _abilities = new List<Ability>();
```

```
        #region IPlayerSpecV3 Members

        public IPlayerSpecV3 SetName(string nickName)
        {
            this._nickName = nickName;
            return this;
        }

        public IPlayerSpecV3 SetAbility(Ability ability)
        {
            if (!this._abilities.Contains(ability))
                this._abilities.Add(ability);
            return this;
        }

        public IPlayerSpecV3 ApplyAbility(Ability abl)
        {
            var ability=this._abilities.Find(t => t == abl);
            ability.Apply();
            return this;
        }

        #endregion
    }


    #endregion

    #region Extensions for Fluent Interface

    public static class EnumerableExtensions
    {
        public static IEnumerable<IPlayerSpecV3> SetAbility(this
IEnumerable<IPlayerSpecV3> players,Ability ability)
        {
            foreach (IPlayerSpecV3 player in players)
            {
                player.SetAbility(ability);
            }
            return players;
        }

        public static IEnumerable<IPlayerSpecV3> ApplyAbility(this
IEnumerable<IPlayerSpecV3> players,Ability ability)
        {
```

```
        foreach (IPlayerSpecV3 player in players)
        {
           player.ApplyAbility(ability);
        }
        return players;
     }
  }

  #endregion

  class Program
  {
     static void Main(string[] args)
     {
        Ability shoot=new Ability{Name="Shoot...Shoot...Shoot"};
        Ability moveLeft=new Ability{Name="Move to the Left"};
        Ability moveBack = new Ability { Name = "Move to the Back" };
        Ability lookBack=new Ability{Name="I sense danger may be near!"};

        List<IPlayerSpecV3> aTeam = new List<IPlayerSpecV3>
        {
           new PlayerV3{ PlayerName="Red Leader"},
           new PlayerV3{PlayerName="Orange 1"},
           new PlayerV3{PlayerName="Victor Echo 2"}
        };

        aTeam
           .SetAbility(shoot)
           .SetAbility(moveLeft)
           .SetAbility(moveBack)
           .SetAbility(shoot)
           .ApplyAbility(moveBack)
           .ApplyAbility(moveLeft)
           .ApplyAbility(shoot);
     }
  }
}
```

Bu örnekte **IEnumerable<IPlayerSpecV3> arayüzü(Interface)** için geliştirilmiş iki **Extension** metod kullanılmaktadır. Bu genişletme metodları tahmin edeceğiniz üzere **IPlayerSpecV3**arayüzü tarafından taşınabilen*(ki örneğimizde bu PlayerV3 sınıfına ait nesne örnekleridir)* referanslar
üzerinde **SetAbility** ve **ApplyAbility** fonksiyonelliklerinin uygulanabilmesini sağlamaktadır. Bu sayede **PlayerV3** tipinden nesne örneklerinden oluşan bir koleksiyon üzerinde toplu olarak **Fluent Interface** etkisi gerçekleştirilebilir.

Söz gelimi **Main** metodu içerisindeki kod parçalarına dikkat edildiğinde **aTeam** isimli nesne örneği üzerinden **SetAbility** ve **ApplyAbility** çağrıları gerçekleştirilmiş ve sonuç olarak **Red Leader, Orange 1** ve **Victor Echo 2** isimli tanklar için ortak yürütme işlemleri icra ettirilmiştir. Aşağıdaki örnek ekran çıktısında bu durum daha net bir şekilde görülebilmektedir.



Sanıyorum ki **Fluent Interface** kullanımı şimdi daha da bir anlam kazanmış oldu İlerleyen zamanlarda bu tip uç prensipleri incelemeye devam etmeye çalışacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

UsingFluentInterface.rar (30,94 kb)

Daha iyi Kodlama için Basit Öneriler (2010-12-19T18:01:00)

c#,c# temelleri,.net framework,

Merhaba Arkadaşlar,

Bilenler bilir, uzun süredir **.Net Framework 2.0** üzerinde yazılmış ve **Visual Studio 2005** ortamında geliştirilmeye devam edilen bir bankacılık uygulamasında görevliyim. Buradaki işim müşterinin yeni isteklerini sisteme katmak/katmaya çalışmak olarak düşünülebilir

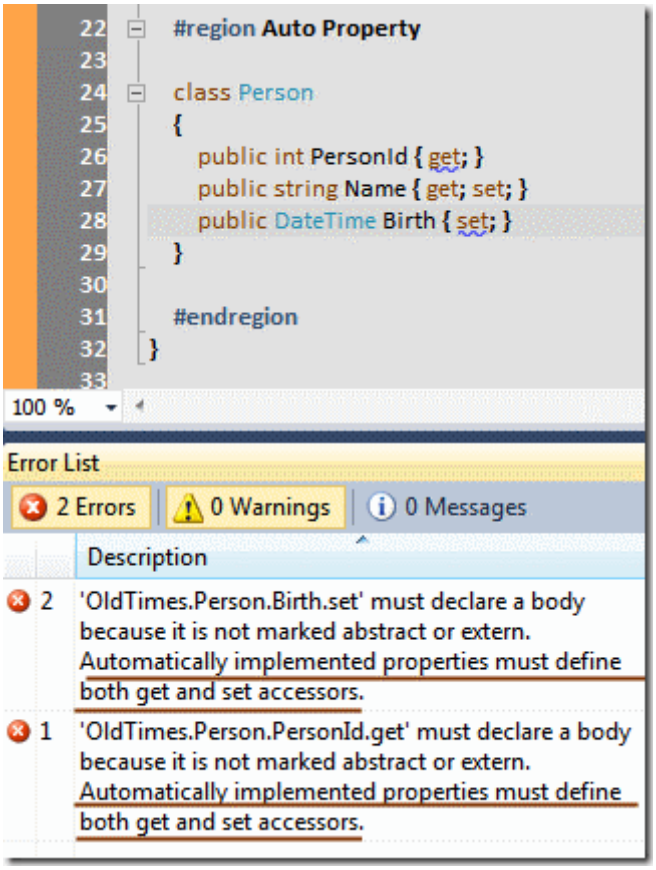
Aslında bu yeni görevime atanırken **Visual Studio 2010** ile uğraştığımı hatırlıyorum da…Kendimi **Ice Age** filmindeki buz çağı devrine dönmüş canlı bir yaşam formu gibi zannetmiştim

Geliştirilmiş olan ürüne ait proje kodları biraz eski tarihli olduğundan ilk zamanalarda içinde kaybolduğumu belirtebilirim. Günlerce **Debug** yapıp süreçlerin nasıl işlediğini anlamaya çalıştım. (E haliyle koda **dökümantasyon** yazmassanız, **XML Comment'** ler koymassanız, proje geliştirilmekteyken **sürekli eleman sirkülasyonuna** izin verir ve herkesin günü kurtarmak adına kodlama yapmasına davetiye çıkartırsanız,**analiz dökümanlarını** saklamaz veya güncellemesseniz olacağı da budur)

Aslında daha okunaklı, daha efektif ve zaman zaman daha verimli kod üretmek için bir kaç küçük noktaya dikkat etmekte yarar olabilir. Bu sayede sanıyorum ki kodlarımız en azından daha şık duracaktır Gelin bu bir kaç küçük püf noktadan bir kaçına hep birlikte bakalım.

1 – Auto Property' ler ve Private Block' lar

İlk olarak özellikle **POCO(Plain Old CLR Objects)** gibi tip tanımlamalarında yaygın olarak kullandığımız ve özellikle **LINQ(Language Integrated Query)** tarafında da çok işimize yarayan**Auto Property** kavramına bir bakalım. Bazı durumlarda özelliklerin **sadece okunabilir(Read Only)** ya da **yazılabilir (Write Only)** olmalarını isteyebiliriz. **Auto Property** kullanmadığımız durumlarda **get** veya **set** bloklarını yazmayarak bu kolayca sağlanabilir. Ancak **Auto Property**' lerde **get** veya **set** bloklarının her ikisinin de yazılması zorunludur. Buna rağmen normal **Property** yazımındaki felsefe ile hareket ettiğimizde derleme zamanında aşağıdaki ekran görüntüsünde yer alan hata mesajı ile karşılaşırız.

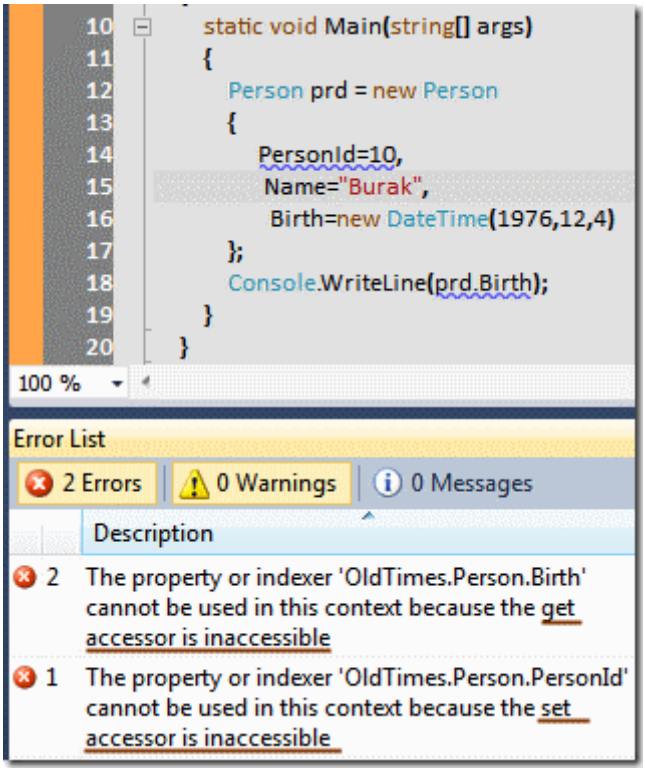


Görüldüğü üzere sadece **get** veya sadece **set** bloğu yazarak **readonly** veya **writeonly** özellik tanımlanamaz. Ancak **private** gibi bir **erişim belirleyicisini(Access Modifier)** devreye alırsak amacımıza ulaşabiliriz. İşte örnek bir kod parçası.

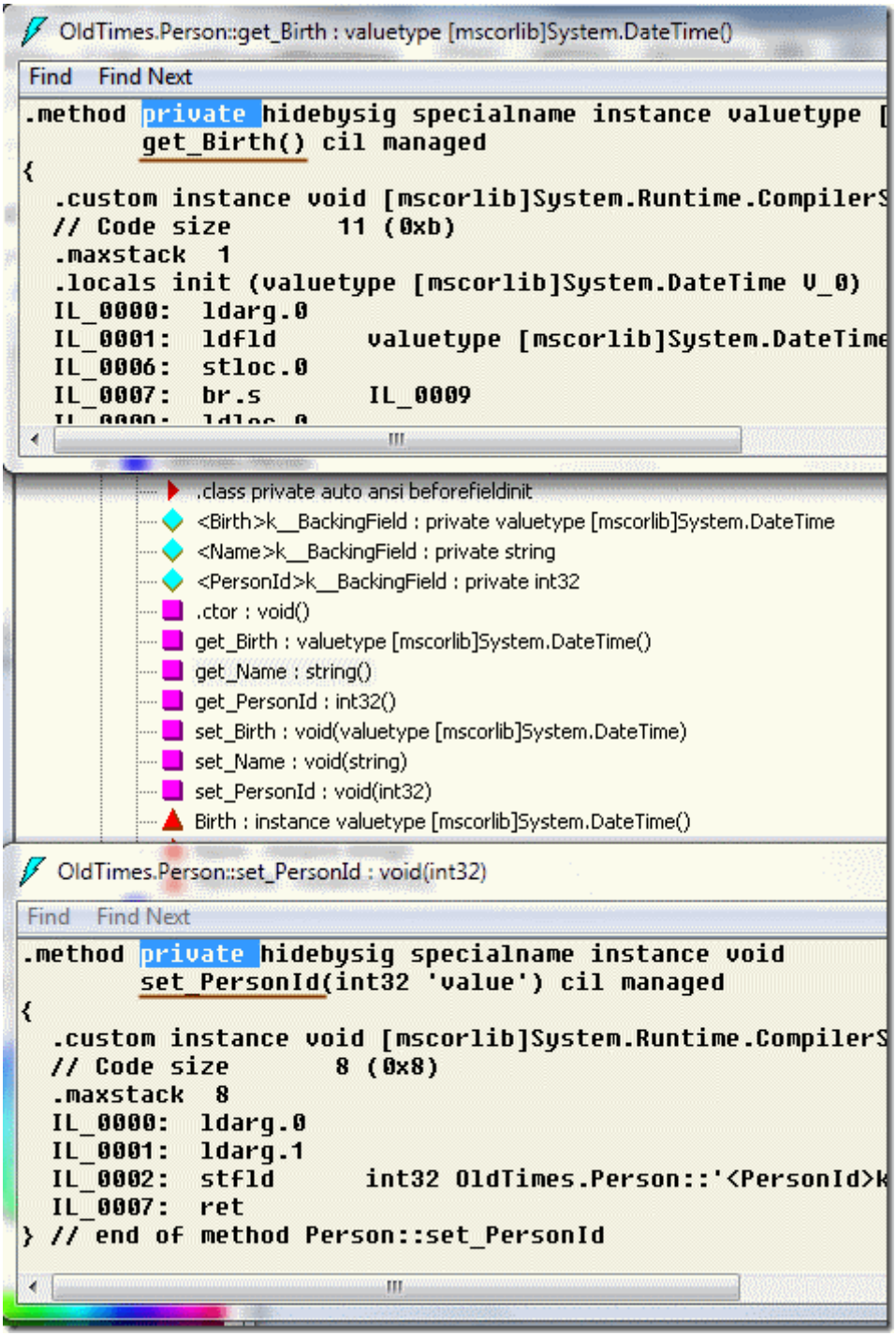
using System;

```
namespace OldTimes
{
   class Program
   {
      static void Main(string[] args)
      {

         #region Auto Property ve Private get,set

         Person prd = new Person
         {
             PersonId=10,
              Name="Burak",
               Birth=new DateTime(1976,12,4)
         };
         Console.WriteLine(prd.Birth);

         #endregion

      }

   }

   #region Auto Property

   class Person
   {
      public int PersonId { get; private set; }
      public string Name { get; set; }
      public DateTime Birth { private get; set; }
   }

   #endregion
}
```

Dikkat edileceği üzere **PersonId** ve **Birth** özelliklerinde **get** ve **set** bloklarının başında **private** bildirimi yapılmıştır. Bu durumda **PersonId** sadece okunabilir, **Birth** ise sadece yazılabilir özellikler olarak değerlendirilmektedir. Hatta kodun bu halini derlediğimizde aşağıdaki hata mesajlarını aldığımızı görürüz.



Peki üretilen **IL(Intermediate Language)** çıktısına baktığımızda Bu durumda aşağıdaki sonuçlar ile karşılaşırız.

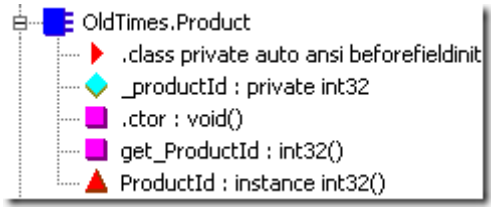


Dikkat edilmesi gereken nokta **private** olarak işaretlenmiş **get** ve **set** metodlarının varlığıdır. üstelik bu metodların içerisinde iş yapan kod parçaları da mevcuttur. Ancak bu ip ucunda dikkat edilmesi gereken bir husus daha vardır. Bu farkı görmek için eski stilde **ReadOnly** bir özelliğin nasıl tanımlandığına bakalım.

```
namespace OldTimes
{
   class Product
```

```
    {
        private int _productId=0;

        public int ProductId
        {
            get { return _productId; }
        }
    }
...
```

Product tipi içerisinde yer alan **ProductId** özelliği **Readonly** olarak tanımlanmıştır. Nitekim bir **set** bloğu yoktur. Bunun doğal olarak **IL** çıktısı ise aşağıdaki gibi olacaktır.



Farkı görebildiniz mi? Dikkat edileceği üzere **set_ProductId** isimli bir metod **IL** üretimi içerisine dahil edilmemiştir. Belki de **auto property'** lerin en sevmediğim yanı budur. Keşke sadece **get**veya sadece **set** tanımlamasına izin verilseymiş.

2 - ?? Operatörü

Zannedersem, **C#** dili içerisinde atalarından gelen **ternary operatörünü(?:)** bilmeyen, hatta kullanmayan yoktur. Tamam tamam kabul ediyorum. çoğu durumda **if…else…** bloklarını kullanmayı tercih etmekteyiz. Ancak **?: operatörü** kadar ilginç olan ve genellikle pek kullanmadığımız bir operatörümüz daha vardır. **Null kontrol operatörü(??)**. İlk olarak aşağıdaki kod parçasını göz önüne alarak işe başlayalım.

```
using System;

namespace OldTimes
{
    class Program
    {
        static void Main(string[] args)
        {
            #region ?? Operator

            IPv4 incomingIp=null;
            SendPacket(incomingIp);
            #endregion
        }
```

```
    #region ?? Operator

    static void SendPacket(IPv4 targetAddress)
    {
       IPv4 ip=null;
       if (targetAddress == null)
       {
          ip = new IPv4 { Block1 = 127, Block2 = 0, Block3 = 0, Block4 = 1 };
       }
       Console.WriteLine("{0} adresine gönderim yapılıyor",ip.ToString());
    }

    #endregion
  }

  #region ?? Operator
  class IPv4
  {
     public byte Block1 { get; set; }
     public byte Block2 { get; set; }
     public byte Block3 { get; set; }
     public byte Block4 { get; set; }

     public override string ToString()
     {
        return String.Format("{0}.{1}.{2}.{3}", Block1, Block2, Block3, Block4);
     }
  }
  #endregion
}
```

çoğu zaman bazı referans örneklerinin **null** değere sahip olup olmadıklarını denetlememiz gerekir. Yukarıdaki örnek kod parçasında bu durum analiz edilmeye çalışılmaktadır.**SendPacket** isimli **static** metod **IPv4** isimli sınıfa ait bir nesne örneğini parametre olarak alır. Metod içerisinde öncelikli olarak bir **null** kontrolü yapılır. Bunun için de standart ve yaygın yol **if…else** blokları kullanılmıştır. Aslında metod içerisinde **ternary** operatörü kullanılaraktan da aynı işlem yapılabilir.

```
ip = targetAddress == null ? new IP { Line1 = 127, Line2 = 0, Line3 = 0, Line4 = 1 } :
targetAddress;
```

? işaretinden önce **targetAddress** değişkeninin **null** olup olmadığına bakılmaktadır. **?** ile **:** işaretleri arasındaki kısım koşulun **true** olması halini ele alırken, **:** işaretinden sonraki kısım **false** olma durumunu değerlendirmektedir. Aslında burada **null** kontrolü yapıldığından **??** operatörü de ele alınabilir. Aşağıdaki gibi

ip = targetAddress ?? new IP { Line1 = 127, Line2 = 0, Line3 = 0, Line4 = 1 };

Daha yalın daha okunaklı olduğunu ifade edebilir miyiz? Bu sorunun cevabı duruma göre değişir. Ancak en azından şirketinizin bir kod standardı var ise, bu tip **null** kontrollerinde nasıl bir yol izlenilmesi gerektiği ve hangi operatörlerin kullanılması gerektiği açıktır. Söz gelimi daha önceden çalıştığım şirketlerin birisinde **?:** operatörünün kullanımı yasaklanmıştır. (Sebebini hiç sormayın ama sonuç itibariyle bir standart vardı en azından )

3 – As operatörü

Hazır **??** operatörüne değinmişken **as** operatörünü de es geçmemek taraftarıyım. Yine projelerimizde pek çok nokta da bir referans örneğinin **null** olmaması halinde dönüştürme işlemi yapılarak ilerlenilmesi söz konusudur. Aşağıdaki kod parçasını göz önüne alarak 3ncü maddemizi kavramaya çalışalım.

```
using System;

namespace OldTimes
{
   class Program
   {
      static void Main(string[] args)
      {
         #region AS Operator

         DrawShape(new Rectangle(), 10, 5);
         DrawShape(new Circle(), 3, 6);

         #endregion
      }

      #region AS Operator

      static void DrawShape(Shape shape,int x,int y)
      {
         Rectangle rect = null;
         Circle crcl = null;

         if (shape is Rectangle)
         {
            rect = (Rectangle)shape;
            rect.Draw(x, y);
         }
         else if (shape is Circle)
```

```
        {
            crcl = (Circle)shape;
            crcl.Draw(x, y);
        }
    }

    #endregion
}

#region AS Operator

public class Shape
{
}

public class Circle
    : Shape
{
    public void Draw(int x, int y)
    {
        Console.WriteLine("{0}:{1} koordinatına Daire",x,y);
    }
}

public class Rectangle
    : Shape
{
    public void Draw(int x, int y)
    {
        Console.WriteLine("{0}:{1} koordinatına dörtgen", x, y);
    }
}

#endregion
}
```

örnek kod içerisinde yer alan **DrawShape** metodu parametre olarak **Shape** sınıfında bir örnek almakadır. İçeride ise gelen örneğin **Rectangle** veya **Circle** olup olmadığına bakılarak bir dönüştürme işlemi gerçekleştirilmekte ve sembolik bir çizim işlemi yürütülmektedir. Ancak dikkat edilmesi gereken nokta önce **is** operatörü ile bir tip kontrolü yapılması sonrasında ise eğer tip beklenen tip ise bilinçli bir tip dönüştürme işlemi yapılmasıdır. Bir başka deyişle **is** operatörü de kontrol için bir dönüştürme işlemi ele almakta ve bu da gereksiz yere iki tip dönüşümüne neden olmaktadır. Aslında bu kod yerine aşağıdaki teknikte uygulanabilir.

```
Rectangle rect = shape as Rectangle;
Circle crcl = shape as Circle;

if (rect != null)
   rect.Draw(x,y);
else if (crcl != null)
   crcl.Draw(x,y);
```

as operatörü sağ tarafındaki değişkeni sol tarafındaki tipe dönüştürmeye çalışır ve bunda başarılı olamassa geriye **null** değer döndürür. Dolayısıyla tek bir kontrol ve koşul doğru ise dönüştürerek atama işlemi gerçekleştirilmektedir. Sonrasında ise tek yapılması gereken **null** kontrolünü gerçekleştirmek ve buna göre ilgili operasyonu devreye almaktır.

4 – Timespan Yardımcı Metodları

Aslında **.Net Framework** içerisinde yer alan tipleri iyi bir şekilde öğrenmek ve bilmekte her zaman için yarar vardır. Hiç ummadığımız yerlerde işimize yarayacak fonksiyonellikler bulabiliriz. örneğin bunlardan birisi zaman farkı belirtilmesi gereken yerlerde kullanılması önerilen **TimeSpan** tipidir. Ne demek istediğimi biraz daha net anlayabilmek için gelin aşağıdaki örnek kod parçasını göz önüne alalım.

```
using System;
using System.Threading;

namespace OldTimes
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Timespan

         double r1=Calculate(1, 100, 10);
         Console.WriteLine(r1);

         #endregion
      }

      #region

      static double Calculate(int x,int y,int duration)
      {
         double result=0;

         for (int i = x; i < y; i++)
         {
```

```
            result+=i;
            Thread.Sleep(duration);
        }
        return result;
    }

    #endregion
  }
}
```

Calculate metodu sembolik olarak belirli aralıkta sayıların toplamını hesap etmek üzere çalışmakta olan bir metoddur. Metodun **for** döngüsü içerisinde ise belirli süreliğine çalışmakta olan **Thread**' in **Sleep** metodu yardımıyla uyutulması sağlanmaktadır. Aslında bu sürenin **milisaniye** cinsinden olduğunu hepimiz biliyoruz. Yine de zaman zaman unuttuğumuz anlar olabiliyor ve milisaniye miydi yoksa saniye miydi diyebiliyoruz? Dolayısıyla geliştiricinin var ise ve eğer yazılmışsa **XML Comment**' ine bakmasın gerekiyor. Gelin bu işi **Developer**' ın insiyatifinden çıkartalım. İşte **TimeSpan** kullanımı…

```
using System;
using System.Threading;

namespace OldTimes
{
  class Program
  {
    static void Main(string[] args)
    {
      #region Timespan

      double r1=Calculate(1, 5, TimeSpan.FromSeconds(1)); // o andan itibaren 1er saniye aralıklarla
      Console.WriteLine(r1);
      double r2=Calculate(1, 5, new TimeSpan(0, 0, 2)); // 0 saat 0 dakika 2 saniye aralıkla
      Console.WriteLine(r2);
      double r3 = Calculate(1, 5, TimeSpan.FromMilliseconds(50)); // o andan itibaren 50şer milisaniye aralıklarla
      Console.WriteLine(r3);
      double r4 = Calculate(1, 5, new TimeSpan(0, 0, 0, 0, 250)); // 0 gün 0 saat 0 dakika 0 saniye 250 milisaniye aralıkla

      #endregion
    }

    #region
```

```
        static double Calculate(int x,int y,TimeSpan duration)
        {
            double result=0;

            for (int i = x; i < y; i++)
            {
                result+=i;
                Thread.Sleep(duration);
            }
            return result;
        }

        #endregion
    }
}
```

Dikkat edileceği üzere **Calculate** metodunun son parametresi **TimeSpan** tipi ile değiştirilmiştir. Buna göre **Thread.Sleep** metoduna parametre olarak nekadarlık bir süre geçirileceğini belirlemek çok daha kolaydır. örnek kullanımlarda bu amaçla **TimeSpan** tipinin **static** ve **aşırı yüklenmiş yapıcı metodlarından(Overloaded Constructors)** yararlanılmaktadır. Buna göre **1 saniye, 2 saniye, 50 milisaniye** ve **250 milisaniye** aralıklarla işlemler yaptırılmaktadır. Hatta dilerseniz aralığı gün bazında bile belirtebilirsiniz 😃

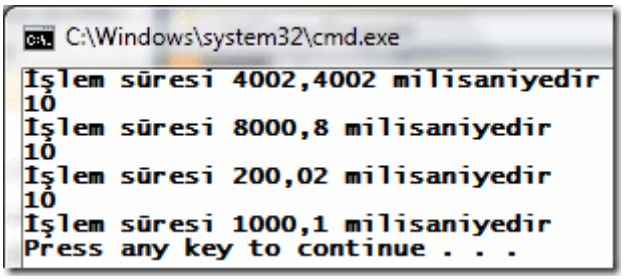
5 – Stopwatch Sınıfı

Hazır yukarıdaki kod parçasına değinmişken…Gerçektende belirtilen sürelere göre işlemlerin yapıldığını nasıl ölçebiliriz hiç düşündünüz mü? 🙂 Yani 1' den 5' e kadar olan sayıların toplamında her bir iterasyonda 2 saniye duraksama olursa tahminlerimize göre 8 saniye kadar bir toplam süre söz konusu olmalıdır öyle değil mi? Hatta bu 8 saniye de 8000 milisaniye olarak düşünülebilir. Bunu ölçmek için **Calculate** metodunun içerisinde aşağıdaki düzenlemeleri yaptığımızı düşünebiliriz.

```
static double Calculate(int x,int y,TimeSpan duration)
{
    DateTime startTime = DateTime.Now;
    double result=0;

    for (int i = x; i < y; i++)
    {
        result+=i;
        Thread.Sleep(duration);
    }
    DateTime endTime = DateTime.Now;
    TimeSpan distance = endTime - startTime;
```

```
    Console.WriteLine("İşlem süresi {0} milisaniyedir",distance.TotalMilliseconds);
    return result;
}
```

İşte sonuçlar;

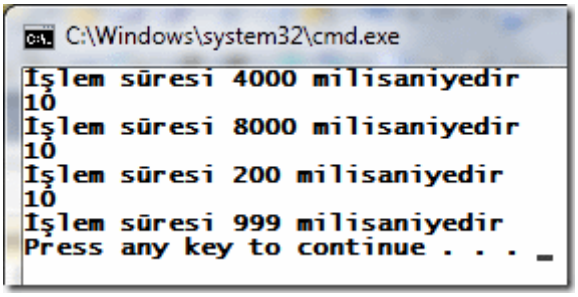


Görüldüğü gibi **DateTime** tipine ait iki nesne örneğinden yararlanılmış ve aradaki farka bakılarak işlemin ne kadar zaman aldığı hesap edilmiştir. Peki daha şık bir kodlama söz konusu olabilir mi? Evet olur

```
static double Calculate(int x,int y,TimeSpan duration)
{
    Stopwatch watcher = Stopwatch.StartNew();
    double result=0;

    for (int i = x; i < y; i++)
    {
        result+=i;
        Thread.Sleep(duration);
    }
    watcher.Stop();
    Console.WriteLine("İşlem süresi {0} milisaniyedir",watcher.ElapsedMilliseconds);
    return result;
}
```

Bu kez başlangıç noktasında **static StartNew** metodu ile bir **StopWatch** nesnesi örneklenmiştir. Bu nesne örneği devam eden kod satırları boyunca bir kronometre görevini üstlenmektedir. İşlemler tamamlandığında ortaya çıkan süre farkını hesap etmek için **Stop** metodu kullanılmaktadır. **Stop** metoduna yapılan çağrı sonrasında ise **watcher** nesne örneğinin**Elapsed** ön ekli özelliklerinden yararlanılarak ilgili süre farklarına ulaşılmaktadır.

Görüldüğü üzere başlangıç ve bitiş zamanlarını tespit etmek için iki farklı **DateTime** değişkeni tanımlamak yerine, zaten bu tip kronometre işlemleri için tasarlanmış**StopWatch** tipini kullanmak kodun anlamsal bütünlüğü açısından daha verimli görülmektedir. Hatta çalışma zamanı sonuçlarına baktığımızda **StopWatch** tipinin aslında daha tutarlı değerler döndürdüğünü de görebiliriz.



Anlattıklarımızdan yola çıkacak olursak eğer şu tip sonuçlara da varabiliriz;

- Kodlama yaparken şirketimizin belirlediği standartlar var ise bunlara harfiyen uymakta,
- Kodlama yaparken ilerleyen yılllarda başka bir geliştiricinin bakabileceğini düşünerek yazmakta,
- Bir önceki maddede belirttiğimiz ve ilerleyen yıllarda proje başına gelecek geliştiricinin, Dummy User olduğunu varsayarak, olayı iyi anlayabilmesi için ne kadar yorum satırı, XML Comment gerekiyorsa(tabi abartmadan) yazmakta,
- özellikle .Net Framework içerisinde ki tipleri boş vakitlerimizde araştırıp, var olan bir projenin hangi noktasında avantaj sağlayabileceğini incelemekte

yarar vardır

Peki başka ip uçları olabilir mi? Elbette olabilir. Bunları da ilerleyen yazılarımızda incelemeye devam edeceğiz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

OldTimes.rar (25,69 kb) **[örnek Visual Studio 2010 Ultimate ile geliştirilmiş ve test edilmiştir]**

jQuery İçerisinden Bir WCF Servisini Kullanmak (2010-12-19T08:27:00)

wcf,jquery,

Kahramanımız **Netspecter Malezyadaki** müşt erisi ile buluşmak üzere **café'** de beklerken, detektiflik işine girdiğinden beri en çok sevdiği içecek olan **Java Chip Chocolate'** ını keyifli bir şekilde yudumlamaktadır.

Güneş batmış ve hava çoktan kararmıştır. Müşterileri çoğunlukla buluşmalara geç kalır. Aslında çevrede oturanları gizlice gözlemlediğinde onlardan birisinin kalkıp yanına geleceğini ve kendisini müşterisi olarak tanıtacağını gayet iyi bilmektedir. Bir açıdan

müşterilerinin aslında kendinden önce geldiğini ama tedrigin oldukları için yaklaşmakta zorlandıklarını bilmektedir. Daha önce bu çok sık başına gelmiştir.

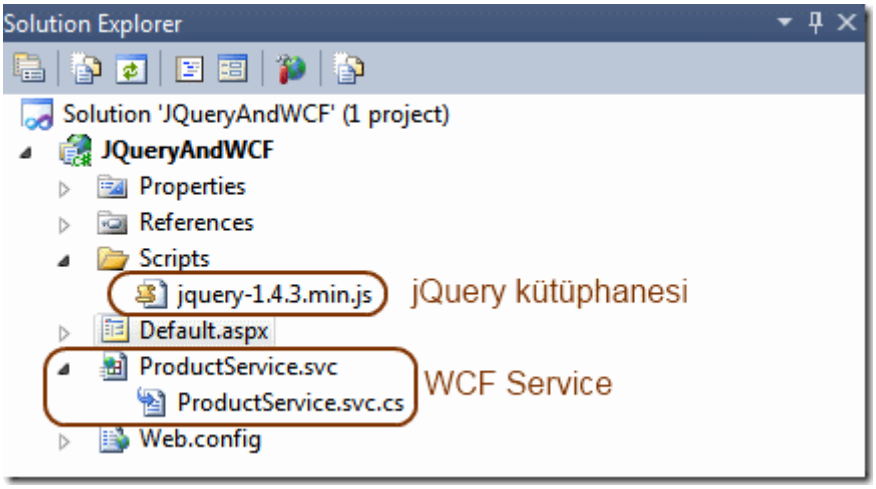
Masadaki metal peçetelikten arkasına doğru baktığında çok fazla dikkat çekmeyen ama müşterisi olabilecek bir kişinin oturduğunu uzun süre önce fark etmiştir. Cam kapıdaki yansımadan okumakta olduğu gazeteye tersten baktığı anlaşılmaktadır. Derken tahmin ettiği gibi olur…Adam ayağa kalkar ve masasına doğru yavaş ve tedirgin adımlarla yürür. **Netspecter** sakin bir şekilde **Java Chip Chocolate'** inden bir yudum daha alır. Elini pardesüsünün cebine götürür ve en gelişmiş silahı **38lik Obfuscator**' unu hazırlar. Derken adam karşısına gelir ve….

"Bay **Netspecter**…Ben müşteriniz **jQuery**" der

Merhaba Arkadaşlar,

Son yıllarda özellikle **Web** uygulamalarında **jQuery**' nin oldukça fazla yaygınlaştığını görmekteyiz. özellikle **Asp.Net MVC(Model View Controller)** disiplinin de…Ben her ne kadar**Javascript** vey **jQuery** tarafında uzman olmasam da sonuçta bu istemcilerin çağrıda bulunabileceği **WCF(Windows Communication Foundation)** servisleri olabileceğini biliyorum Dolayısıyla bu günkü yazımızda **jQuery** içerisinden **JSON(JavaScript Object Notation)** ve **XML(eXtensible Markup Language)** formatında veri sunan operasyonlara sahip bir **WCF** servisinin nasıl kullanılabileceğini incelemeye çalışıyor olacağız.

İlk olarak örneğimizde **jQuery**' nin güncel **1.4.3** sürümlü versiyonunu kullandığımızı belirtmek isterim. Söz konusu **javascript** kütüphanesinin en son sürümüne http://jquery.com/adresinden ulaşabilirsiniz. Bu kütüphaneyi çok doğal olarak bir **Asp.Net Web Application** projesi içerisinde kullanıyor olacağız.
Aşağıdaki **Solution** görüntüsünde projenin içerisinde yer alan önemli enstrümanları görebilirsiniz.



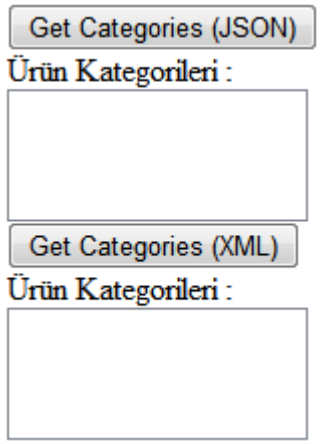
Solution içeriğinden de anlaşılacağı üzere **Scripts** klasörü altında **jQuery** kütüphanemiz yer almakta olup bir de **WCF** Servis örneği kullanılmaktadır. Dilerseniz yola **WCF** servisimizi geliştirerek devam edelim.Bu amaçla uygulamamıza **AJAX-enabled WCF Service** tipinden bir öğe ekliyoruz.

```
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;

namespace JQueryAndWCF
{
   [ServiceContract(Namespace = "http://www.azon.com/Services/Product")]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   public class ProductService
   {
      private static Category[] Categories = new Category[]{
            new Category{ CategoryId=1, Name="Kitap"},
            new Category{ CategoryId=2, Name="Dergi"},
            new Category{ CategoryId=3, Name="Dvd"},
            new Category{ CategoryId=4, Name="Cd"}
      };

      [OperationContract]
      [WebGet(ResponseFormat=WebMessageFormat.Json)]
      public Category[] GetCategoriesJson()
      {
         return Categories;
      }

      [OperationContract]
      [WebGet(ResponseFormat = WebMessageFormat.Xml)]
      public Category[] GetCategoriesXml()
      {
         return Categories;
      }
   }

   public class Category
   {
      public int CategoryId { get; set; }
      public string Name { get; set; }
   }
}
```

ProductService isimli **Ajax-Enabled WCF Service** içerisinde **GetCategoriesJson** ve **GetCategoriesXml** isimli iki servis operasyonu yer almaktadır. Adlarından da anlaşılacağı üzere, **JSON** ve**XML** formatında veri döndürmek üzere tasarlanmış bu operasyonlar, **Categories** isimli **Category** tipinden bir diziye ait çalışma zamanı örneğini geriye döndürmektedirler.

JSON formatında veri çıktısı üretmek için, dikkat edileceği üzere **WebGet** niteliğinin **ResponseFormat** özelliğine **WebMessageFormat.Json** sabit değeri atanmıştır. Benzer şekilde **XML** çıktısı üretmek içinde söz konusu sabitin **WebMessageFormat.Xml** değeri kullanılmıştır. Tabi **WebGet** niteliğini kullanmış olmamız nedeni ile söz konusu servis operasyonlarına **HTTP** protokolünün **GET** metodu yardımıyla erişilmesi söz konusudur. Bu önemlidir nitekim **jQuery** içerisinden yapacağımız servis çağrısında **GET** metoduna göre talepte bulunulması gerekmektedir.

Dilerseniz **Default.aspx** sayfamızı geliştirerek ilerlemeye çalışalım. İlk etapta amacımız **jQuery** içerisinde **JSON** ve **XML** formatlı çıktıları çekebilmek olacaktır. Sonraki aşamada ise gelen veri içeriğini istemci tarafında değerlendirmeye gayret edeceğiz. **Asp.Net** sayfasını aşağıdaki şekilde görüldüğü gibi tasarlayabiliriz.



<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs" Inherits="JQueryAndWCF.Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script type="text/jscript" src="Scripts/jquery-1.4.3.min.js">
</script>

<script language="javascript" type="text/javascript">

function GetCategoriesJson() {

```
$.ajax(
{
   type:"GET",
   url:"ProductService.svc/GetCategoriesJson",
   data:"{}",
   contentType: "application/json; charset=utf-8",
   dataType: "json",
   success: OnSuccessJson,
   error: OnErrorJson
});

}

function OnSuccessJson(data, status) {
   var result = data;
}

function OnErrorJson(data, status) {
   alert("Exception");
}

function GetCategoriesXml() {

   $.ajax(
{
   type: "GET",
   url: "ProductService.svc/GetCategoriesXml",
   data: "{}",
   contentType: "application/xml; charset=utf-8",
   dataType: "xml",
   success: OnSuccessXml,
   error: OnErrorXml
});

}

function OnSuccessXml(data, status) {
   var result = data;
}

function OnErrorXml(data, status) {
   alert("Exception");
}

</script>
```

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
     <input type="button" OnClick="GetCategoriesJson()" value="Get Categories
(JSON)" />
      <br />
      ürün Kategorileri : <br />
      <asp:ListBox ID="lstCategoriesJson" runat="server"  Width="150px"/>
      <br />
      <input type="button" OnClick="GetCategoriesXml()" value="Get Categories
(XML)" />
      <br />
      ürün Kategorileri : <br />
      <asp:ListBox ID="lstCategoriesXml" runat="server"  Width="150px"/>
   </div>
   </form>
</body>
</html>
```

Evetttt Bakalım burada neler yaptık?

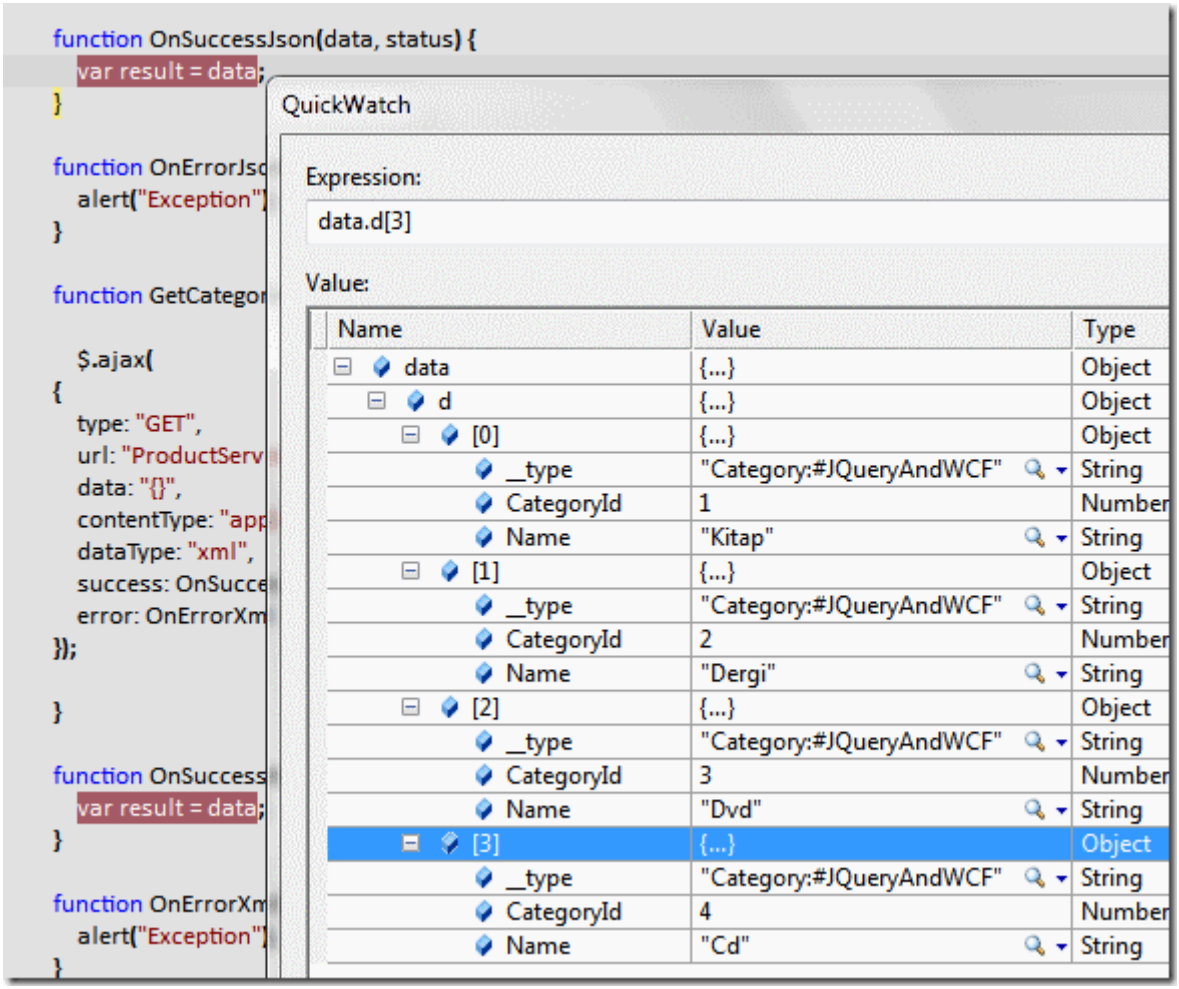
İlk olarak **jQuery** kullanmak istediğimiz için bu kütüphaneyi bildirimemiz gerekmektedir. Dolayısıla ilk **script** elementi içerisinde ilgili **jQuery** dosyasını işaret etmemiz şart. Bunun dışında **javascript** fonksiyonlarına baktığımızda **GetCategoriesJson** ve **GetCategoriesXml** isimli iki önemli operasyon olduğunu görmekteyiz. Her iki fonksiyonda kendi içerisinde **$.ajax(** ile başlayan bir metod çağrısında bulunmaktadır ki yazının kalbi de bu kod içeriğidir.

Bu çağrı içerisinde tahmin edileceği üzere **WCF** servisinin ilgili operasyonuna bir talepte bulunulmaktadır. **type** özelliğine atanan değer ile **HTTP Get** metoduna göre bir talep gönderileceği işaret edilmektedir. **url** özelliğinde ise servis adı ve çağırılacak olan servis operasyonu bilgisi tanımlanır. Servis tarafında tasarlamış olduğumuz operasyonlar herhangibir parametre almadığından, **data** özelliğine bir değer gönderimi yapılmamıştır. Ancak parametre söz konusu olduğunda **data** özelliğine gerekli değişken bildirimlerinin de yapılması gerekir.**contentType** ile servisten dönen içerik tipi belirlenirken **dataType** özelliği ile de dönen verinin tipi belirlenmiş olur.

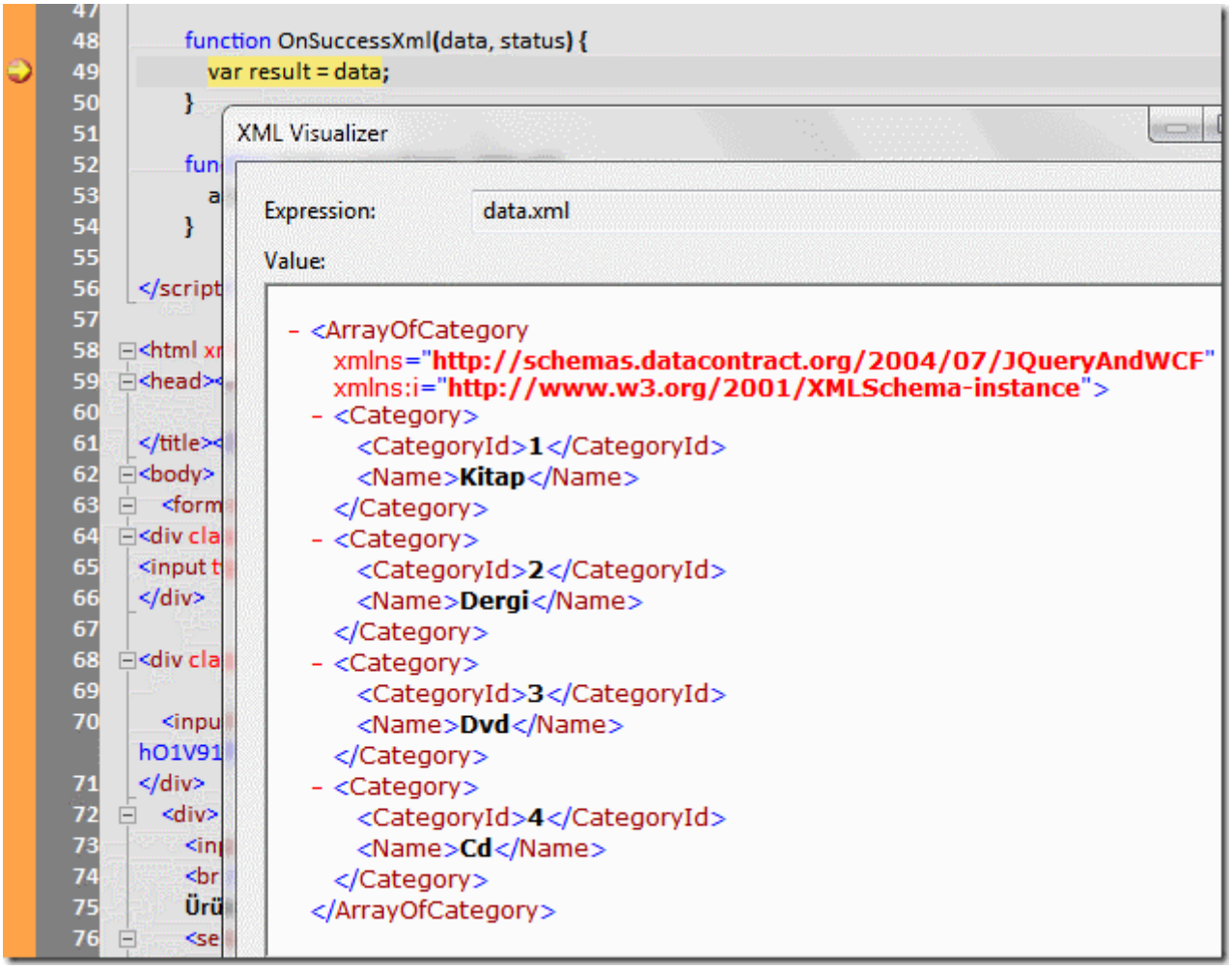
Diğer yandan yapılan bu servis çağrılarının sonuçlarını iki şekilde değerlendirmemiz gerekmektedir.. Bu amaçla servis operasyonuna yapılan çağrı başarılı bir şekilde tamamlandıysa **success**, eğer hatalı şekilde sonlandıysa da **error** özelliklerine atanan **javascript** fonksiyonlarını çağırılmaktadır. Tabiki bu **javacript** fonksiyonlarını

çağırmak için örneğimizde iki adet **Button** kontrolü ele alınmıştır. Bu kontroller dikkat edileceği üzere **Postback** işlemi yapan **ASP.NET** sunucu kontrollerinden değil aksine tipi **Button** olan standart**HTML input** elementleridir. **OnClientClick** niteliklerine atanan metod adları, **jQuery** ile ilgili servis çağrısını yapmak üzere tasarlanmış **javascript** fonksiyonlarını işaret etmektedir.

İlk olarak uygulamamızı **debug** modda çalıştırıp **OnSuccessJson** ve **OnSuccessXml** fonksiyonlarına gelen **data** parametrelerinin ne şekilde oluşturulduğunu gözlemlemeye çalışalım. İşte **JSON** çıktısı.



Dikkat edileceği üzere **Category** tipinden 4 adet nesne örneği bir dizi olarak **data** değişkeni içerisine doldurulmaktadır. Eğer **XML** formatlı talepte bulunan **Button** bileşenini kullanırsak bu durumda **OnSuccessXml javascript** fonksiyonuna gelen **data** değişkeninin içeriğinin aşağıdaki şekildeki gibi oluşturulduğunu görebiliriz.



Dikkat edileceği üzere **Category** tipinden olan dizinin içeriği **XML** formatında elde edilmektedir. Tabi **JSON** ve **XML** çıktıları farklı şekillerde ele alınmalıdır. **JSON** tarafında nesne bazlı bir yaklaşım daha kolay bir şekilde ele alınabilirken **XML** çıktısı için biraz **Node, Element, Attribute** seviyesinde düşünmek ve **Parse** işlemlerini buna göre yapmak gerekmektedir.

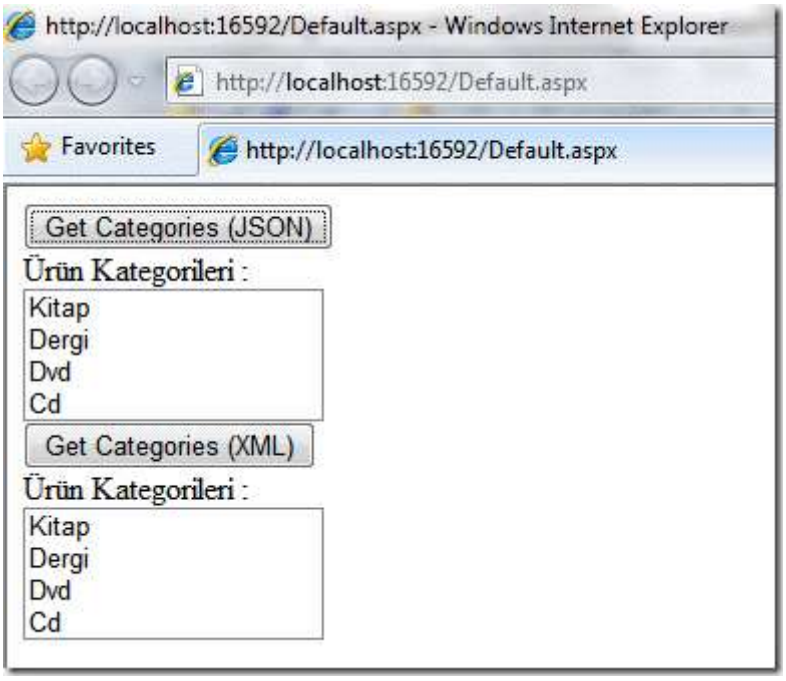
Şu an geldiğimiz noktaya bakdığımızda, **jQuery** içerisinden **WCF** servis operasyonlarının çağırılması ve **JSON** ile **XML** formatında veri alınması söz konusudur. Bir de bu verileri kullanabilirsek süper olmaz mı? Bu amaçla servis çağrılarının başarılı olması sonucu devreye giren **Success** metodlarındaki **javascript** kodlarını aşağıdaki gibi değiştirmemiz yeterli olacaktır.

```
<script language="javascript" type="text/javascript">

   function GetCategoriesJson() {

      $.ajax(
      {
         type: "GET",
         url: "ProductService.svc/GetCategoriesJson",
```

```
        data: "{}",
        contentType: "application/json; charset=utf-8",
        dataType: "json",
        success: OnSuccessJson,
        error: OnErrorJson
    });

}

function OnSuccessJson(data, status) {
    var ddl = document.getElementById("lstCategoriesJson");

    //Minik bir temizlik
    for (i = ddl.options.length - 1; i >= 0; i--) {
        ddl.remove(i);
    }

   for (i = 0; i < data.d.length; i++) {

        var option = document.createElement("option");
        option.innerHTML = data.d[i].Name.toString();
        option.value = data.d[i].CategoryId.toString();
        ddl.appendChild(option);
    }
}

function OnErrorJson(data, status) {
    alert("Exception");
}

function GetCategoriesXml() {

    $.ajax(
    {
        type: "GET",
        url: "ProductService.svc/GetCategoriesXml",
        data: "{}",
        contentType: "application/xml; charset=utf-8",
        dataType: "xml",
        success: OnSuccessXml,
        error: OnErrorXml
    });

}
```

```
    function OnSuccessXml(data, status) {
        var ddl = document.getElementById("lstCategoriesXml");

        //Minik bir temizlik
        for (i = ddl.options.length - 1; i >= 0; i--) {
            ddl.remove(i);
        }

        for (i = 0; i < data.childNodes[0].childNodes.length; i++) {
            var option = document.createElement("option");
            option.value = data.childNodes[0].childNodes[i].childNodes[0].text;
            option.innerHTML = data.childNodes[0].childNodes[i].childNodes[1].text;
            ddl.appendChild(option);
        }
    }

    function OnErrorXml(data, status) {
        alert("Exception");
    }

</script>
```

Tabi **JSON** ve **XML** formatındaki veri dönüşleri birbirlerinden farklı şekilde ele alınmak zorundadır. özellikle **JSON** tarafında nesne bazlı bir yaklaşım söz konusu olduğundan **data**içeriğinden **Category** tipi için tanımlanmış olan **Name** veya **CategoryId** gibi çalışma zamanı özelliklerine ulaşmak oldukça kolaydır. Ne varki **XML** veri okuması sırasında **Node**' lar içerisinde*(özellikle childNodes attribute ile gelen listelerde)* dolaşmak gerekir. Her iki fonksiyonda basit anlamda sayfa üzerinde yer alan liste kontrollerini bulmakta ve **option** tipinden elementler ilave etmektedir. Bu elementlerin **innerHTML** niteliklerine **Name** , **value** niteliklerine ise **CategoryId** değerleri atanmaktadır. Uygulamamızı bu haliyle çalıştırdığımızda aşağıdaki ekran görüntüsünde yer alan sonuçları elde ederiz.



Görüldüğü gibi **jQuery** kütüphanesi içerisinden bir **WCF** servis çağrısını gerçekleştirmek ve operasyon çağrısı sonucu üretilen veriyi **javascript** tarafında ele almak son derece kolaydır. Tabi bu noktada akla gelen sorulardan biriside bu tip bir işlevselliğin **ASP.NET MVC** tarafında nasıl ele alınabileceğidir Bunu da ilerleyen yazılarımızdan birisinde ele almaya çalışıyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

JQueryAndWCF.rar (47,04 kb) **[örnek Visual Studio 2010 Ultimate sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Servis Operasyonlarını Kod Yardımıyla İzlemek - Event Kullanımı (2010-12-19T03:00:00)

wcf,windows communication foundation,

Merhaba Arkadaşlar,

Suyun dibinden bir baloncuk yükselmeye başlar. Taze ve temiz hava ile temas etmek için can atmaktadır. Ancak kat etmesi gereken epey bir mesafe vardır. Daha da hızlanabilmek ister. Yükselirken karşısına bir baloncuk daha çıkar ve doğrudan ona koşar. Artık ikisi birlikte, daha hızlı yükselmektedirler.

“Birlikten kuvvet doğar” der, alttan gelen ötekine. Ama artık havasızlıktan boğulmak üzeredirler. Neyse ki yollarına başka bir baloncuk daha çıkar. üç baloncuk birleşerek daha

büyük bir tane oluşturur. Süratler artmıştır. Gün ışığına, taze havaya çok, çok az kalmıştır. Ve nihayettt…Hımmmm!!! Temiz hava…

Bu şairane diyemeyeceğim kadar kötü girişten sonra baloncukların konumuzla ne alakası olduğunu düşünebilirsiniz elbette Aslında bu gün irdeleyeceğimiz konunun özünde Olaylar(Events) yer almakta. Ayrıca söz konusu bir olay tanımının, üst tarafta doğru*(ama kodun içerisinden baktığımızda da alt tarafa doğru yürüdüğünü düşünebiliriz)* aktarılması söz konusu. *(WPF tarafında buna sıkça rastlandığını ve Event Bubbling şeklinde ifadeler ile tanımlandığını belirtelim)*

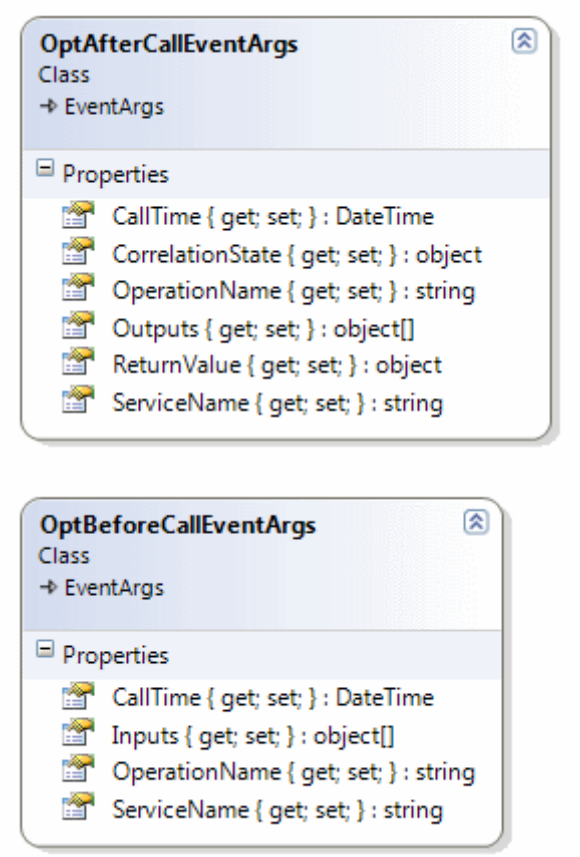
Servis Operasyonlarını Kod Yardımıyla İzlemek başlıklı yazımızda, **IIS(Internet Information Services)** dışında **Self-Hosted** modelinde yayınlanan servislere ait operasyon çağrılarının nasıl izlenebileceğini incelemeye çalışmıştık. Geliştirmiş olduğumuz tipler yardımıyla, istemcilerden servis operasyonlarına gelen çağrı öncesi ve sonrası durumları, **Text** tabanlı olarak dosyaya kaydetmiştik. **Text** tabanlı dosyaya kaydetme işlemi aslında bir zorlama olarak düşünülebilir. Nitekim söz konusu yazıda uygulanan çözüme göre, operasyon çağrıları ancak **text** tabanlı dosyaya bakılarak takip edilebilir

Oysaki operasyon çağrılarını işletim sisteminin **Event Log'** larına yazdırtmak ya da, **XML** tabanlı bir dosyaya aktarılmasını sağlamak isteyebiliriz. Hatta söz konusu izlerin veritabanı üzerindeki bir tabloya yazdırılması da düşünülebilir vb… Bir başka deyişle operasyon çağrılarının izlenmesi sırasında oluşan log verilerini, herhangibir kaynağa doğru yazdırmak isteyebiliriz. Böyle bir durumda, tasarlamış olduğumuz tiplerin, onları kullanan **object user**' lara alternatif bir yol sunması gerekmektedir. Nitekim oluşturulan **log** verisinin nereye yazılacağına **object user'** ın karar vermesi, çok daha esnek bir izleme yapısı oluşturulmasını sağlayacaktır. Peki bunu nasıl gerçekleştirebiliriz?

Aslında elimizde önemli ve büyük bir koz vardır. **Olaylar(Events). Object User'** lar*(bir başka deyişle servis operasyonlarını izlemek ile ilişkili kodlamaları yapanlar)* tasarladığımız servis davranışı içerisinde tanımlayacağımız **olaya/olaylara(Events)** abone olaraktan, üretilen **log'** ları nereye isterlerse yazdırabilirler.

Dolayısıyla yazımızın bundan sonraki kısımlarında olay tabanlı bir geliştirme yapacağımızı belirtmek isterim. Olaylar ile ilişkili olarak C# Temelleri - Olayları(Events) Kavramak başlıklı yazıya bakmanızı ve eski bilgilerinizi tazeleminizi önerebilirim

Senaryomuzda olay metodlarına bilgi aktarmamız da son derece yerinde olacaktır. Bu amaçla **EventArgs** tipinden türeyen ve aşağıdaki içeriğe sahip olan iki tip tasarladığımızı düşünebiliriz. Nitekim operasyon çağrısı öncesi ve sonrası durumlara ait bilgileri taşımak niyetinde olduğumuzdan iki ayrı argüman tipi işimizi görebilir *(veya ortak üyeleri bir üst tip içerisinde toplayacak bir yol da izleyebilirsiniz)*



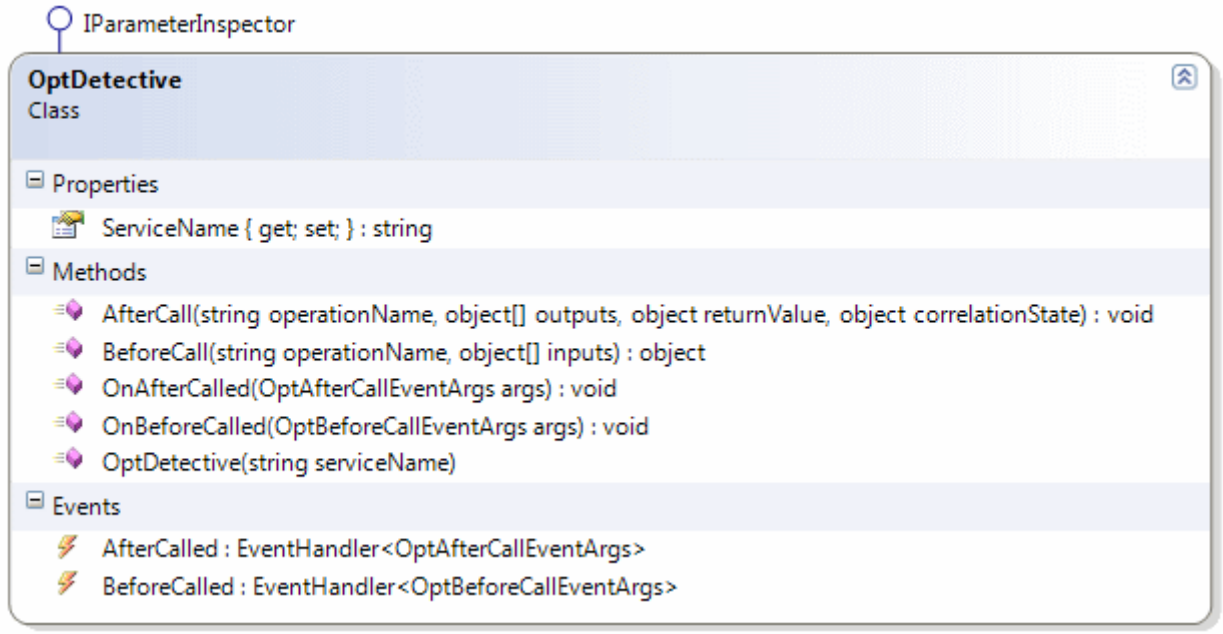
OptAfterCallEventArgs sınıfı **EventArgs** tipininden türetilmiştir ve özellikle **IParameterInspector** arayüzü üzerinden gelen **AfterCall** metodunun parametrik bilgilerini ve ek diğer verileri taşımak üzere tasarlanmıştır. Bu anlamda, operasyona yapılan çağrı sonrasında, hangi servisin hangi operasyonunun icra edilmekte olduğu, hangi değerin geriye döndürüldüğü, output parametrelerinin içeriği ve çağrı zamanı gibi bilgiler ele alınmaktadır. Sınıfın basit içeriği aşağıdaki gibidir.

```
using System;

namespace InspectorLib
{
   public class OptAfterCallEventArgs
      :EventArgs
   {
      public string ServiceName { get; set; }
      public string OperationName { get; set; }
      public object ReturnValue { get; set; }
      public object CorrelationState { get; set; }
      public object[] Outputs { get; set; }
      public DateTime CallTime { get; set; }
   }
}
```

OptBeforeCallEventArgs sınıfı da benzer şekilde **EventArgs** tipinden türetilmiştir. Bu sınıfa ait nesne örnekleri ile de, **IParameterInspector** arayüzü dolayısıyla gelen **BeforeCall**metodunun parametrik bilgileri ve ek verilerin, ilgili olay metoduna aktarılması hedeflenmektedir. Buna göre hangi servis için hangi operasyon çağrısının yapılacağı, operasyona gelen parametre bilgileri ve operasyonun icra zamanı gibi veriler toplanabilmektedir. Söz konusunu sınıfın basit içeriği de aşağıdaki gibidir.

```
using System;

namespace InspectorLib
{
   public class OptBeforeCallEventArgs
      :EventArgs
   {
      public string ServiceName { get; set; }
      public string OperationName { get; set; }
      public object[] Inputs { get; set; }
      public DateTime CallTime { get; set; }
   }
}
```

Bu işlemin ardından **IParameterInspector** arayüzünü implemente eden yeni izleyicimizin**(OptDetective sınıfı)** içeriğini de aşağıdaki gibi yazabiliriz.




```
using System;
using System.ServiceModel.Dispatcher;

namespace InspectorLib
{
```

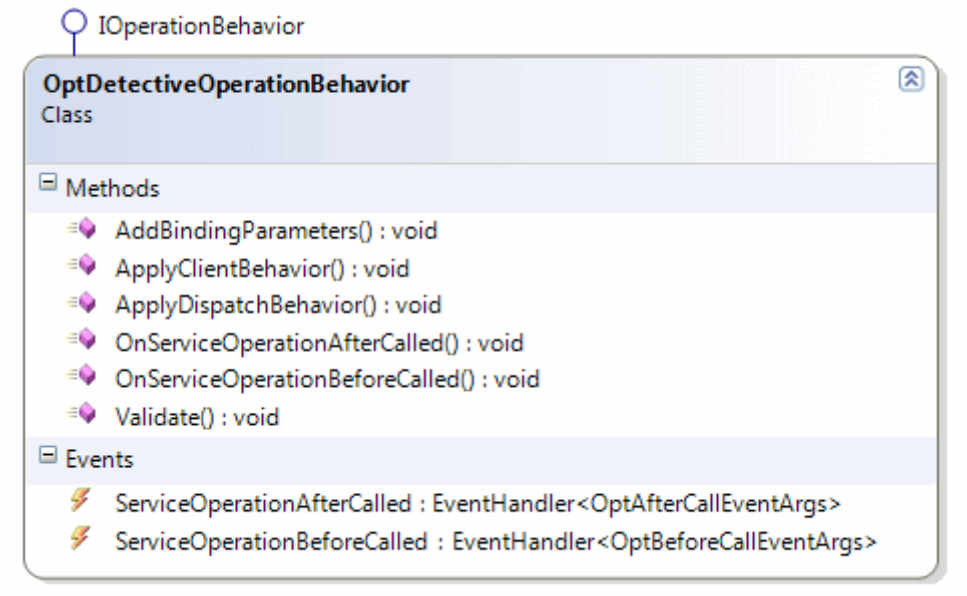
```
public class OptDetective
   :IParameterInspector
{
   public string ServiceName { get; set; }
   // Olay tanımlamaları yapılır
   public event EventHandler<OptAfterCallEventArgs> AfterCalled;
   public event EventHandler<OptBeforeCallEventArgs> BeforeCalled;

   // Olay metodları
   public void OnAfterCalled(OptAfterCallEventArgs args)
   {
      // Eğer OptAfterCalled olayına abone(Subscribe) olunmuşsa çağır
      if (AfterCalled != null)
         AfterCalled(this, args); // Olayı tetikleyen çalışma zamanındaki OptDetective
nesne örneği this ile ifade edilirken, olay metoduna taşınacak bilgiler de
OptAfterCallEventArgs tipinden olan args değişkeni ile taşınır.
   }
   public void OnBeforeCalled(OptBeforeCallEventArgs args)
   {
      // Eğer OptBeforeCalled olayına abone(Subscribe) olunmuşsa çağır
      if (BeforeCalled != null)
         BeforeCalled(this, args); // Olayı tetikleyen çalışma zamanındaki OptDetective
nesne örneği this ile ifade edilirken, olay metoduna taşınacak bilgiler de
OptBeforeCallEventArgs tipinden olan args değişkeni ile taşınır.
   }

   public OptDetective(string serviceName)
   {
      ServiceName = serviceName;
   }
   #region IParameterInspector Members

   public void AfterCall(string operationName, object[] outputs, object
returnValue, object correlationState)
   {
      // Olay metodu çağırılır ve parametre olarak OptAfterCallEventArgs tipinden nesne
örneği verilir. Bu nesne ile gerekli bilgilerin olay metodu içerisine taşınması sağlanmış
olunur.
      OnAfterCalled(
         new OptAfterCallEventArgs
         {
            ServiceName=this.ServiceName,
             CallTime=DateTime.Now,
              CorrelationState=correlationState,
               OperationName=operationName,
```

```
                Outputs=outputs,
                ReturnValue=returnValue
            }
            );
        }

        public object BeforeCall(string operationName, object[] inputs)
        {
            // Olay metodu çağırılır ve parametre olarak OptBeforeCallEventArgs tipinden
nesne örneği verilir. Bu nesne ile gerekli bilgilerin olay metodu içerisine taşınması
sağlanmış olunur.
            OnBeforeCalled(
               new OptBeforeCallEventArgs
               {
                   ServiceName=this.ServiceName,
                    Inputs=inputs,
                     CallTime=DateTime.Now,
                      OperationName=operationName
               }
               );
            return null;
        }

        #endregion
    }
}
```

OptDetective sınıfı içerisinde iki adet **event** tanımı yapıldığı görülmektedir. **AfterCalled** ve **BeforeCalled**. Ayrıca bu olayların **manuel** olarak tetiklenmesi için de, **OnAfterCalled**ve **OnBeforeCalled** isimli metodlar yazılmıştır. Her iki metod içerisinde de, bağlı oldukları olayların yüklenip yüklenmediği kontrol edilmekte ve eğer yüklüyseler çağırılmaları sağlanmaktadır.

Olaylarların tetiklenmesi sonucu **AfterCall** ve **BeforeCall** isimli arayüz metodları çağırılır. Bu çağrılar sırasında da uygun olan argüman tipleri örneklenmekte ve kullanılmaktadır. örneğin **BeforeCall** fonksiyonu çağırıldığında, **OnBeforeCalled** metodu devreye girmekte ve **OptBeforeCallEventArgs** tipinden nesne örneği ile servis adı, operasyon adı, çağrı zamanı ve operasyon input değerleri gibi bilgiler, olay metoduna aktarılmaktadır.

Daha önceki yazımızdan da hatırlayacağınız üzere, operasyon ve servis seviyesinde kullanılan özel davranış tiplerimiz bulunmaktadır. Bunlarda özel operasyon davranışı olanı,**OptDetective** tipini kullanacaktır. Ancak daha da önemlisi, özel operasyon davranışının(Custom Operation Behavior), **OptDetective** üzerindeki olaylara abone olmasıdır. Bu tipi aşağıdaki gibi tasarlayabiliriz.

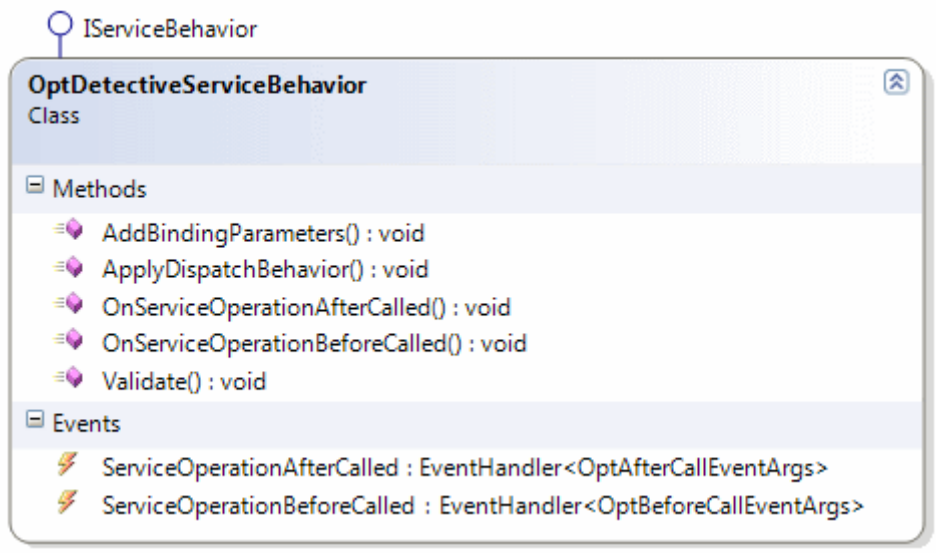



```
using System;
using System.ServiceModel.Description;

namespace InspectorLib
{
   public class OptDetectiveOperationBehavior
      :IOperationBehavior
   {
      // Olay tanımlamaları
      public event EventHandler<OptAfterCallEventArgs>
ServiceOperationAfterCalled;
      public event EventHandler<OptBeforeCallEventArgs>
ServiceOperationBeforeCalled;

      // Olay metodlar
      public void OnServiceOperationAfterCalled(OptAfterCallEventArgs args)
      {
         // ServiceOperationAfterCalled olayı yüklenmişse, bu olay sonrası devreye girecek
metodu çağır
         if (ServiceOperationAfterCalled != null)
            ServiceOperationAfterCalled(this, args);
      }
      public void OnServiceOperationBeforeCalled(OptBeforeCallEventArgs args)
      {
         // ServiceOperationBeforeCalled olayı yüklenmişse, bu olay sonrası devreye
girecek metodu çağır
         if (ServiceOperationBeforeCalled != null)
            ServiceOperationBeforeCalled(this, args);
      }
```

```
        #region IOperationBehavior Members

        public void AddBindingParameters(OperationDescription operationDescription,
System.ServiceModel.Channels.BindingParameterCollection bindingParameters)
        {
        }

        public void ApplyClientBehavior(OperationDescription operationDescription,
System.ServiceModel.Dispatcher.ClientOperation clientOperation)
        {
        }

        public void ApplyDispatchBehavior(OperationDescription operationDescription,
System.ServiceModel.Dispatcher.DispatchOperation dispatchOperation)
        {
            string serviceName = null;
            if (dispatchOperation.Parent.Type != null)
            {
                // Service adı yakalanır
                serviceName = dispatchOperation.Parent.Type.Name;
                //OptDetective tipine ait nesne örneği oluşturulur ve servis adı yapıcı metoduna
parametre olarak gönderilir.
                OptDetective optDtcv = new OptDetective(serviceName);
                // OptDetective nesne örneğinin AfterCalled ve BeforeCalled olaylarına abone
olunur. Anonymouse Method' lar yardımıyla da ilgili olay metodları tetiklenir
                optDtcv.AfterCalled += (snd,arg) => {
OnServiceOperationAfterCalled(arg); };
                optDtcv.BeforeCalled += (snd, arg) => {
OnServiceOperationBeforeCalled(arg); };
                // Operasyon için gerekli inspector tipi bildiririlir
                dispatchOperation.ParameterInspectors.Add(optDtcv);
            }
        }

        public void Validate(OperationDescription operationDescription)
        {
        }

        #endregion
    }
}
```

Dikkat edileceği üzere özel operasyon davranışı içerisinde, **OptDetective** tarafından sunulan **AfterCalled** ve **BeforeCalled** olaylarına abonelik işlemi gerçekleştirilmektedir. Bunun için**ApplyDispatchBehavior** metodu içerisinde gerekli düzenlemeler yapılmıştır.

Diğer yandan **OptDetectiveOperationBehavior** tipinin kendisi de, iki adet olay bildirimi yapmaktadır.**ServiceOperationAfterCalled** ve **ServiceOperationBeforeCalled**. Benzer bir olay bildirim ve uygulama şekli özel servis davranışı için de yapılmalıdır. Aynen aşağıda görüldüğü gibi.




```
using System;
using System.ServiceModel.Description;

namespace InspectorLib
{
      public class OptDetectiveServiceBehavior
      :IServiceBehavior
   {
      // Olay tanımlamaları
      public event EventHandler<OptAfterCallEventArgs>
ServiceOperationAfterCalled;
      public event EventHandler<OptBeforeCallEventArgs>
ServiceOperationBeforeCalled;

      // Olay metodlar
      public void OnServiceOperationAfterCalled(OptAfterCallEventArgs args)
      {
         // ServiceOperationAfterCalled olayı yüklenmişse, bu olay sonrası devreye girecek
metodu çağır
         if (ServiceOperationAfterCalled != null)
            ServiceOperationAfterCalled(this, args);
      }
      public void OnServiceOperationBeforeCalled(OptBeforeCallEventArgs args)
      {
         // ServiceOperationBeforeCalled olayı yüklenmişse, bu olay sonrası devreye
girecek metodu çağır
```

```
            if (ServiceOperationBeforeCalled != null)
                ServiceOperationBeforeCalled(this, args);
        }

        #region IServiceBehavior Members

        public void AddBindingParameters(ServiceDescription serviceDescription,
System.ServiceModel.ServiceHostBase serviceHostBase,
System.Collections.ObjectModel.Collection<ServiceEndpoint> endpoints,
System.ServiceModel.Channels.BindingParameterCollection bindingParameters)
        {
        }

        public void ApplyDispatchBehavior(ServiceDescription serviceDescription,
System.ServiceModel.ServiceHostBase serviceHostBase)
        {
            foreach (ServiceEndpoint endpoint in serviceDescription.Endpoints)
                foreach (OperationDescription operation in endpoint.Contract.Operations)
                {
                    // Custom Operation Behavior nesne örneği tanımlanır.
                    OptDetectiveOperationBehavior optBevahior = new
OptDetectiveOperationBehavior();
                    // Bu nesne örneğinin ServiceOperationAfterCalled ve
ServiceOperationBeforeCalled olaylarına abone olunur
                    optBevahior.ServiceOperationAfterCalled += (snd, args) => {
OnServiceOperationAfterCalled(args); };
                    optBevahior.ServiceOperationBeforeCalled += (snd, args) => {
OnServiceOperationBeforeCalled(args); };
                    operation.Behaviors.Add(optBevahior);
                }
        }

        public void Validate(ServiceDescription serviceDescription,
System.ServiceModel.ServiceHostBase serviceHostBase)
        {
        }

        #endregion
    }
}
```

OptDetectiveServiceBehavior tipi daha önceki yazımızdaki örnekten de bilindiği üzere servis davranışı olarak uygulanmaktadır. Bu tip içerisinde de **AfterCall** ve **BeforeCall** takipleri için birer olay metodu tanımlandığı görülmektedir. Ayrıca, **ApplyDispatchBehavior** metodu içerisinde yer alan döngüde, her bir servis

operasyonu için gerekli operasyon davranışı bildirimi yapılırken, bu tipe ait olaylara da abone olunduğu görülmektedir.

Tanımladığımız bu yeni tipler olay bazlı bir modeli kullanmaktadır. Teorimize göre **OptDetectiveServiceBehavior** tipini kullanarak yeni bir davranış yükleyen servisler, **OptDetective**içerisinde yer alan olay metodlarına erişebiliyor olmalıdır. Tabi bir önceki yazımızdan farklı olarak özel servis ve operasyon davranış tiplerinin **nitelik(Attribute)** bazlı olmadıkları görülmektedir. Bu nedenle **ServiceHost** nesne örneği için kod tarafında bilinçli olarak gerekli davranış bildirimleri yapılmalıdır. Daha önceki yazımızda kullandığımız örnek üzerinden ilerlediğimiz için aynı servis host uygulamasını ve servis tiplerini kullanabiliriz. Bu amaçla Form kodlarını aşağıdaki gibi geliştirdiğimizi düşünebiliriz.

```
using System;
using System.ServiceModel;
using System.Windows.Forms;
using AdventureWorksFinance;
using InspectorLib;

namespace HostApp
{
   public partial class Form1
      : Form
   {
      ServiceHost host=null;

      public Form1()
      {
         InitializeComponent();
         btnOpen.Enabled = true;
         btnClose.Enabled = false;
      }

      private void btnOpen_Click(object sender, EventArgs e)
      {
         host=new ServiceHost(typeof(AccountService));

         host.Opened += (s, ea) =>
         {
            lstOperationCalls.Items.Add(String.Format("Servis açık
{0}",DateTime.Now.ToLongTimeString()));
            btnOpen.Enabled = false;
            btnClose.Enabled = true;
         };
         host.Closed += (s, ea) =>
```

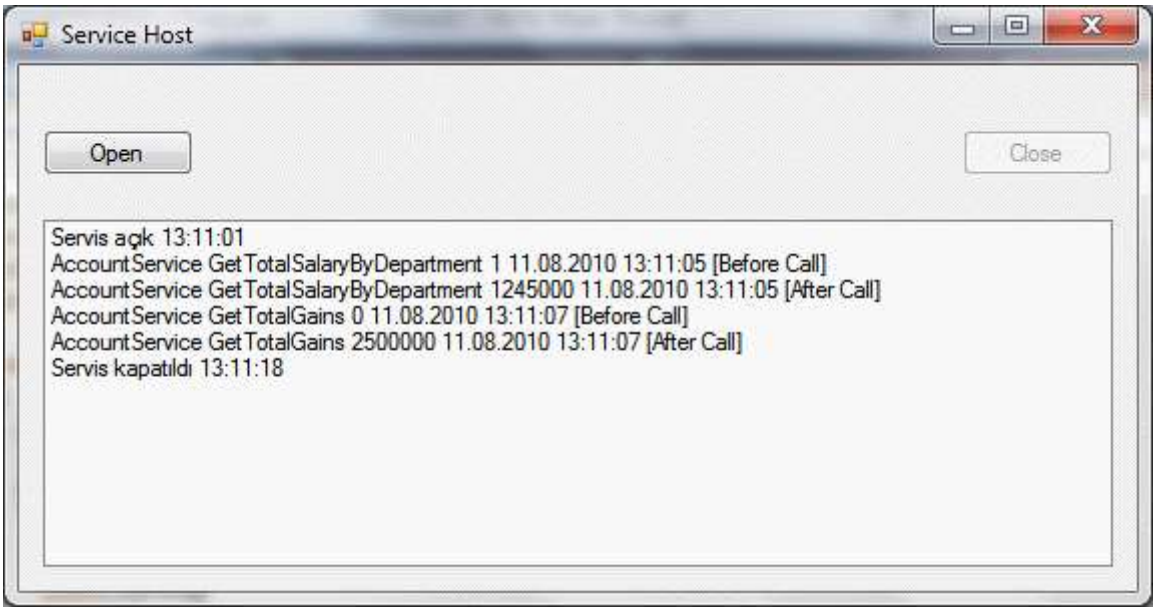
```
        {
            lstOperationCalls.Items.Add(String.Format("Servis kapatıldı {0}",
DateTime.Now.ToLongTimeString()));
            btnOpen.Enabled = true;
            btnClose.Enabled = false;
        };

        OptDetectiveServiceBehavior behavior = new OptDetectiveServiceBehavior();
        if (host.Description.Behaviors.Find<OptDetectiveServiceBehavior>() == null)
        {
            host.Description.Behaviors.Add(behavior);
        }
        // OptDetectiveServiceBehavior nesne örneği oluşturulup, servis davranışı olarak
eklendikten sonra ilgili olay metodlarına abone olunur.
        behavior.ServiceOperationAfterCalled += (snd, arg) => {
            string info = String.Format("{0} {1} {2} {3} [{4}]", arg.ServiceName,
arg.OperationName, arg.ReturnValue, arg.CallTime,"After Call");
            lstOperationCalls.Items.Add(info);
        };
        behavior.ServiceOperationBeforeCalled += (snd, arg) => {
            string info = String.Format("{0} {1} {2} {3} [{4}]", arg.ServiceName,
arg.OperationName, arg.Inputs.Length, arg.CallTime,"Before Call");
            lstOperationCalls.Items.Add(info);
        };

        host.Open();
    }

    private void btnClose_Click(object sender, EventArgs e)
    {
        host.Close();
    }
  }
}
```

Dikkat edileceği üzere servis davranışı yüklendikten sonra gerekli olay metodları da bağlanmıştır. Burada **AfterCall** veya **BeforeCall** olayları gerçekleştirildiğinde sembolik olarak**ListBox** kontrolüne bir takım bilgiler yazdırılmıştır. Bu bilgilerin **OptAfterCallEventArgs** ve **OptBeforeCallEventArgs** tiplerinden geldiği unutulmamalıdır

Geliştirici bu noktada tamamen özgürdür. Sonuçta olayla ilişkili bilgiler, en alttaki **OptDetective** tipinden, buradaki olay metodlarına kadar taşınmaktadır. Bu bilgiler istenilen şekilde kullanılabilir. Yani **ListBox** içeriğine yazdırılmaları şart değildir. Daha

önceden de belirttiğimiz gibi veritabanına, XML dosyasına, sistem Event Log' larına vb yerlere yazılabilirler.

Uygulamanın çalışma zamanı sonuçlarına baktığımızda aşağıdaki ekran görüntüsünde yer alan çıktı ile karşılaşırız. Yuppiii!!!



Görüldüğü gibi olay metodları başarılı bir şekilde devreye girmiş ve istemcinin **GetTotalSalaryByDepartment** ve **GetTotalGains** operasyonlarına yaptığı çağrılara ilişkin bazı bilgiler elde edilmiştir. Peki bu işleyiş sırasındaki çağrı hiyerarşisi nedir? Aslında olaylar nesneler üzerinden birbirlerine aktarılmaktadır. Bu aktarım işlemlerinin gerçekleşmesi içinse, **OptDetectiveServiceBehavior** isimli servis davranışı içerisinden **OptDetectiveOperationBehavior** isimli operasyon davranışı tipinin olaylarına ve**OptDetectiveOperationBehavior** içerisinden de **OptDetective** tipi içerisindeki olaylara abone olunmuştur. Bu noktada kodu debug ederek ilerleminizde yarar olacaktır. Bu şekilde geçişleri çok rahar takjp edebileceğinizi görebilirsiniz. Bu kutsal görevi de siz değerli okurlarıma bırakıyorum Biraz uzun bir anlatım oldu sanırım. Ama umuyorum ki işinize yarayacaktır. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

OperationCallTracingV2.rar (181,95 kb) **[örnek Visual Studio 2010 Ultimate sürümünde geliştirilmiş ve test edilmiştir]**

Regex ve Performans İpuçları - Interpreted ve Compiled Farkı, Bir de Sürpriz (2010-12-19T02:00:00)

c#,c# temelleri,

Merhaba Arkadaşlar,

Formula 1 merakı olanlar, yarışan araçların mühendislik olarak birbirlerine çok yakın teknolojiler ile üretildikleri ve benzer olduklarını bilirler. Gerçi bazı zamanlarda ön plana çıkan araçlar da söz konusudur. Frenaj veya hızlanma sistemlerine getirilen iyileştirmeler sonucu, diğer yarış araçlarının pilotları kim olursa olsun belirgin bir şekilde öne fırlarlar.

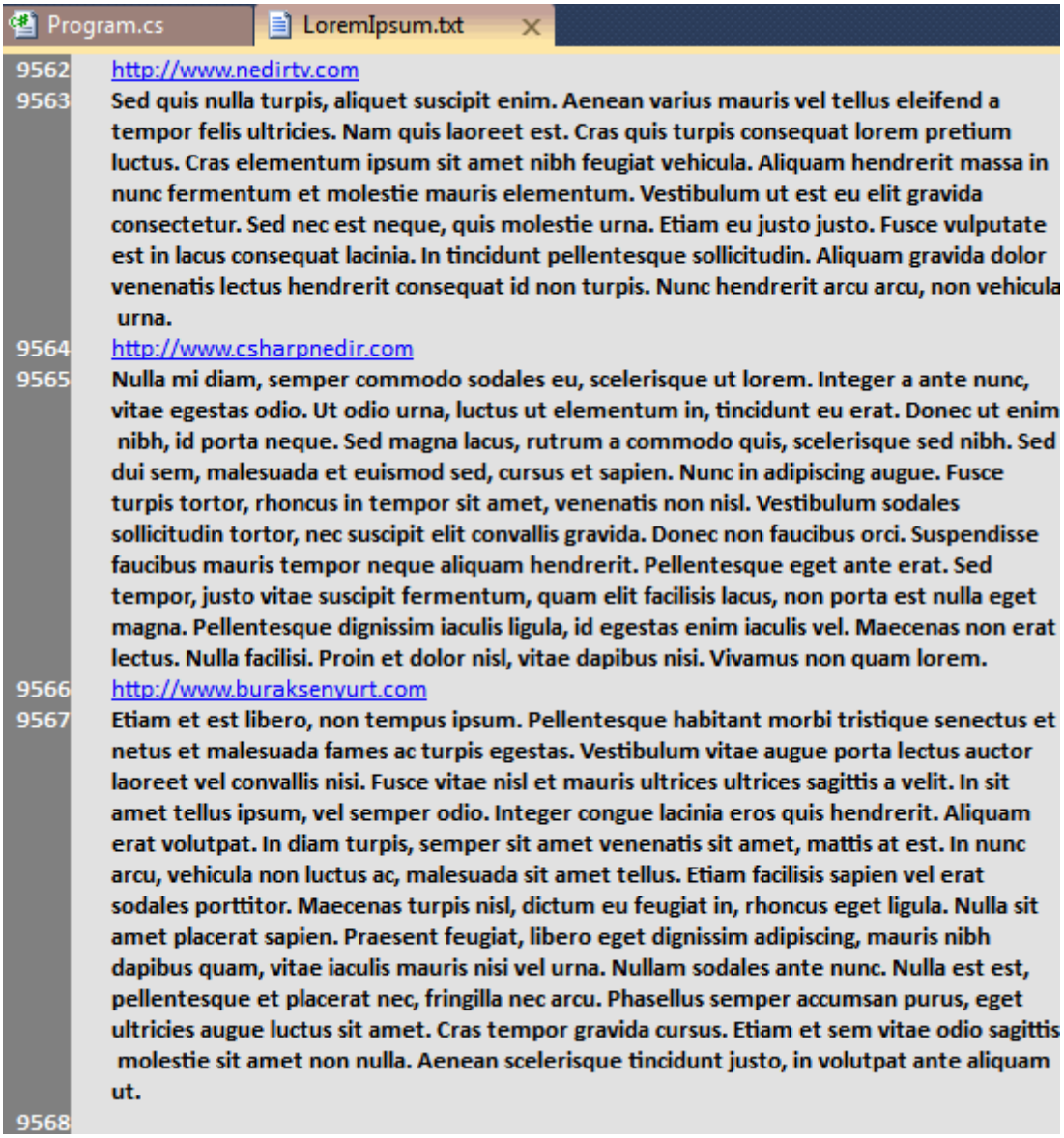
Ancak bazen de araçlar bir birlerine o kadar denktir ki, yarışın kaderini ve sonuçlarını sürücüler ile **Pit-Stop'** lar sırasında yapılan kritik değişiklikler belirler. örneğin lastik seçimlerİ, ön veya arka kanatların açısal değerleri, rüzgarın hızına göre yapılan ayarlamalar, yakıt tankının ne kadar doldurulacağı vb…

E tabi en iyi sonuçları elde edebilmek için takımlar yıl boyu sayısız test sürüşü gerçekleştirir ve sürekli olarak istatistikler tutarak raporlamalarda bulunur ve stratejik kararları veren yöneticilerin önlerini daha iyi görmelerini sağlamaya çalışırlar. çok doğal olarak yazılım süreçlerinde de benzer durumlar söz konusu değil midir? 🙂

İnce ayarlar çekilmiş bir yazılım, zaman zaman çok hızlı sonuçlar verebilir. Hatırlayacağınız üzere Regex ve Performans İpuçları – Otomatik Cache başlıklı bir önceki yazımızda son derece sıcak bir gecede, **Regex** tipinin performanslı kullanımına ilişkin ilk ip ucunu aktarmış ve sonuçlarını incelemeye çalışmıştık. **Regex** kullanımında dikkat çeken noktalardan bir diğeri de*(bir başka deyişle yapılabilecek ince ayarlardan bir diğeri de)* **yorumlanarak(Interpret)** veya önceden **derlenerek(Compiled)** çalıştırılabilen ifadeler ile ilişkilidir.

Bu notkada **Regex** tipine ait nesne örneklerinin oluşturulması sırasında devreye giren **RegexOptions** enum sabitinin **Compiled** değerinin kullanılması halinde ilgili **regex** deseninin derlenmiş halinin kullanılması söz konusudur. Normal şartlar altında **Compiled** değeri belirtilmediği takdirde, **Interpret** moda göre kontroller yapılmaktadır. Yani çalışma zamanı desen ile ilgili satıra geldiğinde bir takım işlemleri gerçekleştirir. Aslında konuyu teknik hatları ile irdelemek dışında örnek bir kod parçası üzerinde ilerleyerek çalışma zamanı sonuçlarına bakmamız da yarar vardır.

örnek senaryomuzda bu kez **150** paragraflık Lorem Ipsum içeriğinin defalarca arttırılmış ve aşağıdaki şekilden de görüleceği üzere aralara bir kaç **URL** adresi serpiştirilmiş bir versiyonu kullanılmaktadır. Söz konusu içerik **LoremIpsum.txt** isimli **Text** tabanlı bir dosyada toplanmış olup **9567** satırlık bir test içeriği üretilmiştir.



Gelelim test için ele alacağımız örnek kodlarımıza.

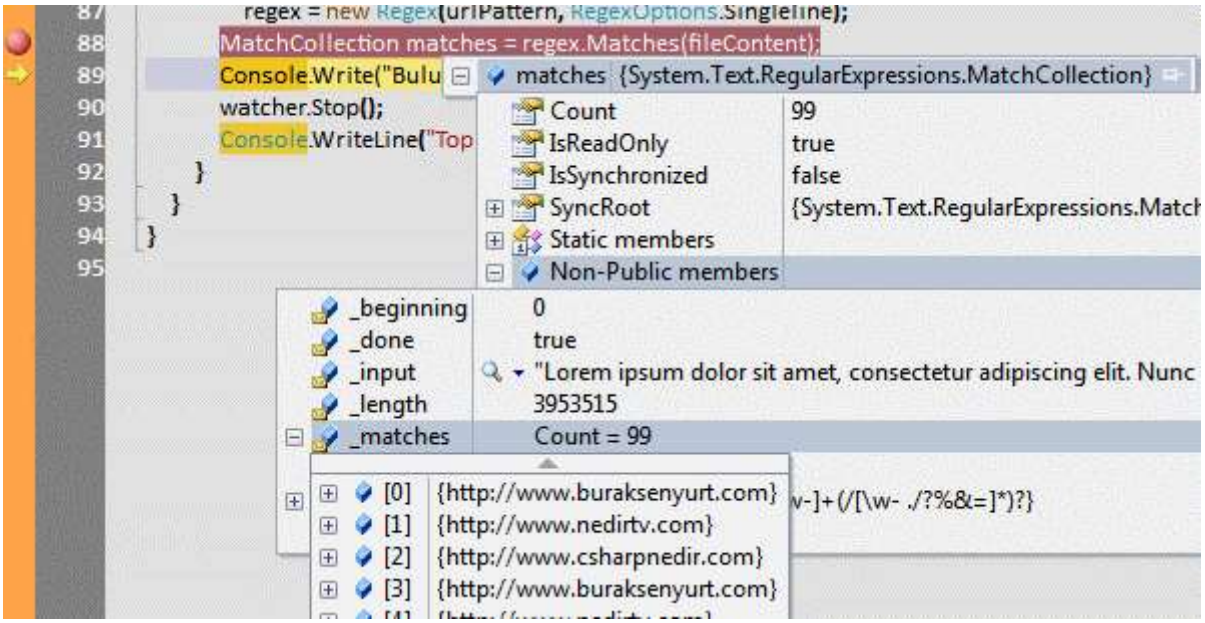
```
using System;
using System.Diagnostics;
using System.IO;
using System.Text.RegularExpressions;

namespace RegexTips2
{
   class Program
   {
      static void Main(string[] args)
      {
         string urlPattern = @"http(s)?://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?";
         string fileContent=File.ReadAllText(Path.Combine(Environment.CurrentDirectory,
"LoremIpsum.txt"));
```

```
        #region Matches Metodu Kullanımı(Yorumlamalı)

        Console.WriteLine("***Matches (Interpreted)***");
        for (int i = 0; i < 10; i++)
        {
           MatchesTest(urlPattern, fileContent,false);
        }

        #endregion

        #region Match ve NextMatch Kullanımı(Yorumlamalı)

        Console.WriteLine("***Match ve NextMatch (Interpreted)***");
        for (int i = 0; i < 10; i++)
        {
           NextMatchTest(urlPattern, fileContent,false);
        }

        #endregion

        #region Matches Metodu Kullanımı(Derlemeli)

        Console.WriteLine("***Matches (Compiled)***");
        for (int i = 0; i < 10; i++)
        {
           MatchesTest(urlPattern, fileContent, true);
        }

        #endregion

        #region Match ve NextMatch Kullanımı(Derlemeli)

        Console.WriteLine("***Match ve NextMatch (Compiled)***");
        for (int i = 0; i < 10; i++)
        {
           NextMatchTest(urlPattern, fileContent, true);
        }

        #endregion
    }

    private static void NextMatchTest(string urlPattern, string fileContent,bool
isCompiled)
    {
        Stopwatch watcher = new Stopwatch();
        watcher.Start();
```

```
        Regex regex=null;
        if (isCompiled)
           regex = new Regex(urlPattern, RegexOptions.Multiline |
RegexOptions.Compiled);
        else
           regex = new Regex(urlPattern, RegexOptions.Multiline);
        Match match = regex.Match(fileContent);
        int foundedMatch = 0;
       if (match.Success)
        {
           do
           {
              foundedMatch++;
              match = match.NextMatch();
           } while (match.Success);
        }
        watcher.Stop();
        Console.WriteLine("Bulunan eşleşme sayısı {0}.Toplam süre {1}", foundedMatch,
watcher.ElapsedMilliseconds);
     }

     private static void MatchesTest(string urlPattern, string fileContent,bool
isCompiled)
     {
        Stopwatch watcher = new Stopwatch();
        watcher.Start();
        Regex regex = null;
       if (isCompiled)
           regex = new Regex(urlPattern, RegexOptions.Singleline |
RegexOptions.Compiled);
        else
           regex = new Regex(urlPattern, RegexOptions.Singleline);
        MatchCollection matches = regex.Matches(fileContent);
        Console.Write("Bulunan eşleşme sayısı {0}.", matches.Count);
        watcher.Stop();
        Console.WriteLine("Toplam süre {0}", watcher.ElapsedMilliseconds);
     }
  }
}
```

Bu örnek **Console** uygulamasının **Main** metoduna ait kodlarda iki test metodu olduğu görülmektedir. **MatchesTest** isimli metod **Regex** nesne örneğinin **Matches** isimli fonksiyonunu değerlendirmektedir. **Matches** metodu ilk parametre olarak **regex** desenini almaktadır.

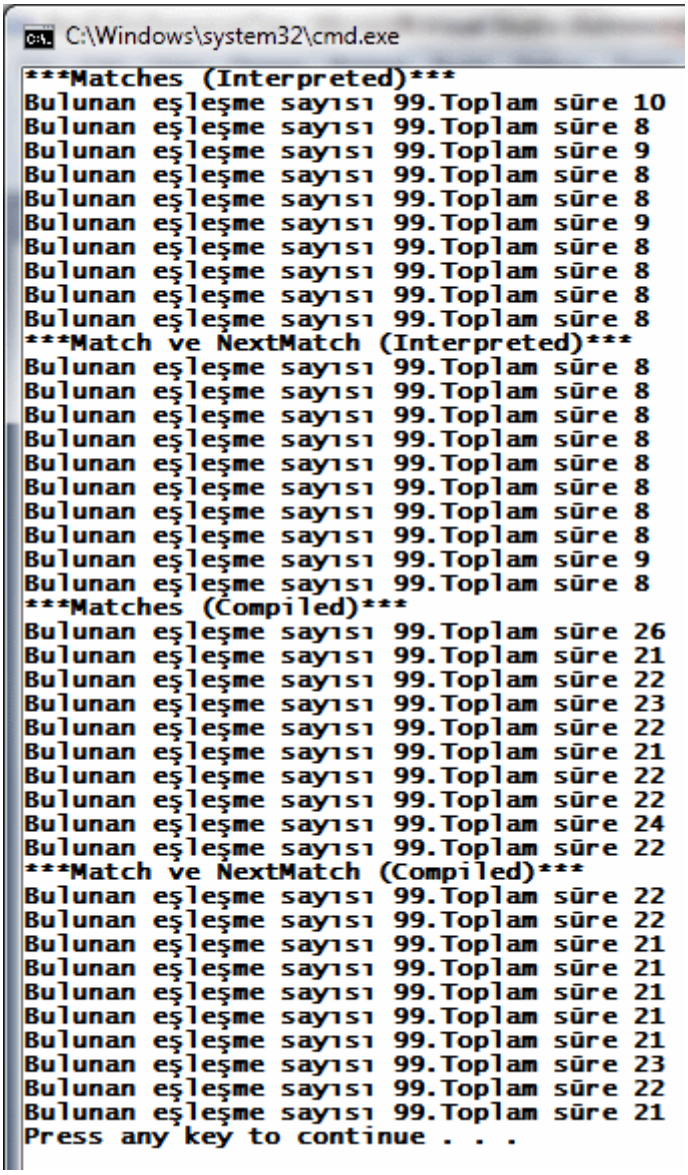
örneğimizde bir önceki yazımızda olduğu gibi bir **URL** deseni ele alınmaktadır. Bir başka deyişle **LoremIpsum.txt** içerisinde **URL** formatına uygun olan cümlelerin bulunması hedeflenmektedir. **Matches** metodu, **MatchCollection** tipinden bir koleksiyon döndürmektedir ve bu koleksiyon içerisinde, **URL** desenine uygun olan cümleler Match tipinden nesne örnekleri halinde yer almaktadır. Aşağıdaki **Debug** zamanı resminde URL desenin uygun olan cümlelere ait bir görüntü yer almaktadır.



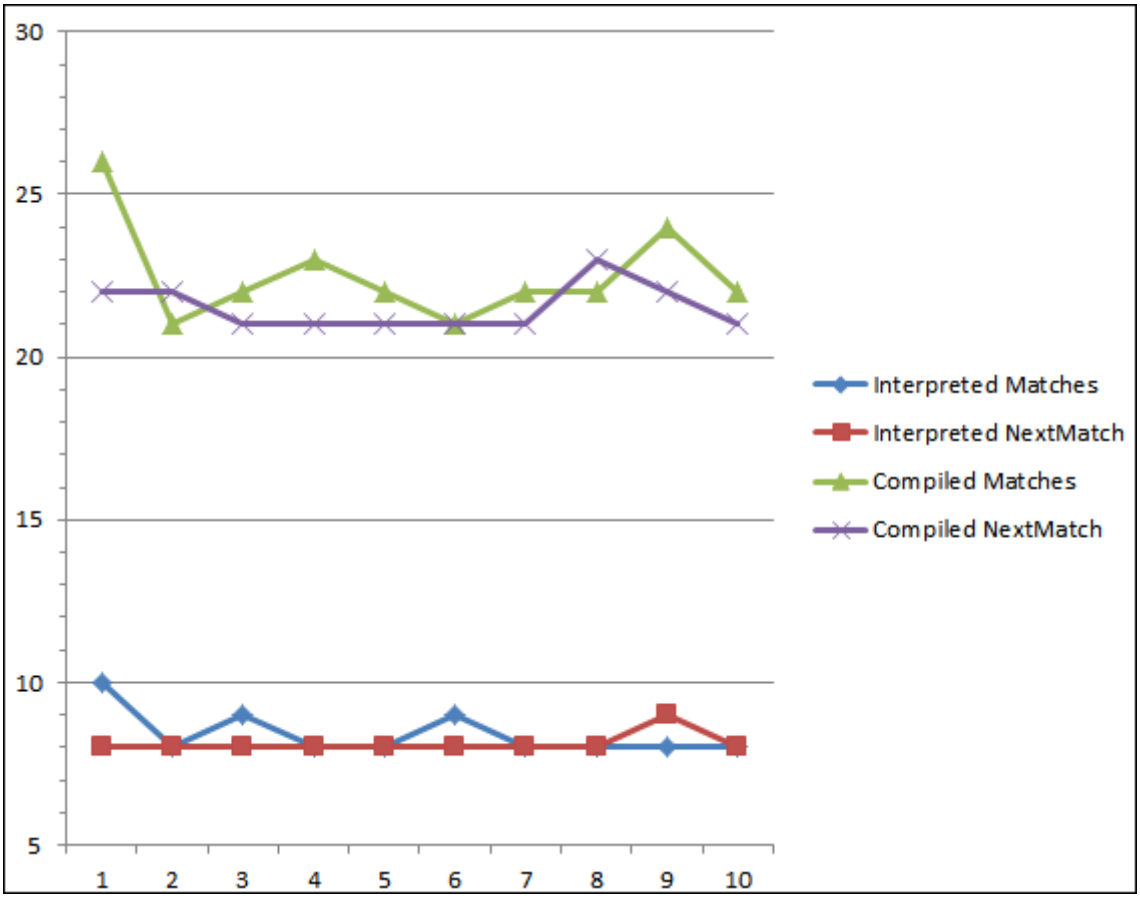
Tabi **Debug** görüntüsünden de anlaşılacağı üzere içeriğe ulaşmak için kod tarafında **foreach** gibi bir döngüden yararlanılması gerekmektedir.

NextMatchTest metodu ise daha farklı bir yaklaşım kullanmaktadır. Bu metodda öncelikli olarak **Match** metodu ile desene uygun bir cümle olup olmadığı kontrol edilmekte ve eğer varsa*(ki bu durumda Match nesne örneği Success değerini verecektir)* arkasından **do…while** döngüsü yardımıyla bir sonraki desen kontrolü operasyonu ile çalışmaya devam edilmektedir.

Her iki test metodu içerisinde **Regex** nesne örneği oluşturulurken **RegexOptions enum** sabitinin ilgili değerleri kullanılmakta ve metodların **Compiled** modda mı yoksa **Interperted** modda mı çalışacakları belirlenmektedir. Buna göre çalışma zamanı sonuçlarına baktığımızda aşağıdaki örnek çıktı ile karşılaştığımızı görürüz*(Tabi ki bu sonuçlar uygulamanın çalıştırıldığı sistemin çevresel özelliklerine göre farklılık gösterecek ancak kimin daha hızlı olduğu konusu pek fazla değişmeyecektir* 🙂 *)*

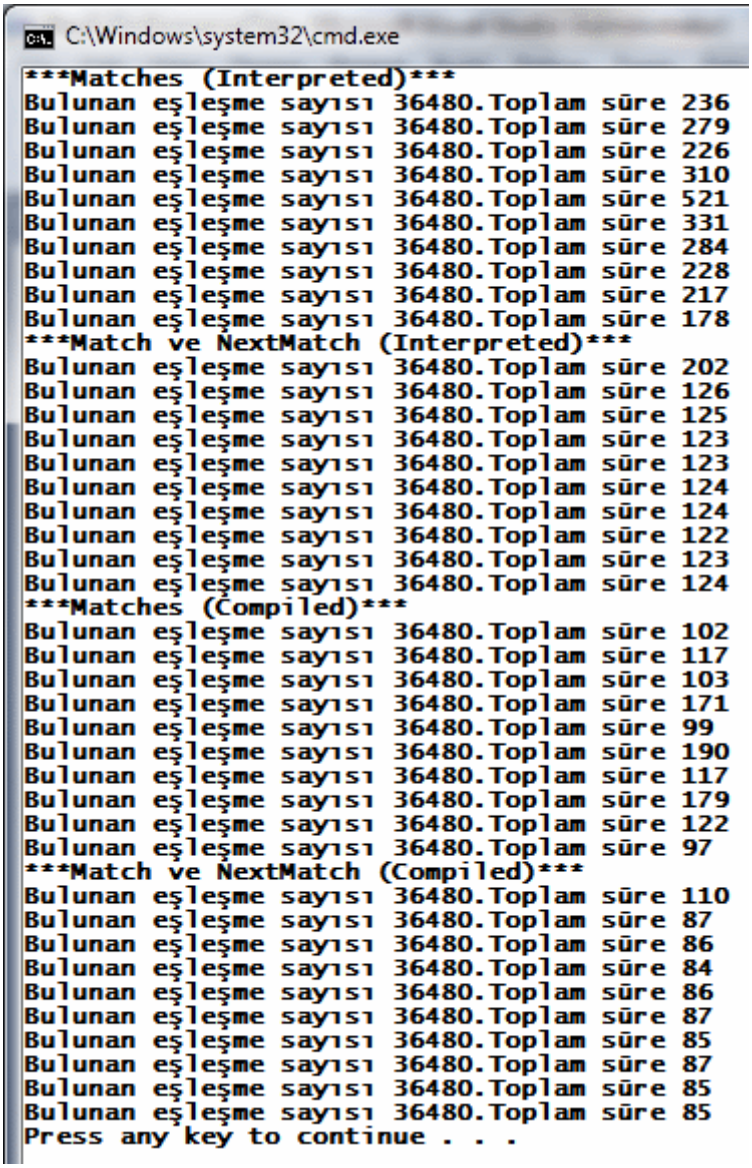


Interpeted ve **Compiled** moda göre **Matches** ve **NextMatch** metodlarının kullanımının arka arkaya **10** kere tekrar ediliği bu testin sonuçları aşağıdaki **Excel** grafiğinden daha net bir şekilde anlaşılabilir.

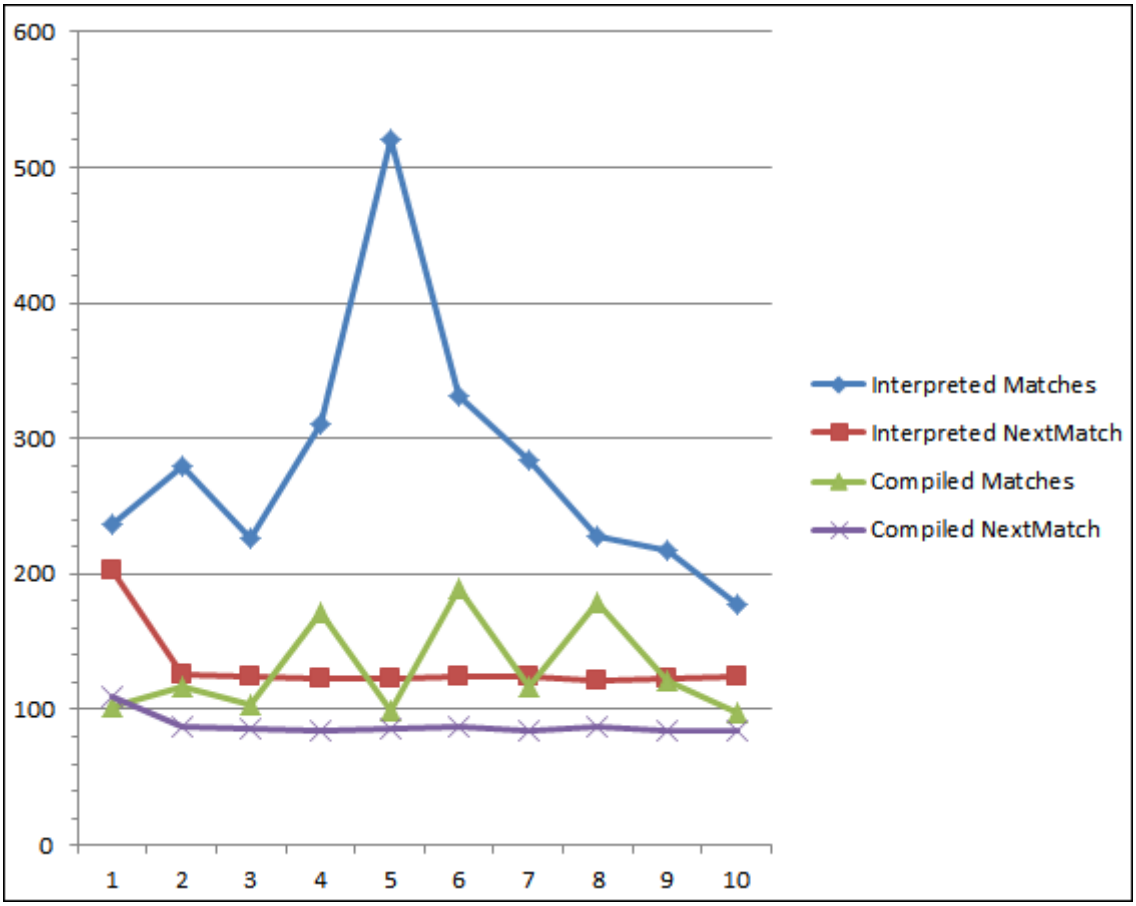


Burada en çok dikkat çeken nokta **Interpreted** mod ile, **Compiled** moda göre çok daha hızlı sürelerde sonuç alınabilmesidir. Bunun en büyük nedenlerinden birisi, **Compiled** modda , desenin ilk kullanıldığı sırada oluşan başlatma işlemleri için yapılan zaman kaybıdır. Ancak durum her zaman bu şekilde de gelişmeyebilir 😬

Ne demek istiyorum acaba? Gelin daha önceki yazımızda ele aldığımız **160bin** satırdan oluşan ve sadece doğru ve yanlış **URL** bilgileri içeren **text** dosyasını göz önüne alalım. Bu kez dosyanın satır sayısını **80bin** olarak tutacağız. İşte yakaladığım çalışma zamanı sonuçlarından bir tanesi.



Ve bu sonuçlara göre oluşan **Excel** grafiğinin yeni hali.



Dikkat edileceği üzere **Compiled** çalışma zamanı sonuçlarında yer yer **Interpreted** moda göre daha hızlı süreler elde edilebildiği görülmektedir. Hatta **NextMatch** metodunun kullanıldığı senaryo ile en hızlı erişim süreleri elde edilmiştir(İlk denemedeki hariç)

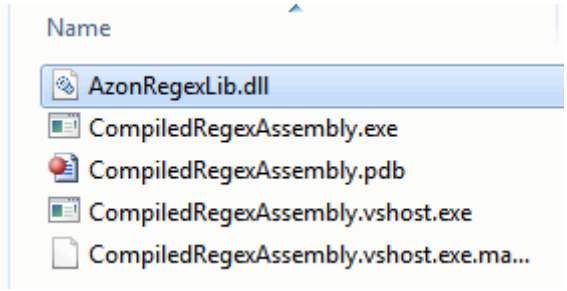
Şu da unutulmamalıdır ki burada **Instance** üzerinden çağırdığımız **Matches** veya **Match** gibi metodların, **Regex** tipi üzerinden çağrılabilen ve otomatik ön belleklemeyi kullanan **Static** versiyonları da mevcuttur. Bu versiyonların kullanımının daha hızlı olabileceğini düşünebiliriz. Ancak ben örneklerimdeki testler sırasında ve **Base Class Library**takımının konu ile ilgili araştırma yazılarında **Instance** üzerinden çağırılan **Interpreted** metodların, **static** olan versiyonlarına göre daha hızlı olabileceğini de gördüm. Bu konunun araştırılmasını da siz değerli okurlarıma bırakmak istiyorum

Yazının Ana Konusu Dışında Bir Mevzu

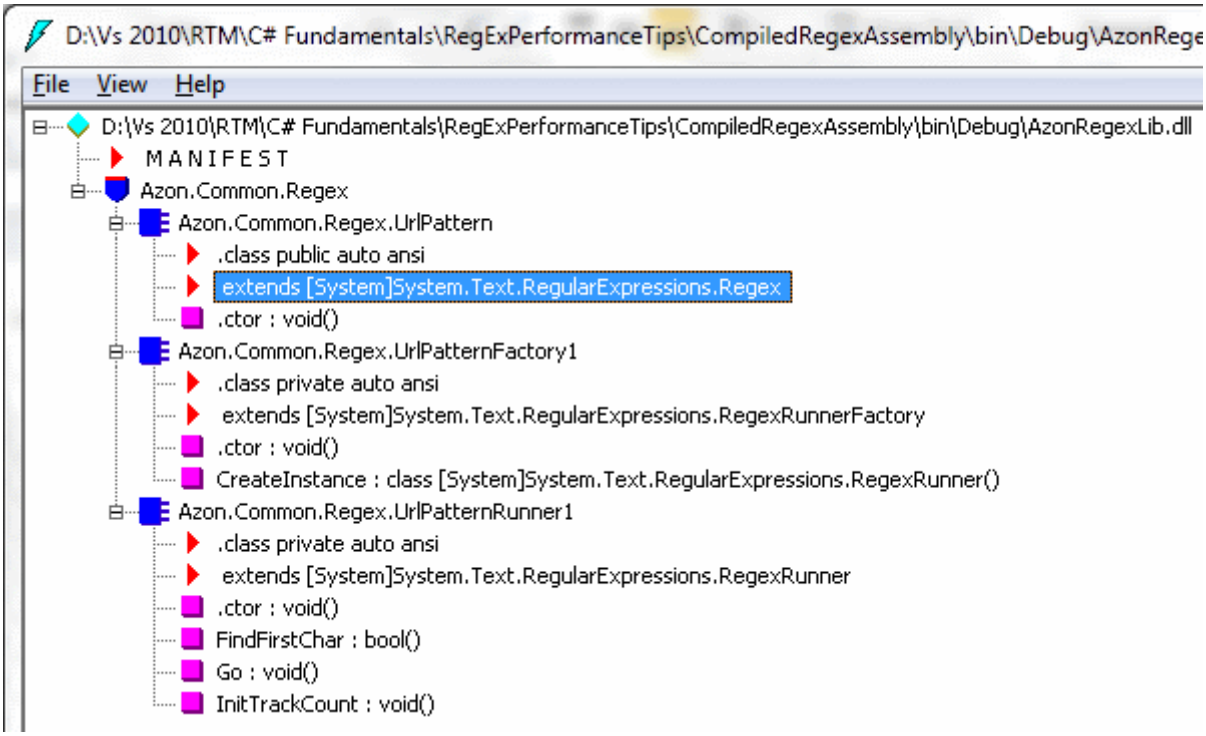
Hazır **Compiled** modda çalışabilen **Regex** metodlarına değinmişken bir de bunların ayrı bir **Assembly** içerisine nasıl derlenip kullanılabileceklerini incelemeye ne dersiniz? Burada amacımız derlenmiş **Regex** ifadelerini ayrı bir **assembly** içerisinde saklamak ve kullanmaktır. Bu amaçla örnek olarak aşağıdaki gibi bir kod parçasını göz önüne alabiliriz.

```
using System.Reflection;
using System.Text.RegularExpressions;

namespace CompiledRegexAssembly
{
   class Program
   {
      static void Main(string[] args)
      {
         // Derlenecek örnek Regex deseni
         string urlPattern = @"http(s)?://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?";
         // Derleme ile ilişkili ön bilgiler. Desen, RegexOptions enum sabiti değeri vb...
         RegexCompilationInfo compInfo =new RegexCompilationInfo(urlPattern,
RegexOptions.Multiline, "UrlPattern", "Azon.Common.Regex", true);
         RegexCompilationInfo[] regexes = { compInfo };
         // Kaydedilecek Assembly için gerekli bilgileri içeren AssemblyName nesne örneği
oluşturulur
         AssemblyName assemName = new AssemblyName("AzonRegexLib,
Version=1.0.0.1001, Culture=neutral, PublicKeyToken=null");
         // Regex bilgileri ilgili Assembly içerisine CompileToAssembly metodu ile
kaydedilir.
         Regex.CompileToAssembly(regexes, assemName);
      }
   }
}
```

Kodun çalıştırılması sonucu **AzonRegexLib.dll** isimli bir **Assembly'** ın uygulamaya ait **Exe** ile aynı yere çıkartıldığı görülecektir.



Eğer söz konusu **Assembly** içeriğine kendimize işkence yaparak **ILDASM(Intermediate Language Disassembler Tool)** aracı yardımıyla bakarsak, aşağıdaki ekran görüntüsünde yer alan içeriğe ulaşabiliriz.



UrlPattern isimli sınıf **Regex** tipinden türetilmiştir. **Yapıcı metoda(Constructor)** baktığımızda ise aşağıdaki **IL** içeriğinin üretildiğini görebiliriz.

```
.method public specialname rtspecialname
       instance void  .ctor() cil managed
{
  // Code size       51 (0x33)
  .maxstack  4
  IL_0000:  ldarg.0
  IL_0001:  call       instance void
[System]System.Text.RegularExpressions.Regex::.ctor()
  IL_0006:  ldarg.0
  IL_0007:  ldstr      "http(s)\?://([\\w-]+\\.)+[\\w-]+(/[\\w- ./\?%&=]*)\?"
  IL_000c:  stfld      string [System]System.Text.RegularExpressions.Regex::pattern
  IL_0011:  ldarg.0
  IL_0012:  ldc.i4.s   2
  IL_0014:  stfld      valuetype [System]System.Text.RegularExpressions.RegexOptions
[System]System.Text.RegularExpressions.Regex::roptions
  IL_0019:  ldarg.0
  IL_001a:  newobj     instance void Azon.Common.Regex.UrlPatternFactory1::.ctor()
  IL_001f:  stfld      class
[System]System.Text.RegularExpressions.RegexRunnerFactory
[System]System.Text.RegularExpressions.Regex::factory
  IL_0024:  ldarg.0
  IL_0025:  ldc.i4.s   4
  IL_0027:  stfld      int32 [System]System.Text.RegularExpressions.Regex::capsize
```

```
  IL_002c:  ldarg.0
  IL_002d:  call       instance void
[System]System.Text.RegularExpressions.Regex::InitializeReferences()
  IL_0032:  ret
} // end of method UrlPattern::.ctor
```

0001 numaralı **IL** koduna baktığımızda **Regex** tipinden bir nesne örneğinin oluşturulması için gerekli yapıcı metodun çağırıldığını görebiliriz. Sonrasında **0007**numaralı satırda **URL** deseni için **string** tipinden yerel bir değişkenin tanımlandığı gözlemlenir. İlerleyen kısımlarda dikkat çeken noktalardan birisi de **UlrPatternFactory1** isimli bir tip için nesne
örneklenmesidir. **UrlPatternFactory1** sınıfı **System.Text.RegularExpressions isim alanında(Namespace)** yer alan **RegexRunnerFactory** tipinden türeyen bir fabrikadır. özellikle**CreateInstance** metodunun içeriği dikkate değerdir.

```
.method public virtual instance class
[System]System.Text.RegularExpressions.RegexRunner
        CreateInstance() cil managed
{
  // Code size       6 (0x6)
  .maxstack  1
  IL_0000:  newobj     instance void Azon.Common.Regex.UrlPatternRunner1::.ctor()
  IL_0005:  ret
} // end of method UrlPatternFactory1::CreateInstance
```

Görüldüğü üzere **UrlPatternRunner1** tipine ait bir nesne örneğinin üretildiği görülmektedir. **UrlPatternRunner1** tipi ise **System.Text.RegularExpressions** isim alanında yer alan**RegexRunner** sınıfından türetilmiştir. **UrlPatternRunner1** tipinin içeriğinde yer alan operasyonlar tahmin edileceği üzere desen araştırma işlemlerinin yapılması için gerekli fonksiyonellikleri de içermektedir. Açıkçası bu tipin operasyonlarının içeriklerini tam olarak incelemeye çalıştığımızda çok fazla yere gittiğimiz için takibin zorlaştığını itiraf edebilirim 😕

Aslında bu noktada söz konusu **assembly'** ı örnek bir projede kullanmamızda yarar olacağı kanısındayım. Gelin örnek bir **Console** uygulamasından söz konusu **Assembly'** ı referans ederek **LoremIpsum.txt** dosyası içeriği üzerinden test edelim. İşte örnek uygulama kodlarımız.

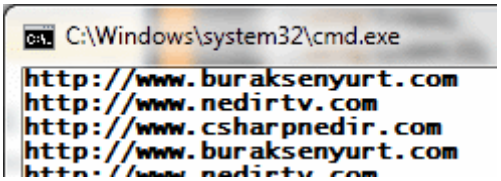
```
using System;
using System.IO;
using System.Text.RegularExpressions;
using Azon.Common.Regex;

namespace UsingCompiledRegexAssembly
{
```

```
   class Program
   {
      static void Main(string[] args)
      {
         string fileContent=File.ReadAllText(Path.Combine(Environment.CurrentDirectory,
"LoremIpsum.txt"));
         UrlPattern pattern = new UrlPattern();
         MatchCollection matches=pattern.Matches(fileContent);
         foreach (Match mtch in matches)
         {
            Console.WriteLine(mtch.Value);
         }
      }
   }
}
```

Diğer örnekleri düşündüğümüzde herhangibir yerde kontrol edilecek bilgiye ait **desen(Regex Pattern)** tanımının yapılmadığını görebiliriz. Sadece desene uygun olan verilerin yer aldığı dosya içeriği parametre olarak verilmiştir. Nitekim söz konusu desen tanımı ve bununla ilişkili başlangıç işlemleri zaten derlenmiş olan **assembly** içerisinde yer almaktadır. çok doğal olarak derlenmiş bir **assembly** üzerinden yapılan çağrılar, **instance** ile yapılan çağrılara nazaran daha performanslı olacaktır. Ama esneklik yönünden de derlenmiş assembly kullanımının bazı dez avantajları vardır. örneğin dinamik olarak bir desen bildirilemez. Desen zaten önceden belirlenmiş ve assembly içerisine gömülmüştür. Hatta kodun ilerleyen kısımlarında **Case Sensitive'** liğin dikkate alınmaması veya alınması gereken bir durumda **RegexCompilationInfo** tipi ile ilgili seçeneklerinin yeniden düzenlenmesi gerekmektedir ki bu mümkün değildir.

Aslında mümkün değildir derken biraz kolay kaçtığımızı ifade edebiliriz. Nitekim **Reflection** yardımıyla başlangıç ayarlarının parametrik olarak verilmesi, yeniden derlenmesi ve dinamik olarak yüklenerek kullanılması mümkündür. Hatta bu noktada dynamic keyword' ünün de bir çok noktada işi kolaylaştıracağını ifade edebiliriz. Yine de işin içerisinde dinamik olarak çalışma zamanına yük getirecek ve performansı olumsuz yönde etkileyecek bir çalışma mekanizması söz konusudur.

Yani **Assembly**' ın yeniden üretilmesi ve referans edilmesi gerekmektedir. Açıkçası tek bir desen değil ama n sayıda desenin kullanıldığı ve başlangıçtaki konfigurasyon seçeneklerinin belli olduğu senaryolarda kullanılması daha doğru olabilir. **foreach** döngüsünün çalışmasına göre **LoremIpsum** dosyasında geçerli **URL** formatında olan tüm cümleler elde edilebilecektir.



Tabi buraya kadar bahsettiklerimizi göz önüne aldığımızda **Regular Expression** kontrollerinde hangi tekniği kullanacağımız yönünde kafamızda bir sürü soru oluşmuş olabilir. Aslında aynı **Regular Expression** nesnesinin defalarca kullanıldığı senaryolarda **static** üyelerin kullanılması daha cazip görünmektedir. Diğer yandan desenlerin başlangıçtaki opsiyonel seçeneklerinin belli olduğu durumlarda ise, derlenmiş versiyonlarını kullanmak daha mantıklı olabilir. İşin gerçeği son sözü söylemek için test sonuçlarına bakmak bence en doğrusudur Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

RegExPerformanceTipsV2.rar (246,51 kb) **[örnek Visual Studio 2010 Ultimate Sürümü üzerinde Geliştirilmiş ve Test Edilmiştir]**

Bana Bir Struct Yaz. Yok Yok Bana Bir Class Yaz. (2010-12-19T01:30:00)

c#,c# temelleri,

Merhaba Arkadaşlar,

Aralık…2003 yılı. Dışarısı oldukça soğuk ve ben evdeyim. Camdan dışarıya baktığımda dışarıda pek kimseyi göremiyorum. Soğuktan dolayı sakin olan sokağımız daha da bir yanlız. Bu arada askerden döndükten sonra iş aramakla geçirdiğim 8nci ayın içerisindeydim. Neyseki mesleki kariyerimde ilerlemek için yeni bir heyecanım var. C#Nedir? ile daha ilk günlerimi yaşıyorum.

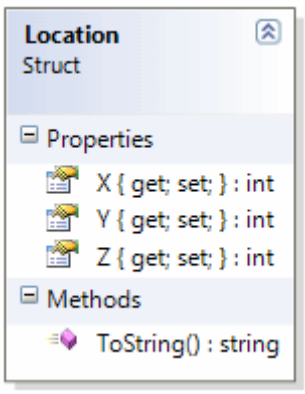
O zamanlar en çok yaptığım iş, öğrendiklerimi Türkçe yazım dili ile olabildiğince doğru bir şekilde paylaşmaktı. Nitekim öğrenmenin en iyi yolunun öğrenilenleri anlatmakla mümkün olduğuna inanmaktayım. Halen daha bu düşüncemin arkasındayım.

Elbette o zamanların verdiği acemilik nedeniyle, şimdi okuduğumda kaliteli olmadığını düşündüğüm yazılar üretmekteydim. örneğin 13 Aralık günü yayınladığım Struct(Yapı) Kavramı ve Class(Sınıf) ile Struct(Yapı) Arasındaki Farklar başlıklı yazım gibi

Geçtiğimiz günlerde düşündüm de bu tip konuları arada sırada baştan ele almak ve yeniden örnekleyerek anlatmakta yarar olabilirdi. Hiç olmassa şu anda C# dilinin temellerini öğrenmekte olan arkadaşlarımız için. öyleyse gelin, hiç vakit kaybetmeden yola koyulalım.

Bu yazımızda, **Struct(Yapı)** ile **Class(Sınıf)** tipleri arasındaki temel farklılıkları irdelemeye çalışıyor olacağız. Ancak benzerlikleri de yakalamaya gayret edeceğiz. *(Hemen şunu hatırlatalım; **.Net Framework,** 5 temel veri tipi tanımlar. Bunlar **Class, Struct, Enum, Interface** ve **Delegate** tipleridir)* özellikle **Struct** tipinin kullanımına ilişkin örnekler geliştireceğiz.

Aslında her iki tip arasındaki farklılıklar, uygulama geliştirirken hangi tipi tercih edeceğimiz açısından oldukça önemlidir. **C#** programlama dilini yeni öğrenen birisi için ilk akla gelen, bellek üzerindeki tutuluş biçimlerinin farklı olmasıdır.
Hatta **Struct'** ların **Değer Türü(Value Type), Class'** ların ise **referans türü(Reference Type)** olduklarının, temel seviyede bilgi sahibi olan tüm programcılar farkındadır. Ancak fazlası da olabilir Bu temel farklılıklardan bazılarını örnek kodlar yardımıyla irdelemeye çalışmaya ne dersiniz? Başlamadan önce örnek bir **Struct** tipini göz önüne alalım.




```
using System;

namespace StructvsClass
{
   struct Location
   {
      public int X { get; set; }
      public int Y { get; set; }
      public int Z { get; set; }

      public override string ToString()
      {
         return String.Format("({0},{1},{2})", X, Y, Z);
      }
   }
}
```

Eminim ki **Location** isimli **Struct** tipimiz pek çok deneyimli programcıya **.Net Framework** içerisindeki **Point** yapısını hatırlatmaktadır Temel olarak bir nesnenin uzaydaki yerini belirtmek için kullanabileceğimiz bu tip ayrıca **Object** tipinden kalıtsal olarak gelen **ToString** metodunu da **ezmektedir(Override)**. Şimdi dilerseniz **Struct** tipi ile ilişkili vakalarımızı analiz etmeye ve **Class**' lar ile aradaki farklılıkları görmeye çalışalım.

Vaka 1 : Metod Parametresi Olarak Kullanılmaları Hali

Struct' lar ile **Class**' lar arasındaki farklılıkları anlamak adına akla gelen ilk örnek, bu tiplerin metod parametresi olarak kullanılmaları halidir. Aslında burada farkı oluşturan durum **C#** tarafında metod parametrelerinin her zaman için **değer türü olarak(Pass By Value)** bilgi taşımalarıdır. Değer türü olarak bilgi taşınması, metoda aktarılan parametrik değişkenlerin aslında metod içinde kopylanarak iş yapması anlamına gelmektedir. Tabi burada dikkat edilmesi gereken nokta ise şudur; Sınıflar birer referans türüdür ve aslında metodlara değer türü olarak geçirilen bu referanslara ait adres bilgileridir. Sanırım spagetti cümlelere geçiş yaptık. Bu nedenle aşağıdaki örnek kod parçasını göz önüne alarak ilerleyelim.

```
using System;

namespace StructvsClass
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Case 1 : Struct' lar Metod parametresi olduklarında

         Location planeLocation = new Location { X = 12, Y = 33, Z = 41 };
         Console.WriteLine(planeLocation.ToString());
         IncreaseLocation(planeLocation);
         Console.WriteLine(planeLocation.ToString());

         #endregion
      }

      static void IncreaseLocation(Location loc)
      {
         loc.X += 10;
         loc.Y += 10;
         loc.Z += 10;
      }
   }
}
```

IncreaseLocation isimli metod **Location** türünden bir parametre almaktada ve **X,Y,Z** özelliklerinin değerlerini onar birim arttırmaktadır. Programın giriş noktası olan **Main** metodu içerisinde ise**planeLocation** isimli bir **Location** nesne örneği üretilip bu metoda parametre olarak geçirilmektedir. örneğin çalışma zamanı çıktısına baktığımızda aşağıdaki sonuç ile karşılaşırız.

İşte bir **Struct'** ın metod parametresi olarak kullanılmasındaki tipik davranış şekli. Değer türü olarak tüm içeriği ile birlikte metod içinde kopyalanan bir nesne söz konusu olduğundan, **IncreaseLocation** çağrısından önce ve sonrasındaki **planeLocation** içeriği değişmemiştir. Bu içerik sadece **IncreaseLocation** içerisinde değişime uğramıştır. Kısacası, metoda aktarılan değişkenin orjinal veri yapısında bir bozulma söz konusu değildir. Durumu biraz daha dramatize etmek ve özellikle sınıf ile yapı arasındaki farkı ortaya çıkartmak adına, **Location** tipini **class** haline getirip örneği tekrardan çalıştırabiliriz. Bu durumda sonuçlar aşağıdaki gibi olacaktır.

Görüldüğü üzere **planeLocation** değişkeninin **X,Y** ve **Z** değerlerinin orjinal halleri, metod çağrısından sonra bozulmuştur. Bunun tipik nedeni referans türü olan sınıfların, metod parametrelerine değer türü olarak geçirilmeleri sırasında, adreslerinin taşınmasıdır. Dolayısıyla metod içerisine kullanılan **loc** isimli değişken ile, **planeLocation** isimli değişken bellek üzerindeki aynı adres alanlarını işaret etmektedir.

Elbette **Struct** tipinin metodlara referans türü olarka geçirilmesi de mümkün olabilir. Bu durumda **ref** veya **out** parametrelerinden yararlanılması yeterli olacaktır. Aşağıda bu duruma ilişkin örnek bir kullanım söz konusudur.

```
using System;

namespace StructvsClass
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Case 1 : Struct' lar Metod parametresi olduklarında

         Location planeLocation = new Location { X = 12, Y = 33, Z = 41 };
         Console.WriteLine(planeLocation.ToString());
```

```
            DecreaseLocation(ref planeLocation);
            Console.WriteLine(planeLocation.ToString());

            #endregion
        }
        static void DecreaseLocation(ref Location loc)
        {
            loc.X -= 10;
            loc.Y -= 15;
            loc.Z -= 12;
        }
    }
}
```

Bu durumda **planeLocation** değişkeni, **Decrease** isimli metoda referans olarak geçirilecektir. Dolayısıyla metod içerisinde **loc** değişkeninin veri içeriğinde yapılan değişiklikler,**planeLocation** değişkeninin içeriğini de bozacaktır. Aşağıdaki çalışma zamanı görüntüsünde bu durum açık bir şekilde görülmektedir.

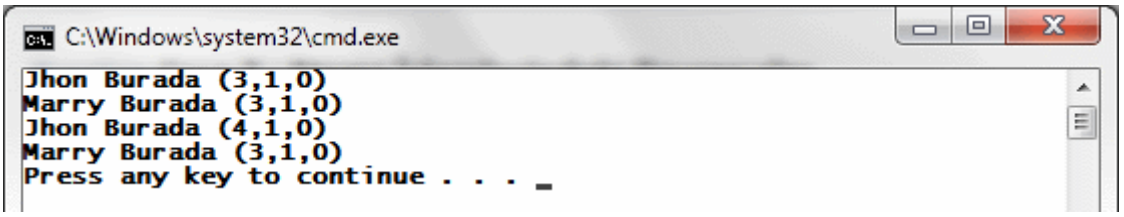
Vaka 2 : Atama İşlemlerindeki Davranışlar

Sınavda yapılar ile sınıflar arasındaki temel farklılıkları sorsalar sanıyorum ki herkesin aklına gelecek ilk seçenek atama işlemlerindeki davranışsal farklılıktır. Durumu anlamak için hemen aşağıdaki kod parçasını göz önüne alalım.

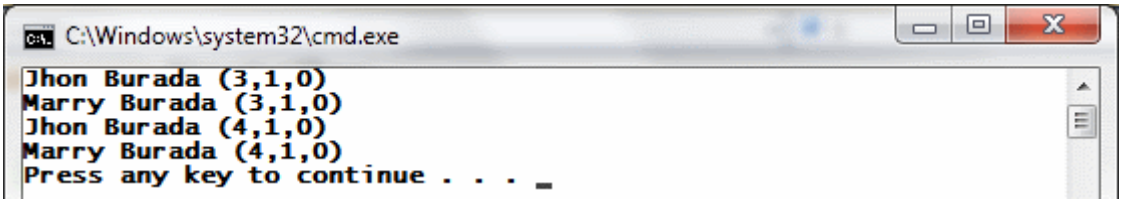
```
Location jhonLocation = new Location { X=3,Y=1 };
Location marryLocation = jhonLocation;

Console.WriteLine("Jhon Burada {0}",jhonLocation.ToString());
Console.WriteLine("Marry Burada {0}", marryLocation.ToString());

jhonLocation.X++;

Console.WriteLine("Jhon Burada {0}", jhonLocation.ToString());
Console.WriteLine("Marry Burada {0}", marryLocation.ToString());
```

Kodun ilk kritik noktası **jhonLocation** değişkeninin **marryLocation** değişkenine atandığı yerdir. Bu atama sonrasında **marryLocation** değişkenin veri içeriği ile **jhonLocation'** ın veri içeriği eş olacaktır. Yani tipik bir değişken kopyalama işlemi söz konusudur. Diğer yandan **jhonLocation'** ın **X** alanının değerinin arttırılması

sonrasında **marryLocation'** ın **X**değerinde bir değişim olmadığı görülecektir. Nitekim her ikisi de bellek üzerinde ayrı lokasyonlarda duran değişkenlerdir. örneğin çalışma zamanı çıktısı aşağıdaki gibidir.



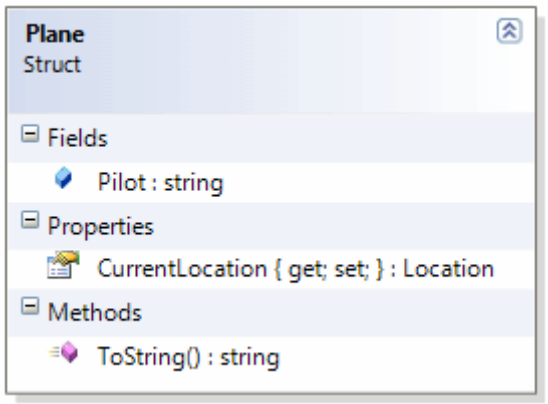
Peki ya **Location** bir sınıf olsaydı? Bu durumda çalışma zamanı çıktısı aşağıdaki gibi olacaktı.



Görüldüğü üzere **X** alanının değerinin arttımı, **marryLocation** değişkeninin **X** değeri için de geçerli olmuştur. Bu son derece doğaldır nitekim atama sonrası kopyalanan içerik değil referans adresleridir. Bu sebepten atama sonrası **jhonLocation** değişkeninin veri içeriğinde yapılan değişiklikler çok doğal olarak **marryLocation** içinde geçerli olacaktır. Bu durumda geliştiricinin karar vermesi gereken soru şudur: **Marry** ile **Jhon** birlikte hareket etmeli midir? Yoksa istedikleri noktada birbirlerinde ayrı olarak hareket edebilirler mi?

Vaka 3 : Struct Tipinden özellik(Property) Kullanılması Hali

Bu vakayı canlandırmak için ikinci bir **Struct** tipine daha ihtiyacımız olacak. **Plane** isimli yapımızı aşağıdaki gibi tasarladığımızı düşünelim.



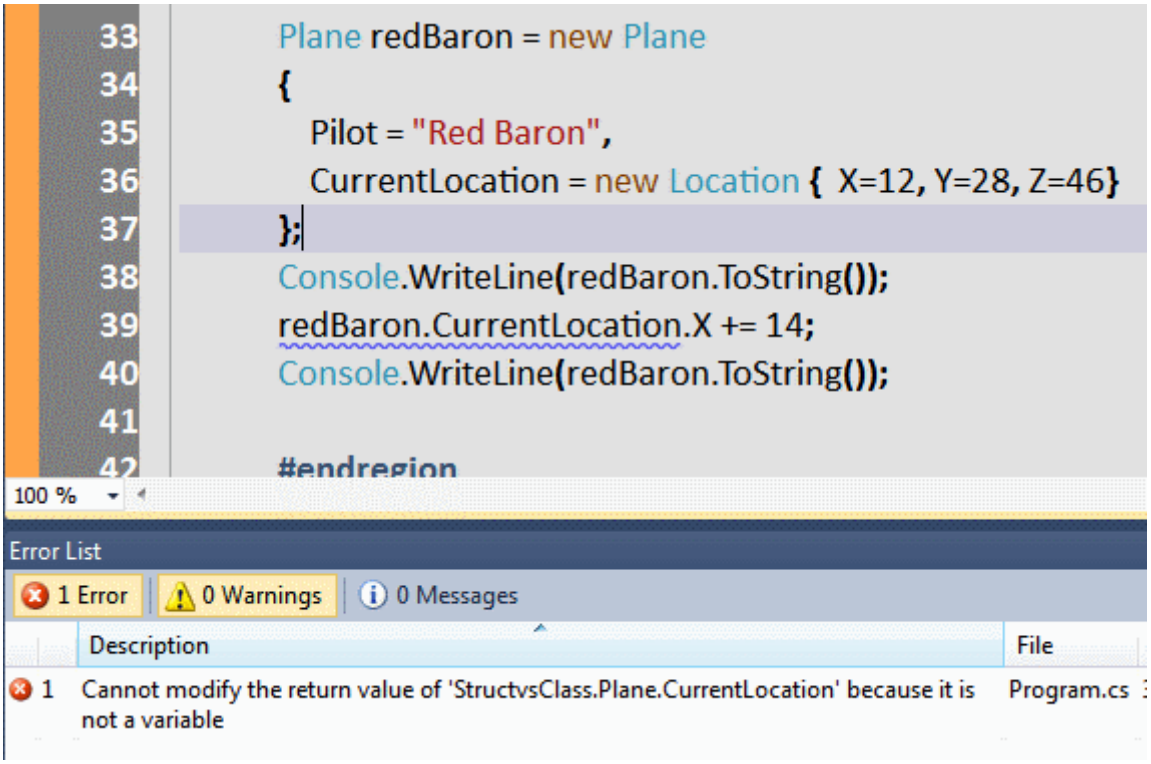

```
struct Plane
{
```

```
   public string Pilot;
   public Location CurrentLocation { get; set; }

   public override string ToString()
   {
      return String.Format("{0} şu anda {1} lokasyonundadır",
         String.IsNullOrEmpty(Pilot)?"Yok":Pilot,
         CurrentLocation.ToString());
   }
}
```

Plane isimli yapı içerisinde **Location** tipinden bir **özellik(Property)** bulunmaktadır. Şimdi **Main** metodu içerisinde aşağıdaki örnek kod parçasını geliştirdiğimizi düşünelim.

```
Plane redBaron = new Plane
{
   Pilot = "Red Baron",
   CurrentLocation = new Location {  X=12, Y=28, Z=46}
};
Console.WriteLine(redBaron.ToString());
redBaron.CurrentLocation.X += 14;
Console.WriteLine(redBaron.ToString());
```

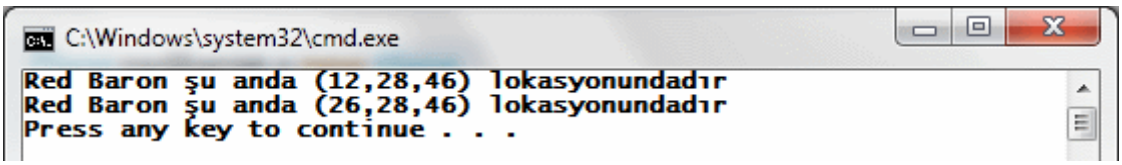
İlk olarak **Plane** tipinden bir nesne örneği üretilmektedir ve **CurrentLocation** isimli özellik **initialize** edilirken **X,Y** ve **Z** özelliklerine ilk değerleri verilmektedir. Kodun ilerleyen kısımlarında**CurrentLocation** özelliği üzerinden **X** alanına gidilerek değerinin **14** birim arttırıldığı görülmektedir. Herhangiri sorun olabilir mi? Evet olabilir…Nitekim kod derlendiğinde, aşağıdaki hata mesajının üretildiği görülecektir.



Dikkat edileceği üzere **CurrentLocation** özelliğinin üzerinden **X** değeri değiştirilememektedir. Aslında değişim söz konusudur fakat özellik kullanılması nedeniyle yeniden bir **initialize**işlemi yapılmasını gerektirmektedir. Kod aşağıdaki hale getirildiğinde bir sorun kalmayacaktır.

```
Plane redBaron = new Plane
{
   Pilot = "Red Baron",
   CurrentLocation = new Location {  X=12, Y=28, Z=46}
};
Console.WriteLine(redBaron.ToString());
// redBaron.CurrentLocation.X += 14;
redBaron.CurrentLocation = new Location
{
   X = redBaron.CurrentLocation.X + 14
   , Y = redBaron.CurrentLocation.Y
   , Z = redBaron.CurrentLocation.Z
};
Console.WriteLine(redBaron.ToString());
```

Bu durumda çalışma zamanı çıktısı aşağıdaki gibi olacaktır.



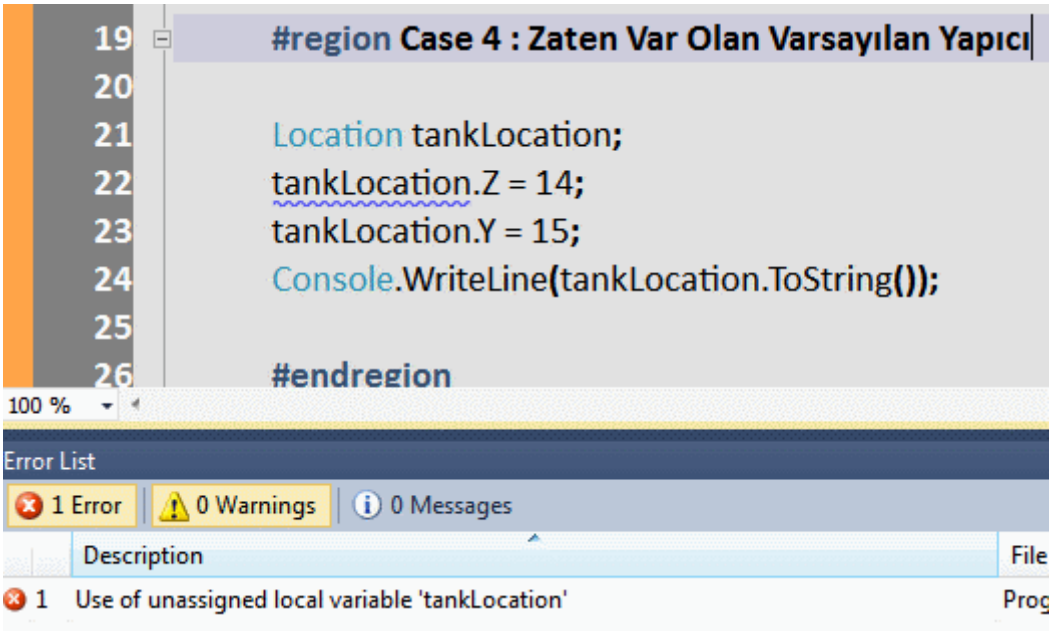
Elbette **Location** yapısı **class** olarak tanımlanmış olsaydı bu tip bir hata ile karşılaşılmayacaktı.

Vaka 4 : Yapıları örneklemeden Kullanabilme Hali

Struct tipleri aslında değer türü olduklarından, veri içerikleri başlangıçta mutlaka ilk değerlerine atanmalıdır. Bu durum **Stack** bellek bölgesinde tutulmalarından kaynaklanmaktadır. Bu sebepten normal şartlarda **Struct** tipleri için **varsayılan yapıcı metodun(Default Constructor)** yazılmasına izin verilmemektedir. Aslında değer türü olduklarından bu tipleri başlatırken varsayılan yapıcı metod kullanılması da zorunlu değildir. Bu nedenle sistem otomatik olarak varsayılan yapıcı metodu atamaktadır. Diğer yandan yapılar tipik bir değer türü değişkeni olaraktan da kullanılabilirler. Fakat burada tuhaf bir durum söz konusudur. **Location** isimli **Struct** tipimizi bu anlamda göz önüne alalım ve aşağıdaki kodu yazdığımızı düşünelim.

```
Location tankLocation;
tankLocation.Z = 14;
tankLocation.Y = 15;
Console.WriteLine(tankLocation.ToString());
```

Aslında bu son derece mantıklı bir kod parçasıdır. Nitekim **Struct** bir değer türü olduğundan **int, double, bool** gibi tanımlanıp kullanılabilmelidir. Yani varsayılan yapıcı metod veya başka bir versiyon ile örneklenmeden kolayca kullanılabilmelidir. Ancak kodumuzu bu haliyle derlediğimizde aşağıdaki derleme zamanı hatasını aldığımızı görürüz.

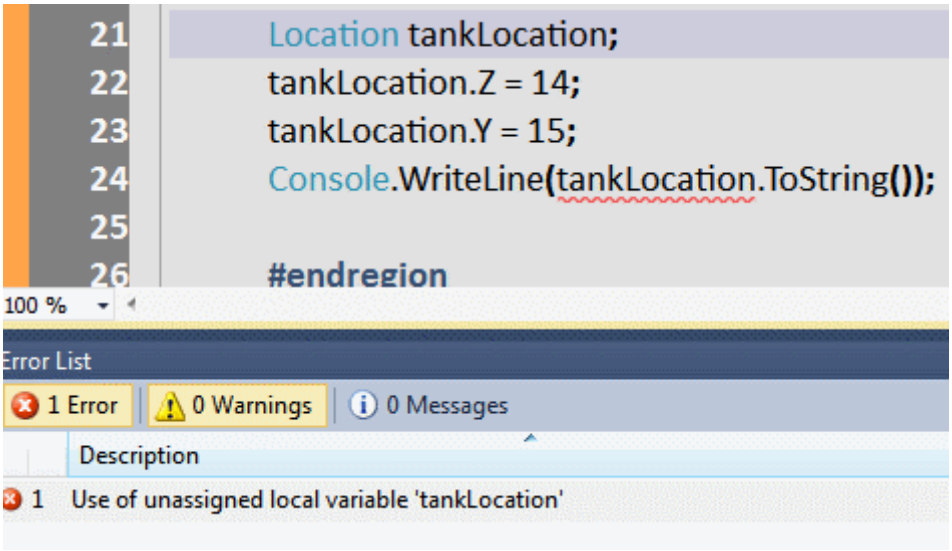


Hımmm! İlginç Neden böyle bir hata aldık ki? Aslında sınıfların yapamadığı bir şeyi yapıyor olmalıydık. Acaba **Struct** tipleri kıl mı?

Sorun bu tip bir kullanımın sadece **alanlar(Fields)** için geçerli olmasıdır.
üstelik **public** olan alanlar için geçerlidir. Hatta ne kadar **public** alan var ise her birinin ilk değerinin de atanması gerekmektedir. Dolayısıyla **Location** tipini aşağıdaki hale getirerek devam edelim. (Elveda özellikler ☹)

```
namespace StructvsClass
{
   struct Location
   {
      public int X;
      public int Y;
      public int Z;
...
```

Bu sefer de aşağıdaki hata mesajı ile karşılaşırız.



Aslında bir önceki hata mesajı ile aynı görünmesine rağmen arada bir fark vardır. İlk hata mesajının verildiği yer ile ikincisi farklıdır. Son hata mesajının sebebi tüm alanlara ilk değer atanmayışıdır. Dolayısıyla kodun en azından aşağıdaki gibi olmasında yarar vardır.

```
Location tankLocation;
tankLocation.Z = 14;
tankLocation.Y = 15;
tankLocation.X = 12;
Console.WriteLine(tankLocation.ToString());
```

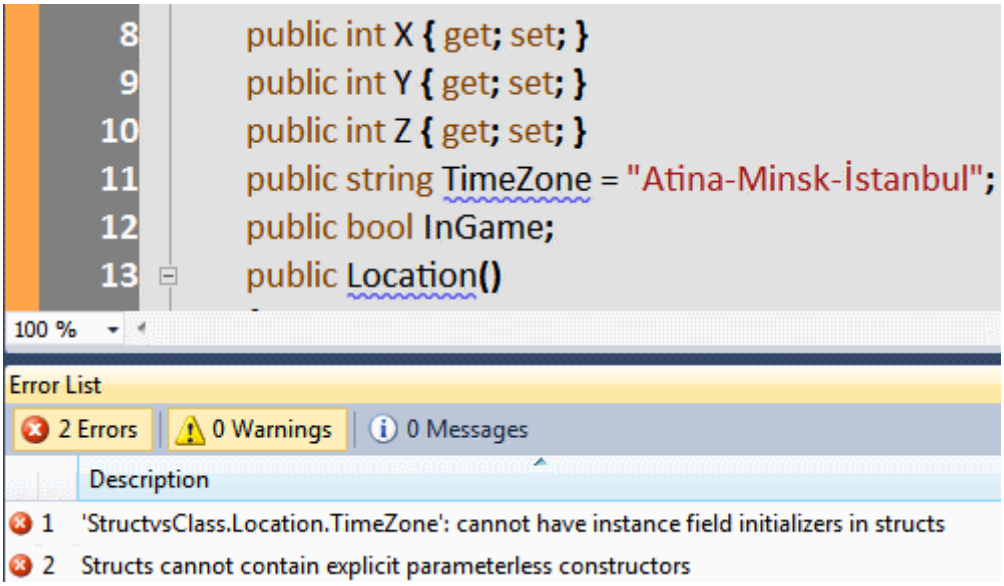
Dikkat edileceği üzere **X,Y** ve **Z** alanlarının tamamına ilk değer atanmıştır. Elbette yapıcı metod yardımıyla bir nesne örneklediğimizde bu tip bir atama zorunluluğu bulunmamaktadır.

Vaka 5 : üyelerin Varsayılan Değerleri(Default Values)

Struct' lar aslında **Class** tipleri kadar esnek bir yapı sunmazlar. Bunu içerdikleri **alanlara(Fields)** varsayılan başlanıç değerlerini atarken çokça görebiliriz. Dilerseniz **Location** yapısını aşağıdaki gibi tasarladığımızı planlayalım.

```
using System;

namespace StructvsClass
{
   struct Location
   {
      public int X { get; set; }
      public int Y { get; set; }
      public int Z { get; set; }

      public string TimeZone = "Atina-Minsk-İstanbul";
      public bool InGame;

      public Location()
      {
         InGame = true;
      }

      public override string ToString()
      {
         return String.Format("({0},{1},{2})", X, Y, Z);
      }
   }
}
```

Location yapısına **TimeZone** isimli **string** tipinden ve **InGame** isimli bool tipinden birer alan eklenmiştir. Geliştiricinin amacı, bu iki değişkenin başlangıçta varsayılan değerlere sahip olmasıdır. Ancak kod bu şekliyle derlendiğinde aşağıdaki derleme zamanı hatalarının alındığı görülür.



Dikkat edileceği üzere **TimeZone** değişkenine varsayılan değer atanamamaktadır. Ayrıca varsayılan yapıcı metodun var olamayacağı ifade edilmektedir. Buna göre **InGame**değişkenin ilk değer atanması için varsayılan yapıcı metodun kullanılması planları da suya düşmüştür. Halbu ki **Location** tipi bir sınıf olarak tasarlanmış olsaydı! 😀 Bu durumda bir derleme zamanı hatası alınmayacaktı 😉

Vaka 6 : Performans

Aslında **Stack** bellek bölgesini kullanmalarından dolayı **Struct** ile çalışmanın performansı arttırıcı bir etken olduğu düşünülebilir. Ne var ki yapıların birbirlerine atanması, metodlara parametre olarak geçirilmesi veya döndürülmesi her zaman için **değer yoluyla(By Value)** olmaktadır. Bir başka deyişle **referans yolu(By Reference)** ile olmamaktadır. Bu da zaman içerisinde bellek üzerinde pek çok kopya nesnenin oluşması anlamına gelmektedir. Dolayısıyla sizde pek çok geliştirici gibi büyük bir olaslıkla sınıfları kullanmayı tercih edebilirsiniz.

Karar Verirken

Peki ya nasıl karar vereceğiz? **Struct** mı kullanalım, **Class** mı kullanalım? Hangisini tercih etmeliyiz? Sanıyorum ki cevaplanması en zor sorulardan birisi de bu. Hatta az önce bu yazıyı yazarken yanıma gelen çalışma arkadaşım şunu deyince **"Valla Hocam bunca zamandır yazılım geliştiriyorum, hiç bir projemde Struct kullanıldığını görmedim"** 😬 Aslında yaygın kanı halen daha devam etmekte. **16 byte'** tan daha küçük veri toplulukları için **Struct** kullanılması önerilmektedir. Ayrıca sadece veri anlamında bir tipten söz ediyorsak, **Struct**kullanımı **Class** tipine nazaran daha anlamlı olabilir.

Sınıfları nesne yönelimli özellikleri tamamıyla destekleyen bir tip olarak düşünmek çok daha doğru bir yaklaşımdır. Nitekim **Struct** tiplerinin türetilmesi mümkün değildir ve bu**OOP(Object Oriented Programming)** çervecesinde önemli bir ilke ihlalidir.

Aslında karar vermek için **Struct** ve **Class** arasındaki farklılıklar dışında, benzerlikleri de bilmenin yararı vardır. Bu sebepten aşağıdaki tablodan da yararlanabiliriz.

Mesele	Class	Struct
Türleri nedir?	Referance Type(Değer Türü)	Value Type(Referans Türü)
Dahili Tipler İçerebilirler mi?(Nested Types)	Evet	Evet
Property, Method, Field, Event, Indexer, Constant üyeler içerebilirler mi?	Evet	Evet
Türetilebilirler mi?	Evet	Hayır
Generic olabilirler mi?	Evet	Evet
Partial Type olabilirler mi?	Evet	Evet
Bilinçli olarak varsayılan yapıcı(Default Constructor) tanımlanabilir mi?	Evet	Hayır
Sealed olabilirler mi?	Evet	Zaten hep öyleler
Kullanılabilmeleri için ille de new operatörü ile örneklenmeleri mi gerekir?	Evet	Hayır
Interface' leri uygulayabilirler mi?	Evet	Evet
Yapıcı metodları aşırı yüklenebilir mi?	Evet	Evet (Ama varsayılan yapıcı asla yerinden kıpırdamaz)
Static üyeler içerebilirler mi	Evet	Evet

Aradan geçen 7 yıldan sonra bir önceki yazıya bakıyorum da 😃 Aslında bazı kuralların hiç değişmediği ap açık ortada. **.Net Framework**' ün içerisinde pek çok noktada kullanılan **Struct** veri tipi, halen daha projelerde göz önüne alınabilir, alınmalıdır. özellikle bu ihtiyacı farkediyor olmakta bir yazılımcı için ve hatta proje için son derece önemlidir. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

StructvsClass.rar (24,56 kb) **[örnek Visual Studio 2010 Ultimate Sürümü üzerinde Geliştirilmiş ve Test Edilmiştir]**

Servis Operasyonlarını Kod Yardımıyla İzlemek (2010-12-19T01:20:00)

wcf,wcf 4.0,windows communication foundation,iparameterinspector,service operation behavior,

Merhaba Arkadaşlar,

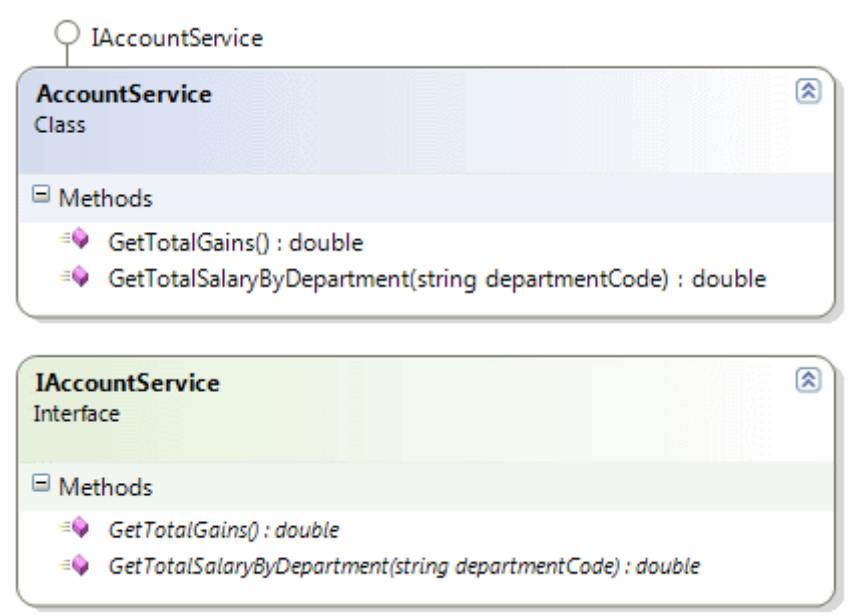
Savaş meydanlarında ezelden beri uygulanan istihbarat ve haber alma teknikleri, avantaj sağlamak açısından en önemli unsurların başında gelmektedir. Karşı tarafın nerede olduğunu izlemek, ne yaptığını bilmek, istenen bir anda ne kadar ateş gücüne sahip olduğunu tespit etmek çok önemlidir. örneğin karşı tarafın şu anda denizde hareket etmekte olan veya karasal alanda saklanmakta olan kuvvetlerini görmek istediğimiz durumlarda keşif yapılması ve istihbarat toplanması stratejik anlamda önemli avantajlar sağlayacaktır.

Yakın zamana kadar bu işler için yüksek irtifalardan, ses hızının bir kaç kat üstünde uçan uçaklar görev alırdı ki halen daha pek çok hava kuvvetlerince kullanılmaktadır. örneğin bir zamanlar Birleşik Devletler ile Sovyetler Birliği arasında krize neden olan U2' ler, şekli ve harcadığı yakıt ile ün salıp emekli olan SR71 Blackbird' ler vs…

Ancak artık uzaktan kumanda edilebilen, düştüklerinde pilot kaybı yaşanmasını engelleyen, insansız olmaları nedeni ile çok yüksek irtifalara kadar çıkabilen **İnsansız Hava Araçları(Unmanned Aeiral Vehicle)** var. Aslında uzun süredir varlar*(üstelik ben Generals oyununda da sıklıkla bunları kullanıyorum* Hatta geçtiğimiz günlerde bizimde gurur kaynağımız olan **ANKA** isimli insanız hava aracımız, ilk kez hangardan çıkarak uçuşunu gerçekleştirdi. Peki bu İHA' ların bu günkü yazımız ile bir bağlantısı var mı? Pek olduğunu söyleyemeyiz.

Aslında bu günkü yazımızda bir **WCF(Windows Communication Foundation)** servisinin operasyonlarına gelen çağrılar hakkında istihbarat toplamaya çalışıyor olacağız. Ancak bunun için standart **Trace** ve **Monitoring** özellikleri yerine kod yardımıyla ilerleyeceğiz. Normal şartlarda **IIS(Internet Information Services)** üzerinden yayınlanan bir **WCF** servisine gelen operasyon çağrılarını izlemek son derece kolaydır. Bu amaçla konfigurasyon dosyasında gerekli ayarların yapılması yeterlidir. Diğer taraftan istenirse **Windows Server AppFabric** ile **IIS** üzerine gelen eklentilerden yararlanarak bu tip izleme ayarları kolaylıkla yapılabilir.

Lakin bizim amacımız geliştirme safhasında **Self-Hosted** tekniğine göre yayınlanan servis operasyonlarına gelen çağrıları yakalamak ve basit bilgiler almaktır. Geliştireceğimiz bu senaryonun bize sağlayacağı önemli artılar da vardır. İlk etapta kod yardımıyla özelleştirilmiş bir servis davranışının çalışma zamanındaki **ServiceHost** örneğine nasıl entegre edilebileceği öğrenilecektir. Diğer yandan operasyonlara olan taleplerin ve dönüşlerin çok basit mana da izlenmesi sağlanacak ve karmaşık **Trace** çıktıları arasında kaybolunmayacaktır. Dilerseniz hiç vakit kaybetmeden işlemlerimize başlayalım. Başlangıçta aşağıdaki **sınıf diagramında(Class Diagram)** görülen yapıya sahip bir servis kütüphanemiz olduğunu düşünelim.



Buna göre **IAccountService** isimli **Servis Sözleşmemizin(Service Contract)** içeriği aşağıdaki gibidir.

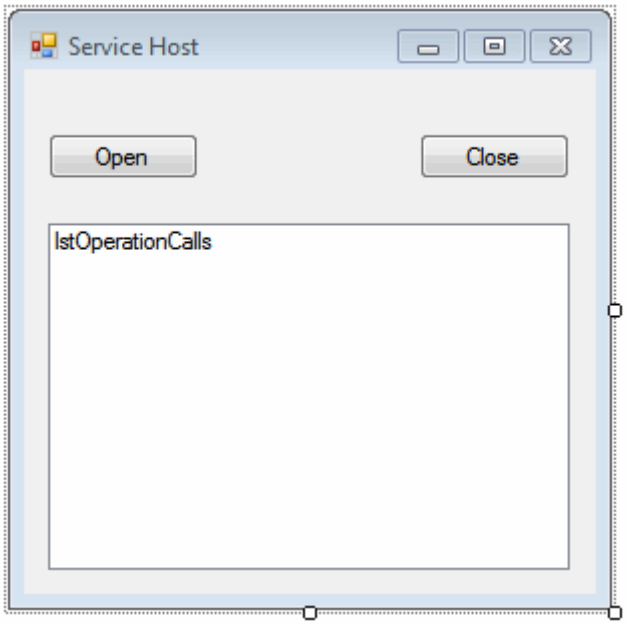
```
using System.ServiceModel;

namespace AdventureWorksFinance
{
   [ServiceContract(Name="AdventureWorks.FinanceServices",
Namespace="http://AdventureWorks/Finance/Services")]
   public interface IAccountService
   {
      [OperationContract]
      double GetTotalSalaryByDepartment(string departmentCode);

      [OperationContract]
      double GetTotalGains();
   }
}
```

Diğer taraftan söz konusu servis sözleşmesini uygulayan **AccountService** sınıfının içeriği de aşağıdaki gibidir.

```
namespace AdventureWorksFinance
{
   public class AccountService
      : IAccountService
   {
      #region IAccountService Members
```

```
        public double GetTotalSalaryByDepartment(string departmentCode)
        {
            return 1245000;
        }

        public double GetTotalGains()
        {
            return 2500000;
        }

        #endregion
    }
}
```

Bu servis içeriğine sahip olan **kütüphanemiz(WCF Service Library),** bir **WinForms** uygulamasına referans edilmiş durumdadır. **WinForms** uygulaması **Host** program olarak düşünülebilir. Yani **ServiceHost** tipi yardımıyla **AccountService** hizmetini dış dünyaya sunmak üzere tasarlanmaktadır. Bu amaçla arayüzümüzü aşağıdaki gibi tasarladığımızı düşünebiliriz.



Buna göre kullanıcı **Open** başlıklı düğmeye basarak servisi yayına sunacak ve **Close** düğmesi ile de kapatacaktır. WinForms uygulaması aşağıdaki **app.config** içeriğine sahiptir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
```

```
        <serviceBehaviors>
          <behavior name="TcpServiceBehavior">
            <serviceMetadata/>
            <serviceDebug includeExceptionDetailInFaults="false" />
          </behavior>
        </serviceBehaviors>
      </behaviors>
      <services>
        <service name="AdventureWorksFinance.AccountService"
behaviorConfiguration="TcpServiceBehavior">
          <endpoint address="" binding="netTcpBinding" bindingConfiguration=""
            contract="AdventureWorksFinance.IAccountService">
          </endpoint>
          <endpoint address="mex" binding="mexTcpBinding"
bindingConfiguration=""
            contract="IMetadataExchange" />
          <host>
            <baseAddresses>
              <add
baseAddress="net.tcp://localhost:4505/AdventureWorksFinance/AccountService/" />
            </baseAddresses>
          </host>
        </service>
      </services>
  </system.serviceModel>
</configuration>
```

Servisimiz **TCP** protokolüne göre yayın yapmaktadır. Ayrıca **MexTcpBinding** bağlayıcı tipini kullanarak **Metadata** yayını da yapmaktadır. Dolayısıyla istemciler kendileri için gerekli**Proxy** tiplerini üretmek amacıyla **WSDL** içeriğine ulaşabilirler.

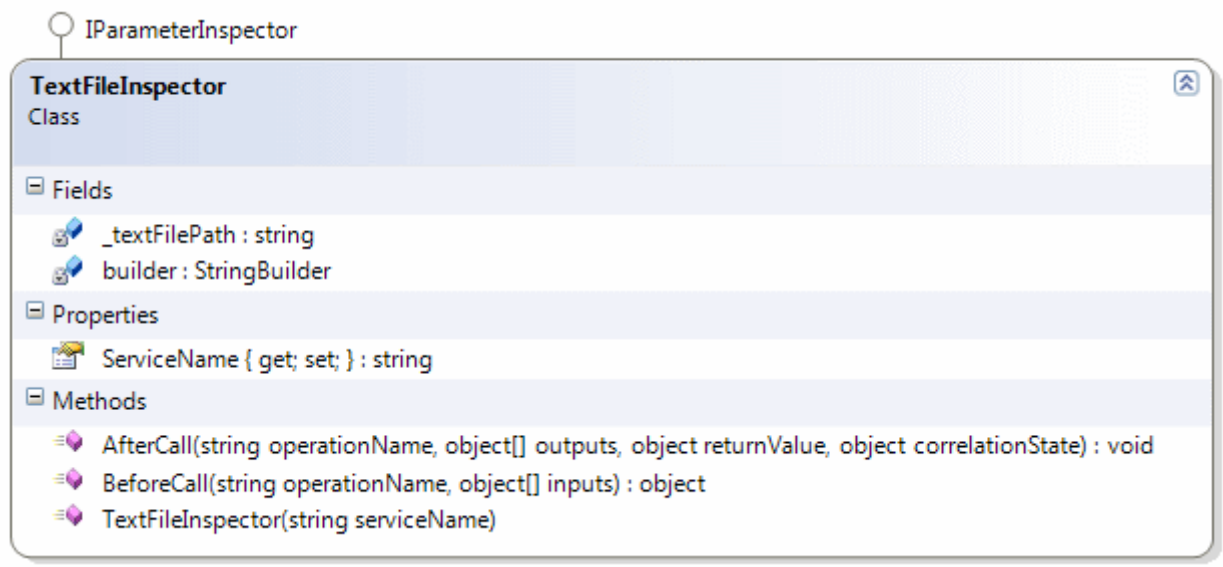
Gelelim WinForms' un başlangıçta arka plandaki kodlarına*(Tam anlamıyla Buton Arkası Kodlarına* 😬*)*

```
using System;
using System.ServiceModel;
using System.Windows.Forms;
using AdventureWorksFinance;

namespace HostApp
{
  public partial class Form1
    : Form
  {
    ServiceHost host=null;
```

```
        public Form1()
        {
            InitializeComponent();
            btnOpen.Enabled = true;
            btnClose.Enabled = false;
        }

        private void btnOpen_Click(object sender, EventArgs e)
        {
            host=new ServiceHost(typeof(AccountService));
            host.Opened += (s, ea) =>
            {
                lstOperationCalls.Items.Add(String.Format("Servis açık
{0}",DateTime.Now.ToLongTimeString()));
                btnOpen.Enabled = false;
                btnClose.Enabled = true;
            };
            host.Closed += (s, ea) =>
            {
                lstOperationCalls.Items.Add(String.Format("Servis kapatıldı {0}",
DateTime.Now.ToLongTimeString()));
                btnOpen.Enabled = true;
                btnClose.Enabled = false;
            };
            host.Open();
        }

        private void btnClose_Click(object sender, EventArgs e)
        {
            host.Close();
        }
    }
}
```

çok basit olarak özetlemek gerekirse, **ServiceHost** sınıfına ait nesne örneği üretilirken **AccountService** tipini parametre olarak almaktadır. Yani AccountService servisi yayına alınmaktadır. Bu servis üzerinden **Open** ve **Close** işlemlerinin uygulanması halinde **Opened** ve **Closed** olay metodları devreye girmektedir.

Kahveleriniz yanınızda değil mi ? Artık asıl işlemlerimize başlayabiliriz. İlk olarak içeriği aşağıda görülen ve **IParameterInspector arayüzünü(Interface)** uygulayan bir sınıf geliştirerek yola çıkalım.



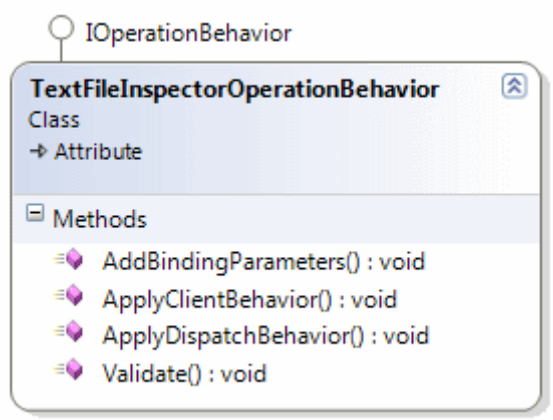

```
using System;
using System.IO;
using System.ServiceModel.Dispatcher;
using System.Text;

namespace InspectorLib
{
   public class TextFileInspector
      :IParameterInspector
   {
      public string ServiceName { get; set; }
      private string _textFilePath = Path.Combine(Environment.CurrentDirectory,
"OperationCallLog.txt");
      private StringBuilder builder = new StringBuilder();

      public TextFileInspector(string serviceName)
      {
         ServiceName = serviceName;
      }
      #region IParameterInspector Members

      public void AfterCall(string operationName, object[] outputs, object
returnValue, object correlationState)
      {
         builder.AppendLine(String.Format("After Call -> Service {0}, Time {1}, Operation
Name {2}", ServiceName,DateTime.Now.ToLongTimeString(), operationName));
         builder.AppendLine("Return Value");
         builder.AppendLine(returnValue.ToString());
```

```
            StreamWriter writer = File.AppendText(_textFilePath);
            writer.WriteLine(builder.ToString());
            writer.Close();
        }

        public object BeforeCall(string operationName, object[] inputs)
        {
            builder.AppendLine(String.Format("Before Call -> Service {0}, Time {1},
Operation Name
{2}",ServiceName,DateTime.Now.ToLongTimeString(),operationName));
            builder.AppendLine("Inputs");
            foreach (object input in inputs)
            {
                builder.AppendLine(inputs[0].ToString());
            }

            StreamWriter writer = File.AppendText(_textFilePath);
            writer.WriteLine(builder.ToString());
            writer.Close();

            return builder.ToString();
        }

        #endregion
    }
}
```

TextFileInspector sınıfı, **IParameterInspector arayüzünü(Interface)** uygulamaktadır. Bu arayüz görüldüğü üzere **AfterCall** ve **BeforeCall** isimli iki metodun ezilmesini istemektedir.**BeforeCall** metodu ile operasyona yapılan çağrıya ait bilgiler yakalanabilir. Aslında operasyon başlatılmadan önce, yakalandığı yer olarakta düşünebiliriz. **AfterCall** metodu hangi operasyona çağrı yapılığına dair **operationName** parametresini kullanmaktadır. Diğer yandan **object** dizisi tipinden olan **inputs** parametresi ile, çağrıda bulunulan operasyona gelen giriş değerleri öğrenilebilir. Benzer şekilde **AfterCall** fonksiyonunu da, operasyondan istemciye cevap dönerken değerlendirilebilir. Böylece istemciye hangi operasyon için hangi değerin döndürüldüğü yakalanabilir. Her iki metod da, **Text** tabanlı bir dosyaya bilgi yazmakta ve raporlamayı bu şekilde gerçekleştirmektedir. Ama bu bir zorunluluk değildir. Burada **ListBox**içeriğine bilgilendirme amaçlı öğe eklenmesi de sağlanabilir, bir veritabanı tablosuna yazma işlemi de gerçekleştirilebilir.

çok doğal olarak bu sınıfın çalışma zamanına bir şekilde öğretilmesi gerekmektedir. Bu noktada servis metodları ve servise söz konusu tipin bildirilmesi gerektiğini düşünebiliriz. Servis operasyonları için bu tip bir davranış uygulanacağı **nitelikler(Attributes)** yardımıyla kolayca öğretilebilir Dolayısıyla ilk

olarak servis operasyonları için **TextFileInspector**tipinin bir davranış olarak bildirilmesi gerekmektedir. Bu amaçla aşağıdaki sınıfı tasarlamamız yeterli olacaktır.

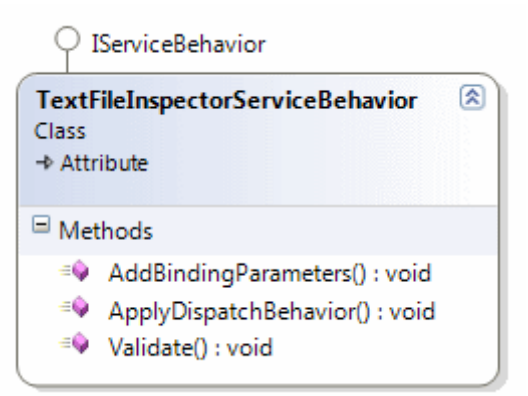



```
using System;
using System.ServiceModel.Description;

namespace InspectorLib
{
   [AttributeUsage(AttributeTargets.Method)]
   public class TextFileInspectorOperationBehavior
      :Attribute,IOperationBehavior
   {
      #region IOperationBehavior Members

      public void AddBindingParameters(OperationDescription operationDescription,
System.ServiceModel.Channels.BindingParameterCollection bindingParameters)
      {
      }

      public void ApplyClientBehavior(OperationDescription operationDescription,
System.ServiceModel.Dispatcher.ClientOperation clientOperation)
      {
      }

      public void ApplyDispatchBehavior(OperationDescription
operationDescription, System.ServiceModel.Dispatcher.DispatchOperation
dispatchOperation)
      {
         if(dispatchOperation.Parent.Type!=null)
            dispatchOperation.ParameterInspectors.Add(new
TextFileInspector(dispatchOperation.Parent.Type.Name));
      }
```

```
        public void Validate(OperationDescription operationDescription)
        {
        }

        #endregion
    }
}
```

TextFileInspectorOperationBehavior sınıfı, metodlara uygulanabilen bir operasyonel davranış tipi olarak tasarlanmıştır. Bu nedenle **IOperationBehavior** arayüzünü uygulamaktadır. Bu arayüzün senaryomuza göre en önemli metodu ise **ApplyDispatchBehavior** fonksiyonudur. Bu fonksiyon içerisinde dikkat edileceği üzere **TextFileInspector** isimli sınıfımıza ait bir nesne örneğinin **ParameterInspectors** koleksiyonuna eklendiği görülmektedir. Dolayısıyla operasyonların yakalanması sırasında kullanılacak olan tipin bildirimi sağlanmıştır. Buna göre çalışma zamanında **TextFileInspectorOperationBehavior** niteliğinin uygulandığı servis operasyonlarına gelen çağrılarda, hangi Inspector tipinin devreye gireceği bildirilmiş olmaktadır.

Aslında bu nitelik bildirimi yeterlidir. Nitekim servis operasyonlarına söz konusu niteliğin uygulaması ile zaten çalışma zamanı bilgilendirilmiş olacaktır. Ancak bir servisin tüm operasyonlarının izlenmesi istendiği bir durum söz konusu ise tüm operasyonların başında **TextFileInspectorOperationBehavior** uygulamak yerine, sınıfın başında bir nitelik ile servis davranışının bildirilmesi daha doğrudur. Bu nedenle servis sınıfına uygulanabilen bir davranışın bildirilmesi önemlidir. İşte bu amaçla aşağıdaki sınıfı yazmamız yeterli olacaktır.




```
using System;
using System.ServiceModel.Description;

namespace InspectorLib
{
    [AttributeUsage(AttributeTargets.Class)]
    public class TextFileInspectorServiceBehavior
        :Attribute,IServiceBehavior
```

```
    {
        #region IServiceBehavior Members

        public void AddBindingParameters(ServiceDescription serviceDescription,
System.ServiceModel.ServiceHostBase serviceHostBase,
System.Collections.ObjectModel.Collection<ServiceEndpoint> endpoints,
System.ServiceModel.Channels.BindingParameterCollection bindingParameters)
        {
        }

        public void ApplyDispatchBehavior(ServiceDescription serviceDescription,
System.ServiceModel.ServiceHostBase serviceHostBase)
        {
            foreach (ServiceEndpoint endpoint in serviceDescription.Endpoints)
                foreach (OperationDescription operation in endpoint.Contract.Operations)
                    operation.Behaviors.Add(new TextFileInspectorOperationBehavior());
        }

        public void Validate(ServiceDescription serviceDescription,
System.ServiceModel.ServiceHostBase serviceHostBase)
        {
        }

        #endregion
    }
}
```

Attribute olarak sınıflara uygulanabilen bir tip söz konusudur. Buna göre örneğimizde yer alan **AccountService** sınıfına uygulanabilecek bir tiptir.
Ayrıca **IServiceBehavior** arayüzünün bir implementasyonu söz konusudur ki buna göre yazılması gereken en önemli metod **ApplyDispatchBehavior** fonksiyonudur. Bu fonksiyon içerisinde servis içerisindeki tüm **Endpoint'** ler dolaşılmaktadır. Bu **Endpoint'** lerin her birinin de sözleşmelerinde tanımlanmış olan operasyonlarına gidilmektedir.
Yani **OperationContract** niteliği uygulanan metodlarına ulaşılır. Bulunan her bir operasyon için de, az önce tanımlanan **TextFileInspectorOperationBehavior** davranış örneği bildirimi yapılmaktadır.

Bu arada **TextFileInspector**, **TextFileInspectorOperationBehavior** ve **TextFileInspectorServiceBehavior** sınıflarını ayrı bir kütüphane altında toplamamızda yarar vardır. Nitekim söz konusu tipleri **Host** uygulama da ve servis kütüphanesinde kullanmamız gerekmektedir.

Peki ne servis sınıfında ne de operasyon metodlarında yukarıda tanımlanmış olan nitelikler kullanılmazsa ve biz yine de TextFileInspector tipini kullandırtmak istersek. Bu durumda örneğin Host uygulama tarafında aşağıdaki kodlamaları yapmamız yeterli olacaktır.

```
TextFileInspectorServiceBehavior tfServiceBehavior = new
TextFileInspectorServiceBehavior();
TextFileInspectorServiceBehavior sb =
host.Description.Behaviors.Find<TextFileInspectorServiceBehavior>();
if (sb == null)
   host.Description.Behaviors.Add(tfServiceBehavior);

host.Open();
```

Görüldüğü üzere **ServiceHost** nesne örneği üzerinden **Open** metodu çağırıldıktan sonra servise **TextFileInspectorServiceBehavior** davranışının uygulanıp uygulanmadığına bakılmaktadır. Eğer uygulanamışsa **Behaviors** koleksiyonuna **Add** metodu ile ekleme işlemi gerçekleştirilmektedir.

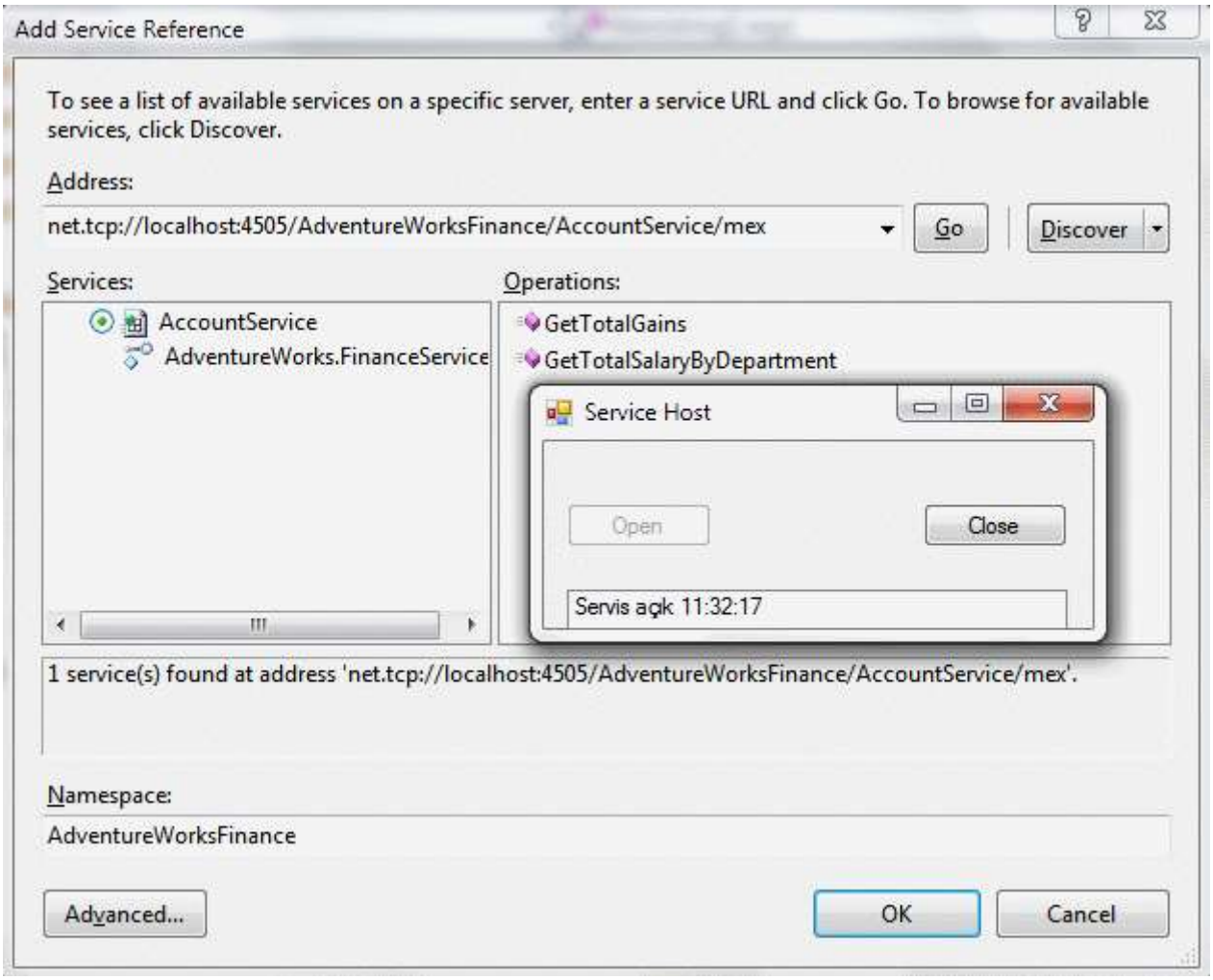
Şimdi gelelim operasyon izleme işleminin niteliklerimiz yardımıyla nasıl bildirileceğine. Bu amaçla AccountService sınıfını aşağıdaki gibi güncellememiz yeterli olacaktır.

```
using InspectorLib;

namespace AdventureWorksFinance
{
   //[TextFileInspectorServiceBehavior]
   public class AccountService
      : IAccountService
   {
      #region IAccountService Members

      [TextFileInspectorOperationBehavior]
      public double GetTotalSalaryByDepartment(string departmentCode)
      {
         return 1245000;
      }

      [TextFileInspectorOperationBehavior]
      public double GetTotalGains()
      {
         return 2500000;
      }
```

```
        #endregion
    }
}
```

Bu örnek kullanımda operasyon metodlarının
başında **TextFileInspectorOperationBehavior** niteliği uygulanmıştır. Böylece çalışma zamanına hangi operasyonların izleneceği bildirilmiş olmaktadır. Ancak istersek sınıfın başındaki **TextFileInspectorServiceBehavior** niteliğini etkinleştirerek, servis içerisindeki tüm metodların,**TextFileInspectorOperationBehavior** uygulanmadan izlenebilmesi de sağlanabilir

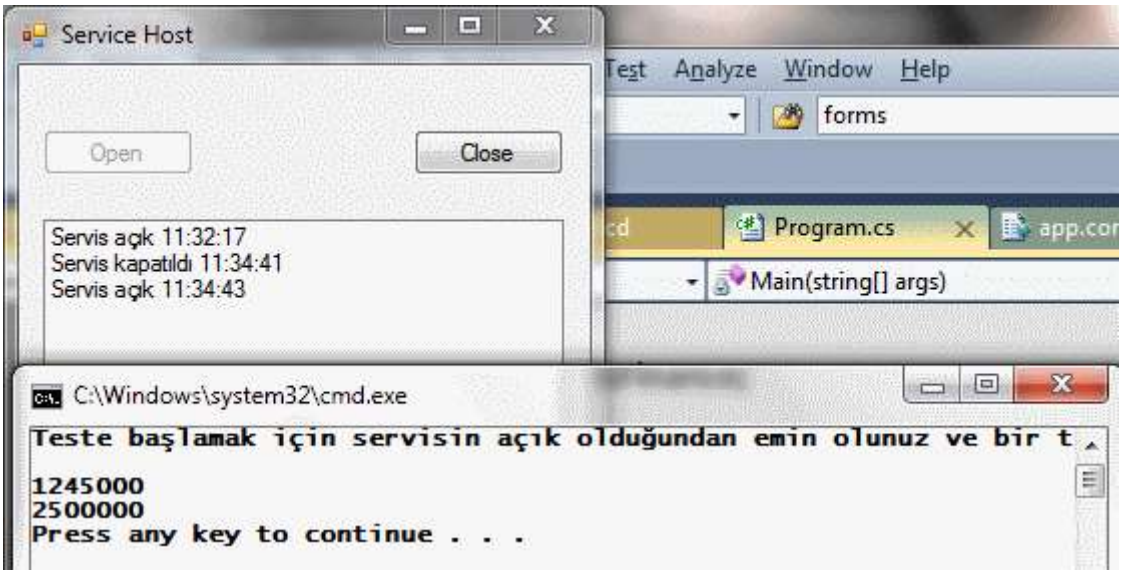
Şimdi senaryomuzu test etmeye başlayabiliriz. Tabi bu amaçla basit bir istemci uygulama yazmamız ve servisimizi referans etmemiz gerekmektedir. Burada **Self-Hosted** bir servis çalışma zamanı söz konusu olduğundan, **proxy** üretimi için **host** uygulamanın öncelikli olarak çalışıtırlması ve servisin açılması gerektiği unutulmamalıdır. Bu işlemin ardından Mex talebi yapılabilir.



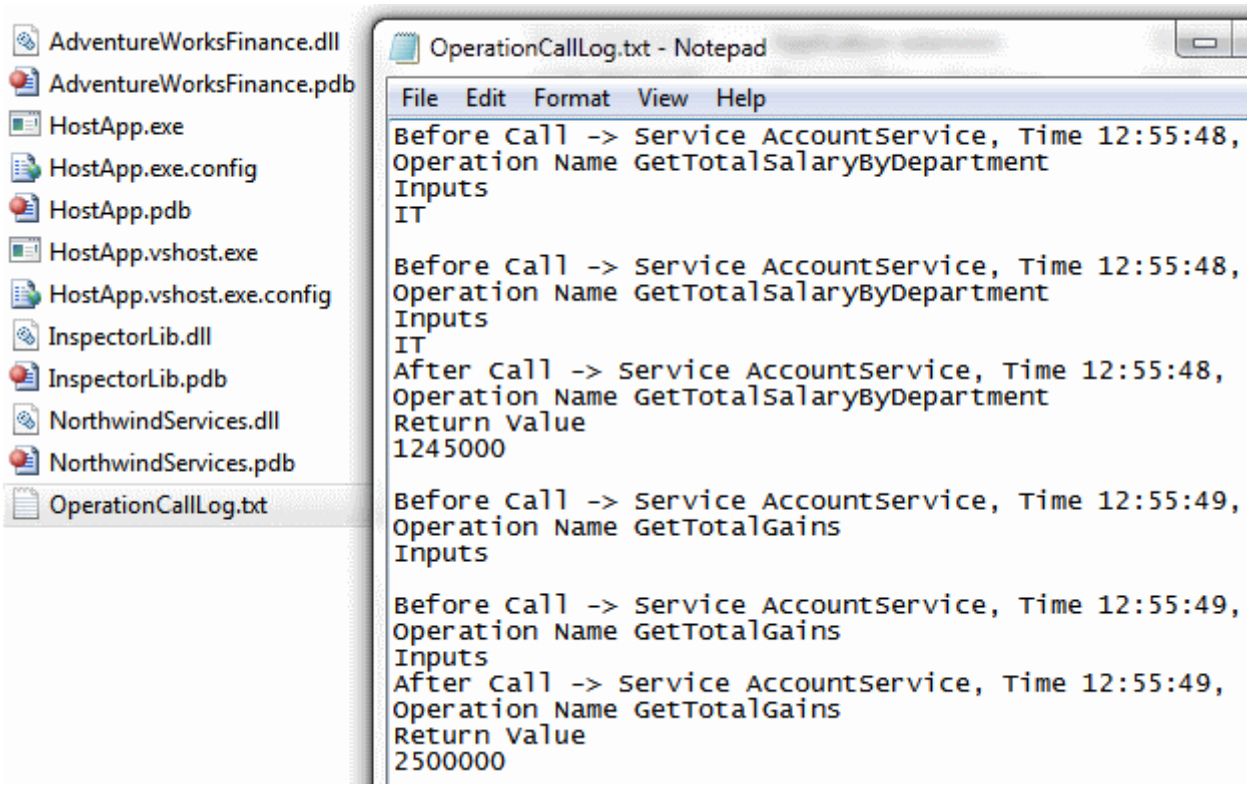
Test amacıyla aşağıdaki kodları yazabiliriz.

```
using System;
using ClientApp.AdventureWorksFinance;
using System.Threading;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Teste başlamak için servisin açık olduğundan emin olunuz ve
bir tuşa basınız");
         Console.ReadLine();
         AdventureWorksFinanceServicesClient proxy = new
AdventureWorksFinanceServicesClient("NetTcpBinding_AdventureWorks.FinanceS
ervices");
         double result=proxy.GetTotalSalaryByDepartment("IT");
         Console.WriteLine("{0}",result.ToString());
         Thread.Sleep(1000);
         double totalGains=proxy.GetTotalGains();
         Console.WriteLine("{0}",totalGains.ToString());
      }
   }
}
```

öncelikli olarak **host** ve sonrasında **istemci** uygulamayı çalıştırdığımızı düşünecek olursak aşağıdaki ekran görüntüsündekine benzer sonuçlarla karşılaşırız.



Ancak bizim için önemli olan ve ulaşmak istediğimiz nokta text dosyası içerisine atılan bilgilerdir. İşte sonuçlar.



Görüldüğü üzere **GetTotalSalaryByDepartment** ve **GetTotalGains** metodlarına yapılan çağrıların öncesi ve sonrasına ait bilgiler için **OperationCallLog.txt** isimli dosya oluşturulmuş ve içeriği üretilmiştir.

Sonuç olarak operasyonlara yapılan çağrıları izlemek ve istihbarat toplamak adına buradaki senaryodan yararlanılabilir. Elbette senaryonun geliştirilmesi gereken pek çok yanı vardır. Söz gelimi şu anki örnekte

- dosya tabanlı **IO** işlemlerinin sayısı oldukça fazla olabilir. özellikle eş zamanlı gelecek operasyon çağrılarında **StreamWriter** tipinin **exception** verme olasılığı*(Paylaşılan dosya kaynağı nedeni ile)* an meselesidir
- Dosya tabanlı olarak yapılan veri toplama işlemi yerine, veritabanı odaklı bir çözümde geliştirilebilir. Yani operasyon çağrılarına ait bilgiler dosya yerine bir veritabanı tablosuna yazdırılabilir.
- Senaryo tek bir servis örneğini izleyecek şekilde çalışmaktadır. Oysaki host uygulamanın birden fazla servisi dış dünyaya sunma olasılığı bulunmaktadır.

Bu gibi durumları iyice irdeleyerek ilerlemekte yarar vardır. İşte size çalışmak için bir sürü ev ödevi Bu arada örnek Solution içerisinde ikinci bir servis daha yer almaktadır. Bu serviside devreye alarak operasyon izleme işlemlerini birden fazla servis için gerçeklemeye çalışabilirsiniz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

OperationCallTracing.rar (155,62 kb) **[örnek Visual Studio 2010 Ultimate Sürümünde Geliştirilmiş ve Test Edilmiştir]**

TCP Bazlı WCF Service ve Silverlight İstemcileri (2010-12-19T01:10:00)

nettcpbinding,wcf,wcf services,silverlight 4.0,silverlight,

Merhaba Arkadaşlar,

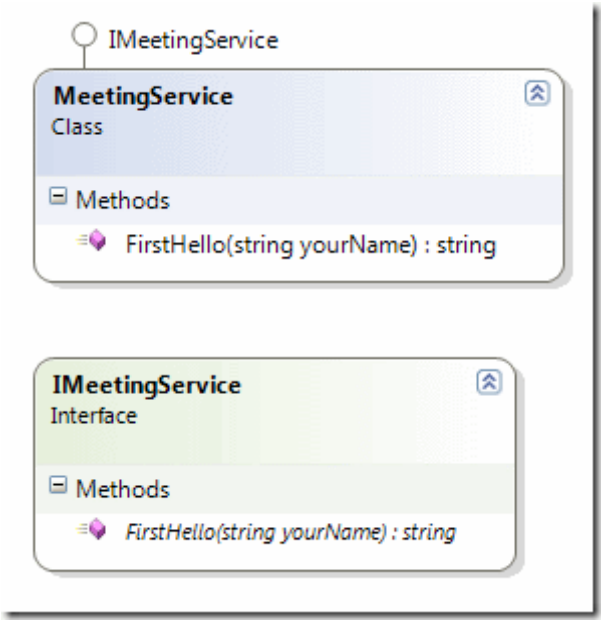
Bir yazar, hazırlayacağı hikaye için çoğu zaman çevrede dolaşıp malzeme toplar. Olayın kahramanlarını tasvir etmek için çevredeki insanları göz önüne alır. Hatta gezdiği yerleri inceler. Bu açıdan bakıldığında iyi yazarların aslında çok iyi birer gözlemci olduğu söylenebilir.

Sonuç olarak yazarın elinde bir senaryo taslağı oluşur. Artık tek yapması gereken sakin bir köşe bulmak ve daktilosunun başına geçerek*(ki günümüde büyük bir olasılıkla bu diz üstü bir bilgisayar olacaktır)* yazmaya başlamaktır. çözülmesi en zor olan parçaların başında kitaba bir isim bulmak ve ilk giriş cümlesini yazmak gelmektedir. Her ne kadar bu güne kadar yazılmış bir kitabım olmasa da böyle olduğunu tahmin etmekteyim.

Bu gün yazımız içinde elimizde bir takım malzemelerimiz bulunmakta. Bir adet **TCP** bazlı olarak çalışan **WCF(Windows Communication Foundation)** servisi. Bu servisi kullanan **Silverlight 4.0**tabanlı bir istemci. **TCP** bazlı servisimiz son derece zıpkın bir delikanlı aslında. Nitekim şirketin iç ağı üzerinden **Binary** tabanlı mesaj formatını kullandığı için ondan daha hızlısı neredeyse yok gibi. Diğer yandan **Silverlight** istemcimiz son derece yakışıklı ve zengin bir kız*(Rich Internet Application)*. İşte bu yazımızda bu iki kişiyi buluşturmaya çalışıyor olacağız. Ne varki arada zıpkın delikanlının bir de ablası var ki o da **IIS(Internet Information Services)** mahallesinde oturmayı istemeyen ama hep hayal eden **Self-Hosted** stilde yazılmış bir **Console** uygulaması. Neredeyse içi kap kara olmuş birisi*(Ama biz çalışma zamanı ekranında onun içindeki iyiliği beyaza boyayıp çıkartacağız)* Bakalım abla, kızın, erkek arkadaşına ulaşmasına izin verecek mi? 😃

Varsayılan Olarak

Normal şartlar altında **Silverlight** istemcilerinin genellikle **HTTP** bazlı çalışan **WCF** servislerini kullanması söz konusudur. Hatta **WCF RIA Services**' ler en sık kullanılanıdır. Ancak**Intranet** tabanlı bir sistemde **TCP** bazlı **WCF** Servisleri de söz konusu olabilir. Dilerseniz olayı örnekleyerek canlandırmaya çalışalım. İlk olarak elimizin altında aşağıdaki gibi bir **WCF Service Library** içeriğinin olduğunu düşünelim.



Servis sözleşmemiz oldukça basit bir içeriğe sahip.

```
using System.ServiceModel;

namespace YourMeetingServices
{
   [ServiceContract]
   public interface IMeetingService
   {
      [OperationContract]
      string FirstHello(string yourName);
   }
}
```

Sadece bir Merhaba demek için gerekli operasyon tanımını içermekte. Bu operasyon ise aşağıdaki gibi uygulanmakta.

```
namespace YourMeetingServices
{
   public class MeetingService
     : IMeetingService
   {
      #region IMeetingService Members

      public string FirstHello(string yourName)
      {
         return string.Format("Merhaba {0}", yourName);
      }
```

```
      #endregion
   }
}
```

Bu servis kütüphanesini referans ederek host eden **Console** uygulamasının içeriği ise aşağıdaki gibi.

```
using System;
using System.ServiceModel;
using YourMeetingServices;

namespace ServerApp
{
   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(
            typeof(MeetingService)
            );
         host.Opened += (o, e) =>
            {
               Console.WriteLine("Servis dinlemede");
            };
         host.Closed += (o, e) =>
            {
               Console.WriteLine("Servis kapatıldı");
            };
         host.Open();
         Console.WriteLine("çıkmak için bir tuşa basınız");
         Console.ReadLine();
         host.Close();
      }
   }
}
```

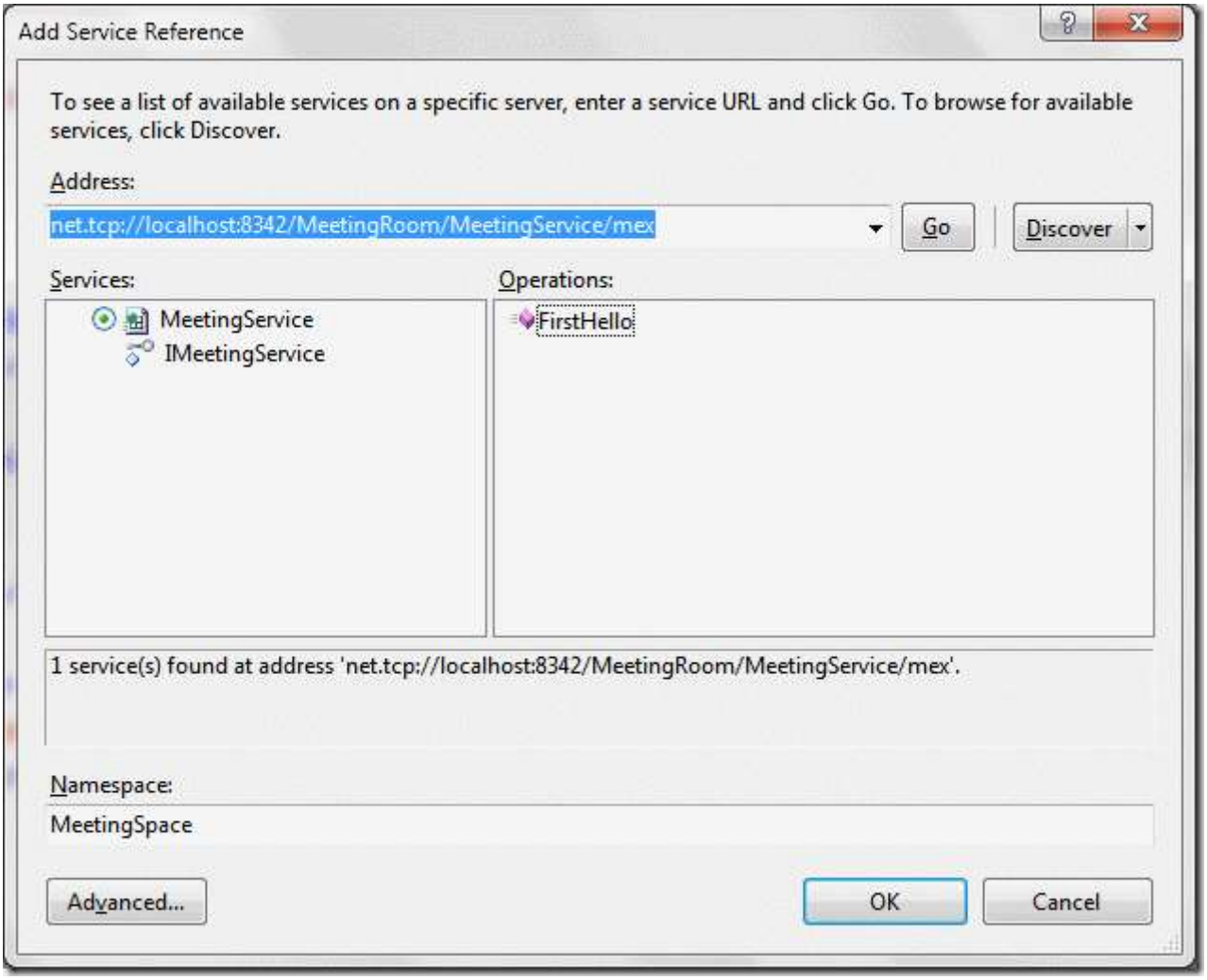
çok doğal olarak config dosyası içeriğinin de aşağıdaki gibi olduğunu söyleyebiliriz.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <behaviors>
         <serviceBehaviors>
            <behavior>
               <serviceMetadata/>
```

```
            </behavior>
          </serviceBehaviors>
        </behaviors>
        <services>
          <service name="YourMeetingServices.MeetingService">
            <endpoint address="" binding="netTcpBinding"
contract="YourMeetingServices.IMeetingService"/>
            <endpoint address="mex" binding="mexTcpBinding"
contract="IMetadataExchange" />
            <host>
              <baseAddresses>
                <add
baseAddress="net.tcp://localhost:8342/MeetingRoom/MeetingService/" />
              </baseAddresses>
            </host>
          </service>
        </services>
    </system.serviceModel>
</configuration>
```

Buna göre servisimiz **TCP** bazlı olaraktan **net.tcp://localhost:8342/MeetingRoom/MeetingService/** adresi üzerinden yayın yapmaktadır. Ayrıca **mexTcpBinding** bağlayıcı tipini kullanan ve **Mex**son ekini var olan **Base Address** tanımına ekleyerek, **service metadata publishing** yapan bir **EndPoint'** e de sahiptir.

Buna göre basit bir **Console** istemcisinin söz konusu servisi kullanması için tek yapması gereken, Servis uygulaması çalışırken **Add Service Reference** seçeneğinde aşağıdaki şekilde görülen adres tanımlamasını kullanmak olacaktır.

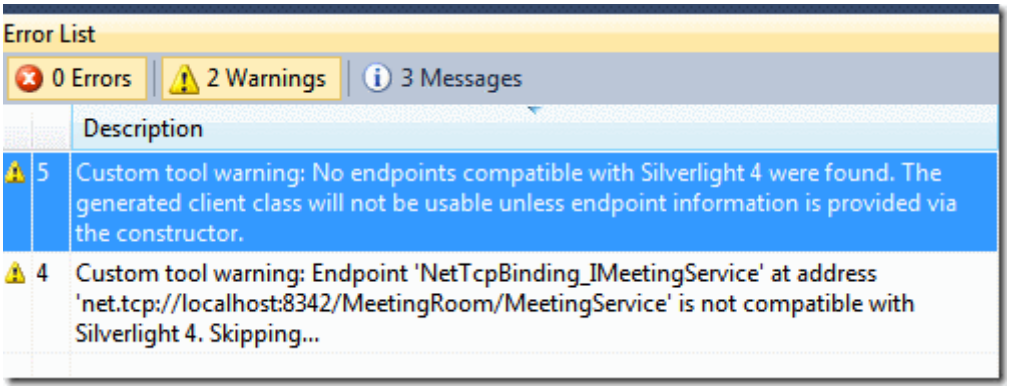


Buna göre sıradan bir istemcinin söz konusu servisi kullanması mümkün ve kolaydır.

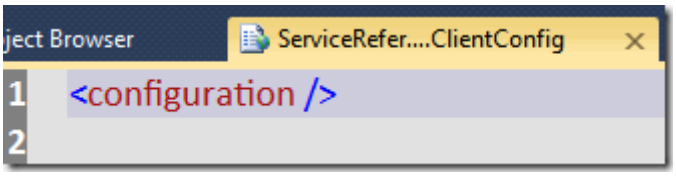
İlk Tanışma

Ancak söz konusu istemci bir **Silverlight** uygulaması ise biraz daha sıkıntılı bir durumla karşı karşıya olabiliriz. Dilerseniz bu durumu analiz etmek için yukarıda yazmış olduğumuz servisimizi bir **Silverlight** istemcisine referans etmeye çalışalım. Tabiki Sunucu uygulamanın çalışıyor olması gerektiğini hatırlatmayacağım.

Sunucu uygulama çalışıyor ve servis iletişime açık iken, **Silverlight** uygulamasına referans ekleme işlemi başarılı olacaktır. Ancak aşağıdaki şekildeki gibi iki adet **Warning'** in de oluştuğu görülecektir.



üstelik üretilen config dosyası içeriğine bakıldığında aşağıdaki görüntü ile karşılaşılır.



Uppsss!!! İlginç bir durum. Nitekim **Endpoint** üretimlerinin yapılmaması bir yana, **WCF** servisi ile olan iletişim için gerekli hiç bir ayar da bulunmamaktadır. Buna göre **Abla**' nın, iletişimi engellediğini ifade edebiliriz.

İkinci Karşılaşma

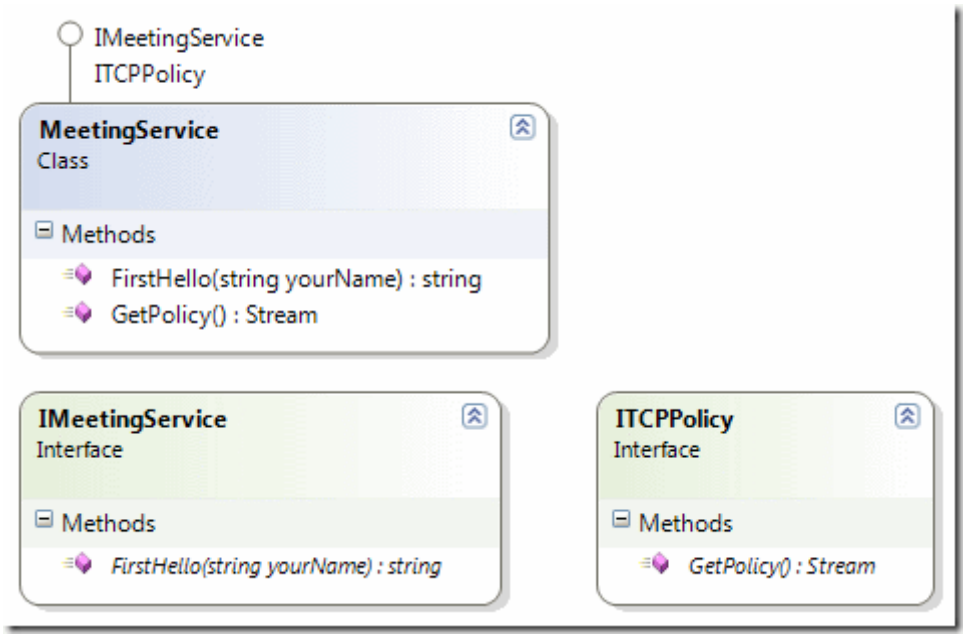
İlk karşılaşmanın başarısızlığı üzerine servis ile ilişkili olarak bir takım ilkelerin uygulanması gerektiğini söyleyebiliriz. Her şeyden önce sunucunun, **Silverlight** istemcilerinin**HTTP** bazlı olarak **80** portu üzerinden erişebilmelerine izin veriyor olması gerektiği ip ucunu verebiliriz. Bir başka deyişle **Cross Domain** için **ClientAccessPolicy.xml** ve doğru içeriğin uygulanması gerekliliği söz konusudur. Bu amaçla servis kütüphanemize aşağıdaki servis sözleşmesini eklediğimizi düşünelim.

```
using System.IO;
using System.ServiceModel;
using System.ServiceModel.Web;

namespace YourMeetingServices
{
   [ServiceContract]
   public interface ITCPPolicy
   {
      [OperationContract, WebGet(UriTemplate = "/clientaccesspolicy.xml")]
      Stream GetPolicy();
   }
}
```

Burada **WebGet** niteliğinin de uygulandığı*(ki bunun için System.ServiceModel.Web.dll assembly' ının projeye referans edilmesi gerekir)* bir operasyon yer almaktadır. Bu operasyon geriye **Stream**tipi tarafından taşınabilen bir referans döndürmektedir. Söz konusu operasyona ulaşılırken **HTTP Get** metodunda **Clientaccesspolicy.xml** son ekinin kullanılacağı da ifade edilmektedir. Bir başka deyişle bu sözleşme istemci için gerekli olan **ClientAccessPolicy** içeriğini sunacaktır. çok doğal olarak bu sözleşmenin ilgili tipe uygulanıyor olması gerekmektedir. Aynen aşağıda görüldüğü gibi.

```
using System;
using System.IO;
using System.ServiceModel.Web;
using System.Text;

namespace YourMeetingServices
{
   public class MeetingService
      : IMeetingService,ITCPPolicy
   {
      #region IMeetingService Members

      public string FirstHello(string yourName)
      {
         return string.Format("Merhaba {0}", yourName);
      }

      #endregion

      #region ITCPPolicy Members

      public Stream GetPolicy()
      {
         string content =
File.ReadAllText(Path.Combine(Environment.CurrentDirectory,
"PolicyContent.xml"));
         WebOperationContext.Current.OutgoingResponse.ContentType =
"application/xml";
         return new MemoryStream(Encoding.UTF8.GetBytes(content));
      }

      #endregion
   }
}
```

Yapılan bu değişiklikler sonucuda servis kütüphanesinin içeriğinin özetle aşağıdaki şekile görüldüğü gibi olduğunu ifade edebiliriz.



GetPolicy metodunun uygulanışı içerisindeki en önemli nokta sır gibi duran **content** değişkeninin değeridir. Bu değer içeriği aşağıdaki gibi olan **PolicyContent.xml** dosyasından getirilmektedir. Söz konusu dosyanın **output** klasörü sunucu uygulamaya ait exe çıktısının olduğu yer olarak belirtilmiştir.

```
<?xml version="1.0" encoding="utf-8"?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from http-request-headers="*">
        <domain uri="*"/>
      </allow-from>
      <grant-to>
        <socket-resource port="8342" protocol="tcp" />
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

Dikkat edileceği üzre **TCP** bazlı **8342** numaralı port için garanti verilmiştir. Tabi buna göre sunucu uygulama üzerinde de bir takım değişikliklerin yapılması gerekmektedir. Aslında**App.config** dosyası içerisinde aşağıdaki değişiklikleri yapmamız yeterli olacaktır.

```
<?xml version="1.0"?>
<configuration>
    <system.serviceModel>
        <bindings>
            <netTcpBinding>
                <binding name="TcpBindingConfiguration">
```

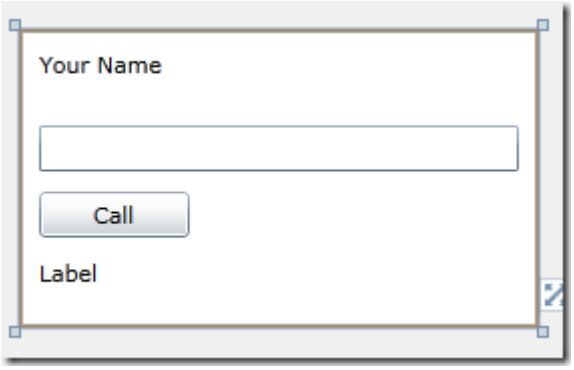
```
            <security mode="None" />
          </binding>
        </netTcpBinding>
      </bindings>
      <behaviors>
        <endpointBehaviors>
          <behavior name="WebBehavior">
            <webHttp />
          </behavior>
        </endpointBehaviors>
        <serviceBehaviors>
          <behavior name="">
            <serviceMetadata />
          </behavior>
        </serviceBehaviors>
      </behaviors>
      <services>
        <service name="YourMeetingServices.MeetingService">
          <endpoint address="" binding="netTcpBinding"
bindingConfiguration="TcpBindingConfiguration"
            name="TcpEndpoint" contract="YourMeetingServices.IMeetingService" />
          <endpoint address="mex" binding="mexTcpBinding" name="MexTcpEndpoint"
            contract="IMetadataExchange" />
          <endpoint address="" behaviorConfiguration="WebBehavior"
binding="webHttpBinding"
            name="WebHttpEndpoint"
contract="YourMeetingServices.ITCPPolicy" />
          <host>
            <baseAddresses>
              <add baseAddress="http://localhost:8080" />
              <add
baseAddress="net.tcp://localhost:8342/MeetingRoom/MeetingService" />
            </baseAddresses>
          </host>
        </service>
      </services>
    </system.serviceModel>
<startup><supportedRuntime version="v4.0"
sku=".NETFramework,Version=v4.0"/></startup></configuration>
```

İlk dikkati çeken nokta **WebHttp** davranışını destekleyen **WebHttpBinding** bağlayıcı tipini kullanan ve **ITCPPolicy** sözleşmesini sunan bir **Endpoint'** in eklenmiş olmasıdır. üstelik bu**Endpoint, base** olarak **http://localhost:8080** adresini kullanmaktadır.

Diğer yandan **TCP** Bazlı iletişim sağlayan **Endpoint** için **security mode** değeri **none** olarak işaret edilmiştir. Bu durumda **Silverlight** uygulamamızda**net.tcp://localhost/8342/MeetingRoom/MeetingService/mex** adresi üzerinden yapılan referans ekleme işlemi sonrasında, daha önceden aldığımız **Warning** mesajlarının kalktığı ve aşağıdaki istemci **config** içeriğinin oluştuğu görülecektir.

```
<configuration>
  <system.serviceModel>
    <bindings>
      <customBinding>
        <binding name="TcpEndpoint">
          <binaryMessageEncoding />
          <tcpTransport maxReceivedMessageSize="2147483647"
maxBufferSize="2147483647" />
        </binding>
      </customBinding>
    </bindings>
    <client>
      <endpoint address="net.tcp://localhost:8342/MeetingRoom/MeetingService"
        binding="customBinding" bindingConfiguration="TcpEndpoint"
        contract="MeetingSpace.IMeetingService" name="TcpEndpoint" />
    </client>
  </system.serviceModel>
</configuration>
```

Artık **Silverlight** istemcisi üzerinden bir test gerçekleştirebiliriz. Bu amaçla **MainPage** içeriğini aşağıdaki gibi oluşturduğumuzu düşünelim.



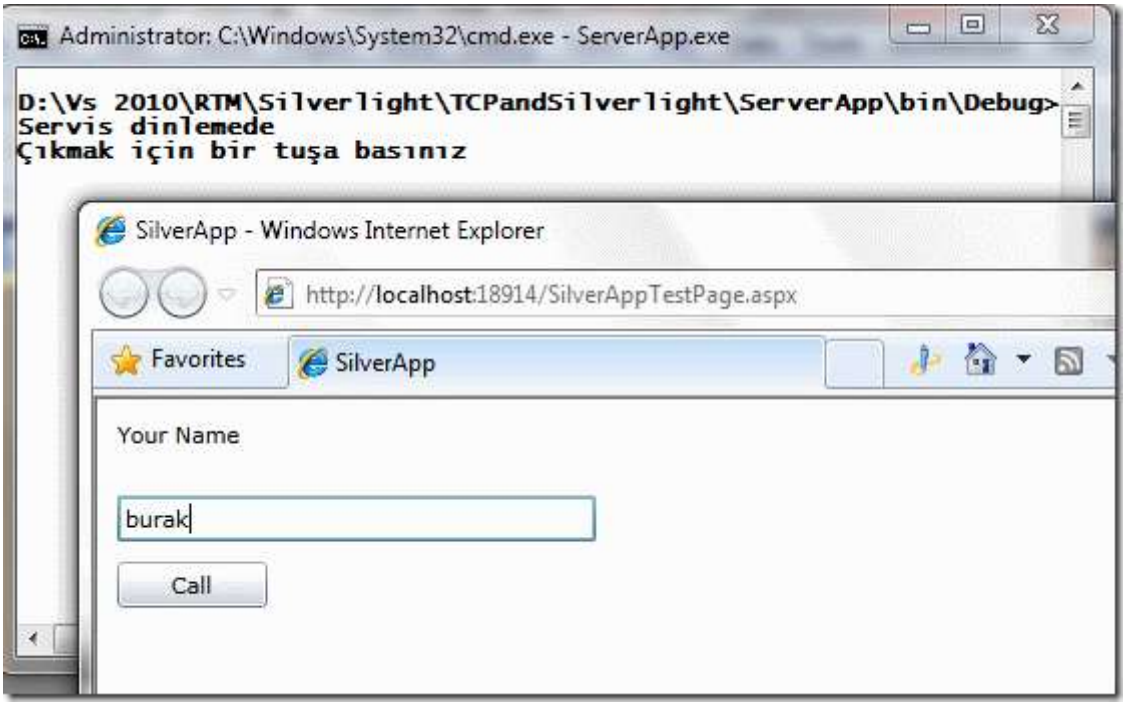
MainPage.xaml.cs

```
using System;
using System.Windows;
using System.Windows.Controls;
using SilverApp.MeetingSpace;
```

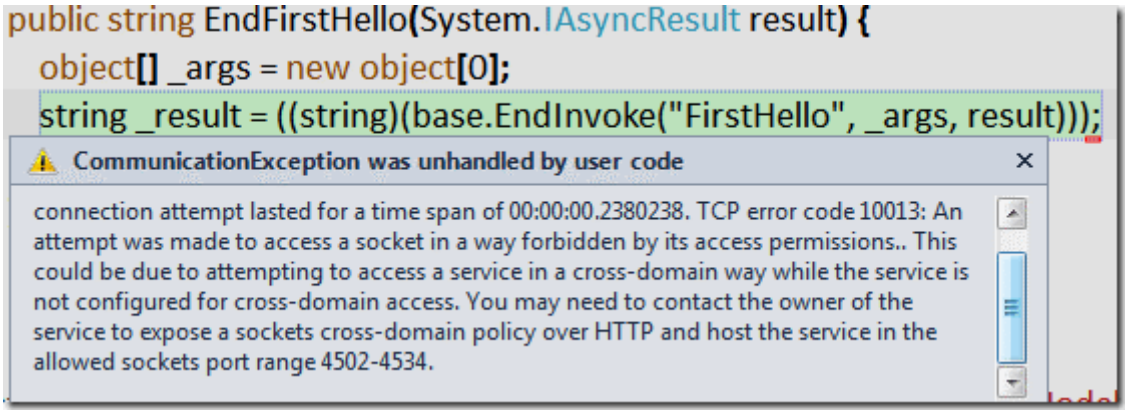
```
namespace SilverApp
{
   public partial class MainPage : UserControl
   {
      MeetingServiceClient proxy = null;

      public MainPage()
      {
         InitializeComponent();
         proxy = new MeetingServiceClient("TcpEndpoint");
         proxy.FirstHelloCompleted += new
EventHandler<FirstHelloCompletedEventArgs>(proxy_FirstHelloCompleted);
      }

      void proxy_FirstHelloCompleted(object sender, FirstHelloCompletedEventArgs e)
      {
         if (e.Error != null)
            lblCallResult.Content = "Bir sorun oluştu";
         else if (e.Cancelled)
            lblCallResult.Content = "İşlem iptal edildi";
         else
           lblCallResult.Content = e.Result;
      }

      private void btnCall_Click(object sender, RoutedEventArgs e)
      {
         proxy.FirstHelloAsync(txtYourname.Text);
      }
   }
}
```

Görüldüğü üzere, **config** dosyasında **TcpEndpoint** için tanımlanmış olan ayarlara göre ilgili **WCF** servisine asenkron olarak bir çağrı gerçekleştirilmektedir.

Şimdi ilk testimizi yapalım. önce sunucu uygulamayı ardından da **Silverlight Web** uygulamamızı çalıştıralım. İlk etapta her şey güllük gülistanlıktır. Aynen aşağıdaki şekilde görüldüğü gibi.



Ancak **Call** başlıklı **Button** kontrolüne bastığımızda aşağıda görülen **çalışma zamanı istisnasını(Runtime Exception)** aldığımızı görürüz.



Aslında buradaki hata mesajının tam içeriği şöyledir;

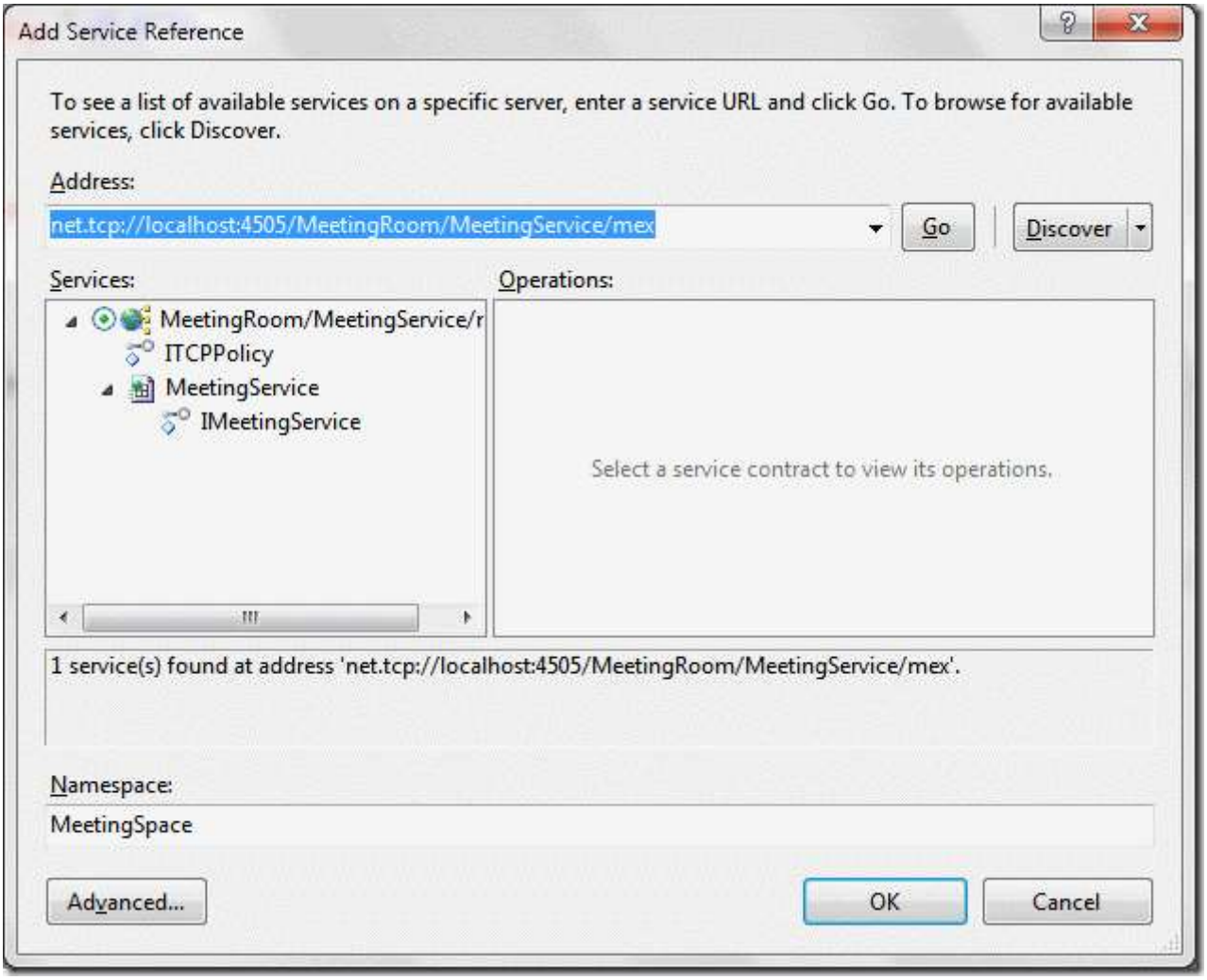
Could not connect to net.tcp://localhost:8342/MeetingRoom/MeetingService. The connection attempt lasted for a time span of 00:00:00.2250225. TCP error code 10013: An attempt was made to access a socket in a way forbidden by its access permissions.. This could be due to attempting to access a service in a cross-domain way while the service is not configured for cross-domain access. You may need to contact the owner of the service to expose a sockets cross-domain policy over HTTP and host the service in the allowed sockets port range 4502-4534.

Anlaşılacağı üzere bir **Cross Domain Policy** sorunsalı baş göstermiştir gibi durmaktadır.

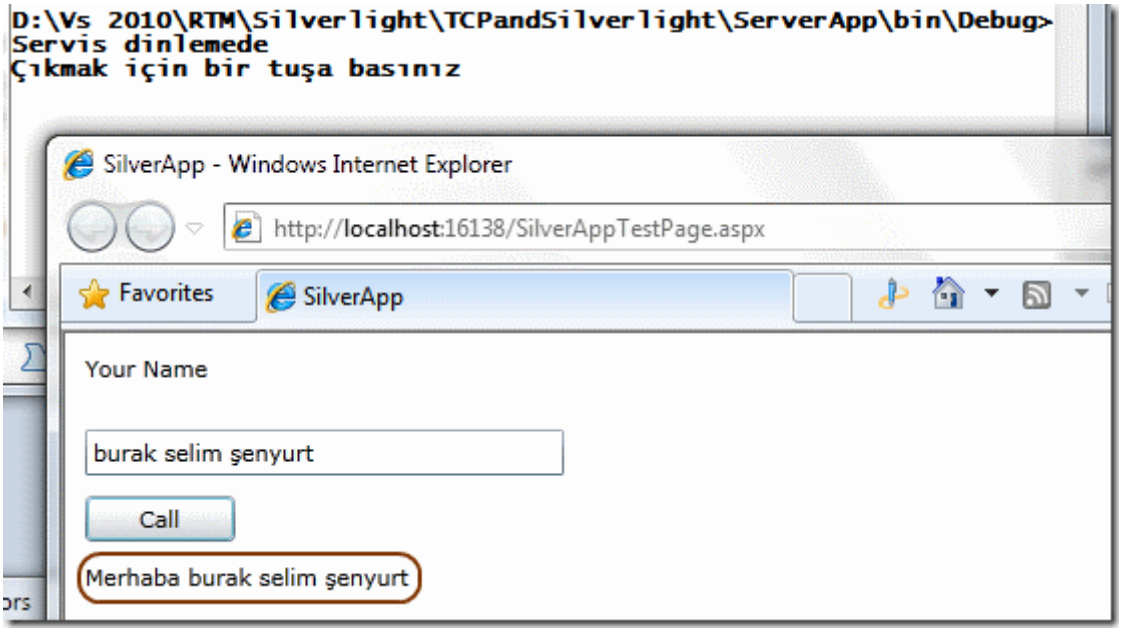
Son Karşılaşma

Aslında Servisin geliştirilme mantığına göre **IIS**' in olmadığı bir durum simüle edilmektedir. Nitekim **Siverlight** istemcilerinin **TCP** bazlı bir servisi tüketmesi için bu servisin **IIS** üzerinde host edilmesi ve **ClientAccessPolicy.xml** dosyası ile desteklenerek gerekli güvenlik izinlerinin verilmesi yeterlidir. Dolayısıyla bizim geliştirdiğimiz senaryoda **Servis** uygulamasının **IIS** gibi çalıştığı varsayılabilir. Buna göre ilk olarak **HTTP base address** tanımlamasının **http://localhost:80** şeklinde değiştirilmesi düşünülmelidir.

Diğer yandan çalışma zamanındaki hata mesajı **4502** ile **4534** numaralı portlar arasında bir değerin kullanılmasını beklemektedir. Bu sebepten servis tarafındanki **TCP based address**değerinin de örnek olarak **net.tcp://localhost:4505/MeetingRoom/MeetingService** şeklinde değiştirilmesi düşünülebilir. Yani 4502 ile 4534 arasında bir port değeri atanmalıdır. Ancak bu da yeterli olmayacaktır. Nitekim **policy** içeriğini teşkil eden **XML** dosyasında yer alan port numarası da **4505** olarak ayarlanmalıdır. Tüm bu değişiklikler **Silverlight** istemcisine servis referansının yeniden eklenmesini gerektirecektir.



Artık tanışmak için son bir deneme yapılabilir. İşte sonuç.



Görüldüğü üzere servis tarafındaki metod başarılı bir şekilde çalışmıştır. Peki bu kadar zahmete girmeye gerek var mıdır? Aslında olmadığını söylersem şu anda bana çok kızabileceğinizi düşünüyorum. Ancak var. Nitekim **IIS(Internet Information Services)** üzerinde **WAS(Windows Process Activation Service)** kullanımı sayesinde **host**edebileceğimiz **TCP** bazlı bir servisin, doğru **ClientAccessPolicy.xml** içeriği ile bir **Silverlight** istemcisi tarafından kullanılabilmesi mümkündür. Şu anda umuyorum ki içinizde **@#$½!:=|<>** gibi bir şey demiyorsunuzdur 😐 Tabi yapılan örneğe göre kafalarda hale soru işaretleri oluşabilir. Söz gelimi **80** yerine örneğin **4508** numaralı bir port üzerinden iletişim geçerli olsa(**http://localhost:4508** şeklinde) Yine de hatalar ile karşılaşır mıyız acaba? 😉 Bu sorunun araştırılmasını siz değerli okurlarıma bırakıyorum. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

TCPandSilverlight.rar (1,80 mb) **[örnek Visual Studio 2010 Ultimate üzerinde ve Silverlight 4.0 odaklı olarak geliştirilmiştir]**

Yıllar Sonra Yeniden Enum Sabitleri (2010-12-19T01:05:00)

c#,c# temelleri,enum type,

Merhaba Arkadaşlar,

Aslında çok şanslı bir insanım. çünkü çocukluğumdan beri tatile gidebiliyorum. Yazlığımızın olduğu Avşa adasına her sene iki günlüğüne dahi olsa gitmekteyim. İlk yıllarda adaya hareket eden nostaljik gemiler vardı. Bu

gemiler yaklaşık olarak altı saat gibi bir sürede **İstanbul**' dan adaya ulaşmaktaydı.

İlerleyen yıllarda hızlandıkça deniz yüzeyinin üstünde yükselen kanatlı gemilerle karşılaştık. Ne yazık ki bu hızlı gemiler daha çok dalgasız göller için yapıldıklarından her tür havada gidemiyorlardı.

Derken bu günlerde çok meşhur olan **Mavi Marmara** gemisinin seferlere başladığını gördük. Ancak bunun çok öncesinde deniz otobüsü seferleri de başlamıştı. Deniz otobüsleri ile adaya 3 saat gibi bir sürede gidilebiliyordu. Tabi kötü hava koşullarında*(özellikle fırtınalı günlerde)*, hırçın **Marmara Denizinde** bu gemilerle seyahet etmekte ayrı bir maceraydı. çok sık başıma gelen bir durum du. Hemen her yaz, en az bir veya iki kere fırtınaya yakalanıyordum.

Tabi Avşa adasına ulaşmanın tek yolu gemi ve deniz otobüsü değil. **Tekirdağ** ve **Silivri** üzerinden de araba taşıyan motorlar ile de gidebilirsiniz. üstelik bu motorların ağırlık merkezleri ve tasarımları nedeniyle, katamaran tipinden olan deniz otobüslerine göre daha az salladıklarını ifade edebilirim. Bir diğer yolda **Yenikapı'** dan hızlı feribot ile **Bandırma'** ya geçmek, oradan yarım saatlik bir yol ile **Erdek** limanına gelmek ve yine araba taşıyan gemilerle Avşa adasına ulaşmaktır.

Yine böyle fırtınalı bir gündü ve ben **Bizitek** firmasında henüz **Junior** geliştiriciydim. Şirketimin bana tahsis ettiği **IBM R-51** diz üstü bilgisayarımı kullanmaktaydım. **Avşa**' dan döndüğüm bir pazar günü yine çok fırtınalı bir havaya denk geldim. Kusanlar, bağırıp çağıranlar, panikleyenler…Açıkçası benim de çok hoşuma giden bir durum değil di*(Zaten deniz otobüsüne bindiğinizde kaptanınız "Açıkta Deniz vardır!Rahatsız olanların binmemesi önemle rica olunur!" derse, ve bunu sinirli bir şekilde söylerse durum vahimdir)*Derken koltuğunda dimdik oturup, sağ sola bakmadan ince bir kitap okuyan bir yolcu ile göz göze geldim. Bana **"Sürükleyici bir kitap…Fırtınayı unutturuyor…"** dedi. Bu sözden etkilenmiştim. Ne yaptım dersiniz? Laptop' umu açtım ve **8/28/2005** tarihinde C#Nedir? üzerinden yayınlanan Numaralandırıcıları Kullanmak İçin Bir Sebep başlıklı makalemi yazdım.

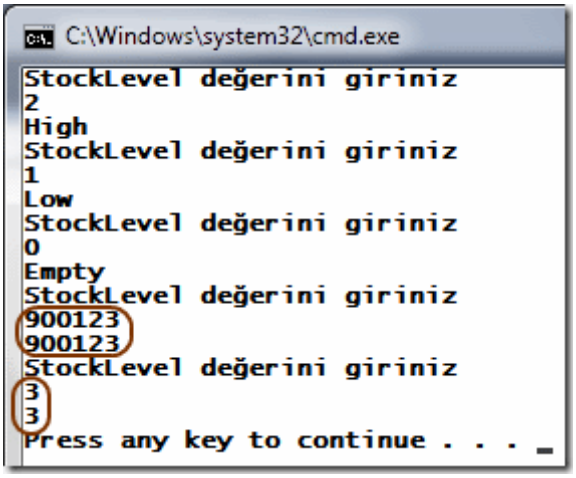
Bu makalemde Asp.Net tabanlı bir web uygulamasında **DataGrid** ve **Enum** sabitlerinin kullanımına değinmekteydim. Bu aslında, **.Net Framework**' ün 5 temel tipinden birisi olan **Enum** sabitleri ile*(Diğerleri Class, Struct, Interface, Delegate tipleridir)* belki de en önemli haşır neşir oluşumdu. Derken bir sene sonrasında bu kez **Netron** firmasında eğitmen olarak görev yapmaktayken yine**C#Nedir?** topluluğunda, **10/30/2006** tarihli C# Temelleri : Enum Sabitinin Bilinmeyen Yönleri isimli makalemi yayınladım. Bu sefer Enum sabitlerini daha detaylı inceliyor ve özellikle Enum tipi üzerinden erişilen üyeler yardımıyla nasıl işlemler yapılabileceğini ele alıyordum.

Sene oldu **2010**. Hatta **2010** yılının ikinci yarısının başındayız. **Enum** sabitleri ile uğraşmayalı da hayli zaman olmuş aslında. İşte bu yazımızda **Enum** sabitlerinin başımıza

dert olabileceği bir kaç vakayı ele almaya çalışıyor olacağız. öyleyse hızlı bir başlangıç yapalım ve aşağıdaki kod parçasını göz önüne alalım.

```
using System;

namespace HelloAgainEnums
{
   class Program
   {
      static void Main(string[] args)
      {
         for (int i = 0; i < 5; i++)
         {
            Test();
         }
      }

      private static void Test()
      {
         Console.WriteLine("StockLevel değerini giriniz");
         int userInput = Int32.Parse(Console.ReadLine());
         StockLevel sLevel = (StockLevel)userInput;
         Console.WriteLine(sLevel);
      }
   }

   enum StockLevel
   {
      Empty,
      Low,
      High
   }
}
```

örnek kod parçasında **StockLevel** isimli bir **enum** sabiti kullanılmaktadır. Bu sabitin **Empty**, **Low** ve **High** isimli 3 değeri bulunmaktadır. Bilindiği üzere **enum** sabiti değerleri varsayılan olarak **0**sayısalından başlamaktadır. Kodun kritik olan kısmı **Test** metodunun içeriğidir. Dikkat edileceği üzere kullanıcıdan sayısal bir değer girilmesi istenmektedir. Bu değer **userInput** isimli**int** tipinden değişken içerisine alınmaktadır. önemli olan nokta **userInput** değişkeninin **bilinçli(Explicitly)** olarak **StockLevel** tipine dönüştürülmesidir. Bu mümkündür.**Derleme(Compile)** veya **çalışma Zamanında(Runtime)** herhangibir hata oluşmayacaktır. Mı acaba? Gelin örnek bir çalışma zamanı çıktısına bakalım.



İlk üç denemede **2, 1** ve **0** değerleri girilmiştir. Ancak son iki denemede aslında **enum** sabiti içerisinde olmayan değerlerin girildiği görülmektedir. Kod hata vermemiştir. Dönüştürme işlemi başarılı olmuştur. Ancak kavramsal olarak bir hata vardır. **Enum sabitine ait olmayan** bir takım değerler ile çalışılmaktadır. çok tabi olarak **enum** sabitlerini kullanan pek çok geliştirici bu detayı göz ardı etmiş olabilir. Peki ya çözüm nedir?. **Test** metodunu aşağıdaki gibi değiştirdiğimizi düşünelim.

```
private static void Test()
        {
            Console.WriteLine("StockLevel değerini giriniz");
            int userInput = Int32.Parse(Console.ReadLine());
            if (Enum.IsDefined(typeof(StockLevel), userInput))
            {
                StockLevel sLevel = (StockLevel)userInput;
                Console.WriteLine(sLevel);
            }
            else
            {
                Console.WriteLine("Girilen sayısal değer StockLevel enum sabiti içerisinde yer almamaktadır.");
            }
        }
```

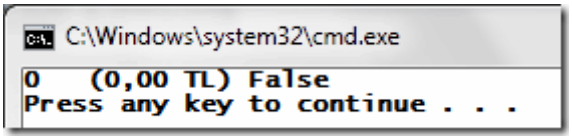
Enum sınıfı üzerinden erişilebilen **IsDefined** metodu ilk parametre olarak **enum sabiti** tipini almaktadır. Bu nedenle **typeof** operatörü kullanılmıştır. İkinci parametre **object** tipindendir ve kullanıcının girdiği **int** değeri taşımaktadır. **Metod**, **StockLevel enum sabiti içerisinde userInput ile girilen değerin var olup olmadığına bakmakta** ve buna göre **true** veya **false**değer döndürmektedir. Tabiki if kontrolü içerisinde bilinçli olarak dönüştürme işlemi yapılması şart değildir. **Enum** sınıfının **Parse** metodu kullanılabilir. Hatta **TryParse**metodundan yararlanılarak, dönüştürme başarılı olduğu takdirde değer atanması yapılması da sağlanabilir.

Aslında **Enum** sınıfının **Parse** ve **TryParse** kullanımları ile **IsDefined** çağrımına gereksinimin ortadan kalkabileceğini ifade edebilir miyiz? Bunu bir düşünün ve araştırın.

Gelelim diğer bir vakaya. Normal şartlarda **Enum** sabitleri içerisine yazılan değerler için sayısal atamada bulunulmaz. Varsayılan değerler ne ise bunlar kullanılmaktadır. Buna göre ilk sabit değeri **0** ile başlar ve diğerleri de otomatik olarak artarak devam eder. Ne varki **0** değerinin kullanımı ile ilişkili bir durum vardır. Aşağıdaki kod örneğini göz önüne alalım.

```
using System;

namespace HelloAgainEnums
{
   class Program
   {
      static void Main(string[] args)
      {
         Product prd = new Product();
         Console.WriteLine(prd.ToString());
      }
   }

   class Product
   {
      public int ProductId { get; set; }
      public string Name { get; set; }
      public decimal ListPrice { get; set; }
      public bool InStock { get; set; }

      public override string ToString()
      {
         return String.Format("{0} {1}  ({2}) {3}", ProductId, Name,
ListPrice.ToString("C2"),InStock);
      }
   }
}
```

Bu kod parçasında **Product** sınıfına ait bir nesne örneğinin varsayılan **yapıcı metod(Default Constructor)** ile üretildiği ve **ezilmiş(Override) ToString** fonksiyonu ile de içeriğinin ekrana yazdırıldığı görülmektedir. Temelleri hatırlayacak olursak, bu tip bir durumda sınıf üyelerine varsayılan değerler atanacaktır. Sayısal değerler için **0**, kayan noktalı sayısal değerler için **0.0**, **bool** değerler için **false**, **referans tipleri** içinse **null**. Dolayısıyla çalışma zamanı çıktısı aşağıdaki gibi olacaktır.

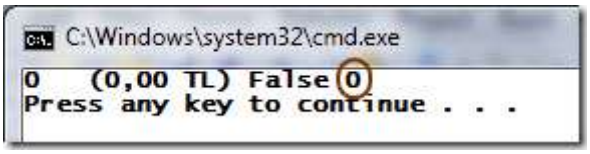


Peki ya aşağıdaki gibi bir kod söz konusu olduğunda.

```
using System;

namespace HelloAgainEnums
{
   class Program
   {
      static void Main(string[] args)
      {
         Product prd = new Product();
         Console.WriteLine(prd.ToString());
      }
   }

   class Product
   {
      public int ProductId { get; set; }
      public string Name { get; set; }
      public decimal ListPrice { get; set; }
      public bool InStock { get; set; }
      public StockLevel Level { get; set; }

      public override string ToString()
      {
         return String.Format("{0} {1}  ({2}) {3} {4}", ProductId, Name,
ListPrice.ToString("C2"),InStock,Level);
      }
   }

   enum StockLevel
   {
      Empty=10,
      Low=12,
      High=18
   }
}
```

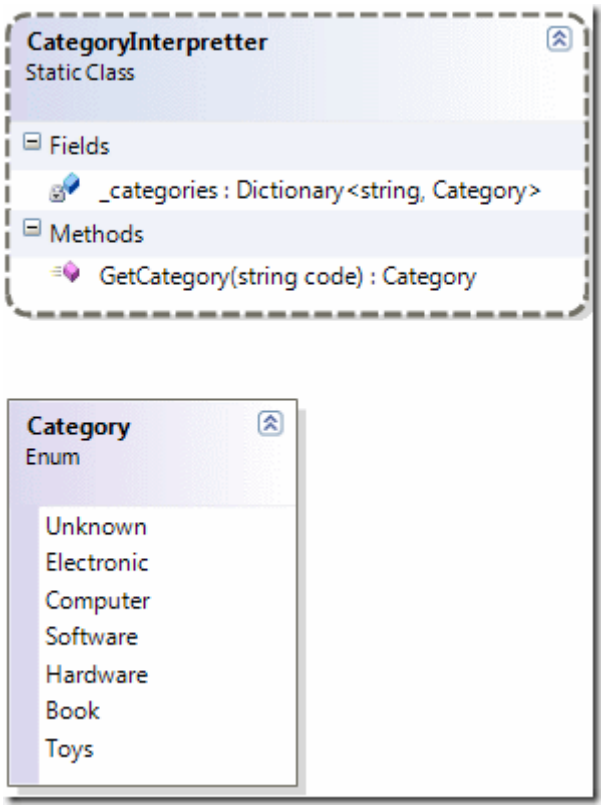
Bu sefer **StockLevel** isimli **Enum** sabitinde varsayılan değerler kullanılmamıştır. Bunun yerine **10, 12** ve **18** değerleri verilmiştir. Diğer yandan **Product** isimli sınıfta **StockLevel** tipinden **Levelözelliğinin(Property)** bulunduğu gözden kaçmamalıdır.

Az önceki kod parçasında sınıfların üyelerinin, aksine bir işlem yapılmadıkça varsayılan değerlere atandığını görmüştük. Bu durumda **Level** özelliğinin değeri aşağıdaki ekran çıktısındaki gibi olacaktır.



İşte istemediğimiz bir durum. Bu nedenle 0 varsayılan değerinin enum sabitleri içerisinde **mutlaka kullanılması gerektiğini** ifade edebiliriz. Tabi çözümlerden birisi, **enum** içerisinde **Unknown**gibi **0** değerine set edilmiş bir sabit kullanmaktır.

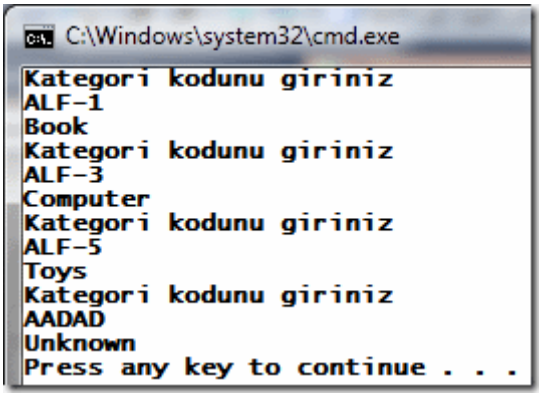
Yazımızda Enum sabitleri ile alakalı iki önemli vakayı incelemeye çalıştık. Şimdi ilk vakamıza tekrar dönüş yapıyor olacağız ve konuyu farklı bir açıdan değerlendirmeye çalışacağız. **Enum** sabitleri ile **int** tipleri arasındaki dönüşümlerde **cast** operatörlerinin kullanılması aslında birbirlerine kuvvetle bağlı yapılar oldukları için sorunlara neden olabilmektedir. Söz gelimi **enum** sabitlerinin **int** karşılıklarının bir veri kaynağında tutulduğu durumlarda, enum sabiti içerisinde veya veri kaynağındaki sayısal değerlerde yapılacak değişikliker, istenmeyen sonuçlara neden olabilmektedir. Bu nedenle belki de **enum** sabitleri ile **int** değerler arasındaki dönüştürmelerde farklı bir çözüm yoluna gidilmelidir. Hatta olayı sadece int ile enum sabitleri arasında bir dönüştürme olarak düşünmeyebiliriz de. Söz gelimi sayı ve harflerden oluşan **string** bir ifadenin de, bir **enum** sabiti değerine karşılık gelmesini isteyebiliriz. Durumu daha net bir şekilde kavrayabilmek adına dilerseniz aşağıdaki kod içeriğini bir göz önüne alalım.



```
using System;
using System.Collections.Generic;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         for (int i = 0; i < 4; i++)
         {
            Test();
         }
      }
      static void Test()
      {
         Console.WriteLine("Kategori kodunu giriniz");
         string categoryCode = Console.ReadLine();
         Category ctgry = CategoryInterpretter.GetCategory(categoryCode);
         Console.WriteLine(ctgry.ToString());
      }
   }

   public static class CategoryInterpretter
   {
      private static readonly Dictionary<string, Category> _categories =new
Dictionary<string,Category>
      {
         {"ALF-1",Category.Book},
         {"ALF-2",Category.Electronic},
         {"ALF-3",Category.Computer},
         {"ALF-4",Category.Software},
         {"ALF-5",Category.Toys},
         {"ALF-6",Category.Hardware}
      };
      public static Category GetCategory(string code)
      {
         Category result;
         if (_categories.TryGetValue(code, out result))
            result = _categories[code];
         else
            result = Category.Unknown;
         return result;
      }
   }
```

```
    public enum Category
    {
        Unknown,
        Electronic,
        Computer,
        Software,
        Hardware,
        Book,
        Toys,
    }
}
```

Şimdi örnek uygulamamızda neler yaptığımıza bir bakalım. **CategoryInterpretter** isimli **static** sınıf içerisinde değişmez olarak tanımlanmış **private** erişim belirleyicisine sahip bir**Dictionary<string,Category>** koleksiyonu bulunmaktadır. Bu koleksiyonunun **key** verileri **string** olup, **Category enum** tipi içerisindeki birer sabite karşılık gelmektedir. Bunun dışında **GetCategory**isimli **static** metod, parametre olarak gelen **string** bilginin karşılığını **enum** sabiti içerisinden getirmek üzere oluşturulmuştur. Bu sayede **ALF-1=Category.Book** gibi bir eşleştirme yapılabilmesi kavramsal olarak mümkündür. Uygulamanın çalışma zamanı çıktısına baktığımızda aşağıdaki örnek sonuçlar ile karşılaşırız.



Görüldüğü üzere ilk 3 deneme de başarılı çevirmeler yapılmıştır. Son denemede girilen **string** bilgi(AADAD), **Dictionary<string,Category>** tipi içerisinde var olmadığından, varsayılan olarak **Category.Unknown** değeri elde edilmiştir.
Bu **GetCategory** metodu içerisinde alınmış bir tedbirdir.
Aslında **TryGetValue** metodunun kullanılmaması halinde çalışma zamanına **KeyNotFoundException** tipinden bir istisna fırlatılacaktır ki bu da hatalı veri girişlerine karşılık bir çözüm olarak düşünülebilir.

```
C:\Windows\system32\cmd.exe
Kategori kodunu giriniz
sfsdf

Unhandled Exception: System.Collections.Generic.KeyNotFoundException: The given
key was not present in the dictionary.
   at System.Collections.Generic.Dictionary`2.get_Item(TKey key)
   at ClientApp.CategoryInterpretter.GetCategory(String code) in D:\Vs 2010\RTM\
C# Fundamentals\HelloAgainEnums\ClientApp\Program.cs:line 40
   at ClientApp.Program.Test() in D:\Vs 2010\RTM\C# Fundamentals\HelloAgainEnums
\ClientApp\Program.cs:line 20
   at ClientApp.Program.Main(String[] args) in D:\Vs 2010\RTM\C# Fundamentals\He
lloAgainEnums\ClientApp\Program.cs:line 12
Press any key to continue . . .
```

Böylece geldik bir yazımızın daha sonuna. Bu yazımızda yıllar sonrasında Enum sabiti kavramına tekrardan dönüş yaparak farklı açılardan ele almaya çalıştık. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HelloAgainEnums.rar (44,39 kb)

C# 4.0 Default Parameter Kullanımına Dikkat! (2010-12-19T01:00:00)

c# 4.0,c# temelleri,default parameters,optional parameters,named parameters,

Merhaba Arkadaşlar,

2004 ve 2005 yıllarında uzun bir süre editörlüğünü yaptığım C#Nedir? topluluğunun düzenlediği **C# Akademi** eğitimlerinde, yarı zamanlı eğitmen olarak görev yapmıştım. Genellikle **C#** programlama dilinin basit ve temel konularını, ayrıca **Object Oriented** özelliklerini aktarmaya çalışırdım. Elbette sınıfımdaki öğrencilerim yanda görüldüğü gibi her zaman pür neşe olmazlardı.

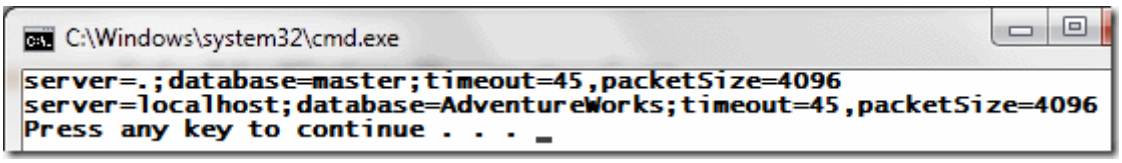
Ancak insan zaman içerisinde profesyonelleşme yolunda ilerledikçe konuları çok daha farklı açılardan ele alması gerektiğini de öğreniyor. Profesyonel bir eğitmenin en iyi yaptığı işlerin başında, en zor konuları çöp adam kullanarak anlatmak gelmektedir. Tabi eğitmenin gerçek hayat tecrübelerini ve ip uçlarını da aktarıyor olması, profesyonelliğinin diğer bir göstergesidir. Böyle bir eğitmenin vereceği önerileri pür dikkat dinlemekte yarar vardır.

Ben eğitmenliği bırakalı uzun bir süre oldu ama makale yazarken veya görsel ders çekerken, konunun anlatımı sırasında yukarıdaki hususlara dikkat etmeye çalışıyorum. Bu anlamda bazen çok basit olarak görünen bir konunun, aslında derinlere inildiğinde dikkat edilmesi gereken noktalar içerdiğini sürekli vurgulamaya çalışan yazıları da hazırlama uğraşısı içerisindeyim. İşte bu yazımızın konusu da; **C# 4.0** ile birlikte gelen yeni dil

özelliklerden birisi olan **Default Parameters** ile ilişkili tuzaklar. öncelikli olarak konuya aşağıdaki hazır kod parçası ile başlayalım.

```
using System;

namespace DefaultAndOptionalParametersCase
{
   class Program
   {
      static void Main(string[] args)
      {
         Connection myConn = new Connection();
         Console.WriteLine(myConn.ToString());
         myConn = new Connection("localhost", "AdventureWorks");
         Console.WriteLine(myConn.ToString());
      }
   }

   class Connection
   {
      public string Server { get; set; }
      public string Database { get; set; }
      public int Timeout { get; set; }
      public int PacketSize { get; set; }

      #region Constructors

      public Connection(string server,string databaseName,int timeout,int packetSize)
      {
         Server = server;
         Database = databaseName;
         Timeout = timeout;
         PacketSize = packetSize;
      }
      public Connection(string server, string databaseName, int timeout)
         : this(server, databaseName, timeout, 4096)
      {
      }
      public Connection(string server, string databaseName)
         : this(server, databaseName, 45, 4096)
      {
      }
      public Connection()
         : this(".", "master", 45, 4096)
```

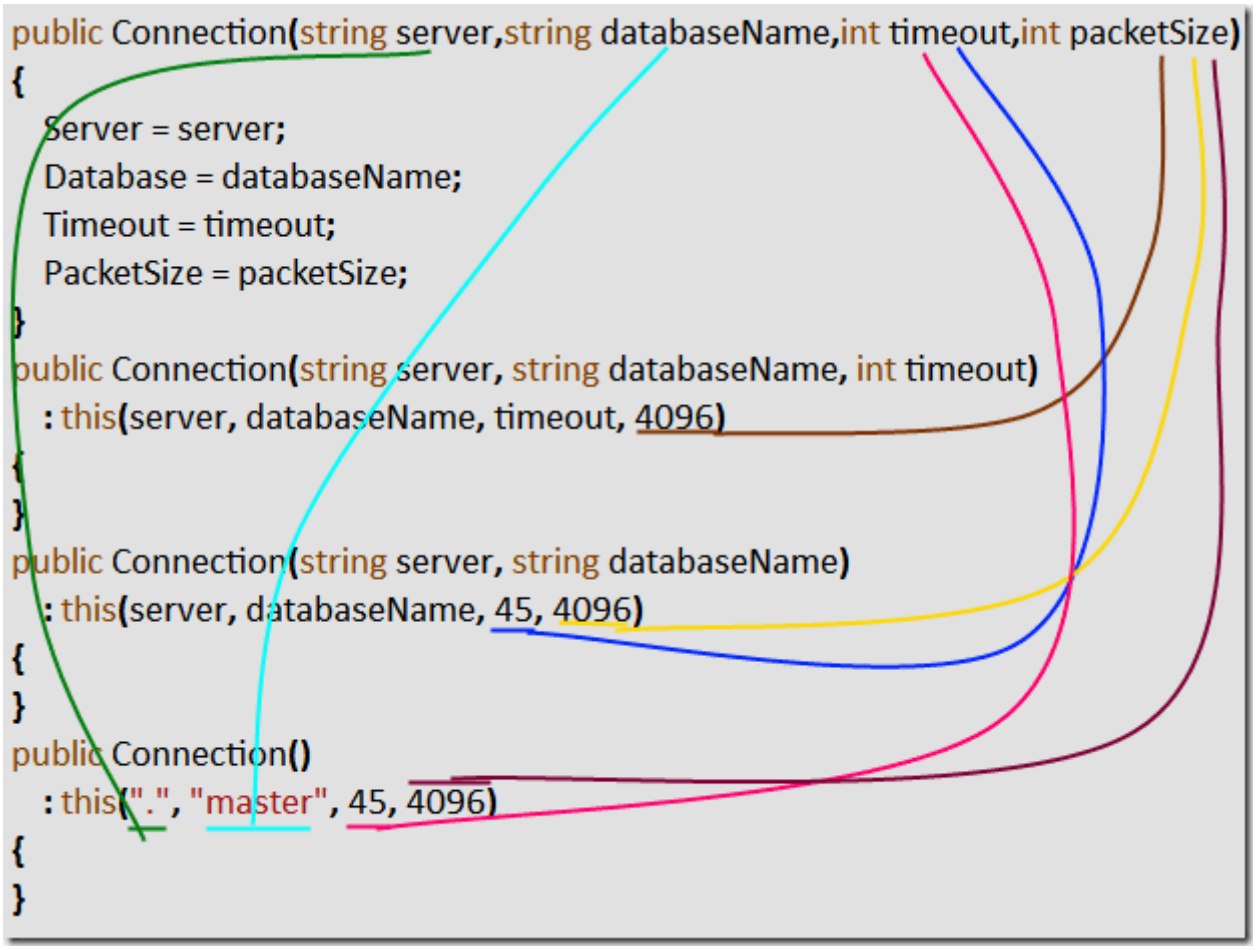
```
        {
        }

        #endregion

        public override string ToString()
        {
            return String.Format("server={0};database={1};timeout={2},packetSize={3}",
Server, Database, Timeout, PacketSize);
        }
    }
}
```



Bu kod parçasında dikkat etmemiz gereken nokta **Constructor** metodlarıdır. Görüldüğü üzere en fazla sayıda parametre alan yapıcı metod, diğer yapıcı metodlar tarafından kullanılmaktadır. Burada **this** anahtar kelimesini takip eden ifadeler içerisinde gerekli aktarma işlemlerinin yapıldığı görülebilir.

Eski bilgilerimizi bir hatırlayalım. Bilindiği üzere **yapıcı metodlarda(Constructors) this** yerine **base** anahtar kelimesini kullanarak, metod parametrelerinin bir üst sınıftaki versiyonuna gönderilmesi de sağlanabilir.

Tabi burada **C# 4.0** ile gelen **Default Parameters** yeteneğinin devreye girmesi ile **n sayıda metod yerine tek bir metodun kullanılması** söz konusu olabilir. Nitekim ele aldığımız örnek senaryoda yapıcı metodların tek yaptığı, uygun olan versiyona parametre değerlerini taşımaktır. Dikkat edileceği üzere sadece tek bir yapıcı metod içerisinde özellik değer atama işlemleri yapılmaktadır. Diğer yapıcı metodlar sadece parametre değerlerini taşımak için kullanılmaktadır. Aşağıdaki şekilde bu durum ifade edilmeye çalışılmaktadır.

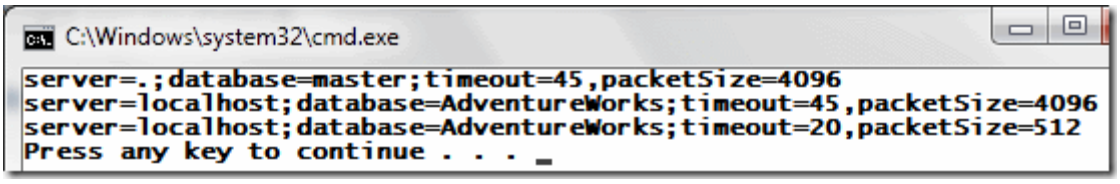


Aslında **Constructor** kullanımının buradaki amacı, **Connection** tipine ait nesne örneklerinin oluşturulması sırasında alternatif versiyonları varsayılan parametre değerlerine göre sunabilmektir. Bu amaç düşünüldüğünde **Default Parameters** yeteneği önemli bir avantaj sağlamaktadır. Gelin kodumuzu **Default Parameters** kabiliyetini kullanarak aşağıdaki hale getirelim.

```
using System;

namespace DefaultAndOptionalParametersCase
{
   class Program
   {
      static void Main(string[] args)
      {
         Connection myConn = new Connection();
         Console.WriteLine(myConn.ToString());
         myConn = new Connection("localhost", "AdventureWorks");
         Console.WriteLine(myConn.ToString());
         myConn = new Connection("localhost", "AdventureWorks",20,512);
         Console.WriteLine(myConn.ToString());
```

```
        }
    }

    class Connection
    {
        public string Server { get; set; }
        public string Database { get; set; }
        public int Timeout { get; set; }
        public int PacketSize { get; set; }

        public Connection(string server=".", string databaseName="master", int
timeout=45, int packetSize=4096)
        {
            Server = server;
            Database = databaseName;
            Timeout = timeout;
            PacketSize = packetSize;
        }

        public override string ToString()
        {
            return String.Format("server={0};database={1};timeout={2},packetSize={3}",
Server, Database, Timeout, PacketSize);
        }
    }
}
```

Dikkat edileceği üzere **tek bir yapıcı metod** kullanımı söz konusudur. Bir başka deyişle kod kısalmıştır. Yapıcı metodun parametrelerinde verilen varsayılan değerler sayesinde,**Connection** tipine ait nesne örneklerinin oluşturulması şekillendirilmiştir. örneğin, çalışma zamanı çıktısı aşağıdaki gibi olacaktır.



Aslında işin içerisine **Named Parameters** kullanımını da katmamız yerinde olacaktır. Neden? **Main** metodu içerisindeki aşağıdaki kod satırını göz önüne alalım.

myConn = new Connection("localhost", "AdventureWorks",**20,512**);

Geliştirici kodu yazarken parametrelerin ne anlama geldiğini, isimlerinden veya varsa eğer **XML Comment'** lerden çıkartabilir. Ancak tamamlanmış kodun okunması

sırasında **20** ve **512** rakamlarının en anlama geldiği kolayca anlaşılamayabilir. İşte bu noktada parametreleri isimlendirerek kullanmak aşağıdaki okunurluğu sağlayacaktır.

myConn = new
Connection(**server:**"localhost", **databaseName:**"AdventureWorks", **timeout:**20, **packetSize:**512);

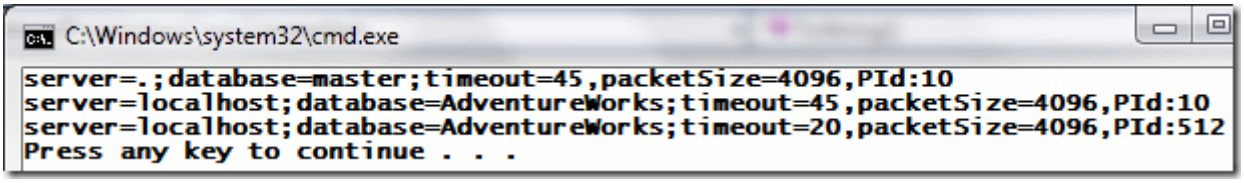
Parametre Sayısının Arttırılması

Gelelim **default parameters** kullanımında dikkatli olmamız gereken hususlara. İlk olarak parametre sayısının arttırılması durumunu göz önüne alacağız. Ancak senaryonun oluşumunda **Named Parameters** kullanmadığımızı varsayıyoruz. Bu amaçla **Connection** tipine ait yapıcı metodu aşağıdaki gibi değiştirdiğimizi düşünelim.

```
using System;

namespace DefaultAndOptionalParametersCase
{
   class Program
   {
      static void Main(string[] args)
      {
         Connection myConn = new Connection();
         Console.WriteLine(myConn.ToString());
         myConn = new Connection("localhost", "AdventureWorks");
         Console.WriteLine(myConn.ToString());
         myConn = new Connection("localhost", "AdventureWorks",20,512);
         Console.WriteLine(myConn.ToString());
      }
   }

   class Connection
   {
      public string Server { get; set; }
      public string Database { get; set; }
      public int Timeout { get; set; }
      public int PacketSize { get; set; }
      public int ProcessId { get; set; }

      public Connection(string server=".", string databaseName="master", int
timeout=45,int processId=10, int packetSize=4096)
      {
         Server = server;
         Database = databaseName;
         Timeout = timeout;
```

```
            PacketSize = packetSize;
            ProcessId = processId;
        }

        public override string ToString()
        {
            return
String.Format("server={0};database={1};timeout={2},packetSize={3},PId:{4}", Server,
Database, Timeout, PacketSize,ProcessId);
        }
    }
}
```

Kodda sadece **processId** isimli bir metod parametresi eklendiğini görmekteyiz. Bu aslında sonradan yapılan bir değişiklik olarak düşünülmelidir. Bir başka deyişle geliştirdiğimiz projelerde sonradan varsayılan parametre eklenmesi söz konusu olabilir. Buna göre çalışma zamanı çıktısı aşağıdaki gibi olacaktır.



Dikkatinizi çeken bir nokta var mı?

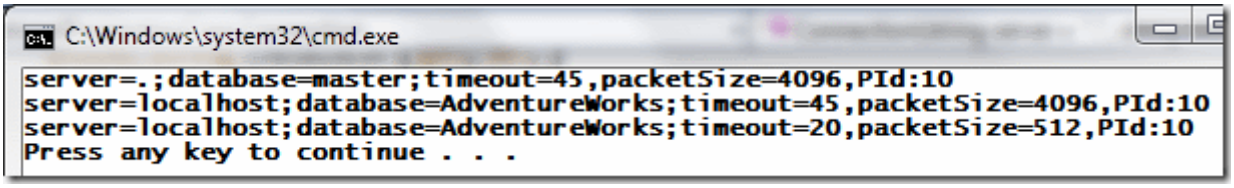
Son çıktıya göre **ProcessId** değerinin **512** olduğu görülmektedir. Oysaki **512** değeri daha önceki kodlamaya göre **PacketSize** özelliği için atanan bir değerdir. Bir başka deyişle yanlış bir değer ataması söz konusudur. İşin kötü yanı bu senaryoda derleme zamanında bir hata veya uyarı mesajı alınmamaktadır. Dolayısıyla kodun hatalı çalışması olasıdır.

öyleyse varsayılan parametre kullanımı gibi senaryolarda, metodlara yeni parametrelerin eklenmesi söz konusu ise, bu parametrelerin en sona eklenmesi daha doğru olacaktır. **Named Parameters**aslında köklü çözüm olsa da, ilgili tip metodlarını kullanan diğer geliştiricilerin bu kullanımı göz ardı etmesi ihtimali vardır.

Yani metod yapısını aşağıdaki gibi değiştirmemiz doğru bir çalışma zamanı çıktısı elde etmemizi sağlayacaktır.

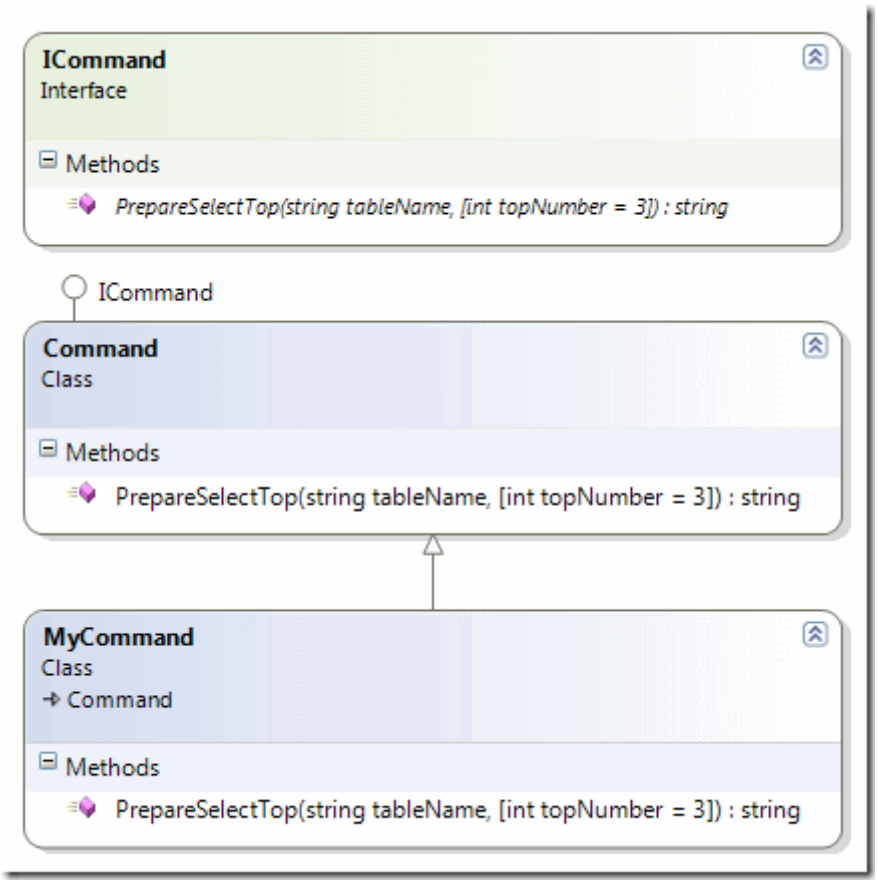
public Connection(string server = ".", string databaseName = "master", int timeout = 45, int packetSize = 4096**,int processId = 10**)

,sonucu çalışma zamanı çıktısı aşağıdaki gibidir.



Türetme(Inheritance) ve Varsayılan Parametreler

Gelelim diğer bir vakaya. Bu vaka çok daha kritik ve önemlidir. Nitekim işin içerisinde **türetme(Inheritance)** kavramı vardır. Konuyu netleştirmek için aşağıdaki sınıf şemasına sahip örnek kod parçasını göz önüne alarak ilerleyelim.




```
using System;

namespace DefaultAndOptionalParametersCase
{
   class Program
   {
      static void Main(string[] args)
      {
         MyCommand myCmd = new MyCommand();
         ICommand iCmd = myCmd;
         Command cmd = myCmd;
```

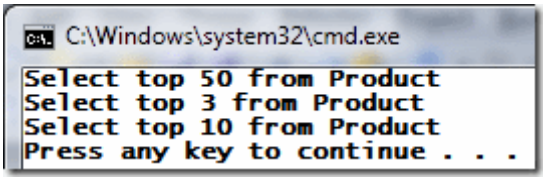
```
            Console.WriteLine(myCmd.PrepareSelectTop("Product"));
            Console.WriteLine(iCmd.PrepareSelectTop("Product"));
            Console.WriteLine(cmd.PrepareSelectTop("Product"));
        }
    }

    interface ICommand
    {
        string PrepareSelectTop(string tableName, int topNumber = 3);
    }
    class Command
      : ICommand
    {
        #region ICommand Members

        public virtual string PrepareSelectTop(string tableName, int topNumber = 10)
        {
            return String.Format("Select top {0} from {1}",topNumber,tableName);
        }

        #endregion
    }
    class MyCommand
      : Command
    {
        public override string PrepareSelectTop(string tableName, int topNumber = 50)
        {
            return String.Format("Select top {0} from {1}",topNumber,tableName);
        }
    }
}
```

Aslında bu senaryo Temeller Kolay Unutulur (C# – Implicitly Name Hiding Sorunsalı) başlıklı yazımızdan size tanıdık gelecektir.

Sınıf şemasından da görüleceği üzere **ICommand arayüzünü(Interface)** uygulayan **Command** isimli bir tip ve bundan türeyen **MyCommand** sınıfı söz konusudur. **MyCommand**sınıfı, **Command** tipinde **virtual** olarak tanımlanmış ve aslında **ICommand** arayüzü tarafından zorunlu hale getirilmiş **PrepareSelectTop** metodunu ezmektedir**(Overriding)**.

Kritik olan yer **Main** metodu içerisindeki değişken atamalardır. Dikkat edileceği üzere **ICommand** ve **Command** tipinden olan değişkenlere aynı **MyCommand** nesne örneği atanmıştır. Eğer çok biçimlilik ilkesini biliyorsak, **iCmd** ve **cmd** isimli nesne

örnekleri üzerinden yapılan **PrepareSelectTop** çağrılarının
aslında **MyCommand** tipindeki metod içeriğine doğru yapılması gerektiğini biliriz. Buna göre de tüm **Select** sorgularında **Top 50** değerinin kullanılıyor olması gerekmektedir. Oysaki çalışma zamanı çıktısı aşağıdaki gibi olacaktır.



Görüldüğü gibi son iki çağrıda **topNumber** için **Default Parameter** değerleri tanımlandıkları yerdekiler olmuştur. **ICommand** için **3** iken **Command** için **10** olarak ele alınmıştır. Tam bu noktada **"Amanın! Yoksa ICommand ve Command tipleri çok biçimlilik göstermiyorlarmış!"** diye haykırabilirsiniz. Ama dereyi görmeden paçaları sıvamamak lazım. Nitekim uygulamayı **debug** modda değerlendirdiğimizde, aslında tüm **PrepareSelectTop** çağrılarının, **MyCommand** içinden yapıldığı görülecektir.

Sorun tamamen **Default Parameter'** lar ile alakalıdır. Söz gelimi **ICommand** üzerinden yapılan çağrı sonucu **topNumber** değeri aşağıdaki gibi olacaktır.

```
51  class MyCommand
52    : Command
53  {
54    public override string PrepareSelectTop(string tableName
55      , int topNumber = 50)
56    {                    topNumber 3
57      return String.Format("Select top {0} from {1}",topNumber,tableName);
58    }
59  }
```

veya **Command** tipi için şu şekilde olacaktır.

```
51  class MyCommand
52    : Command
53  {
54    public override string PrepareSelectTop(string tableName
55      , int topNumber = 50)
56    {                    topNumber 10
57      return String.Format("Select top {0} from {1}",topNumber,tableName);
58    }
59  }
```

Böyle bir vakanın oluşmasının sebebi **Defaul Parameter**'ların **çalışma zamanı(Runtime)** yerine **derleme zamanında(Compile Time)** çözümleniyor olmalarıdır. Bu durum IL(Intermediate Language) kodunda açık bir şekilde görülebilir ve ispatlanabilir.

```
.method private hidebysig static void  Main(string[] args) cil managed
{
  .entrypoint
  // Code size       68 (0x44)
  .maxstack  3
  .locals init ([0] class DefaultAndOptionalParametersCase.MyCommand myCmd,
           [1] class DefaultAndOptionalParametersCase.ICommand iCmd,
           [2] class DefaultAndOptionalParametersCase.Command cmd)
  IL_0000:  nop
  IL_0001:  newobj     instance void
DefaultAndOptionalParametersCase.MyCommand::.ctor()
  IL_0006:  stloc.0
  IL_0007:  ldloc.0
  IL_0008:  stloc.1
  IL_0009:  ldloc.0
  IL_000a:  stloc.2
  IL_000b:  ldloc.0
  IL_000c:  ldstr      "Product"
  IL_0011:  ldc.i4.s   50
  IL_0013:  callvirt   instance string
DefaultAndOptionalParametersCase.Command::PrepareSelectTop(string,
                                                           int32)
  IL_0018:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_001d:  nop
  IL_001e:  ldloc.1
  IL_001f:  ldstr      "Product"
  IL_0024:  ldc.i4.3
  IL_0025:  callvirt   instance string
DefaultAndOptionalParametersCase.ICommand::PrepareSelectTop(string,
                                                            int32)
  IL_002a:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_002f:  nop
  IL_0030:  ldloc.2
  IL_0031:  ldstr      "Product"
  IL_0036:  ldc.i4.s   10
  IL_0038:  callvirt   instance string
DefaultAndOptionalParametersCase.Command::PrepareSelectTop(string,
                                                           int32)
  IL_003d:  call       void [mscorlib]System.Console::WriteLine(string)
  IL_0042:  nop
```

```
  IL_0043:  ret
} // end of method Program::Main
```

IL kodunda yer alan **ldc** komutlaraına bakıldığında **Defualt Parameter** değerlerinin, tip tanımlamaları sırasında yazıldığı gibi **set** edildiği açık bir şekilde görülebilmektedir.

Kolayca gözden kaçabilecek bir durum olduğu için tehlikeli bir vaka olduğunu ifade edebiliriz. Dolayısıyla en azından bu senaryoya göre **Default Parameter** kullanımını aslında **Interface**seviyesinde bırakmak çözüm olarak düşünülebilir.

Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

DefaultAndOptionalParametersCase.rar (25,52 kb)

Temeller Kolay Unutulur (C# - Implicitly Name Hiding Sorunsalı) (2010-12-19T00:50:00)

c#,object oriented programming,oop,c# temelleri,polymorhysm,

Merhaba Arkadaşlar,

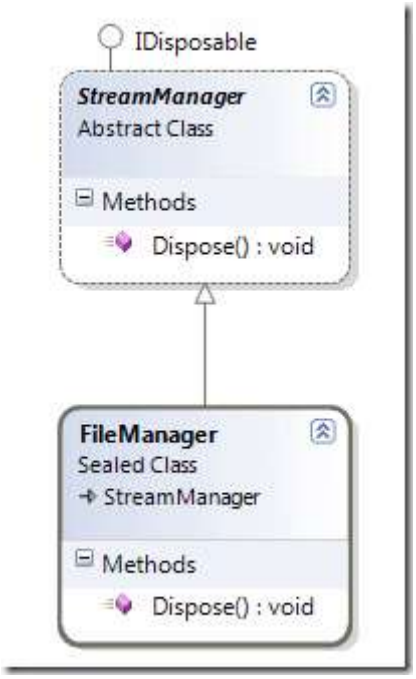
Sizde benim gibi basketol tutkunu musunuz? Aslında ülkemizde hemen herkesin birincil olarak futbol merakı olması beklenir. Oysaki bende diğer pek çok arkadaşım gibi birincil olarak basketbol' a meraklıyımdır. Aslında lise yıllarında sevgili **Michael Jordan** ve **Chicago Bulls** ile başlayan bu merakım sonrasında **Efes Pilsen, ülker, Tofaş** gibi takımlarla daha da artmıştır. üniversite yıllarından beri bu takımların pek çok maçına gitmişimdir ve halen daha gitmeye çalışmaktayım*(Tabi bir dönem Tofaş profesyonel basketbol şubesini kapatmıştı…)* Tabi şu anda 11 aylık S(h)arp Efe buna pek müsade etmiyor. Ama yine de takım ayırt etmeksizin özellikle avrupa arenasındaki pek çok maça gitmeye çalışıyorum.

Basketbol denilince aklıma gelen en önemli şahsiyetler arasında ise değerli spiker **Murat Murathanoğlu** ve değerli yorumcu **İsmet Badem** ikilisi gelmektedir. Her ne kadar uzun bir süre önce yollarını ayırmış olsalar da, özellikle **İsmet Badem**' in hemen her maçta gençlere verdiği basketbol ip uçları halen kullaklarımdadır. özellikle basketbolun **temellerinin(Fundamentals)** çok önemli olduğunu genç basketbolcu adaylara sürekli ifade etmiştir, etmektedir.

Aslında bakarsanız temeller bir programlama dili için de son derece önemlidir. Profesyonel geliştiriciler, yazdıkları uygulamalarının çeşitliliği ve kullanılan araçlar düşünüldüğünde zaman içerisinde programlamanın temel kavramlarını kolayca unutabilir. Temellerin yer yer tekrar edilmemesi veya zaman içerisinde geriye dönüp bakılmaması bunun en büyük nedenlerindedir.

özellikle **C#, Java** gibi **nesne yönelimli(Object Oriented)** programlama dillerinin temelleri son derece önemlidir ve bu temeller bir süre sonra profesyonel bir geliştirici için artık bisiklet sürmek gibi unutlmayacak unsurlara dönüşmelidir. İşte bu yazımızda kolayca unutulabilen **bilinçsiz üye gizleme(Implicitly Name Hiding)** ile alakalı bir vakayı ele almaya çalışıyor olacağız.

Dilerseniz hiç vakit kaybetmeden sorunu örnek uygulama üzerinden masaya yatırarak ilereleyelim. Bu amaçla aşağıdaki **sınıf çizelgesindeki(Class Diagram)** tiplerin kullanıldığı bir Console uygulamasını geliştirdiğimizi düşünelim.




```
using System;

namespace ImplicitHiding
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Case 1

         // Bu vakaya göre her Dipose çağrısı için FileManager tipine ait Dispose metodu içeriğinin devreye gireceği düşünülmektedir.
```

```
            FileManager fm = new FileManager();
            StreamManager sm = fm;
            IDisposable dm = fm;

            // FileManager örneğine ait Dispose metodunu çağırmak için 3 yol olduğunu
düşünebiliriz.
            // kendisi üzerinden
            fm.Dispose();
            // türediği base class üzerinden
            sm.Dispose();
            // türediği base class' ın implement ettiği Interface tipi üzerinden
            dm.Dispose();

            #endregion
        }
    }

    abstract class StreamManager
        :IDisposable
    {
        #region IDisposable Members

        public void Dispose()
        {
            Console.WriteLine("Stream Manager için Dispose çağrısı\n");
        }

        #endregion
    }

    sealed class FileManager
        : StreamManager
    {
        // Buradaki Warning mesajı geliştirici tarafından göz ardı edilmiş olabilir. Bazen
sadece Build Succeeded mesajı yeterlidir.
        public void Dispose()
        {
            Console.WriteLine("FileManager için Dispose çağrısı");
            base.Dispose();
        }
    }
}
```

Uygulamada **StreamManager** isimli **abstract** tipten olan **sınıf(Class), IDisposable** arayüzünü implemente etmektedir. Biraz daha ilerlemeden önce buradaki bazı temelleri de hatırlamamızda yarar olacağı kanısındayım;

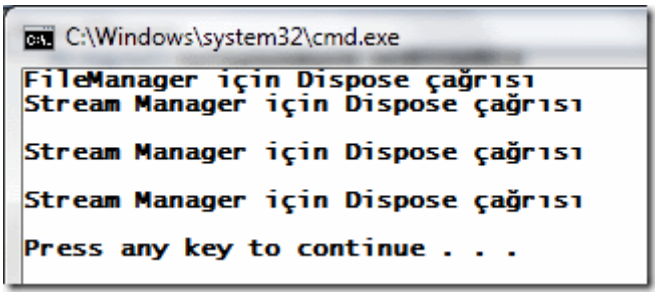
- **abstract** tipler örneklenemezler ve **çok biçimli(Polymorhic)** bir yapıya sahiptirler. Diğer yandan kendilerinden **türeyen tipler(Derived Types)** için mutlaka uygulanması gereken üyeleri içerebilirler ki bunlarda **abstract** olarak tanımlanırlar.
- **IDisposable** bir **arayüz tipidir(Interface Type)** ve **CLR** tarafında bellek yönetimi ile ilişkili bildirim içerir. **Arayüzler** sınıflar gibi işlevsel bloklara sahip üyeler içeremezler. Sadece kendisini uygulayan tiplerin yapması zorunlu olan bildirimleri barındırırlar. **Dispose** metodu **IDisposable** arayüzünün belirttiği zorunlu üyelerden birisidir. Arayüzlerde çok biçimli yapıya sahiptirler.
- çok biçimlilik(Polymorphsym) aslında türeyen tiplere ait nesne örneklerinin, türedikleri tiplere ait değişkenler tarafından taşınmaları halinde ortaya çıkan bir özelliktir. Burada **üst tipte(base type)** tanımlanıp, **alt tipte(sub type)** ezilen üyelerin büyük önemi vardır. Nitekim üst tipe atanan bir alt tip nesne örneğinin ezilen üyesi, üst tip üzerinden çağırılabilmektedir. Bir başka deyişle üst tipler yeri geldiklerinde alt tipler gibi davranış gösterebilmektedir. çok biçimlilik adı da zaten buradan gelmektedir.
- Bir tipin çok biçimli yapıda olması özellikle **Plug-In** tabanlı programlama da önemlidir. Nitekim üst tipleri ve ezilebilen*(ezilmesi zorunlu olan)* üyeleri tanıyan bir sistemin genişletilmesinde, bu kurallara uyan tiplerin entegrasyonu söz konusudur.
- **sealed** olarak tanımlanmış olan **FileManager** tipi aslında kendisinden türetme yapılamayan kısır bir sınıftır.
- **Dispose** metodları çoğunlukla tipe ait nesne örneklerinin **Garbage Collector(GC)** tarafından toplandığı noktalarda devreye girmektedir. Dolayısıyla **IDisposable** ile **GC'** ye bir takım ek talimatlar yollanabilir ve özellikle **unmanaged** tarafla ilişkili kaynak temizleme işlemleri bu metod üzerinden icra edilir.

Bu temelleri hatırladıktan sonra vakamıza dönebiliriz. **FileManager** sınıfı **StreamManager** tipinden türetilmiştir. **StreamManager** ise **IDisposable** arayüzünü uygulamaktadır. Dolayısıyla **StreamManager** tipinin ezmesi gereken bir **Dispose** metodu söz konusudur. Lakin kritik nokta **FileManager** tipi içerisinde de bir **Dispose** metodunun yazılmış olmasıdır.

Main metodu içerisindeki kod bloğuna bakıldığında **FileManager** tipinin örneklenip*(fm isimli değişken)* **StreamManager** tipinden bir değişkene atandığı görülmektedir. Diğer yandan **IDisposable** arayüzüne ait bir değişkene de aynı örnek atanmıştır. Her ne kadar **StreamManager** ve **IDisposable,** örneklenemeyen tipler olsalar da bu, değişken olarak tanımlanamayacakları anlamına gelmemelidir.

Bunlara ek olarak **IDisposable** ve **StreamManager'** ın çok biçimli tipler oldukları da ortadadır. Dolayısıyla **sm** ve **dm** değişkenleri üzerinden

çağırılan **Dispose** metodlarının,**polimorfizim(çok biçimlilik)** nedeni ile aslında taşınmakta olan **fm** nesne örneğinin **Dispose** fonksiyonuna doğru olması düşünülebilir. öyleyse uygulamanın çalışma zamanındaki çıktısına bir bakalım.



Dikkat edileceği üzere **fm** nesne örneği üzerinden yapılan çağrıda **FileManager** tipine ait **Dispose** metodu yürütülmüştür. Ancak **base class** ve uygulanan **IDisposable** arayüzleri üzerinden yapılan **Dispose** çağrılarında bu böyle olmamıştır. Her iki çağrıda da **sub class** olan **FileManager** tipine ait **Dispose** metodu yerine **StreamManager** tipinin **Dispose**metodunun icra edildiği görülmektedir. Acaba gerçekten böyle midir? **Debug** noktalarını koyarak ilerlediğimizde böyle olduğu ispat edilebilir.

StreamManager üzerinden **Dispose** çağrısı yapıldığında *(sm.Dispose(); satırı)* kod, **StreamManager sm=fm** atamasına rağmen **FileManager** tipinin **Dispose** metodunauğramamaktadır.

IDisposable üzerinden **Dispose** çağrısında *(dm.Dispose();)* kod **IDisposable dm=fm**; atamasına rağmen **FileManager** tipinin **Dispose** metoduna uğramamış ve yine **StreamManager** tipinin**Dispose** metodu çağırılmıştır ki sanırım en ilginç olanı da budur.

Peki bu durumların oluşmasının sebebi nedir? Aslında sorun bilinçsiz olarak yapılan üye gizleme **(Implicitly Name Hiding)** operasyonundan kaynaklanmaktadır. Nitekim şu anda **FileManager**içerisinde, üst tipteki ile aynı isimde olan bir metod tanımı söz konusudur ve çalışma zamanı bu kullanımı gördüğünde varsayılan olarak üst tipe ait üyeleri çağırmaktadır. Bir başka deyişle üst tip üyesi alt tipi gizlemektedir. Gerçi bilinçsiz bir şekilde üye gizleme yapıldığı pek doğru değildir. Nitekim **Visual Studio IDE'** si geliştiriciyi bu noktada aşağıdaki gibi uyarmaktadır.

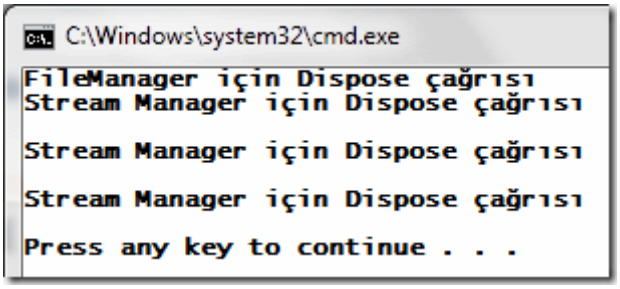
Ancak dikkatsiz bir geliştirici çok kalabalık bir projede bu tip bir **warning** mesajını kolayca gözden kaçırabilir. Aslında çoğu zaman geliştirici için projenin başarılı bir şekilde derlenmesi yeterli olmaktadır. Tabi **ReSharper** gibi araçları kullanmıyorsak ya da **TFS(Team Foundation Server)** altında geliştirme yapıp **Policy'** ler uygulayarak **Warning'** ler aşılmadan kodun derlenmesini engellemiyorsak bu tip gözden kaçırmaların sayısı artacaktır.

Peki bu temeli bilen bir geliştirici sorun haline gelen vakaya düşmemek için ne yapmalıdır?

İlk akla gelen yöntem **üst sınıf(base class)** içerisinde tanımlı olan ve **IDisposable** arayüzü üzerinden gelen **Dispose** metodunun virtual olarak tanımlanmasıdır.

```
abstract class StreamManager
    :IDisposable
{
    #region IDisposable Members

    public virtual void Dispose()
    {
        Console.WriteLine("Stream Manager için Dispose çağrısı\n");
    }

    #endregion
}
```

Ancak virtual olarak yapılan tanımlama da yeterli değildir. Halen daha **base class'** ta yer alan aynı isimli **Dispose** metodunun alt tiptekini gizlemesi durumu devam etmektedir. Dolayısıyla çalışma zamanındaki durum değişmeyecek ve aşağıdaki çıktı alınmaya devam edecektir.



Abstract bir üye olarak **Dispose** metodunun tanımlanması da düşünülebilir ancak üzerinde çalıştığımız senaryo da geçerli bir kullanım değildir.

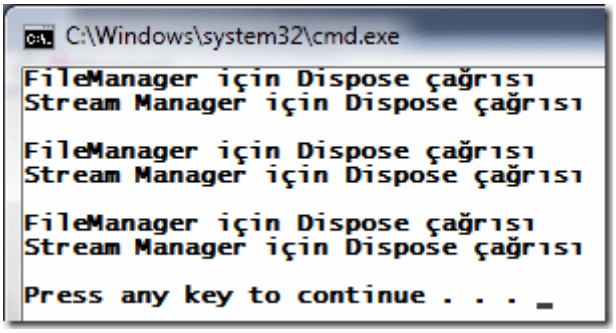
Bilindiği üzere **abstract** üyeler herhangibir şekilde kod bloğu içermezler ve türeyen tipler içerisinde mutlaka **ezilmek(override)** zorundadırlar.

Aslında üst sınıfın üye gizleme işlemini yapması istenmiyorsa yapılması gereken yol alt tip içerisindeki ezme(override) işleminin açık bir şekilde gerçekleştirilmesidir. Söz gelimi **virtual** kullanımı söz konusu ise kodun aşağıdaki şekilde düzenlenmesi yeterli olacaktır.

```
abstract class StreamManager
    :IDisposable
{
    #region IDisposable Members

    public virtual void Dispose()
    {
        Console.WriteLine("Stream Manager için Dispose çağrısı\n");
    }

    #endregion
}

sealed class FileManager
    : StreamManager
{
    public override void Dispose()
    {
        Console.WriteLine("FileManager için Dispose çağrısı");
        base.Dispose();
    }
}
```

Dikkat edileceği üzere **FileManager** tipi içerisinde yer alan **Dispose** metodu **override** bildirimi ile tanımlanmıştır. Yani açık bir şekilde üst tipten

gelen **Dispose** metodunun ezilmesi ve yerine bu fonksiyon gövdesinin çağırılması gerektiği belirtilmiştir. Bu durumda çalışma zamanı çıktısı tam da istediğimiz gibi olacaktır.



Tabi şu durumda unutulmamalıdır. Gerçekten bilinçli bir şekilde **üye gizleme(Name Hiding)** işleminin yapılması gerekiyorsa, **new** operatörünün kullanılması gerekmektedir. Aşağıdaki kod parçasında olduğu gibi.

```
sealed class FileManager
      : StreamManager
   {
      public new void Dispose()
      {
         Console.WriteLine("FileManager için Dispose çağrısı");
         base.Dispose();
      }
   }
```

Elbette bu durumda üst tipte yer alan **Dispose** metodunun **virtual** olmasına da gerek yoktur. **new** operatörünün kullanılması, bir anlamda geliştiricinin de ne yaptığının farkında olması demektir. Yazımızda değerlendirdiğimiz bu örnek vakanın aslında kulağımıza bir cümleyi küpe yaptığı da gerçektir: **"Siz siz olun Warning mesajlarını dikkate alın"**Böylece geldik bir yazımızın daha sonuna. Tekrardan örüşünceye dek hepinize mutlu günler dilerim.

ImplicitHiding.rar (22,03 kb)

TPL Senkronizasyonu Sağlamak - 2 (Interlocked) (2010-12-19T00:45:00)

tpl,task parallel library,synchronization primitives,parallel programming,.net framework 4.0,c# 4.0,interlocked,

Merhaba Arkadaşlar,

"Seçimi zaten yaptın. Şimdi onu anlaman gerekli." Sanırım **Matrix** filminde **Neo** i

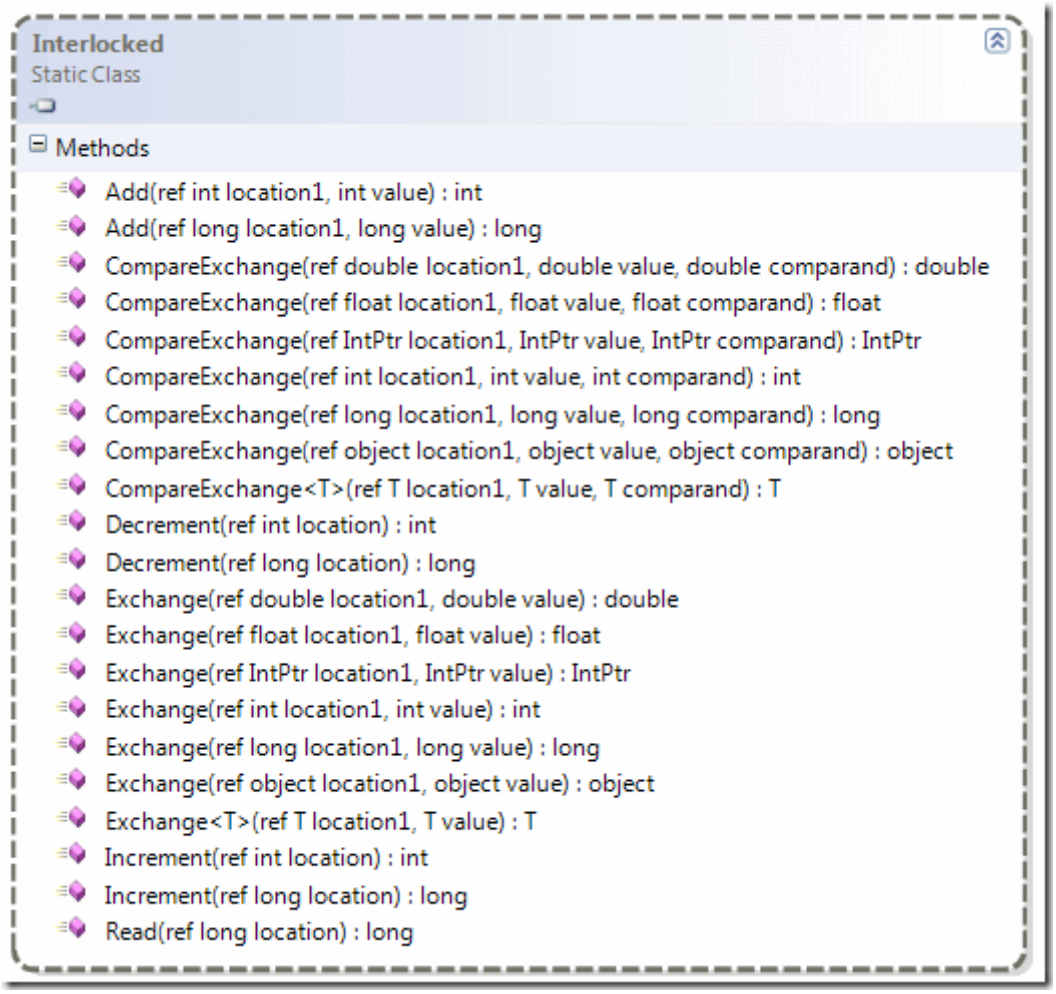
le **Oracle'** ın felsefe içeren ve uzun uzun düşünüldüğünde akla son derece anlaşılır gelen bir kaç sohbetinde geçen repliklerden birisi de buydu. Aslında ben bunu kendi hayatımda zaman zaman **"çözümü uyguladın. Şimdi onun her parçasının ne anlama geldiğini öğrenmen gerekli"** diye çeviriyorum.

Yazımızın giriş kısmını oluşturan bu paragrafın üretiliş amacı ise en sonda geliştireceğimiz örnek olacak aslında. Gerçekten de bazen çeşitli vakaların çözüme ulaşmasında, gerekli teknikleri uyguladıktan sonra onları anlamaya çalışmak daha öğretici olabilmekte. Merak ediyor musunuz?

Hatırlayacağınız üzere TPL Senkronizasyonu Sağlamak – 1 başlıklı yazımız ile **Task Parallel Library(TPL)** tarafında senkronizasyon kullanımını incelemeye başlamıştık. Aslında başımıza iş mi açtık bilemiyorum ama sonuç itibariyle kritik bir konu olduğunda sanıyorum ki hepimiz hem fikiriz. önceki yazımızda değerlendirdiğimiz senaryoda, **lock keyword**kullanımı ile izole edilmiş bir veri alanının, farklı iş parçaları tarafından nasıl güvenli bir şekilde kullanılabileceğini analiz etmiştik. üstelik bu **keyword'** ün aslında arka planda**Monitor** tipini kullandığını da **Intermediate Language(IL)** kodunda görmüştük. Elbette iş parçalarının senkronizasyonu için kullanılabilecek farklı tipler de söz konusudur.**Interlocked** sınıfı gibi.

Interlocked Kullanımı

System.Threading isim alanı altında yer alan **Interlocked** tipine ait atomic metodlar işletim sisteminin ve donanımın özelliklerini kullanarak senkronizasyonu daha yüksek performans ile kullanmayı vaat ederler. Aşağıdaki sınıf diagramında Interlocked tipinin üyleri görülmektedir.



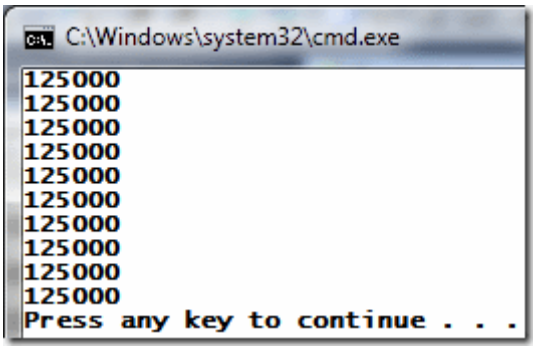
Interlocked tipi şekilden de görüleceği üzere **static** bir sınıftır*(Dolayısıyla sadece static üyeler içerebilir ve örneklenemez)* Temel olarak **Add,CompareExchange, Decrement,Exchange,Increment** ve **Read** metodları ile bunların **aşırı yüklenmiş(Overloaded)** versiyonlarını içermektedir. Metodların genel yapısı incelendiğinde atomic fonksiyonların **int, long,double, float, object** gibi tipler ile çalıştığı görülmektedir. Dilerseniz metodların işlevsellikleri hakkında kısa bilgiler vererek ilerlemeye çalışalım.

Exchange metodu ile bir değer ataması yapılabilmektedir. **Exchange** metodu dikkat edileceği üzere **double, float, IntPtr, int, long** ve **object** tipleri ile çalışmaktadır. Ayrıca generic bir versiyonu da bulunmaktadır. Bunun dışında **1er arttırma** ve **azaltma** işlemleri için kullanılabilecek basit **Increment** ve **Decrement** metodları mevcuttur.
Ayrıca **Int32** veya **Int64** tipinden sayısal değerleri toplayan **Add** metoduna da sahiptir. **Interlocked** tipinin kullanışlı olan metodlarından birisi de **CompareExchange** fonksiyonudur. Bu fonksiyon, parametre olarak gelen sayısal değerlerin eşitliğini kontrol etmekte ve aynı iseler **Exchange** işlemini gerçekleştirmektedir. Aşağıdaki kod parçasında **Interlocked** için örnek bir kullanım söz konusudur.

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace TPLSynchronization
{
   class Program
   {
      static void Main(string[] args)
      {
         for (int i = 0; i < 10; i++)
         {
            TestMethod();
         }
      }

      static void TestMethod()
      {
         Task[] tasks = new Task[5];
         Plane f14Tomcat = new Plane();

         for (int i = 0; i < 5; i++)
         {
            tasks[i] = new Task(() =>
               {
                  for (int j = 0; j < 12500; j++)
                  {
                     Interlocked.Exchange(ref f14Tomcat.Altitude, (j+1)* 10);
                  }
               }
            );
         }

         foreach (Task task in tasks)
         {
            task.Start();
         }

         Task.WaitAll(tasks);

         Console.WriteLine("{0}",f14Tomcat.Altitude);
      }

      class Plane
      {
```

```
        public int Altitude;
      }
   }
}
```

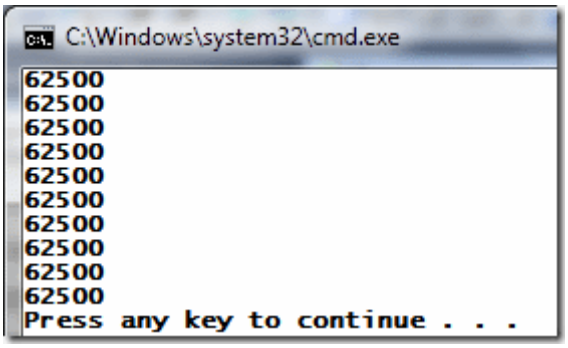
Plane sınıfı içerisinde **Altitude** isimli **int** tipinden bir **Alan(Field)** yer almaktadır. Bu değer ilgili **Task** örnekleri tarafından izole edilen yerel değişken olarak ele alınmaktadır. Dikkat edileceği üzere **Task** örneklerinin oluşturulduğu ifade içerisinde yer alan kod bloğundan **Exchange** metodu kullanılarak **Altitude** değerinin **(j+1)*10** birim kadar arttırılması sağlanmaktadır. Bu işlem **5** ayrı **Task** tarafından gerçekleştirilmektedir ve bildiğiniz üzere senkronizasyon için bir tedbir alınmadığı takdirde, her bir deneme de sonuçlar farklı olacaktır. örnek uygulamanın çalışma zamanı çıktısı ise aşağıdaki gibidir.



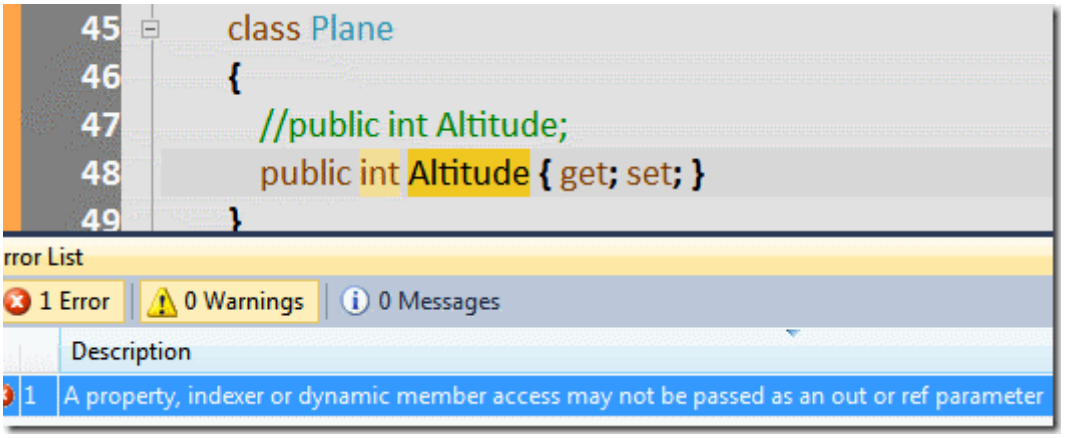
Görüldüğü üzere **TestMethod** için yapılan **10 ayrı denemenin sonucuda aynıdır**. Bu örnekte **Exchange** metodu kullanılmıştır. Ancak bazen çok daha basit işlemler söz konusu olabilir. Söz gelimi **1er arttırma** veya **azaltma** gibi. Bu durumda **Interlocked** tipinin **static Increment** veya **Decrement** metodlarını kullanaraktan da gerekli senkronizasyonu sağlayabiliriz. örneğimizde **Altitude** değerini **10000** kez **1er birim arttırmak** için yapmamız gereken tek şey kodda aşağıdaki değişikliği yapmaktan ibarettir.

```
static void TestMethod()
{
   Task[] tasks = new Task[5];
   Plane f14Tomcat = new Plane();

   for (int i = 0; i < 5; i++)
   {
      tasks[i] = new Task(() =>
         {
            for (int j = 0; j < 12500; j++)
            {
               Interlocked.Increment(ref f14Tomcat.Altitude);
               //Interlocked.Exchange(ref f14Tomcat.Altitude, (j+1)* 10);
            }
         }
      );
   }
```

Bu kodun çalışma zamanı çıktısı ise aşağıdaki gibi olacaktır.



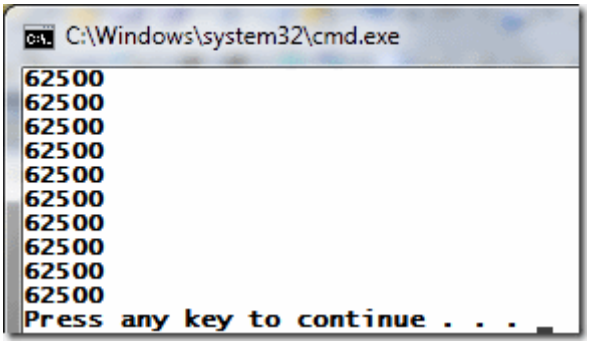
Yanlız **Interlocked** tipinin atomic fonksiyonları göz önüne alındığında üzerinde işlem yapılan asıl değişkenlerin **ref** tipinden aktarıldığı görülmektedir. Bu durumda **Plane**tipinin **Altitude** isimli **alanının(Field)**, **özellik(Property)** olarak tasarlanması bir sorun teşkil etmektedir.



Bu durumda **Altitude** özelliğini **local** bir değişken olarak ele almak ve sonrasında **Interlocked** tipinin ilgili fonksiyonlarında kullanmak gerekmektedir. Bu amaçla kod parçasını aşağıdaki gibi değiştirebiliriz.

```
static void TestMethod()
      {
         Task[] tasks = new Task[5];
         Plane f14Tomcat = new Plane();
         int altitude = f14Tomcat.Altitude;

         for (int i = 0; i < 5; i++)
         {
            tasks[i] = new Task(() =>
               {
                  int result = 0;
                   for (int j = 0; j < 12500; j++)
                   {
                      result=Interlocked.Increment(ref altitude);
                      //Interlocked.Exchange(ref f14Tomcat.Altitude, (j+1)* 10);
                   }
                   f14Tomcat.Altitude = result;
               }
            );
         }

         foreach (Task task in tasks)
         {
            task.Start();
         }

         Task.WaitAll(tasks);

         Console.WriteLine("{0}",f14Tomcat.Altitude);
      }
```

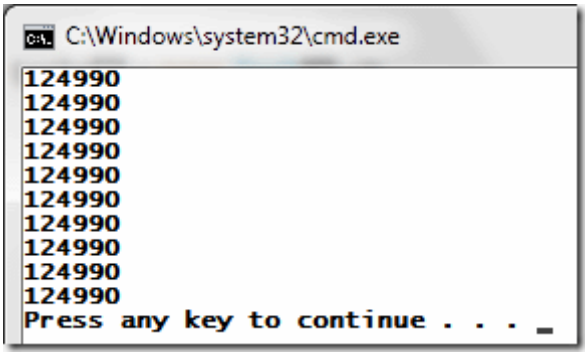
Burada dikkat edilmesi gereken noktalardan birisi de, **Increment** metodunun geriye bir sonuç döndürüyor olmasıdır. Bu aslında yapılan arttırma işleminin bir sonucudur. Dolayısıyla **result**değerini tekrardan **Altitude** özelliğine atamak yeterlidir. İşte çalışma zamanı sonuçları.



Görüldüğü gibi yine aynı sonuçlar üretilmiştir. **Ancak!!!!** Dikkatli olmamız da yarar vardır. Son örneğimizdeki yerel değişken taktiği gerçekten doğru mudur? Bu kullanımı**Increment** yerine **Exchange** metodu içinde yaptığımızı düşünelim.

```
static void TestMethod()
      {
         Task[] tasks = new Task[5];
         Plane f14Tomcat = new Plane();
         int altitude = f14Tomcat.Altitude;

         for (int i = 0; i < 5; i++)
         {
            tasks[i] = new Task(() =>
               {
                  int result = 0;
                  for (int j = 0; j < 12500; j++)
                  {
                     result=Interlocked.Exchange(ref altitude, (j+1)* 10);
                  }
                  f14Tomcat.Altitude = result;
               }
            );
         }
```

Ve çalışma zamanı sonuçları.

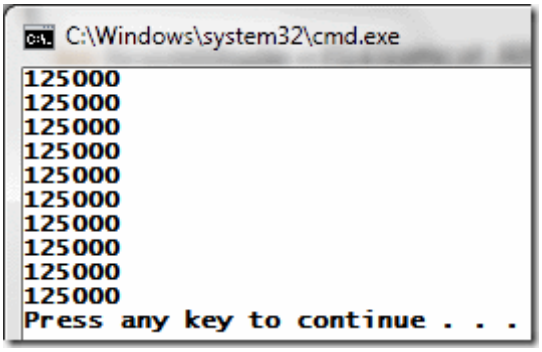


Tüm denemelerin aynı sonucu üretmesi gayet güzel. Ancak aynı senaryonun **Field** içeren kullanımında elde ettiğimiz sonuçlar **125000** dir. Oysaki aynı sonuçları üretmeleri beklenmektedir. öncelikle aynı sonuçları üretecek kod parçasını geliştirelim. Bu amaçla kodumuzu aşağıdaki gibi güncelleştirmemiz gerekmektedir.

```
static void TestMethod()
      {
         Task[] tasks = new Task[5];
         Plane f14Tomcat = new Plane();
         int altitude = f14Tomcat.Altitude;

         for (int i = 0; i < 5; i++)
         {
            tasks[i] = new Task(() =>
               {
                  int firstAltitude = f14Tomcat.Altitude;
                  int localAltitude = firstAltitude;

                  for (int j = 0; j < 12500; j++)
                  {
                     localAltitude = (j + 1) * 10;
                  }
                 int sharedAltitude=Interlocked.CompareExchange(ref altitude,
localAltitude, firstAltitude);
                f14Tomcat.Altitude=sharedAltitude;
                }
            );
         }

         foreach (Task task in tasks)
         {
            task.Start();
         }

         Task.WaitAll(tasks);
```

```
        Console.WriteLine("{0}",f14Tomcat.Altitude);
    }
```

Kodu bu şekilde çalıştırdığımızda istediğimiz sonuçları elde ettiğimizi görebiliriz.



Koda baktığımızda **Task** örneklenmesi sırasında **firstAltitude** ve **localAltitude** isimli iki değişken üretildiği görülmektedir. Bu değişkenlerden **localAltitude** aslında yerel değişken olarak kullanılmaktadır. Bir başka deyişle sadece tanımlandığı **Task** örneğine ait bir değişkendir. Ancak bir de **5 Task** örneğinin ortaklaşa kullandıkları ve asıl amacımız olan **Altitude** özelliğinin değeri söz konusudur. Bu nedenle her bir **Task'** in paylaşılan değişkenin son değerini alabilmesi içinde **startedAltitude** isimli değişken kullanılmaktadır.

Task gövdesi içerisinde yer alan **for** döngüsü dikkat edileceği üzere yerel değişkeni arttırmaktadır. **for** döngüsünü takip eden satırda ise **CompareExchange** metoduna bir çağrıda bulunulduğu görülmektedir. Bu metod, paylaşımlı verinin güncellenmesi amacıyla kullanılmaktadır. Burada paylaşımlı veri ile başlangıçtaki değer arasında bir kıyaslama yapılır. örneğimize göre eğer **localAltitude** ve **altitude** değerleri birbirlerine eşitse, **localAltitude**' un anlık güncel değeri **altitude** içerisinde saklanır. Aksi durumda herhangibir operasyon gerçekleştirilmez.

Burada karşılaştırma ve veri değiş tokuşu fonksiyonellikleri söz konusudur ve her ikisi de aslında tek bir atomic operasyon olarak ele alınmaktadır. **CompareExchange** metodunun dönüşü aslında metoda gelen ilk parametrenin orjinal değeridir.
Aslında **Interlocked** tipinin bu efsane kullanımını daha net kavramak adına MSDN ' de ilgili içeriğe bakmakta yarar vardır. Ben sadece kapıyı gösterebiliyorum. Oradan geçmesi gereken kişi sizsiniz.

Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

TPLSynchronization_Interlocked.rar (23,07 kb) **[örnek Visual Studio 2010 Ultimate Sürümünde geliştirilmiş ve test edilmiştir]**

TPL Senkronizasyonu Sağlamak - 1 (2010-12-19T00:40:00)

tpl,task parallel library,.net framework 4.0,synchronization primitives,parallel programming,

Merhaba Arkadaşlar,

Pek çoğumuzun anahtarlığında sayısız anahtar bulunmaktadır. özellike gerilim filmlerinde bu anahtarlardan doğru olanı bulmak ve anahtar deliğine sokmak, hep zaman alan başarısız kaçış girişimleri olarak sahnelenir. Genellikle bu başarız girişimlerin sonunda ne olduğu malumdur. Ancak ister gerilim filmi olsun ister olmasın sonuçta anahtar deliğine herhangibir zamanda takılabilecek sadece tek bir anahtar söz konusudur. üstelik bu anahtar, aynı yere başka bir anahtarın takılmasına da izin vermez. Aslında izin verip vermemesi, anahtarı tutan veya kullanan kişinin elindedir.

Aslında şu anda varmak istediğim nokta **lock** kelimesidir. **lock**, çok kanallı uygulamalarda **Thread** senkronizasyon işlemlerinde ele alınan tekniklerden yanlızca birisidir. çok doğal olarak farklı amaçlarla ele alınan tipler de söz konusudur.

Tüm bunlara ek olarak uzun zamandır bilinen **Task Parallel Library** ve doğal olarak yeni paralel programlama yapısı mevcuttur. Dolayısıyla aynı senkronizasyon vakaları **TPL** içerisinde de geçerlidir ve bu amaçla eklenmiş yeni özellikler bulunmaktadır. İşte bu yazımız ile birlikte **TPL** içerisinde senkronizasyon konusunu incelemeye çalışıyor olacağız.

TPL ve Shared Data Isolation başlıklı yazımızda, **n sayıda Task** örneğinin ortaklaşa kullandıkları bir veri alanı üzerindeki işlemlerinin, ne gibi sonuçlara yol açabileceğini incelemiş ve bunun önün geçmek için basit bir kaç yolu ele almıştık. Hatırlayacağınız üzere değerlendirdiğimiz senaryoda, **Plane** tipinden olan nesne örneğine ait **Altitude** özelliğinin değerinin, **Task** blokları içerisinde değiştirildiği nokta, sorunun oluşmasına neden olan kritik bölgeydi.

```
for (int i = 0; i < 5; i++)
        {
           tasks[i] = new Task(() =>
          {
              for (int j = 0; j < 1000; j++)
              {
                f16.Altitude += j - 5;
              }
           }
```

```
        );
        tasks[i].Start();
    }
```

Aslında ezelden beri **Multi-Thread** programlamada, **Thread'** lerin senkronizasyonu konusu öyle ya da böyle bir şekilde kulağımıza gelmiştir. çok doğal olarak benzer ihtiyaçlar **.Net Framework 4.0** Paralel Programlama alt yapısı içinde geçerlidir. Ancak **.Net Framework 4.0** tarafında **Lightweight Primitives** adında yeni ve daha basit senkronizasyon tipleri de söz konusudur. Bu tipler klasik senkronizasyon tiplerine göre daha kola uygulanabilir. Tabi klasik senkronizasyon **Primitive**' lerinin uygulanması her ne kadar zor olsa da, birden fazla **Application Domain** üzerinde kontrole izin vermektedirler. Oysa ki **LightWeight Primitive**' ler sadece tek bir **Application Domain** için uygulanabilir. Olayı daha fazla karmaşıklaştırmadan önce senkronizasyon ile neyi ifade ettiğimizi ortaya koymamızda yarar vardır.

"Herhangibir t anında kritik olan bölgeyde sadece tek bir Task örneğinin işlem yapmasını sağlamak"

Senkronizasyonu sağlamak için **.Net Framework** üzerinde önceden tanımlı çeşitli **Primitive**' lerden yararlanılmaktadır. Aslında özel veri tipleri olarak düşünebileceğimiz**Primitive**' ler, kritik bölgelere doğru yapılan **Task** erişimlerini kontrol altına alan varlıklar olarak düşünülebilirler.

Genel işleyiş şekline bakıldığında, kritik bölgede yer alan veriye erişmek isteyen bir **Task** örneği, bu isteğini ilk önce **Primitive**' e iletmektedir. Eğer söz konusu kritik bölge müsait ise,**Task** işlemlerini icra etmeye başlayabilir. Ancak müsait değilse, **Task**, **Primitive** nesnesinin belirlediği ilkelere göre beklemede kalacaktır. Bu anda, söz konusu bölgede çalışmakta olan **Task**örneği de, işini bitirdiğinde **Primitive**' e bildirimde bulunacaktır. Bu bildirimin ardından **Primitive** nesne, bekleyen **Task** örneğinin söz konusu bölgede işlem yapmasına izin verecektir. Tabi işlem yapmaya başlayan **Task** örneği, söz konusu bölgeyi en azından kilitlediğini, **Primitive**' e bildirecektir. Elbette gerçek hayat senaryolarında durum bu anlatım şeklinde olduğu gibi basit değildir. **N** sayıda **Task** söz konusu olduğunda birbirlerine göre önceliklerini belirlemek gibi ihtiyaçlarımız da olabilir.

Bu yazımızla birlikte söz konusu senkronizasyon tiplerini basit bir şekilde incelemeye başlıyor olacağız. Haydi başlayalım.

En Basit Askerimiz : lock

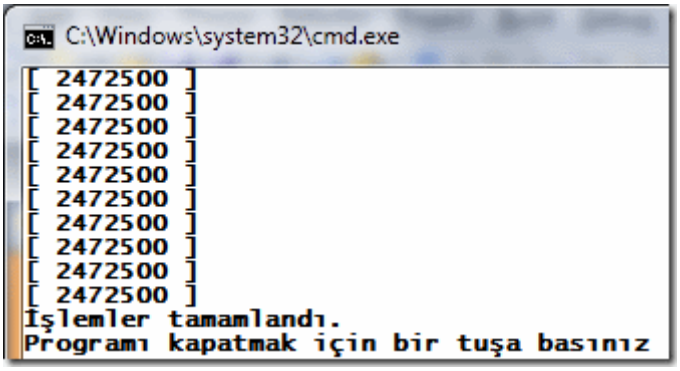
lock anahtar kelimesi aslında **System.Threading.Monitor** sınıfının daha basit bir şekilde uygulanmasını sağlayan bir **Primitive** olarak düşünülebilir. Normal şartlar altında kritik bölgeyle olan etkileşimlerde **Monitor** tipinin **Enter, TryEnter, Exit** gibi metodlarının kullanılması gerekmektedir. **Monitor** tipi **HeavyWeight Primitive** olarak bilinmektedir ve

daha alt seviyedeki vakaların karşılanmasında kullanılmaktadır. Aşağıdaki kod parçasında daha önceki yazımızda ele aldığımız sorunlu senaryonun **lock** ile çözümü gösterilmektedir.

```
using System;
using System.Threading.Tasks;

namespace SynchronizationPrimitives
{
   class Program
   {
      static void Main(string[] args)
      {
         for (int i = 0; i < 10; i++)
         {
            TestMethod();
         }

         Console.WriteLine("İşlemler tamamlandı.\nProgramı kapatmak için bir tuşa
basınız");
         Console.ReadLine();
      }

      private static void TestMethod()
      {
         Plane f16 = new Plane();
         Task[] tasks = new Task[5];

         object obj = new object();

         for (int i = 0; i < 5; i++)
         {
            tasks[i] = new Task(() =>
            {
               for (int j = 0; j < 1000; j++)
               {
                  lock (obj)
                  {
                     f16.Altitude += j - 5;
                  }
               }
            }
            );
            tasks[i].Start();
         }
```

```
            Task.WaitAll(tasks);

            Console.WriteLine("[ {0} ]", f16.Altitude);
        }
    }

    class Plane
    {
        public int Altitude { get; set; }
    }
}
```

İlk olarak tüm **Task** örneklerinin, kritik bölgede işlem yapılırken **lock** bloğuna girdiklerini ifade edebiliriz. Bu **lock** bloğu içerisindeki çalışma süresi boyunca, yürütücü **Task** dışında başka bir **Task** örneğinin **Altitude** değerini değiştirmesine izin verilmemektedir. **lock** keyword' ü kullanılırken **object** tipinden bir nesneyi parametre olarak alır. Bu nesne de aslında tüm **Task** örnekleri için ortaktır. Uygulamayı kaç kere çalıştırırsanız çalıştırın beklediğimiz aynı sonuçları elde ettiğimizi görebiliriz. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.



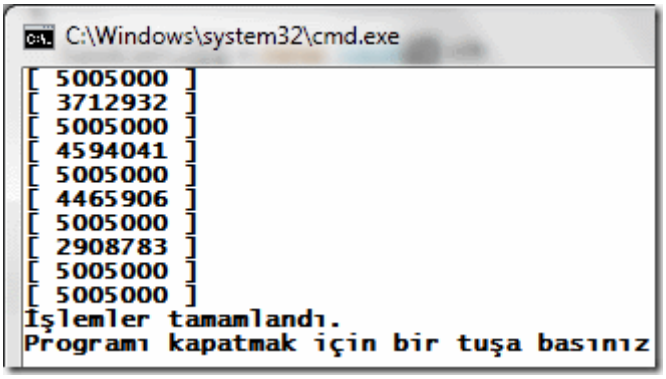
Şimdi senaryomuzu biraz daha ilginçleştirelim. Bu sefer iki ayrı **Task** kümesinin, **Altitude** özelliği üzerinden farklı hesaplamalar yaptığını varsayacağız. Bu amaçla vaka kodumuzu aşağıdaki gibi geliştirdiğimizi düşünelim.

```
using System;
using System.Threading.Tasks;

namespace SynchronizationPrimitives
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 0; i < 10; i++)
            {
```

```
            TestMethod();
        }

        Console.WriteLine("İşlemler tamamlandı.\nProgramı kapatmak için bir tuşa
basınız");
        Console.ReadLine();
    }

    private static void TestMethod()
    {
        Plane f16 = new Plane();
        Task[] taskSet1 = new Task[5];
        Task[] taskSet2 = new Task[5];

        for (int i = 0; i < 5; i++)
        {
            taskSet1[i] = new Task(() =>
            {
                for (int j = 0; j < 1000; j++)
                {
                    f16.Altitude += j - 5;
                }
            }
            );
            taskSet1[i].Start();
        }

        for (int i = 0; i < 5; i++)
        {
            taskSet2[i] = new Task(() =>
            {
                for (int j = 0; j < 1000; j++)
                {
                    f16.Altitude += j+7;
                }
            }
            );
            taskSet2[i].Start();
        }

        Task.WaitAll(taskSet1);
        Task.WaitAll(taskSet2);
```

```
            Console.WriteLine("[ {0} ]", f16.Altitude);
        }
    }

    class Plane
    {
        public int Altitude { get; set; }
    }
}
```

Bu sefer **taskSet1** ve **taskSet2** isimli iki farklı **Task** dizisinin **Altitude** özelliği üzerinde gerçekleştirdiği farklı işlemler söz konusudur. çok doğal olarak ortaklaşa kullanılan değişken söz konusu olduğundan çalışma zamanında aynı sonuçların elde edilmesi nadir bir durumdur. Uygulamanın aşağıdaki örnek çıktısında bu durum açık bir şekilde görülebilir.



Bildiğiniz üzere her denemenin aynı sonucu üretiyor olmasını beklemekteyiz.
Peki **lock** mekanizmasını bu tip senaryoda nasıl kullanabiliriz? Aşağıdaki kod parçasında bu çözümleme işlemi değerlendirilmektedir.

```
using System;
using System.Threading.Tasks;

namespace SynchronizationPrimitives
{
    class Program
    {
        static void Main(string[] args)
        {
            for (int i = 0; i < 10; i++)
            {
                TestMethod();
            }

            Console.WriteLine("İşlemler tamamlandı.\nProgramı kapatmak için bir tuşa
basınız");
```

```
    Console.ReadLine();
}

private static void TestMethod()
{
    Plane f16 = new Plane();
    Task[] taskSet1 = new Task[5];
    Task[] taskSet2 = new Task[5];

    object obj = new object();

    for (int i = 0; i < 5; i++)
    {
        taskSet1[i] = new Task(() =>
        {
            for (int j = 0; j < 1000; j++)
            {
                lock (obj)
                {
                    f16.Altitude += j - 5;
                }
            }
        }
        );
        taskSet1[i].Start();
    }

    for (int i = 0; i < 5; i++)
    {
        taskSet2[i] = new Task(() =>
        {
            for (int j = 0; j < 1000; j++)
            {
                lock (obj)
                {
                    f16.Altitude += j + 7;
                }
            }
        }
        );
        taskSet2[i].Start();
    }

    Task.WaitAll(taskSet1);
    Task.WaitAll(taskSet2);
```

```
            Console.WriteLine("[ {0} ]", f16.Altitude);
        }
    }

    class Plane
    {
        public int Altitude { get; set; }
    }
}
```

Dikkat edileceği üzere ilk örneğimizdeki kodlama stilinden farklı bir uygulanış biçimi yoktur. Dikkat edilmesi gereken nokta ise, her iki **lock** bloğu içinde aynı **object** örneğinin kullanılmış olmasıdır.

lock Aslında Montior Tipini Kullanır

Daha önceden de bahsettiğimiz üzere **lock** bloğu aslında **Monitor** tipinin ilgili metodlarını kullanmaktadır. Bu durum kodun arka planda üretilen **IL** çıktısında rahat bir şekilde görülebilir.

```
.method public hidebysig instance void  '<TestMethod>b__3'() cil managed
{
  // Code size       85 (0x55)
  .maxstack  4
  .locals init ([0] int32 j,
           [1] bool '<>s__LockTaken1',
           [2] object CS$2$0000,
           [3] bool CS$4$0001)
  IL_0000:  nop
  IL_0001:  ldc.i4.0
  IL_0002:  stloc.0
  IL_0003:  br.s       IL_0048
  IL_0005:  nop
  IL_0006:  ldc.i4.0
  IL_0007:  stloc.1
  .try
  {
    IL_0008:  ldarg.0
    IL_0009:  ldfld      object SynchronizationPrimitives.Program/'<>c__DisplayClass6'::obj
    IL_000e:  dup
    IL_000f:  stloc.2
    IL_0010:  ldloca.s   '<>s__LockTaken1'
    IL_0012:  call       void [mscorlib]System.Threading.Monitor::Enter(object, bool&)
    IL_0017:  nop
    IL_0018:  nop
```

```
    IL_0019:  ldarg.0
    IL_001a:  ldfld      class SynchronizationPrimitives.Plane
SynchronizationPrimitives.Program/'<>c__DisplayClass6'::f16
    IL_001f:  dup
    IL_0020:  callvirt   instance int32 SynchronizationPrimitives.Plane::get_Altitude()
    IL_0025:  ldloc.0
    IL_0026:  ldc.i4.7
    IL_0027:  add
    IL_0028:  add
    IL_0029:  callvirt   instance void
SynchronizationPrimitives.Plane::set_Altitude(int32)
    IL_002e:  nop
    IL_002f:  nop
    IL_0030:  leave.s    IL_0042
  }  // end .try
  finally
  {
    IL_0032:  ldloc.1
    IL_0033:  ldc.i4.0
    IL_0034:  ceq
    IL_0036:  stloc.3
    IL_0037:  ldloc.3
    IL_0038:  brtrue.s   IL_0041
    IL_003a:  ldloc.2
    IL_003b:  call       void [mscorlib]System.Threading.Monitor::Exit(object)
    IL_0040:  nop
    IL_0041:  endfinally
  }  // end handler
  IL_0042:  nop
  IL_0043:  nop
  IL_0044:  ldloc.0
  IL_0045:  ldc.i4.1
  IL_0046:  add
  IL_0047:  stloc.0
  IL_0048:  ldloc.0
  IL_0049:  ldc.i4     0x3e8
  IL_004e:  clt
  IL_0050:  stloc.3
  IL_0051:  ldloc.3
  IL_0052:  brtrue.s   IL_0005
  IL_0054:  ret
} // end of method '<>c__DisplayClass6'::'<TestMethod>b__3'
```

IL(Intermediate Language) koduna
baktığımızda **Monitor.Enter** ve **Monitor.Exit** metod çağrılarının gerçekleştirildiği

görülmektedir. üstelik **lock** ifadesi içerisine alınan kod kısmı için arka planda bir **try…finally** bloğu oluşturulmuştur. **Finally** bloğunda gerçekleştirilen **Monitor.Exit** çağrısı, tahmin edileceği üzere her ne olursa olsun devreye girecektir. Tabi ki **Monitor** tipinin bilinçli olarak kullanılması gerektiği durumlarda söz konusu olabilir. Bu ve diğer durumları ilerleyen yazılarımızda ele almaya çalışıyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

SynchronizationPrimitives.rar (22,26 kb) **[örnek Visual Studio 2010 Ultimate sürümünde geliştirilmiş ve test edilmiştir]**

TPL ve Shared Data Isolation (2010-12-19T00:35:00)

tpl,task parallel library,shared data isolation,parallel programming,c# 4.0,visual studio 2010 ultimate,

Merhaba Arkadaşlar,

Sanıyorum ki, **“Bir elin nesi var, iki elin sesi var”** deyimini bilmeyen yoktur. Bir matematikçi olarak tüme varım yaparsam, n tane elin çok daha sesli olduğunu ispat etmek isteyebilirim. Ne varki dünya kupasındaki n tane elin tuttuğu Vuvuzela’ ların çıkarttığı sesi düşününce, bu teoremden hemen vazgeçebilirim de. Neyseki konumuz bu değil. Konumuz paralel kütüphanede, paylaşımlı verileri nasıl ele alabileceğimiz.

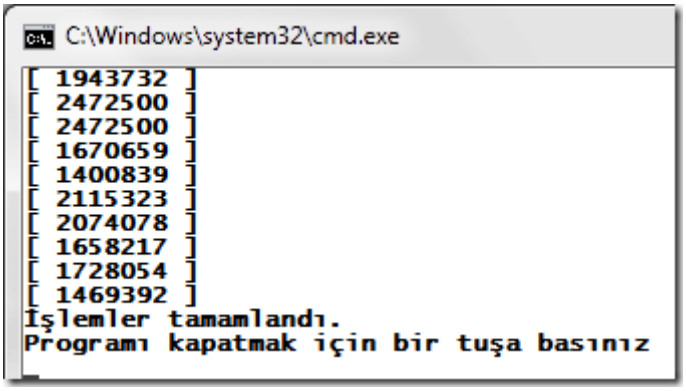
Task Parallel Library alt yapısını kullanarak geliştirdiğimiz paralel kodlarda önem arz eden konulardan birisi de, paylaşılan verilerin hesaplamalara katıldığı durumlardaki sonuç tutarlılıklarının nasıl sağlanacağıdır. Bunun bilinen bir kaç yolu vardır. Aslında bir tanesi ve en basiti kodu tamamen senkron olarak geliştirmektir. Yani paralel çalıştırmak gibi bir maceraya hiç girmemektir. Diğer bir yol ise **Task** örnekleri içerisinde ele alınan paylaşılmış verilerin izole edilerek kullanılmasıdır. Aslında durumu daha kolay bir şekilde analiz edebilmek için önce sorunu masaya yatıralım. Bu amaçla aşağıdaki kod parçasını göz önüne alarak ilerleyebiliriz.

```
using System;
using System.Threading.Tasks;

namespace SharedDataScenarios
{
   class Program
   {
      static void Main(string[] args)
      {
```

```
            for (int i = 0; i < 10; i++)
            {
               TestMethod();
            }

            Console.WriteLine("İşlemler tamamlandı.\nProgramı kapatmak için bir tuşa
basınız");
            Console.ReadLine();
        }

        private static void TestMethod()
        {
          Plane f16 = new Plane();
           Task[] tasks = new Task[5];

            for (int i = 0; i < 5; i++)
            {
               tasks[i] = new Task(() =>
              {
                  for (int j = 0; j < 1000; j++)
                  {
                     f16.Altitude += j - 5;
                  }
               }
               );
               tasks[i].Start();
            }

            Task.WaitAll(tasks);

            Console.WriteLine("[ {0} ]", f16.Altitude);
        }
    }

    class Plane
    {
       public int Altitude { get; set; }
    }
}
```

örnek kod parçasında **Altitude** isimli **int** tipinden **özelliği(Property)** olan **Plane** isimli bir sınıf yer almaktadır. **TestMethod** içerisinde ise bu sınıfa ait **f16** isimli bir nesne örneği kullanılmaktadır. **TestMethod** içerisinde **5 adet Task** nesnesi örneklenmektedir. Bu örneklere ait lambda ifadelerinde ise, **f16** örneği üzerinden ulaşılan **Altitude** özelliğinin değeri **1000** kere **5** birim arttırılmaktadır. Bu işlemi **5** farklı Task örneğinin yaptığı

unutulmamalıdır. **Main** metodu içerisinde ise **TestMethod** isimli fonksiyonun arka arkaya **10** defa çalıştırıldığı görülmektedir. Buradaki amaç, **10** farklı denemenin sonuçlarını irdelemektir. Nitekim her çalıştırılışta farklı sonuçlar alma ihtimalimiz çok yüksektir. Aşağıdaki ekran görüntüsünde olduğu gibi.



Aslında aynı süreç **10** kere çalıştırılmış ve hemen her seferinde **Plane** nesne örneğinin **Altitude** değeri farklı hesap edilmiştir. Oysaki başlangıçta **0** olan bu değer bir **Task**örneği içerisinde **1000** defa **5** birim arttırılmaktadır. Bununda **5** farklı **Task** örneği ile yapıldığı düşünüldüğünde toplamda elde edilen sonuçların aslında her deneme için aynı olması beklenmektedir. örnek çıktıda ise sadece **2472500** için iki hesaplamanın aynı olduğu görülmektedir. Sorun ne olabilir?

Paralel olarak başlatılan **Task** örneklerine ait kod blokları, işlemcinin durumuna göre farklı zamanlarda devreye girer ve yürütülürler. Buradaki zaman farkları çok küçük birimlere denk gelse de, örnekteki gibi ortak olarak paylaşılan veriler söz konusu olduğunda beklenmeyen sonuçların üretilmesine neden olabilirler. Geliştirilen örnek düşünüldüğünde; çalışma zamanında herhangibir t anındaki j değerleri, her bir Task için farklı olabilir. Yani **Task 1**, **500** nolu **j** değerini işlemekteyken, **Task 2** halen **345nci** değerinde olabilir. Dolayısıyla ortak olarak kullanılan **Altitude** değeri, her denemede farklı artımlarla değiştirilebilmektedir. Peki nasıl bir çözüm uygulanabilir?

Aslında her bir **Task** örneğinin kendi kapsamında ele alacağı bir değişken ile bu sorun çözülebilir. Ancak ek olarak, en sonda kümülatif bir hesaplama yapılması ve işlerin birleştirilmesi de gerekmektedir. Bunu aşağıdaki örnek kod parçasında daha net bir şekilde görebiliriz.

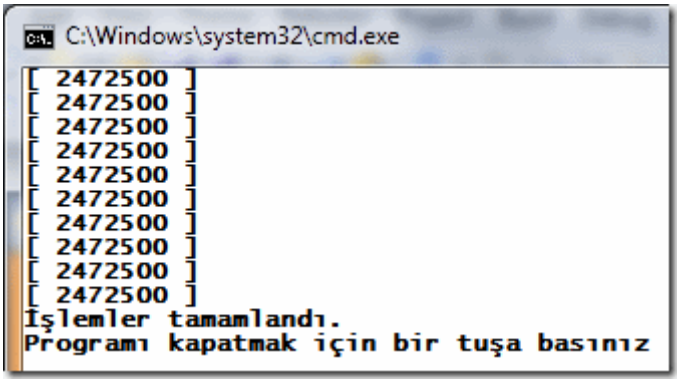
```
using System;
using System.Threading.Tasks;

namespace SharedDataScenarios
{
   class Program
   {
      static void Main(string[] args)
      {
```

```
        for (int i = 0; i < 10; i++)
        {
            TestMethod();
        }

        Console.WriteLine("İşlemler tamamlandı.\nProgramı kapatmak için bir tuşa
basınız");
        Console.ReadLine();
    }

    private static void TestMethod()
    {
        Plane f16 = new Plane();
        Task<int>[] tasks = new Task<int>[5];

        for (int i = 0; i < 5; i++)
        {
            tasks[i] = new Task<int>((os) =>
            {
                int currentAltitude = (int)os;

                for (int j = 0; j < 1000; j++)
                {
                    currentAltitude += j - 5;
                }

                return currentAltitude;
            },f16.Altitude
            );
            tasks[i].Start();
        }

        Task.WaitAll(tasks);

        for (int i = 0; i < tasks.Length; i++)
        {
            f16.Altitude += tasks[i].Result;
        }

        Console.WriteLine("[ {0} ]", f16.Altitude);
    }
}

class Plane
{
```

```
        public int Altitude { get; set; }
    }
}
```

Kodda, **Task** örnekleri oluşturulurken, her birinin kendi **Altitude** değişkeni ile çalışması için gerekli düzenlemeler yapılmıştır. Dikkat edileceği üzere **Task** sınıfına ait **yapıcı metod(Constructor)** ikinci bir parametre daha almaktadır. Bu parametre, lambda ifadesi içerisine taşınacak olan bir **Object State** değişkenini ifade etmektedir. Dolayısıyla bir **Task**örneklenirken, kendi kapsamında bir **Altitude** değişkeni ile çalışacaktır.

Tabiki yerel olarak ele alına bu değişkenlerin kodun dışarısında tekrardan bütünleştirilmesi gerekeceğinden **Task** örneklerinden birer sonuç döndürülmesi gerekmiştir. Söz konusu dönüş değerleri daha sonradan toplanmış ve aşağıdaki çalışma zamanı çıktısının oluşması sağlanmıştır.



Görüldüğü üzere 10 denemenin her birisinde **aynı** sonuç elde edilmiştir.

TLS(Thread Local Storage)

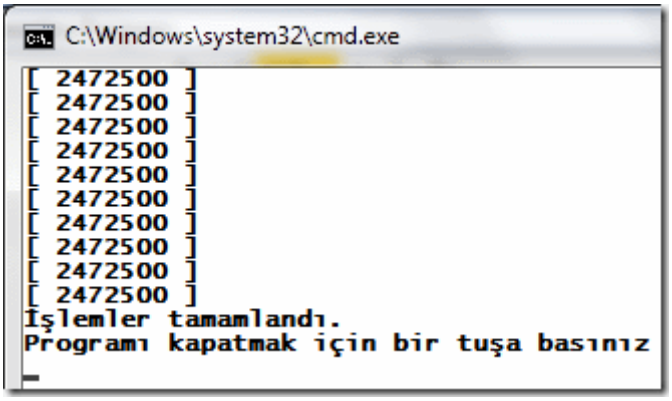
Aslında yukarıda geliştirilen kod parçasında **.Net çalışma zamanının** herhangibir zorlayıcılığı olmadığını ifade edebiliriz. Bu zorlayıcılık içinse, **Thread Local Storage** isimli özel alanlardan yararlanabiliriz. Bu depoları birden fazla iş parçası için ayrıştırılmış özel veri alanları olarak düşünülebiliriz. **TLS** vakasına göre, her bir **Thread**kendisine ait yerel depolama alanına sahiptir. Bu durumda herhangibir iş parçası, bir diğerinin **TLS** bölgesine müdahalede bulunamaz. Bir başka deyişle diğer bir iş parçasının izole edilmiş veri bölgesini okuyamaz veya yazamaz. Söz konusu alanlar, yönetimli **kod tarafında(Managed Code) ThreadLocal** sınıfı yardımıyla ele alınabilirler. Biraz önce geliştirdiğimiz örnek kod parçasında **TLS** kullanımını aşağıdaki gibi gerçekleştirebiliriz.

```
using System;
using System.Threading.Tasks;
using System.Threading;
```

```
namespace SharedDataScenarios
{
   class Program
   {
      static void Main(string[] args)
      {
         for (int i = 0; i < 10; i++)
         {
            TestMethod();
         }

         Console.WriteLine("İşlemler tamamlandı.\nProgramı kapatmak için bir tuşa
basınız");
         Console.ReadLine();
      }

      private static void TestMethod()
      {
         Plane f16 = new Plane();
         Task<int>[] tasks = new Task<int>[5];
         ThreadLocal<int> local = new ThreadLocal<int>();

         for (int i = 0; i < 5; i++)
         {
            tasks[i] = new Task<int>((os) =>
            {
               local.Value = (int)os;

               for (int j = 0; j < 1000; j++)
               {
                  local.Value += j - 5;
               }
               return local.Value;
            },f16.Altitude
            );
            tasks[i].Start();
         }

         Task.WaitAll(tasks);

         for (int i = 0; i < tasks.Length; i++)
         {
            f16.Altitude += tasks[i].Result;
         }
```

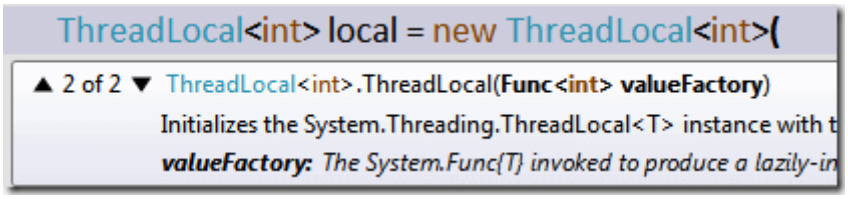
```
            Console.WriteLine("[ {0} ]", f16.Altitude);
        }
    }

    class Plane
    {
        public int Altitude { get; set; }
    }
}
```

Bu kez **System.Threading isim alanı(Namespace)** altında yer alan **ThreadLocal<T>** tipinden bir örnek oluşturulmuş ve lambda ifadesi içerisindeki **Object State** atamasında kullanılmıştır. İşlemlerin tamamı bu örneğe ait **Value** özelliği üzerinden yapılmaktadır. Sonuçlar bir önceki ile aynı olacaktır.



ThreadLocal<T> Lazy Initialization Kullanımı ve Tuzak

ThreadLocal<T> tipinin kullanımında değerlendirilebilecek bir versiyon daha bulunmaktadır. Aşırı yüklenmiş olan yapıcı metod, **Func<int>** tipinden bir **tesmilci(Delegate)** almaktadır.

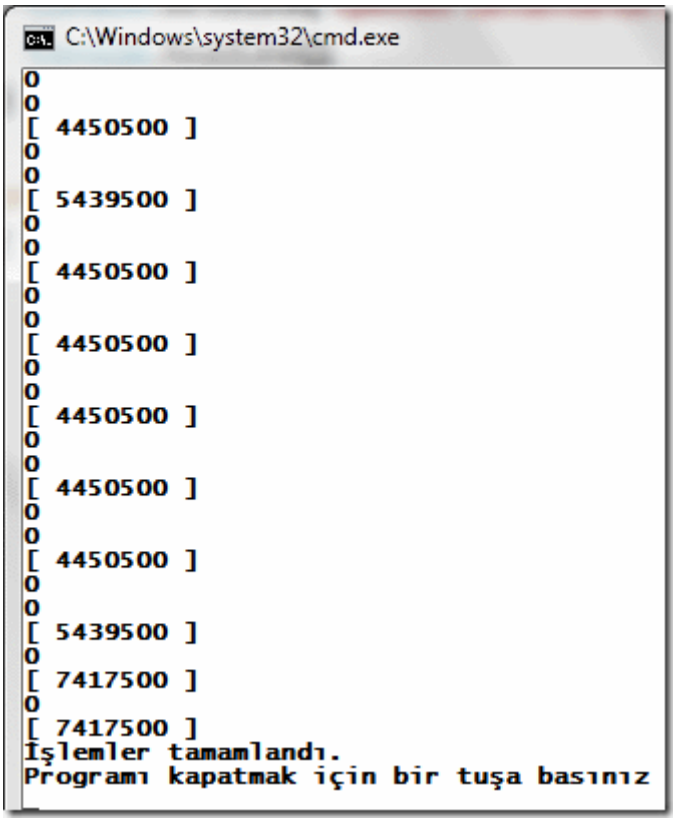


Bu versiyonda **Lazy Initialization** söz konusudur. Bu sebepten çok dikkatli olunması gerekmektedir. **Func<T>** temsilcisi, izole edilmiş veri değişkeninin oluşturulması aşamasında devreye girecek bloğu işaret etmektedir. Lakin bu blok, **ThreadLocal<T>** örneğinin **Value** özelliği çağırılıncaya kadar devreye girmeyecektir. İşte bu sebepten yazımızda ele aldığımız senaryo için yine farklı sonuçların elde edilmesi söz konusu olabilir. Kodumuzu buna göre aşağıdaki gibi geliştirdiğimizi düşünelim.

```
using System;
using System.Threading.Tasks;
using System.Threading;

namespace SharedDataScenarios
{
   class Program
   {
      static void Main(string[] args)
      {
         for (int i = 0; i < 10; i++)
         {
            TestMethod();
         }

         Console.WriteLine("İşlemler tamamlandı.\nProgramı kapatmak için bir tuşa
basınız");
         Console.ReadLine();
      }

      private static void TestMethod()
      {
         Plane f16 = new Plane();
         Task<int>[] tasks = new Task<int>[5];
         ThreadLocal<int> local = new ThreadLocal<int>(
            () =>
            {
               Console.WriteLine("{0}",f16.Altitude);
               return f16.Altitude;
            }
         );

         for (int i = 0; i < 5; i++)
         {
            tasks[i] = new Task<int>(() =>
            {
               for (int j = 0; j < 1000; j++)
               {
                  local.Value += j - 5;
               }
               return local.Value;
            }
            );
            tasks[i].Start();
         }
```

```
            Task.WaitAll(tasks);

            for (int i = 0; i < tasks.Length; i++)
            {
               f16.Altitude += tasks[i].Result;
            }

            Console.WriteLine("[ {0} ]", f16.Altitude);
         }
      }

      class Plane
      {
         public int Altitude { get; set; }
      }
}
```

Dikkat edileceği üzere **Task** örneklerinin kullanacağı **Altitude** değerinin, izole edilmiş yerel değişkene set edilmesi işlemi, **ThreadLocal** örneklemesi yapılırken **Func<T>** temsilcisinin işaret ettiği blok içerisinde yapılmaktadır. Bu nedenle Object State kullanımına gerek kalınmamıştır. Ne varki çalışma zamanı sonuçları beklediğimiz gibi olmayacaktır.



Herşeyden önce 10 denemenin çoğu kendi aralarında farklı sonuçlar vermiş ve hatta bir önceki örnekteki ile alakası olmayan çıktılar üretilmiştir. Diğer

yandan **ThreadLocal<T>**örneklemelerinin hemen her bir deneme için*(nitekim sonlarda bire düşmüştür ve her çalışmada bu değişebilir)* **2** kez çağırıldığı görülmektedir. Bunun sebebi ise, örneğin geliştirildiği makinenin **çift çekirdekli** olmasıdır.
öyleki **TLS** tekniğinde **Task** örnekleri değil **Thread'** ler söz konusudur ve makine çift çekirdekli olduğundan aslında çalışma zamanında iki **Thread** yürümektedir ki bunlarda her **5 Task** örneğini paylaşır. Piuvvvvvv!!!

Bu yazımızda biraz karmaşık ve anlaşılması zor olan bir konuyu gündeme getirmeye çalıştık. Aslında bu noktada ele alacağımız daha bir sürü vaka var. Fakat paralel programlamanın, **Yönetimli Kodda(Managed Code)** bile olsa çok dikkatli uygulanması gerektiğini bir kere daha gördük. özellikle son örneğimize göre **ThreadLocal** kullanımı sırasında çok dikkatli olunması ve testlerin mutlaka iyi bir şekilde yapılması gerektiğini ortaya koyduk. Her an biz tuzağa düşebiliriz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

SharedDataScenarios_.rar (25,91 kb)**[örnek Visual Studio 2010 Ultimate Sürümünde Geliştirilmiş ve Test Edilmiştir]**

Paralel Programlamada İstisna Yönetimi (2010-12-19T00:30:00)

parallel programming,tpl,task parallel library,exception handling,c# 4.0,visual studio 2010 ultimate,

Merhaba Arkadaşlar,

Balık tutmak farklı bir hobidir. özellikle olta ile balık yakalamaktan büyük keyif alanlar vardır. *(Hatta laf aramızda, şirkette yanımda oturan çalışma arkadaşımın çekmecesinde, bir olta takımı var)*

çoğu zaman Haliç köprüsü gibi alanlarda yandaki resimde olduğu gibi bu işin sevdalılarını görebiliriz. Kimisi sabahın erken saatlerinde gelip, akşamın geç saatlerine kadar burada olta sallar ve "Rastgele" der.

Benim ne yazık ki bu tip bir hobim yok. Hatta bu günkü yazımızda sizlere **balık yakalamayı** da anlatacak değilim. Onun yerine **Task Parallel Library** için **istisna yakalama** vakaları üzerinde duracağım.

Task örneklerinin kullanıldığı senaryolarda, bloklar içerisinde yer alan işlevselliklerin doğurabileceği çalışma zamanı istisnalarını ele almak, son derece önemlidir. Nitekim paralel çalışmakta olan blokların beklenmedik bir şekilde sonlandırılması söz konusudur. İşte bu yazımızda **Task** örnekleri içerisinde oluşabilecek istisnaların nasıl ele alınabileceğini incemelye çalışıyor olacağız.

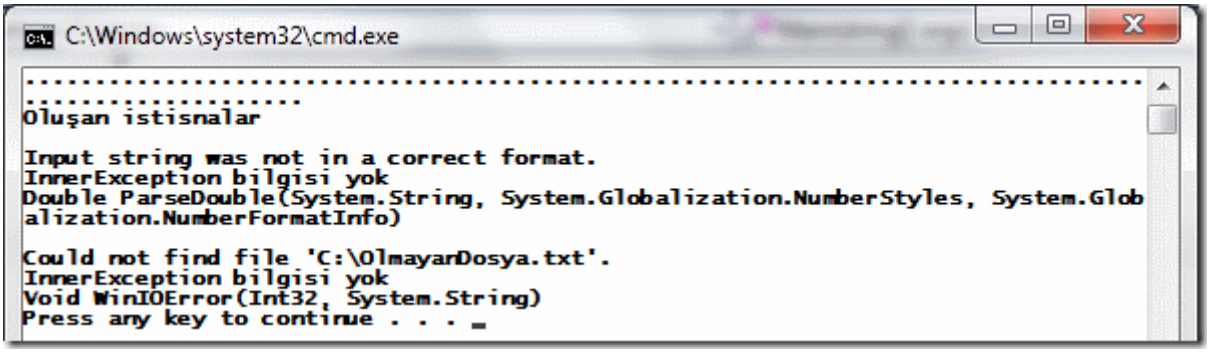
Wait, WaitAll, WaitAny Tetikleyicileri

Bir veya daha fazla **Task** örneği tarafından başlatılan paralel işleyişlerde, bekletme metodlarının çağırılması halinde, ortama fırlayabilecek **Exception** örneklerinin yakalanması mümkündür. Aşağıdaki kod parçasında 3 farklı **Task** bloğu için bir istisna senaryosu ele alınmaya çalışılmıştır.

```
using System;
using System.IO;
using System.Linq;
using System.Threading.Tasks;

namespace HandlingExcpetions
{
   class Program
   {
      static void Main(string[] args)
      {
         Task task1 = new Task(() =>
            {
               File.Open("C:\\OlmayanDosya.txt", FileMode.Open);
            }
         );
         Task task2 = new Task(() =>
            {
               string number = "oniki";
               double point = Convert.ToDouble(number);
            }
         );
         Task task3 = new Task(() =>
            {
               for (int i = 0; i < 100; i++)
               {
                  i++;
                  i--;
                  i *= 1;
                  Console.Write(".");
               }
            }
         );

         task3.Start();
         task2.Start();
         task1.Start();
```

```
        try
        {
           Task.WaitAll(task1, task2, task3);
        }
        catch (AggregateException agrExcp)
        {
           Console.WriteLine("\nOluşan istisnalar");
           var excpInfos = from e in agrExcp.InnerExceptions
                           select new
                           {
                              e.TargetSite,
                              e.Message,
                              InnerException=e.InnerException!=null?e.InnerException.Message:
"InnerException bilgisi yok"
                           };
           foreach (var excp in excpInfos)
           {
              Console.WriteLine("\n{0}\n{1}\n{2}",excp.Message,excp.InnerException,exc
p.TargetSite);
           }
        }
     }
  }
}
```

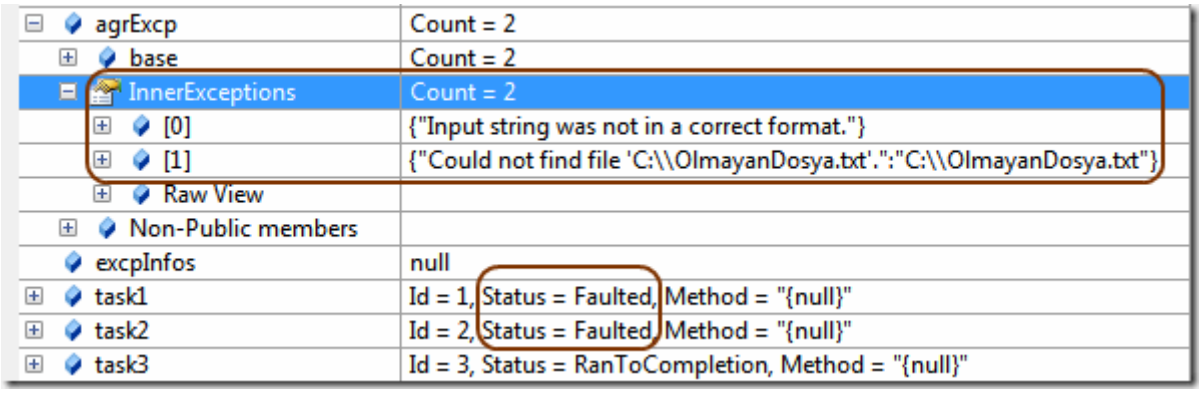
örnek kod parçasında 3 farklı Task örneği oluşturulduğu görülmektedir. **Task1** içerisinde sistemde olmadığı düşünülen bir dosya açılmaya çalışılmaktadır. **Task 2** ile alakalı blok içerisinde ise, metinsel bir ifadenin sayısal dönüştürülmesi söz konusudur. Task 3 ile ilişkili kod bloğunda ise herhangibir istisna durumu söz konusu değildir. örnek uygulamayı çalıştırdığımızda aşağıdaki ekran görüntüsünde yer alan çalışma zamanı çıktısı ile karşılaşırız.



Burada dikkat edilmesi gereken noktalardan birisi, **Task 1** ve **Task 2** içerisinde oluşan istisnaların **Task 3**' ün başlattığı bloğun çalışmasını etkilememesidir. Diğer yandan oluşan

istisnaların tamamı, **AggregateException** tipinin **InnerExceptions** özelliği içerisinde toplanmaktadır.

Aslında **try…catch** bloğu içerisine yer alan **Task.WaitAll** metodu çağrısı ile, tüm **Task** örneklerinin işleyişleri bitene kadar, bu işleyişlerin sahibi olan **Main Thread'** in duraksatılması sağlanmaktadır. **WaitAll** metodunun **try…catch** bloğu içerisine olması nedeniyle de, tamamlanması beklenen **Task** bloklarında oluşan istisnalar, **AggregateException** tarafından toplanmatadır. Eğer **catch** bloğu içerisinde **breakpoint** konularak durulur ve yerel değişkenlere bakılırsa, aşağıdaki ekran görüntüsünde yer alan sonuçlar ile karşılaşılır.



Burada dikkat edilmesi gereken en önemli noktalardan birisi de, **Task** örneklerinin **Status** özelliklerinin değerleridir. Dikkat edileceği üzere **task1** ve **task2** isimli örnekler **Faulted** durumunda kalmışlardır. Bu değerler, uygulamalarda olup biten herşeyi tutan **log** mekanizmaları için veya iş sürecinin akan diğer kısımları için önemlidir. çok doğal olarak **task3** örneğinin durumu**RanToCompletion** olarak kalmıştır. Yani başarılı bir şekilde işleyişini tamamlamıştır. Gelelim diğer bir mevzuya…

AggregateException Nesne örneğine üzerinden Handle Metodunun Kullanılması

AggregateException nesne örneği üzerinden erişilebilen **Handle** metodu ile, **Task** örneklerine ait bloklar içerisinde oluşacak istisnalar arasında dolaşılabilmektedir. özellikle **n** sayıda **Task**bloğunun takip edildiği durumlarda, beklenen bir istisnanın oluşması halinde nasıl hareket edileceğine karar vermek için kullanılabilecek yollardan birisidir.

Bu metod parametre olarak **Func<Exception,bool>** tipinden bir **Temsilci(Delegate)** almaktadır. Buna göre hangi istisnanın ele alınmak istediği ilk parametre ile belirtilmektedir. Diğer yandan söz konusu temsilci geriye **bool** değer döndürecek şekilde ayarlanmıştır. Bu değer spesifik olarak belirlenen istisnanın/istisnaların ele alınması halinde true olmalıdır. Handle kullanımını daha iyi kavrayabilmek adına aşağıdaki örnek kod parçasını göz önüne alalım.

```
using System;
using System.IO;
using System.Linq;
using System.Threading.Tasks;

namespace HandlingExcpetions
{
   class Program
   {
      static void Main(string[] args)
      {
         Task task1 = new Task(() =>
         {
            File.Open("C:\\OlmayanDosya.txt", FileMode.Open);
         }
         );
         Task task2 = new Task(() =>
         {
            string number = "oniki";
            double point = Convert.ToDouble(number);
         }
         );
         Task task3 = new Task(() =>
         {
            for (int i = 0; i < 100; i++)
            {
               i++;
               i--;
               i *= 1;
               Console.Write(".");
            }
         }
         );

         task3.Start();
         task2.Start();
         task1.Start();

         try
         {
            Task.WaitAll(task1, task2, task3);
         }
         catch (AggregateException agrExcp)
         {
            agrExcp.Handle(e =>
```

```
            {
                if (e is FileNotFoundException)
                {
                    Console.WriteLine("Dosya bulunamadı");
                    return true;
                }
                else if (e is FormatException)
                {
                    Console.WriteLine("Dönüştürme hatası");
                    return true;
                }
                else
                  return false;
            }
        );
    }
  }
}
```

Bir önceki örnekte yer alan senaryonun aynısı söz konusudur. Ancak bu kez **catch** bloğu içerisinde **Handle** metodu kullanılmıştır. **Handle** ile işaret edilen blokta, e ile temsil edilen referansın olası değerleri kontrol
edilmektedir. **FileNotFoundException** veya **FormatException** olması halleri ele alınmıştır. **true** döndürdüğümüz yerlerde, oluşan **istisnai** durumların geliştirici tarafından ele alındığını ifade edilmektedir. Program kodunun çalışma zamanı çıktısı aşağıdaki gibi olacaktır.

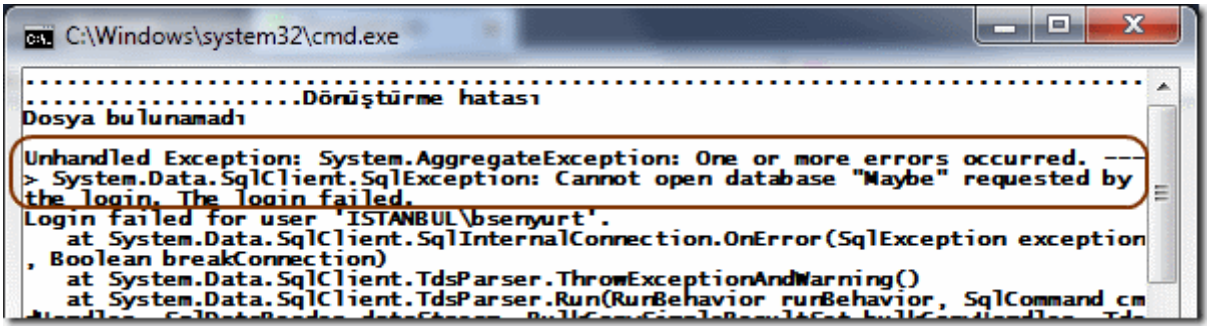
Tabi burada dikkat edilmesi gereken ayrı bir durum daha vardır.
Olaya **4ncü** bir **Task** örneğini daha kattığımızı düşünelim ve kodumuzu buna göre aşağıdaki gibi düzenleyelim.

```
using System;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using System.Data.SqlClient;

namespace HandlingExcpetions
{
   class Program
```

```
{
   static void Main(string[] args)
   {
      Task task1 = new Task(() =>
      {
         File.Open("C:\\OlmayanDosya.txt", FileMode.Open);
      }
      );
      Task task2 = new Task(() =>
      {
         string number = "oniki";
         double point = Convert.ToDouble(number);
      }
      );
      Task task3 = new Task(() =>
      {
         for (int i = 0; i < 100; i++)
         {
            i++;
            i--;
            i *= 1;
            Console.Write(".");
         }
      }
      );
      Task task4 = new Task(() =>
         {
            SqlConnection conn = new SqlConnection("data
source=.;database=Maybe;integrated security=sspi");
            conn.Open();
         }
      );

      task3.Start();
      task2.Start();
      task1.Start();
      task4.Start();

      try
      {
         Task.WaitAll(task1, task2, task3,task4);
      }
      catch (AggregateException agrExcp)
      {
         agrExcp.Handle(e =>
```

```
                {
                    if (e is FileNotFoundException)
                    {
                        Console.WriteLine("Dosya bulunamadı");
                        return true;
                    }
                    else if (e is FormatException)
                    {
                        Console.WriteLine("Dönüştürme hatası");
                        return true;
                    }
                    else
                        return false;
                }
            );
        }
    }
  }
}
```

örneği çalıştırdığımızda aşağıdaki gibi bir sonuçla karşılaşırız.



Dikkat edileceği üzere uygulama istem dışı bir şekilde sonlanmıştır. **Task 4** nesne örneğine ait kod bloğunda, var olmayan bir veritabanına doğru bağlantı oluşturulmaya çalışılmaktadır. Bunun doğal sonucu bir **SqlException** örneğidir. Ancak **catch** bloğu içerisinde kullandığımız **Handle** metodunda söz konusu istisna ele alınmadığı için kod doğrudan **else** bloğuna girmiştir. Bir başka deyişle ele alınmamış bir istisna söz konusudur. özellike **Handle** metodunda bu duruma dikkat edilmesi gerekir.

IsFaulted özelliğine Bakmak

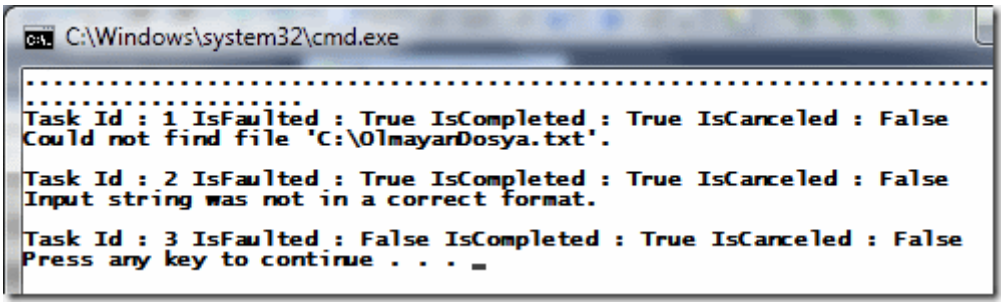
Bazı durumlarda istisnaların **catch** bloğu içerisinde, **AggregateException** üzerinden ele alınması yerine, **Task** örneklerine ait **IsFaulted** özelliklerine bakılaraktan da ilerlenebilir. **Task** örnekleri üzerinden erişilebilen **IsFaulted**, **IsCompleted**, **IsCanceled** gibi özellikler sayesinde, Task' in

çalışma sonucu ile ilişkili bilgiler yakalanabilmektedir. IsFaulted özelliği de istisna senaryolarında değerlendirilebilir. Aşağıdaki kod parçasında bu durum ele alınmaktadır.

```
using System;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using System.Data.SqlClient;

namespace HandlingExcpetions
{
   class Program
   {
      static void Main(string[] args)
      {
         Task[] tasks={
         new Task(() =>
         {
            File.Open("C:\\OlmayanDosya.txt", FileMode.Open);
         }
         ),
        new Task(() =>
         {
            string number = "oniki";
            double point = Convert.ToDouble(number);
         }
         ),
         new Task(() =>
         {
            for (int i = 0; i < 100; i++)
            {
               i++;
               i--;
               i *= 1;
               Console.Write(".");
            }
         }
         )
         };

         foreach (Task t in tasks)
         {
            t.Start();
         }
```

```
            try
            {
               Task.WaitAll(tasks);
            }
            catch (AggregateException agrExcp)
            {
            }

           foreach (Task t in tasks)
            {
               Console.WriteLine("\nTask Id : {0} IsFaulted : {1} IsCompleted : {2}
IsCanceled : {3}",t.Id,t.IsFaulted,t.IsCompleted,t.IsCanceled);
               if (t.IsFaulted)
               {
                  foreach (Exception excp in t.Exception.InnerExceptions)
                  {
                     Console.WriteLine("{0}",excp.Message);
                  }
               }
            }
         }
      }
   }
```

Aynı senaryoyu bu kez **Task** tipinden örneklerden oluşan bir **Array** üzerinde ele almaktayız. Dikkat edileceği üzere **Task** örneklerinin her birinin **IsFaulted** özelliği kontrol edilmekte ve eğer **true**ise **InnerExceptions** özelliği ile belirtilen koleksiyona gidilerek, oluşan istisnai durumlara ait mesaj bilgilendirmeleri yapılmaktadır. örnekte **catch** bloğunda herhangibir işlem yapılmadığına dikkat edilmelidir. Uygulamayı çalıştırdığımızda aşağıdaki sonuçlar ile karşılaştığımızı görürüz.



Escalation Policy

Peki ya **trigger** görevini üstlenen **Wait**… metodlarından herhangibirini kullanmadığımızda ne olur?

Her ne kadar **CLR** ortamı, fırlatılan istisnaları yakalıyor olsa da, **trigger** metodlarının kullanılmadığı durumlarda bunun ne zaman olacağı kestirilemeyebilir. Dahası Wait metodlarının kullanılmak istenmediği durumlarda olabilir. İşte bu durumda **TaskScheduler** tipinin **UnobservedTaskException** olay metodunu kullanarak, söz konusu durumlar için devreye giren standart **ilke(Policy)**değiştirilebilir. Bir başka deyişle **Exception** yakalanma işleminde kendi yazdığımız olay metodunun devreye girmesini sağlayabiliriz. Ancak bu konuyu işleyebilmemiz için öncelikle**TaskScheduler** kavramını öğrenmemiz, kavramımız gerekmektedir. Bu konuyu ilerleyen yazılarımızda irdelemeye çalışıyor olacağız.

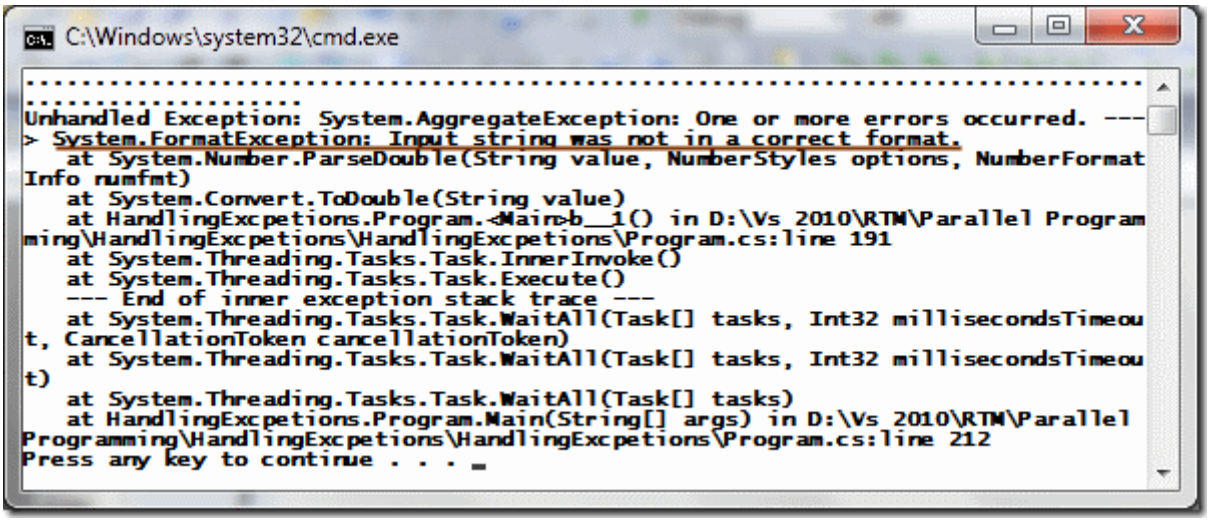
Elimizde Olta da, Misina da, Kurşun da Yok

Son olarak hiç bir **Exception** kontrolü yapmadığımızda ne olduğuna bir bakalım. Bu amaçla aşağıdaki kod parçasını göz önüne alabiliriz.

```
using System;
using System.IO;
using System.Linq;
using System.Threading.Tasks;
using System.Data.SqlClient;

namespace HandlingExcpetions
{
   class Program
   {
      static void Main(string[] args)
      {
         Task[] tasks ={
          new Task(() =>
          {
             File.Open("C:\\OlmayanDosya.txt", FileMode.Open);
          }
          ),
         new Task(() =>
          {
             string number = "oniki";
             double point = Convert.ToDouble(number);
          }
          ),
          new Task(() =>
          {
             for (int i = 0; i < 100; i++)
             {
                i++;
                i--;
```

```
                i *= 1;
                Console.Write(".");
            }
        }
        )
        };

        foreach (Task t in tasks)
        {
           t.Start();
        }
        Task.WaitAll(tasks);

        Console.WriteLine("Program sonu");
    }
  }
}
```

Burada herhangibir **try…catch** bloğu kullanılmamıştır. Ancak senaryomuza göre **Task'** lerden ortama fırlayan iki **Exception** söz konusudur. çalışma zamanına baktığımızda ise sadece bir tanesinin **CLR(Common Language Runtime)** tarafından yakalandığını ve uygulamanın da istem dışı sonlandırıldığını görürüz. üstelik sadece bir **Exception**mesajı yakalanmış ve uygulamanın son satırı bile çalıştırılmamıştır. Açıkçası uygulamanın bu şekilde istem dışı sonlanması zaten **CLR'** dan beklenen bir davranıştır.



Yapılan örneklerde gözden kaçırılmaması gereken bir husus daha vardır. **Task** bloklarında **Exception** oluşan noktalarda, kod bir sonraki satıra geçmeyecektir. Dolayısıyla, ardışıl olarak oluşan istisna durumları olsa bile, aynı **Task** örneği için sadece ilk **Exception** içeriğinin yakalanması söz konusudur.

Böylece geldik bir yazımızın daha sonuna. Bu yazımızda **Task Parallel Library** için **Exception** kontrol senaryolarını incelemeye çalıştık. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HandlingExcpetions.rar (22,91 kb) [örnek Visual Studio 2010 Ultimate Sürümünde Geliştirilmiş ve Test Edilmiştir]

Task Wait,WaitAll,WaitAny (2010-12-19T00:25:00)

c# 4.0,parallel programming,.net framework 4.0,visual studio 2010 ultimate,task parallel library,waiting tasks,

Merhaba Arkadaşlar,

Task Süreçlerinde Bilinçli Olarak Duraksatma başlıklı bir önceki yazımızda **CancellationToken.WaitHandle.WaitOne, Thread.Sleep** ve**Thread.SpinWait** metodlarından yararlanarak bir Task çalışmasının bekletme işlemlerinin nasıl yapılabileceğini incelemeye çalışmıştık.
özellikle **WaitOne** metodunun, **CancellationToken.WaitHandle** özeliği üzerinden çalıştırıldığını unutmayalım. Diğer yandan tüm bu teknikleri Task gövdesi içerisinde gerçekleştirmiştik. Bunun doğal sonucu olarakta yürütülmekte olan Task işlevlerinin duraksatılmasını sağlamıştık.

Ancak incelediğimiz bu teknikler dışında, **Task** nesne örnekleri veya **Task** sınıfı üzerinden kullanılabilecek farklı bekletme teknikleri de söz konusudur.

Aslında **Task** örnekleri üzerinden **Result** değerlerinin okunmaya çalışılması, söz konusu **Task'** in işleyişini tamamlayıncaya kadar, çağıran uygulamanın bekletilmesi anlamına gelmektedir. Fakat bunun dışında kullanılabilecek **Wait, WaitAll** ve **WaitAny** gibi metodlar söz konusudur. Tüm bu metodların kullanımında amaç, **Task** örneğini/örneklerini başlatan uygulamanın, belirtilen şartlar doğrultusunda bekletilmesini sağlamaktır. Bir önceki yazımızda incelediğimiz konular düşünüldüğünde bu önemli bir farktır. Bu seferki hedefimiz çağıran uygulamanın/metodun duraksatılmasıdır. Dilerseniz söz konusu modellere ait örnek uygulamalarımızı geliştirerek ilerleyelim.

Wait Tekniği

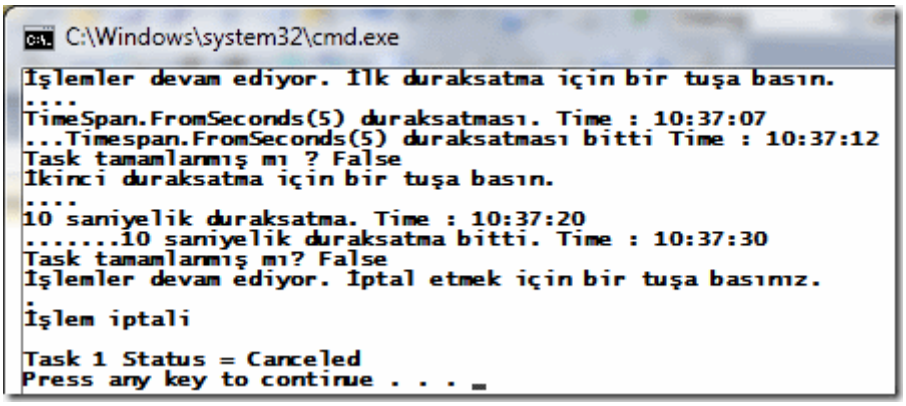
Bu modelde, **Task** nesne örneği üzerinden çağırılan **Wait** metodu ile, **Task** örneğini başlatan fonksiyon veya uygulamanın duraksatılması söz konusudur.
Aslında **Task** örneğini başlatan en azından uygulamanın ana **Thread'** idir.

Dolayısıyla **Wait**, **Main Thread'** in duraksatılmasına neden olmaktadır. Durumu daha açık bir şekilde kavrayabilmek adına aşağıdaki kod parçasını göz önüne alalım.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace WaitingScenarios2
{
   class Program
   {
      static void Main(string[] args)
      {
         IEnumerable<int> numbers = Enumerable.Range(0, 100000000);

         CancellationTokenSource tokenSource = new CancellationTokenSource();
         CancellationToken token = tokenSource.Token;

         Task startedTask = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers.Count(); i++)
            {
               token.ThrowIfCancellationRequested();
               i++;
               i--;
               i *= 2;
               Console.Write(".");
            }
         }
         , token);

         token.Register(() =>
         {
            Console.WriteLine("İşlem iptali");
         }
         );
         Console.WriteLine("İşlemler devam ediyor. İlk duraksatma için bir tuşa basın.");
         Console.ReadLine();

         Console.WriteLine("TimeSpan.FromSeconds(5) duraksatması. Time :
{0}",DateTime.Now.ToLongTimeString());
         bool waitStatus=startedTask.Wait(TimeSpan.FromSeconds(5));
         Console.WriteLine("Timespan.FromSeconds(5) duraksatması bitti Time :
```

```
{0}\nTask tamamlanmış mı ? {1}\nİkinci duraksatma için bir tuşa
basın.",DateTime.Now.ToLongTimeString(),waitStatus);
            Console.ReadLine();

            Console.WriteLine("10 saniyelik duraksatma. Time : {0}",
DateTime.Now.ToLongTimeString());
            waitStatus=startedTask.Wait(10000);
            Console.WriteLine("10 saniyelik duraksatma bitti. Time : {0}\nTask tamamlanmış
mı? {1}\nİşlemler devam ediyor. İptal etmek için bir tuşa
basınız.",DateTime.Now.ToLongTimeString(), waitStatus);
            Console.ReadLine();

            tokenSource.Cancel();

            Console.ReadLine();
            Console.WriteLine("Task 1 Status = {0}", startedTask.Status);

        }
    }
}
```

Tedbiri elden bırakmadık ve yine bir iptal işlemini işin içerisine kattık. Ancak odaklanacağımız nokta tabiki bu değil. Kodun iki farklı satırında **Wait** metodunun kullanıldığını görmekteyiz. İlk kullanımda **o anki andan itibaren 5 saniyelik bir duraksatma** gerçekleştiriyoruz. Diğer metod kullanımından ise **10000 mili saniyelik süre bildirimi yaparak 10 saniyelik bir duraksatma** icra ettirmekteyiz. **Wait** metodu **bool** tipinden değer döndürmektedir. **False** değer dönmesi, ilgili **Task** bloğunun herhangibir sebepten işyelişini tamamlamamış olması veya iptal edilmesi anlamına gelmektedir. Tahmin edileceği üzere **Task'** in başarılı bir şekilde tamamlanması sonucu bu değer **True** olacaktır. örneğimizi çalıştırdığımızda, aşağıdaki ekran görüntüsündekine benzer sonuçlar ile karşılaşabiliriz.



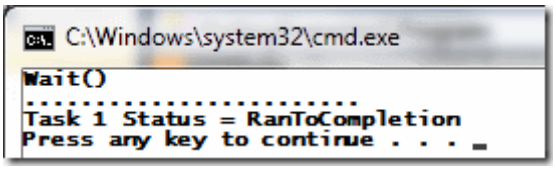
Dikkat edileceği üzere duraksatmalar da bekleyen aslında **Console** uygulamasının kendisidir. Yani **Main Thread'** dir. Ana uygulama, belirtilen süreler boyunca

duraksamıştır. Lakin, **Task** gövdesi içerisinden başlatılan for döngüsü, bu duraksatmalar sırasında işleyişine devam etmiştir.

Wait metodunun aşırı yüklenmiş versiyonlarına bakıldığında parametre almayan bir versiyonunun daha olduğu görülmektedir. Sıklıkla kullanılan bu versiyona göre ana uygulama, söz konusu **Task** örneği işleyişini tamamlayıncaya kadar duraksatılacaktır. **Wait()** kullanımına ilişkin örnek kod parçası aşağıdaki gibidir.

```
IEnumerable<int> numbers = Enumerable.Range(0, 10000000);

        Task startedTask = Task.Factory.StartNew(() =>
        {
           for (int i = 0; i < numbers.Count(); i++)
           {
              i++;
              i--;
              i *= 2;
              Console.Write(".");
           }
        }
        );

        Console.WriteLine("Wait()");
       startedTask.Wait();
        Console.WriteLine("\nTask 1 Status = {0}", startedTask.Status);
```

Değer aralığı bilinçli olarak küçültülmüştür. çok fazla beklememek için. Burada dikkat edilmesi gereken noktalardan birisi de, parametre almayan **Wait** metodunun geriye **bool** bir değerdöndürmeyişidir. **Task** örneğinin işlettiği kod parçasında herhangibir istisna oluşmadığı varsayıldığında, ana uygulmanın, işleyiş tamamlanıncaya kadar beklemesi söz konusudur.



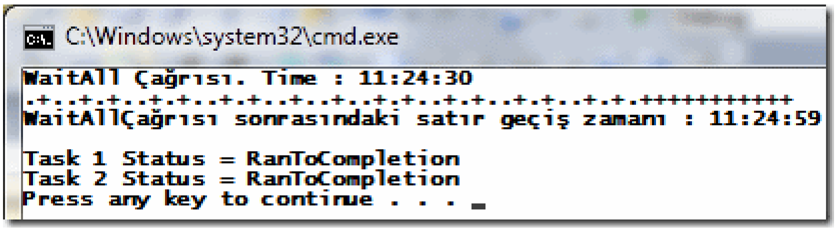
WaitAll Kullanımı

önceki örneklerimizde tek bir **Task** nesne örneği üzerinden, çağıran uygulamanın duraksatılması işlemi gerçekleştirilmiştir. Bir başka deyişle çağıran uygulama, tek bir **Task**nesne örneği için belirli/belirsiz süre bekletilmiştir. Ancak doğal olarak birden fazla **Task** örneğinin yer aldığı bir senaryoda, tüm **Task'** ler için ana uygulamanın bekletilmesi de istenebilir. Hatta bu teknikte temel amaç, birden fazla **Task** örneğinin işleyişi tamamlanıncaya kodun ilerlememesidir. Bu durumda **Task** sınıfının **static WaitAll** metodundan yararlanılabilir. İşte örnek kod parçamız.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace WaitingScenarios2
{
   class Program
   {
      static void Main(string[] args)
      {
         IEnumerable<int> numbers = Enumerable.Range(0, 100000000);
         IEnumerable<int> numbers2 = Enumerable.Range(0, 150000000);

         Task startedTask = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers.Count(); i++)
            {
               i++;
               i--;
               i *= 2;
               Console.Write(".");
            }
         }
         );

         Task startedTask2 = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers2.Count(); i++)
            {
               i++;
               i--;
               i *= 2;
               Console.Write("+");
            }
         }
         );

         Console.WriteLine("WaitAll çağrısı. Time :
{0}",DateTime.Now.ToLongTimeString());
         Task.WaitAll(startedTask, startedTask2);
         Console.WriteLine("\nWaitAllçağrısı sonrasındaki satır geçiş zamanı :
{0}",DateTime.Now.ToLongTimeString());
         Console.WriteLine("\nTask 1 Status = {0}\nTask 2 Status = {0}",
```

```
startedTask.Status,startedTask2.Status);
        }
    }
}
```

örnekte iki **Task** örneği üretilmiş ve başlatılmıştır. Sonrasında ise **Task** sınıfı üzerinden **static WaitAll** metoduna parametre olarak aktarılmışlardır. Bunun sonucu olarak**ana uyglamaya ait iş parçası(Main Thread),** ilgili **Task** işleyişleri tamamlanıncaya kadar duraksatılmaktadır. örnek kod parçasında **Cancel** işlemi ele alınmamıştır. Ancak görsel bir uygulama söz konusu olduğunda, kullanıcının **Task** örneklerinin işleyişleri devam ederken iptal isteğinin gönderilebileceği de göz önüne alınmalıdır. örnek kod parçasının çalışma zamanı çıktısı aşağıdaki gibi olacaktır.



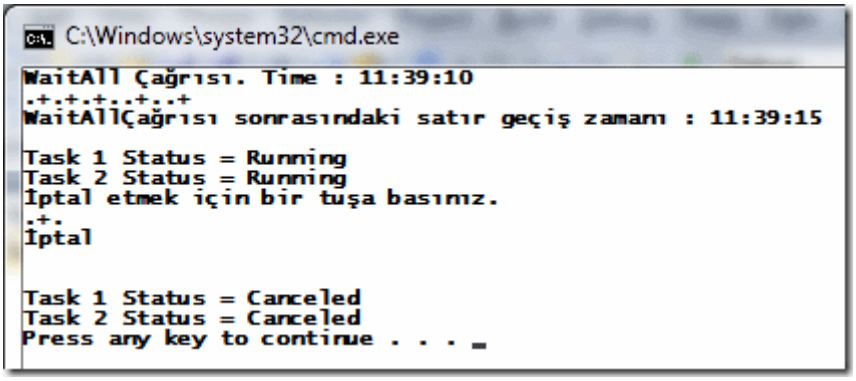
WaitAll metodunda istenirse sürede belirtilebilir. Bir başka deyişle parametre olarak gelen bir **Task** kümesinin çalışması sonucu, yürütücü uygulamanın belirli süre duraksatılması sağlanabilir. Bu durumu analiz etmek için aşağıdaki kod parçasını göz önüne alabiliriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace WaitingScenarios2
{
    class Program
    {
        static void Main(string[] args)
        {
            CancellationTokenSource tokenSource = new CancellationTokenSource();
            CancellationToken token = tokenSource.Token;

            IEnumerable<int> numbers = Enumerable.Range(0, 100000000);
            IEnumerable<int> numbers2 = Enumerable.Range(0, 150000000);

            Task startedTask = Task.Factory.StartNew(() =>
            {
                for (int i = 0; i < numbers.Count(); i++)
                {
```

```
            token.ThrowIfCancellationRequested();
            i++;
            i--;
            i *= 2;
            Console.Write(".");
        }
    }
    ,token);

    Task startedTask2 = Task.Factory.StartNew(() =>
    {
        for (int i = 0; i < numbers2.Count(); i++)
        {
            token.ThrowIfCancellationRequested();
            i++;
            i--;
            i *= 2;
            Console.Write("+");
        }
    }
    ,token
    );

    token.Register(() =>
        {
            Console.WriteLine("İptal");
        }
    );

    Console.WriteLine("WaitAll çağrısı. Time : {0}",
DateTime.Now.ToLongTimeString());
    Task[] tasks = { startedTask, startedTask2 };
    Task.WaitAll(tasks,5000);
    Console.WriteLine("\nWaitAllçağrısı sonrasındaki satır geçiş zamanı : {0}",
DateTime.Now.ToLongTimeString());
    Console.WriteLine("\nTask 1 Status = {0}\nTask 2 Status = {0}\nİptal etmek için
bir tuşa basınız.", startedTask.Status, startedTask2.Status);
    Console.ReadLine();
    tokenSource.Cancel();
    Console.ReadLine();

    Console.WriteLine("\nTask 1 Status = {0}\nTask 2 Status = {0}",
startedTask.Status, startedTask2.Status);
  }
```

```
    }
}
```

örnekte WaitAll metoduna tasks isimli Task[] dizisi aktarılmış ve 5000 milisaniyelik duraksatma değeri verilmiştir. Ayrıca iptal işlemi de yapılabilmektedir. örnek bir çalışma zamanı çıktısı aşağıdaki gibidir.



Dikkat edileceği üzere **WaitAll** çağrısı sonrasındaki satıra geçiş süresi, **5 saniyelik bir duraksamaya** neden olmuştur. Tabi ki bu aralıkta **Task** örneklerinin gövdeleri işleyişlerini sürdürmektedir. **WaitAll** çağrısı aşıldıktan sonraysa, **Task** örnekleri henüz çalışmalarını tamamlamadıklarından **Running** durumunda kalmışlardır. Ki işleyişleri de devam etmektedir. Burada kullanıcı isterse iptal işlemini gerçekleştirebilir ki örneğimizde bu senaryo icra edilmiştir. Dolayısıyla **Task** örneklerinin her ikisi içinde **Status** değerleri **Canceled**olmuştur.

Gelelim diğer bir duraksatma tekniğine.

WaitAny Kullanımı

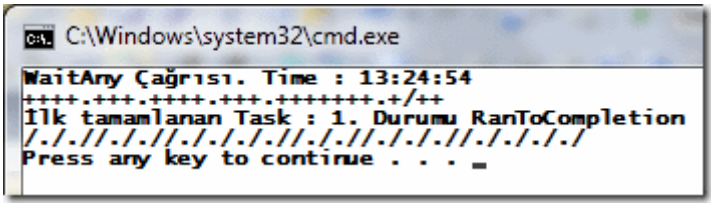
WaitAny metodu ile bir **Task** topluluğundan herhangibiri için, çağırıcının bekletilmesi sağlanabilir. **Task** sınıfı üzerinden çağırılabilen **static WaitAny** metodu, parametre olarak gelen **Task** örneklerinden hangisi tamamlanmışsa, bu örneğin dizi içerisindeki indis değerini döndürür. **–1** değer döndürmesi, zaman aşımına uğrandığını veya**CancellationToken** üzerinden bir iptal talebi gerçekleştirildiğini ifade etmektedir. Ancak bu değerin alınabilmesi içinde, **WaitAny** metodunun uygun olan **aşırı yüklenmiş(Overload)**versiyonunun kullanılması gerekmektedir. Dilerseniz konuyu daha net kavrayabilmek adına aşağıdaki örnek kod parçasını göz önüne alalım.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
```

```
namespace WaitingScenarios2
{
   class Program
   {
      static void Main(string[] args)
      {
         IEnumerable<int> numbers = Enumerable.Range(0, 60000000);
         IEnumerable<int> numbers2 = Enumerable.Range(0, 15000000);
         IEnumerable<int> numbers3 = Enumerable.Range(0, 45000000);

         Task startedTask = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers.Count(); i++)
            {
               i++;
               i--;
               i *= 2;
               Console.Write(".");
            }
         });

         Task startedTask2 = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers2.Count(); i++)
            {
               i++;
               i--;
               i *= 2;
               Console.Write("+");
            }
         }
         );

         Task startedTask3 = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers3.Count(); i++)
            {
               i++;
               i--;
               i *= 2;
               Console.Write("/");
            }
         }
         );
```

```
            Task[] tasks = { startedTask, startedTask2, startedTask3 };
            Console.WriteLine("WaitAny çağrısı. Time : {0}",
DateTime.Now.ToLongTimeString());
            int completedTaskIndex = Task.WaitAny(tasks);
            Console.Write("\nİlk tamamlanan Task : {0}. Durumu
{1}\n",completedTaskIndex,tasks[completedTaskIndex].Status);
            Console.ReadLine();
        }
    }
}
```

Kod içerisinde 3 farklı **Task** örneğinin çalıştırıldığı görülmektedir. Bu işlemler sonrasında **WaitAny** metodu çağırılmış ve ana uygulama Thread' inin duraksatılması sağlanmıştır. örneği çalıştırdığımızda aşağıdaki ekran çıktısındakine benzer bir sonuç ile karşılaşmamız olasıdır.



Burada dikkat edilmesi gereken nokta, üç **Task** nesne örneğinden ilk olarak hangisi bitmişse, **WaitAny** metodunun ona ait indis değerini döndürmesidir. örneğimizde dizi içerisinde ikinci sırada yer alan, bir başka deyişle **1 numaralı index değerine sahip olan Task** gövdesi ilk tamamlanan içeriktir. Dolayısıyla **WaitAny** metodu geriye 1 değerini döndürmektedir. Geri dönüşten sonra fark edileceği üzere henüz tamamlanmayan Task örnekleri çalışmalarına devam edecektir.

örnekleri daha iyi kavrayabilmek adına mutlaka çalıştırıp test etmenizi, debug zamanında durarak anlık durumları incelemenizi öneririm.

Böylece geldik bir yazımızın daha sonuna. Bu yazımızda **Task** örneklerinin sahibi olan çalıştırıcıların*(Ana uygulama Thread' i gibi)* nasıl bekletilebileceklerini inclemeye çalıştık. Tabi bu örneklerde istisna fırlatılmasına yönelik vakaları değerlendirmedik. Ancak bu tip durumlarında incelenmesi gerekmektedir. İşte size güzel bir araştırma konusu. **Task Parallel Library** ile ilişkili kavramları incelemeye devam ediyor olacağız. Bir sonraki yazımızda görüşünceye dek hepinize mutlu günler dilerim.

WaitingScenarios2.rar (23,05 kb) **[örnek Visual Studio 2010 Ultimate sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Task Süreçlerinde Bilinçli Olarak Duraksatma (2010-12-19T00:20:00)

task parallel library,c# 4.0,.net framework 4.0,parallel programming,thread,task cancellation,

Merhaba Arkadaşlar,

Pek çoğumuz hayatımızın çeşitli dönemlerinde ayakta veya oturarak bir şeyler için beklemek zorunda kalırız. Kimi zaman seyahat edeceğimiz yere gidecek aracı bekleriz. özellikle uçak seyahatleri gibi gecikmelerin sıklıkla yaşanabileceği durumlardan tutunda, İstanbul trafiğinin akşam saatlerindeki yoğunluğu yüzünden yine belirsiz süre dolmuş beklenmesi gibi hallerle sıklıkla karşılaşılır.

Fakat bazı zamanlarda belirli süreli bekleyişler de söz konusu olabilir. örneğin bir iş görüşmesinin saati bellidir ve gecikme durumları pek söz konusu olmamaktadır. Ya da bir metronun kalkış saati, hatta gideceği yere varış zamanı dahi sabittir. Dolayısıyla metroaya binmek için beklenen ve metro içindeyken gidilecek yere kadar geçen süreler genellikle standarttır. Ancak hangi durum olursa olsun, hayatımızda her zaman için beklemeler söz konusudur ve bunlar zaman zaman tekrar ederek bir yaşam döngüsünün oluşmasına neden olmaktadır.

Beklemeye neden bu kadar taktığımızı düşünebilirsiniz. Aslında beklemeyi seven bir insan değilimdir. Metro' ların yaşam stili bana biraz daha uygundur diyebilirim. çünkü bekleme süreleri sabittir ve buna göre planlama yapmak kolaydır. Ancak hayat her zaman bu kadar kolay planlama yapmayı olanaklı kılmayacak sürprizlerle doludur.

Bu günkü konumuz **Task** nesne örneklerinin işlettikleri süreçleri bilinçli olarak nasıl bekletebileceğimiz ile ilgilidir. Pek çok sebepten dolayı **Task** örneklerinin çalıştırdıkları iş parçalarının belirli süreler boyunca veya süre bağımsız olarak bekletilmeleri istenebilir. Burada **zaman bağımlı** ya da **koşul bağımlı** olarak bekletmelerin/duraksatmaların yapılabilmesi söz konusudur. Genel olarak **3** farklı bekletme tekniğinden söz edebiliriz.

- **CancellationToken** nesne örneği üzerinden ulaşılan **WaitOne** metodu ile
- **Thread** sınıfının static **Sleep** metodu ile
- **Thread** sınıfının static **SpinWait** metodu ile

Şimdi bu farklı teknikleri örnekler yardımıyla incelemeye başlayalım.

CancellationToken.WaitHandle.WaitOne Tekniği

WaitOne metodu yardımıyla belirsiz süreli veya belirli süreli duraksatma işlemleri gerçekleştirebiliriz. Konuyu daha net kavramak adına aşağıdaki örnek kod parçasını göz önüne alalım. Oldukça tanıdık gelecek.

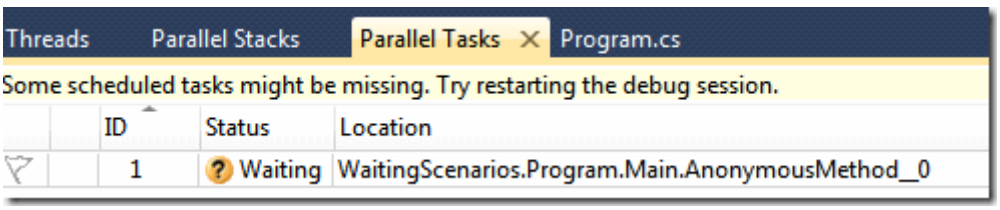
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;

namespace WaitingScenarios
{
   class Program
   {
      static void Main(string[] args)
      {
         IEnumerable<int> numbers = Enumerable.Range(0, 100000000);

         CancellationTokenSource tokenSource = new CancellationTokenSource();
         CancellationToken token = tokenSource.Token;

         Task startedTask = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers.Count(); i++)
            {
               bool waitStatus=token.WaitHandle.WaitOne();
               if (waitStatus == true)
                  throw new OperationCanceledException(token);
               else
                  Console.WriteLine("CancellationToken WaitOne Status = {0}",waitStatus);
               i++;
               i--;
               i *= 2;
               Console.Write(".");
            }
         }
         , token);

         token.Register(() =>
         {
            Console.WriteLine("İşlem iptali");
         }
         );
```

```
            Console.WriteLine("İşlemler devam ediyor. İptal etmek için bir tuşa basınız.");
            Console.ReadLine();

            tokenSource.Cancel();

            Console.ReadLine();
            Console.WriteLine("Task 1 Status = {0}", startedTask.Status);

        }
    }
}
```

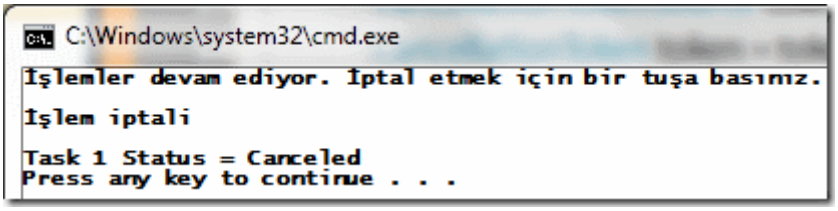
örnekte **Task** bloğu içerisinde yer alan **for** döngüsünün ilk satırında belirsiz süreli bir duraksatma yapıldığı görülmektedir. Nitekim **WaitOne** metodu herhangibir süre parametresi almamıştır. Dolayısıyla şartlar uygun olduğunda **Task'** in sonsuza kadar beklemesi de olasıdır.

Diğer yandan **WaitOne** metodu **bool** bir değer döndürmektedir. Peki nasıl olucaktırda bu değer dönecektir? Az önce belirsiz süreli bir bekletme yaptığımızdan bahsetmiştik. Cevap, iptal talebinin gelmesidir. **CancellationTokenSource** nesne örneği üzerinden gelen **Cancel** çağrısı, **WaitOne** metodunun işleyişini kesmesi, bir başka deyişle geriye anında bool bir değer döndürmesi ve kodun yürümeye devam etmesi anlamına gelmektedir. Böyle bir durumda **WaitOne** metodu **true** değer döndürecektir.

örneği çalıştırdığımızda ve özellikle bir tuşa basarak işlemi iptal etmediğimizde, **Debug** modda **Task** örneğinin anlık durumu aşağıdaki gibi görülecektir.



Dikkat edileceği üzere **Task** nesne örneğinin **Status** değeri **Waiting** olarak set edilmiştir. çalışma zamanında tuşa basılarak işlemin iptal edilmesinin sonucunda ise, aşağıdaki ekran görüntüsünde yer alan durum ile karşılaşılacaktır.

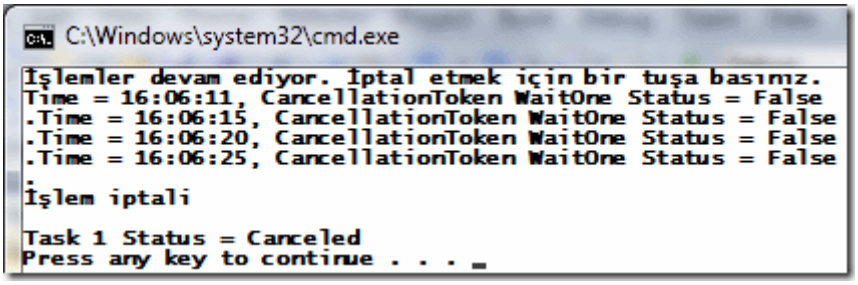


Görüldüğü gibi **Task** örneğinin **Status** değeri **Canceled** olarak değişmiştir.

Tabiki bu örnekte belirsiz süreli bir duraksatma işlemi gerçekleştirilmiş ve içinden çıkılması için **Cancel** çağrısının yapılması şart olmuştur. Ancak **WaitOne** metoduna istenildiğinde bekleme süresi, **milisaniye** cinsinden de verilebilir. örnek kodumuzda **3000** milisaniyelik(yani 3 saniyelik) bir duraksatma yaptırmak istediğimizi düşünelim. Bu durumda **WaitOne** metodunu aşağıdaki gibi kullanmamız yeterlidir.

bool waitStatus=token**.WaitHandle.WaitOne(3000);**

Bu durumda örneğin çalışma zamanı çıktısı aşağıdakine benzer olacaktır.



Farketmişsinizdir…3 saniyelik bir duraksatma yapmamıza rağmen ekrana yazılan bilgiler +2 saniye daha geç gelmektedir. Burada şüpheli şahıs değer aralığıdır. Nitekim**Enumberable** ile üretilen aralığın küçültülmesi(örneğin 10000' e çekilmesi) halinde süreler beklendiği gibi çıkmaktadır. Yani +2 lik kayıp olmamaktadır.

Sonuç itibariyle belirli süreliğine **Task** örneğinin işleyişinin duraksatılması gerçekleştirilmiştir. Bu süreli çalışmalarda **milisaniye** olarak belirtilen periyodun her tamamlanışında, kod bir sonraki satırdan devam etmekte ve **CancellationTokenSource** nesne örneğinin **WaitOne** metodu **false** değeri anında ortama döndürmektedir. Tahmin edileceği üzere bu değerin **true**olmasının **Task** işleyişinin iptal edilmesi gerekmektedir.

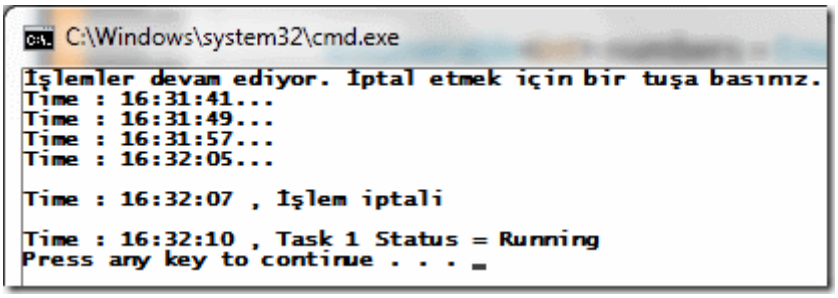
Thread.Sleep Metodunun Kullanılması

Task Parallel Library var olmadan önce, **çok Kanallı(Multi Thread)** uygulamaların geliştirilmesinde en çok haşır neşir olduğumuz tip sanıyorum ki **Thread** sınıfıdır. özellikle beliri durumlarda, çalışmakta olan bir **Thread** örneğini duraksatmak istediğimizde, **static Sleep** metodundan yararlanabiliriz. **Sleep** metodu **Task** örneklerinin duraksatılması içinde kullanılabilir. Bu son derece doğaldır nitekim **TPL** zaten var olan **Thread** sistemi üzerine kurulumuş bir alt yapıdır. İşte örnek kod parçamız.

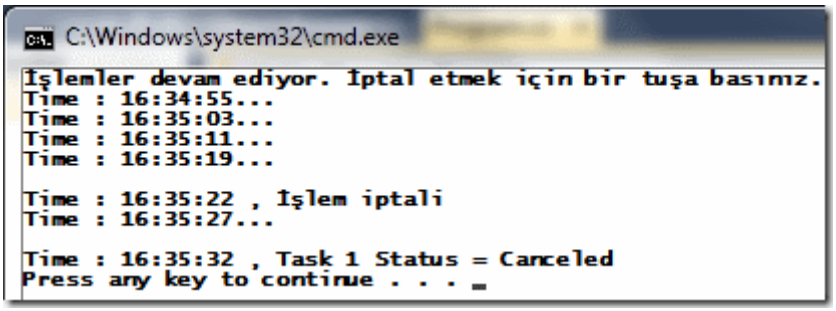
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;
using System.Threading.Tasks;
```

```
namespace WaitingScenarios
{
   class Program
   {
      static void Main(string[] args)
      {
         IEnumerable<int> numbers = Enumerable.Range(0, 10000);

         CancellationTokenSource tokenSource = new CancellationTokenSource();
         CancellationToken token = tokenSource.Token;

         Task startedTask = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers.Count(); i++)
            {
              Thread.Sleep(8000);
               i++;
               i--;
               i *= 2;
               Console.WriteLine("Time : {0}...",DateTime.Now.ToLongTimeString());
               token.ThrowIfCancellationRequested();
            }
         }
         , token);

         token.Register(() =>
         {
            Console.WriteLine("Time : {0} , İşlem
iptali",DateTime.Now.ToLongTimeString());
         }
         );

         Console.WriteLine("İşlemler devam ediyor. İptal etmek için bir tuşa basınız.");
         Console.ReadLine();

         tokenSource.Cancel();

         Console.ReadLine();
         Console.WriteLine("Time : {0} , Task 1 Status = {1}",
DateTime.Now.ToLongTimeString(),startedTask.Status);
      }
   }
}
```

örnekte 8 saniyelik bir duraksatma işlemi söz konusudur. Burada anlaşılması güç bir durum yoktur ancak çalışma zamanında dikkat edilmesi gereken bir nokta vardır. İlk testimizin sonuçları aşağıdaki gibidir.



Şimdi ikinci teste ait ekran görüntüsünü de yazımıza ekleyelim ve iki şekil arasındaki 9 farkı bulmaya çalışalım.



İlk çalışma da işleyişi sonlandırmak için kullanıcı tuşa bastığında bir **Cancel** talebi üretilmektedir. **Cancel** talebi **7nci saniye** de gelmiş ve kullanıcı tekrardan tuşa **10ncu saniye** de basarak uygulamayı sonlandırmıştır. Ne varki **Task** örneğinin **Status** değeri **Running** olarak set edilmiştir. Yani halen çalışıyor olduğu bildirilmektedir. Bunun sebebi, **Task**için **Cancel** çağrısı yapılmış olsa bile, **Thread** tipinin **static Sleep** metodunun belirttiği bekleme süresinin dolmamış olmasıdır. Ancak bu süre dolduktan sonra **Task** örneğinin**Status** değerinin **Canceled** olduğu görülebilir. Bu da **CancellationToken** üzerinden ele alınan **WaitOne** metodu ile **Thread.Sleep** arasındaki en önemli farktır.

Thread.SpinWait Metodunun Kullanımı

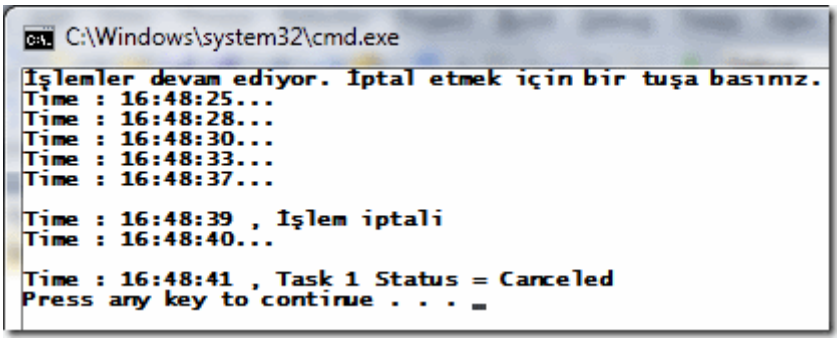
Bu teknikte CPU' nun çalıştıracağı ve duraksatma işlemi için gerekli döngü sayısı belirtilir. İşlemci aslında kendi içerisinde çok hafif bir döngüyü kullanarak duraksatma işlemini gerçekleştirmektedir. Bu nedenle işlemcinin durumu, anlık yoğunluğu gibi kriterler bekleme sürelerinin belirlenmesinde önemli rol oynamaktadır. Gelin örnek kod parçamızı değerlendirerek konuyu anlamaya çalışalım.

```
using System;
using System.Collections.Generic;
using System.Linq;
```

```
using System.Threading;
using System.Threading.Tasks;

namespace WaitingScenarios
{
   class Program
   {
      static void Main(string[] args)
      {
         IEnumerable<int> numbers = Enumerable.Range(0, 100000000);

         CancellationTokenSource tokenSource = new CancellationTokenSource();
         CancellationToken token = tokenSource.Token;

         Task startedTask = Task.Factory.StartNew(() =>
         {
            for (int i = 0; i < numbers.Count(); i++)
            {
               Thread.SpinWait(50000000);
               i++;
               i--;
               i *= 2;
               Console.WriteLine("Time : {0}...", DateTime.Now.ToLongTimeString());
               token.ThrowIfCancellationRequested();
            }
         }
         , token);

         token.Register(() =>
         {
            Console.WriteLine("Time : {0} , İşlem iptali",
DateTime.Now.ToLongTimeString());
         }
         );

         Console.WriteLine("İşlemler devam ediyor. İptal etmek için bir tuşa basınız.");
         Console.ReadLine();

         tokenSource.Cancel();

         Console.ReadLine();
         Console.WriteLine("Time : {0} , Task 1 Status = {1}",
DateTime.Now.ToLongTimeString(), startedTask.Status);
      }
```

```
    }
}
```

SpinWait metoduna verilen **50milyon** değeri tahmin edeceğiniz üzere süreyi değil iterasyon sayısını belirtmektedir. Buna göre işlemcinin 50 milyon kere, hafif bir döngüyü çalıştırarak o anki **Task** örneğinin işleyişini duraksatacağı belirtilmektedir. Kendi sistemimde söz konusu örnek kodun çalışma zamanı sonuçları aşağıdaki gibi gerçekleşmiştir. Ancak bu durum farklı sistemlerde farklı sonuçlar verebilir.



Görüldüğü gibi sabit bir bekleme süresi söz konusu olmamıştır. Bazen 3 saniye bazen 2 saniye ve bazende 4 saniyelik bir duraksama olduğu görülmektedir. Yine doğal olarak bu bekleme sürecinden çıkılması için iptal talebinin yapılması veya tüm işleyişin sona ermesinin beklenmesi yeterlidir.

Bu yazımızda **Task** işleyişlerinin duraksatılması için kullanabileceğimiz teknikleri incelemeye çalıştık. Bir sonraki yazımızda duraksatma işlemleri için kullanılabilecek diğer teknikleri de öğrenmeye çalışıyor olacağız. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WaitingScenarios.rar (23,76 kb) **[örnek Visual Studio 2010 Ultimate sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Persisted Memory-Mapped Files (2010-12-17T19:30:00)

.net framework 4.0,c# 4.0,memory-mapped files,visual studio 2010 ultimate,

Merhaba Arkadaşlar,

Sene 1997. üniversite 3ncü sınıf öğrencisiyim. Eskiden lisanslı bir basketbolcu olan uzun boylu arkadaşım Serkan ve Babası ile birlikte Tepebaşındaki bilgisayar fuarındayız. Serkan, kendisine bir bilgisayar almak istiyor. Tabi Baba'sının gelmesinin nedeni işlemci mimarilerini, ram teknolojilerini çok iyi bilmesi değil. Tamamen

duygusal :$ Ben de Serkan arkadaşıma teknik olarak destek verip önerilerde bulunuyorum. Derken 3müz arasında şöyle bir konuşma geçiyor;

Burak : Serkan bak Pentium 200 MMX var. Creative CD sürücü. üstelik uzaktan kumandalı. Diamond Stealth ekran kartı 2 mb. İyidir. Seagate Hard Disk. Süper.

Serkan : Abi kaç ram var bunda?

Burak : Abi iki model var. Biri 32 Mb diğeri 64 mb. Ben derim ki 64 Mb olanı al.

Serkan : Baba bu 64 Mb olanı alalım.

Baba : Kaç dolar fark var arada?

Serkan : X dolay fark var Baba.

Baba : 32 Mb olanı alalım.

Serkan : Ama baba, 64 Mb olan daha iyi. Uygulamalar için daha çok yer tahsis ediyor. Daha rahat çalışacak uygulamalar. Aynı anda daha çok uygulama çalışacak.

Baba : Sen önce 32 Mb olanı doldur, sonrasına bakarız :S

Bu minik hikayeden sonra şu an geldiğimiz noktaya baktığımızda Gb' larca Ram' den bahsediyoruz. Hatta işlemcilerin birincil ve ikincil ön bellek kapasitelerinin de oldukça yükseldiğini görüyoruz. Tabi bu bana göre yazılımların istedikleri donanımsal ihtiyaçların bir sonucu. Gelelim bu günkü konumuza. Bu günkü konumuzda aslında sanal bellek(Virtual Memory) ile alakalı ve özellikle çok büyük boyutlu dosyalar ile çalışan uygulamalar söz konusu olduğunda bir o kadar da önemli bir mevzu.

.Net Framework 4.0 ile birlikte gelen yeniliklerden birisi de **Memory-Mapped File** kullanımı. Herşeyden önce **Memory-Mapped File** kavramının ne anlama geldiğini irdeleyerek işe başlayalım.

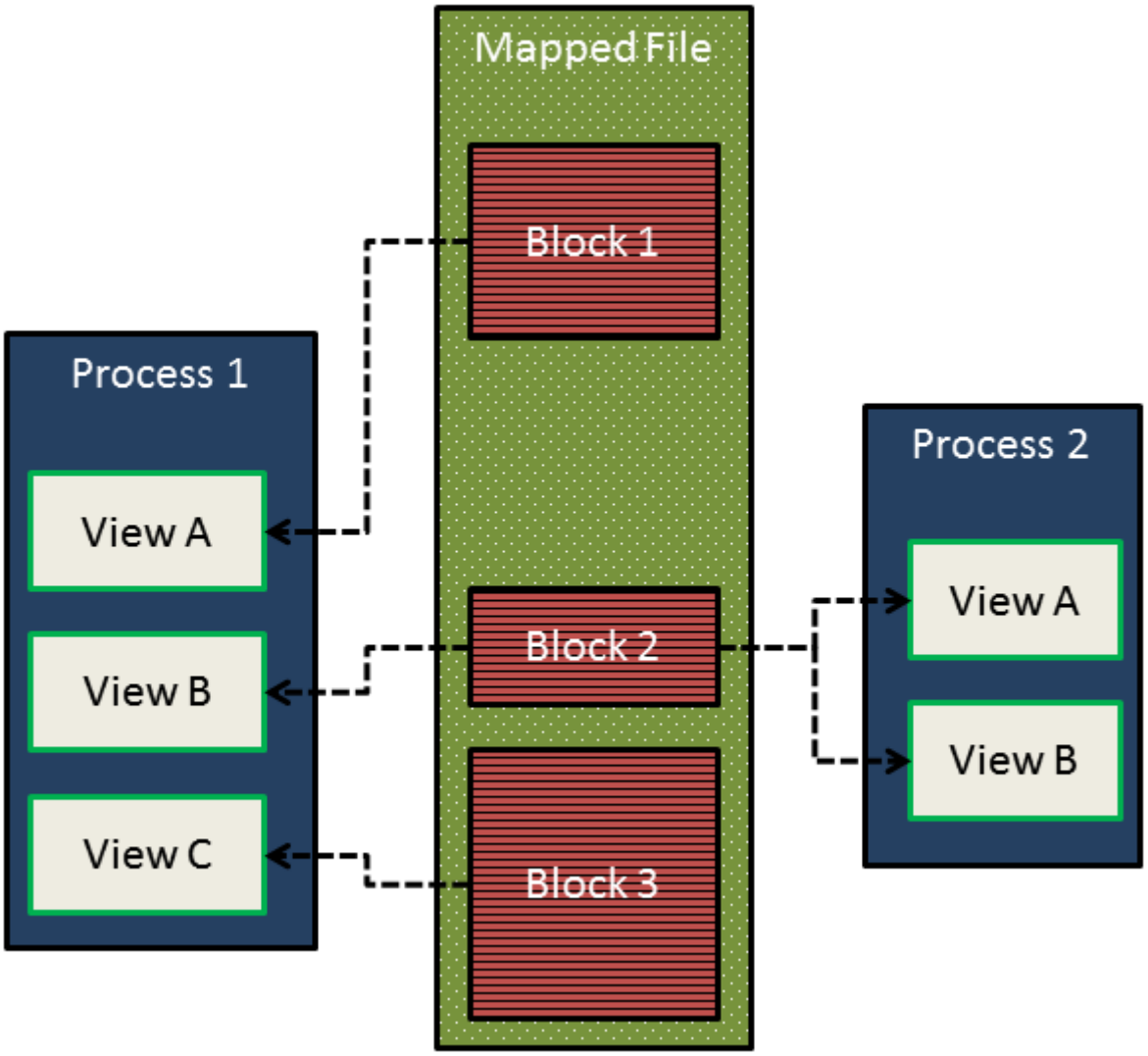
Memory-Mapped dosyalar adından da anlaşılacağı üzere bellek üzerine açılmış içerikler olarak düşünülebilirler. Aslında fiziki bir dosyanın tüm içeriğinin **Virutal Memory** üzerinde oluşturulması ve uygulamanın mantıksal adres alanı içerisinde yer alması söz konusudur. İçeriğin yüklendiği bellek alanı, farklı uygulama **Process'** leri tarafından da ele alınabilir. Buna göre farklı **Process**' ler bellek üzerine açılmış dosya içeriklerinde okuma ve yazma gibi işlemleri yapabilirler. **.Net Framework 4.0, Memory-Mapped dosyalarınManaged Code** tarafından da ele alınabilmesini sağlamaktadır. Nitekim bu versiyona kadar **unmanaged code** yardımıyla ele alabildiğimiz bir kavramdır. Dolayısıyla **C++** ve **Win32 API**üzerinde değerlendirdiğimiz düşük seviyeli bir konudur.

Memory-Mapped dosyalar iki şekilde değerlendirilmektedir. **Persisted** ve **Non-Persisted** olarak. **Persisted** modelinde bellek üzerinde açılan dosya ile fiziki dosya

arasında bir ilişki bulunmaktadır. Bu modele göre bellek üzerindeki içerikte değişiklik yapan son **Process** işini tamamladığında, yapılanların fiziki dosyaya yansıtılması söz konusudur. **MSDN** kaynaklarına göre bu model, **çok büyük boyutlu dosyaların farklı Process' ler tarafında ele alındığı durumlarda** tercih edilmektedir.

İkinci modele göre bellek üzerine açılan içerik ile fiziki kaynak arasında herhangibir bağ yoktur. Bir başka deyişle fiziki disk üzerindeki bir dosya işaret edilmemektedir. Dolayısıyla bellek üzerinde **Process'** ler tarafından yapılan değişiklikler fiziki bir kaynağa yansıtılmamaktadır. Daha çok **IPC(Inter Process Communication) tipindeki iletişimlerim söz konusu olduğu durumlarda** tercih edilen bir modeldir.

Aslında **Memory-Mapped File** kavramını aşağıdaki şekil ile biraz daha anlaşılır hale getirebiliriz.



Bu şekilde, **Memory-Mapped File** içerisindeki farklı blokların, farklı **Process'** ler tarafından nasıl ele alınabildiği temsil edilmektedir. Buna göre örneğin **Block 2, Process 1** ve **2** içerisindeki farklı **View'** lar ile ifade edilebilmektedir. Aslında **Memory-Mapped** dosyaları ile çalışabilmek için mutlaka **View** oluşturmak gerekmektedir. **View** nesneleri, ilgili bellek alanında açılan dosya içeriğinin tamamını

işaret edebileceği gibi bir kısmını da içerebilir. Bir dosya bloğunun farklı **View** nesneleri de oluşturulabilmektedir**(Multiple Views)**. Zaten dosya boyutunun, uygulamanının **mantıksal bellek alanının(Logical Memory Space)** dışına taştığı durumlarda **Multi-View** nesnelerinin oluşturulması şarttır. özellikle **Gb** boyutuna varan dosyalar ile çalışan uygulamalar düşünüldüğünde **Multi-View** kullanımı kaçınılmazdır.

View nesnelerinin de iki çeşidi bulunmaktadır. **Stream Access View** ve **Random Access View**. Eğer dosya içeriğine sıralı olarak erişilecekse**(Sequential Access) Stream Access View** nesneleri kullanılır. **Non-Persisted** ve **IPC** kullanımı söz konusu olduğu durumlarda önerilen metoddur. **Persisted** modelin kullanıldığı durumlarda ise **Random Access View** nesneleri tercih edilmelidir. Bu teorik bilgilerden sonra dilerseniz basit bir örnek ile konuyu anlamaya çalışalım.

İlk olarak aynı **Solution** içerisinde iki farklı **Console** uygulaması geliştiriyor olacağız. Bu **Console** uygulamalarından bir tanesi sistemde yer alan bir **MP3** dosyasını **Memory-Mapped File**olarak oluşturacak ve içinden belirli bir aralığı **View** olarak üretecektir. Diğer uygulamada, aynı dosyanın belleğe açılmış alanına erişecek ve içerisinde farklı bir bloğu **View**olarak kullanacaktır. İlk olarak **ProcessB** olarak adlandırdığımız uygulama kodlarını ele alalım.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.MemoryMappedFiles;
using System.IO;

namespace ProcessB
{
   class ProgramB
   {
      static void Main(string[] args)
      {
         // Persisted Memory-Mapped File Kullanımı

         Console.Title = "Process B";
         string filePath = @"D:\MusicFile.mp3";

         using (MemoryMappedFile mappedFile =
MemoryMappedFile.CreateFromFile(filePath, FileMode.Open, "Mapped1"))
         {
            using (MemoryMappedViewAccessor view =
mappedFile.CreateViewAccessor(30000, 50000))
            {
               for (int i = 0; i < 20; i++)
```

```
            {
               Console.Write("{0} ",view.ReadByte(i));
            }
         }
         Console.WriteLine("Kapatmak için bir tuşa basınız");
         Console.ReadLine();
      }
   }
}
```

Kod parçasında ilk olarak **MemoryMappedFile** tipinden bir örnek üretildiği görülmektedir. Bu üretim işlemi için **CreateFromFile** metodundan yararlanılmaktadır. Bu metod **Persisted** modele göre bir **Memory-Mapped File** oluşturmaktadır. Bu sebepten dolayı fiziki disk üzerinde var olan bir dosya adresini parametre olarak almaktadır.

Metodun ilk parametresi fiziki dosya adresidir. İkinci parametre ile dosyanın açılacağı ifade edilmektedir. Son parametre ile de **Memory-Mapped File** için bir isim verilmektedir. Bu isim önemlidir. Nitekim diğer bir uygulama tarafından ilgili bellek adresindeki alana erişilmek istendiğinde kullanılacaktır.

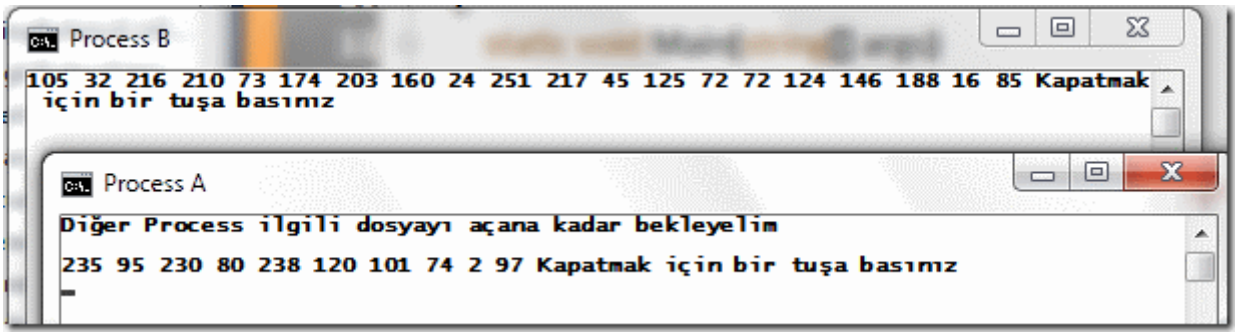
Program kodunun ilerleyen kısmında **MemoryMappedFile** nesne örneğinin **CreateViewAccessor** metodundan yararlanılarak bir **MemoryMappedViewAccessor** oluşturulmaktadır. Bir başka deyişle bir **View** üretildiğini söyleyebiliriz. Bu üretim işlemi sırasında verilen değerler ise bellek üzerinde okunmak istenen bloğun başlangıcı ile okunacak uzunluğu ifade etmektedir. Bir başka deyişle kaçıncı **byte'** tan itibaren ne kadar uzunlukta okuma yapılacağı belirtilir. **for** döngüsü içerisinde sembolik olarak **View** şeklinde ele alınan bloğun ilk **20 byte'** ı ekran yazdırılmaktadır. çok doğal olarak burada veri üzerinde değişiklikler de yapılması söz konusu olabilir. Gelelim diğer **Console** uygulamasının kodlarına.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO.MemoryMappedFiles;

namespace ProcessA
{
   class ProgramA
   {
      static void Main(string[] args)
      {
         Console.Title = "Process A";
```

```
        Console.WriteLine("Diğer Process ilgili dosyayı açana kadar bekleyelim");
        Console.ReadLine();

        using (MemoryMappedFile mappedFile =
MemoryMappedFile.OpenExisting("Mapped1"))
        {
           using (MemoryMappedViewAccessor view2 =
mappedFile.CreateViewAccessor(70000, 120000))
           {
              for (int i = 0; i < 10; i++)
              {
                 Console.Write("{0} ", view2.ReadByte(i));
              }
           }
           Console.WriteLine("Kapatmak için bir tuşa basınız");
           Console.ReadLine();
        }
     }
  }
}
```

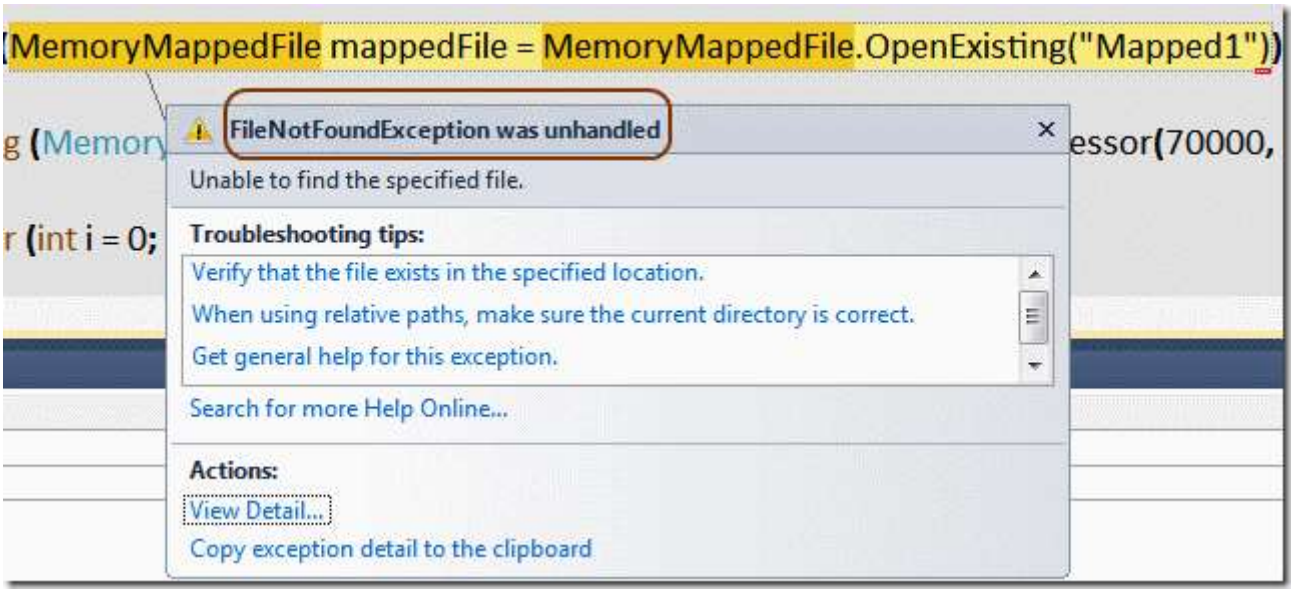
Bir önceki koda benzer olaraktan yine bir **MemoryMappedFile** ve **View** üretimi söz konusudur. Ancak **MemoryMappedFile** üretimi için **OpenExisting** metodundan yararlanılmaktadır. Bu metoda verilen parametre dikkat edileceği üzere bir önceki uygulamanın oluşturduğu **Memory-Mapped File** için verilen takma isimdir. Dolayısıyla program kodunun bu noktasında eğer bellek üzerinde **Mapped1** isimli bir **Memory-Mapped File** varsa ilgili nesne örneklenebilmektedir.

Nesne örneklemesini takiben yine bir **MemoryMappedViewAccessor** oluşturulmaktadır. Bu kez farklı bir blok ele alınmakta ve yine içeriğinin ilk **10 byte**' lık bölümü ekrana yazdırılmaktadır. önce **ProgramB** ardından **ProgramA** çalıştırılacak şekilde **Solution** ayarlanırsa aşağıdaki çalışma zamanı görüntüsü elde edilecektir



Bu örnekte iki farklı **Process'** in aynı **Memory-Mapped File** üzerinde farklı bloklarını işaret eden **View'** lar oluşturarak çalıştığı gösterilmektedir. Ancak tabiki dikkat edilmesi gereken bazı durumlar söz konusudur. örneğin **Memory-Mapped File** içeriğini ilk

üreten **Process** diğeri çalışmadan önce kapatılır ve diğer **Process Mapped1** isimli bellek alanına erişmeye çalışırsa, aşağıdaki istisna mesajı ile karşılaşılır.



Bu son derece doğaldır. Nitekim **ProcessB** başlatılmamış, **using** blokları dışına çıkılmış veya uygulama sonlandırılmıştır. Ancak tüm bunlar gerçekleşirken**ProcessA** henüz çalıştırılmamış olabilir.

Geliştirdiğimiz örnekte bellek üzerine açtığımız dosya içeriğinde bir yazma işlemi gerçekleştirilmemiştir. Ancak **CreateFromFile** metodunun kullanılması halinde **Persisted** model söz konusudur. Dolayısıyla yapılan değişiklikler son olarak fiziki dosyaya da aktarılabilir.

Biraz dinlenmeye ne dersiniz. Takip eden yazımızda **Non-Persisted** modele göre **Memory-Mapped File** kavramını ele almaya çalışıyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

MemoryMappedFileKavrami.rar (40,21 kb) **[örnek Visual Studio 2010 Ultimate sürümü üzerinde geliştirilmiş ve test edilmiştir]**

WCF Öğreniyorum Ders 3-Bağlayıcılar (2010-12-13T23:30:00)

wcf,wcf 4.0,wcf öğreniyorum,windows communication foundation,nedirtv?com,

Merhaba Arkadaşlar,

Hız kesmeden NedirTv?com liderliğinde sürdürdüğümüz **WCF öğreniyorum Webiner** serimize devam ediyoruz. Bu webinerimizde ağırlık olarak**Binding Type(Bağlayıcı Tip)** kavramını anlamaya çalışacağız.

Sunum üzerindeki tablolardan bazı **Binding** tiplerini ve aralarındaki farklılıkları tartışıyor olacağız. örnek uygulamamızda ise bir servisi birden fazla **Endpoint** üzerinden host edeceğiz. Ayrıca **istemci(Client)** ile **servis(Service Host)** arasındaki mesajlaşmalara ait **log** dosyasına bir göz atıp farklılıkları en azından bir kaç **Binding** tipi için kavramaya çalışacağız. Her zamanki gibi webinerimize ait sunum dosyası, **Solution'** ın son hali ve ekran kaydına aşağıdaki linklerden ulaşabilirsiniz. Bir sonraki webinerimizde görüşünceye dek hepinize mutlu günler dilerim.

Sunum Dosyası : WCF 4.0 - Ders 3 - Bindings.pptx (318,15 kb)

Solution Son Hali : WCF_Ogreniyorum8_12_2010.rar (434,12 kb)

NedirTv?com Bağlantısı

WCF Öğreniyorum Ders 2-Veri Sözleşmeleri II (2010-12-06T08:40:00)

windows communication foundation,wcf ogreniyorum,wcf,data contracts,surrogate types,wcf service application,

Merhaba Arkadaşlar,

Nedirtv?com liderdliğinde sürdürdüğümüz **WCF öğreniyorum Webiner(Webcast)** serisinin 3ncü dersini de(biliyorsunuz ders numaralarımız 0dan başlıyor) tamamlamış bulunuyoruz. öncelikli olarak katılan tüm arkadaşlarımıza çok teşekkür ederim.

3ncü dersimizde WCF Servisimizi IIS üzerinde host edebilecek şekilde bir **WCF Service Application** şablonunu kullandık. Bu sayede **HTTP** bazlı olarak yayınlama yapan bir **WCF Servisinin Web** üzerinden erişilebilirliğini değerlendirdik. Ayrıca Entity bazlı bir kütüphanenin veri içeriğini istemci tarafına sunarken **Surrogate** tipindeki veri sözleşmelerini nasıl değerlendirebileceğimize baktık. Söz konusu servise ait operasyonları ele alan istemci tarafında da, bu veri sözleşmelerini nasıl kullanabileceğimizi gördük. Her zaman ki gibi, **örnek uygulama kodlarını, Powerpoint sunum dosyasını** ve **Nedirtv?com** üzerine eklenmiş olan ekran kayıtlarını aşağıda linkerden tedarik edebilirsiniz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

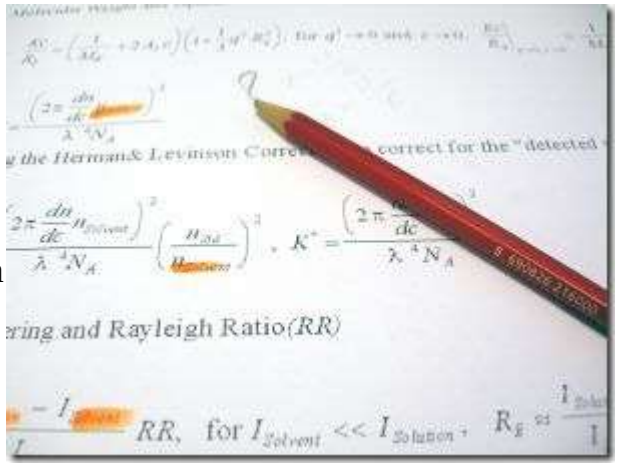
Sunum Dosyası : WCF 4.0 - Ders 2 - Veri Sozlesmeleri II.pptx (314,90 kb)

Solution Son Hali : WCF Ogreniyorum.rar (336,05 kb)

NedirTv?com bağlantısı

Tuple Nedir? Anlamak, Bilmek İstiyorum. (2010-12-01T16:55:00)

c# 4.0,tuple,



Merhaba Arkadaşlar,

Matematik Mühendisliği eğitimi almış birisi olarak hayatımın önemli bir kısmını teorem ispatlarına harcadığımı itiraf edebilirim. Tabi yaşamımın mesleki olarak kırılma anı sanıyorum ki üniversite yıllarında bilgisayara merak salmam ve programlama ile alakalı dersleri daha çok sevmemdi. Kısacası Matematik üzerine eğilmekten vazgeçip*(ki bundan biraz pişmanlık duyduğumu ifade edebilirim)* yazılım alanında ilerlemeyi kafama koymuştum.

Ama tabiki insanoğlu Matematik' ten kaçamıyor. Hayatının belirli noktalarında öyle ya da böyle karşılaşıyor. Söz gelimi çalıştığım projelerin bazılarında, müşterinin sorunlarının çözümü için matematiksel modellerin kullandığını söyleyebilirim. örneğin bir üretim hattının minimum maliyet gibi değerlere ulaşabilmesi için çalıştıracağı optimizasyon modelleri, dünyanın en zor matematik formüllerini içermektedir. Yazıya yaptığımız bu giriş sizleri korkutmasın. Bu günkü konumuzun Matematik ile çok alakası yok. Ancak Matematik dünyasından bir ismin yazılım tarafına geçirlişini inceleyeceğimizi ifade edebilirim. Bu günkü konumuz, **.Net Framework 4.0** ile birlikte gelen **Tuple**.

Tuple' ın Türkçe' deki kelime karşılığı **Demet** olarak ifade edilmektedir. Matematikte elementlerin sıralanmış bir liste tasarımı olarak tanımlanmaktadır. Diğer yandan **İlişkisel Veritabanı Sistemlerinde(Relational Database Management Systems)** tablo içerisindeki bir satır olarak düşünülür. Dolayısıyla burada da sütunların sıralı bir dizisinden oluşan liste şeklinde ifade edildiğinde, Matematiksel tanımını da işaret ettiği ifade edilebilir. Bilgisayar literatüründe birden fazla parçadan oluşan yapı anlamında kullanılmaktadır. **Python**dilinde ise içeriği değiştirilemeyen bir dizidir. **Python** demişken. Aslında fonksiyonel programlama dillerinde **Tuple** kavramının uzun süredir var olduğunu da ifade edebiliriz. Hatta **F#** programlama dilinde de tuple kullanımı mevcuttur.

Not : *Bildiğiniz üzere **Pyhton, LISP, Perl** gibi dynamic dillerde tip belirtmeden değişken tanımlayabilmek mümkündür. Oysaki **C#** gibi **static** dillerde tip belirtme zorunluluğu vardır. Nitekim derleme işlemi sırasında değişkenlerin tiplerinin belli olması gereklidir. Bu dinamik dillerin değişken tanımlamada ve kullanmada sağladığı esnekliğin **static** dilde olmadığı anlamanı gelmektedir. Ancak **static** diller bu sayede daha az hata meyillidir.*

Diğer yandan **Tuple** kavramı, **C#** gibi static dillerde en azından **4.0** versiyonuna kadar geçerli değildir. **.Net Framework 4.0** ile birlikte gelen önemli yeniliklerden birisi de bildiğiniz üzere dinamik diller ile olan etkileşimdir. Bu etkileşimin bir meyvesi veya ön

hazırlığı olarak **Tuple** kavramının **.Net Framework** bünyesine katıldığını da ifade edebiliriz.

Bu kadar laf kalablığından sonra dilerseniz örnekler ile **Tuple** tipini incelemeye çalışalım. **System** isim alanı altında yer alan **Tuple** sınıfı, **Tuple** nesnelerinin üretimi için bir**fabrika(Factory)** görevi üstlenmektedir. Bunun için static Create metoduna ve **aşırı yüklenmiş(Overload)** versiyonlarına sahiptir. **Tuple** tiplerini kullanmak için çeşitli sebepler ve senaryolar düşünülebilir. Şimdi bu durumları analiz etmeye çalışalım.

1- Metodlardan İsimsiz Tip(Anonymous Type) Döndürülemediği Durumlar

Aslında **Anonymous** tipler özellikle **LINQ(Language Integrated Query)** sorguları için tasarlanmıştır. **LINQ** sorgulaması sonucu elde edilen sonuç kümesinden anlık olarak karar verilen bir tipin ele alınması istendiği durumlarda oldukça değerlidir. Lakin bu tipler metodlardan geriye döndürülememektedir. Aşağıdaki örnek kod parçasını göz önüne alalım.

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace TupleKavrami
{
   class Program
   {
      static void Main(string[] args)
      {
         List<Person> employees = new List<Person>
         {
            new Person{ PersonId=1,Name="Burak", Surname="Şenyurt",
City="İstanbul",Salary=1000},
            new Person{ PersonId=2,Name="Maykıl", Surname="Cordın",
City="Chicago",Salary=1250},
            new Person{ PersonId=3,Name="Şakiyıl", Surname="Oniyıl", City="Los
Angles",Salary=1100}
         };

         List<Tuple<string, string,double>> result = GetEmployees(employees);
         foreach (Tuple<string,string,double> r in result)
         {
            Console.WriteLine("{0} {1} Salary :
{2}",r.Item1,r.Item2,r.Item3.ToString("C2"));
         }
      }
```

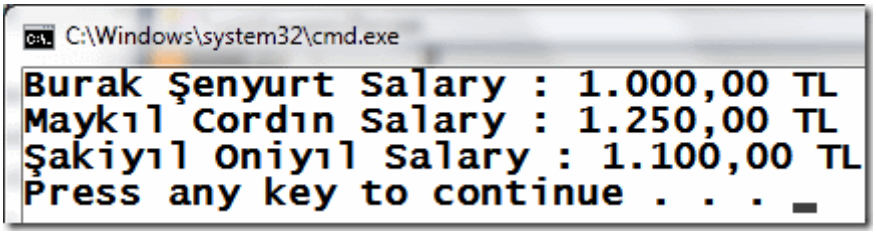
```
      static List<Tuple<string, string,double>> GetEmployees(List<Person>
employees)
      {
         var result = (from e in employees
                       select new Tuple<string, string,double>(e.Name,
e.Surname,e.Salary)).ToList<Tuple<string,string,double>>();

         return result;
      }
   }

   class Person
   {
      public int PersonId { get; set; }
      public string Name { get; set; }
      public string Surname { get; set; }
      public string City { get; set; }
      public double Salary { get; set; }
   }
}
```

Bu kod parçasında kayda değer pek çok yeni özellik bulunmaktadır. **GetEmployees** isimli metod, parametre olarak aldığı **List<Person>** tipinden bir koleksiyon içeriğini sorgulamaktadır. Bu sorgulama sonucunda aslında bir **anonymous** tip ile ifade edilebilecek yeni bir yapı değerlendirilmektedir. **Person** tipi içerisinde **PersonId,Name,Surname,City** ve **Salary** özellikleri yer alırken sorgu sonucu elde edilen satırlarda **Name,Surname** ve **Salary** özelliklerinin olması istenmektedir. Bu özellikler tahmin edileceği üzere bir **anonymous** tip içerisinde yer alabilirler. Ancak **Anonymous** tipin metoddan geriye döndürülemediğini ifade etmiştik. İşte bu noktada devreye **Tuple** tipini kattık.

Dikkat edilecek olursa **LINQ** sorgusu içerisinde **Tuple<string,string,double>** tipinden bir tanımlama yapılmaktadır. Bu noktada kodun o anki satırında 3 alandan oluşacak bir tip tasarladığımızı düşünebiliriz. Bu tipin özelliklerine ait değerler ise o anki **Person** nesne örneği içerisinden alınmaktadır.

Tuple tipi ile ilişkili olarak dikkate değer noktalardan biriside **foreach** bloğu içerisinde kendisini göstermektedir. **Tuple** tipinin örneklenen versiyonunda belirtilen **generic** parametrelere göre özellikler **Item1, Item2** ve **Item3** olarak ayarlanmıştır. Bu bir handikap olarak düşünülebilir. Nitekim ele aldığımız örnekteki vakka düşünüldüğünde **Person** tipi içerisinden çekilen**Name,Surname** ve **Salary** alan adlarının **Item1, Item2** ve **Item3** yerine kullanılması daha esnektir. Bildiğiniz üzere **Anoynmous** tiplerde, oluşacak nesnenin o anki özelliklerinin adları aynen korunabileceği gibi geliştirici tarafından da tanımlanabilmektedir. Bu yeteniğin **Tuple** tipine de ekleneceğini ümit etmekteyim.

Tabi bir diğer önemli nokta **Tuple** nesne örneklemesi sırasında kullanılan **T** tipinin **generic**' liğidir. Buna göre **Tuple** tipinin içereceği diğer tiplerin aynı olması zorunlu değildir. örnekte**string,string,double** tipinden bir oluşum söz konusudur. çalışma zamanı sonuçları aşağıdaki gibi olacaktır.



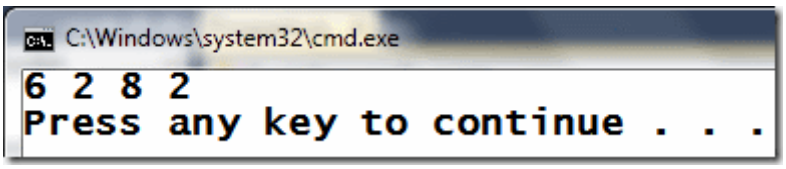
2 – Metodlardan Birden Fazla Değer Döndürmek İstediğimiz ve out Parametrelerini Tercih Ettiğimiz Durumlar.

Aslında bunun için pek çok yol mevcuttur. **out** tipinden metod parametreleri veya dönüş tipi olarak bir koleksiyon ya da dizi kullanılması söz konusu olabilir. Ancak **Tuple** tipi de bu anlamda değerlendirilebilir. Söz gelimi aşağıdaki kod parçasını göz önüne alalım.

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace TupleKavrami
{
   class Program
   {
      static void Main(string[] args)
      {
         int r1, r2, r3, r4;
         Calculate(4, 2, out r1, out r2, out r3, out r4);
         Console.WriteLine("{0} {1} {2} {3}",r1,r2,r3,r4);
      }

      static void Calculate(int x, int y, out int result1, out int result2, out int result3, out int result4)
      {
         result1 = x + y;
         result2 = x - y;
         result3 = x * y;
         result4 = x / y;
      }
   }
}
```

Bu kod parçasında yer alan **Calculate** metodu dört işlemi gerçekleştirmekte ve elde edilen sonuçları **out** tipinden metod parametreleri ile geriye döndürmektedir. Bu örnekte **out** parametrelerinin kullanımı yerine geri dönüş tipinin **Tuple** olarak tanımlaması da düşünülebilir. Aşağıdaki kod parçasında bu durum ele alınmaktadır.

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace TupleKavrami
{
   class Program
   {
      static void Main(string[] args)
      {
         Tuple<int, int, int, int> calculation = CalculateV2(4, 2);
         Console.WriteLine("{0} {1} {2}
{3}",calculation.Item1,calculation.Item2,calculation.Item3,calculation.Item4);
      }

      static Tuple<int, int, int, int> CalculateV2(int x, int y)
      {
         return Tuple.Create<int, int, int, int>(x + y, x - y, x * y, x / y);
      }
   }
}
```



CalculateV2 isimli metod geriye **Tuple<int,int,int,int>** tipinden bir nesne örneği döndürecek şekilde programlanmıştır. Bu seferki kod örneğimizde **Tuple** nesne örneğini oluşturmak için**Tuple** sınıfının **static Create** metodu tercih edilmiştir. Tabi bu örnekte **Tuple** nesne örneği içerisindeki tüm tipler **integer** olarak tanımlanmıştır. Ancak böyle bir zorunluluk olmadığını da daha önceden belirtmiştik. Yani **Tuple** tipi, kendi içerisinde farklı tipleri barındırabilmektedir. Tabi sonuçları ele aldığımız kod satırından **Tuple** nesne örneği içerisindeki değerlere yine **Item1, Item2 , Item3** ve **Item4** isimli özellikler ile erişebildiğimize dikkat edelim.

Aslında **Tuple** nesne örnekleri oluşturulurken daha kısa kod notasyonları da tercih edilebilir. **CalculateV2** metodunun içeriğini bu anlamda aşağıdaki gibi de kullanabiliriz.

```
static Tuple<int, int, int, int> CalculateV2(int x, int y)
      {
         //return Tuple.Create<int, int, int, int>(x + y, x - y, x * y, x / y);
         return Tuple.Create(x + y, x - y, x * y, x / y);
      }
```

Dikkat edileceği üzere burada **Create** metodunun parametrelerine doğrudan hesaplama sonuçları atanmış ama herhangibir şekilde tip bilgisi belirtilmemiştir.

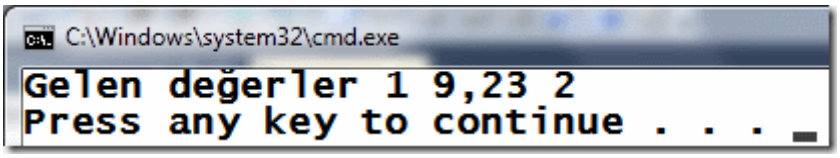
3 – Bir metoda birden fazla parametre göndermek istediğimiz bazı durumlarda

Zaten bir metoda **n** sayıda parametre gönderebilmekteyiz. Ancak bizim müdahale edemediğimiz önceden tanımlanmış tip metodları, sadece tek parametre alarak **n** sayıda parametre aktarımını biraz zorlaştırmaktadır. Bu tip durumlar elbetteki çözümsüz değildir. Söz gelimi dizi veya koleksiyon yardımıyla birden fazla parametre taşınması söz konusu olabilir. Ancak bu tip bir vakada **Tuple** tiplerinden de yararlanabiliriz. Bu anlamda parametre alabilen metodların **Thread** yönetimlerini göz önüne alabiliriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;

namespace TupleKavrami
{
   class Program
   {
      static void Main(string[] args)
      {
         ParameterizedThreadStart ts = new
ParameterizedThreadStart(ThreadMethod);
         Thread t = new Thread(ts);
         t.Start(Tuple.Create(1, 9.23F, 2));
      }

      static void ThreadMethod(object values)
      {
         Tuple<int, float, int> tpl = (Tuple<int,float,int>)values;
         Console.WriteLine("Gelen değerler {0} {1} {2}",tpl.Item1,tpl.Item2,tpl.Item3);
      }
   }
}
```

ParameterizedThreadStart, **Thread**' lerin başlatacağı metodlara parametre aktarımına izin vermektedir. Ancak bu noktada **object** tipinden parametre alan metodlar işaret

edilebilmektedir. örnek kod parçasında **t** isimli **Thread** nesne örneği üzerinden **Start** metodu çağrısı gerçekleştirilirken, bir **Tuple** nesne örneği gönderilmektedir. **ThreadMethod** içerisindeki kod bloğunda ise dikkat edileceği üzere **object** tipinden olan parametre bilinçli bir şekilde **Tuple<int,float,int>** tipine dönüştürüldükten sonra kullanılabilmiştir.



Peki Tuple Tipi için Eleman Sayısı Sınırı var mıdır?

Bu örnekler ile Tuple kullanımının nasıl yapılabildiğini ve hangi durumlarda değerlendirilebildiğini kısaca incelemeye çalıştık. Son olarak **Tuple** tipinin kaç iç tip ile çalışabildiğine bakıyor olacağız. Normal şartlar altında aşırı yüklenmiş **static Create** veya **yapıcı metodlarına(Constructors)** baktığımızda en fazla 8 elemanın söz konusu olduğunu görürüz.

static Create metodu için;

```
public static Tuple<T1, T2, T3, T4, T5, T6, T7, Tuple<T8>> Create<T1, T2, T3, T4, T5, T6, T7, T8>(
   T1 item1,
   T2 item2,
   T3 item3,
   T4 item4,
   T5 item5,
   T6 item6,
   T7 item7,
   T8 item8
)
```

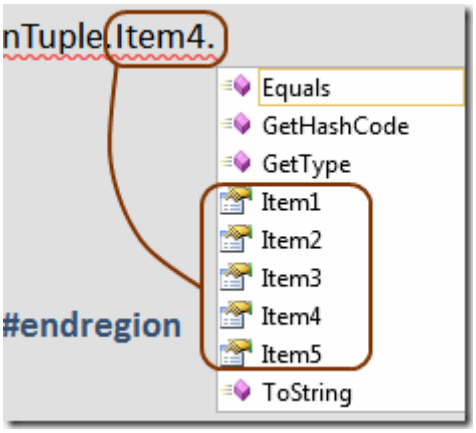
yapıcı metodu için;

```
public Tuple(
   T1 item1,
   T2 item2,
   T3 item3,
   T4 item4,
   T5 item5,
   T6 item6,
   T7 item7,
   TRest rest
)
```

Dolayısıya bu notkada sadece 8 elemanlı bir **Tuple** oluşturabileceğimizi zannedebiliriz. Ancak şöyle bir kural da yoktur; **Tuple içerisindeki item değişkenleri Tuple olamaz** Yani küçük bir hile yapıyor olacağız. Aşağıda görülen örnek kod parçasını ele alalım.

```
var nTuple = Tuple.Create(
            1
            , "Burak"
            , true
            , Tuple.Create(1, 2, 4, 6, 0)
            , new Person
            {
               PersonId = 1
               , Name = "Burak"
               , Surname = "Şenyurt"
               , City = "İstanbul"
               , Salary = 1000 }
            , 90.45F
            , 91023
            ,'A'
            );
```

nTuple isimli değişken ilk etapta 8 elemanlı görülmektedir. Ancak **4ncü** elemanın kendisi de bir **Tuple** nesne örneğidir ve 5 eleman içermektedir. Dolayısıyla bu tip bir kullanım sayesinde 8 eleman zorunluluğu aşılabilir. Tabi bu durumda **4ncü** elemana ulaşıldığında kendi altındaki elemanlarında yine **Item[indis]** isimlendirmesi ile sunulduğu görülebilir.

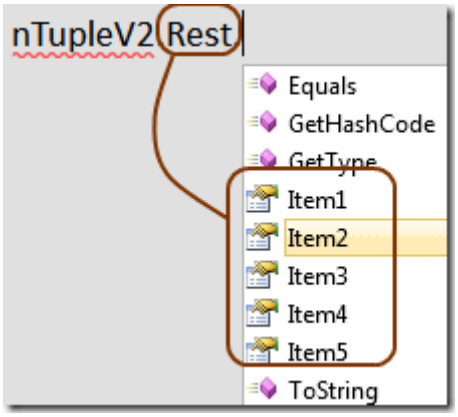


Biraz önceki örnekte **Create** metodu değerlendirilmektedir. Eğer **Constructor** metod için benzer bir durum ele alınmak istenirse en sonda yer alan **Rest** isimli parametrenin kullanılması tercih edilir. Aşağıdaki kod parçasında bu durumda ele alınmaktadır.

```
var nTupleV2 = new Tuple<byte, string, bool, Person, float, int, char, Tuple<int,int,
int, int, int>>
            (
```

```
1
, "Burak"
, true
, new Person
{
   PersonId = 1,
   Name = "Burak",
   Surname = "Şenyurt",
   City = "İstanbul",
   Salary = 1000
}
, 90.45F
, 91023
, 'A'
, Tuple.Create(1, 2, 4, 6, 0)
);
```

Burada dikkat edileceği üzere son parametrede yine bir **Tuple** nesnesi örneklenmiştir. Bu durumda **Rest** isimli özellik üzerinden, en sonda eklenen **Tuple** nesne örneği içeriği yakalanabilir.



Sanıyorum ki **Tuple** tipi ve kullanım alanları hakkında az da olsa fikir sahibi olduk. .Net Framework 4.0 ile gelen yenilikleri incelemeye devam ediyor olacğaız. Bir sonraki yazımızda görüşünceye dek hepinize mutlu günler dilerim.

TupleKavrami.rar (25,36 kb) **[örnek Visual Studio 2010 Ultimate sürümünde geliştirilmiş ve test edilmiştir]**

WCF Öğreniyorum Ders 1-Data Contracts (2010-11-27T11:04:00)

wcf,windows communication foundation,data contracts,wcf öğreniyorum,

Merhaba Arkadaşlar,

Hatırlayacağınız üzere bir süre önce NedirTv?com sponsorluğunda **WCF öğreniyorum** Webiner serimize başlamıştık. **Ders 0** kodlu ilk Webinerimizde **SOA(Service Oriented Architecture)** kavramına kısaca değinmiş, SOA ile WCF arasındaki ilişkiye bakmış ve ardından**WCF(Windows Communication Foundation)** geliştirme modelini incelemeye başlamıştık. İlk dersimizde temel olarak aşağıdaki konuları göz önüne aldığımızı ifade edebiliriz.

- Bir **WCF Servis** geliştirdik Bunu yaparken hazır **Visual Studio proje şablonları** yerine standart bir **class library** şablonu kullandık ve bu anlamda **System.ServiceModel.dll assembly**' ının önemini gördük.
- Servisimizi geliştirirken **ServiceContract** ve **OperationContract niteliklerinin(Attributes)** ne işe yaradığını öğrendik.
- **ServiceContract** niteliğinin **Name** ve **Namespace** özelliklerini kullandık.
- Servis operasyonlarımızın **primitive .net** tipleri ile çalışmasına özen gösterdik.
- Geliştirilen **WCF Servisinin** yayına alınması için bir **Console** uygulaması yazdık ve çalışma zamanını ayağa kaldırırken**ServiceHost** tipinin nasıl kullanıldığını gördük.
- özellikle servisin **host** edildiği uygulamada **Endpoint'** lerin nasıl kullanıldığına şahit olduk ve bu anlamda ilk kez **wsHttpBinding** bağlayıcı tipi ile karşılaştık.
- Servisimizi tüketecek/kullanacak olan istemciler için gerekli **proxy** üretiminde komut satırı araçlarından olan **svcutil'** den yararlandık.
- Son olarak da servis uygulamamızın çalışmasını test ettik

Ders 1 kod adlı webinerimizde ise **Veri Sözleşmeleri(Data Contracts)** kavramına değindik. Bakalım neler anlatmışız.

Sunum WCF 4.0 - Ders 1 - Data Contracts.pptx (314,15 kb)

Webiner Kaydı

Solution Dosyası Son Hali WCF Ogreniyorum.rar (81,74 kb)

Microsoft PDC İçeriğini OData Servisi Üzerinden Elde Etmek (2010-11-18T07:18:00)

wcf,odata,wcf data services,open data protocol,

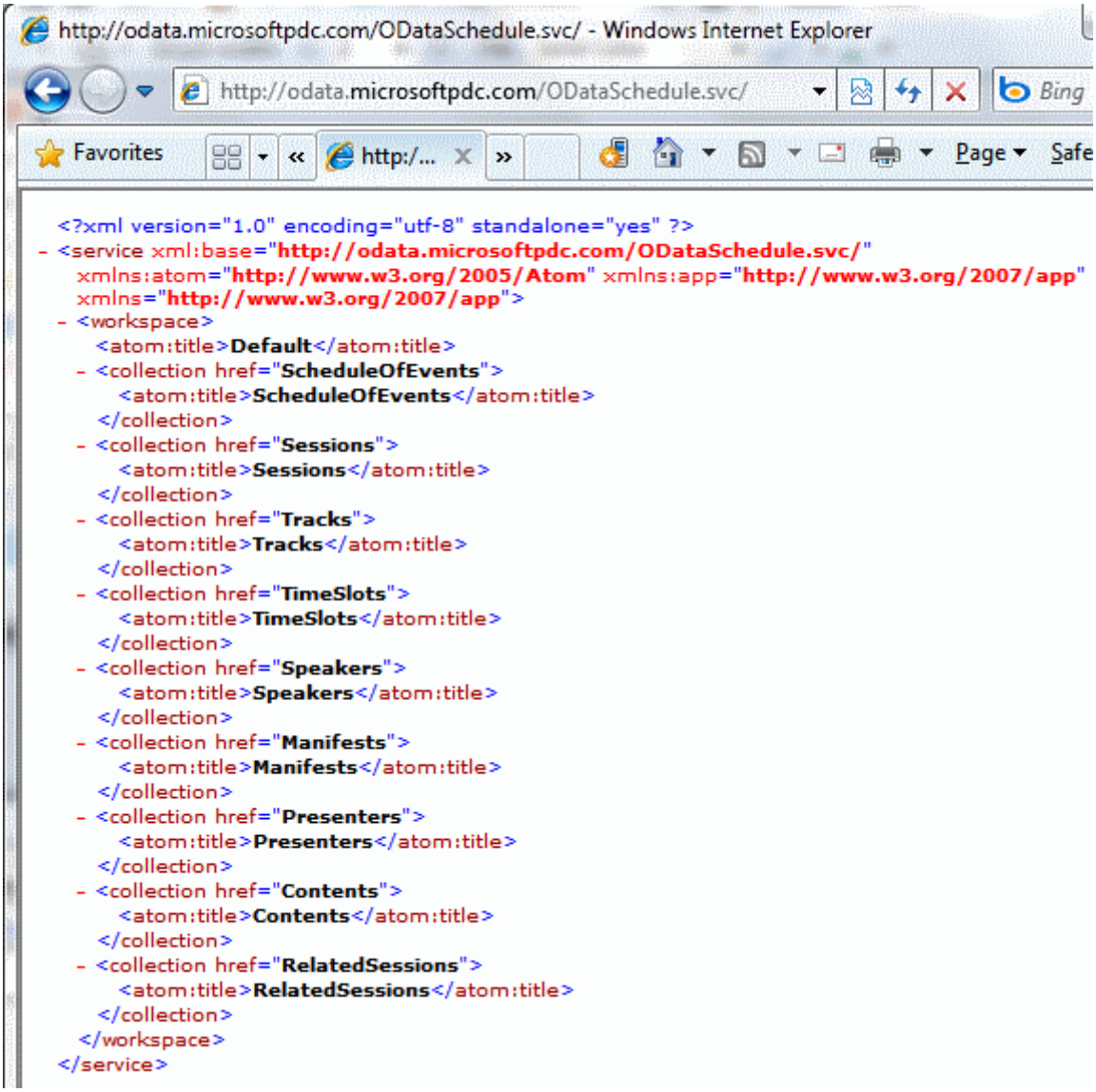
Merhaba Arkadaşlar,

Bildiğiniz üzere bir süre önce **Microsoft PDC 2010** etkinlikleri gerçekleştirildi. Online olaraktan da canlı izleyebildiğimiz sunumlarda **Microsoft'** un çok değerli sunumlarına ve anlatımlarına şahit olduk. Her **PDC** konferansında olduğu gibi bu sene yapılan etkinliklere ait görüntü kayıtları, **Download** edilmeye açıldıkları andan itibaren de ilgi odağı oldular

İlginç olan noktalardan birisi ise, **PDC'** de sunulan içeriklerin ve detaylı bilgilerinin **Open Data Protocol(ODATA)** formatında ve bir **WCF Data Service**aracılığıyla dış dünyaya sunuluyor olmasıydı.

Aslına bakarsanız bu tip bir veri paylaşımı benim gibi servis tarafı ile ilgilenen pek çok geliştirici için tek bir anlama gelmektedir : **"Git kendi uygulamanı yaz ve PDC Session bilgilerini servis aracılığıyla çek"** İşte Kurban bayramının ortasında olduğumuz şu günlerde ele aldığımız blog yazımızın konusu da bu olacak.

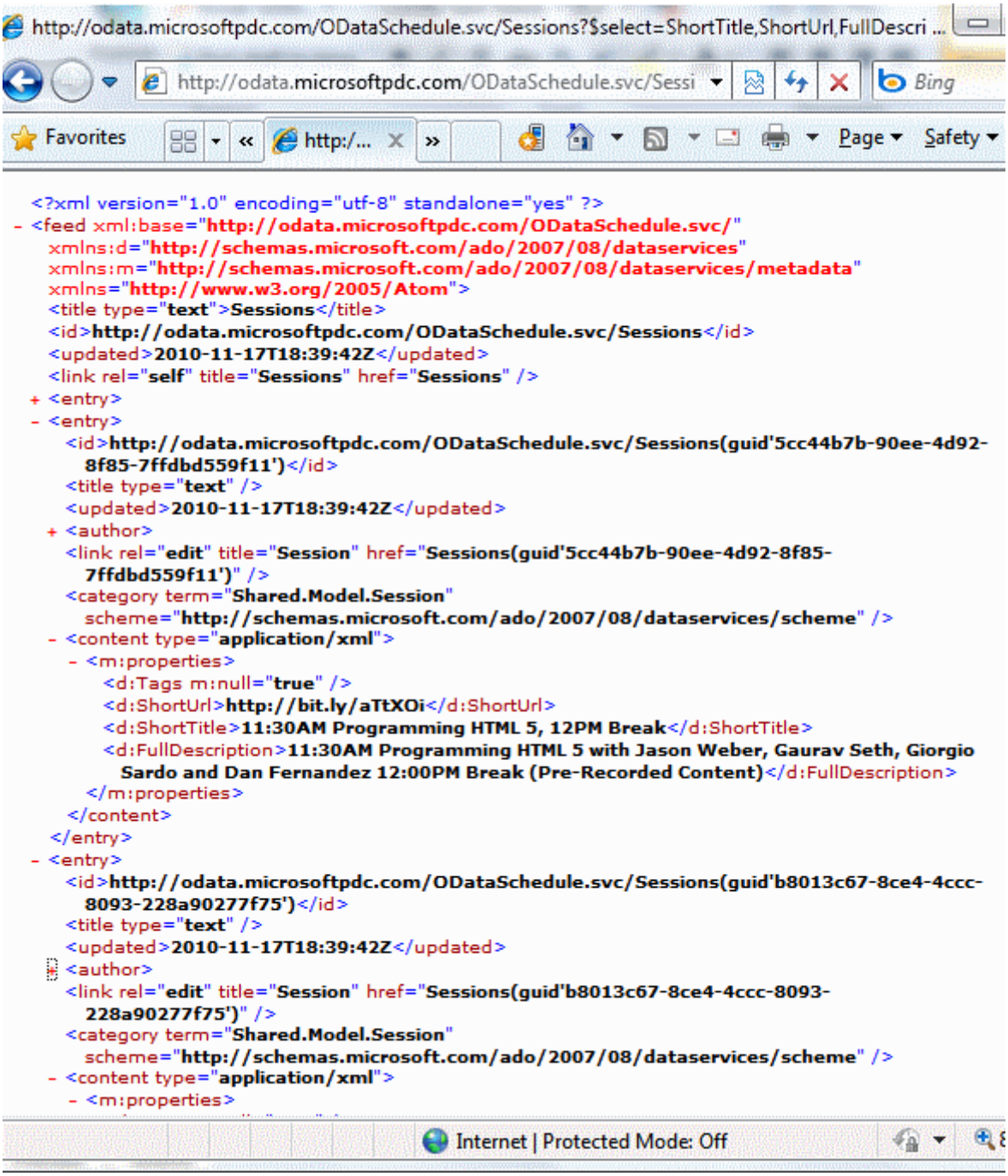
İlk olarak **PDC 2010'** a ait bilgilerin nereden yayınlandığına bakarak başlamamızda yarar olacağı kanısındayım. Şu an itibariyle http://odata.microsoftpdc.com/ODataSchedule.svc/adresinden bir paylaşım yapılmaktadır. Hatta her hangibir tarayıcı uygulaması ile baktığımızda aşağıdaki **Atom** veri içeriğinin üretildiğini görebiliriz.



Servis tarafından çekilen bu içerik sanıyorum ki **WCF Data Service** geliştiren veya kullananlara tanıdık gelecektir. Aslında bir anlamda tarayıcı üzerinden sorgulanabilir veri içeriğinin söz konusu olduğunu ifade edebiliriz. Söz gelimi

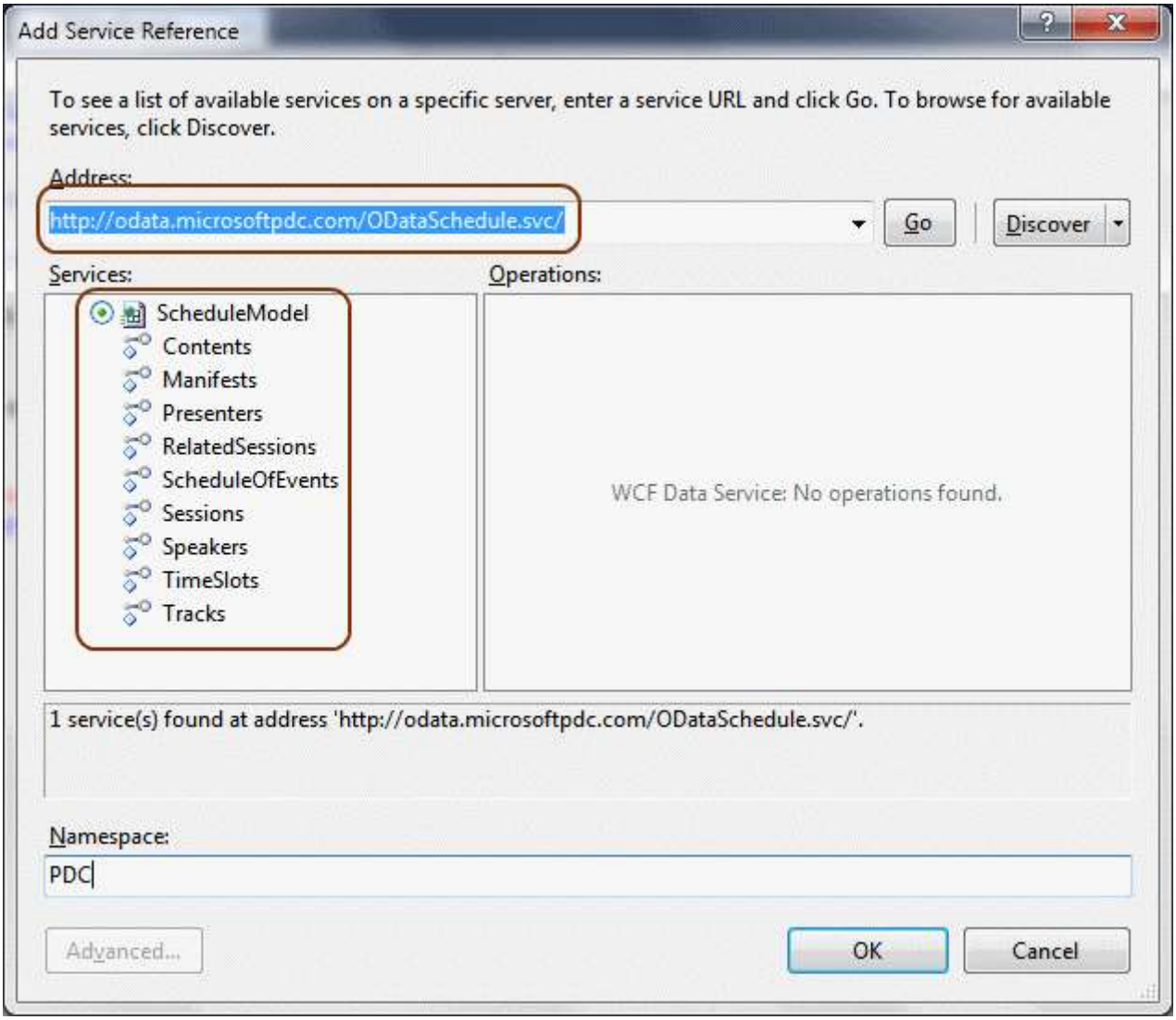
http://odata.microsoftpdc.com/ODataSchedule.svc/Sessions?$select=ShortTitle,ShortUrl,FullDescription,Tags&$orderby=ShortTitle

şeklinde bir URL sorgulamasının sonucu olarak **ShortTitle, ShortUrl, FullDescription** ve **Tags** bilgilerinden oluşan, ayrıca **ShortTitle** içeriğine göre **A' dan Z' ye sıralı** olarak gelen bir listeyi elde ederiz. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.

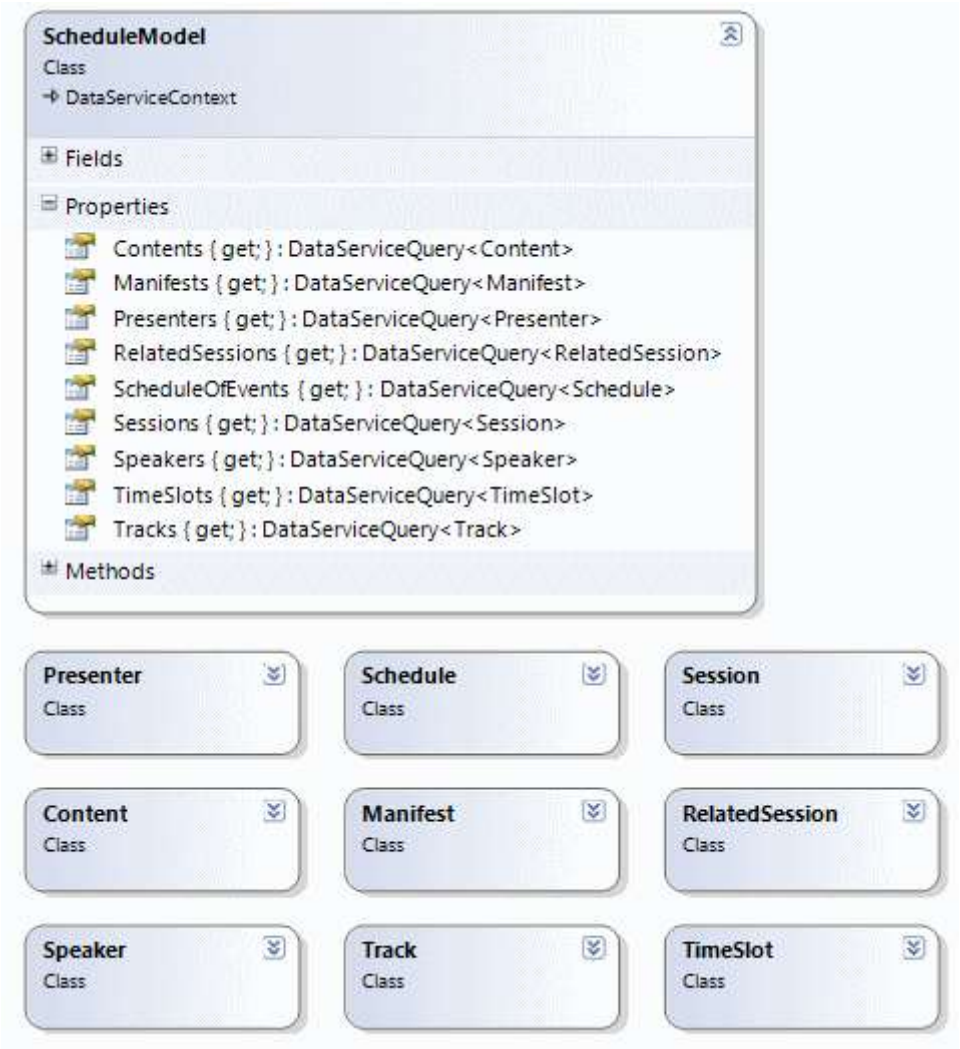


Dikkat edileceği üzere gerçekleştirilen oturumlara ait detaylı bilgileri bu servis üzerinden tedarik edebiliriz. Söz gelimi **Download** edilebilir materyallere ait bağlantı adreslerini*(Powerpoint Sunumlar, WMV ve MP4 dosyaları)*, konuşmacılara ait kısa öz geçmişleri, oturumlar ile ilişkili başlık, açıklama, kategori ve daha pek çok bilgiyi elde etme şansına sahibiz. Yazıyı hazırlarkenki ilk amacımız **Download edilebilir içeriklere ulaşmaktır.** Bunun için **Sessions** koleksiyonundan yararlanmamız yeterli olacaktır. Ancak tabiki de diğer **Entity Set** içeriklerini de değerlendirip çok daha detaylı bir arayüz uygulaması geliştirebilirsiniz *(Hatta sıkı takipçilerinden olduğum **Mike Taulty'** nin daha geçen günlerde yayınlamış olduğu oldukça iddiali bir Silverlight uygulaması da söz konusudur)*

örneği bir Web uygulaması olarak geliştirebiliriz. *(Web uygulamasını tercih etmemin en büyük sebeplerinden birisi de, Download Link' lerine doğal destek verecek olmasıdır)* İlk etapta**PDC** için geliştirilmiş **WCF Data Service'** inin projeye referans edilmesi gerekmektedir. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.



Dikkat edileceği üzere servis tarafından **ScheduleModel** isimli bir **Context** tipi getirilmektedir. **PDC** isimli **namespace** altında yer alacak **Proxy** bileşeninin üretimi sonucu **Solution** içerisine açılan tiplerin sınıf diagramı görüntüsü de aşağıdaki gibi olacaktır.



DataServiceContext türevli olan **ScheduleModel** içerisinden standart **LINQ(Language INtegrated Query)** sorgularını kullanarak ilerleyebilir ve özellikle **Sessions** özelliği ile ifade edilen koleksiyon içeriğini ele alabiliriz. Bu andan itibaren kodlama tarafında farklı şekillerde ilerlememiz de mümkündür. Söz gelimi istediğimiz veri içeriklerini bir **Web User Control** üzerinde toplayabilir veya**GridView** gibi veri-bağlı kontrollerden birisine servis yardımıyla çekebiliriz.

Biz örneğimizde Web User Control tipini kullanarak ilerlemeye gayret edeceğiz. Hatta iki adet **Web User Control** tasarlayacağımızı ifade edebilirim. Bunlardan birisi **Sessions** ile ilişkili **Title, Description,Tags, Thumbnail Photo** bilgilerini taşıyor olacak. Diğer kontrolümüz ise **Download** edilebilir içeriğe ait **Title** ve en önemlisi de indirme işlemi için gerekli bağlantı bilgilerini barındırıyor olacak. İşte **SessionInfo** isimli ilk **ascx** bileşenimiz.

SessionInfo.ascx

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="SessionInfo.ascx.cs" Inherits="PDC2010.SessionInfo" %>
<style type="text/css">
    .style1
```

```
    {
        width: 100%;
    }
    .style3
    {
        width: 84px;
        font-weight: bold;
    }
</style>

<table cellpadding="3" cellspacing="1" class="style1" frame="box"
    style="font-size: small">
    <tr>
        <td class="style3" rowspan="4" valign="top">
            <asp:Image ID="imageThumbnailPhoto" runat="server" />
        </td>
        <td class="style3">
            Title</td>
        <td>
            <asp:Label ID="labelTitle" runat="server" ForeColor="#FF3300"
Text="Label"></asp:Label>
        </td>
    </tr>
    <tr>
        <td class="style3">
            Description</td>
        <td>
            <asp:Label ID="labelDescription" runat="server" Text="Label"></asp:Label>
        </td>
    </tr>
    <tr>
        <td class="style3" valign="top">
            Tags</td>
        <td>
            <asp:Label ID="labelTags" runat="server" ForeColor="#999966"
Text="Label"></asp:Label>
        </td>
    </tr>
    <tr>
        <td class="style3" valign="top">
            Downloads</td>
        <td>
            <asp:PlaceHolder ID="holderLinks" runat="server"></asp:PlaceHolder>
        </td>
```

```
   </tr>
</table>
```

ve kod

```
using System.Web.UI.WebControls;

namespace PDC2010
{
   public partial class SessionInfo
      : System.Web.UI.UserControl
   {
      public string Title
      {
         set { labelTitle.Text = value; }
      }
      public string Description
      {
         set { labelDescription.Text = value; }
      }
      public string ThumbnailLink
      {
         set { imageThumbnailPhoto.ImageUrl = value; }
      }
      public string Tags
      {
         set { labelTags.Text = value; }
      }
      public PlaceHolder LinkHolder
      {
         get{return holderLinks;}
      }
   }
}
```

SessionInfo kontrolü içerisinde **Title, Description, Tags** bilgilerini tuttuğumuz **Label** bileşenlerinin **Text** değerlerine erişmek için bir kaç **Property** kullanılmaktadır. Bununla birlikte **Thumbnail**resmi için de bir **Image** kontrolü kullanılmakta ve bu kontrolün **ImageUrl** özelliğine **ThumbnailLink** üzerinden değer atanması sağlanmaktadır.

Her bir **Session** için **n** sayıda **Download Link** söz konusu olabilir. MP4, WMV gibi formatlardaki içeriklere ait bağlantı bilgileri **Content Entity** tipi üzerinde tutulmaktadır. Bu bilgilere erişildikten sonra yine bir **Web User Control** bileşeninden yararlanılmakta ve söz konusu bileşene ait çalışma zamanı kontrol örnekleri, **LinkHolder** özelliği

ile, **SessionInfo** üzerindeki **PlaceHolder**bileşeninin **Controls** koleksiyonuna eklenmektedir. **ContentLink** olarak adlandırdığımız bu bileşenin içeriği ise aşağıdaki gibidir.

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="ContentLink.ascx.cs" Inherits="PDC2010.ContentLink" %>
<style type="text/css">
   .style1
   {
      width: 100%;
   }
   .style2
   {}
   .style6
   {
      font-style: italic;
      font-weight: bold;
      width: 78px;
   }
   .style7
   {
      width: 87%;
   }
</style>

<table cellpadding="3" cellspacing="1" class="style1">
   <tr>
      <td class="style6">
         Title</td>
      <td class="style7">
         <i>
         <asp:Label ID="labelTitle" runat="server" Text="Label"></asp:Label>
         </i>
      </td>
   </tr>
   <tr>
      <td class="style2" colspan="2">
         <asp:HyperLink ID="linkDownloadUrl"
runat="server">Download</asp:HyperLink>
      </td>
   </tr>
</table>
```

ve kod içeriği;

```
using System;

namespace PDC2010
{
   public partial class ContentLink : System.Web.UI.UserControl
   {
      public string Title
      {
         set { labelTitle.Text = value; }
      }
      public string Url
      {
         set {linkDownloadUrl.NavigateUrl= value;}
      }
   }
}
```

Görüldüğü üzere ContentLink bileşeni üzerinde **Title** ve **Url** bilgileri tutulmakta olup bunların ilgili özelliklerine erişim için yine **Property'** lerden yararlanılmaktadır.

Bu kontroller sayesinde **WCF Data Service** tarafına gönderilen **LINQ** sorguları sonrası yüklenecek olan veri içeriklerinin, görsel bileşen bazındaki karşılıkları da tasarlanmış olmaktadır. Artık **Default.aspx** sayfasının tasarımsal ve kodsal içeriğini kodlayabiliriz

Default.aspx içeriği;

```
<%@ Page Title="Home Page" Language="C#" MasterPageFile="~/Site.master"
AutoEventWireup="true"
   CodeBehind="Default.aspx.cs" Inherits="PDC2010._Default" %>
<asp:Content ID="HeaderContent" runat="server"
ContentPlaceHolderID="HeadContent">
</asp:Content>
<asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="MainContent">
   <h2>
      Microsoft PDC 2010
   </h2>
   <p>
      <asp:PlaceHolder ID="holderSessions" runat="server"></asp:PlaceHolder>
   </p>

</asp:Content>
```

kod kısmı;

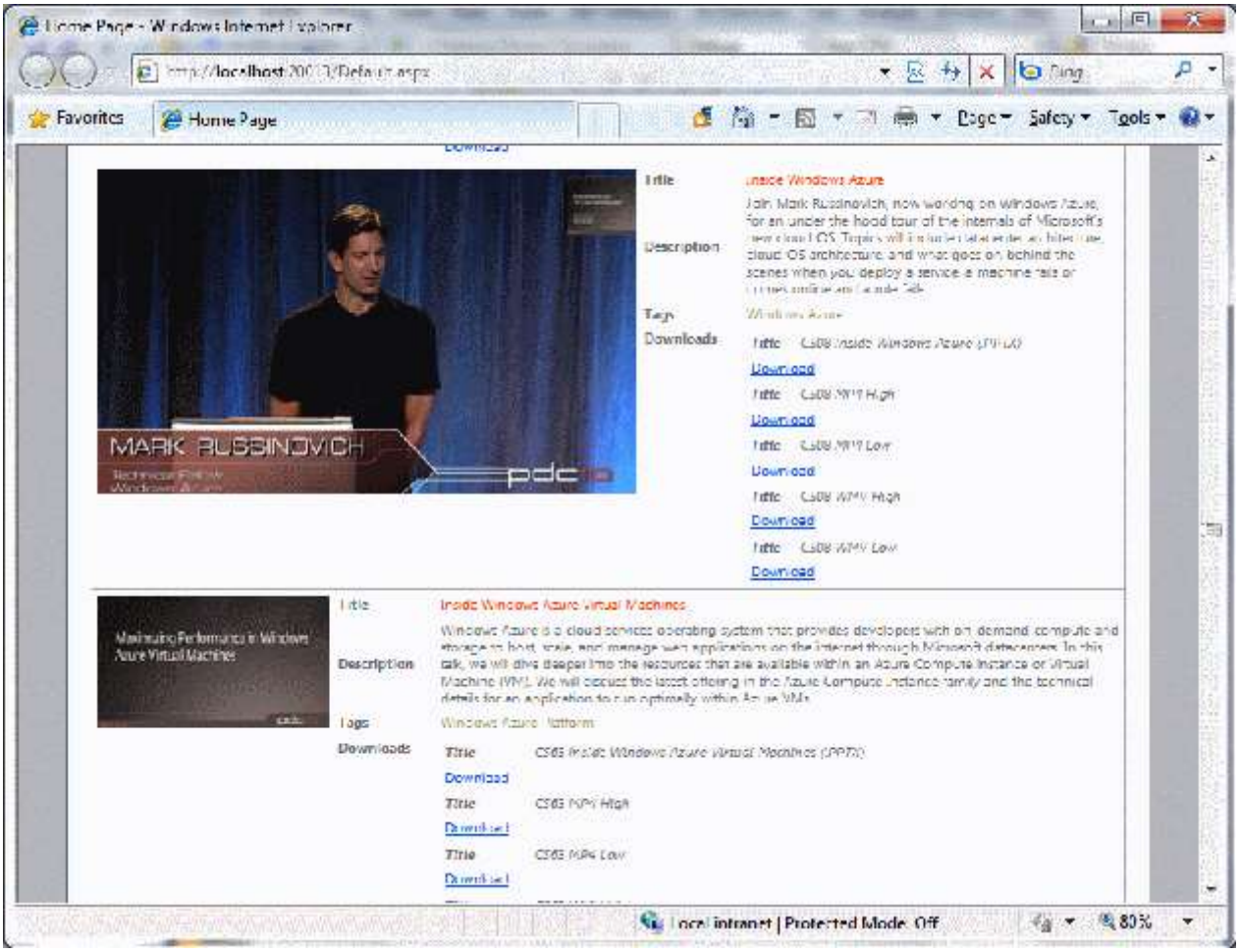
```
using System;
using System.Linq;
using PDC2010.PDC;

namespace PDC2010
{
   public partial class _Default : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         ScheduleModel model = new ScheduleModel(new
Uri("http://odata.microsoftpdc.com/ODataSchedule.svc/"));
         var sessions = from p in model.Sessions
                        orderby p.FullTitle
                        select new
                        {
                          p.FullTitle,
                           p.FullDescription,
                          p.Tags,
                          p.ThumbnailUrl,
                           p.DownloadableContent
                        };

        foreach (var s in sessions)
         {
            SessionInfo sInfo=LoadControl("~/SessionInfo.ascx") as SessionInfo;
            sInfo.Title = s.FullTitle;
            sInfo.Description = s.FullDescription;
            sInfo.ThumbnailLink = s.ThumbnailUrl;
            sInfo.Tags = s.Tags;

            if (s.DownloadableContent.Count > 0)
            {
              foreach (var c in s.DownloadableContent)
               {
                  ContentLink cLink = LoadControl("/ContentLink.ascx") as ContentLink;
                  cLink.Title = c.Title;
                  cLink.Url = c.Url;
                  sInfo.LinkHolder.Controls.Add(cLink);
               }
            }

            holderSessions.Controls.Add(sInfo);
         }
      }
```

```
    }
}
```

İlk olarak **ScheduleModule** tipine ait bir nesne örneği oluşturulduğu görülmektedir ki **yapıcı metodu(Constructor)** parametre olarak **WCF Data Service** adresini almaktadır. Sonrasında standart bir **LINQ** sorgusu yazılmış ve **Sessions Entity** içeriğine gidilerek bazı bilgilerin alınması ve bunların bir **anonymous type(İsimsiz Tip)** içerisinde birleştirilmesi sağlanmıştır. Bu akılcı bir yaklaşımdır nitekim Sessions tipi içerisindeki tüm özelliklere ihtiyacımız yoktur

Her bir **Sessions** nesne örneği üzerinden **DownloadableContent** özelliğine giderek indirilebilir içerik bilgilerinin **Title** ve **Url** bilgilerine ulaşılmaktadır. Elbette her bir **Sessions** nesne örneği için bu işlem söz konusudur. Her bir **Sessions** için bir **SessionInfo Web User Control** nesnesi örneklenirken, her bir **Content** nesne örneği için de **ContentLink Web User Control'** üne ait örneklemeler yapılmakta ve sayfaya eklenmeleri sağlanmaktadır.

çok doğal olarak sayfaya ait **Load** metodunda yaptığımız bu işlemler bir kaç saniyelik zaman kaybına neden olacaktır. Burada servisten verinin alınıp indirilmesi ve işlenmesi, süre kaybına neden olan etkenlerin başında gelmektedir. Dolayısıyla **asenkron** olarak verinin yüklenmesi ve hatta **AJAX** tabanlı bir **Web Control** içerisinde bu yükleme işleminin yapılması çok daha doğru bir yaklaşımdır. Bu kritik noktayı bir kenara bırakıp uygulamamızı çalıştırdığımızda ise aşağıdaki ekran görüntüsündekine benzer sonuçlar ile karşılaştığımızı görürüz.



Bu noktada dilerseniz **Windows** veya **WPF** tabanlı bir **Desktop** uygulaması ya da **Silverlight** tabanlı bir **Rich Internet Application**' da geliştirebilirsiniz. Servis dünyasını seviyorum Bu yazıda ele aldığımız **PDC** servisinin **OData** formatında veri içeriği sunuyor olması sayesinde tamamen platform bağımsız istemciler geliştirebilir ve tüm **PDC** içeriğini bu uygulamalar üzerinde değerlendirebiliriz. Ben kapıyı gösterdim, geçecek olan sizsiniz Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

PDC2010.rar (200,14 kb)

WCF Öğreniyorum Ders 0-Temeller (2010-11-16T01:30:00)

wcf,wcf 4.0,nedirtv?com,wcf ogreniyorum,

Merhaba Aradaşlar,

Uzun bekleyiş sona erdi ve sonbaharın gelmesi ile birlikte editörlüğünü yapmakta olduğum NedirTv?Com bünyesindeki

yeni**Webinerlerimize(WebCasts)** başlamış olduk

önümüzdeki **8 Webiner(Webcast)** boyunca sizlere **WCF(Windows Communication Foundation)** kavramını öğretmeye çalışıyor olacağım. Bu 8 bölümlük seride **Microsoft** standartlarına göre **Level 100** ile **Level 200** arasında geziniyor olacağız. Ancak **2011** başlarında ikinci bir seri ile seviyemizi **Level 200'** ün üstüne çıkartmayı da planlılyoruz

İlk bölümümüzde temel SOA kavramlarından bahsederek işe başladık ve basit bir WCF servis örneği geliştirerek bunu kullanan örnek istemci uygulamayı yazdık. Dersimizde

- **System.ServiceModel.dll**' inin önemini,
- **servis sözleşmesinin(Service Contract)** ne anlama geldiğini ve nasıl yazıldığını,
- bir Servisi Host etmek için nasıl bir kodlama yapılması gerektiğini,
- istemci tarafı için gerekli **Proxy** üretiminde komut satırı araçlarından **SvcUtil** programının nasıl kullanıldığını,
- ve diğer mevzuları öğrenmeye, anlamaya çalıştık.

Webinerimizin ekran görüntüsüne, örnek kod ve Powerpoint dosyalarına aşağıdaki linklerden ulaşabilirsiniz. Bir sonraki dersimizde görüşmek dileğiyle.

NedirTv?com üzerinden izlemek veya indirmek için

WCF 4.0 - Introduction - Microsoft.pptx (399,90 kb)

WCF Ogreniyorum.rar (64,13 kb)

MemoryCache (2010-11-08T12:34:00)

asp.net 4.0,caching,windows forms,memorycache,in memory caching,asp.net caching,

Merhaba Arkadaşlar,

Deadline…Benim gibi yazılım geliştirici olan pek coğumuzun sevmediği kelimelerin başında geldiğinden eminim Ancak kaçınılmaz bir gerçek olduğunu da biliyoruz. Her şeye rağmen onunla yaşamak veya yaşamasını öğrenmek zorundayız.

Tabi **Deadline**' lar her zaman için bir proje için söz konusu olmayabiliyorlar. Söz gelimi şu sıralar hazırlanmakta olduğum **Microsoft Teknoloji Günler Akşam Sınıfı Asp.Net 4.0** eğitiminin 3 gün öncesi de benim için

bir **Deadline***(Aslında ilke olarak her seminerin 3 gün öncesinden tam olarak hazır olmayı benimsemişimdir)* Bu deadline zamanına hızla yaklaştığım şu günlerde uykusuz geceler ile hazırlanmaya devam ediyorum. Ne varki oldukça yoğun ve zorlu bir projenin de içerisinde yer almaktayım. Ama ne demişler **"No Sacrifice No Victory"** Bakalım bu gece ki konumuz neymiş

Asp.Net tarafında performans tarafında göz önüne alınan kriterlerden birisi de ön bellekleme mekanizmalarının kullanılmasıdır. özellikle nesne tabanlı ön bellekleme işlemlerinde **Cache** tipinden sıklıkla yararlanıldığını görürüz. Bu tip yardımıyla herhangibir nesne örneğinin, içeriği ile birlikte bellek üzerinde tamponlanması mümkündür. Bu kullanıma göre ön bellekleme seçeneklerini zaman bazlı olarak değerlendirebiliriz***(Absolute Expiration, Sliding Expiration)***. Hatta özellikle **SQL** tarafında, tablo bazlı bağımlılıklar oluşturabilir ve buna göre ön bellekte tutulma sürelerini tablodaki değişikliklere bağımlı hale getirebiliriz***(SqlCacheDependency)***. Buna ilaveten dosya bazlı bağımlılıklar da oluşturmamız mümkündür***(File Based Dependency)***.

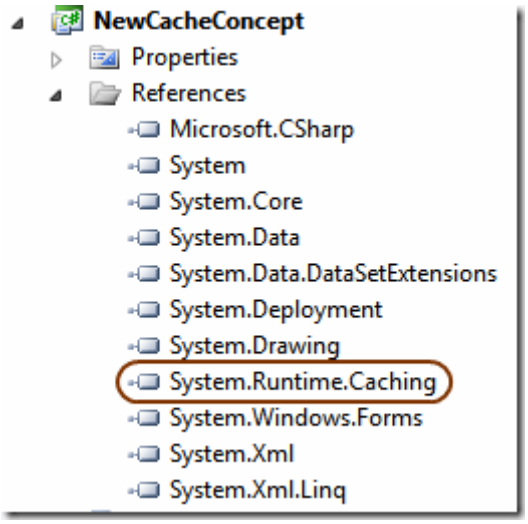
Tabi günümüzde ve öncesinde ön bellekleme için kullanılabilecek farklı alt yapılar da bulunmaktadır. Söz gelimi **Enterprise Library** ile birlikte gelen **Caching Application Block** ve hatta dağıtık mimari stratejilerini barındıran ve **Clustered** ön bellekleme modelini sunan **Velocity** kod adlı **Windows Server AppFabric Distributed Caching**. Ancak ön bellekleme işlemlerinde **Asp.Net** tarafından sunulan **built-in** ön bellekleme motoru en popüler olanlarından birisidir diyebiliriz.

Lakin özellikle **Windows Forms, WPF, Console, Class Library** gibi web dışında kalan projeler için enteresan bir durum da söz konusudur. Bu proje çeşitlerinde de istenirse **Asp.Net Cache** tipi kullanılabilir. Tek yapılması gereken söz konusu projeye **System.Web.dll assembly**' ının referans edilmesidir

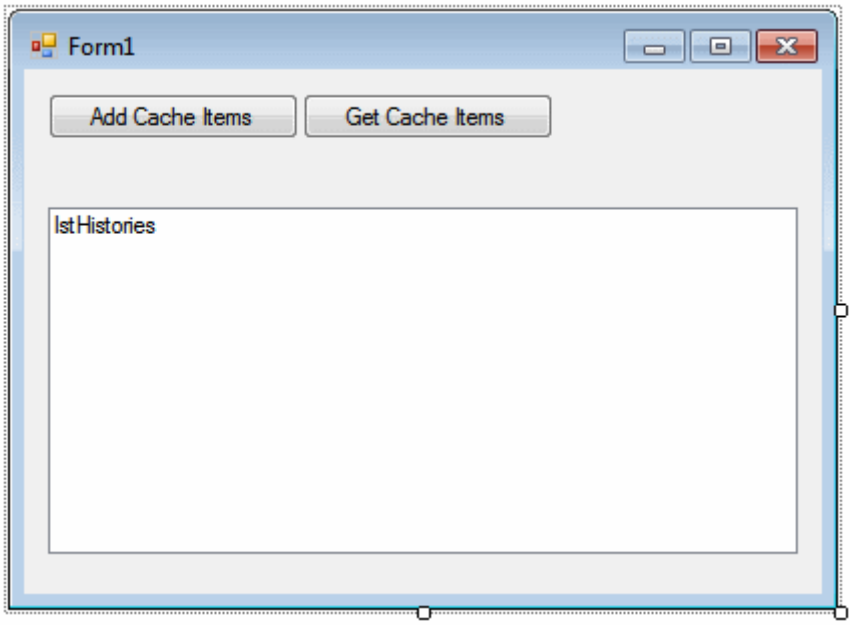
Cümlenin sonundaki **Confused Simle Emoticon**' u mutlaka dikkatinizi çekmiştir. Evet gerçekten de **Asp.Net** tarafının **System.Web.Caching** isim alanında(Namespace) yer alan **Cache** tipini örneğin bir **Windows Forms** uygulamasında kullanabiliriz. Ancak burada ters olan durum aslında **Web** tarafı için tasarlanmış ve o mimari alana ait olan bir **assembly**' ının***(System.Web.dll)***, konu ile pek alakası olmayan desktop tipinden bir uygulamaya referans edilmiş olmasıdır.

İşte bu genel sıkıntı nedeni ile **.Net Framework 4.0** sürümünde gelen yeni bir **in-memory Caching** alt yapısı mevcuttur*(In-Memory olsa da aslında özelleştirilebilir ve farklı provider' lar ile farklı kaynaklara doğru ön bellekleme işlemleri yapılabilir)*. Bu yapı aynı zamanda **Asp.Net 4.0** yenilikleri arasında değerlendirilmektedir. Buna göre **Asp.Net**' in **Caching** stratejisi, web bağımlılığından çıkartılmakta ve tüm **.Net** uygulamalarının kullanabileceği bir model haline getirilmektedir. Bu sayede, içerisinde barındırdığı temel tipler yardımıyla web programcılarının aşina olduğu ön bellekleme teknikleri, web ortamı dışındaki uygulama çeşitleri tarafından da kullanılabilir.

Yeni alt yapı(Infrastructure) **System.Runtime.Caching.dll assembly**' ı içerisinde yer almaktadır. Dilerseniz bu **Cache** tipinin kullanımını örnek bir uygulama üzerinden değerlendirmeye çalışalım. Bu amaçla basit bir **Windows Forms** uygulaması geliştiriyor olacağız. İlk olarak söz konusu uygulamaya aşağıdaki ekran görüntüsünde yer aldığı üzere**System.Runtime.Caching assembly**' ını referans ederek işe başlayabiliriz.



örnek uygulamamıza ait **Windows Form** tasarımı ise aşağıdaki gibi olabilir. Burada **Add Cache Items** isimli düğmeye basıldığında bazı nesne örneklerinin ön belleğe atılması işlemleri gerçekleştiriliyor olacaktır. Diğer taraftan **Get Cache Items** başlıklı düğmeye basıldığında ise, ön bellek üzerinde o anda durmakta olan nesne örneklerinin elde edilmesi işlemi gerçekleştirilecektir.



Windows Forms uygulamasına ait kod içeriğini ise aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.IO;
using System.Runtime.Caching;
using System.Windows.Forms;

namespace NewCacheConcept
{
   public partial class Form1 : Form
   {
      string filePath = Path.Combine(Environment.CurrentDirectory,
"ASP_NET_4_and_Visual_Studio_2010_Web_Development_Overview.pdf");
      // In-Memory Caching için kullanılacak olan nesne örneklenir
      MemoryCache mCache = MemoryCache.Default;

      public Form1()
      {
         InitializeComponent();
      }

      private void btnAddToCache_Click(object sender, EventArgs e)
      {
         // Dosya bazlı bir bağımlılık politikası üretimi
         CacheItemPolicy policy = new CacheItemPolicy(); // önce Policy tipi oluşturulur

         // Policy tipi için yeni bir dosya değişikliğini takip eden monitor nesnesi eklenir
         HostFileChangeMonitor cMonitor = new HostFileChangeMonitor(
            new List<string>
            {
               filePath
            }
            );
         // Monitor örneğin ilkelere eklenir
         policy.ChangeMonitors.Add(cMonitor);
         // ön belleğe PDF tipinden olan dosya bir byte[] dizisi şeklinde eklenir. İkinci
parametre de dependency belirtilir
         mCache.Add("Aspnet40", File.ReadAllBytes(filePath), policy);

         // Custom Type türevli bir örneğin ön belleğe eklenmesi
         Person burak = new Person();
         burak.Name = "Burak Selim";
         burak.Surname = "Şenyurt";
         burak.Salary = 1000.25;
```

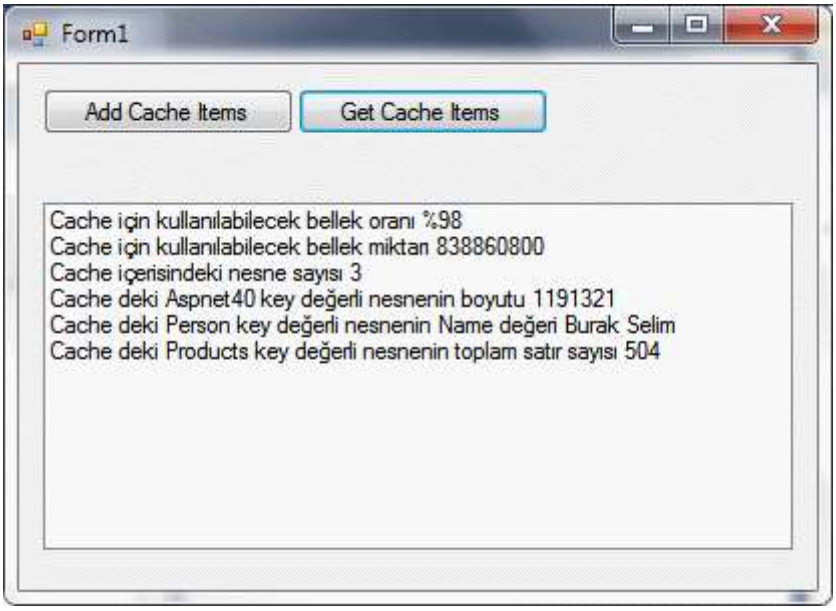
```
        burak.BirthDate = new DateTime(1976, 12, 4);
        mCache.Add("Person", burak, DateTimeOffset.Now.AddSeconds(30)); // Eklenme anından itibaren 30 saniye süreyle ön bellekte tutulacağı DateTimeOffset yardımıyla belirtilir

        // ön belleğe DataTable türünden bir nesne örneğinin eklenmesi
        DataTable table = new DataTable();
        using (SqlConnection conn = new SqlConnection("data source=.;database=AdventureWorks;integrated security=SSPI"))
        {
           SqlDataAdapter adapter = new SqlDataAdapter("Select * From Production.Product", conn);
           adapter.Fill(table);
        }
        mCache.Add("Products", table, DateTimeOffset.Now.AddMinutes(2)); // DataTable içeriği eklenme anından itibaren iki dakika süreyle ön bellekte tutulacaktır
    }

    private void btnGetCacheInfo_Click(object sender, EventArgs e)
    {
       GetDashboard();
    }

    private void GetDashboard()
    {
       // Fiziksel olarak ön bellek ile ilişkili bazı bilgilerin çekilmesi
       lstHistories.Items.Add(String.Format("Cache için kullanılabilecek bellek oranı %{0} ", mCache.PhysicalMemoryLimit));
       lstHistories.Items.Add(String.Format("Cache için kullanılabilecek bellek miktarı {0}", mCache.CacheMemoryLimit));
       lstHistories.Items.Add(String.Format("Cache içerisindeki nesne sayısı {0}", mCache.GetCount()));
    }

    private void btnGetCacheItem_Click(object sender, EventArgs e)
    {
       GetDashboard();
       // byte[] tipinden ön belleklenmiş olan PDF dosyasının elde edilişi
       CacheItem cItem=mCache.GetCacheItem("Aspnet40");
       if (cItem != null)
       {
          byte[] cValue = cItem.Value as byte[];
          lstHistories.Items.Add(String.Format("Cache deki {0} key değerli nesnenin boyutu {1}", cItem.Key, cValue.Length));
       }
```

```
        // Person tipinden ön belleklenen nesnenin elde edilişi
        CacheItem cItemPerson = mCache.GetCacheItem("Person");
        if (cItemPerson != null)
        {
            Person cachedPerson = cItemPerson.Value as Person;
            lstHistories.Items.Add(String.Format("Cache deki {0} key değerli nesnenin
Name değeri {1}", cItemPerson.Key, cachedPerson.Name));
        }

        // DataTable tipinden ön belleklenen nesnenin elde edilişi
        CacheItem cItemTable = mCache.GetCacheItem("Products");
        if (cItemTable != null)
        {
            DataTable table = cItemTable.Value as DataTable;
            lstHistories.Items.Add(String.Format("Cache deki {0} key değerli nesnenin
toplam satır sayısı {1}", cItemTable.Key, table.Rows.Count));
        }

        // Belirli key bilgilerine ait değerlerin çekilmesi
        var values = mCache.GetValues(keys: new string[] { "Person", "Products" });
    }
  }
}
```

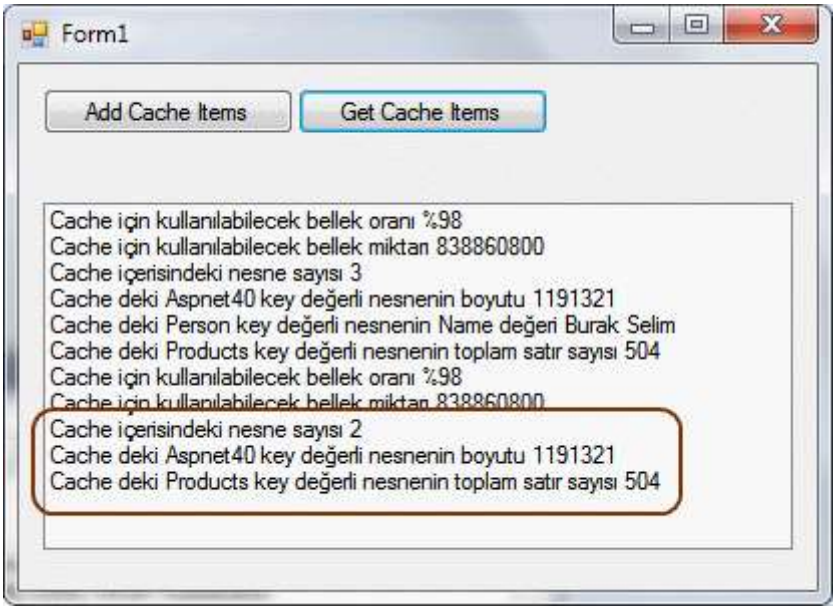
Olayın kalbi **MemoryCache** isimli nesne örneğidir. Bu nesne örneğinden yararlanılarak **byte[], Person** ve **DataTable** tipinden nesne örneklerinin ön belleğe atılması işlemleri gerçekleştirilmektedir. özellikle **PDF** formatındaki dosya içeriğinin*(ki içerisinde Asp.Net 4.0 ile gelen yenilikler anlatılmaktadır)* **byte[]** dizisi olarak belleğe atılışı sırasında bir de **ilke(Policy)** uygulandığına dikkat edilmelidir. öyleki bu ilkeye göre bir dosya bağımlılığı oluşturulmuştur. Gerçi **PDF** dosyasının değiştirilmesi pek söz konusu olmasa da önemli olan bağımlılığın nasıl yaratıldığıdır. Bu, tipik olarak dosya da değişiklik olması halinde ön bellekte duran nesnenin düşürülmesi anlamına gelmektedir. Yani web tarafından aşina olduğumuz **File Dependency** olayı

Person ve **DataTable** tipinden olan nesne örneklerinin ön bellekte tutulması ise kesin süre sonlu olarak bildirilmiştir***(Absolute Time Expiration)***. Yani ön belleğe atılan nesne örnekleri belirlediğimiz süre kadar tutulacaklardır. Dikkat edilmesi gereken noktalardan bir diğeri de, uygulamanın çalıştığı makinenin ön bellekleme için kullanılabilecek alan değerlerinin yüzdesel ve miktarsal olarak elde edilebiliyor olmasıdır.

Şimdi örneğimizi çalıştırarak ilk testimizi yapalım.

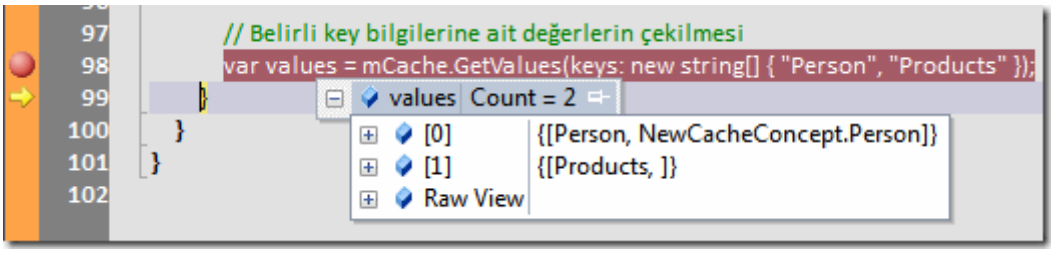


Dikkat edileceği üzere ön belleğe atılan **3** nesne örneği söz konusudur ve bunlara ait bir takım bilgilerde elde edilebilmektedir. Söz gelimi **PDF dosyasının boyutu**, **Person nesne örneğinin Name değeri** ve **DataTable içerisinde duran toplam satır sayısı** gibi. Diğer yandan önem arz eden konulardan biriside **Absolute Expiration** sürelerinin dolması halinde ne olacağıdır. Söz gelimi **Person** nesne örneği ön belleğe atıldıktan sonra **30 saniye boyunca yaşayabilir**. Eğer süre dolduktan sonra bellek üzerinde kalan nesnelere bakılırsa **Person** nesne örneğinin artık olmadığı rahatlıkla görülebilir.*(Diğer yandan uygulama kapatıldığı takdirde ön belleğe atılan tüm nesnelerin geçerliliği ortadan kalkacaktır. Bir başka deyişle süreleri veya bağımlılıkları bozulmasa dahi, uygulama açık iken eklenen ön bellek nesnelerine, uygulama tekrar açıldığında erişilemeyecektir)*



Şu an itibariyle ön bellekte 2 nesne örneği yer almaktadır ve **Person** tipine ait örnek bunların arasında değildir. Nitekim kendisi için belirlenen ön bellekte yaşama süresi aşılmıştır.

Aslında ön bellekte duran n sayıda nesne olduğunda bunlardan sadece belirli **Key** adlarına sahip olanlarının elde edilmesi de bazı senaryolarda işimize yarayabilir. Bu noktada **MemoryCache** nesne örneği üzerinden çağırılabilen **GetValues** metodu oldukça işe yaramaktadır. Aşağıdaki **Debug** zamanı ekran görüntüsünde bu nesne örneği içerisinde **Person** ve **Products**adları ile işaret edilen nesne örneklerine erişilebildiği işaret edilmektedir.



Görüldüğü üzere **MemoryCache** nesne örneğinden yararlanarak **Asp.Net** tarafında sık kullanılan **in-memory Caching** tekniğini, **System.Web.dll assembly**' ına bağımlı olmadan web harici herhangibir uygulamada ele alabilmekteyiz. **MemoryCache** nesnesinin kullanımı ile ilişkili olarak detaylı referans bilgisine MSDN üzerinden ulaşabilirsiniz. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ASPNET40Ogreniyorum.rar (2,31 mb) **[örnek Visual Studio 2010 Ultimate sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Microsoft Teknoloji Günleri Akşam Sınıfı 7-Asp.Net 4.0 ile Gelen Yenilikler (2010-11-05T11:47:00)

asp.net 4.0,microsoft teknoloji günleri,

Etkinlik ile ilişkili download bilgileri aşağıdaki gibidir

Sunum : ASP.NET 4.0 - Introduction.pptx (550,34 kb)

örnek Solution (Whitepaper dökümanını da içerir) : ASPNET40Ogreniyorum.rar (4,20 mb)

Ekran görüntüsü kayıtlarını indirmek için tıklayın

Merhaba Arkadaşlar,

Benim için yine hareketli bir **Kasım** ayı. çıraklık başvurularının değerlendirilmesi, blog paylaşımları, şirkette yürütülmekte olan yoğun projeler, **S(h)arp Efe** bu hareketliliği arttıran en büyük nedenlerin başında gelmekte. Bir diğer sebepte, **23 Kasım 2010 Salı** günü ~~gerçekleştirilecek~~ gerçekleşen olan **Microsoft Teknoloji Günleri Akşam Sınıfı** etkinliklerinin **7ncisi.** 7nci eğitimimizde **ASP.NET 4.0** ile birlikte gelen yenilikleri

ele ~~alıyoruz~~ aldık. **Gerçi bu konunun asıl uzmanı MVP Uğur Umutluoğlu hocamızdır.** Ancak bende elimden geldiğince katılan arkadaşlarımızı bilgilerimle aydınlatmaya ~~çalışacağım~~ çalıştım.

ASP.NET ' in, her sürümü ile birlikte **Major** değişikliklerin hakkını fazlasıyla veren bir ürün olduğunu hepimiz biliyoruz Bu eğitimimizde her ne kadar 3 saatlik kısa bir zamanımız olsa da gelen yeniliklerin çoğu üzerinde ~~tartışıyor ve konuşuyor olacağız~~ tartışma fırsatımız oldu. Aslında **ASP.NET 4.0** tarafına baktığımızda aşağıdaki listede yer alan uzun yeniliklerin olduğunu görmekteyiz. Tüm bu yenilikleri aşağıda maddeler halinde bulabilirsiniz.

Core Services

Web.config File Refactoring
Extensible Output Caching
Auto-Start Web Applications
Permanently Redirecting a Page
Shrinking Session State
Expanding the Range of Allowable URLs
Extensible Request Validation
Object Caching and Object Caching Extensibility
Extensible HTML, URL, and HTTP Header Encoding
Performance Monitoring for Individual Applications in a Single Worker Process
Multi-Targeting

Ajax

jQuery Included with Web Forms and MVC
Content Delivery Network Support
ScriptManager Explicit Scripts

Web Forms

Setting Meta Tags with the Page.MetaKeywords and Page.MetaDescription Properties
Enabling View State for Individual Controls
Changes to Browser Capabilities
Routing in ASP.NET 4
Setting Client IDs
Persisting Row Selection in Data Controls
ASP.NET Chart Control
Filtering Data with the QueryExtender Control
Html Encoded Code Expressions
Project Template Changes
CSS Improvements
Hiding div Elements Around Hidden Fields

Rendering an Outer Table for Templated Controls
ListView Control Enhancements
CheckBoxList and RadioButtonList Control Enhancements
Menu Control Improvements
Wizard and CreateUserWizard Controls 56
Html Encoded Code Expressions
Project Template Changes
CSS Improvements
Hiding div Elements Around Hidden Fields
Rendering an Outer Table for Templated Controls
ListView Control Enhancements
CheckBoxList and RadioButtonList Control Enhancements
Menu Control Improvements
Wizard and CreateUserWizard Controls 56

Visual Studio 2010 Web Development Improvements

Improved CSS Compatibility
HTML and JavaScript Snippets
JavaScript IntelliSense Enhancements

Web Application Deployment with Visual Studio 2010

Web Packaging
Web.config Transformation
Database Deployment
One-Click Publish for Web Applications
Resources

Elbette kısıtlı olan zamanımız bize, tüm bu yenilikler üzerinde konuşma fırsatı ~~vermeyecektir~~ vermedi. Ancak en azından önemli olanlarını ve özellikle **Search Engine Optimization(SEO)** için getirilen iyileştirmeleri analiz etmeye ~~çalışıyor olacağız~~ çalıştık. ~~Etkinliğimize katılmak için buradan gideceğiniz kayıt formunu doldurmanız yeterli olacaktır.~~

Katılan tüm arkadaşlarımıza canı gönülden teşekkürlerimi sunuyorum. Tekrardan görüşmek dileğiyle...

Minicik Session İçeriği (2010-11-03T00:10:00)

asp.net,asp.net 4.0,session,session compression,

Merhaba Arkadaşlar,

Bildiğiniz üzere bir süredir **Microsoft Teknoloji Günleri Akşam Sınıfı** etkinliklerini gerçekleştirmekteyiz. **Kasım** ayının konusu ise hemen her sürümünde köklü ve önemli yenilikler ile birlikte gelen **Asp.Net' in 4.0 sürümü** Web programlamanın gelişen ihtiyaçları nedeniyle **Asp.Net** alt yapısında da her major sürümde fazlasıyla yenilik bulunmakta.

Tabi yine bildiğiniz üzere bendeniz web konusunda uzman değilim. Ağırlıklı olarak **servis yönelimli mimari(Service Oriented Architecture)** tarafı ile ilgilenmekteyim. Ancak eğitmenlik yaptığım dönemlerden kalan bir hastalık olsa gerek, **.Net**' in diğer pek çok alanı ile de ilgilenmekteyim. Eğitime blog yazısını hazırladığım tarih itibaryile oldukça az bir süre kaldı. **çıraklık başvuruları, Microsoft gönüllü çalışmaları, 11 haftalık proje planı olup bizden 4 haftada bitirmemizi istedikleri kocaman POC çalışması, S(h)arp Efe** derken zamanı etkin kullanmak konusunda sıkıntılar yaşamaya başladım

Aslında **.Net Framework 4.0** tarafını uzun süredir incelememe rağmen, en ince detaylarına kadar girmeden konuya hakim olmanın zor olacağını da gayet iyi biliyordum. Hatta bana göre yazarak anlatmak öğrenmenin en iyi yollarından birisi. İşte bu bi dolu düşünce altında başladığım gece çalışmasının sonucu olan küçük bir blog girdisi ile karşınızdayım. Bu yazımızda **Asp.Net 4.0** tarafında gelen önemli yeniliklerden birisi olan serileştirilebilir **Session** içeriğini ufaltmak*(daha teknik bir tabirle sıkıştırmak)* konusuna değiniyor olacağız.

Session bildiğiniz üzere web çalışma modelinde önemli bir yere sahip. özellikle oturum bazlı olarak veri tutulması istenen durumlarda, sunucu tarafında kullanılabilecek seçeneklerden birisi olduğunu ifade edebiliriz. **Session** içeriklerini kullanmak da son derece basit. Bu anlamda **Asp.Net**' in başlangıcından beri var olan **HttpSessionState** tipinden yararlanıldığını biliyoruz. Tabi**Session** kullanımında dikkat edilmesi gereken bazı hususlarda var. Söz gelimi varsayılan olarak **In-Proc** mod adı verilen ve çalışmakta olan **Asp.Net** uygulamasına ait **Worker Process** ile ilişkili bellek alanlarında tutulan **Session** içeriklerinin, **SQL** veritabanı veya farklı bir **State Service**' e ait process altında tutulması da söz konusu. Bu farklı tutuluş şekillerine ilaveten **object** tipi ile

çalışabilen bir yapıdan bahsettiğimizi de ifade etmek isterim. Yani her tür .Net nesnesini atayabilirsiniz. Hımmm

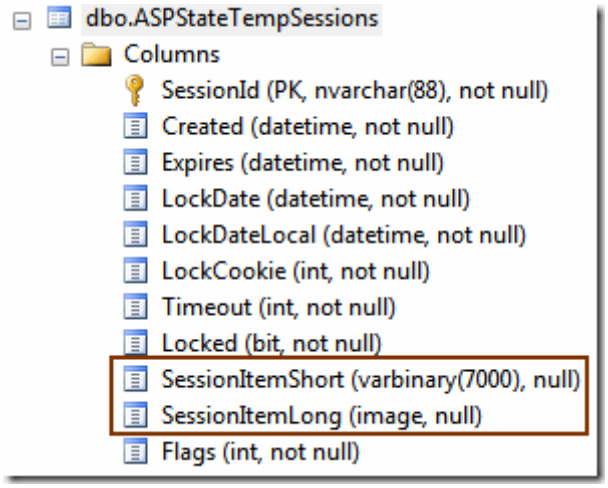
Şimdi **In-Proc** mod dışındaki modları göz önüne alalım. **SQL** sunucusu*(SQLServer modu)* veya **State Service(StateServer modu)** kullanan modları. Bu modlar kullanıldığında **Session**içerisine atılan verilerin serileştirilerek tutulması söz konusudur. Bu son derece mantıklıdır çünkü **object** tipi içerisine çok büyük boyutlu nesnelerin dahi atılması mümkündür. Ancak bu durum zamanla **SQL** veya **State Service** modlarının kullandığı alanların önemli ölçüde şişmesine de neden olabilir. İşte **Asp.Net 4.0** ile **Session** ayarlamaları için eklenen**compressionEnabled** özelliği sayesinde, söz konusu **Session** içeriklerinin önemli ölçüde sıkıştırılması da mümkündür. Bu noktada **System.IO.Compression.GZipStream** tipinin büyük bir rol oynadığını ifade edebiliriz. Nitekim bu tip sayesinde serileştirilmiş olan verinin sıkıştırılması söz konusudur.

Yazımızın ilerleyen kısımlarında sıkıştırmanın bu noktadaki önemini vurgulamak açısından basit bir örnek üzerinden ilerlemeye çalışıyor olacağız. örnek senaryomuzda **Session** bilgilerini **SQL** veritabanı üzerinde tutuyor olacağız. Bu amaçla bir ön hazırlık yapmamız gerekiyor. **Visual Studio 2010 Command Prompt** üzerinden **aspnet_regsql**aracını kullanarak söz konusu veritabanını*(örneğimizde SessionDB)* oluşturabiliriz.

Kurulum ;

```
C:\Program Files (x86)\Microsoft Visual Studio 10.0\VC>aspnet_regsql -S localhost -E -ssadd -sstype c -d SessionDB

Start adding session state.

....

Finished.

To use this custom session state database in your web application, please specify it in the configuration file by using the 'allowCustomSqlDatabase' and 'sqlConnectionString' attributes in the <system.web>\<sessionState> section.
```

Bu kurulum işlemi sonrasında **SessionDB** içerisinde oluşturulan **ASPStateTempSessions** tablosunun yapısı da aşağıdaki gibi olacaktır. İki alan; **SessionItemShort** ve **SessionItemLong** şu an ki vakamız için önem arz etmektedir. Bu alanlarda serileştirilen **Session** içerikleri tutulmaktadır. Bununla birlikte serileşen içeriğin büyüklüğüne göre iki alandan bir tanesine veri eklenmesi de söz konusu olacaktır.



Şimdi basit bir **Asp.Net Web Application** üzerinden ilerleyelim. Söz konusu web uygulamasında en önemli nokta **Session** kullanımına ait konfigurasyon ayarlarıdır. **Default.aspx** sayfamız son derece basit bir fonksiyonelliğe sahiptir. **Button** kontrolüne basıldığında **Logo** isimli bir sınıf örneğinin **Session** nesnesi olarak atılması söz konusudur. İşte örnek uygulama kodlarımız.

```
using System;
using System.IO;
using System.Data.SqlClient;
using System.Data;

namespace OldStyleSession
{
   [Serializable]
   public class Logo
   {
      public string FileContent { get; set; }
      public string Name { get; set; }
      public DateTime CreationTime { get; set; }
   }

   public partial class _Default
      : System.Web.UI.Page
   {
      protected void btnAddToSession_Click(object sender, EventArgs e)
      {
         Logo lg = new Logo
         {
            Name="Car"
            , CreationTime=DateTime.Now
            , FileContent=File.ReadAllText(Server.MapPath("/Bilgiler.txt"))
         };
```

```
            Session.Add("Logo", lg);
        }

        protected void btnWriteSession_Click(object sender, EventArgs e)
        {
            using (SqlConnection conn = new SqlConnection("data
source=.;database=SessionDB;integrated security=SSPI"))
            {
                SqlCommand cmd = new SqlCommand("Select
SessionItemShort,SessionItemLong From AspStateTempSessions Where
SessionId=@SessionId", conn);
                cmd.Parameters.AddWithValue("@SessionId", txtSessionId.Text);
                conn.Open();
                SqlDataReader reader =
cmd.ExecuteReader(CommandBehavior.CloseConnection);
                if (reader.Read())
                {
                  var itemShort=reader.GetSqlBytes(0);
                   var itemLong = reader.GetSqlBytes(1);
                   Response.Write(
                      String.Format("SessionItemShort length = {0}, SessionItemLong length =
{1}"
                      ,itemShort.IsNull?0:itemShort.Length
                      ,itemLong.IsNull?0:itemLong.Length
                      )
                      );
                }
                reader.Close();
            }
        }
    }
}
```

Görüldüğü üzere **Session** özelliğinin **Add** metodundan yararlanılarak **lg** isimli **Logo** nesne örneğinin ilave edilmesi söz konusudur. Diğer yandan bir de üretilen **Session** için yazılan**SessionItemLong** ve **SessionItemShort** alanlarının içeriklerinin **byte** cinsinden uzunluklarını ekrana yazdırabileceğimiz bir metodumuz da bulunmaktadır. Bu metodu üretilen **Session** içeriklerinin boyutlarını kıyaslamak için kullanıyor olacağız.

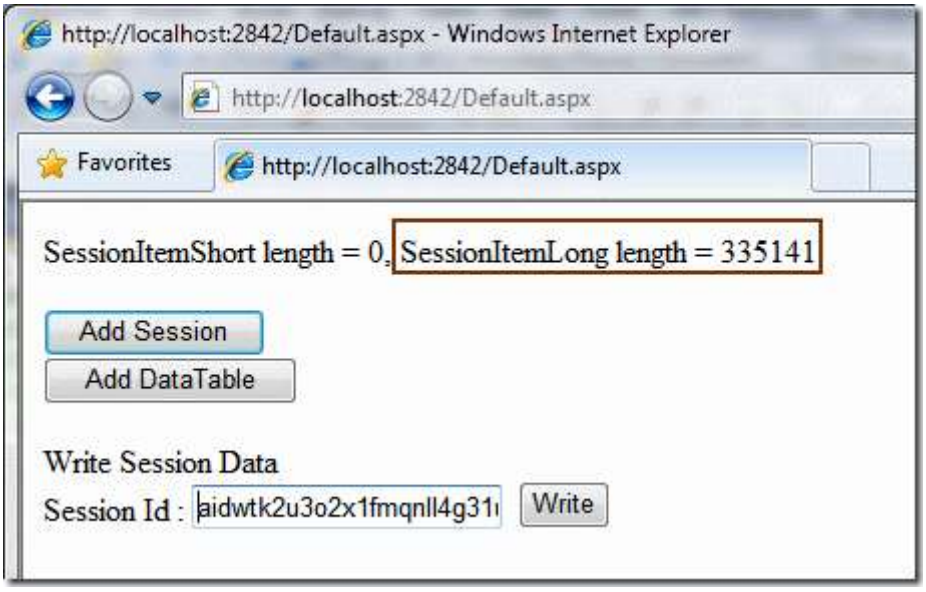
örnekte kullanılan **Bilgiler.txt** isimli **Text** tabanlı dosya **328 Kb** büyüklüğünde anlamsız bir içeriğe sahiptir. Burada **text** tabanlı bir içerik söz konusu olduğundan sıkıştırma işlemini ele alacak olan algoritma çarpıcı sonuçlar üretecektir. İlk etapta **web.config** dosyasının içeriğini aşağıdaki gibi tasarladığımızı düşünebiliriz.

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <sessionState allowCustomSqlDatabase="true" sqlConnectionString="data
source=.;database=SessionDB;integrated security=sspi" mode="SQLServer"
compressionEnabled="false"/>
    <compilation debug="true" targetFramework="4.0">
    </compilation>
  </system.web>
</configuration>
```

SessionDB isimli bir veritabanı adı kullandığımızdan **allowCustomSqlDatabase** özelliğine **true** değeri atanmıştır. **sqlConnectionString** bilgisinde ise **Session** bilgilerinin yazılacağı veritabanı bağlantısı belirtilmektedir. **mode** niteliğine atanan değer ile **Session** içeriklerinin **SQL** veritabanı üzerinde*(sqlConnectionString ile belirtilen bağlantıya doğru)* tutulacağı ifade edilmektedir. Şu an için **compressionEnabled** özelliğine **false** değeri atanmıştır. Bu şekilde aslında **Asp.Net 4.0** öncesinde olduğu gibi standart bir **Session** tutma işlemi yapılacağı belirtilmektedir. örneği bu şekilde test ettiğimizde **ASPStateTempSessions** tablosunda üretilen **Session** satırına ait sorgulama sonucu, **SessionItemLong** alanına **binary** bir içeriğin serileştirilmiş olduğu görülecektir.

Sql tarafında Compression kullanılmadığı haldeki durum;

İlk olarak sıkıştırma işlemini kullanmadığımız durumu ele alalım. **Add Session** işleminden sonra, üretilen **SessionID** değerini web sayfamız üzerinde kullanırsak aşağıdaki sonuçlar ile karşılaşırız.

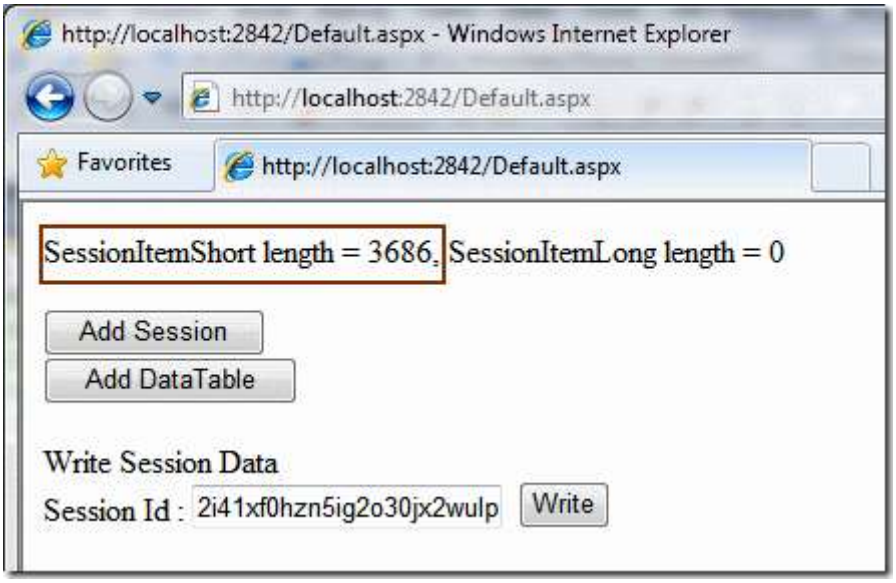


Dikkat edileceği üzere **335141** uzunluğunda bir byte içeriği söz konusudur.

Gelelim sıkıştırılma durumuna. Bu sefer **compressionEnabled** özelliğine **true** değerini vermemiz yeterli olacaktır.

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <sessionState allowCustomSqlDatabase="true" sqlConnectionString="data
source=.;database=SessionDB;integrated security=sspi"
mode="SQLServer" compressionEnabled="true"/>
    <compilation debug="true" targetFramework="4.0">
    </compilation>
  </system.web>
</configuration>
```

Sql tarafında Compression kullanılması haldeki durum ise aşağıdaki gibi olacaktır;



Görüldüğü üzere bir önceki vakanın tersine serileştirilebilir içerik **SessionItemLong** alanı yerine **SessionItemShort** içerisine eklenmiştir. Bununla birlikte söz konusu sıkıştırılmış verinin içeriği **3686** dır

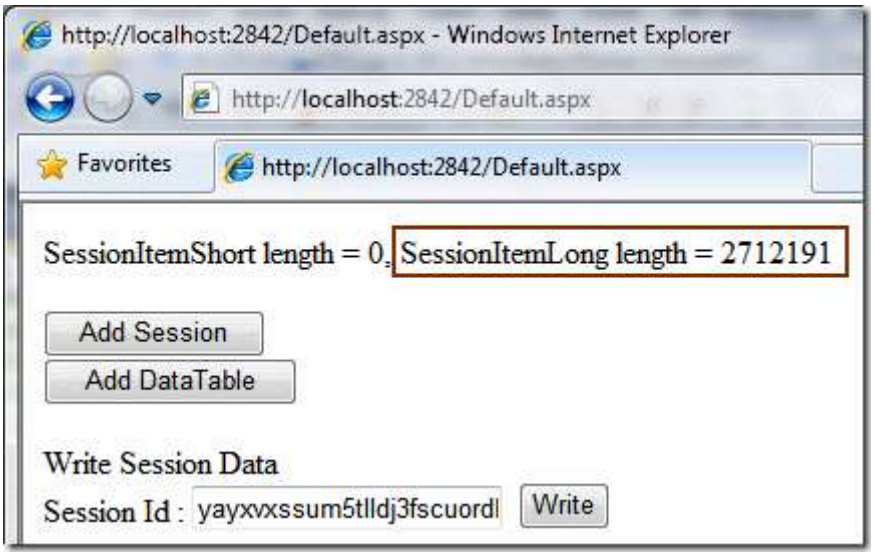
Sıkıştırma algoritmasının uygulanması sonucu neredeyse
standart **session** içeriğinin **%0,0109983559158682 üne** kadar verinin küçültülmesi söz konusudur. Elbette burada **text**tabanlı bir içerik kullanıldığından söz konusu farkın oluşması doğaldır. özellikle **binary** içerikten oluşan*(örneğin resim formatı gibi)* bir verinin serileştirilmesi esnasında veri boyutlarında önemli bir fark olmayadabilir. Bu durumu analiz etmek için aşağıdaki kod parçasını göz önüne alalım.

```
protected void btnAddDataTable_Click(object sender, EventArgs e)
{
   using (SqlConnection conn = new SqlConnection("data
```

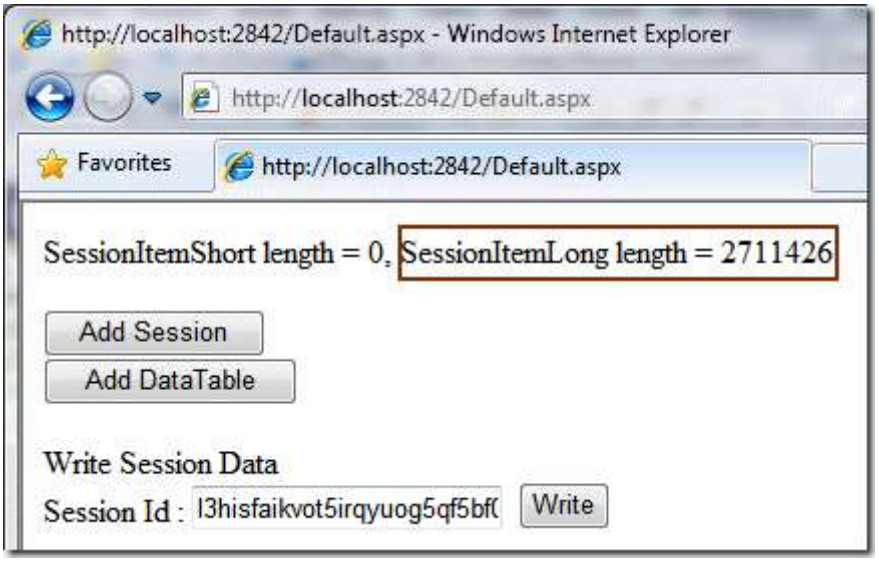
```
source=.;database=AdventureWorks;integrated security=SSPI"))
   {
      using (SqlDataAdapter adapter = new SqlDataAdapter("select * from
Production.ProductPhoto", conn))
      {
         DataSet ds = new DataSet("ProductPhoto");
         adapter.Fill(ds);

         Session.Add("ProductPhoto", ds);
      }
   }
}
```

Bu kez **ProductPhoto** isimli tablonun tüm içeriğini çektiğimiz bir **DataSet** örneğini **Session**' a ekliyoruz. Yine ilk olarak **compressionEnabled** niteliğinin **false** değere sahip olduğu ve sonrasında**true** olması halini göz önüne alalım. Bu durumda aşağıdaki örnek çıktılar elde edilecektir.

Sıkıştırılma kapalı iken;



Sıkıştırılma açık iken;



Sıkıştırılmama durumunda **2712191** iken sıkıştırılma durumunda **2711426**. Yani sadece **% 1,000282139361355** oradanın bir sıkıştırma söz konusu olmakta. Değer mi? Değmez. Buna göre serileştirilebilir içeriklerin sıkıştırılabilir olmaları da önem kazanmaktadır. *(Nitekim bir PDF veya JPEG dosyasını sıkıştırdığınızda önemli ölçüde bir sıkıştırma olmadığını görürüz)* İnandırıcı geldi mi?

Varsayımsal Yaklaşım

Tahmini olarak **Asp.Net** çalışma zamanının **Session** verisini nasıl işlediğini düşünelim. Sıkıştırma modu kapalı iken, **binary** formatta serileştirme işlemi yapılıp verinin içeriye atılıyor olması söz konusu olabilir. Diğer yandan sıkıştırma modu açık olduğunda, serileşen içeriğin olduğu gibi yazılmadan önce belki de bir **GZipStream** işleminden geçirilmesi ve sıkıştırılmaya çalışılması söz konusu olabilir.

Bu varsayım altında şu şekilde ilerliyor olacağız. **DataSet** nesnesinin kendisini standart bir **binary** serileştirme işlemine tutacağız. Sonrasında ise bu içeriği **GZipStream** ile sıkıştırmayı deneyip içeriklerin boyutlarına bakacağız. İspat için bu tekniği kullanıyor olacağız. Haydi parmakları sıvayalım. İşte örnek kod parçamız.

```
using (SqlConnection conn = new SqlConnection("data
source=.;database=AdventureWorks;integrated security=SSPI"))
{
   using (SqlDataAdapter adapter = new SqlDataAdapter("select * from
Production.ProductPhoto", conn))
   {
      DataSet ds = new DataSet("ProductPhoto");
      adapter.Fill(ds);

      BinaryFormatter formatter = new BinaryFormatter();
      FileStream fs= new
FileStream("c:\\DataSet.bin",FileMode.OpenOrCreate,FileAccess.Write);
```

```
        formatter.Serialize(fs, ds);
        fs.Close();

        FileStream fsZip= new
FileStream("c:\\DataSetZip.bin",FileMode.OpenOrCreate,FileAccess.Write);
        GZipStream gStream = new GZipStream(fsZip, CompressionMode.Compress);
        byte[] dataSetBytes=File.ReadAllBytes("C:\\DataSet.bin");
        gStream.Write(dataSetBytes, 0,dataSetBytes.Length);
        gStream.Close();
        fsZip.Close();
    }
}
```

Kod parçasından görüldüğü üzere ilk olarak Binary formatta bir **DataSet** içeriğini serlişetirmekteyiz. Bu
zaten **Session** tarafında **SessionItemLong** veya **SessionItemShort** alanına atılan serileştirilebilir içeriktir. Kodun ilerleyen kısımlarında ise serileştirilen
içeriği **GZipStream** sınıfından yararlanarak sıkıştırmaya çalışıyoruz. Bunun sonucu olarak üretilen dosya içeriklerinin boyutlarına baktığımızda ise aşağıdaki ekran görüntüsünde yer alan sonuçları elde ederiz.

Dikkat edileceği üzere serileştirilmiş içerik ile serileştirilmiş içeriğin sıkıştırılmış versiyonları arasında boyut olarak pek bir fark yoktur. Bu da **DataSet** tipinin ve özellikle**ProductionPhoto** içerisindeki **binary** alanlarının iyi bir şekilde sıkıştırılamıyor olmalarından kaynaklanmaktadır. Son geliştirdiğimiz örnek tamamen ve
tamamen **GZipStream** ile sıkıştırma tekniğinin her zaman işe yaramayacağını ve veri boyutunda daima önemli ölçüde bir değişikliğe neden olmayacağını göstermek üzere ele alınmıştır bunu unutmayalım. Sanıyorum ki iyi kullanıldığı takdirde **Session** sıkıştırması özellikle çok fazla oturumun açıldığı web uygulamalarında, **SQL Server** veya **State Server** mod kullanılması halinde önemli yer kazancı sağlayacak şekilde fayda getirmektedir. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

OldStyleSession.rar (137,04 kb)

.Net Framework 4.0 System.IO.File Tarafındaki Yenilikler (2010-10-26T09:45:00)

base class library,.net framework 4.0,file,system.io,

 Merhaba Arkadaşlar,

Bu yazımızda ele alacağımız konu ile ilişkili olarak kullanacağım giriş resmi için uzun bir süre araştırma yapmak zorunda kaldım. Sanırım yazıyı yazdığım bu sıcak yaz gününde devrelerim istediğim randımanı vermedi. Ancak en azından yandaki resim, anlatacağım ilk konu ile doğrudan alakalı olarak düşünülebilir. Bu resimde üst üste binmiş onlarca metre yüksekliğe varan dosya dolapları olduğu ifade edilemekte. Bunların hepsinin tek bir dosya içerisinde birleştirildiğini düşünün. üstelik bu dosya **text tabanlı** olsun.

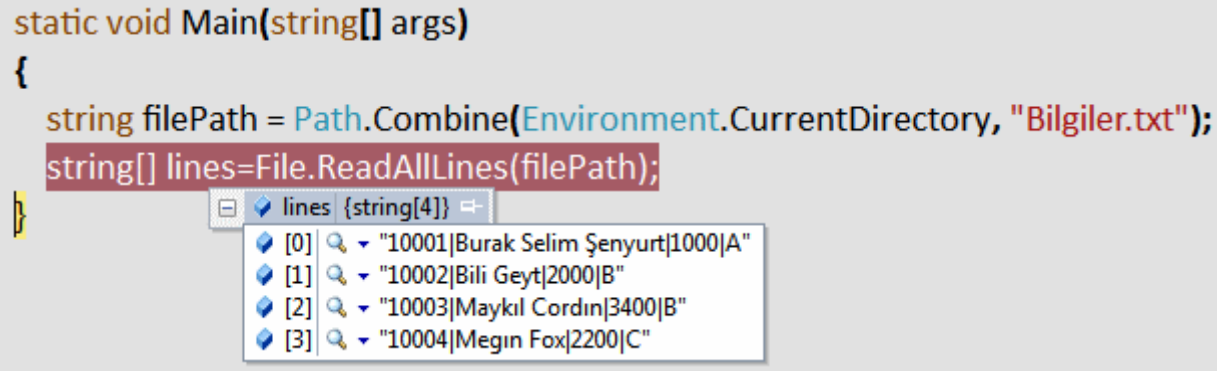
Bu tip bir dosyanın bir bankadan gönderilme olasılığı aslında çok yüksektir. Geçmiş deneyimlerimi düşündüğümde son derece olağan bir durum. örneğin bundan önce çalıştığım ve outsource olarak görev yaptığım bankada, boyutları 600 mb' ın üzerinde olan text tabanlı dosyalar sistemler arasında dolaşıp durmaktaydı. Hatta bu dosyalardan bazıları **SSIS** paketlerine sokularak **veritabanı** ortamına aktarılmaktaydı.
Hatta **SSIS** uzmanı olmamama rağmen Proje Yöneticisi tarafından bir dönem bana da böyle bir iş verildiğini ifade etmek isterim 😕

Tabi bu tip dosyaların en büyük özelliği de veri taşımak amacıyla satır ve sütun kavramlarını kullanmalarıdır. Ayrıca verinin çok yalın bir formatta taşınması ve her tür platform tarafından kolayca ele alınması, hatta çıplak gözle*(Tabi zaman zaman. Nitekim benim üzerinde çalıştığım SSIS paketlerinin sütunlarının sayısı 250' ye varmaktaydı* 😬 *)* rahatlıkla okunabilmesi gibi avantajlar da söz konusudur. Buna göre aşağıdaki gibi bir text içeriği doğru bir veri saklama biçimi olarak düşünülebilir.

Bilgiler.txt | Program.cs

```
1  10001|Burak Selim Şenyurt|1000|A
2  10002|Bili Geyt|2000|B
3  10003|Maykıl Cordın|3400|B
4  10004|Megın Fox|2200|C
```

Burada birbirleriyle | işaretleri şeklinde ayrılmış sütunlar söz konusudur. Toplam 4 satırdan oluşan veri içeriğini programatik ortamda da okumak son derece kolaydır. Bunun için, kolaya kaçan geliştiriciler **.Net Framework 2.0** ile birlikte **File** sınıfına eklenmiş **static ReadAllLines** metodunu kullanır. Aşağıdaki kod parçasında görüldüğü gibi.



ReadAllLines metodu parametre olarak dosya adresini almakta ve içeriğinde tüm satırları **string** tipinden bir diziye aktarmaktadır. **Debug** çıktısında, string dizisine aktarılan içerik net bir şekilde görülebilir. Tabi bu adımdan sonra elde edilen **string[]** dizisi üzerinde dolaşılıp | işaretlerine göre ayrıştırma yapılarak sütunlara da kolayca erişilebilir. Ancak önemli bir sorun da vardır?

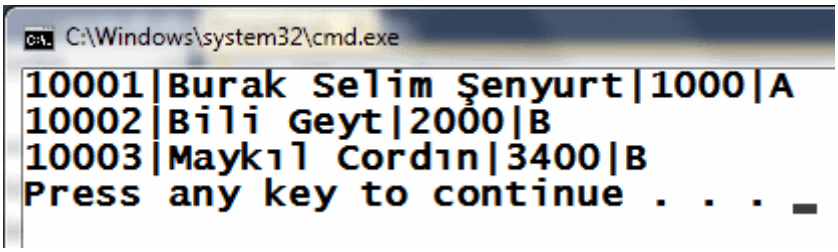
Boyutu çok yüksek olan bir dosyanın ReadAllLines metodu yardımıyla okunmasının sakıncıları var mıdır?

Aslında en büyük sakınca **ReadAllLines** metodu, dosyanın tüm satırlarını **string[]** diziye aktarana kadar, içinde çalıştığı **Thread'** i duraksatmaktadır. Bir başka deyişle**ReadAllLines** metodu ile okunan satırların tamamı **string** dizisine aktarılmadan kod bir sonraki satıra inmeyecektir. İşte **Base Class Library->IO** bünyesinde getirilen yeni **static** metodlardan birisi bu vakayı çözmek amacıyla getirilmiştir. **File.ReadLines** metodu.

ReadLines static metodunun **aşırı yüklenmiş(Overload)** iki versiyonu vardır. Her ikiside satırları okunacak dosyayı parametre olarak almaktadır. Metodlar geriye **IEnumerable<string>Arayüzü(Interface)** tarafından taşınabilecek bir referans döndürmektedir. **ReadLines** metodu aslında dosyanın satırları arasında gezinmeyi sağlayacak bir numaralandırıcıyı hazırlamaktadır. Bu sebepten kod satırı anında alt satıra inebilir. üstelik tüm satırların yüklenmesi de söz konusu değildir. örneğin çok büyük bir **text** dosyasının sadece ilk 3 satırını almak istediğimizde aşağıdaki gibi bir kod parçası işe yarayacaktır.

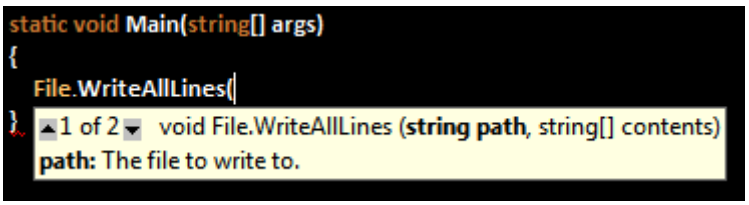
```
using System;
using System.IO;
using System.Collections.Generic;

namespace NewIOFeatures
{
   class Program
   {
      static void Main(string[] args)
      {
```

```
            string filePath = Path.Combine(Environment.CurrentDirectory, "Bilgiler.txt");
            //string[] lines=File.ReadAllLines(filePath);
            IEnumerable<string> lines=File.ReadLines(filePath);
            int i = 0;
            foreach (string line in lines)
            {
               if (i < 3)
               {
                  Console.WriteLine(line);
                  i++;
               }
               else
                  break;
            }
         }
      }
}
```



Bu kod parçasında görüldüğü üzere **foreach** döngüsü ile **Bilgiler.txt** dosyasının satırları üzerinde hareket edilmeye başlanmaktadır. **Integer** tipinden olan **i** değişkeninden yararlanılarak bir sayaç oluşturulmuş ve ilk 3 satır okunduktan sonra **break keyword**' ü yardımıyla döngüden çıkılması sağlanmıştır. Tabi şunu hatırlatmamızda yarar vardır. **foreach** döngüsü yardımıyla dosya içerisindeki tüm satırların dolaşılması **ReadAllLines** metodunun yaptığı işin aynısıdır. Dolayısıyla duruma göre uygun olan metodu kullanmakta yarar vardır. **System.IO** tarafında **BCL** takımı tarafından getirilen yeniliklerden birisini bu şekilde özetlemiş olduk. Gelelim diğer bir yeniliğe.

File tipinin dosya okuma ve yazma işlemlerinde sağladığı kolaylık bilinmektedir. Ancak yine de bazı iyileştirmeler gerekmiştir. Söz gelimi **WriteAllLines** metodunun **.Net 4.0** öncesindeki versiyonunu ele alalım.



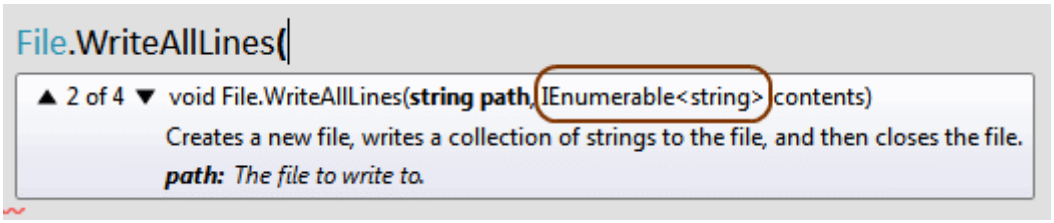
WriteAllLines metodu bir önceki sürümde iki aşırı yüklenmiş versiyona sahiptir. Bu metod belirtilen dosyaya, satır bazlı string bir içeriğin eklenmesi amacıyla kullanılmaktadır. çok güzel. Ama eksik. Nitekim versiyonda dikkati çeken nokta, sadece string tipinden bir dizinin kullanılabiliyor olmasıdır. Peki bunun ne gibi bir sakıncası olabilir?

string[] tipinden olmayan, örneğin List<string> gibi bir koleksiyon içeriğinin WriteAllLines metodu yardımıyla dosyaya yazılabilmesi için, önce string[] dizisine çevrilmesi gerekmektedir.

Visual Studio 2008 üzerinde yazılmış örnek bir kod parçası ile bu durumu görelim.

```
using System;
using System.Collections.Generic;
using System.IO;

namespace IO
{
   class Program
   {
      static void Main(string[] args)
      {
         string filePath=Path.Combine(Environment.CurrentDirectory,"Bilgiler.txt");
         List<string> content=new List<string>{
            "10001|Burak Selim Şenyurt|1000|A"
            ,"10002|Bili Geyt|2000|B"
            ,"10003|Maykıl Cordın|3400|B"
            ,"10004|Megın Fox|2200|C"
         };

         File.WriteAllLines(filePath, content.ToArray());
      }
   }
}
```

Burada **List<string>** tipinden olan koleksiyonun içeriğinin **Bilgiler.txt** dosyasına yazıldığını görmektesiniz. Ancak **WriteAllLines** metodunun içeriğine dikkat edilecek olursa, **content** isimli değişkenin **ToArray** metodu yardımıyla **string[]** dizisine dönüştürüldüğü görülmektedir. İşte **BCL** takımı **.Net Framework 4.0**'
da **WriteAllLines** metodunun yeni bir **aşırı yüklenmiş(Overload)** versiyonunu yazarak bu durumu düzeltmiştir.



Buna göre daha önceki versiyonda yapmak zorunda kaldığımız **ToArray** dönüştürmesine gerek yoktur. Son olarak yazdığımız kodun **Visual Studio 2010** üzerinde geliştirilen yeni hali aşağıdaki gibidir.

```
string filePath = Path.Combine(Environment.CurrentDirectory, "Bilgiler.txt");
        List<string> content = new List<string>{
           "10001|Burak Selim Şenyurt|1000|A"
           ,"10002|Bili Geyt|2000|B"
           ,"10003|Maykıl Cordın|3400|B"
           ,"10004|Megın Fox|2200|C"
        };

        File.WriteAllLines(filePath, content);
```

Bu yenilik sayesinde **WriteAllLines** metodunun **string Array** dışındaki **IEnumerable<string>** arayüzünü uygulayan tipler ile de çalışabilmesi sağlanmıştır. Gelelim bu yazımızda değineceğimiz son yeniliğe.

WriteAllLines metodu **content** değişkeni ile aldığı içeriği ilgili dosyaya yazarken dosyanın sürekli yeniden oluşturulmasına neden olmaktadır. Oysaki dosya sonuna ilave yapmamız da gerekebilir. **.Net Framework 4.0** öncesindeki sürümde **AppendAllLines** isimli bir metod bulunmamaktadır. Ancak **.Net Framework 4.0** ile birlikte bu metod **File** tipine eklenmiştir ve aşağıdaki örnek kod parçasında olduğu gibi kullanılabilir.

```
using System;
using System.IO;
using System.Collections.Generic;

namespace NewIOFeatures
{
   class Program
   {
      static void Main(string[] args)
      {
         string filePath = Path.Combine(Environment.CurrentDirectory, "Bilgiler.txt");
         List<string> content = new List<string>{
            "10001|Burak Selim Şenyurt|1000|A"
            ,"10002|Bili Geyt|2000|B"
```

```
            ,"10003|Maykıl Cordın|3400|B"
            ,"10004|Megın Fox|2200|C"
        };

        File.WriteAllLines(filePath, content);

        List<string> newLines = new List<string>
        {
            "10006|Komiser Kolomb|2345|Z"
        };
        File.AppendAllLines(filePath, newLines);
    }
  }
}
```

AppendAllLines metodu **IEnumerable<string>** tipi ile çalışmakta ve bu tipin çalışma zamanındaki içeriğini parametre olarak verilen dosyanın sonuna eklemektedir.

Tabi **BCL** takımı tarafından getirilen daha pek çok yenilik daha söz konusudur. örneğin bunlardan bazılarını hemen araştırmaya başlayabilirsiniz.

- Directory.EnumerateFiles
- Directory.EnumerateDirectories
- Directory.EnumerateFileSystemEntries

vb...

Yeri geldikçe bunları toplu olarak ele almaya çalışıyor olacağız. Bu yazımızda **File** tipi için öne çıkan bazı yenilikleri gördük. Buna göre;

- **ReadLines** metodu yardımıyla satır bazlı bir **text** içeriğinin bir seferde değil ama bir iterasyon yardımıyla okunabileceğini gördük.
- **WriteAllLines** metodunun sadece **string Array** tipi ile çalışan versiyonu yerine **IEnumerable<string>** ile çalışan bir versiyonunun geliştirildiğini gördük. Bu sayede bir önceki sürümde söz konusu olan **ToArray** metodu ile diziye dönüştürme zorunluluğunun ortadan kalktığına şahit olduk.
- Son olarak dosya sonuna **IEnumerable<string>** ile işaret edilen bir içeriğin eklenebilmesini sağlayan **AppendAllLines** metoduna değindik.

Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

NewIOFeatures.rar (21,53 kb) **[örnek Visual Studio 2010 Ultimate sürümünde geliştirilmiş ve test edilmiştir]**

BTAkademi Seçilecek Çırak için Destek Oluyor (2010-10-25T07:41:00)

Merhaba Arkadaşlar,

Hatırlayacağınız üzere bir süre önce paylaşım alanında benden bayrağı alarak çok daha ilerilere götürecek bir çırak adayı için bir çalışma başlattığımı duyurmuştum. Söz konusu program hakkındaki detaylara Sizden Daha İyi Daha Faydalı Olacağım (çırak Aranıyor) başlıklı blog yazısından da ulaşabilirsiniz.

Blog yazısını hazırladığım tarih itibariyle **65 başvuru** olduğunu ve şu anda **3ncü soru** evresinde bulunulduğunu belirtmek isterim. Kolay olan ilk soruyu takiben eden ikinci ve üçüncü sorular adayları bir hayli terletti/terletmekte. Adaylarımızı bekleyen **37 soru** daha var. Bu soruları yarışmadan sonra yayınlayıp ne amaçla kullanıldıklarını da sizlerle paylaşıyor olacağım

btakademi Tabi bu zorlu süreç sonunda seçliecek olan çırağı bekleyen güzel bir ödül olduğunu da artık rahatlıkla ifade edebilirim. çırak programını planladıktan kısa bir süre sonra, uzun süre editörlüğünü yaptığım **CSharpNedir?com'** un ve **BTAkademi** firmasının kurucusu olan **Sefer Algan**' a**(MVP)** konuyu dile getirdim. Kendisi bu programa katkıda bulunmaktan büyük memnuniyet duyacağını dile getirdi. Buna göre de kazanan çırak, **BTAkademi**' **de seçeceği bir eğitime ücretsiz olarak katılma hakkını elde edecek**. öncelikle bu güzel katma değeri için sevgili meslektaşım **Sefer Algan**' a teşekkür etmek istiyorum.

çırak programı ile ilişkili olarak kazanamayan arkadaşlarımız için de bir ödül olacağını*(en azından planladığımı)* şimdiden belirtmek isterim. Bu konuda da bazı girişimlerde bulunuyorum. Eğer bir aksilik olmassa gerçekleşiyor olacak. Katılan ve programın sonuna kadar kalacak olan tüm çırak adaylarımıza tekrardan başarılar dilerim. Zorlu bir maratona büyük bir cesaret ile atılmış olmaları benim için çok önemliydi.

Bunlarla birlikte çırak programı ile ilişkili olaraktan kafalara takılan sorulardan birisi de, **"kazanamayan adayların ne kazanacağı"** idi Aslında program içerisinde yer alan **40** soru da, adayları şu an bulundukları noktadan daha ileri bir noktaya götürüyor olacak. Aday' lar bu sorulara cevap verirken veya hazırlanırken, ne gibi güçlükler ile karşılaşacaklarını, disiplinli olarak çalışmanın ne kadar zor olabileceğini, nelerden fedakarlık edilmesi gerektiğini vb… faydaları çıkartıyor olacaklar. Bu nedenle kazanamayacak adaylarımızın da söz konusu sürece 4 elle sarılmalarında büyük önem olduğunu bir kez daha hatırlatmak istiyorum. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

NedirTv?com Söyleşileri-.Net Framework 4.0 ile Gelen Yenilikler Bölüm 3 (2010-10-21T20:47:00)

nedirtv?com söyleşileri,.net framework 4.0,entity framework,windows presentation foundation,wpf 4.0,ef 4.0,

Merhaba Arkadaşlar,

Malumunuz bir süredir NedirTv?com söyleşilerine ara vermek zorunda kaldık. Bu gecikmelerde iş yoğunluğumuzun ve ufak tefek rahatsızlıkların*(ağırlıklı olarak gribal enfeksiyonlar)* rol aldığını ifade edebilirim.

Bundan önceki son bölümümüzde **.Net Framework 4.0** ile birlikte gelen yenilikleri incelemeye devam etmiş ancak bazı konuları değerlendirememiştik. İşte **3ncü** bölümde bu konulardan **Entity Framework 4.0** ile **Windows Presentation Foundation 4.0** yapılarını tartışmaya çalıştık.

Aslında söyleşiye başlarken **.Net Framework 4.0** ile birlikte gelen yenilikleri 3ncü bölümümüz ile sonlandırmayı planlamıştık ama tahmin edeceğiniz üzere konu konuyu açtı ve konuşmak istediğimiz **Windows Communication Foundation 4.0, Workflow Foundation 4.0, Visual Studio 2010 ALM** gibi mevzular **4ncü** bölüme kaldı

Yine de **.Net Framework 4.0** ile birlikte gelen yenilikleri **4** bölümde ele almanın daha iyi olacağını düşünüp mutlu olduk. Umarım **3ncü** bölüm ile de sizleri bilgilendirebilmişizdir. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Söyleşiyi dinlemek için tıklayın

Generic Lazy Tipi Olmasaydı (2010-10-18T11:05:00)

c# 4.0,bcl,base class library,lazy initialization,lazy,singleton design pattern,lock,thread safe,

 Merhaba Arkadaşlar,

Aklım üniversite kampüsünün çimlerinde hala...Pek çok gün İstinye Park' a giden öğle arası servisimiz, dönüş yolunda İTü kütüphanesi önünden geçmekte*(Geçmekte idi...Sonrasında o yol trafiğe kapatıldı)* öğrencilerin çimlere yayılarak mavi gök yüzünü seyre dalmasına her zaman imreniyorum. çoğu zaman bu psikolojideki öğrencinin kafasında oluşan sorunlar bellidir. Kız arkadaş veya erkek arkadaş sorunu, maddi sorunlar, dersler, vize ve finaller...Tabi iş hayatına giren ben gibi insanlar aynı çimlere yatmaya kalksa, kafada dönen sorunların sayısı azalacağına büyük ihtimalle artacaktır. Hele ki yazılımla uğraşıyorsanız mavi göğü seyre dalarken geçen bulutların çoğu birer **Component** haline dönüşecek ve üstünüze üstünüze gelecektir. Neyse...

Hatırlayacağınız üzere **Tembellik Etmek İstiyorum (Generic Lazy Tipi ile Et)** başlıklı yazımızda **.Net Framework 4.0** ile birlikte **Base Class Library**' ye eklenen yeni tiplerden birisi olan **Lazy<T>** sınıfını irdelemeye çalışmıştık. Bu sınıfın **T** tipi için gerçek anlamda**Lazy Initialization** yaptığını örnekler üzerinden öğrendik. Hatırlayacağınız üzere **Lazy<T>** nesne örnekleri üzerinden **Value** özelliği çağırılmadığı sürece **T**tipinden değer döndüren bir operasyonun çağırılması söz konusu değildir. üstelik **Value** özelliğine sonradan yapılan çağrılarda **T** dönüşü yapan operasyonların tekrardan çağırılmadığı da bilinmektedir. Şimdi Lazy<T> tipini kullanmadan bir sınıf özelliğinin Lazy Initializate işleminde değerlendirilip değerlendirilemeyeceğini düşünerek yola koyulalım. Bu amaçla aşağıdaki gibi bir kod örneğini göz önüne alabiliriz.

```
using System.Collections.Generic;

namespace LazyPart2
{
   class Program
   {
      static void Main(string[] args)
      {
         var productList=ProductCreater.Products;
         var productListAgain=ProductCreater.Products; // Bu çağrı sonrasında
ProductCreator içerisindeki products alanının yeniden örneklenmediği görülecektir
      }
   }
```

```
   class ProductCreater
   {
      private static readonly List<Product> products = new List<Product>
      {
         new Product{ProductId=1,Name="Product X",ListPrice=1},
         new Product{ProductId=2,Name="Product Y",ListPrice=3},
      };

      private ProductCreater()
      {
      }

      public static List<Product> Products
      {
         get
         {
            return products;
         }
      }
   }

   class Product
   {
      public int ProductId { get; set; }
      public string Name { get; set; }
      public double ListPrice { get; set; }
   }
}
```

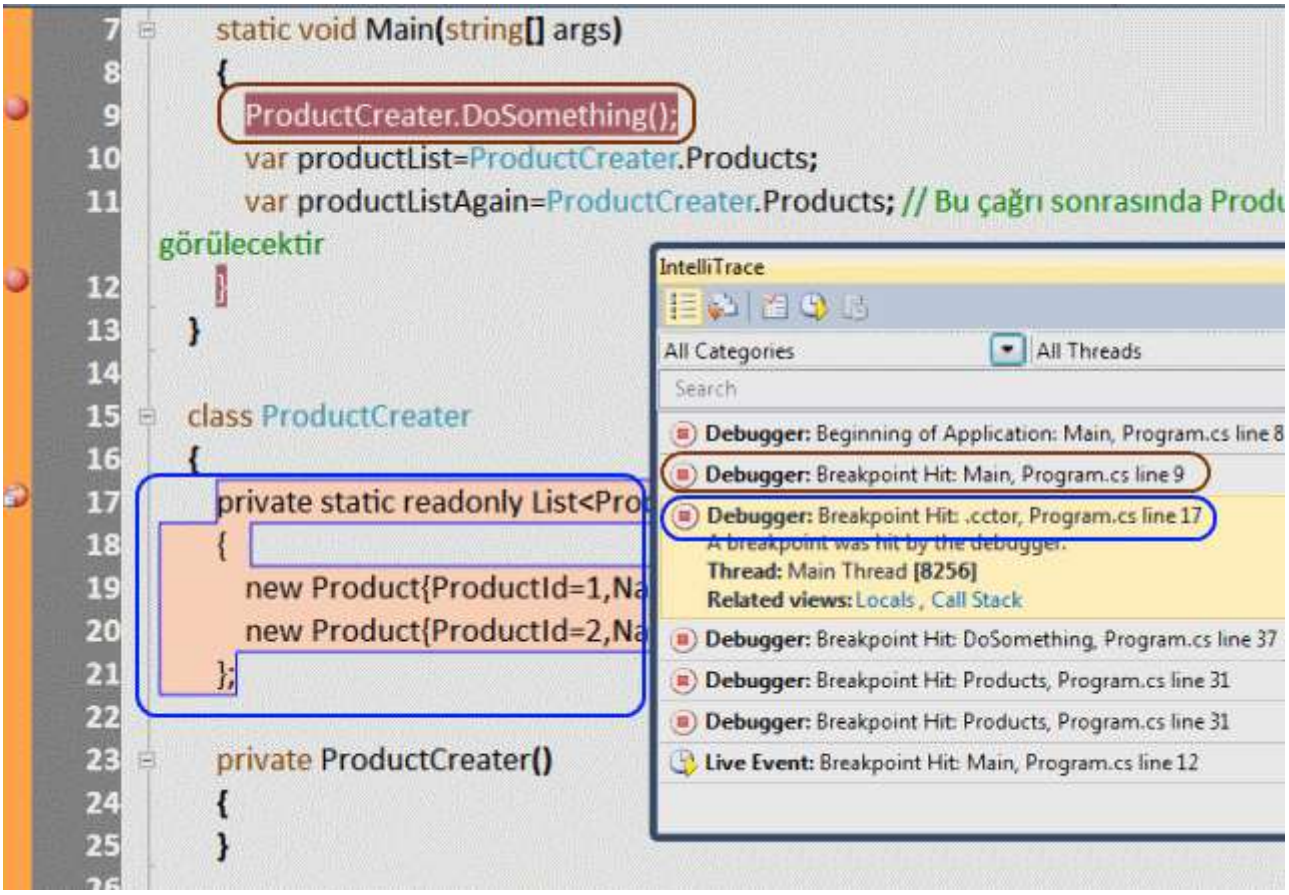
ProductCreator isimli sınıf **new** operatörü ile örneklenemeyecek şekilde tasarlanmıştır. **products** isimli **field** değişkeni **static** olarak tanımlanmıştır ve ilk değerleri atanmıştır.**ProductCreator** tipi üzerinden kullanılmak istenen ürün listesini alabilmek için **static Products** özelliğinin kullanılması yeterlidir. Bu kod **debug** edildiğinde **Products** özelliğine yapılan çağrılardan ilkinde, **products** isimli **field'** ın oluşturulduğu görülecektir. Yani ilk değerler yüklenecektir. İkinci çağrıda ise yeniden bir örnekleme söz konusu olmayacaktır. Bu da aslında tek bir **products** alanının olmasının garantilendiğini gösterir. Hatta şu anki tasarıma göre **ProductCreator** üzerinden çağırılabilen tek **public** üye **Products** özelliği olduğundan ürün listesinin kullanılmak istendiği yerde **initialize** olacağı sonucuna varabiliriz. Ki bu büyük bir aldatmaca olacaktır. Ne demek istediğimizi daha iyi anlamak için kod içeriğini aşağıdaki gibi değiştirdiğimizi düşünelim.

```
using System.Collections.Generic;
```

```
namespace LazyPart2
{
   class Program
   {
      static void Main(string[] args)
      {
         ProductCreater.DoSomething();
         var productList=ProductCreater.Products;
         var productListAgain=ProductCreater.Products; // Bu çağrı sonrasında
ProductCreator içerisindeki products alanının yeniden örneklenmediği görülecektir
      }
   }

   class ProductCreater
   {
      private static readonly List<Product> products = new List<Product>
      {
         new Product{ProductId=1,Name="Product X",ListPrice=1},
         new Product{ProductId=2,Name="Product Y",ListPrice=3},
      };

      private ProductCreater()
      {
      }

      public static List<Product> Products
      {
         get
         {
            return products;
         }
      }

      public static void DoSomething()
      {
         System.Console.WriteLine("Do Something");
      }
   }

   class Product
   {
      public int ProductId { get; set; }
      public string Name { get; set; }
      public double ListPrice { get; set; }
```

```
    }
}
```

Dikkat edileceği üzere **ProductCreator** sınıfı içerisine, geriye bir şey döndürmeyen ve aslında anlamlı bir işte yapmayan **static DoSomething** metodu eklenmiştir. Bu metod modelin çökmesi için yeterlidir. çünkü **ProductCreator.Products** için yapılan ilk çağrıdan önce
gerçekleştirilen **ProductCreator.DoSomething** operasyonu, **products** alanının **initialize**ed ilmesine neden olmaktadır. Bu da **products** alanının kullanılmak istendiği
zaman **initialize** olacağı teorisini ilk tasarladığımız sınıf yapısı düşünüldüğünde çökertmektedir. örnek **debug** edildiğinde bu durumda daha net bir şekilde görülmektedir.



İşleyiş sırasına baktığımızda **DoSomething** çağrısını takiben **17nci** satıra gelindiği görülmektedir ki burada da **product** isimli alanının doldurulması işlemleri söz konusudur. Aslında MSDN araştırması yapıldığında **IL** tarafında **ProductCreator** sınıfı
için **beforefieldinit** isimli tanımlamanın yapılmasının bu **initialize** işlemine neden olduğu ifade edilmektedir. Gerçekten öyle midir acaba? Bu tanımlamayı bir şekilde kaldırmayı başarırsak static products alanı sadece çağırıldığı zaman mı doldurulacaktır. önce IL tarafındaki tanımlamaya bir bakalım.

```
.class private auto ansi beforefieldinit ProductCreater
   extends [mscorlib]System.Object
{
```

```
  .method private hidebysig specialname rtspecialname static void .cctor() cil managed
  {
  }
  .method private hidebysig specialname rtspecialname instance void .ctor() cil managed
  {
  }
  .method public hidebysig static void DoSomething() cil managed
  {
  }
  .property class class [mscorlib]System.Collections.Generic.List`1<class
LazyPart2.Product> Products
  {
    .get class [mscorlib]System.Collections.Generic.List`1<class LazyPart2.Product>
LazyPart2.ProductCreater::get_Products()
  }
  .field private static initonly class [mscorlib]System.Collections.Generic.List`1<class
LazyPart2.Product> products
}
```

Bu noktada bir de **static constructor** kullanımı düşünülebilir. Nitekim **static yapıcı metod** kullanımı **beforefieldinit** tanımlamasını kaldıracaktır. Bu durumu ele almak için öncelikle **ProructCreator** sınıfını aşağıdaki hale getirelim. Hatta sınıfıda **static** olarak tasarladığımızdan, **private** erişim belirleyicisine sahip **Constructor'** unda kaldırılması gerekmektedir.

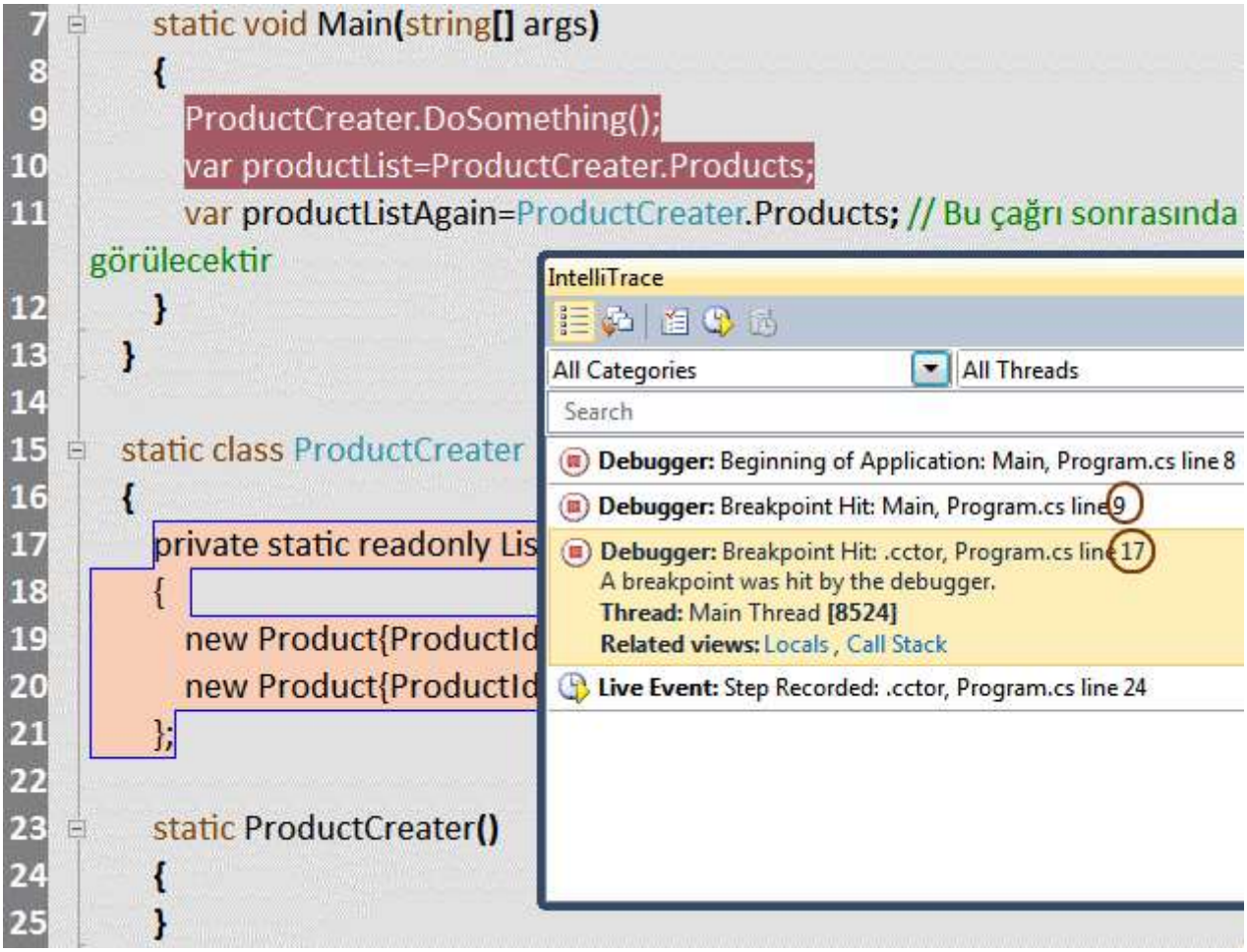
```
 static class ProductCreater
{
  private static readonly List<Product> products = new List<Product>
  {
    new Product{ProductId=1,Name="Product X",ListPrice=1},
    new Product{ProductId=2,Name="Product Y",ListPrice=3},
  };

  static ProductCreater()
  {
  }

  public static List<Product> Products
  {
    get
    {
      return products;
    }
  }
```

```
    public static void DoSomething()
    {
        System.Console.WriteLine("Do Something");
    }
}
```

Bu durumda **IL** çıktısı aşağıdaki gibi olacaktır.

```
.class private abstract auto ansi sealed ProductCreater
    extends [mscorlib]System.Object
{
    .method private hidebysig specialname rtspecialname static void .cctor() cil managed
    {
    }
    .method public hidebysig static void DoSomething() cil managed
    {
    }
    .property class class [mscorlib]System.Collections.Generic.List`1<class
LazyPart2.Product> Products
    {
        .get class [mscorlib]System.Collections.Generic.List`1<class LazyPart2.Product>
LazyPart2.ProductCreater::get_Products()
    }
    .field private static initonly class [mscorlib]System.Collections.Generic.List`1<class
LazyPart2.Product> products
}
```

Dikkat edileceği üzere **beforefieldinit** takısı görülmemektedir. İşte şimdi oldu diyerek rahat rahat koltuklarımıza yaslanabileceğimizi düşünebiliriz. Ama ne yazık ki bir anda irkilmemiz sadece an meselesidir. İşte **Debug** zamanındaki çalışma sırası.



O da ne? **DoSomething** sonrası yine **product** alanının **initialize** edildiği görülmektedir. 😟 Tam bir hüsran diyebilir miyiz? Aslında olayı iyi tarafından ele alabiliriz. Şöyleki; sınıf içerisinde yer alan **static** alanlar, sınıfın başka bir static üyesi(metod, özellik) çağırıldığı takdirde otomatik olarak zaten **initialize** edilmektedir. Demek ki kalıbımızın uygulanış biçimi çok doğru değildir. üstelik **Thread Safe** bir yaklaşım içermediği de gün gibi ortadadır. O halde **ProductCreator** tipimizi aşağıdaki gibi değiştirerek ilerlemeye devam edelim.

```
using System.Collections.Generic;

namespace LazyPart2
{
   class Program
   {
      static void Main(string[] args)
      {
         ProductCreater.DoSomething();
         var productList = ProductCreater.Products;
         var productListAgain = ProductCreater.Products; // Bu çağrı sonrasında
ProductCreator içerisindeki products alanının yeniden örneklenmediği görülecektir
```

```
    }
}

static class ProductCreater
{
    private static List<Product> products = null;
    private static readonly object lockObject = new object();

    static ProductCreater()
    {
    }

    public static List<Product> Products
    {
        get
        {
            if (products == null)
                lock (lockObject)
                {
                    if (products == null)
                    {
                        products = new List<Product>
                        {
                            new Product{ProductId=1,Name="Product X",ListPrice=1},
                            new Product{ProductId=2,Name="Product Y",ListPrice=3},
                        };
                    }
                }

            return products;
        }
    }

    public static void DoSomething()
    {
        System.Console.WriteLine("Do Something");
    }
}

class Product
{
    public int ProductId { get; set; }
    public string Name { get; set; }
    public double ListPrice { get; set; }
```

```
    }
}
```

Bu sefer **ThreadSafe** olan*(Double Check yapıldığına dikkat edelim)* ve gerçekten **Products** özelliği çağırıldığında **initialize** işlemini gerçekleştiren bir tip elde etmiş olduk. Her ne kadar **DoSomething** metodunu çağrısı sonrası çalışma zamanı **products** ve **lockObject** alanlarını oluşturacak olsa da, bizim için önemli olan ürün listesinin **Lazy** olarak başlatılmasıdır. Sanıyorum ki biraz daha elle tutulur bir sonuca vardık. Hatta bu kod parçasına göre en basit **tasarım kalıbının(Design Pattern)** uygulandığı ip ucunu verebiliriz. Ama daha iyisi olabilir mi? Söz gelimi içinde hiç bir iş yapılmayan bir **static** metodumuz var. Bu olmasa. Sınıfımızda static tanımlanmak zorunda değil aslında. Aslında işi yine **.Net** tarafına yıkabiliriz. Aslında ben **Lazy<T>** tipini kullanmak istiyorum diye haykırabiliriz 😆 İşte haykırıyorum. Aşağıdaki kodu göz önüne alalım.

```
using System.Collections.Generic;
using System;

namespace LazyPart2
{
    class Program
    {
        static void Main(string[] args)
        {
            ProductCreater.DoSomething();
            var productList = ProductCreater.Products;
            var productListAgain = ProductCreater.Products; // Bu çağrı sonrasında
ProductCreator içerisindeki products alanının yeniden örneklenmediği görülecektir
        }
    }

    class ProductCreater
    {
        private static readonly Lazy<List<Product>> products = new
Lazy<List<Product>>(
            () =>
            {
                return new List<Product>
                        {
                            new Product{ProductId=1,Name="Product X",ListPrice=1},
                            new Product{ProductId=2,Name="Product Y",ListPrice=3},
                        };
            }
        );
```

```
        public static List<Product> Products
        {
            get
            {
                return products.Value;
            }
        }

        public static void DoSomething()
        {
            System.Console.WriteLine("Do Something");
        }
    }

    class Product
    {
        public int ProductId { get; set; }
        public string Name { get; set; }
        public double ListPrice { get; set; }
    }
}
```

Bu sefer products isimli değişken **Lazy<List<Product>>** tipinden tanımlanmıştır. Diğer yandan **Products** özelliği **products** isimli alanın **Value** özelliğini döndürmektedir. Buna göre çalışma zamanındaki kod icra sırasına bakıldığında **DoSomething** çağrısı sonucu **Product** listesinin oluşturulduğu kod bloğuna girilmediği görülecektir. Hatta **Lazy<T>**tipinin **Lazy Initialize** işlemini gerçekleştirdiği safha da **Thread Safe** bir ortam sağladığını da biliyoruz. Bu nedenle bir önceki yapı da uyguladığımız **lock kilit mekanizmalı double check** içeren kod parçasına da gerek kalmamıştır.

Vuuvvv!!! Baya bir karışık oldu yauv. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

LazyPart2.rar (21,80 kb) **[örnek Visual Studio 2010 Ultimate üzerinde geliştirilmiş ve test edilmiştir]**

Türkçe Microsoft Forumları Hazır (2010-10-18T05:49:00)

microsoft,microsoft technet,msdn,microsoft answers,türkçe,

Merhaba Arkadaşlar,

Microsoft cephesinde 1 sene öncesinden başlayan ve **Newsgroup'** lar yerine **Forum** mantığını devreye alan süreç tamamlanmış bulunuyor. Bu anlamda özellikle **Microsoft Technet, MSDN** ve**Microsoft Answers Forum**'

larının **Türkçe** versiyonlarının da açıldığını bildirmek isterim. Bu **Türkçe** konuşan geliştiriciler için oldukça önemli bir fırsat.

Microsoft tarafından yönetilen forumlara çeşitli teknolojilerle ilgili sorularınızı sorabilir ve yetkili uzmanların ve **Microsoft MVP**'lerinin önceden verdiği cevaplara göz atabilirsiniz.

Kaynakların özellikle **Türkçe** olması gerçekten çok çok önemli bir gelişme.
Teşekkürler **Microsoft**

TechNet

MSDN

MS Answers

Sizden Daha İyi Daha Faydalı Olacağım (Çırak Aranıyor) (2010-10-14T18:30:00)

mentor,mvp,microsoft,

Merhaba Arkadaşlar,

CSharpnedir?com bünyesinde **2003** yılında başladığım, sonrasında kendi bloğum ve NedirTv?com üzerinden devam ettirdiğim editörlük yaşantımda **7nci yılımı** devirmek üzereyim. Bu zaman zarfı içerisinde .Net Framework' ün pek çok alanında sayısız**makale, webcast, screencast** hazırladım. Pek çok **seminere** katılıp siz değerli okurlarım ve meslektaşlarımla bilgi paylaşımında bulundum. Bildiğiniz üzere gönüllü olarak yaptığım bu paylaşımların sonucu olarak, **2006** yılından bu yana **Microsoft** tarafından**MVP** ödülü ile onurlandırılmaktayım.

Asında bu ünvanı almamdaki en büyük sebeplerden birisi makale üretimimdir. Takdir edersinizki makale yazmak, teknik olarak bilgi paylaşımında bulunmak, bunu iş yoğunlu ve karmaşık yaşam döngüsü içerisinde başarılı bir şekilde yapmak ve istikrarı sağlamak son derece zor bir zanaat. Eski bir eğitmen arkadaşımın sıkça söylediği güzel bir özlü söz vardır **"Türk gibi başla, İngiliz gibi devam et…"**

Yani bir işe hevesli, arzulu şekilde başlamak hiç bir zaman için yeterli değildir. İstikrarı korumak, düzenli ve disiplinli bir şekilde planlanan süre dahilinde işi yürütmekte çok önemlidir. Bu felsefeyi makale yazma konusunda da sıklıkla uygulamaya gayret ettim. Ne varki pilimin artık bittiğini hissedebiliyorum

Bu nedenle kendime bir çırak aramaktayım. çünkü öğrendiklerimi, tecrübelerimi, paylaşım anlamındaki bilgi birikimimi, bu işi benden daha iyi yapacak ve üstlenecek birisine aktarmak niyetindeyim.

çırağın görevi son derece basit…**"Bayrağı benden devralmak, gönüllü olarak, hiç bir karşılık beklemeden paylaşıma devam etmek"** Bunu yaparkende kendi tarzını göstermek, anlatım ekolünü oluşturmak, kaliteli içerik üretmek, hazırlanmak, yazmak ve daha pek çok konuda kendini geliştirmek zorunda. İşte ben bu noktada devreye girip kendisine akıl hocalığı(Mentorluk) yapmak arzusundayım.

Sevgili okurlarım arasında eğer gönüllü olanlar var ise veya **"Ben Burak Selim Şenyurt' un öğütleri ile daha iyi bir MVP olacağım, daha iyi paylaşımlar yapacağım, yapmak istiyorum"** diyip benim öğretmenliğimi sahiplenecekler var ise yapmaları gereken tek bir şey var…**"Gerçekten bunu yapacaklarına, bu ideale sahip çıkacaklarına beni inandırmak"**

Tabi bir **Matematik Mühendisi** olduğumdan ve hayatımın önemli bir bölümünü ispatlara adadığımdan, bu çırağın gerçekten beni ikna ediyor olması şart

Gelelim teknik meselelere. Bu çırağın seçim sürecinde dikkat edilecek hususlar nelerdir?

- Başvurularınızı**1 Ocak 2011** tarihine kadar yapabilirsiniz.
- Başvuru için selim@bsenyurt.com adresine mail atabilirsiniz. Sizinle en kısa sürede iletişime geçerim. Maillerinizde Subject kısmında **"çırak Adayıyım"** derseniz ayırt etmem de kolay olacaktır
- Sonuçlar **1 Şubat 2011** tarihinde belli olacak ve dolayısıyla çırağım bu tarih itibariyle göreve başlamış olacaktır.
- çırağa yapacağım **mentorluk süresi ise 1 senedir**. Buna göre çırağın **1 Şubat 2012** tarihinde mezun olmasını hedeflemekteyim.
- Tüm başvuruları gizli tutmayı tercih edeceğim.

Şunu unutmayın. Bu projedeki en büyük amaç; paylaşıma inanan, paylaşmayı seven ve bunu teknik olarak benden çok daha kaliteli ve düzgün bir şekilde yapmak isteyen bir çırağı **1 sene** içerisinde profesyonel yapmaktır. Adayların(başvuranların) bu felsefeye gerçekten inanıyor olmaları benim için çok önemlidir.

Tembellik Etmek İstiyorum (Generic Lazy Tipi ile Et) (2010-10-12T00:20:00)

c# 4.0,bcl,base class library,lazy initialization,lazy,

 Merhaba Arkadaşlar,

Yaz günlerinde pek çok geliştirici tembellik yapmak ister. Hatta benim gibi kocaman bir **üniversite Kampüsü** içerisinde yer alan çalışma ortamınız var ise ve kampüsünüzün çimleri üzerinde yatıp şöyle beş on dakika kestirmeye müsaitse. Tabi tembelliğin çeşitli türevleri vardır. çimler üzerinde uzanmak bunlardan sadece birisi.

Aslında geliştiriciler için **Lazy** olmanın başka manaları da vardır. Olay sadece çimler üzerinde uzanmaktan ibaret değildir anlayacağınız. İlk akla gelen **ORM(Object Relational Mapping)** araçlarının birincil özelliklerinden birisi olan **Lazy Loading** kavramıdır. Kısaca **Entity** tabanlı nesnelerin/koleksiyonlarının gerektiğinde yüklenmesi şeklinde açıklayabiliriz. Ne varki **.Net Framework 4.0** sürümü ile birlikte hayatımıza bir de **Lazy<T>** tipi girmektedir. **Base Class Library** içerisine dahil edilen bu yeni generic sınıf sayesinde **T** tipi için **Lazy Initialization** işlemi gerçekleştirilebilmektedir.

Bu nokta da **Lazy<T>** tipinin iki önemli **özelliği(Property)** olduğunu ifade etmemiz gerekiyor. **Value** ve **IsValueCreated**. Aslında **Value** özelliği ve**ToString()** metodu çağırılana kadar **T** tipi ile ilişkili bir yükleme işlemi yapılmadığını söylersek sanıyorum ki olay daha net bir şekilde anlaşılabilir. Ancak konuyu kod yardımıyla irdelemek elbetteki en iyi yoldur. İlk etapta aşağıdaki **Console** uygulaması kodlarına sahip olduğumuzu düşünelim.

```
using System.Collections.Generic;

namespace LazyInitializations
{
 class Program
 {
  static void Main(string[] args)
  {
   var productList = GetProducts();
   IncreaseListPrice(productList);
  }
```

```
 static void IncreaseListPrice(List<Product> products)
 {
  Console.WriteLine("Liste Fiyatları 1 birim arttırılacak");
  foreach (Product product in products)
  {
   product.ListPrice += 1;
  }
 }

 static List<Product> GetProducts()
 {
  Console.WriteLine("\tProduct listesi oluşturulacak");
  return new List<Product>
  {
   new Product{ProductId=2,Name="Avaya IP Phone", ListPrice=100.99},
   new Product{ProductId=3,Name="Toshiba 106 inc Tv", ListPrice=1200.99},
   new Product{ProductId=4,Name="Hp 6730b Laptop", ListPrice=980.49}
  };
 }
}

class Product
{
 public int ProductId { get; set; }
 public string Name { get; set; }
 public double ListPrice { get; set; }
}
}
```

Bu kod parçasında yer alan **GetProducts** isimli metod **Product** tipinden nesne örneklerinden oluşan bir **List<T>** koleksiyonunu geriye döndürmektedir. Diğer yandan **IncreaseListPrice** isimli metod da, parametre olarak gelen **Product** listesindeki her bir ürünün birim fiyatını 1 birim arttırmaktadır. **Main** metodu içerisinde kod işleyiş sırasına baktığımızda önce **List<Product>**tipinden olan koleksiyonunun doldurulduğunu, sonrasında ise **IncreaseListPrice** isimli metodun çağırıldığını görebiliriz. Bu örnek uygulamada aslında olağanüstü veya şaşırtıcı bir durum söz konusu değildir. Ancak bizi **Lazy Initialization**' a götürecek bir nedende olabilir. Söz gelimi bu kod parçasına göre ürün listesinin elde edilmesi beklenenden uzun sürüyor olabilir ve özellikle **IncreaseListPrice** metod çağrısı öncesinde başka işlerin yapıldığı da düşünülebilir. çok basit bir şekilde sembolize etmek istersek;

```
var productList = GetProducts();
// Bir takım Business çağrılar yapıldığını düşünelim
IncreaseListPrice(productList);
```

Buna göre aslında **ürün listesinin(List<Product>)** sadece işlem yapılacağı yerde yüklenmesini, başlangıçta yüklenerek zaman kaybına neden olunmamasını isteyebiliriz ki bu durumda Business çağrıların bekletilmemesi de sağlanmış olacaktır. İşte böyle bir durumda **Lazy<T>** tipine ait bir nesne örneği gerekli tembelliği bize sunabilir. Nasıl mı?

```
static void Main(string[] args)
{
 // var productList = GetProducts();
 Lazy<List<Product>> lazyProductList = new Lazy<List<Product>>(
    () => GetProducts()
   );
 // Bir takım Business çağrılar yapıldığını düşünelim
 Console.WriteLine("Product listesi yüklendi
mi?  {0}",lazyProductList.IsValueCreated?"Evet":"Hayır");
 IncreaseListPrice(lazyProductList.Value);
 Console.WriteLine("Product listesi yüklendi mi?
{0}", lazyProductList.IsValueCreated ? "Evet" : "Hayır");
 // IncreaseListPrice(productList);
}
```

İlk olarak **Lazy<List<Product>>** tipinden bir nesne örneği oluşturulduğu görülmektedir. Buna göre **T** olarak belirtilen **List<Product>** tipinden nesne örneği döndürecek bir operasyon için **Lazy Initialization** işleminin uygulanacağı bildirilmektedir. **Lazy<T>** tipinin bir kaç **aşırı yüklenmiş yüklenici metodu(Overloaded Constructor)** bulunmaktadır. örnekte **Func<T> temsilcisini(Delegate)** kullanan versiyon ele alınmıştır. Tahmin edileceği üzere burada **GetProducts** isimli metod çağırısı gerçekleştirilmektedir. İzleyen satırda bir takım **Business**işlemlerin yapıldığı düşünüldüğünde **Product** listesinin yüklenmesi işlemlerinin, bu operasyonu engellemediği düşünülebilir. Nitekim henüz **Product** listesiyüklenmemiştir. Bunu kod tarafında anlamanın yolu **Lazy<T>** nesne örneğinin **IsValueCreated** özelliğine bakmtaktır. Diğer yandan uygulamayı **Debug** ettiğimizde tam bu noktada durduğumuz takdirde aşağıdaki ekran görüntüsünü elde ederiz.

```
class Program
{
  static void Main(string[] args)
  {
    // var productList = GetProducts();
    Lazy<List<Product>> lazyProductList = new Lazy<List<Product>>(
      () => GetProducts()
        );
    // Bir takım Business çağrılar yapı
    Console.WriteLine("Product listes
    IncreaseListPrice(lazyProductList.Value);
    Console.WriteLine("Product listesi yüklendi mi? {0}", lazyProductList.Is
    // IncreaseListPrice(productList);
  }
```

lazyProductList ThreadSafetyMode=ExecutionAnd

IsValueCreated	false
IsValueFaulted	false
Mode	ExecutionAndPublication
Value	null
Raw View	

IsValueCreatted özelliği, **Lazy<T>** nesne örneğinin **Value** özelliği kullanılana kadar **False** değere sahiptir. **Value** özelliğinin çağırılması artık yapıcı metod içerisinde **Func<T>**temsilcisi ile bildirilen metodun icra edilmesi anlamına gelmektedir. Bir başka deyişle söz konusu **Product** listesinin yüklenmesi işlemi gerçekleştirilecektir. Bu durumda kod tarafında **IsValueCreated**özelliği **true** değere sahip olacaktır.
Yine **Debug** penceresinden olaya baktığımızda aşağıdaki ekran çıktısı ile karşılaştığımızı görebiliriz.

```
class Program
{
  static void Main(string[] args)
  {
    // var productList = GetProducts();
    Lazy<List<Product>> lazyProductList = new Lazy<List<Product>>(
      () => GetProducts
        );
    // Bir takım Business ç
    Console.WriteLine("Pr
    IncreaseListPrice(lazyProductList.Value);
    Console.WriteLine("Product listesi yüklendi mi? {0}", lazyProductL
    // IncreaseListPrice(productList);
  }
```

lazyProductList ThreadSafetyMode=ExecutionAndPublicati

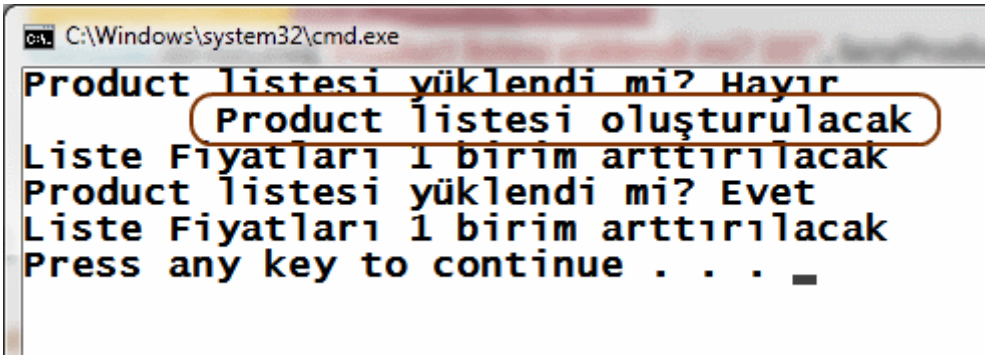
IsValueCreated	true
IsValueFaulted	false
Mode	ExecutionAndPublication
Value	Count = 3
Raw View	

Dikkat edileceği üzere **Value** özelliği **3** adet **Product** nesne örneğini işaret etmektedir.

Lazy<T> kullanımında dikkate değer noktalardan biriside **Value** özelliğine ilk çağrıdan sonra tekrar erişilmesi halidir. Durumu aşağıdaki kod parçası ile irdeleyebiliriz.

```
...
IncreaseListPrice(lazyProductList.Value);
Console.WriteLine("Product listesi yüklendi mi? {0}", lazyProductList.IsValueCreated ?
"Evet" : "Hayır");
// IncreaseListPrice(productList);
var productList = lazyProductList.Value;
IncreaseListPrice(productList);
```

Burada **IncreaseListPrice** metodunun parametresinde **Lazy<T>** nesne örneği için bir **Value** çağrısı yapılmaktadır. Bu **Value** özelliği için yapılan **ilk çağrı** olduğundan**GetProducts()** metodunun çağırılması işlemi bu noktada yapılmaktadır. Ancak son iki satırda **Value** özelliği tekrardan kullanılmıştır. İlk olarak **productList** isimli bir değişkene değer ataması gerçekleştirilmiş arından elde edilen değer **IncreaseListPrice** metoduna parametre olarak geçirilmiştir. Burada **Value** özelliğine bir çağrı yapılması nedeni ile**GetProducts** isimli metodun bir kere daha çalıştırılacağı düşünülebilir. Ama böyle olmayacaktır. Bunu **debug** sırasında görebileceğimiz gibi kodun çalışma zamanı çıktısında da fark edebiliriz.



Görüldüğü gibi "**Product Listesi Oluşturulacak"** ifadesi sadece bir kere çağırılmıştır. Son olarak **Lazy<T>** tipinin **Thread Safety**' yi varsayılan olarak sunduğunu bu nedenle**Concurrent** operasyonlarda ele alınabileceğini de belirtelim. **Base Class Library** içerisine **.Net Framework 4.0** ile birlikte gelen pek çok yeni tip söz konusudur. **Lazy<T>** bunlardan sadece bir tanesiydi. İlerleyen zamanlarda diğer yenilikleri de incelemeye çalışıyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

LazyInitializations_.rar (27,13 kb) **[örnek Visual Studio 2010 Ultimate sürümü ile geliştirilmiş ve test edilmiştir]**

Kod Bazlı Workflow Service Geliştirmek ve Yayınlamak (2010-10-05T01:20:00)

workflow foundation,workflow foundation 4.0,visual studio 2010,workflow service,

 Merhaba Arkadaşlar,

Bildiğiniz üzere bir süredir **NedirTv?com** desteğinde **"Workflow Foundation 4.0 öğreniyorum"** isimli bir seri üzerinde çalışmaktayız. Bu seride başlangıç seviyesinden orta seviyeye kadar, bir kaç ayrı derste **Workflow Foundation** kavramını öğrenmeye gayret ettik. Bu seriye dahil etmek istediğim bir konu da, **Workflow Service** örneklerinin tamamen kod bazında yazılması ve **IIS(Internet Information Services)** dışındaki bir uygulama tarafından host edilmesiydi. Ancak konu biraz karmaşık olduğundan ve tabiri yerinde ise yandaki resimde görülen **Puzzle**' a benzemediğinden, yazı haline getirilmesinin daha iyi olacağına karar verdim. Hem böylece ben de unuttuğum zamanlarda bu yazıma bakarak hatırlayabilirim. öyleyse derin bir nefes alalım ve yola koyulalım.

İlk olarak ne yapmak istediğimizi açık ve net bir şekilde ortaya koymaya çalışalım. **WCF Eco System** yapısının da önemli bir parçası olan**Workflow Service**' ler yardımıyla iş akışlarının servis bazlı olarak dış ortama sunulması mümkündür. Bu noktada özellikle **Visual Studio 2010** tarafında yer alan **WCF Workflow Service Application** şablonu ve **Workflow Designer** işlerimizi inanılmaz ölçüde kolaylaştırmaktadır. Ancak elimizin altında sadece **.Net Framework 4.0** olduğunu düşünelim. Hımmm... Bu durumda **Visual Studio 2010** gibi bir **IDE**' mizin olmadığını da varsayarsak bir **Workflow Service** örneğinin **XAML** içeriğini yazmak istemeyebiliriz. Dolayısıyla kod tarafında **.Net** tiplerinden yararlanarak ilerlemek daha kolay olabilir*(Gerçi biz örneğimizdeki kod parçasını Visual Studio 2010 ile yazıyoruz ama olsun. Kimseye çaktırmayın)*

Ne yapmak istediğimizi sanıyorum ki biraz daha net anlayabildik. En basit haliyle bir **Workflow Service** örneğini kod tarafında oluşturmak istiyoruz. Ancak bu yeterli değil. Nitekim tasarlanan bu **Workflow Service** örneğinin aynı zamanda **host** edilerek kullanıma sunulması da gerekmektedir. İşte bu noktada **IIS(Internet Information Services)** dışında bir uygulamayı geliştirmek istediğimizi düşünebiliriz. İlk etapta basit bir Console uygulaması işimizi görebilir. *(Ancak tabiki WPF, Windows Forms hatta bir Asp.Net uygulamas dahi Workflow Service örneklerini host edip çalıştıracak şekilde tesis edilebilir)* Bu Console uygulaması, **WorkflowServiceHost** tipinden de yararlanarak gerekli

çalışma zamanını tesis edecek ve bildirilen **Workflow Service** örneğini dış ortama sunuyor olacaktır.

İşe başlamadan önce eğer imkanınız var ise bir **Workflow Service** örneğinin **Visual Studio 2010** ortamında WPF tabanlı Designer yardımıyla geliştirilmesini incelemenizi öneririm. Geliştireceğimiz **Workflow Service** bir servis olduğundan istemciden gelecek olan çağrıları kabul etmeli ve bir iş akışı başlatarak sonuçları istemci tarafına yönlendirmelidir. **WCF**tabanlı bir servis söz konusu olduğuna göre istemcinin çağrıda bulunabileceği operasyonların ve dolayısıyla **servis sözleşmesinin(Service Contract)** önceden tanımlanmış olması gerekmektedir. Ki bu sayede istemcilerin söz konusu adresten yayınlanan servis üzerinden hangi operasyonları çağırabileceği belirlenmiş olacaktır. İşte örneğimizde kullanacağımız servis sözleşmesi.

```
[ServiceContract]
  public interface IHelloService
  {
    [OperationContract]
    double Sum(double x, double y);
  }
```

IHelloService interface tipi dikkat edileceği üzere **ServiceContract** niteliği ile imzalanmıştır. Diğer taraftan çok basit olarak **Sum** isimli bir operasyon içermektedir. Söz konusu operasyon istemci tarafına açılacak olan bir fonksiyonelliği belirtmektedir. **Sum** isimli operayonun aldığı **double** tipinden olan iki parametrede, istemci tarafından gönderilecek değişkenler olduğunu göstermektedir.

Peki **Workflow Service** örnekleri dış dünyadan gelen istemci taleplerini nasıl almaktadır? Diyelim ki aldılar ve işettiler. İş akışı sonucu istemci tarafına bir değer göndermek isterlerse bunu nasıl gerçekleştirebilirler? Bu noktada **Receive** ve **SendReply** isimli aktivite bileşenlerinden yararlanıldığını söyleyebiliriz. Dolayısıyla kod tarafında bu aktivite bileşenlerini ele almamız gerekmektedir. Lakin bu aktivite bileşenleri başka bir **Container** içerisinde yer almalıdır. örneğin bir **Sequence** aktivite bileşeni Container olarak düşünülebilir. çok doğal olarak istemcilerin gönderdiği parametrelerin **Sequence** içerisindeki diğer aktivite bileşenleri tarafından da kullanılması gerekebilir. Bu durumda **Sequence** seviyesinde **Variable** tanımlamalarının yapılması uygundur. İhtiyacımız olan **Variable** tanımlamaları ise aşağıdaki gibidir.

```
Variable<CorrelationHandle> __handle=new
Variable<CorrelationHandle>("Request_Handle");
Variable<double> x=new Variable<double>("X");
Variable<double> y=new Variable<double>("Y");
Variable<double> result=new Variable<double>("Result");
```

x, y ve **result** isimli **Variable** tanımlamaları tahmin edeceğiniz üzere toplama işlemi için gereklidir. Ancak burada

birde **__handle** isimli **CorrelationHandle** tipinden **Variable** tanımlaması yer almaktadır. **Workflow Service** örnekleri oluşturulduğunda bildiğiniz üzere istemci ile arada bir **oturum(Session)** oluşmaktadır. Bu noktada özellikle istemciden gelen mesajın sunucu tarafındaki hangi servis örneğine ait olduğunun anlaşılması noktasında **Correlation** çeşitlerinden yararlanılmaktadır. Burada tanımlanan **Variable, Receive** aktivite bileşeni tarafından kullanılacaktır (Detaylar için Correlation Nedir? Yenir mi? İçilir mi?)

Aslında tam bu noktada **Receive** aktivitesini de tanımlayabiliriz. Aşağıdaki gibi.

```
Receive receive=new Receive{
                    CanCreateInstance=true,
                     OperationName="Sum",
                      ServiceContractName="IHelloService",
                      CorrelatesWith=__handle,
                      Content=ReceiveContent.Create(new
Dictionary<string,OutArgument>{
                         {"xValue",new OutArgument<double>(x)},
                         {"yValue",new OutArgument<double>(y)},
                      }
                      )
                };
```

Receive aktivite bileşeni içerisinde **set** edilmiş özellikler önemlidir. **OperationName** ile dikkat edileceği üzere istemciler için kullanılabilecek bir operasyon adı bildirimi yapılmaktadır. Diğer yandan **Content** özelliğine atanan **OutArgument** tipli iki parametre, az önce tanımladığımız **x** ve **y Variable**' larını kullanmaktadır. Bir başka deyişle istemciler için **xValue** ve **yValue**isimli değişkenler tanımlanmıştır. **ServiceContractName** özelliği ile servis sözleşmesi bildirimi yapılmaktadır. **Receive** aktivite bileşeni için gerekli **Correlation** ayarı ise**CorrelatesWith** özelliği ile belirtilmektedir.
Aslında **Receive** aktivite bileşeni biraz sonra kodlayacağımız **WorkflowService** örneğinin **Body** özelliği içerisinde kullanılmak istenebilir. Ancak böyle bir durumda **SendReply** aktivite bileşeninin ihtiyacı olan **Request** özelliğinin atanacağı **Receive** referansı tanımlanamayacaktır. **Receive** aktivite bileşeninin dışarıda tanımlanmasının sebebi budur.
Artık **WorkflowService** nesne örneğini tanımlayarak yolumuza devam edebiliriz. İşte kod içeriğimiz.

```
WorkflowService wfService = new WorkflowService
        {
           Name = "HelloService",
           Endpoints = {
              new Endpoint
           {
               ServiceContractName="IHelloService",
```

```
            Binding=new BasicHttpBinding(),
             AddressUri=new Uri("http://localhost:5001/HelloService")
             , Name="HelloServiceEndpoint"
        }
        },
        ConfigurationName="HelloServiceConfig",
        Body = new Sequence
        {
           Variables={x,y,__handle,result},
           Activities =
           {
              receive,
              new SumActivity{
                  X=x,
                  Y=y,
                 Result=result
              },
              new SendReply{
                  Request=receive,
                   Content=SendContent.Create(new InArgument<double>(result))
              }
           }
        }
    };
```

WorkflowService örneği içerisinde yer alan en önemli özelliklerden birisi **Endpoints**' dir. Bu özelliğe göre birden fazla **Endpoint** bildirimi yapılabilmektedir.
örneğimizde **BasicHttpBinding** tabanlı,**IHelloService** isimli servis sözleşmesini kullanan ve **http://localhost:5001/HelloService** adresi üzerinden yayın yapan
bir **Endpoint** bildirimi söz konusudur. **WorkflowService** örneği oluşturulurken kullanılan **ConfigurationName** özelliği, konfigurasyon dosyası*(örneğimizde app.config)* içerisindeki bir servis bloğunu işaret etmektedir. Aslında bu blokta servisin dış dünyaya **Metadata**paylaşımını yapacağını bildirebiliriz. Bildiğiniz üzere **Metadata Publishing** sayesinde istemcilerin **WSDL** içeriğine ulaşması mümkündür ve bu sayede gerekli **Proxy** tiplerini kolayca üretebilirler. *(Ancak tabiki bazı hallerde özellikle güvenlik sebebi ile Metadata bilgisini istemci tarafına indirilebiliyor olması arzu edilmeyebilir)* İşte **app.config** dosyasının içeriği.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <startup>
     <supportedRuntime version="v4.0"
sku=".NETFramework,Version=v4.0,Profile=Client" />      </startup>
 <system.serviceModel>
   <behaviors>
```

```
    <serviceBehaviors>
      <behavior name="HelloServiceBehavior">
        <serviceMetadata httpGetEnabled="true" httpGetUrl
="http://localhost:5001/HelloService"/>
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <services>
    <service name="HelloServiceConfig"
behaviorConfiguration="HelloServiceBehavior"/>
  </services>
 </system.serviceModel>
</configuration>
```

WorkflowService örneğinin **Body** özelliği içerisinde ise sırasıyla **Receive, SumActivity** ve **SendReply** aktivite bileşenlerini içeren bir **Sequence** aktivite bileşeni tanımlandığı görülmektedir. Bu bileşenin **Variables** özelliğinde ise, tüm alt aktiviteler tarafından kullanılacak olan **x,y, result**ve **__handle** isimli değişken bildirimleri yer almaktadır. **Sequence** aktivite bileşeninin **Activities** özelliğine bakıldığında ise sırasıyla **Receive, SumActivity** ve **SendReply** aktivite bileşenlerinin tanımlandığı gözlemlenmektedir. Durun bir dakika **SumActivity**' mi? Bu bizim tarafımızdan tanımlanmış **CodeActivity<double>** türevli bir aktivite sınıfıdır ve içeriği aşağıdaki gibidir.

```
public class SumActivity
    : CodeActivity<double>
{
    public InArgument<double> X { get; set; }
    public InArgument<double> Y { get; set; }

    protected override double Execute(CodeActivityContext context)
    {
        return X.Get(context) + Y.Get(context);
    }
}
```

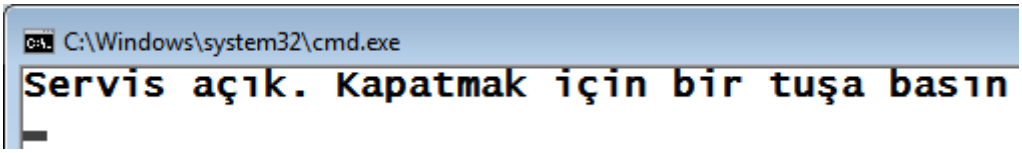
SumActivity tahmin edeceğiniz üzere istemciden gelen **x** ve **y** değişkenlerini kullanmak üzere tasarlanmıştır. Elbette bu nokada küçük bir soru vardır. **Receive** aktivite bileşenine istemci tarafından gelen **xValue** ve **yValue** değeleri, **SumActivity** örneğine nasıl aktarılacaktır? Dikkat edileceği üzere **SumActivity** örneklenirken **X, Y** ve **Result** özelliklerine sırasıyla **x,y** ve**result** değerleri atanmıştır*(Büyük küçük harf ayrımına dikkat edelim)*. Bu değişkenler **Sequence** seviyesindeki **Variable**' lardır ve **Receive** aktivite bileşeni içerisinde **Content**özelliği içerisindeki bildirim yardımıyla set edilmektedir. çok doğal olarak **Result** özelliğine gelen değer, **result Variable**' ına

aktarılmakta ve bu da **SendReply** aktivite bileşeni tarafından kullanılarak istemciye cevap olarak döndürülmektedir*(SendReply bileşeninin Content özelliğine dikkat edelim)*

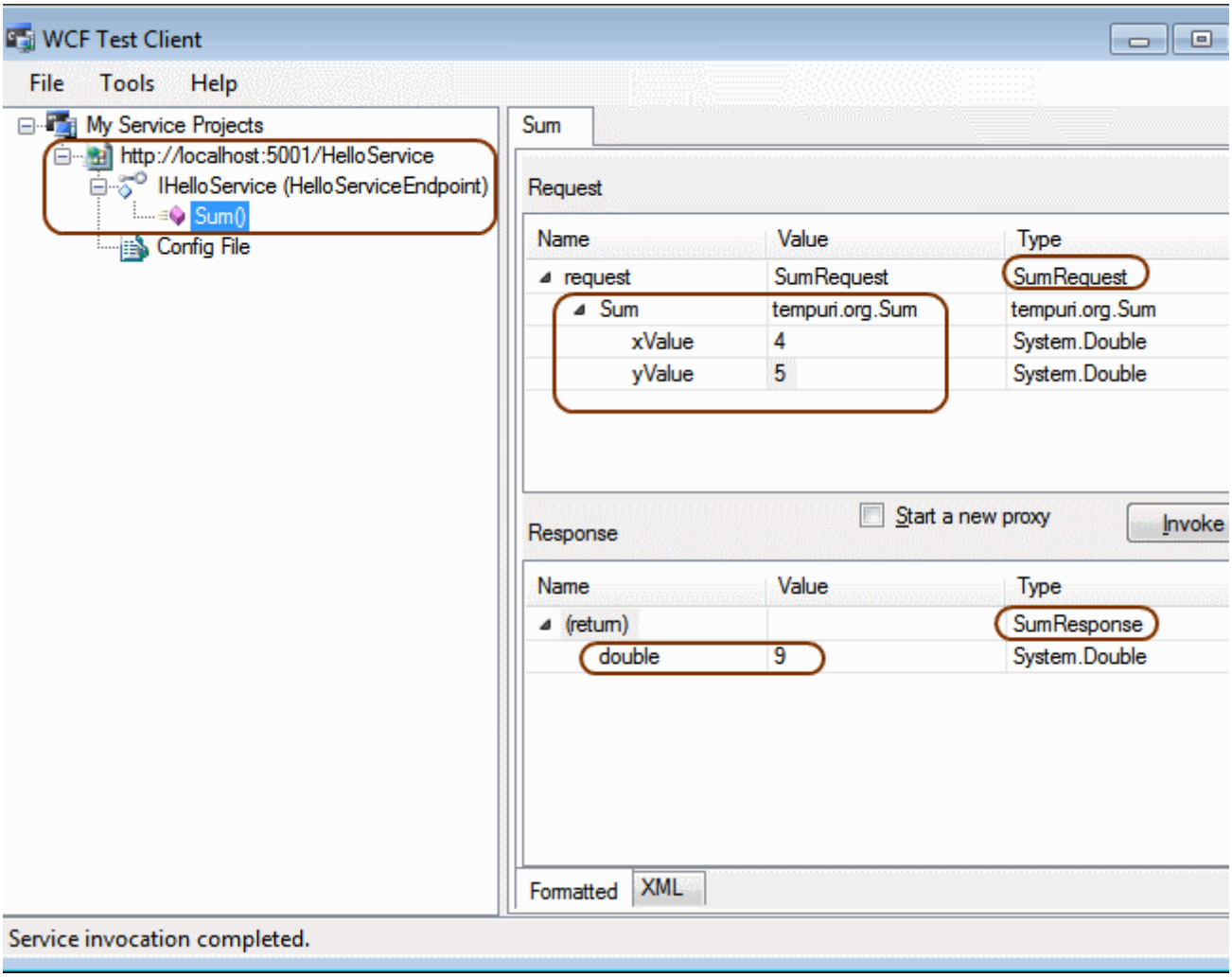
Artık **WorkflowService** örneğini host etmek üzere gerekli kodları aşağıdaki gibi yazabiliriz.

```
WorkflowServiceHost serviceHost = new WorkflowServiceHost(wfService);
serviceHost.Open();
Console.WriteLine("Servis açık. Kapatmak için bir tuşa basın");
Console.ReadLine();
serviceHost.Close();
```

WorkflowServiceHost nesnesi örneklenirken parametre olarak **wfService** nesne örneği verilmektedir. **WorkflowServiceHost** tipi **Workflow Service**' lerin ayağa kaldırılması, servis olarak sunulması, gerekli çalışma zamanı ortamının kurgulanması gibi işlemleri üstelenen bir tip olarak düşünülmelidir. Buna göre istemci uygulama çalıştırıldığında aşağıdaki sonuçlar ile karşılaşmamız gerekmektedir.



Peki ya istemci tarafı? çalışan bu servisin sunduğu **Workflow Service** örneğinin işe yaradığını nasıl göreceğiz? **WcfTestClient** uygulaması bu noktada işimizi görüyor olacaktır. Söz konusu uygulamayı **Visual Studio Command Prompt** üzerinden çalıştırdıktan sonra **http://localhost:5001/HelloService** için bir talepte bulunmamız yeterlidir. Eğer herşey yolunda giderse **Sum** operasyonunu test edebilir ve aşağıdakine benzer sonuçları alabiliriz.



İşte bu kadar. 😃 Görüldüğü gibi **Workflow Service** örneğimizi başarılı bir şekilde çağırdık. Peki ya bundan sonrası? örneğimizi **HTTP** tabanlı olarak tasaladığımızı fark etmişsiniz. Ancak sizde söz konusu örneği **TCP** bazlı çalışacak şekilde geliştirmeyi deneyebilirsiniz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WorkflowConsoleApplication1.rar (28,99 kb)

Struct, Class ve Default Constructors - İnanmak İstiyorum (2010-09-27T18:50:00)

c#,c# temelleri,

Merhaba Arkadaşlar,

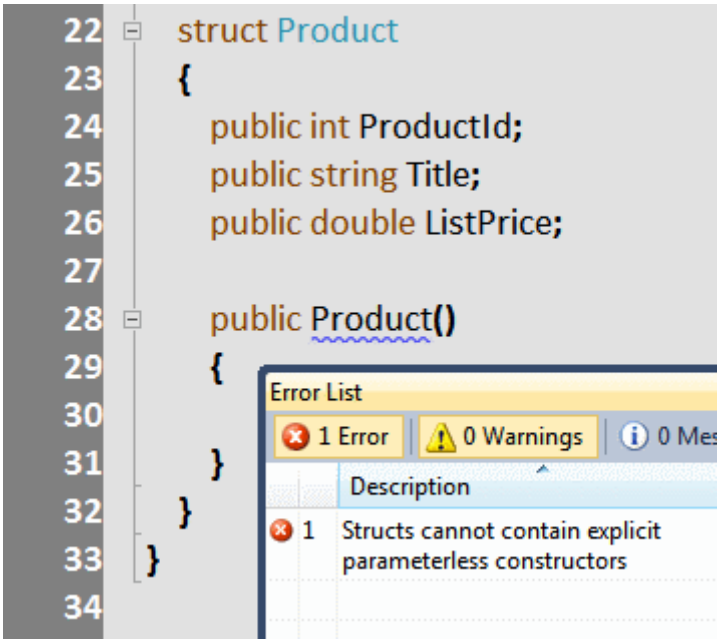
Sanıyorum 90 yıllarda en çok izlediğim dizilerden birisi **The X Files (Gizli Dosyalar)** idi. Her bölümde olağan dışı konuların ele alındığı ve birbirlerine tamamen zıt iki karekter olan ama bu sayede gizli dosyaların işlenmesinde birbirlerini sürekli olarak dengeleyen **Ajan Dana Scully** ve **Fox Mulder**' ın maceraları gerçeklikten uzak olsa da, benim gibi bilim kurgu severleri ekrana bağlamak için yeterliydi. Diziyi her izleyişimde en çok hoşuma giden sahneler ise genellikle **Fox Mulder**' ın bir soru sorup ortağı dahil herkesi bir kaç saniyeliğine de olsa düşünmeye itmesiydi*(yeni nesil argoda ifade etmek istersek dumura uğratmasıydı)*. Geçtiğimiz günlerde yine **X Files** dizisinin bir kaç bölümünü heyecanlı bir şekilde tekrardan izlerken bazen yaptığımız işte de, **Ajan Fox Mulder**' ın ürettiği gibi şüphe uyandırıcı soruların oluştuğunu düşünmeye başladım. Tabi soruyu hemen sormak istemiyorum. Olaya biraz daha heyecan katmakta yarar var. Bakalım bu yazımızın **Fox Mulder**' ı olabilecek miyim?

Konumuz **C#** programlama dilinin temelleri arasında sayılmaktadır. **Class** ve **Struct** tiplerinde **Default Constructor(Varsayılan Yapıcı Metod)** kullanımı. Bildiğiniz üzere **.Net Framework Base Class Library** üzerinde tiplere en üst seviyeden baktığımızda **değer(Value)** ve **referans(Reference)** türleri olarak ikiye ayırmaktayız. **Struct** tipleri değer türü, **Class'** lar ise referans türleri olarak ele alınmaktadır. Bu iki veri türününün bellekte tutuluş şekilleri, aralarındaki atamalarda gösterdikleri davranışsal farklılıklar bir kenara, özellikle kendi geliştirdiğimiz taslaklarında dikkat etmemiz gereken noktaların başında da, yapıcı metod kullanımları gelmektedir. Dilerseniz aşağıdaki kod parçasını göz önüne alarak işe koyulalım.

```
namespace ClassStructAndConstructors
{
   class Program
   {
      static void Main(string[] args)
      {
      }
   }

   class Person
   {
```

```
        public int PersonId;
        public string Name;
        public double Salary;

        public Person()
        {

        }
    }

    struct Product
    {
        public int ProductId;
        public string Title;
        public double ListPrice;

        public Product()
        {

        }
    }
}
```

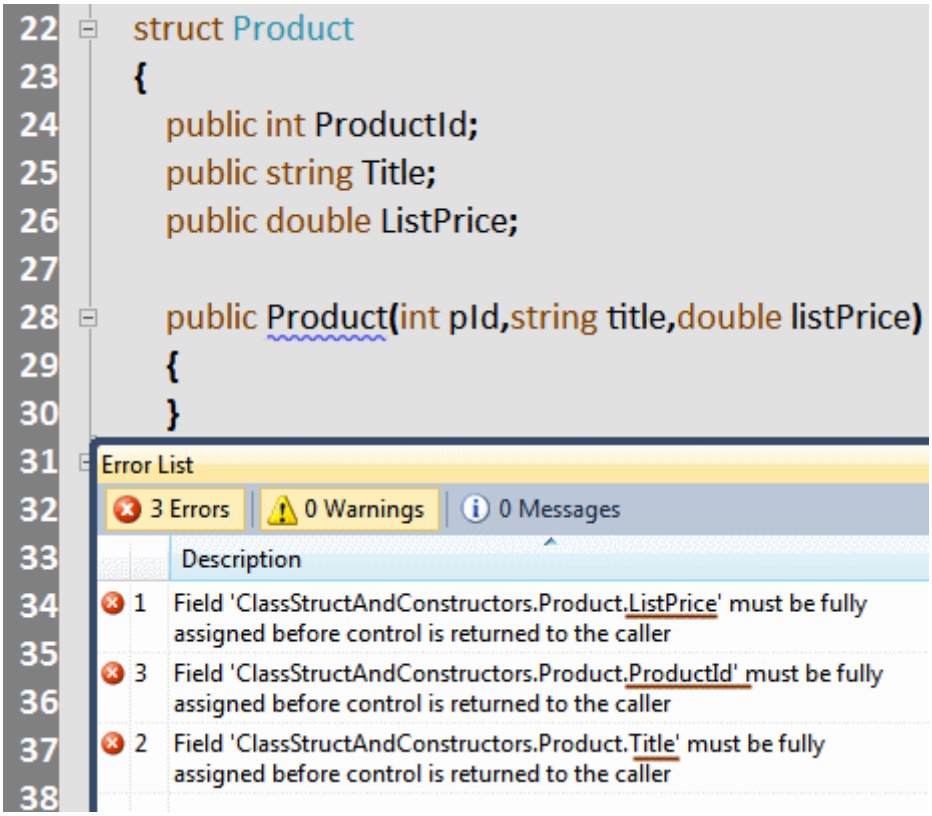
Bu kod parçasında **Person** isimli bir **Class** ve **Product** isimli bir **Struct** tanımlanmıştır. Her iki tip içerisinde de **varsayılan yapıcı metod(Default Constructor)** kullanımı söz konusudur. Ancak kodu derlediğimizde bir hata mesajı ile karşılaştığımızı görürüz.



Kural 1 geliyor : Bu **Struct** olmanın kurallarından birisidir. Buna göre bir **Struct**'ın, **Class**'ların aksine parametresi olmayan yapıcı metod içermesi söz konusu değildir. çok doğal olarak aşağıdaki gibi bir kullanım tercih edilebilir.

```
namespace ClassStructAndConstructors
{
   class Program
   {
      static void Main(string[] args)
      {
      }
   }

   class Person
   {
      public int PersonId;
      public string Name;
      public double Salary;

      public Person(int pId,string name,double salary)
      {

      }
   }

   struct Product
   {
      public int ProductId;
      public string Title;
      public double ListPrice;

      public Product(int pId,string title,double listPrice)
      {
      }
   }
}
```

Bu kez hem **Class** hem de **Struct** için parametre alan yapıcı metodlar söz konusudur. Ancak yine bir **Compile Time** hatası alınılması kaçınılmazdır.



Dikkat edileceği üzere **Class** tipi için parametreli kullanım söz konusu iken, **Struct** için aynı durum gerçeklenememektedir.

Kural 2 geliyor : Bir **Struct** için parametreli yapıcı metod kullanılması halinde, içerideki tüm alanlara ilk değerlerinin atanması gerekmektedir. Dolayısıyla **Struct**' ımıza ait parametreli yapıcı metod içeriği aşağıdaki gibi düzenlenirse sorun kalmayacaktır.

```
public Product(int pId,string title,double listPrice)
{
  ProductId = pId;
  Title = title;
  ListPrice = listPrice;
}
```

Şimdi kod parçasını aşağıdaki gibi değiştirdiğimizi düşünelim.

```
namespace ClassStructAndConstructors
{
   class Program
   {
      static void Main(string[] args)
      {
         Person burak = new Person();
         burak.PersonId = 1001;
         burak.Name = "Burak Selim Şenyurt";
```

```
            burak.Salary = 1;

            Product mouse = new Product();
            mouse.ProductId = 2001;
            mouse.Title = "Microsoft Optical Mouse";
            mouse.ListPrice = 9.99;
        }
    }

    class Person
    {
        public int PersonId;
        public string Name;
        public double Salary;
    }

    struct Product
    {
        public int ProductId;
        public string Title;
        public double ListPrice;
    }
}
```

Bu sefer ne **Class** ne de **Struct** için bir yapıcı metod bildiriminde bulunulmamıştır. Kod başarılı bir şekilde derlenmiştir. **Main** metodu içerisinde ise her iki tipe ait nesne örnekleri oluşturulmuş ve alanlarına bazı değerler atanmıştır. Hımmm...İşte **Fox Mulder sorusu** geliyor?

Struct' lar için açık bir şekilde varsayılan yapıcı metod tanımlayamıyorsak eğer, nasıl oluyorda new Struct() şeklinde bir metod çağrısı yapabiliyoruz?

çok doğal olarak burada **Dana Scully** olsaydı söz konusu soruyu cevaplamak için hasta üzerinde otopsi yapmaya, bir başka deyişle **IL(Intermediate Language)** koduna bakmaya karar verirdi Biz de böyle yapıyor olacağız. *(Bu amaçla **ILDASM** veya **Red Gate' s .Net Reflector** gibi araçlardan yararlanabiliriz)*

Person sınıfı için üretilen IL çıktısı aşağıdaki gibidir.

```
.class private auto ansi beforefieldinit Person
    extends [mscorlib]System.Object
{
    .method public hidebysig specialname rtspecialname instance void .ctor() cil
managed
    {
```

```
    .maxstack 8
    L_0000: ldarg.0
    L_0001: call instance void [mscorlib]System.Object::.ctor()
    L_0006: ret
  }


  .field public string Name

  .field public int32 PersonId

  .field public float64 Salary

}
```

Dikkat edileceği üzere varsayılan bir yapıcı metod otomatik olarak üretilmiştir. Ajan Dana Scully bu durum karşısında **Product** isimli **Struct** için de aynı durumun söz konusu olacağını düşünmektedir. Yüzünde bir tebessümle **Mulder**' a döner ve "Tamam Buldum" der gibi bakar. Oysaki mercekten tekrar baktığında aşağıdaki **IL** çıktısı ile karşılaşacaktır.

```
.class private sequential ansi sealed beforefieldinit Product
  extends [mscorlib]System.ValueType
{
  .field public float64 ListPrice

  .field public int32 ProductId

  .field public string Title

}
```

Fark edileceği üzere burada **ctor** gibi bir metod tanımı bulunmamaktadır. Demek ki yanlış bir yerde arama yapılmaktadır. **Scully** derhal merceği ayarlar ve bu kez **Main** metodu içeriğine odaklanır.

```
.class private auto ansi beforefieldinit Program
  extends [mscorlib]System.Object
{
  .method public hidebysig specialname rtspecialname instance void .ctor() cil managed
  {
    .maxstack 8
    L_0000: ldarg.0
    L_0001: call instance void [mscorlib]System.Object::.ctor()
    L_0006: ret
  }
```

```
  .method private hidebysig static void Main(string[] args) cil managed
  {
    .entrypoint
    .maxstack 2
    .locals init (
      [0] class ClassStructAndConstructors.Person burak,
      [1] valuetype ClassStructAndConstructors.Product mouse)
    L_0000: nop
    L_0001: newobj instance void ClassStructAndConstructors.Person::.ctor()
    L_0006: stloc.0
    L_0007: ldloc.0
    L_0008: ldc.i4 0x3e9
    L_000d: stfld int32 ClassStructAndConstructors.Person::PersonId
    L_0012: ldloc.0
    L_0013: ldstr "Burak Selim \u015eenyurt"
    L_0018: stfld string ClassStructAndConstructors.Person::Name
    L_001d: ldloc.0
    L_001e: ldc.r8 1
    L_0027: stfld float64 ClassStructAndConstructors.Person::Salary
    L_002c: ldloca.s mouse
    L_002e: initobj ClassStructAndConstructors.Product
    L_0034: ldloca.s mouse
    L_0036: ldc.i4 0x7d1
    L_003b: stfld int32 ClassStructAndConstructors.Product::ProductId
    L_0040: ldloca.s mouse
    L_0042: ldstr "Microsoft Optical Mouse"
    L_0047: stfld string ClassStructAndConstructors.Product::Title
    L_004c: ldloca.s mouse
    L_004e: ldc.r8 9.99
    L_0057: stfld float64 ClassStructAndConstructors.Product::ListPrice
    L_005c: ret
  }
}
```

new Person() çağrısı için **IL** tarafında **newobj** çağrısını gerçekleştirilirken, **new Product()** satırına karşılık olarak **initobj** isimli bir çağrının gerçekleştirildiği görülmektedir. **initobj** için **MSDN** kaynaklarında yapılan açıklama şu şekildedir.

.NET Framework Class Library

OpCodes.Initobj Field

Initializes each field of the value type at a specified address to a null reference or a 0 of the appropriate primitive type.

Namespace: System.Reflection.Emit
Assembly: mscorlib (in mscorlib.dll)

Bir başka deyişle **initobj** çağrısının uygulandığı **Struct** içerisindeki **Primitive** tipler için, varsayılan ilk değer atamaları gerçekleştirilmektedir. Buna göre sayısal değerler için**0** veya **0.0**, **bool** için **false** ve **referans** türleri için de **null** değerlerin atanması söz konusudur. Açıkça ifade etmek gerekirse **Struct**' lar için varsayılan bir yapıcı metod söz konusu olmasa dahi, **IL** tarafında bu fonksiyonelliği üstelenen bir çağrı yapılmaktadır *(Base Class Library Team tarafından bu durum implicit default constructor)* olarak adlandırılmaktadır.

Ajan **Scully** gözlerini mercekten kaldırır ve **Fox**' a döner. Bu kez yüzünde haklı bir tebessüm vardır ve **"Endişelerin boşunaymış Mulder. Uzaylıların bu işle hiç bir alakası yok"** der Başka bir macerada görüşünceye dek hepinize mutlu günler dilerim.

ClassStructAndConstructors.rar (21,32 kb) **[örnek Visual Studio 2010 Ultimate RTM sürümü üzerinde geliştirilmiş ve test edilmiştir]**

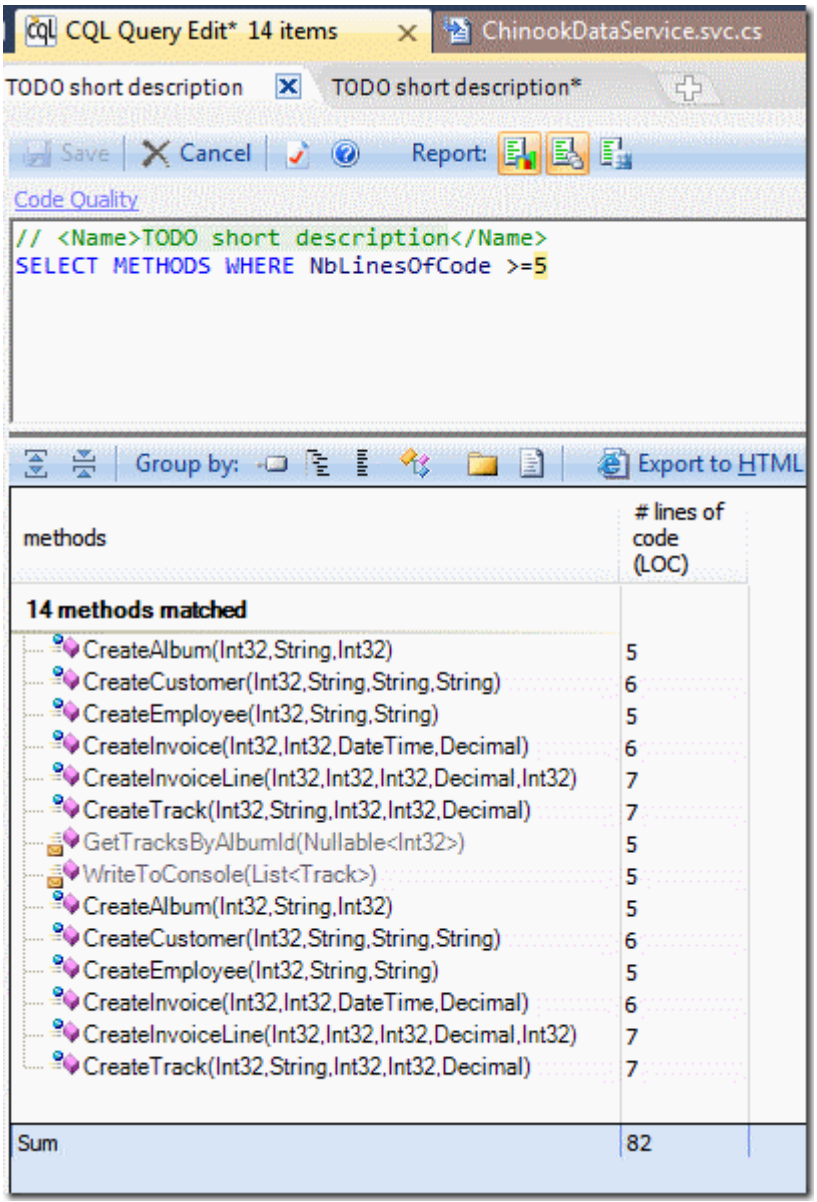
[NDepend Tool ve CQL(Code Query Language) (2010-09-23T11:16:00)]

visual studio 2010,ndepend,cql,code query language,

Merhaba Arkadaşlar,

Yandaki resimde **Visual Studio 2010 Ultimate** ortamına ait bir ara pencere görmektesiniz. Dikkat edeceğiniz üzere

Select Methods Where NbLinesOfCode>=5



şeklinde bir sorgu cümlesi var

Oppsss!!! Bu nasıl bir sorgu cümlesi? Tahmin etmeye çalışalım. Sanki **uygulamadaki kod satır sayısı 5' ten fazla olan metodların adlarını** döndüren bir sorgu cümlesi gibi

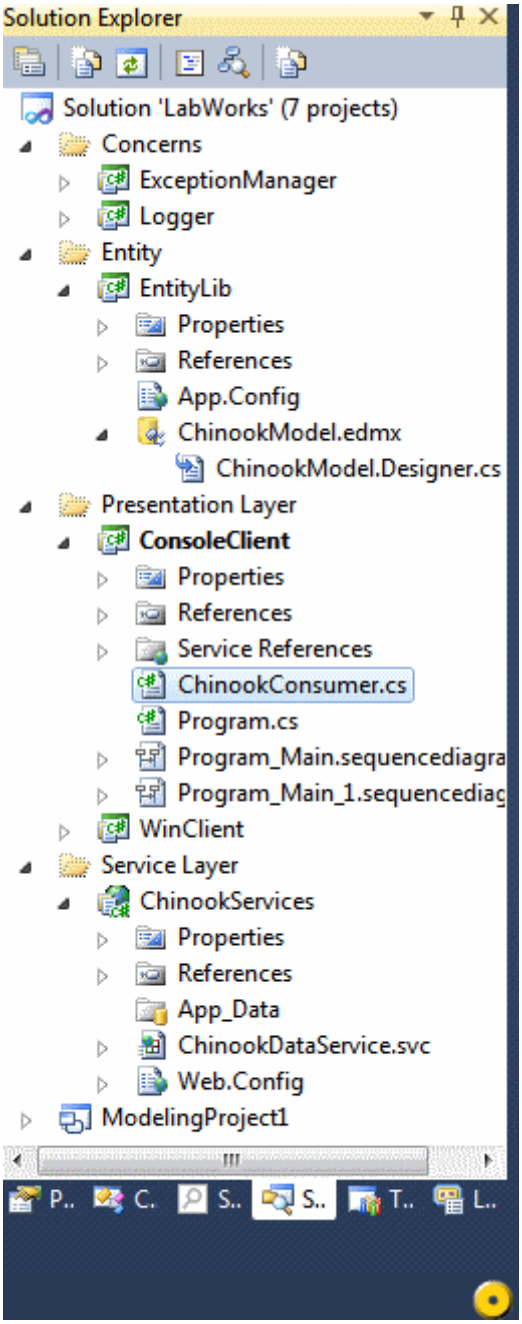
Şu anda bir **Code Query Language** örneğinin **Visual Studio 2010 IDE**' si içerisindeki raporlama sonucunu görmektesiniz.

Peki ben bu sorguya nasıl mı kavuştum?

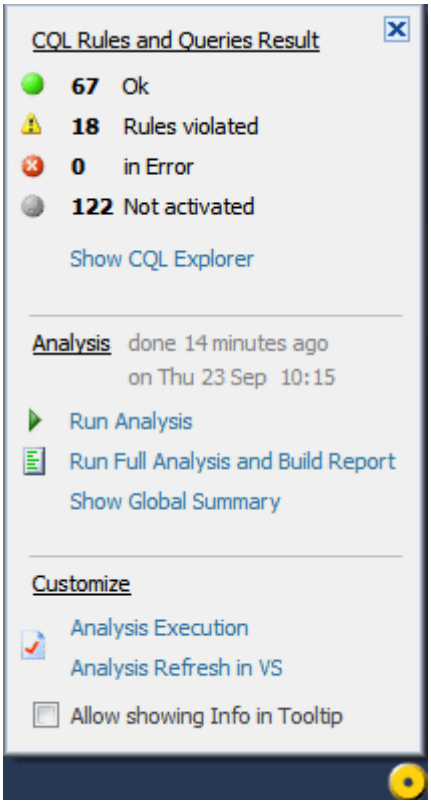
Sevgili **Patrick Smacchia(C# MVP)**, **NDepend** isimli bir **.Net** analiz ürünü geliştirdiklerini bir süre önce bana bildirmişti. Sonrasında ürünün tam sürümünü gönderdi ve bende incelememi, eğer varsa fikirlerimi iletmemi rica etti.

Bildiğiniz üzere **Visual Studio 2010 IDE**' si içerisinde **Architecture** sekmesinde erişip kullanabileceğimiz bazı analiz operasyonları mevcut. Söz gelimi **Assembly'** lar, **tipler vb…** arası bağımlılıkları çıkartabiliyoruz veya bir metodun **Sequence Diagram**' ına ulaşabiliyoruz. Bazı var olan mimari kalıpları uygulatıp doğrulatabiliyoruz. Ancak yine de büyük çaplı projelerin yazılması sırasında, bazı **metric'** lere uyulup uyulmadığını denetlemek, projenin genel görünümünü rapolarmak, daha detaylı görsel şemalar görmek ve derin bilgi edinmek isteyebiliyoruz. Bu tip ihtiyaçları karşılamak için başka araçlar da var ancak**NDepend** özellikle **CQL(Code Query Language)** ve **Raporlama** kısımlarındaki başarısı ile bence biraz daha ön plana çıkıyor.

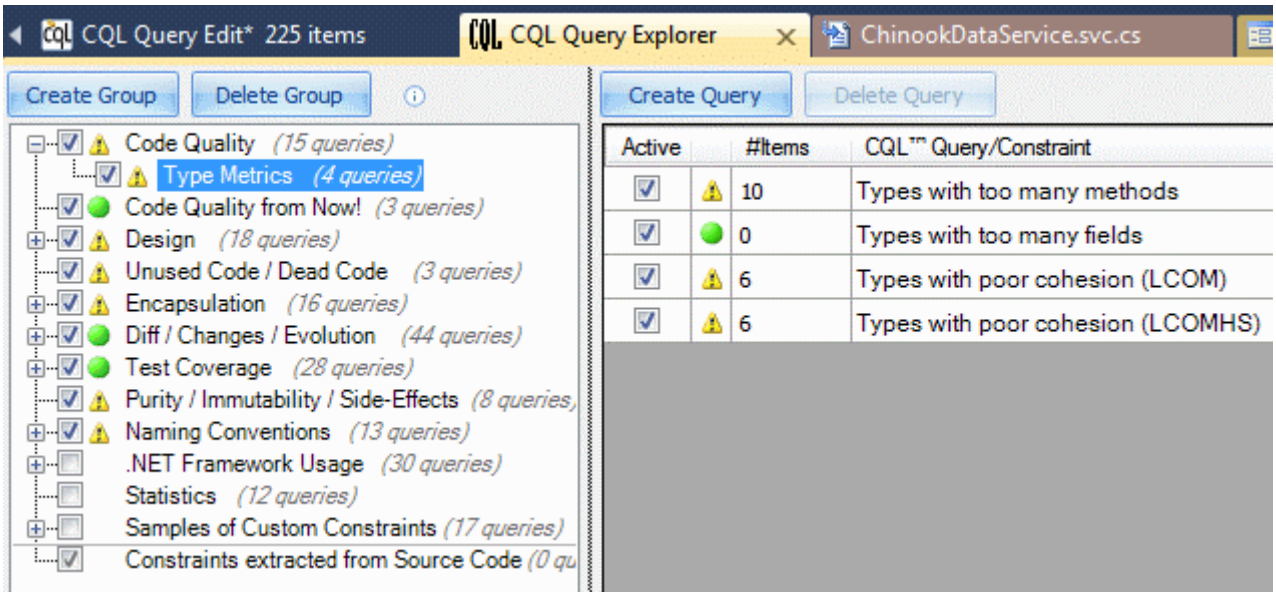
NDepend aracı hem kod kalitesini arttırmak için gerekli metric' lerin kontrolünü sağlıyor hem de az önceki erkan görüntüsünde olduğu gibi **CQL** ile bazı kriterleri sorgulayabilmemize olanak tanıyor. Tabi bu sadece şu ana kadar öğrenebildiğim iki güzel özelliği. Daha da fazlası var. Ancak basit bir test sürüşü ile yolumuza devam edebiliriz. Bu amaçla örnek olarak aşağıdaki ekran görüntüsünde yer alan **Solution** içeriğini göz önüne alabiliriz.



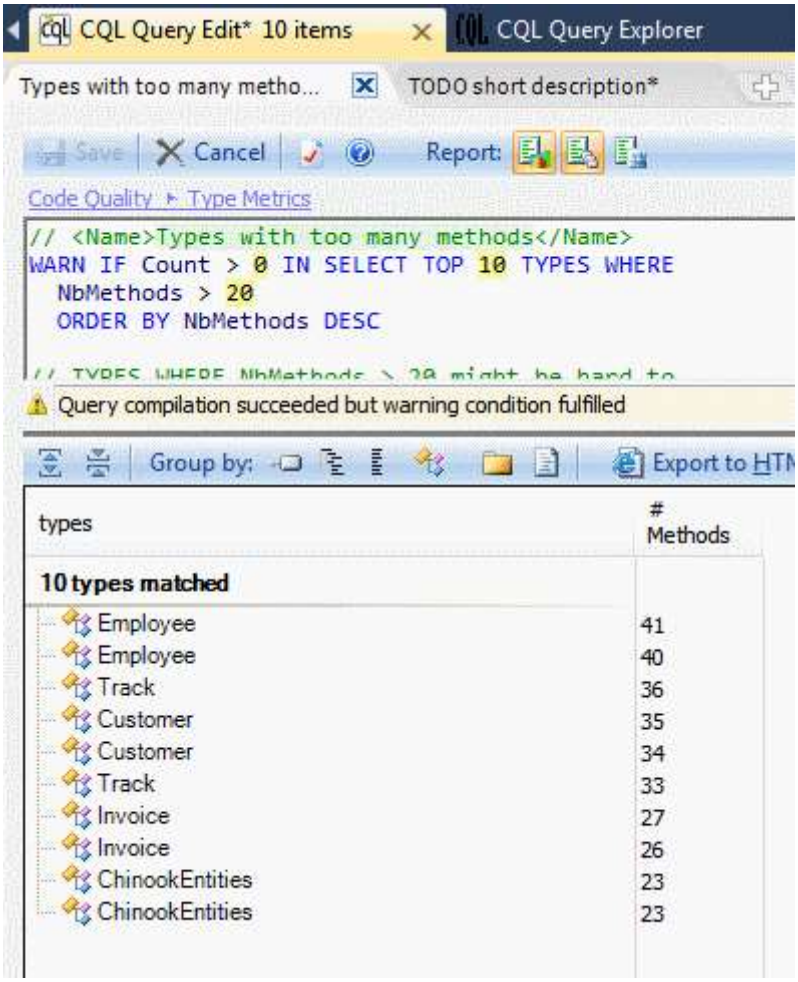
Sağ alt köşede sarı renkli olan daire mutlaka dikkatinizi çekmiştir. Aslında bu **NDepend** aracı ile gelen ve **Popup Menu** açan sihirli bir dairedir 🙂 Ki bastığınızda aşağıdaki ekran görüntüsüne benzer sonuçlar ile karşılaşabilirsiniz.



Dikkat edileceği üzere **18 kural ihlali(18 Rules Violated)** olduğu hemen göze çarpmaktadır **Rules Violated** kısmına tıkladığımızda ise aşağıdaki ara birim ile karşılaştığımızı görürüz.



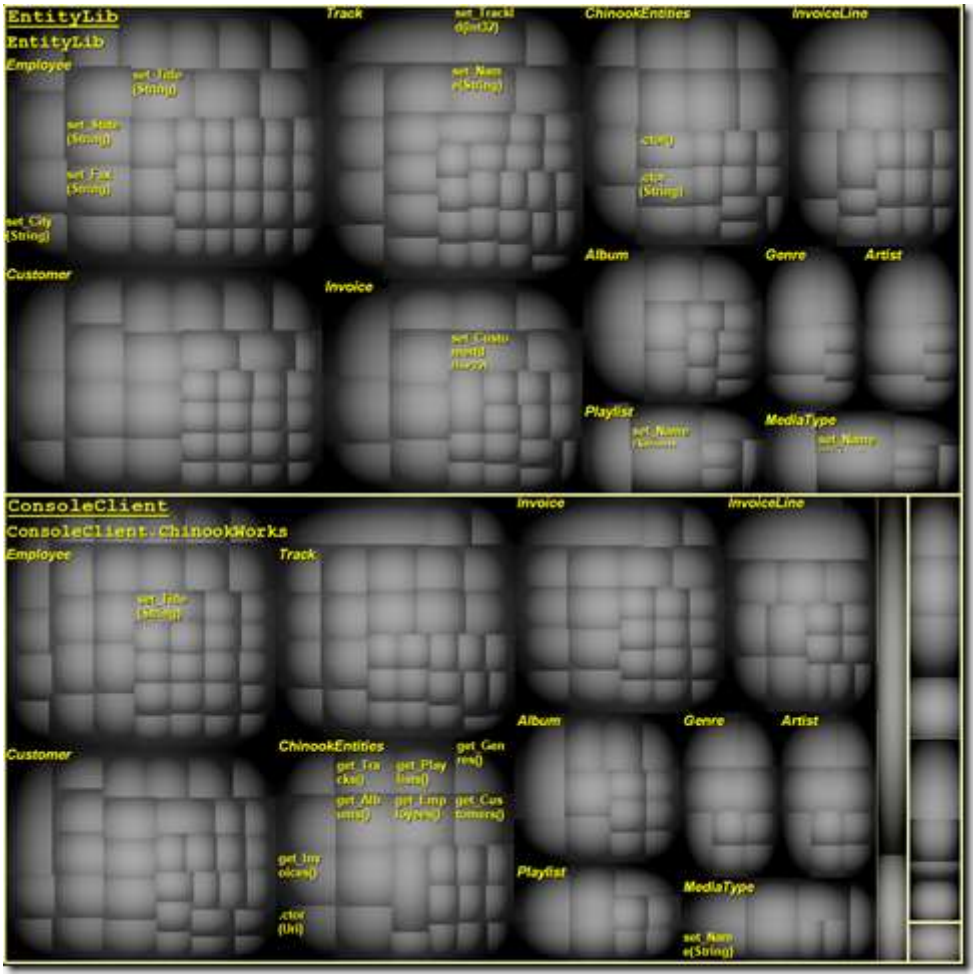
Bu ekran görüntüsünden de anlaşılacağı üzere bazı kalite kriterlerine ait analiz sonuçları üretilmiştir. örneğin kodun kalitesi açısından **Code Quality** grubu ve altındaki **Type Metrics** kısmına bir bakalım. Type Metrics sonuçları içerisinde örneğin **çok fazla sayıda metod içeren 10 öğe** olduğu belirtilmektedir. Eğer bu satıra çift tıkarsak aşağıdaki **CQL** sorgusunu ve sonuçlarını elde ederiz.



Volaaa!!! 😃 Süper. **CQL** sorgusunu okuduğumuzda **20 den fazla metod içeren 10 tip olduğunu** görmekteyiz (çok doğru. Bunların hepsi **Entity Framework** tarafndan üretilen **Entity** tipleri ve**Context** sınıfıdır 😬)Burada tanımlı olan kurala göre bir tipin 20den fazla metod içeriyor olması kodun yönetimi açısından zorlayıcıdır. Tabi bu bir **Warning** olarak karşımıza çıkmaktadır. İsterseniz bu kuralı hiçe sayabilir veya uymak için bir şeyler yapabilirsiniz. Tabi söz konusu kuralları esnetmeniz de mümkündür. Nitekim buradaki metric' ler atlına yenilerini ekleyebilir, yeni gruplar oluşturabilir ve var olan bazı metric' lerin sorgularını değiştirebiliriz.

Halen söz konusu aracı detaylı bir şekilde incelemekteyim. Kurulum sonrasın zaten **Visual Studio 2010, 2008** ve **2005** için de **AddIn** olarak kullanabileceğinizi görebilirsiniz. Şu sıralar özellikle kodun kaliteli olması noktasında son derece hassas bir dönemimdeyim 🙂

NDepend aracının bu yazıya sığmayacak kadar fazla sayıda özelliği var. Söz gelimi projenin analiz raporu basit bir **HTML** çıktısı olarak bizlere sunulmakta ama içerisinde inanılmaz derece de derin bilgiler yer almakta. örneğin aşağıdaki gibi görsel bağımlılık bilgisi içeren görüntüler. Resmen uygulamaların röntgeninin çekildiğini ifade edebiliriz 😃



Hani projeyi analiz etmeye, dökümantasyon çıkartmaya üşenenlerin yardımına koşan araçlardan birisi olduğunu açıkça ifade etmek isterim. Mesajım umuyorum ki gerekli yerlere gider **NDepend** uygulaması ile ilişkili detaylı bilgiye buradan ulaşabilirsiniz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

NedirTv?com Söyleşileri - Yazılım Dünyasının Merak Edilen Soruları (2010-09-23T09:39:00)

nedirtv?com,nedirtv?com söyleşileri,matematik mühendisliği,yazılım,yazılım teknolojileri,

 Merhaba arkadaşlar,

Pek çok arkadaşımızın bildiği üzere lisans eğitimimi **Yıldız Teknik üniversitesi Matematik Mühendisliği** bölümünde gerçekleştirdim. **1993** yılında girdiğim bölümde daha ilk yıllarda **Bilgisayar Programlama** dersinin büyüsüne kapıldım ve **"Bilgisayar Programıcısı"** olmaya karar verdim. **Delphi** ile başlayan merak, **Visual Basic, C++** gibi diller ile devam ederken **2002** yılında kendimi **.Net** ortamında buldum.

Aktif olarak **1999** yılında başladığım iş yaşantım olsun, üniversite yıllarında **Freelance** olarak gerçekleştirmeye çalıştığım projeler olsun pek çok değerli tecrübe edindim. Hiç olmassa nelerin yapılmaması gerektiğini öğreneceğim yaşam dersleri aldım

Hal böyle olunca biriken bilgi birikimi ve tecrübeyi, yazılımcı olmak isteyen arkadaşlarımıza aktarmanın da çok doğru olacağını düşündüm. Bu anlamda sevgili meslektaşım **Burak Mehmet Gürbüz** sağolsun bana çok ama çok yardımcı oldu.

NedirTv?com adına düzenlenen söyleşimizde bana aşağıda yer alan soruları yönelterek genç yazılımcı adaylarımız (hatta yaşlı yazılımcı adaylarımız)için yararlı bir içeriğin oluşmasında en önemli katkıyı sağladı. Kendisine hazırladığı sorular ve söyleşiye ön ayak olması nedeniyle çok teşekkür ediyorum. Umarım yazılımcı adaylarımızın yararlanabileceği bir içerik oluşturmayı başarmışızdır. Ve işte söyleşi sorularımız;

Söyleşi Soruları

1. Yazılımcı olmak için kişinin kendisindeki vasıfları ne yönde olmalıdır?(Hamur versiyonu)
2. Bu süreçte izlenecek yol nasıl çizilmeli ve nereden başlamalı?
3. Nasıl bir çalışma stili benimsenmeli. Buna bağlı olarak nasıl bir çalışma ortamı olmalı.
4. Bu süreçte hakkında bilgi sahibi olunması gereken kişiler ve onların ilginç yaşamları.(feyz almak için)

5. Neden her insanın aklına başlanması gereken yer olarak nesne yönelimli programlama(Object Oriented Programming) geliyor.Eğer yanlışsa ne yapılmalı? Yada temel olarak hangi dil ile başlanmalı?
6. Kaynak seçimi için bilinmesi gerekenler ve teknolojinin gelişime faydalı olması için bilinçli kullanımı.
7. Yazılım ile ilgili şirketin sahip olması gereken özellikleri nedir?
8. Nasıl bir şirkette çalışılmalı?
9. üniversite hayatı boyunca çalışma imkanı bulamayan fakat kendini geliştirmek isteyen kişiler nasıl proje üretebilir?Bunun için bir çözüm varmıdır?
10. öğrendiklerimizin akılda kalıcılığını arttırmak için blog faydalı mı? Bunu daha yararlı hale nasıl getirebiliriz?
11. öğrendiklerimize zenginlik katmak için düşünce dünyamızı ve bakış açımızı geliştirmek için neler yapmalıyız?
12. Kaliteli kod yazmak nedir? Tavsiye ettiğiniz kitap çerçevesinde bunu açıklayabilirmisiniz?
13. İngilizce bilmemiz gerektiği söyleniyor…Bunu dikkate almamız bize nasıl bir fayda sağlayabilir?(İş hayatı dışında yazılım için)
14. Bu işte yaş ne kadar önemli?
15. Kariyer basamaklarında adım adım ilerlerken hangi yaşta hangi mevkiyi hedef olarak belirlemeliyiz?
16. ülkemizde yaş çok önemli bir faktör.Erkekler için askerlik buna dahil. Bunu açıklarken karşılaştığınız örneklerle beraber anlatırmısınız?
17. Yazılımcı olmak için kurs ne kadar gerekli? Eğer gerekli ise nasıl bir birikimle kursa gidilmeli yada zorunlu değilse neden değil?
18. Zaman yönetimi ve yazılım ile ilgili tecrübelerinize dayanarak son olarak tavsiyeleriniz nelerdir?

Söyleşiyi indirmek için tıklayın

Bu arada söyleşi de bahsettiğimiz **OPML** içeriğini feedreader.opml (21,15 kb) bağlantısından tedarik edebilirsiniz.

Microsoft Gelişim Atölyesi Teknoloji Kampında Buluşalım (2010-09-21T18:10:00)

visual studio 2010,visual studio 2010 ultimate,c#,microsoft,gelişim atölyesi,

Merhaba Arkadaşlar,

5,6 Ekim 2010 tarihlerinde **Microsoft Gelişim Atölyesi Teknoloji Kampı** düzenleniyor olacak. Bir aksilik olmassa bende **Visual Studio 2010 ile**

Etkin ürün Geliştirme konulu bir sunum yapıyor olacağım. Sunum için bana ayrılan **45 dakikalık** zaman diliminde aşağıdaki konulardan bahsetmeyi ve ilgili örnekler yapmayı planlıyorum.

Arhitecture Features
- Dependency Graph(Assembly, Class...)
- Sequence Diagram
- Layer Diagram
- Built-In Layer Diagrams(Microsoft Architecture Guide)

IDE Features
- Multi Framework Support(Multi Targeting)
- Extension Manager
- WPF Based
- Faster Add Reference
- Multi Monitor Support

Code Improvements
- Consume-First Development(TDD)
- Code Intellisense
- Call Hierarchy
- Advanced Search with Navigate To
- Highlight References

Dikkat edeceğiniz üzere **Visual Studio 2010** özelliklerini bir **Developer** gözüyle ele

almaya gayret edeceğiz

Etkinliklerde **.NET Framework, Sharepoint 2010 Development, Bulut Bilişim, SQL 2008, Office 2010, Exchange Server 2010, Windows Phone** ve daha pek çok konuda sunumların yapılması planlanmakta. Eğitim takvimi ve detay bilgileri ise önümüzdeki günlerde yine **Microsoft Gelişim Atölyesi** üzerinden yayınlanıyor olacak.Teknoloji ile dop

dolu iki gün bizleri bekliyor Orada görüşmek dileğiyle hepinize mutlu günler dilerim.

LINQ to SQL - EF 4.0 (Aradaki 9 Farkı Bulun) (2010-09-21T03:07:00)

linq to sql,ado.net entity framework 4.0,ef 4.0,entity framework,

Merhaba arkadaşlar,

Evet çok doğru. Hiç bu kadar kısa ve öz yazmamıştım daha önceden. Ama zaman zaman bu kadar kısa yazıp çok fazla şey ifade edilebileceğine de inanmaktayım Hani ilk bakışta herşeyin şak diye kafanızda yer ettiği tablolar olur ya…Bu blog girdisinde de benzer bir resmi göreceğinizi düşünüyorum. Aslında olay bundan çok çok uzun zaman önce gelen bir soru üzerine meydana geldi.

"Entity Framework ile LINQ to SQL arasındaki temel ve belirgin farklılıklar nelerdir? Hangi durumlarda hangisi tercih edilmelidir?"

Soruyu araştırmak ve cevap vermek için aradan baya bir zaman geçti ancak yaptığım incelemeler sonucunda aşağıdaki temel ayrımları tespit ettiğimi ifade edebilirim. Hatta ortak oldukları noktaları da bu tabloda görebilirsiniz.

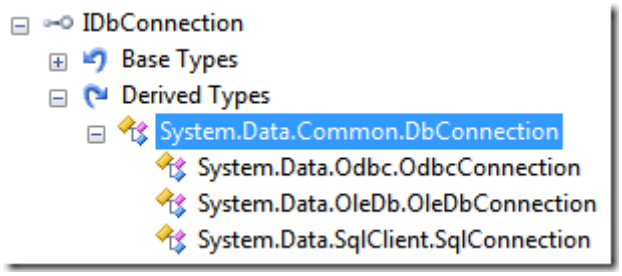
Kavram	**EF 4.0**	**LINQ to SQL**
Sql Server Harici Veritabanı Desteği	Var	Yok gibi
Doğrudan veritabanı bağlantısı	Yok	Var
çoklu tablodan kalıtım(Multiple Table Inheritance)	Var	Yok
Birden fazla tablodan tek bir Entity üretmek	Var	Yok
Conceptual Schema Definition Language(CSDL)	Var	Yok
Storage Schema Definition Language(SSDL)	Var	Yok
Mapping Schema Language(MSL)	Var	Yok
Lazy Loading	Var	Var
Stored Procedures	Var	Var

Tabi bazı noktalarda çok keskin ayrımlar olduğunu ifade edebiliriz. Söz gelimi **LINQ to SQL** takımı doğrudan **.Net'** in **SQL** yönetim bileşenlerini kullanmayı tercih etmektedir. Sanıyorum ki **C#**takımının üyelerinin **LINQ to SQL**' i geliştirmiş olmasının bunda büyük rolü vardır öyleki **EF** takımı doğrudan veritabanı erişimi veya **SQL** nesnelerini**(SqlConnection, SqlCommand vb…)** kullanmak yerine **Conceptual** bir modeli baz alarak depolama ve kütüphane nesnelerinin eşleştirilmesinde **XML** bazlı daha esnek bir yapıyı tercih etmiştir. Dilerseniz **LINQ to SQL'** in arka tarafına kısaca bir bakalım ve **SQL** nesne **kullanımını(ya da bağımlılığını)** görmeye çalışalım.

örnek olarak **Northwind** veritabanını kullandığımız bir **LINQ to SQL** sınıf yapsında üretilen **NorthwindDataContext** tipinin içeriğinde aşağıdaki **yapıcı metodlar(Constructors)** hemen göze çarpacaktır.

```
public partial class NorthwindDataContext : System.Data.Linq.DataContext
{

  private static System.Data.Linq.Mapping.MappingSource mappingSource = new AttributeMappingSource();

Extensibility Method Definitions

  public NorthwindDataContext() :
      base(global::ConsoleApplication9.Properties.Settings.Default.NorthwindConnectionString, mappingSource)
  {
    OnCreated();
  }

  public NorthwindDataContext(string connection) :
      base(connection, mappingSource)
  {
    OnCreated();
  }

  public NorthwindDataContext(System.Data.IDbConnection connection) :
      base(connection, mappingSource)
  {
    OnCreated();
  }

  public NorthwindDataContext(string connection, System.Data.Linq.Mapping.MappingSource mappingSource) :
      base(connection, mappingSource)
  {
    OnCreated();
  }

  public NorthwindDataContext(System.Data.IDbConnection connection, System.Data.Linq.Mapping.MappingSource mappingSource) :
      base(connection, mappingSource)
```

Dikkat edileceği üzere **IDbConnection interface** tipini kullanan iki yapıcı metod versiyonu söz konusudur. Eğer **IDBConnection**' dan türeyen **.Net** tiplerine bakarsak aşağıdaki sonuçlar ile karşılaşırız.



Dikkat edeceğiniz üzere **OdbcConnection**, **OleDbConnection** ve **SqlConnection** tipleri söz konusudur. Buradan **DataContext** türevli olan **NorthwindDataContext** tipinin üretimi sırasında mutlaka**SQL** tarafına bir bağımlılığın olduğunu en azından **.Net** içerisinde var olan **provider** yapısının ele alındığını görebiliriz.

Bu ve diğer farklılıkları aslında ilerleyen yazılarımızda incelemeye çalışıyor olacağım. Siz şimdilik yukarıdaki klavuzu gözünüze kestirin. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Microsoft Teknoloji Günleri Akşam Sınıfı - WCF Eco System Eğitimi Tamamlandı (2010-09-17T18:47:00)

birlikte gelistir,bizspark,microsoft biz spark,microsoft,microsoft teknoloji günleri,windows communication foundation,wcf eco system,wcf data services,wcf ria services,wcf webhttp services,wcf service,wcf 4.0,web http services,windows server app fabric,

Güncelleme : Video kayıtlarını SkyDrive üzerinden indirebilirsiniz.

Merhaba Arkadaşlar,

Ramazan ayı dolayısıyla ertelediğimiz **WCF Eco System** eğitimimizi geçtiğimiz **salı günü(14.Eylül.2010)** başarılı bir şekilde tamamladık. öncelikli olarak eğitime katılan bütün arkadaşlarımıza, **Microsoft** taraf ında yardımlarını esirgemeyen **Buket Şerefli** ve **MVP liderimiz Baransel Doğan**' a, ayrıca seminer boyunca görüntü alan **Mustafa Demirhan** ile fotoğraf işlerini üstelenen **Tuba çebi** arkadaşlarımıza teşekkür etmek istiyorum.

Eğitimimizde **WCF Eco System'** in önemli parçalarından olan **Data Services, WebHttp Services** ve **RIA Services** konularına değinmeye çalıştık. Kısa bir konsept girişten sonra, konu ile ilişkili örnek demolar geliştirdik. Söz konusu örneklere ait**Solution** dosyasını aşağıda bulabilirsiniz.

Bildiğiniz üzere eğitimlerimize katılamayan arkadaşlarımız için video kayıtlarını almaya çalışıyoruz. Bu seferki etkinliğimizde akıllılık edip(Nihayet 😀) ekran görüntüsünü de aldık. **Camtasia Studio** programı ile aldığımız görüntüleri **Render** etme işlemi ise ertesi gün tamamlandı. ~~Şu anda NedirTv?com sitemize **upload** edilmeyi bekleyen **250 Mb'** lık bir eğitim videomuz var. Tabi bunu bir anda **Upload** etme şansımız malesef yok. O yüzden parçalara bölüp atacağız. Sözün özü, eğitim videosunun sitemize eklenmesi biraz zaman alacak gibi görünüyor 🤔 Ancak **Pazartesi günü(20.Eylül.2010)** yapacağımız **Windows Server AppFabric eğitimine** gelen arkadaşlarımızdan isteyenlere ilgili videoyu hemen kopyalama yöntemi ile vereceğimi de buradan belirtmek isterim.~~ Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Sunum : WCF Eco System - Introduction - Innova.pptx (295,85 kb)

Solution : MTGASD4.rar (4,30 mb)

Diamond Problem, C# ve Multiple Inheritance (2010-09-15T18:28:00)

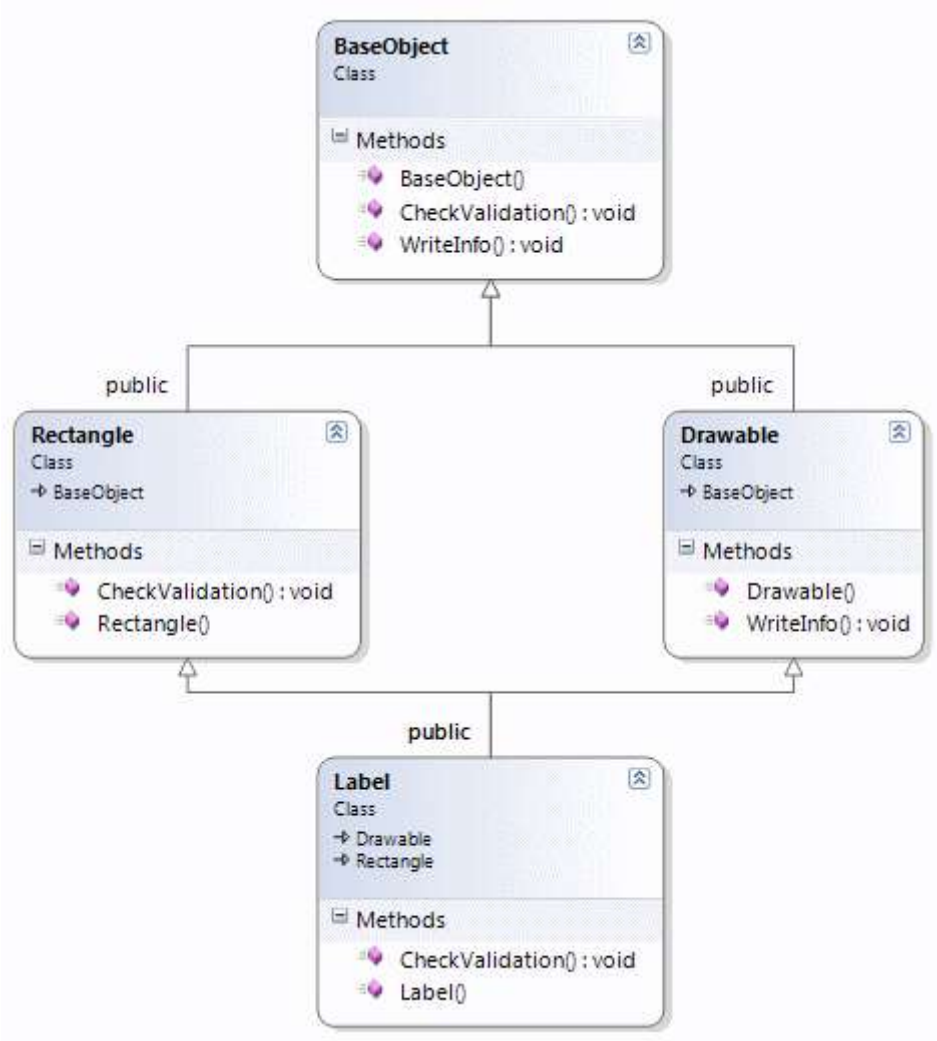
c#,c++,inheritance,multiple inheritance,

Merhaba Arkadaşlar,

Yandaki resimde mükemmel ölçülerde bir elmas taşını görmektesiniz. Sanıyorum ki içerisinde altın orana rastlamak bile mümkündür. Elmas karatına göre derece değerli bir taş olduğu kadar, elde edilmesi de zahmetli ve zordur. Hatta özellikle **Afrika**' nın elmas yönünden zengin olan bazı ülkelerinde bu taş yüzünden pek çok kişi hayatını kaybetmiştir/kaybetmektedir. Ki bu konu günümüz sinemasına da malzeme olmayı başarmıştır.

Elbette bizim bu günkü konumuzun elmaslar ve üzerinde dönen dolaplar ile pek bir alakası bulunmamaktadır. Aslında bu günkü yazımızda **C#**programlama dilinde yasaklanmış olan ama örneğin **C++** ile ele alabileceğiniz **çoklu kalıtım(Multiple Inheritance)** konusuna farklı bir açıdan bakmaya çalışıyor olacağız.

Esasında **C#** programlama dili ile ilişkili kaynaklara baktığımızda, kalıtım ile ilişkili temel kurallardan birisi n sayıda sınıf kullanılarak çoklu kalıtımın yasaklanmış olduğudur. Ancak bunun sebebi veya nedenleri hakkında elle tutulur çok fazla bilgi bulunmaz. Bulunu ama pek anlaşılmaz 😕 Bu nedenle durumu buna izin veren bir dil ile değerlendirmekte yarar vardır. Olaya **C++** tarafından baktığımızda, **Diamond Problem** adı verilen bir sorunsalın, böyle bir yasağa neden olduğunu da ifade edebiliriz. Peki **Diamond Problemi** nedir? Dilerseniz öncelikle bu vakayı ele almaya çalışarak işe başlayalım. Bu amaçla basit bir **Win32 Console Application** projesi geliştiriyor olacağız. Söz konusu proje **C++** ile geliştirilmiş olup aşağıdaki kod içeriğine sahiptir.

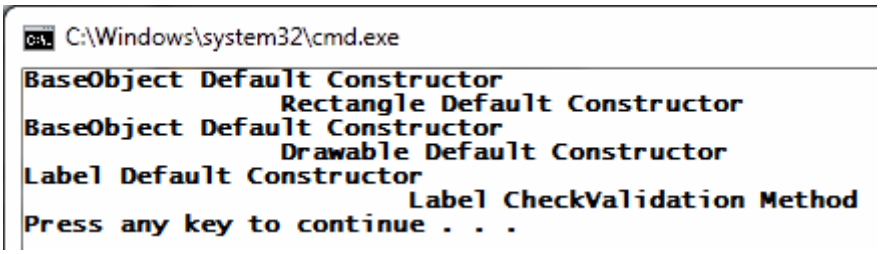



```
#include "stdafx.h"
#include <string>
#include <stdio.h>

class BaseObject
{
public:
   BaseObject(void)
   {
      printf("BaseObject Default Constructor\n");
   }
public:
   virtual void WriteInfo()
   {
      printf("\tBaseObject WriteInfo Method\n");
   }
   virtual void CheckValidation()
   {
```

```
        printf("\tBaseObject CheckValidation Method\n");
    }
};

class Drawable
    :public BaseObject
    //: virtual public BaseObject
{
public:
    Drawable(void)
    {
        printf("\t\tDrawable Default Constructor\n");
    }
    void WriteInfo()
    {
        printf("\t\tDrawable WriteInfo Method\n");
    }
};

class Rectangle
    :public BaseObject
    //:virtual public BaseObject
{
public:
    Rectangle(void)
    {
        printf("\t\tRectangle Default Constructor\n");
    }
    void CheckValidation()
    {
        printf("\t\tRectangle CheckValidation Method\n");
    }
};

class Label
    :public Rectangle,
    public Drawable
{
public:
    Label(void)
    {
        printf("Label Default Constructor\n");
    }
    void CheckValidation()
    {
```

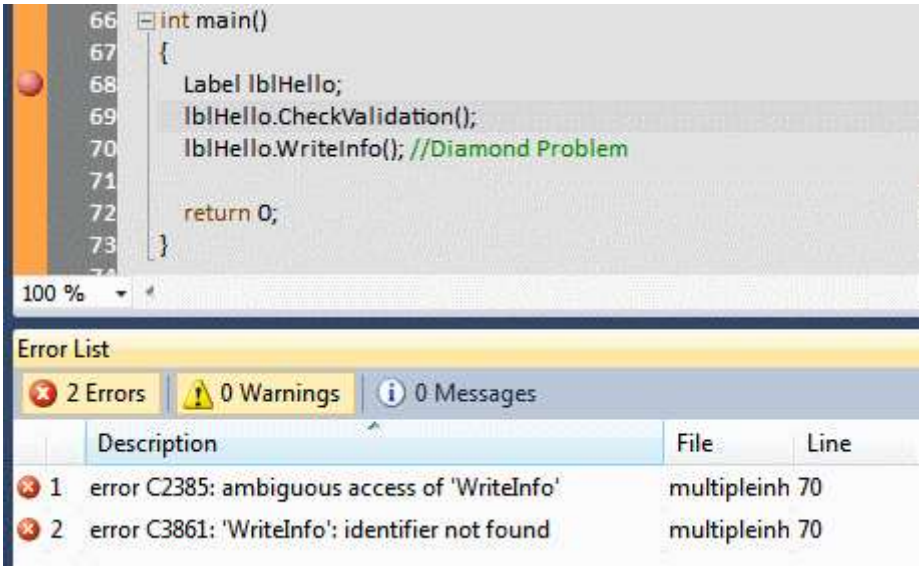
```
        printf("\t\t\tLabel CheckValidation Method\n");
    }
};

int main()
{
    Label lblHello;
    lblHello.CheckValidation();
    // lblHello.WriteInfo(); //Diamond Problem

    return 0;
}
```

Şekilden de görüleceği üzere **Label** isimli sınıf iki sınıftan birden türetilmektedir. Bunlar **Rectangle** ve **Drawable** isimli sınıflardır. Söz konusu iki sınıfta tekil bir kalıtım ile**BaseObject** sınıfından türemektedir. **BaseObject** sınıfı içerisinde **virtual** olarak tanımlanmış iki fonksiyon yer almaktadır.
Bunlardan **CheckValidation** metodu, **Rectangle** tipi içerisinde**ezilmiştir(override)**. Diğer **WriteInfo** metodu da **Drawable** sınıfı içerisinde ezilmiştir. Benzer şekilde **Label** isimli sınıfta **CheckValidation** metodunu ezmektedir. Kodu bu şekilde çalıştırdığımızda aşağıdaki ekran görüntüsünde yer alan sonuçları alırız.

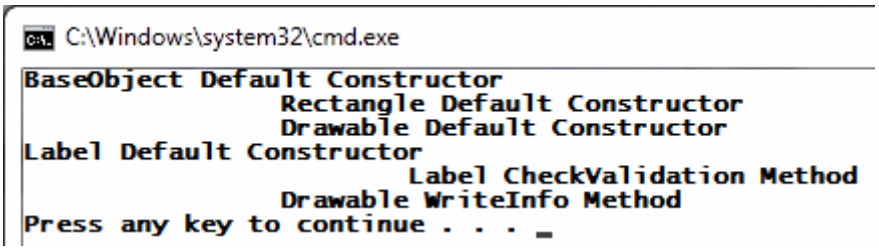


Sonuçlar gayet normaldir. **Label** tipine ait bir nesnenin kullanılması, türetildiği **Drawable** ve **Rectangle** sınıfılarının örneklenmesine neden olmaktadır. Bu sınıflarda hiyerarşiye göre**BaseObject** tipinden türedikleri için her birisi adına birer **BaseObject** örneği üretilmektedir. **Label** örneği üzerinden çağırılan **CheckValidation** metodu, **override** edilmiş olması nedeniyle üst tip yerine **Label** içerisinden yürütülmüştür.

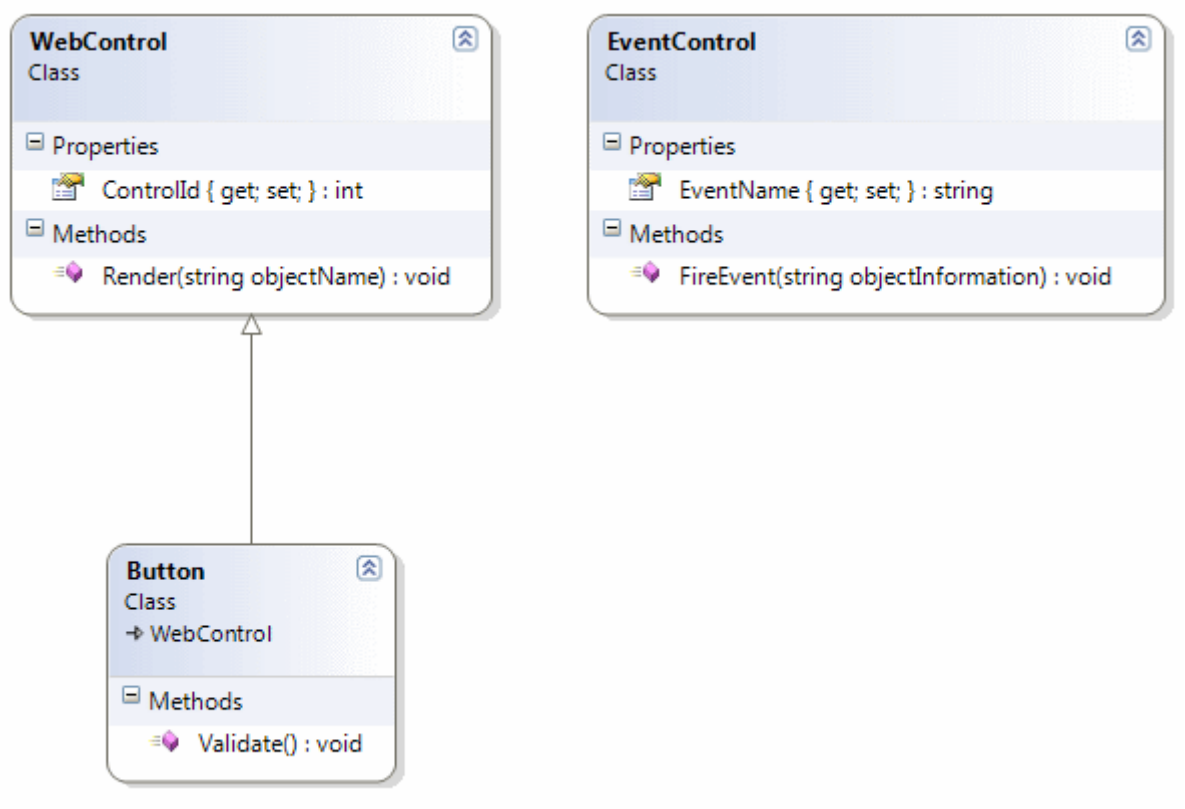
Sıkıntı ise **main** metodu içerisinde **bold** olarak işaretlemiş olduğumuz satırı etkinleştirdiğimizde meydana gelmektedir. Burada **Label** tipinden **lblHello** değişkeni kullanılarak **WriteInfo**metodunun çağırıldığı görülmektedir. Ne varki **Label** sınıfı söz konusu metodu ezmemiştir. **WriteInfo** metodu ezilmemekle birlikte türediği iki üst sınıf içerisinde de ezilmiştir. Bu durumda kod hangi üst **WriteInfo** metodunu çağıracağına karar verememektedir. Dolayısıyla derleme zamanının kafası karışmıştır 😬 Buna göre yorum satırı açıldığında aşağıdaki hata mesajının alındığı görülecektir.



Tabi bu durumun **C++** tarafında çözümü vardır. **Virtual Inheritance** kullanılarak bu sorun çözülebilir. Yani kodda **virtual public** ile başlayan yorum satırlarını türetme için kullandığımızda **Label** nesne örneği üzerinde
çağırılan **WriteInfo** metodunun, **Drawable** tipi içerisinde ezilmiş olan versiyonuna gittiği görülecektir ki buna göre çıktı aşağıdaki gibi olacaktır.



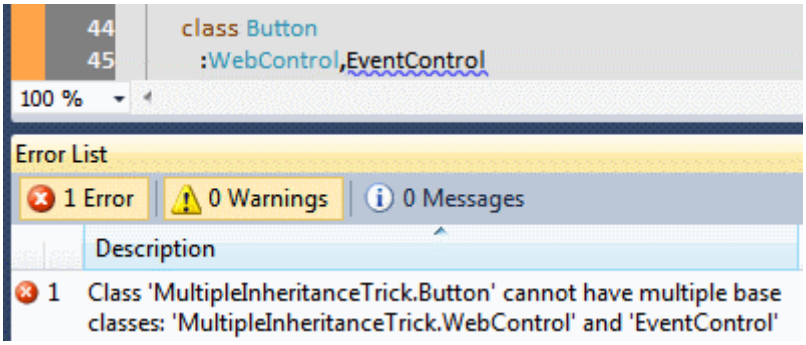
Yasağın bir nedenini öğrenmiş olduk. Tabi bu örnekteki gibi 4 sınıfın işin içerisine girdiği senaryolarda bu pek bir sorun olarak görülmeyebilir. Ancak sınıf ve türetme dallarının arttığı, üye sayısının fazlalaştığı hallerde **Diamond Problem** önemli bir sıkıntı haline gelecektir. Şimdi tekrar **Managed** ortama dönelim ve **C#** tarafında aşağıdaki gibi geliştirme yapmış olduğumuzu varsayalım.




```
using System;

namespace MultipleInheritanceTrick
{
   class WebControl
   {
      public int ControlId { get; set; }

      public void Render(string objectName)
      {
         Console.WriteLine("WriteInfo {0}", objectName);
      }
   }
   class EventControl
   {
      public string EventName { get; set; }

      public void FireEvent(string objectInformation)
      {
         Console.WriteLine("ReadInfo {0}", objectInformation);
      }
   }
```

```
    class Button
      :WebControl
    {
       public void Validate()
       {
          Console.WriteLine("Derived.Validate");
       }
    }

    class Program
    {
       static void Main(string[] args)
       {
          Button helloBtn=new Button();
          //helloBtn.FireEvent("Common Informations");
          helloBtn.Render("Person");
          helloBtn.Validate();
       }
    }
}
```

Burada **WebControl** tipinden türeyen **Button** isimli bir sınıf olduğunu görmekteyiz. Aslında bu sınıfın **EventControl** sınıfından da türemesi iyi olabilirdi. Nitekim bu durumda **Button** sınıfı üzerinden veya içerisinden **WebControl** sınıfındaki **Render** ve **EventControl** sınıfındaki **FireEvent** metodlarına çağrıda bulunabilirdik. Ancak bildiğimiz üzere **Button** sınıfını hem **WebControl** hem de **EventControl** tipinden türetmeye çalışmamız aşağıdaki gibi sonuçlanacaktır.

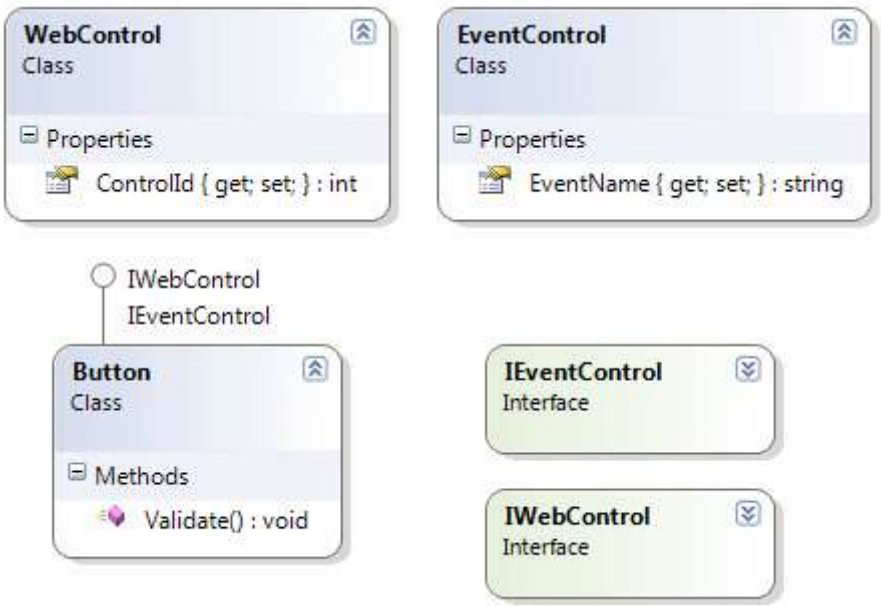


Peki kurallar biraz olsun esnetilebilir mi? 😉

Yani bir şekilde **EventControl** ve **WebControl** tipi üzerinden izin verilen fonksiyonların **Button** tipine ait bir nesne örneği üzerinden çağırılması sağlanabilir mi? Bu konuda Internet üzerinden araştırma yaptığınız takdirde genellikle **LINQ** tarafında çok önemli bir yere de sahip olan **Extension Method**'ların göz önüne alındığını fark edebilirsiniz. örneğin **Miguel A. Castro** bir blog yazısında bu durumu ele almıştır.

Bilindiği üzere **Extension Method**' lar sayesinde tiplerin fonksiyonel olarak genişletilmesi mümkündür. Hatta burada **sealed** gibi işaretlenmiş veya kendi kontrolümüzde olmayan tiplerin ya da türetilemeyen tiplerin genişletilebilirliği de söz konusudur. Diğer yandan **C#** tarafında her ne kadar sınıf seviyesinde çoklu kalıtıma izin verilmese de, bu bir sınıfı birden fazla **Interface**tipinden türetemeyeceğimiz anlamına da gelmemelidir.

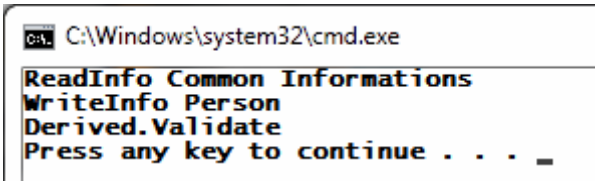
Zaten **C#** tarafında çoklu kalıtım için **interface** mantığının kullanılması önerilmektedir. Aslında senaryomuzda yer alan sınıfları kendi geliştirdiğimiz tipler olarak düşünerek hareket ettiğimizde, içlerindeki bazı metodları **extension method** olarak dışarıya alıp belirli **interface** tiplerine uygulatmamız sorunun anahtar çözümü olmaktadır. Durumu daha iyi anlayabilmek için gelin uygulama kodumuzu aşağıdaki hale getirelim.




```
using System;

namespace MultipleInheritanceTrick
{
   interface IWebControl
   {
   }
   interface IEventControl
   {
   }
   class WebControl
   {
      public int ControlId { get; set; }
   }
   class EventControl
   {
      public string EventName { get; set; }
```

```
    }
    static class WebControlExtensions
    {
        public static void Render(this IWebControl wCtrl, string objectName)
        {
            Console.WriteLine("WriteInfo {0}", objectName);
        }
    }
    static class EventControlExtensions
    {
        public static void FireEvent(this IEventControl eCtrl, string objectInformation)
        {
            Console.WriteLine("ReadInfo {0}", objectInformation);
        }
    }

    class Button
        :IWebControl,IEventControl
    {
        public void Validate()
        {
            Console.WriteLine("Derived.Validate");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Button helloBtn=new Button();
            helloBtn.FireEvent("Common Informations");
            helloBtn.Render("Person");
            helloBtn.Validate();
        }
    }
}
```

Dikkat edileceği üzere **Render** ve **FireEvent** isimli metodlar bulundukları sınıflardan çıkartılarak **Extension Method** haline getirilmiştir. **Render** metodu **IWebControl** arayüzüne uygulanırken, **FireEvent** metodu da **IEventControl** arayüzüne uygulanmaktadır. Tesadüfe bakın ki, **Button** tipimizde hem **IWebControl** hem de **IEventControl** arayüzlerinden türemektedir Bu sebepten dolayı her iki arayüz için de geçerli olan genişletme metodlarını çağırabilecek şekilde tasarlanmış hale geldiğini söyleyebiliriz. İşte size çakma çoklu kalıtım Bu durumda uygulama kodunun çalışma zamanı çıktısı da aşağıdaki gibi olacaktır.



Tabi bu tekniğin de bazı eksik yönleri ve handikapları vardır. Herşeyden önce **extension method** yazılması gerekmiştir ve söz konusu teknik bu sebepten sadece metodlara uygulanabilmektedir. Yani **özellikler(Properties)** için geçerli değildir. Tabi ilerleyen **C#** sürümlerinde belki de **Extension Property** kavramı ile karışaşabiliriz, bilemiyorum

Okuyan arkadaşlarımız kadar benim içinde enteresan bir yazı olduğunu ifade etmek isterim. Ancak bu yazımızda en azından **C++** tarafında bilinen **Diamond** sorunsalına bakarak **C#** tarafına getirilen çoklu kalıtım yasağı için bir sebep bulduğumuzu düşünüyorum. Diğer yandan **Extension Method** ve **Interface**' leri kullanarak çoklu kalıtımın farklı bir şekilde nasıl gerçekleştirilebileceğini de görmüş olduk. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

DiamondProblem.rar (2,00 mb)

NedirTv?com Söyleşileri - .Net Framework 4.0 ile Gelen Yenilikler Bölüm 2 (2010-09-14T22:16:00)

.net framework 4.0,visual studio 2010,managed extensibility framework,asp.net 4.0,asp.net mvc,nedirtv?com,nedirtv?com söyleşileri,

Merhaba Arkadaşlar,

Yeni bir NedirTv?com söyleşimiz ile karşınızdayız. Hatırlayacağınız üzere bir önceki söyleşimizde **.Net Framework 4.0** ile birlikte gelen yeniliklere değinmeye başlamıştık. Söz konusu yenilikleri ikinci bölümü ile incelemeye devam ediyoruz. Bu sefer,

- Managed Extensibility Framework,
- Asp.Net 4.0,
- Asp.Net Model View Controller(MVC),
- Visual Studio 2010 IDE,

konularına değinmeye çalışıyoruz. Tabi tüm konular **.Net Framework 4.0** ile doğrudan ilintili gözükmese de laf lafı açınca bu hale geldiğini ifade etmek isterim

Gördüğünüz üzere halen daha **Parallel Programming, Entity Framework 4.0, WCF 4.0, WF 4.0** gibi konulara değinebilmiş değiliz. Ancak bu konuları da 3ncü bölümümüzde ele alıyor olacağız. Şimdiden keyifli dakikalar dilerim

Söyleşiyi Dinlemek için Tıklayın

NedirTv?Com Söyleşileri - 3 - .Net Framework 4.0 ile Gelen Yenilikler Bölüm 1 (2010-09-07T01:00:00)

nedirtv?com,nedirtv?com söyleşileri,.net framework 4.0,code contracts,side by side execution,clr 4.0,dlr,dynamic language runtime,

Merhaba Arkadaşlar,

NedirTv?com söyleşilerimizin 3ncü bölümünde oldukça geniş bir konuyu ele almaya çalışıyoruz. **.Net Framework 4.0 ile Gelen Yenilikler**

Konunun geniş olması ve üzerinde değil saatler günlerce konuşulabilmesi söz konusu Bu nedenle söyleşi ekibi olarak çok teknik detaya girmeden en az iki bölüm halinde gelen yenilikleri ele almaya çalıştık ve çalışıyor olacağız.

İlk bölümümüzde, sektöreden gelen ve aktif projelerinde **.Net 4.0 Framework'** ü kullanan **Arda çetinkaya** arkadaşımız da(soldan ikinci) bizimle birlikte oldu ve konu hakkındaki tecrübelerini bizlerle paylaştı. Bu ilk bölümümüzde,

- **CLR(Common Language Runtime) 4.0** tarafındaki yenililerini,
- **Exception Handling** genişletmelerini,
- Yeni **Performance Counter**' ları,
- **Side By Side Execution** ve **Code Contracts** gibi kavramları,
- **DLR(Dynamic Language Runtime)** konusunu,

ve diğer başka yenilikleri ele almaya çalıştık.

Umarım sizi gelen yenilikler konusunda bir nebze de olsa bilgilendirebiliyoruzdur? Bir sonraki bölümümüzde görüşünceye dek hepnize mutlu günler dilerim.

Söyleşi için Tıklayın

AttachedToParent Hakkında Detaylar (2010-09-02T20:30:00)

tpl,task parallel library,parallel programming,parallel computing,

 Merhaba Arkadaşlar,

Malum **"her yiğidin farklı bir yoğurt yiğiş tarzı vardır"** derler. Genellikle programlama dilleri veya **.Net Framework** gibi yapılarda da bir sonuca ulaşmak için birden fazla ve farklı yol söz konusu olabilir. Böyle bir durumun oluşmasına neden olan etkenlerin başında, çevresel ortam parametrelerinin farklılaşmasının geldiğini ifade edebiliriz.

çok basit bir kaç örnek vererek olayı kafamızda daha net bir şekilde canlandırmaya çalışalım. Bir koleksiyon içerisindeki öğeleri for veya foreach döngüleri ile dolaşabiliriz. Ya da örneğin bir veri tablosundan veriyi çekmek için, DataTable bazlı bir tekniği veya DataReader bazlı bir yöntemi ele alabiliriz. Ancak öyle vakalar söz konusudur ki, aynı amaç için ele alınabilecek veya değerlendirilebilecek yolların sayısı çok fazladır. Bu fazlalık bir süre sonra karar vermeyi zorlaştırır ve işin içinden çıkılmaz bir duruma düşülebilir.

Söz gelimi **paralel programlama** konusu ile ilgili olarak bir süredir incelediğimiz **Parent-Child Task** ilişkilerini göz önüne alalım. Daha önceki iki yazımızda(Parent-Child Tasks Kavramı, Parent-Child Task Exception Durumlar) sürekli olarak **AttachedToParent** metodunun belirli bir kullanımını ele aldık. Oysa ki, **Child Task**örneklerinin **Parent Task** örneklerinin yaşam döngülerine eklenmelerinde izlenebilecek birden fazla yol bulunmaktadır. Buna göre **Parent Task** örneğine dahil olmak için aşağıdaki tekniklerden herhangibirisinden yararlanılabilir.

- **Task** nesnesine ait **yapıcı metod(Constructor)** içerisinde **TaskCreationOptions.AttachedToParent enum** sabiti bildirimi yapılarak

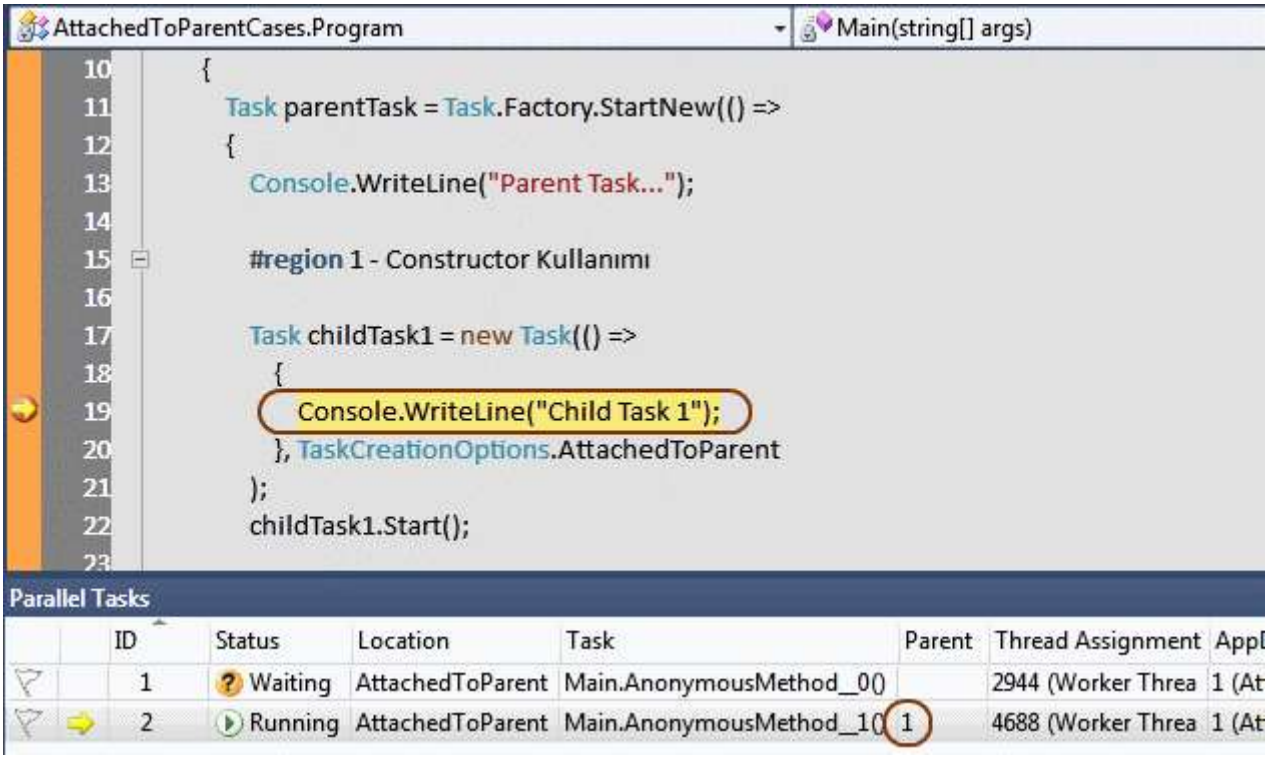
- **Task** sınıfının **static** **StartNew** metodunda **TaskCreationOptions.AttachedToParent enum** sabiti bildirimi yapılarak
- **Task** nesne örneği üzerinden çağırılabilen **ContinueWith** metoduna **TaskContinuationOptions.AttachedToParent enum** sabiti parametresini geçirerek
- **Task.Factory.ContinueWhenAll** **static** metoduna **TaskContinuationOptions.AttachedToParent enum** sabiti parametresini geçirerek
- **Task.Factory.FromAsync** metodu içerisinde **TaskCreationOptions.AttachedToParent enum** sabiti bildirimi yapılarak
- **TaskCompletionSource** nesne örneğini oluştururken, **TaskCreationOptions.AttachedToParent enum** sabitini parametre olarak geçirerek

Yazımızın bundan sonraki bölümlerinde söz konusu çalıştırma seçeneklerinden bazılarını, örnek kodlar yardımıyla incelemeye çalışalım. İlk olarak aşağıdaki kod parçası ile işe başlayabiliriz.

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace AttachedToParentCases
{
   class Program
   {
      static void Main(string[] args)
      {
         Task parentTask = Task.Factory.StartNew(() =>
         {
            Console.WriteLine("Parent Task...");

            #region 1 - Constructor Kullanımı

            Task childTask1 = new Task(() =>
               {
                  Console.WriteLine("Child Task 1");
               }, TaskCreationOptions.AttachedToParent
            );
            childTask1.Start();

            #endregion
```

```
            Thread.Sleep(30000); //Debug modda Parallel Task' leri izlemek için
konulmuştur
        });
        parentTask.Wait();
    }
  }
}
```

Bu kod parçasında yer alan işleyişi kavrayabilmek **childTask1** içerisindeki **Console.Writeline** satırına **BreakPoint** koyarak ilerleyecek ve çalışma zamanında **Parallel Tasks** penceresindeki durumu analiz etmeye çalışacağız. Bu işlemleri diğer kod örnekleri için de tekrar edeceğiz. İşte ilk kod parçamızın debug moddaki durumu.

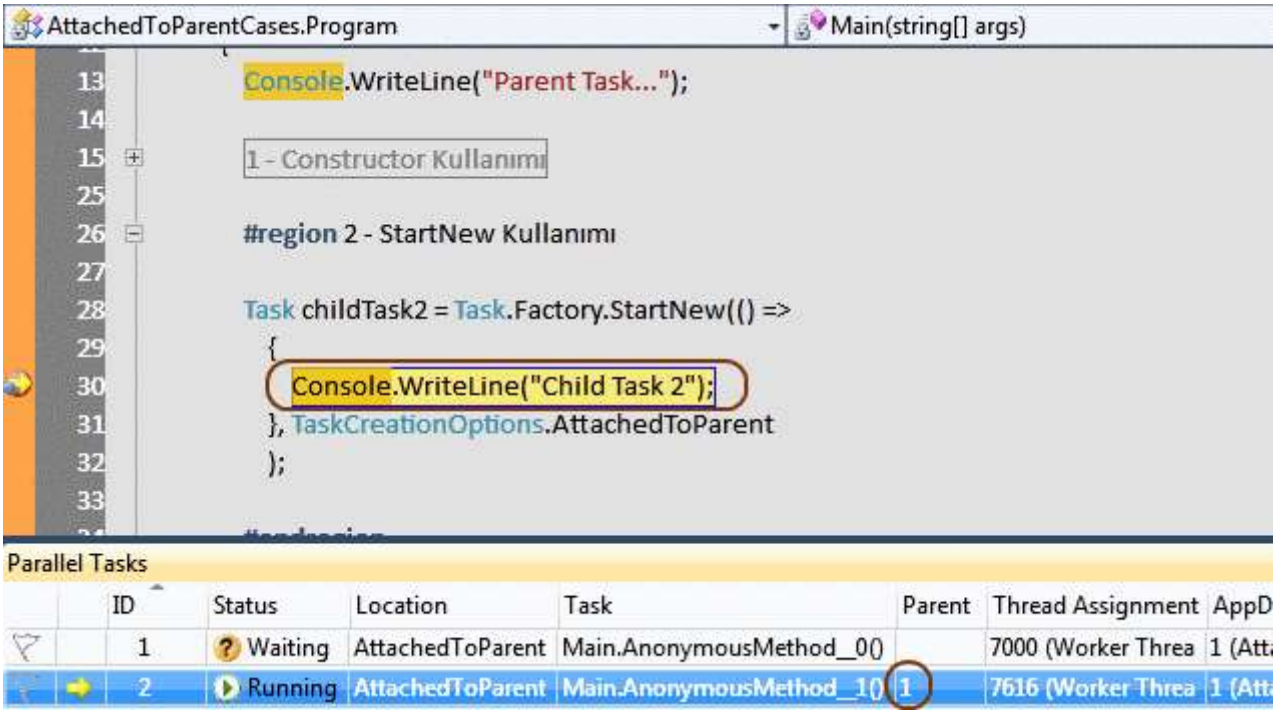


Bu kod örneğinde, **parentTask** nesne örneği **StartNew** metodu ile başlatılırken içerisinde de bir **Child Task** örneği önce **new** operatörü ile oluşturulmakta, sonrasında ise **Start** metodu ile çalıştırılmaktadır. çalışma zamanında **BreakPoint** koyduğumuz noktadan **Parallel Tasks** penceresine baktığımızda iki adet **Task** örneğinin var olduğunu görebiliriz.**parentTask** nesne örneği **Thread.Sleep** metodu nedeniyle **Waiting** modundadır. Diğer yandan başlatılan **Child Task** örneği **Running** modundadır. Ancak burada önemli olan nokta **ID** değeri **2**olan **Task** örneğinin **Parent** değeridir. Dikkat edileceği üzere **2** numaralı **ID** değerine sahip **Task** örneğinin bağlı olduğu **Task**, **1** numaralı **ID** değerine sahip **Task** örneğidir.

Gelelim ikinci kod örneğimize. Bu sefer **Child Task** örneğini **Task.Factory.StartNew** metodu yardımıyla oluşturmaktayız.

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace AttachedToParentCases
{
   class Program
   {
      static void Main(string[] args)
      {
         Task parentTask = Task.Factory.StartNew(() =>
         {
            Console.WriteLine("Parent Task...");

            #region 2 - StartNew Kullanımı

            Task childTask2 = Task.Factory.StartNew(() =>
               {
                  Console.WriteLine("Child Task 2");
               }, TaskCreationOptions.AttachedToParent
               );

            #endregion

             Thread.Sleep(30000); //Debug modda Parallel Task' leri izlemek için
konulmuştur
         });
         parentTask.Wait();
      }
   }
}
```

Gelelim çalışma zamanı çıktısına.



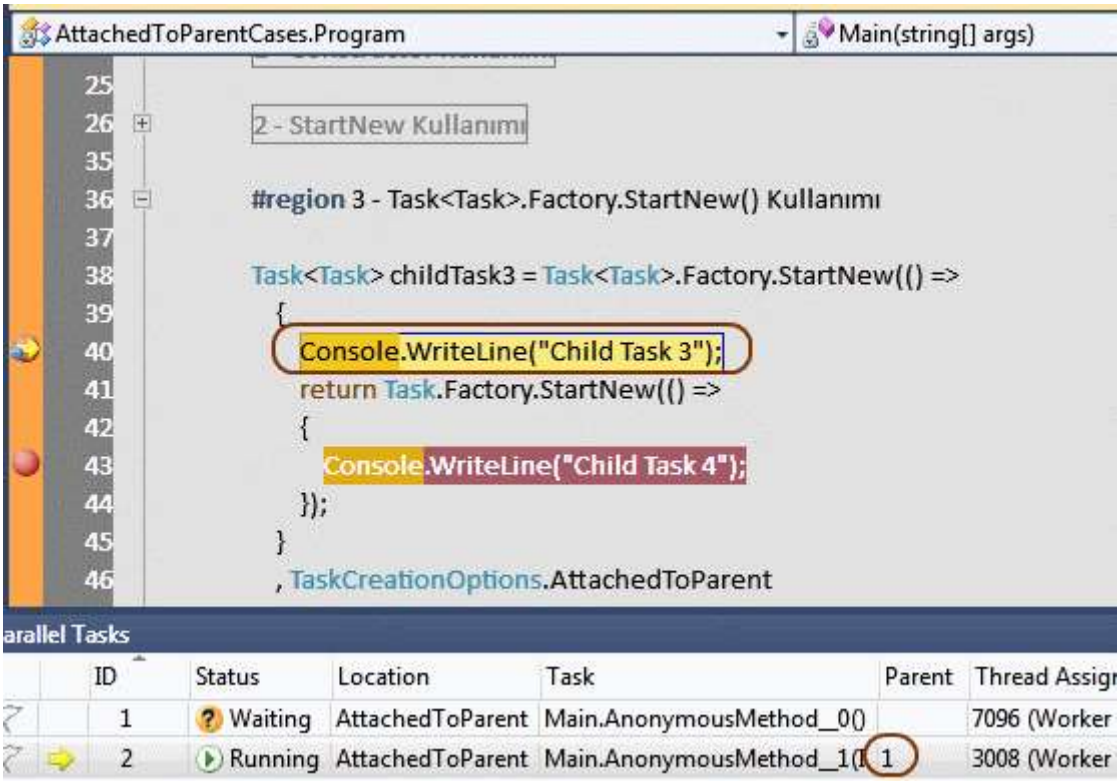
Bir önceki örnektekine benzer olaraktan, **Child Task** nesne örneğinin icra ettiği kod satırında durulduğunda, çalışmakta olan **2** numaralı **ID** değerine sahip **Task** nesne örneğinin dahil olduğu **Parent Task**' in**, 1** numaralı **ID** değerine sahip **Task** olduğu görülebilmektedir ki bu kodumuzda yer alan **parentTask** değişkeninin işaret ettiği **Task**' dir.

İlk iki kod örneğimizde olaylar oldukça nettir ve beklediğimiz şekildedir. Dilerseniz diğer örnekler ile devam edelim ve işleri biraz daha karıştıralım İşte yeni kod parçamız.

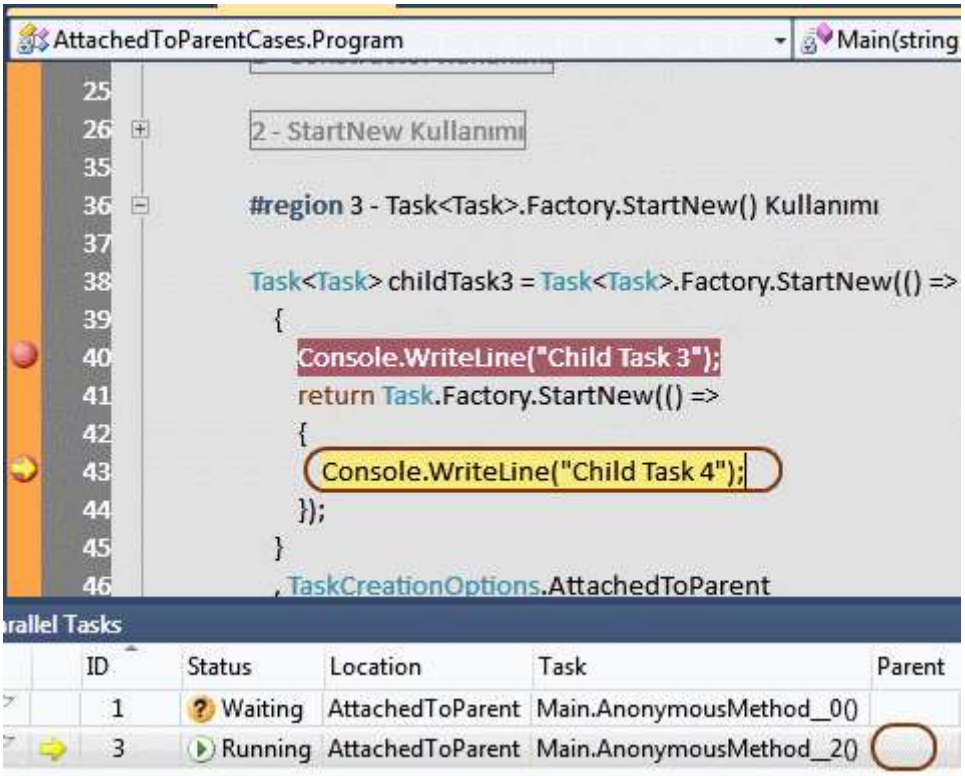
```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace AttachedToParentCases
{
   class Program
   {
      static void Main(string[] args)
      {
         Task parentTask = Task.Factory.StartNew(() =>
         {
            Console.WriteLine("Parent Task...");

            #region 3 - Task<Task>.Factory.StartNew() Kullanımı

             Task<Task> childTask3 = Task<Task>.Factory.StartNew(() =>
               {
```

```
                Console.WriteLine("Child Task 3");
                return Task.Factory.StartNew(() =>
                {
                    Console.WriteLine("Child Task 4");
                });
            }
            , TaskCreationOptions.AttachedToParent
            );

        #endregion


            Thread.Sleep(30000); //Debug modda Parallel Task' leri izlemek için
konulmuştur
        });
        parentTask.Wait();
    }
  }
}
```

Bu sefer biraz daha dikkatli davranmamız gerekiyor. **childTask3** isimli nesne örneği oluşturulurken, içerisinde iş yapan diğer bir **Child Task** başlatılmaktadır. Dikkat edileceği üzere**childTask3** için **AttachedToParent** değeri kullanılmış ancak içerideki **childTask4** için böyle bir bildirimde bulunulmamıştır. Söz konusu yeni kod parçası çalışma zamanında debug edilirken iki noktada durup düşünmek gerekmektedir.



Yukarıdaki duruma göre **childTask3**, **parentTask**' in alt **Task** örneğidir. **Parent** sütünundaki **1** değeri bunu ispat etmektedir. İlginç olan ise **childTask3** içerisinde başlatılan yeni bir **Task**' in içerisindeki **BreakPoint** noktasında durulduğunda ortaya çıkmaktadır.



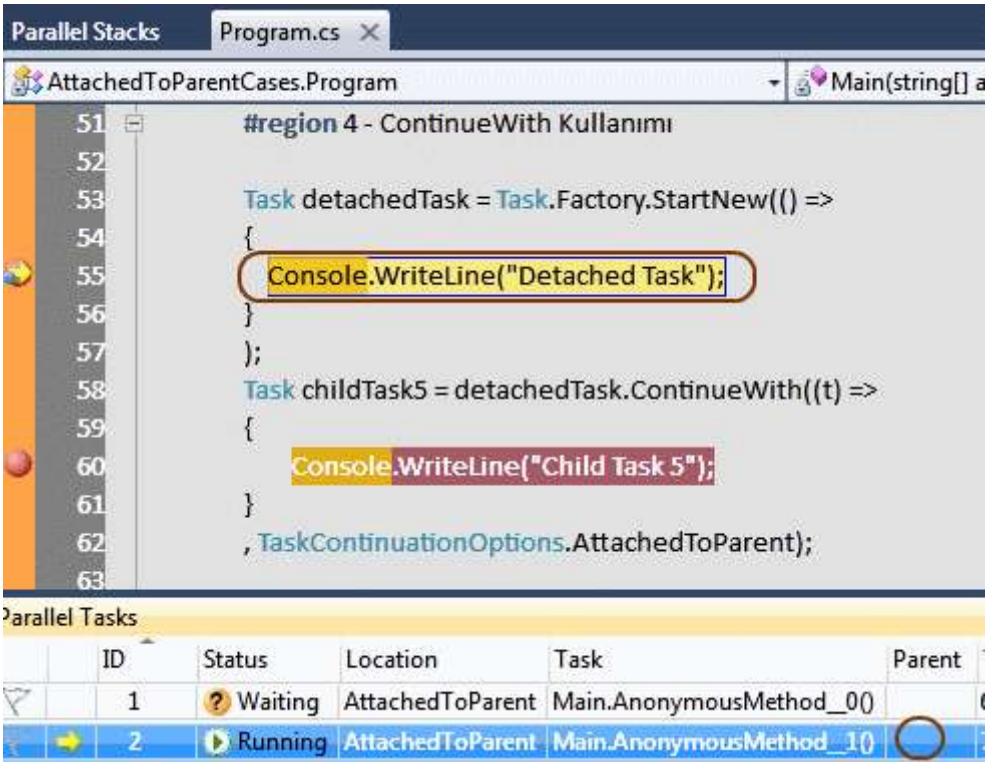
Volaaa!!! Dikkat edilecek olursa en içteki **Task**, **parentTask** nesne örneğinin çalışma zamanındaki yaşam döngüsüne ilave edilmemiştir. Kendi başına çalışan bir **Task** olarak ele alınmaktadır. İşte bu, dikkat edilmesi gereken vakalardan birisidir. Nitekim parent **Task** örneğine **Attach** edilen bir **Task** içerisindeki **Task**' lerin **enum** sabitinin ilgili değeri belirtilmeden **Attach**olmaları gerektiği sanılabilir. Oysaki şu durumda böyle olmadığı görülmektedir.

Gelelim 4ncü kod parçamıza.

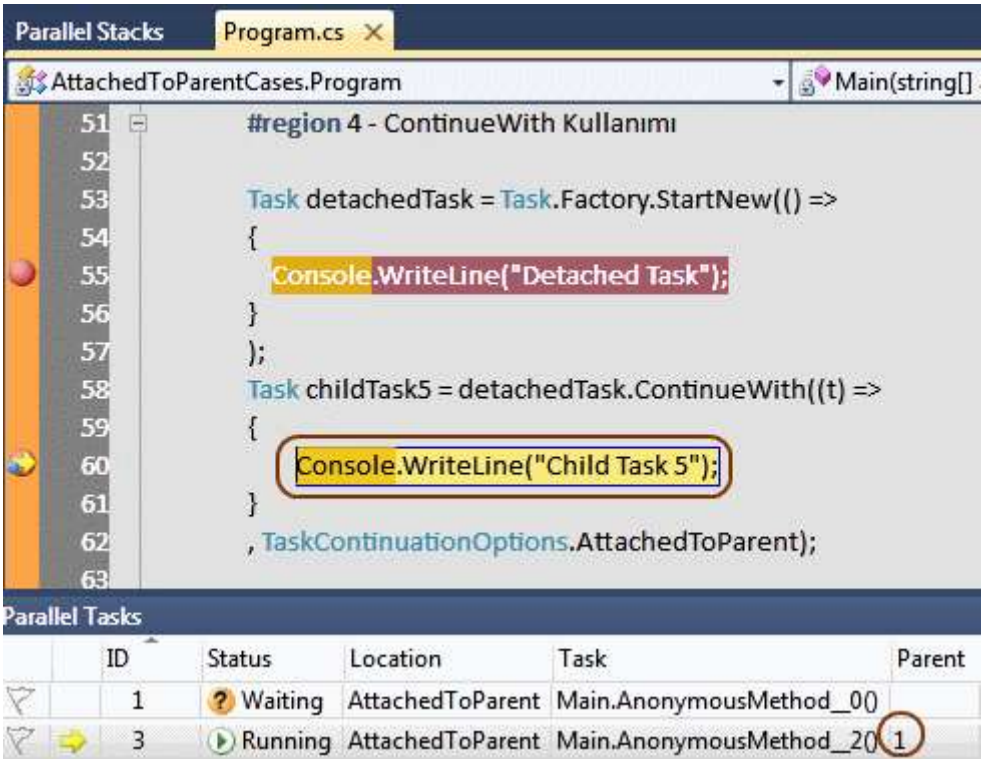
```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace AttachedToParentCases
{
   class Program
   {
      static void Main(string[] args)
      {
         Task parentTask = Task.Factory.StartNew(() =>
```

```
        {
            Console.WriteLine("Parent Task...");

            #region 4 - ContinueWith Kullanımı

            Task detachedTask = Task.Factory.StartNew(() =>
            {
               Console.WriteLine("Detached Task");
            }
            );
            Task childTask5 = detachedTask.ContinueWith((t) =>
            {
                  Console.WriteLine("Child Task 5");
            }
            , TaskContinuationOptions.AttachedToParent);

            #endregion

             Thread.Sleep(30000); //Debug modda Parallel Task' leri izlemek için
konulmuştur
        });
        parentTask.Wait();
      }
   }
}
```

Bu kez **ContinueWith** kullanımı söz konusudur. Dikkat edileceği üzere **Parent Task** örneğine **Attach** edilmeyen **detachedTask** isimli bir **Task** örneği mevcuttur. Lakin **childTask5** isimli nesne örneği oluşturulurken **ContinueWith** metodu kullanılmış ve ayrıca **TaskContinuationOptions.AttachedToParent enum** sabiti ile **parent Task** örneğine **Attach** edileceği belirlenmiştir. Bakalım gerçekten böyle midir? Yine iki noktada **BreakPoint** kullanarak söz konusu durumu analiz etmeye çalışacağız. İlk olarak **detachedTask** içerisinde duralım.



Görüldüğü gibi **detachedTask** örneği açıkça belirtilmediği için **Parent Task** örneğinin yaşam döngüsüne dahil edilmemiştir ki normali de buduru. Ancak ikinci **BreakPoint**noktasına geldiğimizde aşağıdaki ekran görüntüsünde yer alan sonuçlar ile karşılaşırız.

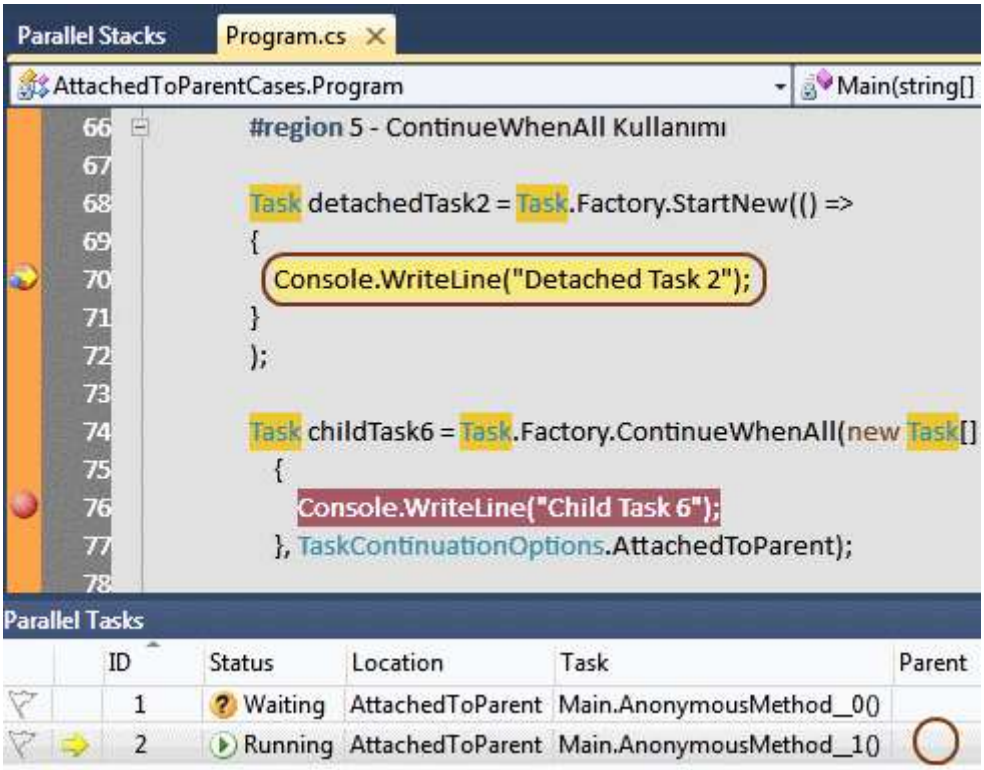


Beklediğimiz gibi **childTask5** nesne örneği **1** numaralı **ID** değerine sahip **parentTask** nesne örneğinin başlattığı yaşam döngüsüne dahil edilmiştir. Dikkat edilmesi gereken nokta,**Attach** edilmeyen bir **Task** örneği ile devam eden başka bir **Task** örneğinin, **Parent Task** yaşam döngüsüne dahil edilebiliyor olmasıdır.

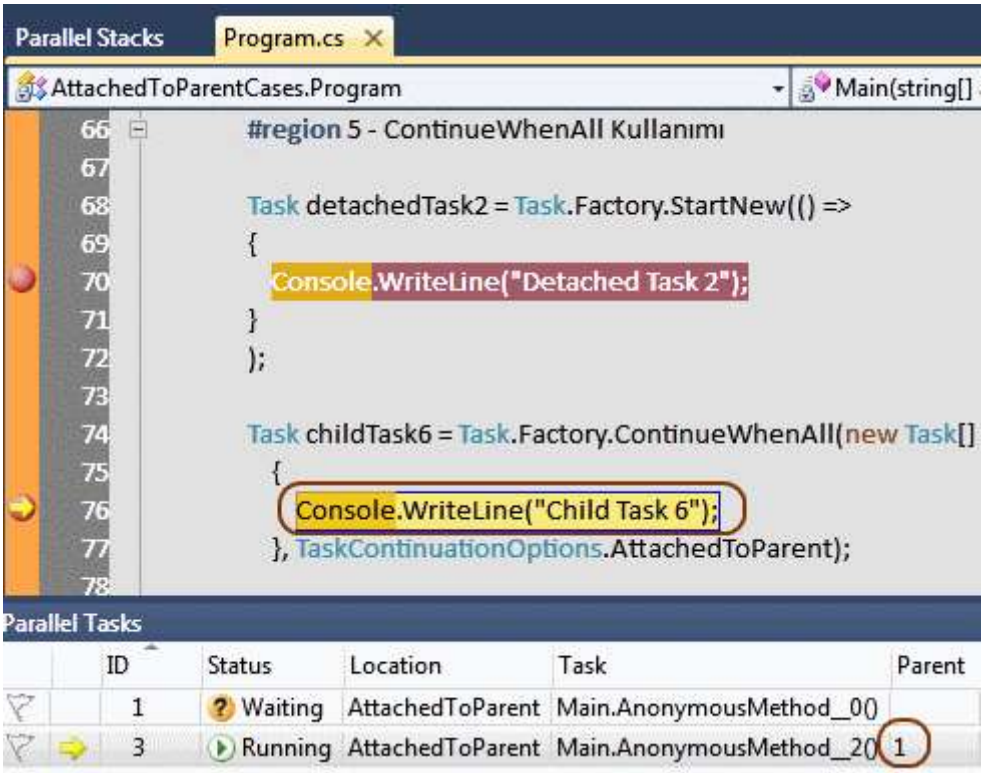
5nci durum ile devam edelim.

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace AttachedToParentCases
{
   class Program
   {
      static void Main(string[] args)
      {
         Task parentTask = Task.Factory.StartNew(() =>
         {
            Console.WriteLine("Parent Task...");

            #region 5 - ContinueWhenAll Kullanımı

            Task detachedTask2 = Task.Factory.StartNew(() =>
            {
               Console.WriteLine("Detached Task 2");
            }
            );

            Task childTask6 = Task.Factory.ContinueWhenAll(new Task[] {
detachedTask2 }, (t) =>
               {
                  Console.WriteLine("Child Task 6");
               }, TaskContinuationOptions.AttachedToParent);

            #endregion

             Thread.Sleep(30000); //Debug modda Parallel Task' leri izlemek için
konulmuştur
         });
         parentTask.Wait();
      }
   }
}
```

Bu kez yine **Detached** olarak tesis edilmiş bir **Task** örneği söz konusudur. **Parent Task** içerisine ilave etme işlemi için ise, **ContinueWhenAll** metodundan yararlanılmaktadır. Bir önceki örneğimizde olduğu gibi iki **BerakPoint** ile ilerlememizde yarar vardır. İşte sonuçlar.



Beklendiği üzere **detachedTask2** kesinlikle **Parent Task** nesne örneğinin başlattığı yaşam döngüsüne dahil edilmemiştir. Ancak bu durum **childTask6** nesne örneği için geçerli değildir.

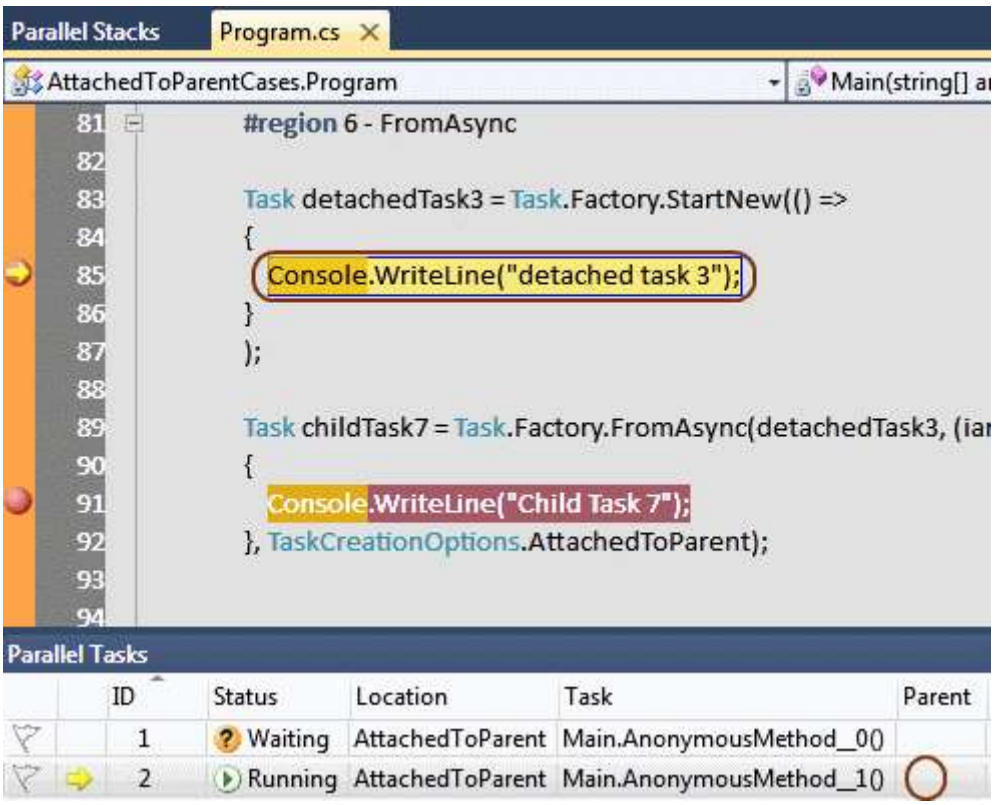


Gelelim bir diğer kod parçamıza. Bu biraz ilginç bir deneyim olacak aslında

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace AttachedToParentCases
{
   class Program
   {
      static void Main(string[] args)
      {
         Task parentTask = Task.Factory.StartNew(() =>
         {
            Console.WriteLine("Parent Task...");

            #region 6 - FromAsync

            Task detachedTask3 = Task.Factory.StartNew(() =>
            {
               Console.WriteLine("detached task 3");
            }
            );

            Task childTask7 = Task.Factory.FromAsync(detachedTask3, (iar) =>
```

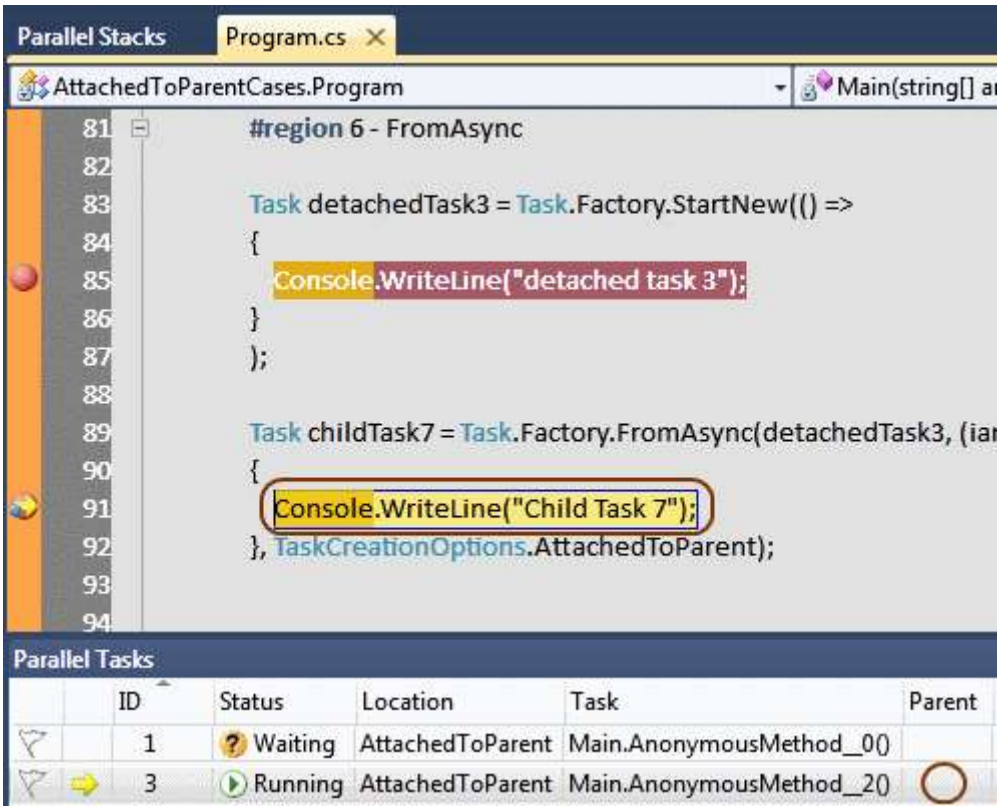
```
            {
                Console.WriteLine("Child Task 7");
            }, TaskCreationOptions.AttachedToParent);


            #endregion

            Thread.Sleep(30000); //Debug modda Parallel Task' leri izlemek için
konulmuştur
        });
        parentTask.Wait();
    }
  }
}
```

Söz konusu örnekte takip ettiğim kaynakların belirttiğine göre, **childTask7** örneğinin **Parent Task** örneğine **Attach** olması beklenmektedir. Bakalım gerçekten böyle midir? Yine iki noktada**BreakPoint** koyarak çalışma zamanındaki durumu incelememizde yarar vardır. İşte ilk durum;



Beklediğimiz gibi **detachedTask3** nesne örneği, **Parent Task** örneğinin yaşam döngüsüne ilave edilmemiştir. Kodun ilerleyen kısımlarında **childTask7** nesne örneği üretilmeye çalışılmış ve bu amaçla **FromAsync** metodundan yararlanılmıştır. Metodun ilk parametresi **detachedTask3** isimli nesne örneğidir. İkinci parametre olarak **IAsyncResult** arayüzü tipinden bir referansı parametre olarak alan **isimsiz**

metod(AnonymousMethod) söz konusudur ve son parametre ile üretilen **Task** örneğinin **Parent Task** örneğine **Attach** edilmesi istenmektedir. Oysaki ikinci**BreakPoint** noktasında durum aşağıdaki gibidir.



3 numaralı **ID** değerine sahip olan **Task, childTask7** nesne örneğini işaret etmektedir ve **Parent** değeri yoktur. Bir başka deyişle bu **Task** örneği, herhangibir **Task** örneğinin*(özellikle parentTask değişkeninin başlattığı)* yaşam döngüsüne katılmamıştır. Bu durumu bende biraz garipsemiş durumdayım ve araştırmalarıma devam etmekteyim. Mutlaka gözden kaçırdığım bir yer olmalı diye düşünüyorum. Belki de siz bana bu konuda yardımcı olabilirsiniz. Nitekim şu anda benim de bir **BreakPoint** anında belirli bir süre beklemem gerekiyor. 🙂

Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

AttachedToParentCases.rar (26,72 kb) **[örnek Visual Studio 2010 Ultimate sürümü üzerinde geliştirilmiş ve test edilmiştir]**

NedirTv?com Söyleşileri 2 - Takım Çalışması (2010-08-27T18:25:00)

nedirtv?com,nedirtv?com söyleşileri,nedirtv,takım çalışması,agile,msf,scrum,visual studio team foundation server,

Merhaba Arkadaşlar,

Hatırlayacağınız üzere geçtiğimiz günlerde NedirTv?com topluluğu sponsorluğunda söyleşilerimize başlamıştık. İlk bölümümüzde **Asp.Net Web Form**' lar ile **MVC(Model View Controller) Framework** arasındaki farklılıkları karşılaştırmalı olarak değerlendirmeye gayret ettik.

Elbette her zaman için bu kadar teknik konuları konuşmayacağız. Nitekim söyleşilerdeki amacımız sizleri çok fazla teknik detaya girerek sıkmak değil. Bu nedenle ikinci söyleşimizde, yazılım firmalarında takım çalışmasını ele aldık.

Bildiğiniz üzere başarılı ürün geliştirmelerinin veya yazılım projelerinin arkasında, sistematik olarak düzgün çalışan, belirli standartları son derece başarılı bir şekilde yerine getiren takımlar yer almaktadır. Takım olgusuna baktığımızda işin içerisinde Product Manager, Project Manager, Architect, Business Analyst, Consultant, Team Leader, Tester, Developer, Database Admin, Database Developer gibi pek çok pozisyon girmektedir. Hatta kullandığımız Visual Studio Team Foundation Server gibi enstrümanlardan, Agile(çevik) süreçlere kadar pek çok kelime yer alır. Bakalım Takım çalışması dediğimizde bu unsunasıl değerlendirilmektedir.

Söyleşiye Ulaşmak İçin

Entity Framework, Data Services, C# 4.0, Excel ve Komple Bir Uygulama (2010-08-26T16:50:00)

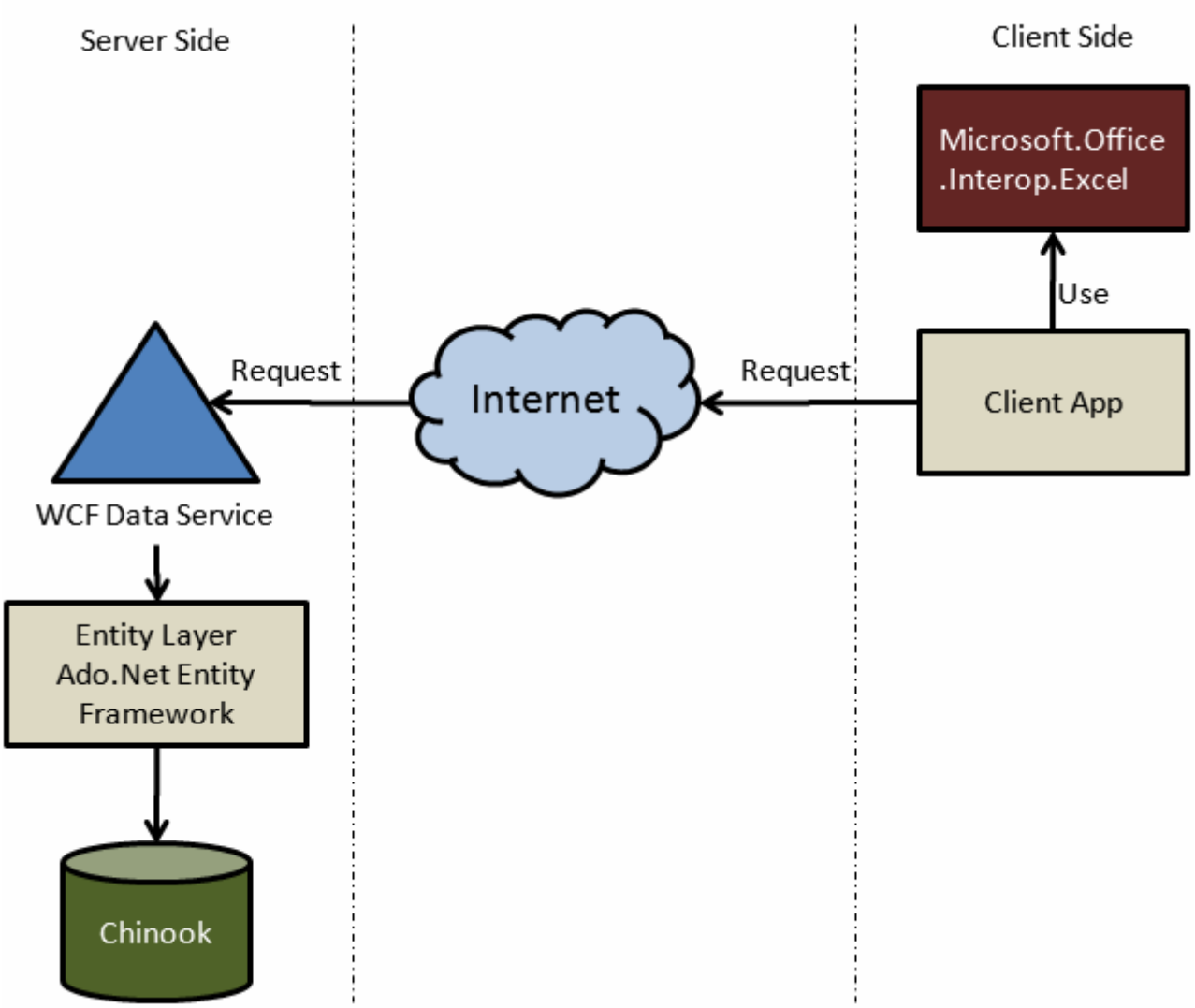
wcf data services,ado.net entity framework 4.0,chinook,c# 4.0,optional and named parameters,dynamic,excel,office interop,visual studio 2010,

Merhaba Arkadaşlar,

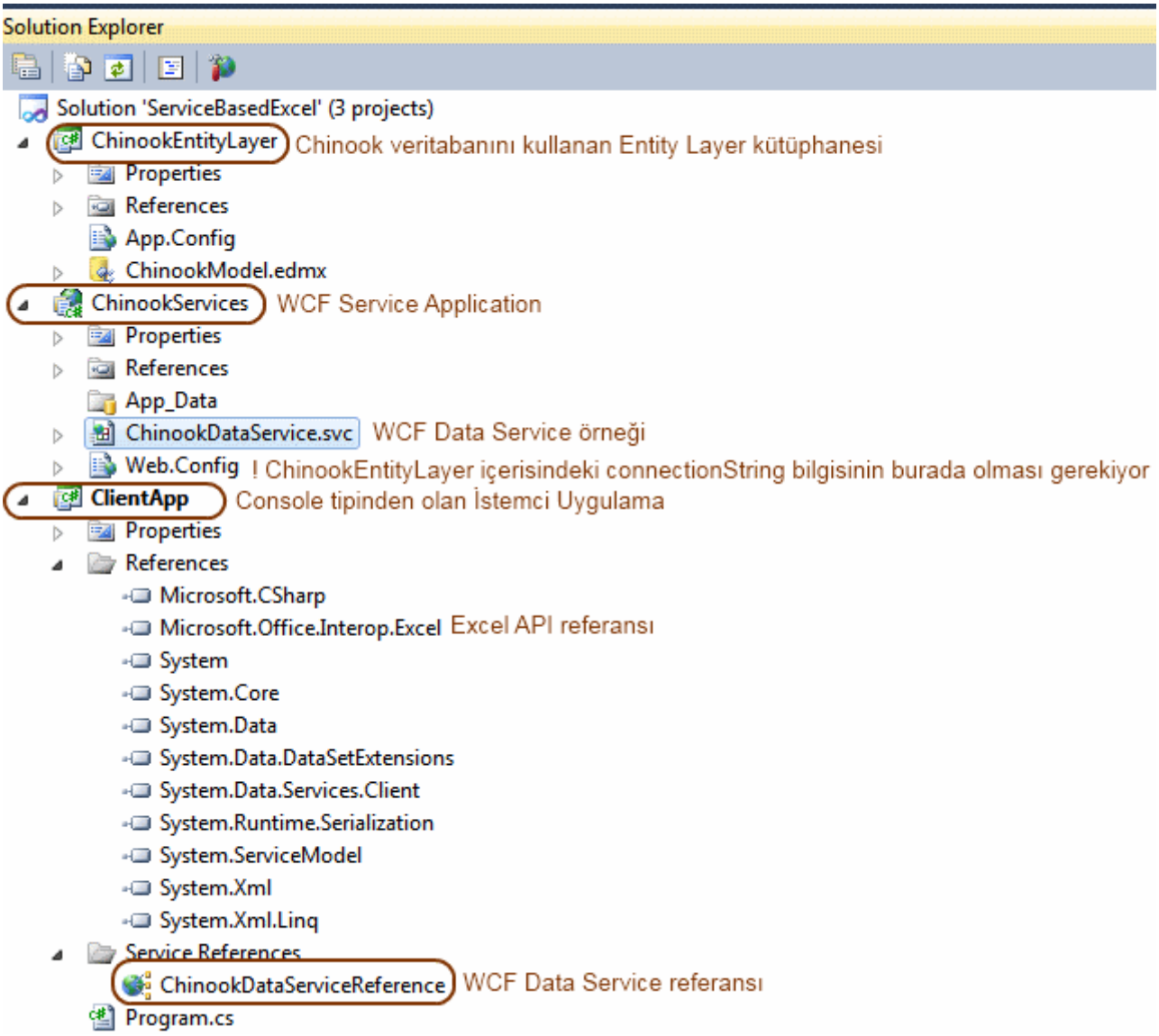
Bildiğiniz üzere bir süre önce **Visual Studio 2010** ve **.Net Framework** ürünlerinin **RTM** sürümleri yayınlandı. Her iki ürünüde sizlerle birlikte , **Microsoft PDC 2008** konferanslarından bu yana gerek yazılarımızla, gerek görsel derslerimizle incelemeye çalışıyoruz. özellikle **.Net Framework 4.0** açısından baktığımızda alet, edevat çantamızın dop dolu olduğunun eminimki hepimiz farkındayız. **Paralel programlamadan** tutun, **WCF Eco System**'e, **C# 4.0** ile birlikte gelen yeniliklerden, **WF 4.0** tarafına kadar pek çok noktada ek kabiliyetler, iyileştirmeler ve daha fazlası söz konusu. Aslında sizde benim gibi zaman zaman bu alet kutusu içerisindeki parçalardan bir kısmını alıp, örnek bir uygulamada kullanmaya çalışarak vaktinizi değerlendirmeye ve dolayısıyla offlama sorununa çare bulmaya çalışıyor olabilirsiniz. İşte bende bu düşünceler eşliğinde, havanın çok güzel olduğu şu bahar aylarında dışarıya çıkıp dolaşma şansını bulmama rağmen, evde kalıp örnek bir uygulama geliştirmeye karar verdim. İşte bu yazımız için alet çantası içinden seçtiklerimiz.

- **Codeplex** üzerinden yayınlanan bir adet Chinook veritabanı 🙂,
- **Ado.Net Entity Framework 4.0**,
- **WCF Data Services**,
- **C# 4.0 Optional, Named Parameters,**
- **Microsoft.Office.Interop.Excel**,
- ve tabiki **Visual Studio 2010**

Gelelim alet çantasından çıkarttığımız araçlar ile yapmak istediğimize...



öncelikli olarak **Chinook** veritabanının içeriğini **Ado.Net Entity Framework** üzerinden dış dünyaya sunan bir **WCF Data Service** örneğimiz olduğunu düşünebiliriz. Bu servisin sunduğu verinin istemcisi olan uygulama ise, talep ettiği içeriği alarak bir **Excel** uygulamas içersinde yayınlıyor olacak. Dolayısıyla **Client** uygulama tarafında **Microsoft.Office.Interop.Excel.dll Assembly**' ının referans edilmesi gerektiğini şimdiden söyleyebiliriz. Bu sayede **Excel API**' si yönetimli kod tarafından rahatlıkla konuşabiliyor olacağız. Diğer taraftan istemci uygulamada **C# 4.0** ile birlikte gelen ve **Office** uygulamaları ile olan etkileşimde büyük avantajlar sağlayan **Named ve Optional Parameters** kavramlarının ele alınacağını da ifade edebiliriz. İlk etapta hedefimiz örnek olarak **Track** tablosundan, istemcinin belirttiği **AlbumId** değerine sahip olan satırları almak ve bunları örnek **Excel** uygulamasında açılacak **Workbook** üzerindeki bir **Sheet**içerisinde göstermek olacak. Projeye ait **Solution** içeriği herşey tamamlandığında aşağıdaki gibi olacaktır.



İlk olarak **ChinookEntityLayer** isimli **Class Library** projesinin geliştirilmesi söz konusudur. Bu **Library** içerisine eklenen **Ado.Net Entity Data Model** içerisine, **Chinook** veritabanında yer alan tüm tabloları ekleyebiliriz. örneğimizde çok basit bir operasyonu göz önüne alıyor olsakta, sizlerin bu örnekten ilham alarak farklı sorguları da işin içerisine katacağınıza eminim **ChinookServices** isimli **WCF Service Application** tipinden olan uygulama, içerdiği **WCF Data Service** sayesinde **Chinook** veritabanına ait **Entity** koleksiyonlarını dış ortama sunmaktadır. Dolayısıyla bu uygulama, **ChinookEntityLayer** isimli sınıf kütüphanesini de referans etmelidir. Diğer yandan önemli olan noktalardan birisi de, **Entity Context** nesnesi tarafından kullanılan **Connection String** bilgisidir. **ChinookEntityLayer** içerisindeki **app.config** dosyasına eklenen **Connection String** içeriğinin aslında **ChinookServices** isimli **WCF Service** uygulamasının **web.config** dosyası içerisinde olması gerekmektedir. çünkü çalışma zamanında oluşturulan **Context** nesne örneğinin yer aldığı proje burasıdır ve bu sebepten çalışma zamanı **Connection String** bilgisini **Web.config** içerisinde arayacaktır.

```
<?xml version="1.0"?>
<configuration>
  <connectionStrings>
    <add name="ChinookEntities"
connectionString="metadata=res://*/ChinookModel.csdl|
res://*/ChinookModel.ssdl|res://*/ChinookModel.msl;
provider=System.Data.SqlClient;provider connection string=&quot;Data Source=.;
Initial Catalog=Chinook;Integrated Security=True;
MultipleActiveResultSets=True&quot;" providerName="System.Data.EntityClient"
/>
  </connectionStrings>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>
  <system.serviceModel>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />
  </system.serviceModel>
</configuration>
```

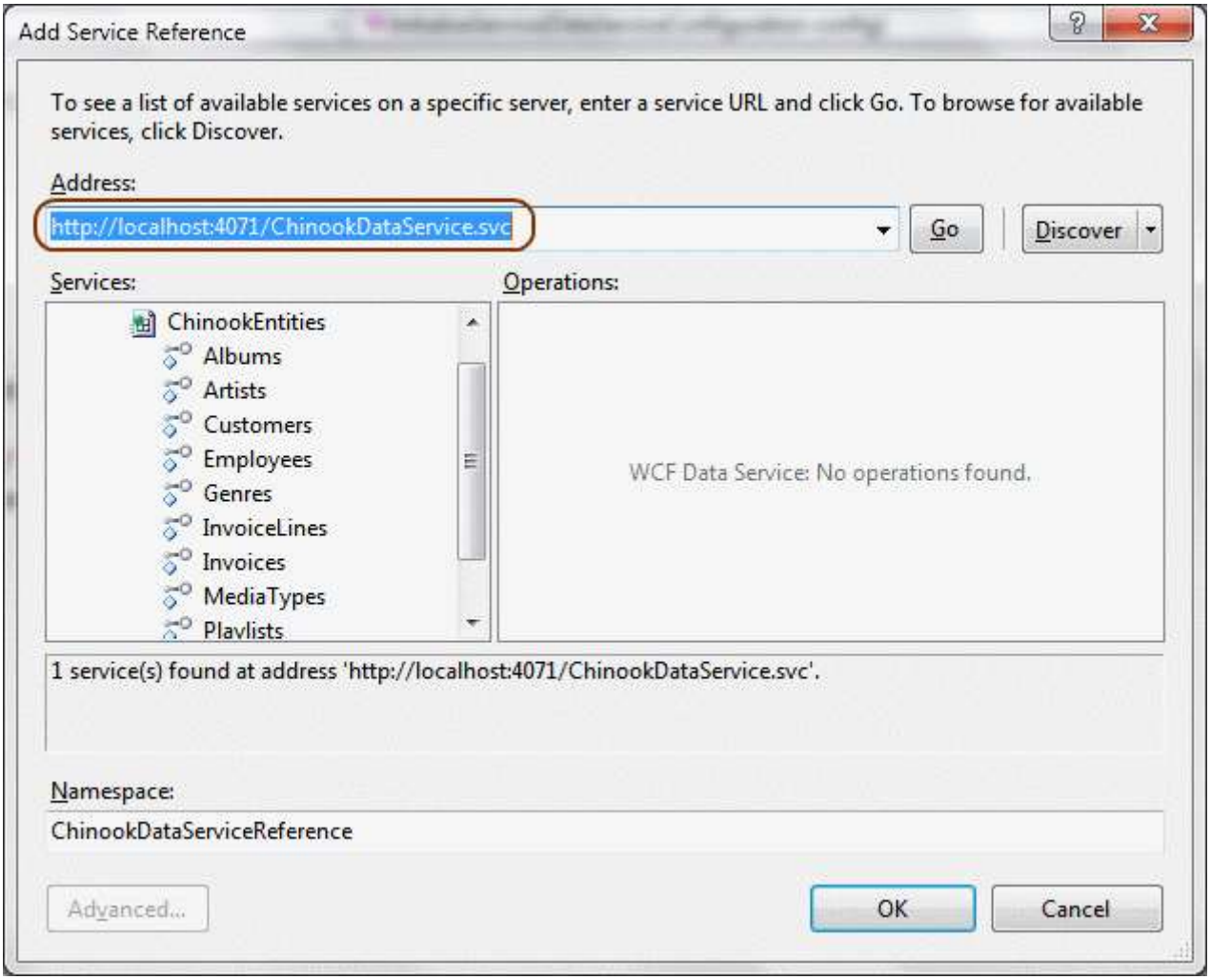
WCF Service uygulaması içerisinde yer alan ChinookDataService isimli WCF Data Service tipinden olan örneğin kod içeriği ise aşağıdaki gibidir.

```
using System.Data.Services;
using System.Data.Services.Common;
using ChinookEntityLayer;

namespace ChinookServices
{
    public class ChinookDataService
        : DataService<ChinookEntities>
    {
        public static void InitializeService(DataServiceConfiguration config)
        {
            config.SetEntitySetAccessRule("*", EntitySetRights.AllRead);
            config.DataServiceBehavior.MaxProtocolVersion =
DataServiceProtocolVersion.V2;
        }
    }
}
```

Buna göre **Chinook** veritabanı içerisindeki **Entity** koleksiyonlarının tamamı sadece okunabilir olacak şekilde dış dünyaya sunulmaktadır.

Artık istemci tarafının geliştirilmesine başlanabilir. **Console** uygulaması tipinden olan istemci tarafına*(Neden Console şeklinde tasarladığımı lütfen sormayın* 🙂

) öncelikle**ChinookDataService** isimli **WCF Data Service** örneğinin referans edilmesi gerekmektedir. Söz konusu servis ile istemci uygulama aynı **Solution** içerisinde yer aldığında **Add Service Reference**seçeneğini aşağıdaki şekilde görüldüğü gibi kullanmak yeterlidir.*(Hatırlanacağı üzere **Astoria** kod adlı **Ado.Net Data Service**' lerin **Visual Studio 2008** üzerinden kullanılan sürümlerinde, **Add Service Reference** seçeneği kullanılamamktaydı. Bunun için **datasvcutil** aracından yararlanmamız gerekiyordu. Tabiki, Data Service için Add Service Reference desteği **Visual Studio 2010** içerisinde mevcut)*



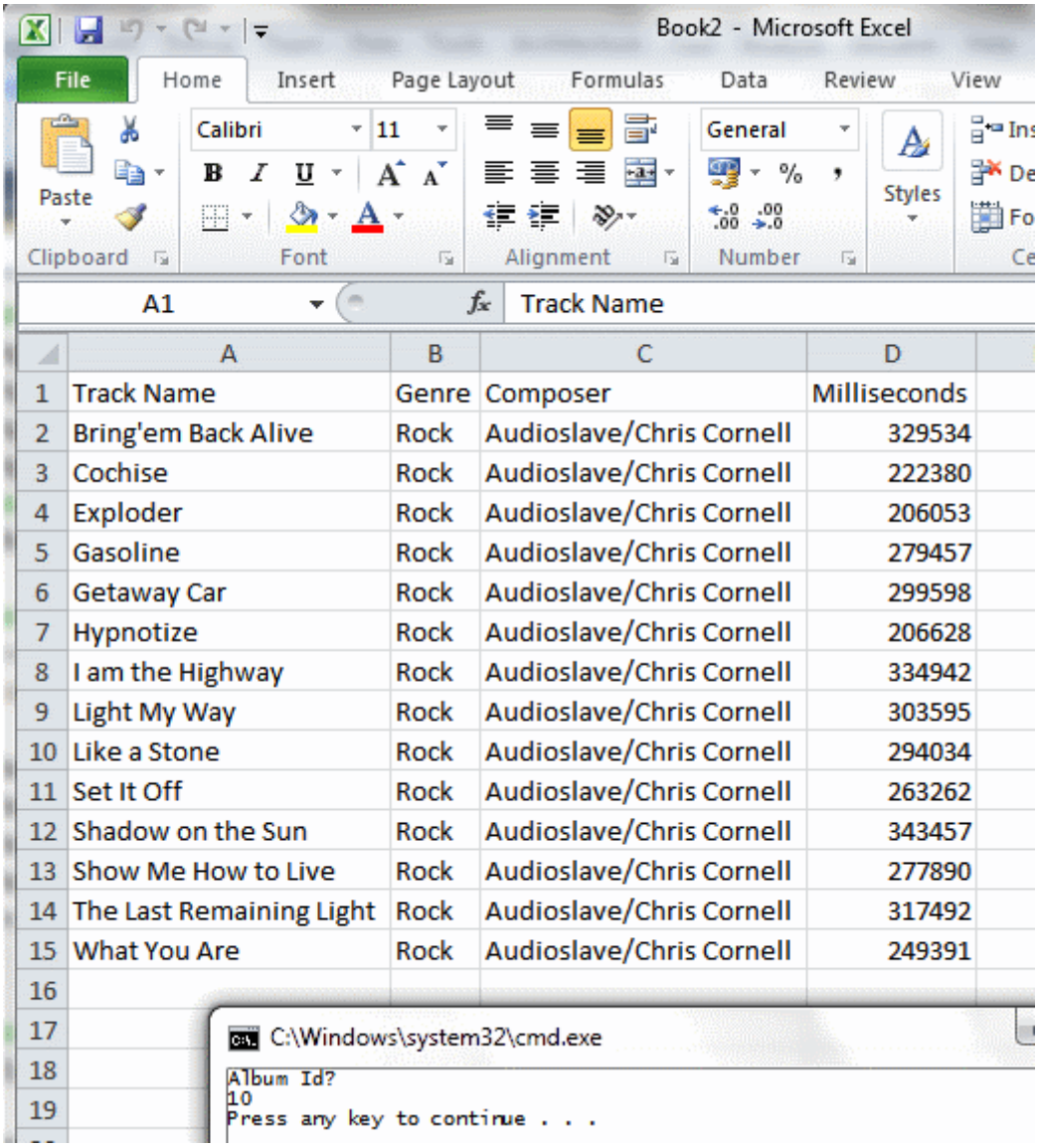
Artık istemci uygulama geliştirilmeye başlanabilir ki belki de işin en heyacanlı kısmı burasıdır 🙂 İşte Console uygulamamız ait kod içeriğimiz.

```
using System;
using System.Linq;
using ClientApp.ChinookDataServiceReference;
using Excel = Microsoft.Office.Interop.Excel;

namespace ClientApp
{
   class Program
```

```
{
    static void Main(string[] args)
    {
        // WCF Data Service örneğini kullanabileceğimi şekilde ChinookEntities nesne örneği oluşturulur.
        // URI Satırı söz konusu WCF Data Service örneğini işaret etmektedir.
        ChinookEntities entities = new ChinookEntities(new Uri("http://localhost:4071/ChinookDataService.svc/"));

        // Kullanıcıdan AlbumId bilgisi istenir
        Console.WriteLine("Album Id?");
        int albumID;
        if(!Int32.TryParse(Console.ReadLine(), out albumID)) //Eğer dönüşüm başarılı değilse 1 numaralı AlbumId değeri baz alınır
            albumID=1;

        // Sorgu cümlesi
        // AlbumId değerine göre Track örnekleri çekilir. Bu işlem sırasında Genre Entity örneklerine de ihtiyacımız olduğundan Expand metodu ile gerekli çağrı yapılır
        var result = from t in entities.Tracks.Expand("Genre")
                     where t.AlbumId == albumID
                     orderby t.Name
                     select t;

        // Bir Excel Application nesnesi örneklenir.
        Excel.Application excApp = new Excel.Application();
        excApp.Visible = true; // Excel uygulamasının görünebilir olacağı belirtilir
        excApp.Workbooks.Add(); // Yeni bir Workbook eklenir

        // Sütun başlıkları set edilmeye başlanır
        excApp.get_Range("A1").Value = "Track Name";
        excApp.get_Range("B1").Value = "Genre";
        excApp.get_Range("C1").Value = "Composer";
        excApp.get_Range("D1").Value = "Milliseconds";
        // Etkin olan Sheet adı belirlenir
        excApp.ActiveSheet.Name = "Track List for Album Id 1";

        // Elde edilen veri kümesindeki sonuçlar Sheet içerisindeki ilgili hücrelere yazdırlır
        int rowNumber = 2;
        foreach (var t in result)
        {
            excApp.get_Range(String.Format("A{0}", rowNumber.ToString())).Value = t.Name;
            excApp.get_Range(String.Format("B{0}", rowNumber.ToString())).Value = t.Genre.Name;
```

```
            excApp.get_Range(String.Format("C{0}", rowNumber.ToString())).Value =
t.Composer;
            excApp.get_Range(String.Format("D{0}", rowNumber.ToString())).Value =
t.Milliseconds;
            rowNumber++;
        }

        // Tüm sütunların uzunlukları içeriklerine göre otomatik olarak genişletilir
        for (int i = 1; i < 5; i++)
        {
            excApp.Columns[i].AutoFit();
        }
    }
  }
}
```

Console uygulaması kullanıcıdan bir **AlbumId** değeri istemektedir. Söz konusu **AlbumId** değerine göre **WCF Data Service** örneğine bir talep gönderilir ve bu talebin karşılığında dönen sonuç kümesi değerlendirilerek **Excel** içerisine alınması sağlanır. Uygulamanın çalışma zamanına ait örnek ekran çıktılarından birisi aşağıdaki gibidir.



	A	B	C	D
1	Track Name	Genre	Composer	Milliseconds
2	Bring'em Back Alive	Rock	Audioslave/Chris Cornell	329534
3	Cochise	Rock	Audioslave/Chris Cornell	222380
4	Exploder	Rock	Audioslave/Chris Cornell	206053
5	Gasoline	Rock	Audioslave/Chris Cornell	279457
6	Getaway Car	Rock	Audioslave/Chris Cornell	299598
7	Hypnotize	Rock	Audioslave/Chris Cornell	206628
8	I am the Highway	Rock	Audioslave/Chris Cornell	334942
9	Light My Way	Rock	Audioslave/Chris Cornell	303595
10	Like a Stone	Rock	Audioslave/Chris Cornell	294034
11	Set It Off	Rock	Audioslave/Chris Cornell	263262
12	Shadow on the Sun	Rock	Audioslave/Chris Cornell	343457
13	Show Me How to Live	Rock	Audioslave/Chris Cornell	277890
14	The Last Remaining Light	Rock	Audioslave/Chris Cornell	317492
15	What You Are	Rock	Audioslave/Chris Cornell	249391
16				
17				
18				
19				

Ta taaaaaa!!! Bence güzel bir örnek oldu. Ancak daha da geliştirilmesi lazım. Her şeyden önce **Console** tipinde olan istemci uygulamadan kurtulmak ve görsel arayüze sahip bir örnek üzerinden ilermelek daha yararlı olacaktır. Bu size bir ödev olabilir mesela. Yazımızı sonlandırmadan önce benim sizlere bir kaç sorum olacak;

- örnekte **C# 4.0** ile birlikte gelen hangi yeni özellikler kullanılmıştır? *(Daha önceki yazılarımızda değindik)*
- **WCF Data Service** örneğine doğru gönderilen sorgu sonucunda elde edilen içeriğe göre, **Excel** üzerinde oluşturulacak sütun adları dinamik olarak belirlenebilir mi?
- **WCF Data Service** içerisinde kullanılan **DataServiceProtocolVersion.V2** değeri ne anlama gelmektedir? *(Daha önceki yazılarımızda değindik)*
- Aynı örnek için şu tip bir sorguyu deneyip **Excel** çıktısını almaya çalışabilir misiniz? **"Composer bazlı Track sayıları?"**
- Uygulamanın sonunda **Excel** tablosunun otomatik olarak kayıt edilmesini sağlayabilir misiniz?

- **WCF Eco System** içerisinde yer alan **Data Service** dışındaki türler nelerdir? Bu servis türleri hangi amaçlarla kullanılmaktadır? *(Daha önceki yazılarımızda değindik)*
- İstemci uygulamaya ait exe dosyasının çıkartıldığı yerde **Excel** ile ilişkili bir **Assembly** bulunmamaktadır. Neden olduğunu bulup açıklayınız? *(Daha önceki yazılarımızda değindik)*

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ServiceBasedExcel_RTM.rar (213,11 kb) **[örnek Visual Studio 2010 Ultimate RTM sürümü üzerinde geliştirilmiş ve test edilmiştir]**

NedirTv?com Söyleşileri 1 - Asp.Net Web Forms vs MVC (2010-08-25T09:28:00)

nedirtv?com,podcast,nedirtv?com söyleşileri,asp.net web forms,asp.net mvc,asp.net,model view controller,

Merhaba Arkadaşlar,

Yandaki resimde kimler var?

Soldan sağa doğru bakacak olursak, NedirTv?com topluluğu lideri ve kurucusu olup aynı zamanda **Asp.Net** kategorisinde **MVP** olan **Uğur Umutluoğlu**, ortada bendeniz **Burak Selim Şenyurt** ve hemen sağ tarafta NedirTv?com editörlerinden **MVC** konusunda uzman olan **Selçuk Yavuz**.

Bu üçlünün an itibariyle aynı firmada çalıştıklarını düşünecek olursak öğle aralarında bir araya gelmeleri ve beyin fırtınaları koparmaları da son derece doğal. İşte 3müz aynı lokasyonda bulunduğumuz gerçeğini göz önüne alarak, **NedirTv?com Söyleşilerine** başlamaya karar verdik.

Açıkçası uzun zaman önce **C#Nedir?** tarafında katıldığım **.Net Radyo** gibi bir söyleşi serisi gerçekleştirmek istediğimizi ifade edebilir. Hatta ilerleyen zamanlarda **Sefer Algan** ve**Oğuz Yağmur**' u da söyleşilerimize davet ediyor olacağız.

İlk söyleşimiz için gerekli insan gücünü ve konu ile bilgi birikimini topladıktan sonra, masaüstü mikrofonumuzu ayarladık ve hemen işe koyulduk. Söyleşilerimizde teknik detayları çok fazla takılmayarak, siz değerli okurlarımızın, yol seyahatleri boyunca dinlerken bir şeyler öğrenebileceği ve daha da önemlisi bazı kararları kolayca verebileceği içerikleri hazırlamaya çalıştık(çalışıyor olacağız).

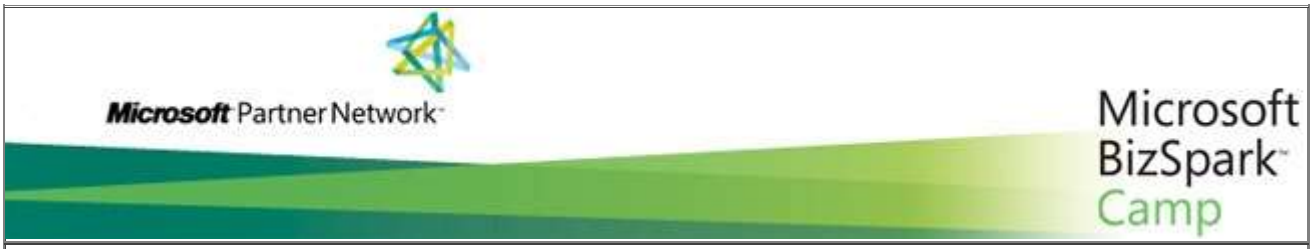
İlk bölümümüzde **Asp.Net** geliştiricileri için önemli olan bir kıyaslamayı ele aldık. Bu amaçla **Asp.Net WebForm'** lar ile **Model View Controller Framework'** ünü karşılaştırmaya karar verdik. özellikle yeni ve sıfırdan inşa edilecek olan web projelerinde **"Web Forms ile mi yoksa MVC ile mi ilerlemeliyiz?"** sorusuna cevap aradığımız **30 dakika 56 saniye** süren söyleşimize, **NedirTv?com** üzerinden erişebilir ve **Windows Media Audio(WMA)** formatındaki versiyonunu indirip dinleyebilirsiniz.

Bundan sonraki söyleşimizde bir aksilik olmassa **"Takım çalışması"** konulu bir içeriği sizinle paylaşıyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Söyleşiye Ulaşmak İçin

Microsoft Teknoloji Günleri Akşam Sınıfı Gün 4 - WCF Eco System (2010-08-19T09:45:00)

birlikte gelistir,bizspark,microsoft biz spark,microsoft,microsoft teknoloji günleri,windows communication foundation,wcf eco system,wcf data services,wcf ria services,wcf webhttp services,wcf service,wcf 4.0,web http services,windows server app fabric,



Microsoft Gelişim Atölyesi

Tarih
14 Eylül 2010 Salı - WCF Eco System

Saat
19.00 - 21.30

Yer
Microsoft İstanbul Ofisi

Eğitmen:
Burak Selim Şenyurt
Microsoft MVP

Microsoft Teknoloji Günleri -
Akşam Sınıfı

Her ay düzenli olarak gerçekleştireceğimiz ve bir seri olarak birbirini takip edecek sınıf etkinliklerimizle 9 ay boyunca siz yazılım geliştiren ve tasarım yapan iş ortaklarımızla birlikte olacağız.

Aşağıda detaylarını paylaştığımız ve sizler için hayli faydalı olacağına inandığımız Microsoft Teknoloji Günleri Akşam Sınıfı Etkinliğimize kaydınızı hemen yaptırabilirsiniz.

**Daha önce 20 Ağustos'ta gerçekleşeceğini duyurduğumuz IV. oturumumuz 14 Eylül 2010 Salı gününe ertelenmiştir.*

Ders İçerikleri:

Tarih
20 Eylül 2010 Pazartesi - Windows Server AppFabric

Saat
19.00 - 21.30

Yer
Microsoft İstanbul Ofisi

Eğitmen:
Burak Selim Şenyurt
Microsoft MVP

1. **25.Mayıs.2010/Salı: C# 4.0 ile Gelen Yenilikler : (2 Saat)** Bu eğitimde C# programlama dilinin 4.0 versiyonu ile birlikte gelen yenilikleri tanıtmaya çalışılmakta ve özellikle, Dynamic diller ile Office API vb yapılar ile olan etkileşim üzerine örnekler geliştirilmektedir. Eğitimde temel olarak dynamic keyword, optional and named parameters, PIA, Co-Contra Variance Generics konularına değinilmektedir.

2. **22.Haziran.2010/Salı: .Net 4.0 ile Paralel Programlama : (2 Saat)** Uzun süredir ev bilgisayarlarımız dahil en az iki çekirdekli sistemler üzerinde çalışabiliyoruz. Son zamanlarda 8 çekirdeğe kadar çıkabilen yeni nesil işlemciler üzerinde paralel programlama yeteneklerini sonuna kadar kullanıp daha performanslı, hızlı, verimli ve ölçeklenebilir uygulamalar geliştirmeye ne dersiniz? İşte tam size göre bir eğitim. Bu eğitim ile .Net 4.0 kütüphanelerini kullanarak kolay bir şekilde nasıl paralel programlama yapabileceğinizi göreceksiniz.

3) 20 Temmuz 2010 Salı: **WCF ile Servis Yaklaşımı**: (2 Saat) .Net 3.0 ile birlikte duyurulan ve tek bir servis geliştirme metodolojisi sunarak daha önceki dağıtık mimari geliştirme tiplerini (Xml Web Services, .Net Remoting, MSMQ vb...) bir çatı altında birleştiren Windows Communication Foundation konulu eğitimdir. Eğitimde özellikle .Net 4.0 ile birlikte gelen yeniliklere de değinilmekte olup asıl amaç WCF ile uygulama geliştirme şekillerinden bir kaçını göstermek ve tanıtmaktır.

4) 14 Eylül 2010 Salı: **WCF Eco System**: (3 Saat) WCF alt yapısı üzerine kurulu olan WCF Eco System içerisinde Data Services, Workflow Services, RIA Services, WebHttp Services ve Core Services tipleri yer almaktadır. Bu eğitimde Data Services, Workflow Services, RIA Services ve WebHttp Services konulu örneklergeliştiriecek ve söz konusu alt yapı ile değerlendirilebilecek hazır servis modelleri irdelenecektir.

5) 20 Eylül 2010 Pazartesi: **Windows Server**

AppFabric: (2 Saat) Bu eğitimde WCF ve Workflow servis örneklerinin izlenmesi, sorunların teşhis edilmesi, örneklerin yaşam döngülerinin takibi, yapılandırma ayarlarının belirlenmesi gibi birçok yönetimsel konuda geliştiriciler ile BT yöneticilerinin daha kolay anlaşabilmelerini de sağlayan Dublin kod adlı Windows Server AppFabric ürün ailesi incelenecektir. özellikle IIS üzerine gelen eklentiler ile söz konusu yönetsel işlemlerin nasıl yapılabildiği hakkında ayrıntılı bilgiyi bu oturumda bulabileceksiniz.

6. **20.Ekim.2010/çarşamba: Worfklow Foundation 4.0 : (3 Saat)** WF 4.0 beraberinde pek çok köklü yenilik ile gelmektedir. Geliştirilen Base Activity Library, paralel programlama desteği, veri akışı için gelen Argument, Variable gibi kavramlar ve daha pek çoğunun ele alındığı eğitimde basit örnekler ile WF modelin tanıtılmaya çalışılmaktadır.

7. **23.Kasım.2010/Salı: Asp.Net 4.0 : (3 Saat)** Web Programlama' nın .Net 4.0 ile birlikte gelen yeni yüzünü görmeye hazır mısınız? Pek çok yeni özellik ile birlikte gelen Asp.Net 4.0' ın anlatıldığı bu eğitimde, temelden orta seviyeye kadar basit bir web uygulaması tasarlanmakta ve konunun daha iyi kavranabilmesi amaçlanmaktadır.

8. **20.Aralık.2010/Pazartesi: Visual Basic 2010 : (2 Saat)**Bu eğitimde Visual Basic 2010 programlama dili ile birlikte gelen pek çok yeni özellik üzerinde durulmakta ve geliştirilen örnekler ile bu kavramlar pekiştirilmeye çalışılmaktadır. Bu noktada AutoImplemented Properties, Collection Initializers, Implicit Line Continuation, Mutlipe Lambda Expressions, Dynamic keyword, Type Equivalance Support gibi konular üzerinde durulmaktadır.

9. **20.Ocak.2011/Perşembe: WPF 4.0 ile Windows Programlama : (3 Saat)** .Net 3.0 ile birlikte duyurulan Windows Presentation Foundation modeli ile zengin kullanıcı deneyimine sahip windows uygulamaları tasarlanabilmektedir. özellikle Windows 7 üzerinde en iyi kullanıcı deneyimini sunan WPF 4.0

ile birlikte gelen yenilikleri öğrenmeye ne dersiniz?

Zamanı Etkin Kullanmak için Ona Hükmetmek (2010-08-16T18:10:00)

teknik dışı konular,zaman yönetimi,

Merhaba Arkadaşlar,

for(;;;)

{

O sabah işe gitmek üzere evden çıktığımda, elimde son ayın bilgisayar dergilerinden birisi yer alıyordu. Uzun süredir arızalı olan kişisel taşıma aracımı*(Dül dül diyebiliriz)* kullanamadığımdan*(Akü, elektrik kontağı, air bag sorunları nedeniyle)*, işe toplu taşıma araçlarını kullanarak gitmekteydim…

Güzel ve güneşli bir gündü. Ancak son yılların en sıcak günlerini yaşadığımızdan hiç şüphem yoktu.

Bindiğim hat otobüsü her zamanki gibi doluydu. Oturacak bir iki yer vardı ki onları da bayanlara, yaşlılara veya çocuklara bırakmak en doğrusu idi. Arkalarda kendime güzel bir köşe buldum. Kulağımda **4Gb** kapasiteli **Philips** marka **MP3** çalarımdan çıkan **ACDC** melodileri çınlıyordu.

önce bilgisayar dergimi açtım ve yeni çıkan bir kaç donanım haberini okumaya başladım. Gelecek bir kaç ay içerisinde monte ederek odamda oluşturmayı planladığım masa üstü sunucusu için parça bilgilerine baktım. Sonra ise MP3 çalarımda daha farklı bir melodiye geçiş yaptım. **Scott Hanselman'** ın Hanselminutes sitesinden yayınlanan **PodCast'** lerinden birisini dinlemeye koyuldum. Konu **ASP.NET MVC 3 Preview 1** idi.

Konuşmanın sonlarına doğru durağıma varmıştım. İndikten sonra şirkete ulaşmak için yaklaşık olarak **7** ile **8 dakika** arası yürümem de gerekiyordu. **Podcast'** i sonuna kadar

dinlemek için iyi bir süreydi. Derken şirketime ve masama ulaştım. üzerimdeki eşyaları çekmecelerime boşalttıktan sonra bilgisayarımın açma tuşuna dokundum ve kendimi hemen lavaboya attım. Sonuçta otobüs ile gidip geliyordum ve bu sıcaklarda pek de hiyjenik olmadıklarını biliyordum. Ellerimi, yüzümü güzelce temizledikten sonra, yol üstünde çay ocağına uğrayıp sabah kahvemi hazırladım ve masama oturdum.

Bu esnada bilgisayarım açılmış ve Feedreader programım çoktan güncel **Feed**' leri indirmeye başlamıştı. Hemen başlıklarına bakarak bir kaç tanesini gözüme kestirdim ve onları okumaya başladım. önümüzdeki **45 dakika** boyunca bunu yapacaktım.

Derken öğlen oldu. Haftanın belirli günleri öğle araları yemek için dışarı çıkmazdım. Bunun yerine teknokent içerisindeki yemekhaneyi kullanır ve bir an önce masama giderdim. Bu gün de böyle yapmayı uygun gördüm çünkü yazmak istediğim güzel bir makalem vardı. Makalenin konusu ile ilişkili araştırmalarımı zaten bir gün öncesinde yapmıştım ve kafamda bir şablon oluşturmuştum. önce özgün örneği yapacaktım ve eğer vakit kalmazsa bir sonraki günün öğle arasında veya akşam mesai sonrasında yazısını hazırlayacaktım.

öğle vaktinin bu şekilde geçişinden sonraysa, günlük şirket işlerinin yapılması ile bir anda akşamın olduğu farkettim. Evet gitmek için kullandığım mahalle otobüsünün duraktan geçmesine, akşam saati trafiği ve edinilen tecrübeler düşünüldüğünde daha **1 saat** vardı. Mesai **18:00**' da bitiyordu ama ben kendimi zaten **17:45**' te tatil moduna almıştım. Bu da toplamda bana **75 dakika – (2 dakika hazırlanma + 8 dakika durağa yürüme süresi)= 65 dakika** boşluk veriyordu. Mükemmel bir süre Yazımı hazırlamaya başlayabilirdim ve öyle yaptım.

Dönüş yolculuğunda ise otobüsün gürültülü sesine rağmen yol boyunca, channel9' da yayınlanan sohbetlerden birisinin **MP3** formatındaki kaydı kulaklarımı doldurmaya başlamıştı. Eve vardığım da her zaman ki gibi ailemle ilgileneceğimden en azından gece herkes uyuyuncaya kadar teknolojiye bir süre ara vereceğimi iyi biliyordum.

S(h)arp Efe ile oynayıp onu mutlu ettikten ve uyuttuktan sonra*(ki onu uyutmak için sallamız şarttır ve uyuması gece 23:00' ı bulur)* tüm gün yorulan eşim de uyuyunca, bana en azından uykuya dalana dek **1** veya öğrenmeye çok açsam **3 ile 4 saat** vakit kalmıştı. Bu vakti Amazon' dan getirttiğim son kitaplardan birisini okuyarak değerlendirdim.

Ertesi sabah…

Kalktım, kişisel bakımımı yaptım ve **Philips** marka **4Gb** kapasiteli **MP3** çalarımı kulağıma takarak otobüs durağına yöneldim. Bu kez kulaklarımı **Metallica'** nın eski melodileri ile yeni albümlerini çok beğendiğim **Faithless** dolduruyordu. Elimde bir kitap veya dergi yoktu. Bu sabah bir şeyler
okumayacak, **Hanselminutes** veya **channel9** dinlemeyecektim…Bu sabah tek kaynağım vardı…Zihnim…

Yol akarken ve otobüsün içerisinden dışarıya doğru bakarken, kafamda bir sürü yeni konu hakkında konuşma geçiyordu. Tabiri yerinde ise, zihnimde kendi kendime seminer veriyor, en son öğrendiklerimi tartışıyor, yazmayı planladığım makaleler üzerine konuşuyordum…

Otobüs şirketin durağına vardığında indim. Bu gün hava biraz bulutlu idi ama yine de sıcak rüzgalar esiyordu. Şirketime doğru **7 ila 8 dakikalık** yolculuğuma başladığımda artık kulağımda **MP3** çaların sesi yoktu. Onu kapatmıştım. çevremden gelen sesler beni rahatsız etmiyordu. Ancak **ITU'** nün **Teknokent ARI 1** binasına giden ara yoluna geldiğimde, sadece ağaçların yapraklarının çıkarttığı sesler vardı. Bu kendi kendine teknoloji paylaşımı yapan zihnim için sunulan mükemmel bir seminer odasıydı aslında. Kafamda fikirler uçuşuyor keşke onları takip eden bir Fiddler aracı olsa diye iç geçiriyordum…

Derken masamdaydım…

// TODO : Yapılacak diğer işler

}

Main(string[] args)

{

Bu uzun giriş aslında sizlere zamanı kullanmak için bir **best practice** olması amacıyla yazıldı. Zamanı yönetmek, her insanın er geç karşılaşacağı en önemli sorunsallardan birisi. Zaman durmuyor ve sürekli olarak ilerliyor. Geriye döndürülemiyor. Geriye dönemiyoruz*(Aslında bu kötü olduğu kadar da iyi bir şey)* özellikle sürekli bir şeylerin öğrenilmesi gereken bir sektörde yer alıyorsak, boş geçen zaman bizlere ağır faturalar çıkartabiliyor. Bu nedenle zamanımızı en iyi şekilde kullanabilmeyi, bir başka deyişle ona hükmetmeyi de bilmeliyiz

İşte bu yazımızda bir yazılımcı olarak gündelik yaşantımızda, yeni bir şeyler öğrenmeye nasıl zaman ayırabileceğimizi maddeler halinde görmeye çalışacağız. Hayatımızda nasıl boşluklar var bir bilseniz Bu boşlukları keşfetmek ve onları değerlendirmek ise bizlerin elinde. İşte değerlendirilebilir zaman aralıklarımızdan bir kaçı…

1 – Toplu Taşıma Yolculukları(Otobüs, Vapur, Metro, Minibüs)

Evet evet biliyorum…Bu sıcakta, balık istifi gibi otobüsler, metro vb…Ama yine de toplu taşıma araçlarında geçen süreler duruma göre en iyi şekilde değerledirilebilir. Bu noktada özellikle **istanbul** gibi büyük bir metropol' deyseniz ve yoğun trafik nedeni ile gününüzün **1**

saati veya daha fazlası yolda geçiyorsa, **MP3** oynatıcısına **Scott Hanselman** ve benzerlerinin **podcast'** lerini ve **channel9** üzerinde yayınlanan görsel derslerin ses kayıtlarını yüklemenin zamanı gelmiştir. Hatta yakın zamanda NedirTv?com üzerinden başlatacağımız **NedirTv Sohbetlerini'** de ekleyebilirsiniz

Tabi eğer yolculuk sırasında oturarak gitme şansınız var ise, bu durumda bir kitap veya dergi okunmasında da yarar olacaktır. çok kalın*(örneğin 1000 sayfa üzerinde)* kitapları tercih etmek pek doğru olmayabilir. Bu kitaplar ağırlıkları nedeni ile*(hem içerik hem de konu bakımından)* zamanla sıkıcı hale gelebilirler. Nitekim kitabı okuduğumuz ortam, deniz kenarı veya yemyeşil parkta bir bank değildir. **Ikarus**otobüslerin, bilemediniz en fazla Mercedes marka olanların gürültülerini içeren veya metronun raylar üzerinde bıraktığı tiz seslerle dolu ve benzeri bir yerdir.

Bu nedenle **In Action(LINQ in Action, Asp.Net MVC in Action vb…)** gibi seriler veya **For Dummies(SOA for Dummies vb)** gibi kitaplar bu tip kısa süreli yolculuklarda daha taşınası ve okunasıdır.

2 – Uzun Süreli Yolculuklar(Gemi, Otobüs, Uçak)

Az önce hani şu kalın, ağır kitaplardan bahsetmiştik ya İşte onları okumak için en uygun yerlerden birisi uzun süreli yolculuklardır. Söz gelimi tatile giderken…

Ancak ister uzun süreli ister kısa süreli yolculuklar olsun, okunacak materyallerin çok yoğun kod içermesi de pek doğru değildir. Nitekim kodları anında tatbik edebileceğimiz bir cihaz yanımızda olmayabilir. Bu nedenle daha çok kalıpları anlatan, yazılım mühendisliği ile ilgili olan kaynakları takip etmek yerinde olacaktır.

Tabi bazı istisnalarda yok değildir. Eğer kitaptaki kodların sonuçları, ekran görüntüleri ile desteklenerekten anlatılıyor ve siz **kafanızda debug**noktalarını belirli bir yere kadar çalıştırabiliyorsanız, bu durumda okuduğunuz kitap size büyük fayda katacaktır. Kafaca **Debug** etmek yazılım konusunda ileri seviyeye gelmek isteyen bir çalışan için de oldukça önemlidir.

3 – Mesai Başlangıçları ve Yemek Dönüşleri

özellikle **Pazartesi** günlerini kimse sevmez sanıyorum ki. Güzel geçen bir hafta sonunun arkasından, yeni bir iş yoğunluğunun başlangıcı. Bunu unutturmanın bana göre tek bir yolu vardır. O da **Feedreader**

Mesaiye başlamadan önce hatta mesai başlasa bile iş yoğunluğuna göre kısa bir süre *(en az 1 saat olmasında yarar vardır)* güncel **Feed'** leri okumanın büyük önemi vardır.

Bu amaçla mutlaka kendinize bir **takip edilesi blog**' lar listesi yapmamız gerekir. Tüm haberleri okumak büyük bir zaman kaybına neden olabilir. Ancak arada mutlaka gözüne kestireceğiniz bir, iki giriş olacaktır. özellikle yenilikleri takip etmek adına mesai başlangıçlarında ve hatta yemek dönüşlerinde ayıracağınız kısa süreler, sizin için eğitici ve bilgilendirici olacaktır.

4 – Yerel Trafik Zorlukları(Kazalar nedeni ile iş çıkış saatlerini geciktirmek)

Aslında **istanbul** gibi büyük ve kalabalık bir şehrin trafiğinin sıkışmasının nedeni yandaki koyunlar değildir elbette 😬

Ancak bazı akşamlar özellikle iki yaka arasında geçiş yapmak saatlerimizi alabilmektedir. Bu konuda karayollarının anlık yol trafik rapolarına bakmakta yarar vardır. Nitekim trafiğin çok yoğun olduğu bir zamanda işten çıkmak yerine belki **1 saat** sonra çıkmakla eve ulaşma süreleri aynı olacaktır.

Peki böyle bir durumda şirkette kaldığınız o **1 saatlik** zaman dilimi içerisinde ne yapabilirsiniz?

Bu tamamen size kalmış. Ancak ben iş dışında bir şeyler yapmanızı öneririm. Bu işlerden kastım tabiki sosyal portallerde gezmek veya video paylaşım sitelerini izlemek değildir. Aslında gün içinde okuduğunuz bir blog girdisinin, kitaptan okuduğunuz bir bölümün veya dergiden edindiğiniz bir fikrin uygulamasını deneyebilirsiniz. Bu **1 saatlik** zaman diliminde yapacağınız kısa örnek uygulamalar, okuduklarınızı pekiştirmeniz açısından size büyük fayda sağlayacaktır.

5 – Hayalinizdeki Şirketin Verdiği Boşluklar (örneğin her cuma öğleden sonralarının araştırma ve iç eğitimlere ayrılması)

Hayalimizdeki yazılım şirketi 😬

Kulağa çok ama çok komik geliyor değil mi? En azından ülkemizde komik geldiğini ifade edebiliriz. Nitekim burada hayal ettiğimiz şirket size haftanın belirli günlerinde, gelişiminiz ile ilişkili belirli konuları araştırmanız için zaman ayırmaktadır. Bu zamanda **10, 15 dakika gibi kısa bir süre değildir**. Bildiğimiz**yarım gündür**. Tabi hayal ettiğimiz bu

şirketin proje planlarında bu yarım günler de hesaba katılır. O yarım günlerde kimse sizden bir geliştirme beklemez.

Şimdi şunu dile getirebilirsiniz. **"Vay Burak Hocam…Ne kral bir şirkette çalışıyorsun."** Aslında bu hayali şirketi ben de hayal etmekteyim.

Ama bunun için bence bizim iş verenlerimizin bilinçlenmesi ve çok yol kat etmesi gerekmekte. Tabi eğer siz bu tip bir şirkette çalışıyorsanız lütfen kıymetini bilin. Nitekim özellikle yabancı pek çok şirkette bu tip çalışmaların olduğunu biliyoruz. Hatta çok eskiden **Netron'** da eğitmen olarak görev almaktayken de bu tip bir çalışma başlatılmıştı.

üstelik çalışmanın güzel yanlarından birisi de şuydu; bu serbest zamanlarda çalışma yapanlar, çalıştıkları konuları bir sonraki hafta diğer ekip arkadaşlarına aktarmaktaydı. Bu tip bir çalışmada hem anlatan hem de dinleyen taraf olduğunuz için, öğrenme sürecindeki verimlilikte yüksek olmaktadır.

6 – Klavye Kullanım Hızları(Paylaşırken Avantaj)

Aslında bu noktada zamanı yönetmek için kullanabileceğimiz bir araçtan bahsediyor olacağız. **Klavye** 😀

Paylaşımcılıkla yanıp tutuşan bir yazılımcı iseniz, bildiklerini, öğrendiklerini ve aktarmak istediklerinizi ele alacağınız bir mekanınız var demektir.

Bu mekan kimi zaman kişisel bloğunuz, kimi zaman yazarlık yaptığınız internet topluluğunuzdur. Ancak bu tip yerlerde paylaşım yapmak için gündelik yaşantınızdan da epey bir vakit ayırmanız gerekecektir.

İşte bu nedenle paylaşmak istediğiniz konuları anlatırken oldukça hızlı klavye kullanmanız sizin avantajınız olacaktır. Nitekim bu sayede çok daha kısa sürede bir yayınlama yapabiliyor olursunuz.

7 – Meşhur Yerli Türk Dizileri ve Akşam Saatleri

Gelelim **meşhur Türk dizilerine** 😃 Malumunuz hemen her akşam, hemen her kanalda uzun metrajlı film sürelerinde oynayan yerli diziler var. Bu dizilere harcanan sürelere ne yazık ki acıyorum. Hani yabancıların bir lafı vardır. Televizyon için **"Aptal**

Kutu” derler Gerçekten de böyledir.

Elbette suçu sadece yerli dizilere bağlamak doğru değil. Kısa oldukları kadar sürükleyici olan, **n sayıda sezondan oluşan** ve bazen sezon sonları da çok kötü biten yabancı diziler de bulunuyor.

özetlemek gerekirse, bir günde **televizyon** başında geçireceğimiz zamanı çok daha etkin bir şekilde kullanabiliriz. Bu zaman diliminde ille de kitaplara gömülüp, bilgisayar başında kodlarda kaybolmak zorunda da değilsiniz. Bunun yerine örneğin yakın çevrede aynı sektörden olan arkadaşlarınızla toplanıp beyin fırtınası da yapabilir, fikirlerinizi paylaşabilirsiniz. Bunu güzel bir kafe de yapabileceğiniz gibi, sahilde denize kenarında ya da parkta çekirdek çıtlatırken de gerçekleştirebilirsiniz. Dolayısıyla **“Aptal Kutu’ yu”** bir kenara bırakın.

Buraya kadar anlattığımız bu **7 madde** ile sizlere kısa da olsa, zamanı daha etkin ve verimli yönetmeniz için gerekli bir kaç ip ucu vermeye çalıştım. Anlatılanları düşündüğümüzde aslında her bireyin kendi zamanını yönetebileceğini, yönetmesi gerektiğini ifade edebiliriz. Dolayısıyla zamanı nasıl yöneteceğinizi en iyi siz bilebilirsiniz. Nitekim gündelik yaşantınızda yaptıklarınıza hakim olan, onlara ayrıdığınız zamanlarda esneklik yapabilecek olan tek kişi sizsiniz. Kendi kendinizin zaman planlayıcısı olmanızı öneririm. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Silverlight - JSON ile Çalışmak (2010-08-13T10:15:00)

silverlight,wcf,wcf webhttp services,json,

 Merhaba Arkadaşlar,

Uzun süredir şöyle deliksiz uyuyamıyordum. Malum evde bir afacan var. Pek uyumayı sevmeyen, sürekli hareket halinde olmak isteyen Sarp Efe izin verdiğinde, eşim ve ben dinlenmek için çeşitli işlere dalıyoruz. Ben uzun süredir **Bulmacalara** takılmış durumdayım. Bir de şu eski dil karşılıklarını isteyen sorular olmasa. Geçtiğimiz günlerde yine böyle bir boşluk yakalamışken, kendimi bulmacalar arasında yüzerken buluverdim.

Ancak bir süre sonra "...eski dildeki karşılığı..." sorularından sıkıldım ve televizyonda neler olduğuna bir akayım dedim.

Televizyonda yandaki resimde görülen adam vardı ve ismi **Jason**' dı. Açıkçası **Jason Statham**' ın fanatiği bir sinemasever olarak bu isim benzerliğinin, böyle korkutucu bir karakter üzerinde olması beni üzmüştü. Nitekim Jason ismini düşününce aklıma gıcır gıcır parlayan **Audi** marka arabalar gelmekteydi. Her neyse...Filme fazla takılmadım ama **Jason, Jason** derken, bu isim **JSON** diye dudaklarımdan süzülmeye başladı. Pek tabi bunun doğal sonucu olarak bilgisayarımın başına oturdum ve **JSON** ile ilişkili bir şeyler yazmaya karar verdim. İşte başlıyoruz

Bildiğiniz üzere **HTTP** bazlı **WCF** servislerinden(WCF WebHttp Services - JSON Formatlı Response üretmek) **JSON(JavaScript Object Notation)** formatında çıktılar yayınlanabilmektedir. Bazı durumlarda istemci tarafı, **JSON** veri içeriği ile çalışmayı tercih edilebilir. özellikle **XML** ile karşılaştırıldığında, **JSON** formatının daha az yer tutan bir yapıya sahip olması, bu seçimin yapılmasında önemli bir etkendir. Biz bu yazımızda bir **WCF WebHttp Service** tarafından yayınlanan **JSON** formatlı veri çıktısının, örnek bir **Silverlight** istemcisi tarafından nasıl ele alınabileceğini incelemeye çalışıyor olacağız.

Silverlight tarafında **JSON** içeriği ile çalışabilmek adına geliştirilmiş **JsonArray, JsonObject, JsonPrimitive** gibi tipler bulunmaktadır. Bu tipler sayesinde **JSON** veri kümesinde yer alan **string, number, Boolean** gibi veri türleri kod içerisinde ele alınabilir. Ayrıca tek **JSON** nesnesi veya bir **JSON** nesne listesinin ele alınması da sağlanabilir. Bu geliştiriciler için önemlidir. Nitekim Web ortamında gelen **JSON** içeriğinin **Parse** edilme işlemleri ile uğraşılmasına gerek kalmamaktadır.

Dilerseniz hiç vakit kaybetmeden örnek bir **Silverlight** uygulaması üzerinden ilerlemeye çalışalım. İşe ilk olarak **IIS** üzerinde **host** edeceğimiz **WCF Rest Service Application** projesini ve aşağıdaki kod içeriğine sahip **LogService** servis örneğini geliştirerek başlayabiliriz.

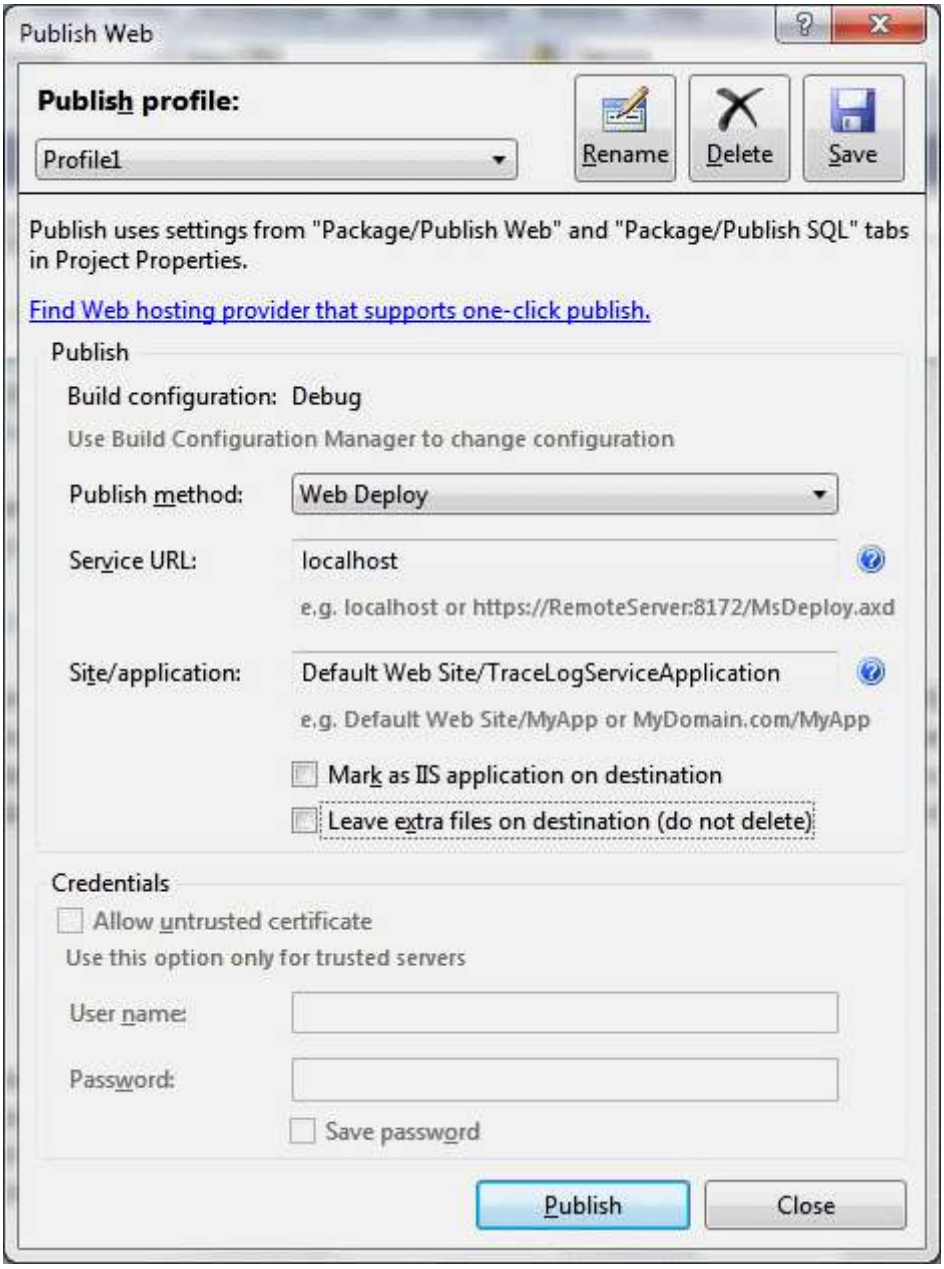
```
using System.Collections.Generic;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;

namespace TraceLogServiceApplication
{
   [ServiceContract(Namespace = "")]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode=InstanceContextMode.PerCall)]
   public class LogService
   {
      [WebGet(UriTemplate =
```

```
"Logs/All",ResponseFormat=WebMessageFormat.Json)]
      public List<Log> GetAllLogs()
      {
         return new List<Log>()
         {
            new Log{ Source="Sql Server", Content="Sql servisi başlatıldı", IsCritical=false,
Level=5},
            new Log{ Source="Sql Server", Content="Sql Agent servisinde hata.",
IsCritical=true, Level=1},
            new Log{ Source="DTC", Content="Dağıtık Transaction nesnesi üretildi.",
IsCritical=false, Level=3},
            new Log{ Source="WF Runtime", Content="Süreç persist edildi",
IsCritical=true, Level=2}
         };
      }
   }

   public class Log
   {
      public string Source { get; set; }
      public string Content { get; set; }
      public int Level { get; set; }
      public bool IsCritical { get; set; }
   }
}
```

LogService içerisinde yer alan **GetAllLogs** isimli servis operasyonu **Log** tipinden bir kaç eleman içeren basit bir **List<Log>** koleksiyonunu geriye döndürmektedir. çalışma zamanında oluşturulacak olan bu içerik, istemci tarafına **JSON** formatında gönderilecektir. Bunun için dikkat edileceği üzere **ResponseFormat** özelliğinin değeri **WebMessageFormat.Json** sabiti olarak belirlenmiştir. Servisimizi bu haliyle test etmek istediğimizde adres satırından **http://localhost:12043/LogService/Logs/All** gibi bir çağrı yapmamız yeterli olacaktır. Bunun sonucunda aşağıdaki **JSON**içeriği üretilecektir.

[{"Content":"Sql servisi başlatıldı","IsCritical":false,"Level":5,"Source":"Sql Server"},{"Content":"Sql Agent servisinde hata.","IsCritical":true,"Level":1,"Source":"Sql Server"},{"Content":"Dağıtık Transaction nesnesi üretildi.","IsCritical":false,"Level":3,"Source":"DTC"},{"Content":"Süreç persist edildi","IsCritical":true,"Level":2,"Source":"WF Runtime"}]

Bu işlemin ardından servisi **IIS** alınta **Publish** etmemiz yeterlidir. **Publish** ayarlarını aşağıdaki resimde görüldüğü gibi belirleyebiliriz.

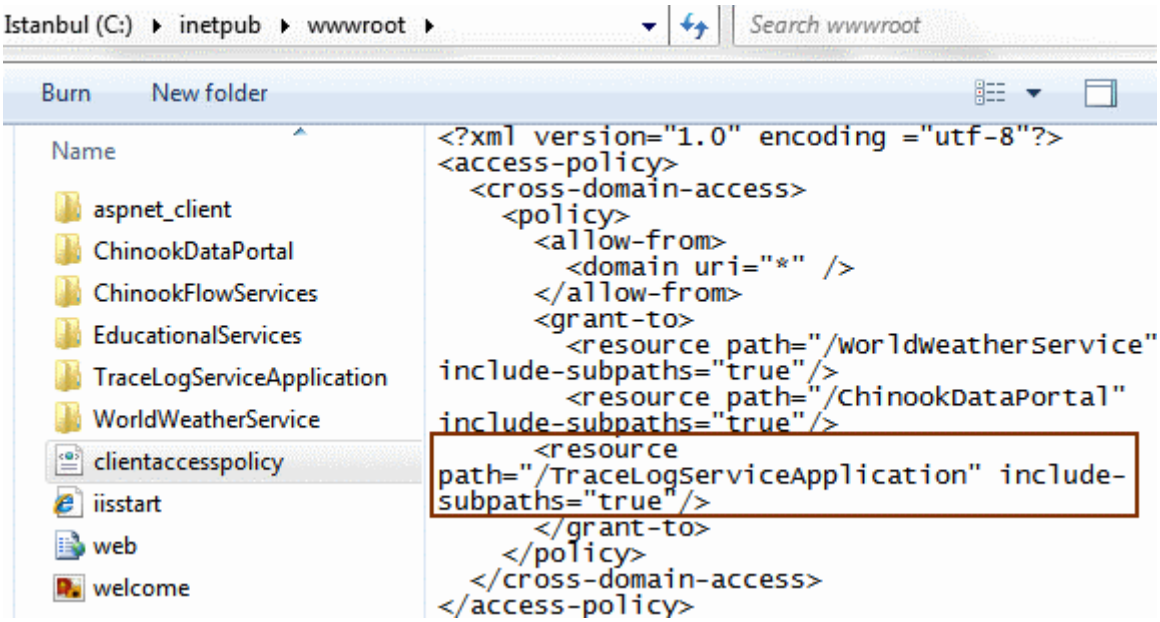


Eğer **Publish** işlemi başarılı olduysa*(**IIS** üzerinden ilgili uygulamanın **Web Application** olarak set edilmesine-Convert to Application seçeneği dikkat ederekten)* herhangibir tarayıcı uygulamadan, **http://localhost/TraceLogServiceApplication/LogService/Logs/All** şeklin de bir çağrıda bulunabiliyor olmamız gerekmektedir ki bu çağrının sonucu olarakta, yukarıdaki **JSON**içeriğine tekrardan ulaşabiliyor olmalıyız.

TraceLogServiceApplication.rar (30,47 kb) **[örnek Visual Studio 2010 Ultimate sürümü üzerinde test edilmiştir]**

Tabi yapmamız gereken bir işlem daha bulunmaktadır. Hatırlayacağınız üzere **Silverlight** istemcileri için **Cross-Domain Policy** sorunsalı mevcuttur. Bu nedenle **IIS** üzerinde daha önceki yazılarda

değindiğimiz **ClientAccessPolicy.xml** dosyasının içeriğini aşağıdaki gibi düzenlememiz ve **TraceLogServiceApplication** için gerekli **garanti haklarını(grant-to)** belirlememiz gerekmektedir.



Artık **Silverlight 4.0** tabanlı istemci uygulamamızı geliştirmeye başlayabiliriz. Bu amaçla, **JsonConsumer** isimli **Silverlight** uygulamamız içerisindeki **MaingPage.xaml** ve kod içerikleri aşağıdaki gibi geliştirilebilir.

MainPage.xaml içeriği;

```
<UserControl x:Class="JsonConsumer.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   mc:Ignorable="d"
   d:DesignHeight="300" d:DesignWidth="400"
xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk">

   <Grid x:Name="LayoutRoot" Background="White">
      <Button Content="Get All Logs" Height="23" HorizontalAlignment="Left"
Margin="24,20,0,0" Name="GetLogsButton" VerticalAlignment="Top" Width="75"
Click="GetLogsButton_Click" />
      <sdk:DataGrid AutoGenerateColumns="True"
ItemsSource="{Binding}" Height="204" HorizontalAlignment="Left"
Margin="24,56,0,0" Name="LogsDataGrid" VerticalAlignment="Top" Width="347"/>
   </Grid>
</UserControl>
```

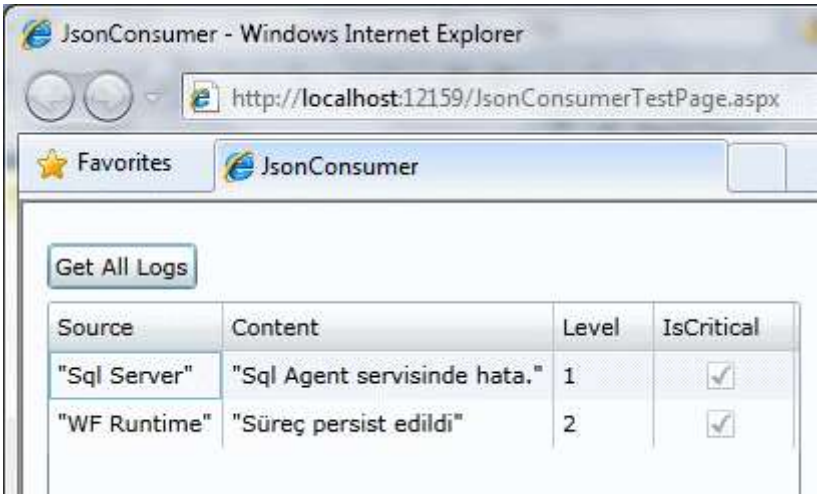
MainPage.xaml.cs içeriği;

```
using System;
using System.IO;
using System.Json;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;

// JsonArray, JsonObject gibi tiplerin kullanılabilmesi için Silverlight projesine
System.Json.dll assembly' ının referans edilmesi gerekmektedir.

namespace JsonConsumer
{
   public partial class MainPage
      : UserControl
   {
      WebClient client = null;

      public MainPage()
      {
         InitializeComponent();

         client= new WebClient();
         client.OpenReadCompleted += new
OpenReadCompletedEventHandler(client_OpenReadCompleted);
      }

      private void GetLogsButton_Click(object sender, RoutedEventArgs e)
      {
         client.OpenReadAsync(new
Uri("http://localhost/TraceLogServiceApplication/LogService/Logs/All"));
      }

      void client_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
      {
         Stream responseStream = e.Result;
         // JsonArray sınıfının static Load metodu, Http Web servisine yapılan talep sonrası
dönen Stream örneğini alır.
         // Load metodu JsonValue tipinden bir referans döndürmektedir ve dizi olarak ele
alabilmek için JsonArray tipine bilinçli bir dönüşüm yapılmıştır.
         JsonArray logs=(JsonArray)JsonArray.Load(responseStream);
         // Elde edilen JSON verisinden IsCritical değeri true olanlar çekilir ve LogInfo
isimli tip içerisinde toplanır.
```

```
            var criticialLogs = from log in logs
                                where log["IsCritical"]
                                select new LogInfo
                                {
                                    Content=log["Content"].ToString(),
                                    Source=log["Source"].ToString(),
                                    Level=log["Level"]
                                };

            // Elde edilen veri kümesi DataGrid kontrolüne veri kaynağı olarak gösterilir
            LogsDataGrid.DataContext = criticialLogs;
        }
    }
    // Servis tarafındaki Log tipinin istemci tarafındaki karşılığı
    public class LogInfo
    {
        public string Source { get; set; }
        public string Content { get; set; }
        public int Level { get; set; }
        public bool IsCritical { get; set; }
    }
}
```

Hatırlayacağınız üzere **WCF WebHttp Service** örneklerine yapılacak olan istemci çağrıları için **WebClient** tipinden yararlanılmaktadır. Bu amaçla **Button** kontrolüne basıldığında, asenkron olarak söz konusu servise bir talepte bulunulmaktadır**(OpenReadAsync)**. Talep sonuçlandığında ise geri bildirim olay metodu devreye girmektedir**(OpenReadCompleted)**. İşte bu olay metodu içerisinde **JSON** veri içeriğinin ele alınması için gerekli işlemler gerçekleştirilmektedir.

Bu metoda ait kod parçasındaki en büyük yardımıcı **JsonArray** tipi ve **Load** fonksiyonudur . Bu fonksiyon, parametre olarak **LogService** isimli **WCF WebHttp Servisine** gönderilen talep sonucu, istemci tarafına indirilen **Stream** referansını kullanmaktadır. Sonuç daha sonradan basit bir **LINQ** sorgusu ile değerlendirilmiş ve örnek olarak kritik seviyedeki log bilgilerinin değerlendirilmesi amaçlanmıştır. Uygulamanın çalışma zamanı görüntüsü aşağıdaki gibi olacaktır.



Görüldüğü gibi **JSON** formatındaki içerik **Silverlight** tarafında başarılı bir şekilde ele alınmış ve **veri bağlı bir kontrol(DataGrid)** ile ilişkilendirilebilmiştir. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

JsonConsumer.rar (1,92 mb) **[örnek Visual Studio 2010 Ultimate sürümü üzerinde test edilmiştir]**

Workflow Foundation Öğreniyorum - Ders 14 - Hani Nerde Asenkron Çalışma Zamanı (2010-08-08T10:05:00)

workflow foundation 4.0,workflowapplication,autoresetevent,asynchronous runtime,

Merhaba Arkadaşlar,

NedirTv?com sponsorluğunda hazırladığımız "Workflow Foundation 4.0 öğreniyorum" görsel eğitim serimizin **14+1=15nci** dersi ile birlikteyiz. Şu ana kadar ki derslerimizde **Workflow** örneklerini çalıştırmak için sadece **WorkflowInvoker** tipinin **static Invoke** metodunu kullandık. Bu senkron çalışma modeli olarak düşünülebilir. Nitekim **Invoke** metodu yardımıyla başlatılan **Workflow** örneği işleyişini tamamlayana kadar, çağırıcı uygulama bir sonraki kod satırına geçiş yapmayacaktır. Ancak **Workflow** örneklerini çalıştıracak olan ve özellikle kullanıcı etkileşimi olan uygulamalarda bu bir sorun teşkil etmektedir. öyleki **Workflow** örneklerinden özellikle uzun süreli olanlar arka planda çalışırken, ana

uygulamanın da işleyişine devam etmesi istenebilir. İşte **WorkflowApplication** sınıfı bu ihtiyaçı karşılamak için kullanılmaktadır. Bakalım nasıl?

Ders 14 - Hani Nerde Asenkron çalışma Zamanı

Süre : 23:22

Dosya Boyutu : 31.5 Mb

örnek : Lesson14.rar

Mp4 Formatında İndirmek İçin Tıklayın

Regex ve Performans İpuçları - Otomatik Cache (2010-08-06T16:05:00)

c#,c# temelleri,c# 4.0,

Merhaba Arkadaşlar,

Şu an yazıyı hazırlamaya çalıştığım an **İstanbul'** un tarihinde gördüğü en sıcak gecelerden birisine denk gelmekte sanırım. Gündüz yaklaşık olarak **53** derece olarak hissedilen sıcaklığı ofisteki kuvvetli klimalar sayesinde fazla hissetmedik belki ama eve dönüş yolunda, gerek otobüslerde gerekse minibüs veya diğer toplu taşıma araçlarında fazlasıyla hissettiğimize eminim

Gece çökmesine ve balkonda oturmama rağmen ne yazık ki yapraklar bile sıcak dolayısıyla kendinden geçmiş durumda ve bu nedenle sallanmak dahi istemiyorlar. Hal böyle olunca serinletici esintilerinde tatile çıktıklarını ifade edebilirim.

Acaba tüm bu yaşadıklarımız, garip olan bu yaz mevsimi, yağmurlarla geçen günler ve aşırı sıcaklar gerçekten de **Küresel Isınmanın**sonuçların mı? Bu konuda dünyadaki 6 derecelik bir ısı değişiminin sonuçlarını anlatan bir kitap okumuştum aslında(6 Derece) Merak edenlere tavsiye ederim.

Neyse. Dilerseniz biz konumuza geri dönelim. Bu yazımızda belki de tek satırlık bir kod parçasının önemine değiniyor olacağız. Ancak sonuçları irdelediğimizde bunun ne kadar önemli bir fark yarattığına da şahit olacağız. Konumuz **Regex** tipinin kullanımına dair ip uçlarından birisi olan otomatik ön bellekleme işlemini ele almakta.

Aslında **Regular Expression** terimini ağırlıklı olarak **Asp.Net Web** uygulamalarından tanımaktayız. Bu anlamda özellikle **RegularExpressionValidator** web kontrolünden

yararlanarak, girilen verinin doğrulanması için bazı desenleri kullanabiliyoruz. Bilindiği üzere bu doğrulama işlemleri **Javascript** ile istemci tarafında ve her ihtimale karşın sunucu tarafında da uygulanmakta*(İstemcinin javascript çalıştırmama olasılığına karşın)*. Tabi işin güzel yanı **Regex** ifadelerinin aslında dilden bağımsız olmaları. **RegularExpressionValidator**kontrolünün **ValidationExpression** özelliğinde yer alan desenlerden bazılarını aşağıda bulabilirsiniz.

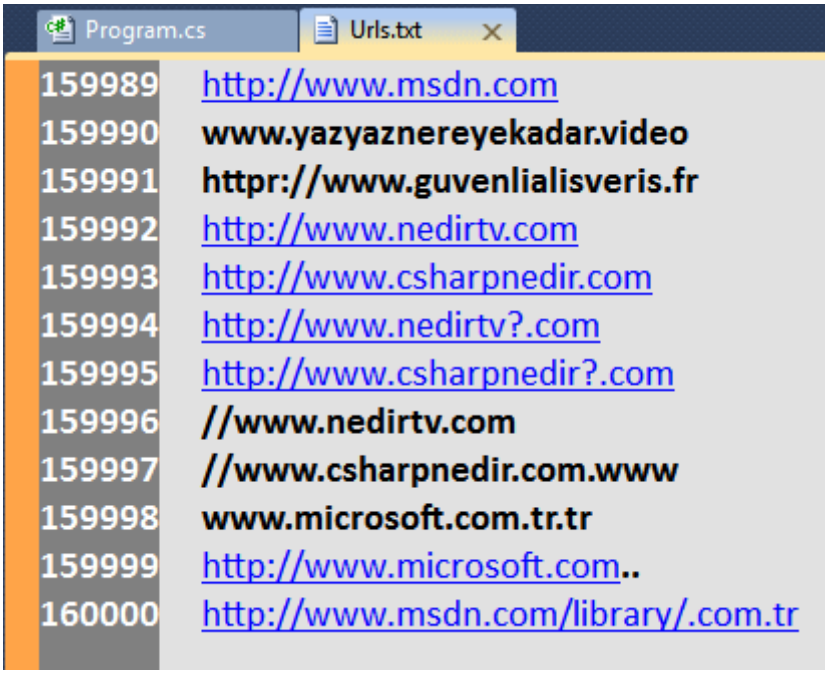
- Internet Email Adres Deseni \w+([-+.']\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*
- Internet URL Deseni http(s)?://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?
- US Phone Number ((\(\d{3}\) ?)|(\d{3}-))?\d{3}-\d{4}
- US Social Security Number \d{3}-\d{2}-\d{4}
- US Zip Code \d{5}(-\d{4})?
- German Phone Number ((\(0\d\d\) |(\(0\d{3}\))?\d)?\d\d \d\d \d\d|\(0\d{4}\) \d \d\d-\d\d?)

Bu desenler yardımıyla kullanıcıların girmiş olduğu verilerin geçerli bir elektronik posta/url adresi, telefon numarası, posta kodu, sosyal güvenlik bilgisi olup olmadığı kolaylıkla kontrol edilebilmekte. Tabi dilersek özel **Regex** ifadeleri de oluşturabiliriz.

Diğer taraftan bazı projelerde **Doğrulama(Validation)** operasyonlarını içeren katmanlarda, **Regex** tipinden yararlanarak verinin kontrol edilmesi işlemleri gerçekleştirilebilmektedir. Bu anlamda**Regex** tipi ve üyeleri bize önemli avantajlar sağlamakta. Ancak **Regex** tipinden yararlanırken zaman zaman performans sorunları ile karşılaşılabilir. Bu noktada ön bellekleme işlemlerinin hızlanma açısından bir avantaj sağladığı ortadadır.

Aslında **.Net Framework 1.1** versiyonunda yer alan **Regex** tipinin nesne örneklemelerinin, **desen(Pattern)** ile ilişkili bir ön bellekleme mekanizması zaten mevcuttur. Ancak **.Net Framework 2.0** ve sonraki versiyonlarda söz konusu desen ön bellekleme işlemi **static IsMatch** metodu üzerine yıkılmıştır. Bir başka deyişle sadece **IsMatch** metodunun, parametre olarak gelen **Regular Expression** ifadesi için ön bellekleme yaptığını ifade edebiliriz. Normal şartlarda kaç desenin ön bellekleneceği bilgisi **Regex** tipinin **static CacheSize özelliği(Property)** ile belirlenebilir. Bu özelliğin varsayılan değeri ise **15** dir.

Şimdi basit bir test uygulaması geliştireceğiz ve aslında ön belleklemenin nasıl bir faydası olduğunu görmeye gayret edeceğiz. Bu amaçla **160bin** satırdan oluşan ve aşağıdaki gibi bazı **URL** adres bilgilerini içeren*(ki çoğu aynı değerlerin tekrarıdı)* **Urls.txt** isimli bir **Text** dosyasını göz önüne alıyor olacağız. Söz konusu dosya içeriğinin bir kısmını aşağıdaki şekilden görebilirsiniz.

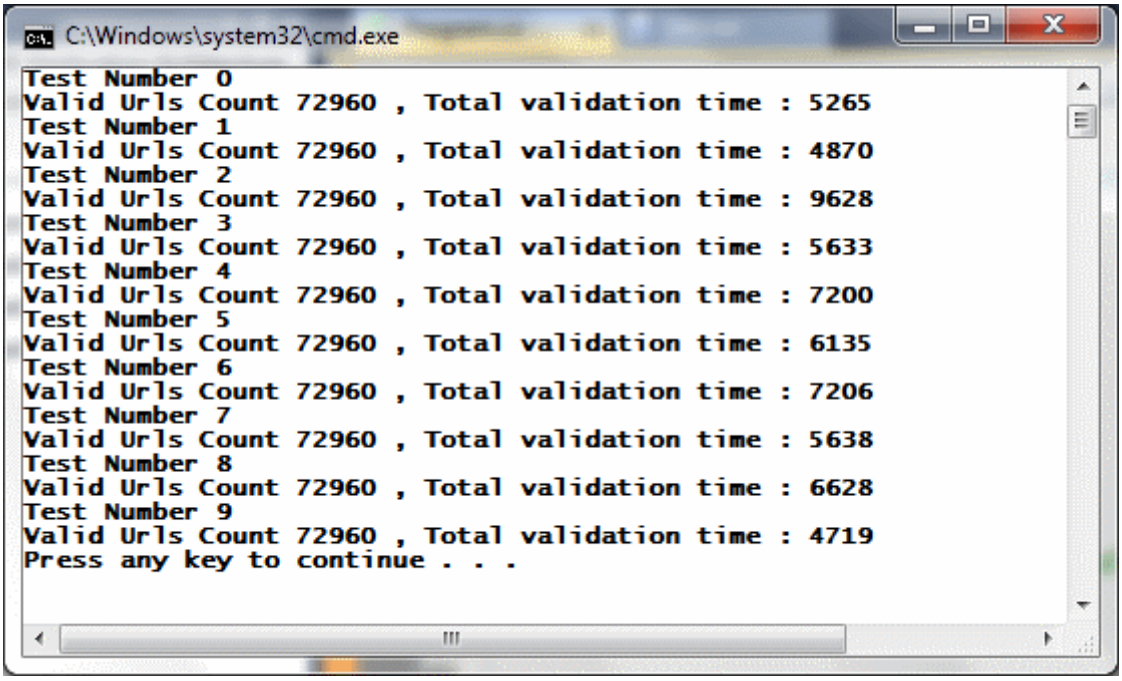


Uygulama içerisinde yer alan akışımız, söz konusu **Text** dosya içeriğindeki her bir satırı okuyacak ve bu adreslerden hangilerinin geçerli olduğunu kontrol edecek şekilde geliştirilecektir. İşte bu amaçla ele alacağımız örnek uygulama kodları;

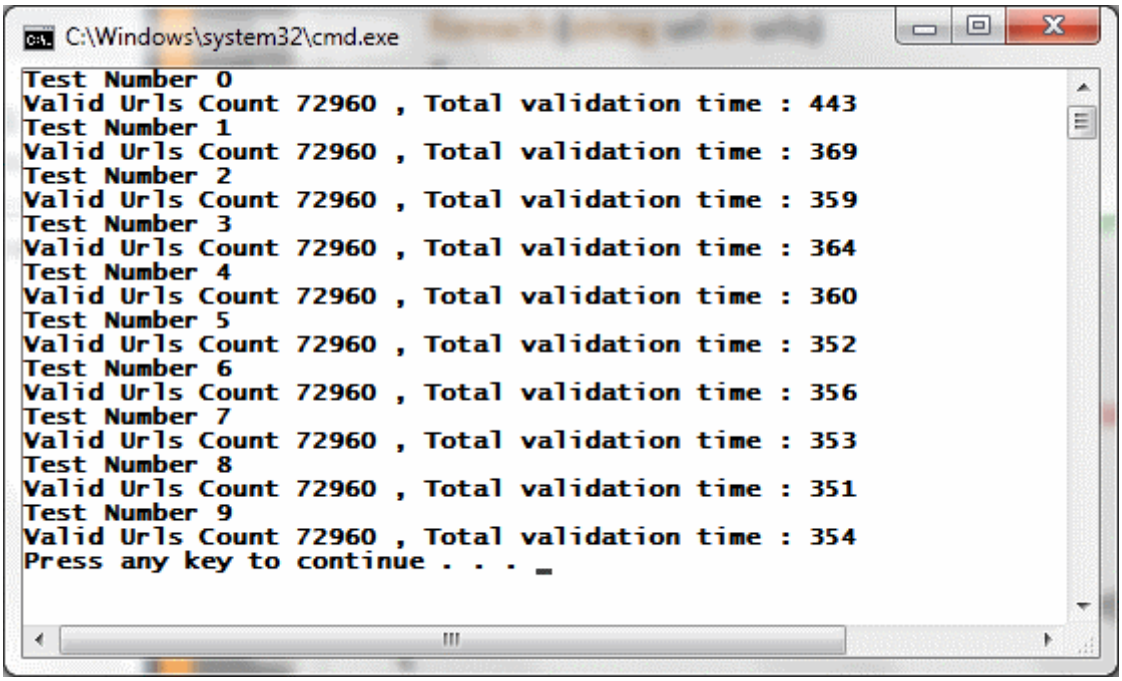
```
using System;
using System.Diagnostics;
using System.IO;
using System.Text.RegularExpressions;

namespace RegExPerformanceTips
{
   class Program
   {
      static void Main(string[] args)
      {
         string urlPattern=@"http(s)?://([\w-]+\.)+[\w-]+(/[\w- ./?%&=]*)?";
         string[] urls =
File.ReadAllLines(Path.Combine(Environment.CurrentDirectory,
"Urls.txt"));
         for (int i = 0; i < 10; i++)
         {
            Console.WriteLine("Test Number {0}",i);
            Process(urlPattern, urls);
         }
      }

      private static void Process(string urlPattern, string[] urls)
      {
```

```
        Stopwatch watcher = new Stopwatch();
        watcher.Start();
        int validCount = 0;

        foreach (string url in urls)
        {
           if (IsValid(urlPattern, url))
              validCount++;
           //Console.WriteLine("{0} {1}",url,IsValid(urlPattern,url)?"is valid":"isn't
valid");
        }

        watcher.Stop();
        Console.WriteLine("Valid Urls Count {0} , Total validation time : {1}",
validCount, watcher.ElapsedMilliseconds);
     }

     static bool IsValid(string pattern,string content)
     {
        Regex regex = new Regex(pattern);
        bool result = regex.IsMatch(content);
        //bool result = Regex.IsMatch(content, pattern);
        return result;
     }
  }
}
```

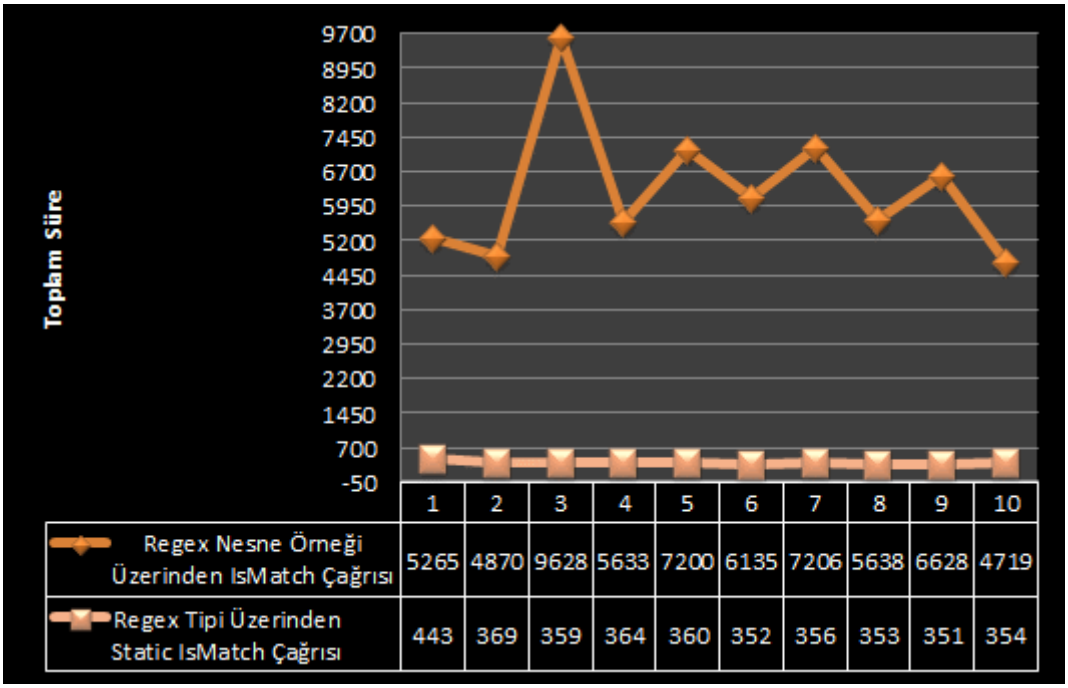
Kodun dikkat edilmesi gereken en önemli noktası **IsValid** metodunun içeriğidir. Söz konusu metod kontrol edilecek **string** bilgi ile ilgili **Regular Expression** desenini parametre olarak almaktadır. İlk test vakası için **Regex** tipinden bir nesne örneğinin oluşturulduğu görülmektedir. **Regex** nesnesi örneklenirken parametre olarak kontrol desenini almaktadır. Sonrasında ise nesne örneği üzerinden yapılan **IsMatch** metod çağrısı ile gerekli kontrol işlemi gerçekleştirilmektedir. Dosya içerisindeki bilgiler için gerekli doğrulama kontrolü süre farklılıklarını irdelemek amacıyla **10** defa üst üste yapılmaktadır. Buna göre ilk test vakamızın çalışma zamanı çıktısı uygulamanın yazıldığı sistemin özelliklerine göre aşağıdaki gibidir.



Yaklaşık olarak **4** ile **9 saniye** arasında değişen süreler söz konusudur. Sürelerin tutarsızlığı bir yana, kontrol işlemlerinin de oldukça da uzun sürdüğü görülmüştür. Peki uzun sürdükleri sonucuna nasıl vardık? Eğer **IsValid** metodu içerisinde **Regex** tipi üzerinden **static IsMatch** metodunu kullandığımız durumda ki çalışma zamanı çıktısına bakarsak, çok daha kısa sürelerde doğrulama işlemlerinin yapıldığını görebiliriz. İşte aynı makine konfigurasyonundaki yeni test vakasının çalışma zamanı sonuçları.



Buna göre test sonuçlarını aşağıdaki **Excel** grafiğinde görüldüğü gibi özetleyebiliriz.



Peki bu korkunç sayılabilecek süre farkı neden oluşmuştur. Yazımızın giriş kısmında da belirttiğimiz üzere **Regex** tipinden nesne örnekleri oluşturulduğunda her seferinde bir desen bildirimi yapılmaktadır ancak bu bildirim sürekli olarak tekrar edilmektedir.
Oysaki **Regex** tipinin **static IsMatch** metodu ilk seferde
kullanılan **Regular Expression** ifadesini ön belleklemekte ve sonraki çağrılarda da bu desen için gerekli iç hazırlıkları yapmadan sadece kontrol işlemine geçiş yapmaktadır. Tabi istenirse varsayılan olarak **15** farklı **Regular Expression** ifadesi için bu bellekleme işlemi kullanılabilir.

Regex tipinin performanslı kullanımına ilişkin farklı ip uçları da mevcuttur. Ancak sucuk gibi terldiğimden bu sıcakta ancak bu kadar yazabildim. Söz konusu ipuçlarını ilereyen yazılarımızda ele almaya çalışıyor olacağız. Umarım havalar buna müsade eder. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

RegExPerformanceTips.rar (33,72 kb) **[örnek Visual Studio 2010 Ultimate Ortamında Geliştirilmiş ve Test Edilmiştir]**

Parent-Child Task Exception Durumları (2010-08-03T09:10:00)

tpl,task parallel library,parallel programming,parallel computing,

 Merhaba Arkadaşlar,

Daha orta okul sıralarındayken havacılığa karşı müthiş bir ilgim vardı. Hiç unutmuyorum o yıllarda **Uçan Türk** dergisinin sıkı bir fanatiğiydim. Pek çok savaş uçağının teknik özelliklerini ezbere bilirdim ve hatta onları arşivlediğim bir not defterim dahi vardı. Uçmaktan korkan birisi olmama rağmen bunu yeneceğimi düşünerekten **Lise** yıllarında **Hava Harp Okuluna** girebilmek için özel bir çalışma programı bile uygulamıştım. Düzenli olarak spor yapıyor, kondisyon arttırmaya çalışıyor, günde değil 3, 5 kere dişlerimi fırçalıyor, gözlerimi yormamak için uykuma özen gösteriyordum.

Tabi öğrenciliğim çok parlak olmadığı için **öSS** sınavında aşmam gereken **150**' lik puan barajı konusunda tereddütler yaşıyordum. Nitekim barajı da geçemedim. Hayallerim yıkılmış mıydı? Elbette hayır. Heleki o yıllardaki çalışma azmimim bana kazandırdığı önemli avantajlar olduğu düşünüldüğünde. Bunlardan birisi de derin detaylara inebilmek için gerekli eforu, gayreti gösterme isteğidir. Neden böyle bir giriş yaptığıma gelince...Bu seferki konumuz **Paralel Programlamada**, ilişkisel **Task** örneklerinin **Exception** yönetimi hakkındadır. Konu sıkıcı ve bir o kadarda detaylıdır. Ama neyseki araştırıp, sıkılmadan derinlerine inmek ve analiz etmek için gerekli gayret mevcut 😉

Hatırlayacağınız üzere Parent-Child Tasks Kavramı başlıklı yazımızda **.Net Framework 4.0** tarafında paralel programlamada önemli bir yere sahip olan**Task** örnekleri arasındaki **Parent**, **Child** ilişkiyi incelemeye çalışmıştık. **Parent-Task** nesne örnekleri arasındaki ilişkilerde bilinmesi gereken konulardan birisi de, istisnaların nasıl ele alındığıdır**(Exception Handling)**. Aslında konuya hızlı bir giriş yaparak ilerlememiz şu aşamada avantajımız olacaktır. Bu nedenle **Visual Studio 2010 Ultimate** ortamında geliştireceğimiz **Console** uygulamasında aşağıdaki kod içeriğinin yer aldığını göz önüne alalım.

```
using System;
using System.Data.SqlClient;
```

```
using System.IO;
using System.Threading;
using System.Threading.Tasks;

namespace ParentChildTasksExceptionHandling
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Case 1

         Task parent = Task.Factory.StartNew(() =>
         {
            Console.WriteLine("Parent task...");

            Task child1 = Task.Factory.StartNew(() =>
               {
                  Console.WriteLine("Child Task 1 başladı...");
                  Thread.Sleep(5000);
                  Console.WriteLine("Child Task 1 bitti...");
               }
            , TaskCreationOptions.AttachedToParent);
            Task child2 = Task.Factory.StartNew(() =>
            {
               Console.WriteLine("Child Task 2 başladı...");
               FileStream stream=File.OpenRead("OlmayanDosya.txt");
               Console.WriteLine("Child Task 2 bitti...");
            }
            , TaskCreationOptions.AttachedToParent);

            Task child3 = Task.Factory.StartNew(() =>
            {
               Console.WriteLine("Child Task 3 başladı...");
               Thread.Sleep(3000);
               Console.WriteLine("Child Task 3 bitti...");
            }
            , TaskCreationOptions.AttachedToParent);

            Task child4 = Task.Factory.StartNew(() =>
            {
               Console.WriteLine("Child Task 4 başladı...");
               Thread.Sleep(10000);
               SqlConnection conn = new SqlConnection();
               conn.Open();
```

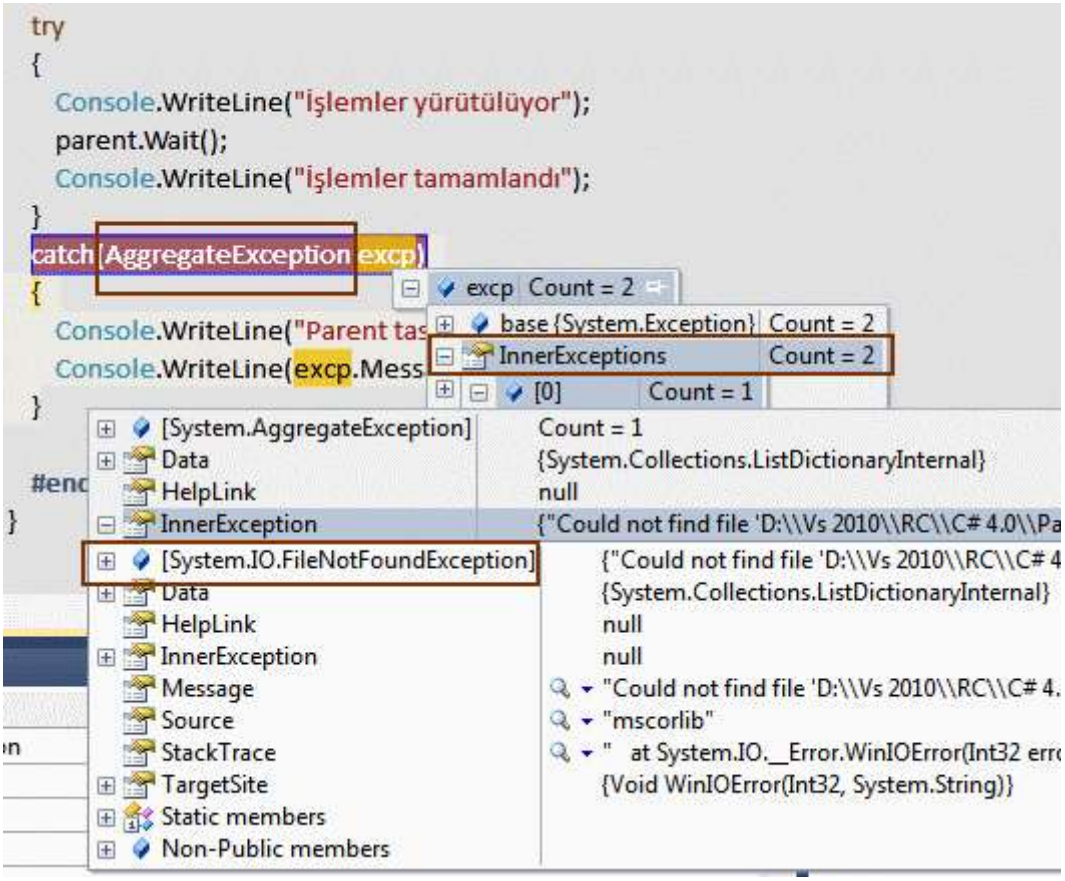
```
            Console.WriteLine("Child Task 4 bitti...");
        }
        , TaskCreationOptions.AttachedToParent);
    }
    );

    try
    {
        Console.WriteLine("İşlemler yürütülüyor");
        parent.Wait();
        Console.WriteLine("İşlemler tamamlandı");
    }
    catch(AggregateException excp)
    {
        Console.WriteLine("Parent task durumu {0}",parent.Status);
        Console.WriteLine(excp.Message);
        foreach (var innerExcp in excp.InnerExceptions)
        {
            Console.WriteLine(innerExcp.InnerException.Message);
        }
    }

    #endregion
  }
 }
}
```

Buradaki kod parçasında bir **Parent Task** ve bunun içerisinde çalışacak şekilde planlanan 4 farklı **Child Task** örneği yer almaktadır. Bu örneklerden birisi, olmayan bir dosyayı açmaya çalışmaktadır. Dolayısıyla **FileNotFoundException** tipinden bir istisna nesnesi üreteceği garantidir. Diğer bir **Task** ise parametresiz bir **SqlConnection** nesnesi oluşturmakta ve belli olmayan bir yere doğru **SQL** bağlantısı açmaya çalışmaktadır ki bu da **InvalidOperationException** türünden bir istisna nesnesinin fırlatılmasına neden olacaktır.

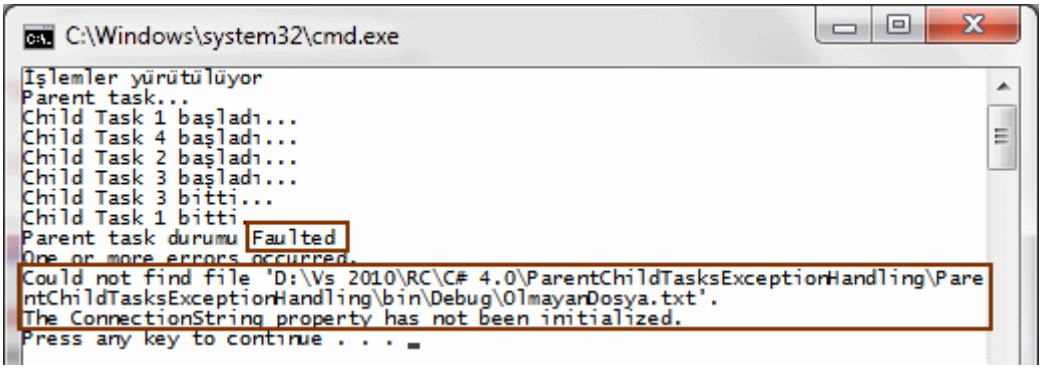
Daha önceki yazımızdan hatırlayacağınız üzere **Child Task** örneklerinin başlattığı metod gövdeleri içerisinde oluşabilecek olan **istisnalar(Exception), Parent Task** örneğinin**Final State** durumunu da doğrudan etkilemektedir. Ayrıca **Child Task**' lerde bir istisna oluşsa bile, diğer **Task** örnekleri çalışmalarını devam ettirmektedir. örnek kod parçamızda **Parent Task** tarafına çıkartılan **Exception** örneklerinin yakalanabilmesi amacıyla **try...catch** bloğuna başvurulduğu görülmektedir.

Burada dikkat edilmesi gereken iki nokta vardır. Bunlardan birisi **try** bloğu içerisinde **Parent Task** nesne örneği üzerinden bilinçli olarak **Wait** metodunu kullanılmış olmasıdır. Bu şekilde uygulamanın ana **Thread** bloğu sonlanmadan **Parent Task** ve

içeriğinin işlerinin tamamlanmasının beklenmesi garanti edilmiş olmaktadır. Diğer yandan ikinci önemli nokta **catch** bloğu içerisinde, **Parent Task** örneği altında çalışan **Child Task**' ler tarafından fırlatılan **Exception** nesnelerinin nasıl yakalandığıdır.
Burada **AggregateException** sınıfının**InnerExceptions** isimli koleksiyonu üzerinden hareket edildiğine dikkat edilmelidir. Uygulama kodu **Debug** modda çalıştırıldığında aşağıdaki sonuçların elde edildiği görülecektir.



Görüldüğü gibi **AggregateException** nesnesi **Child Task**' lerde oluşan istisnaları bir koleksiyon dahilinde saklamaktadır. **Count** değerinin iki dönmesinin sebebi tahmin edeceğiniz üzere iki**Child Task**' in **Exception** fırlatmış olmasıdır. Uygulamamızın çalışma zamanı görüntüsü ise aşağıdaki gibi olacaktır.



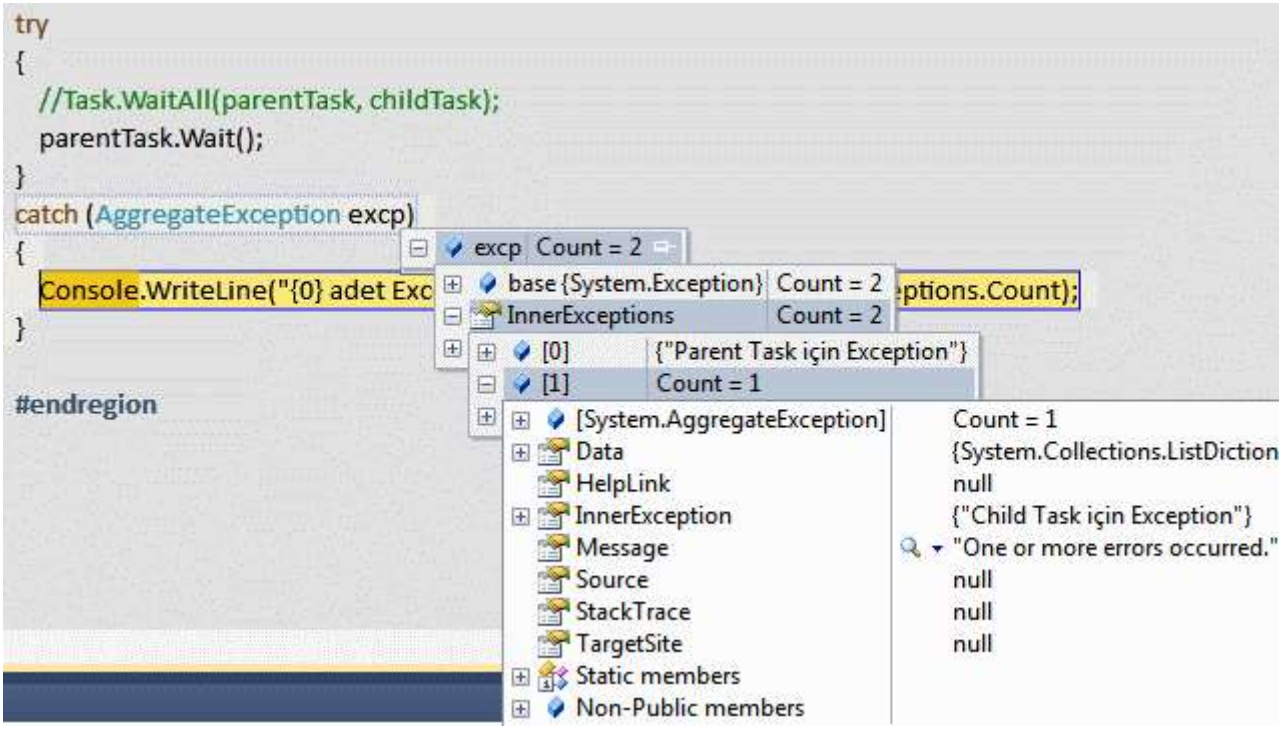
Dikkat edileceği üzere **Parent Task** örneğinin **Final State** durumundaki karşılığı **Faulted** olmuştur ki bu son derece doğaldır. çünkü **Child Task** örneklerinden ikisi **Exception**üretimi bildirmiştir. Bu istisna bildirimlerinin **Child Task**' lerden, **Parent Task**' e bildirildiğini ve **AggregateException** içerisinde toplandıklarını da unutmamak gerekir.

Exception yönetimi ile ilişkili tek durum bu değildir. Şimdi de aşağıdaki kod örneğini göz önüne alalım.

```
using System;
using System.Data.SqlClient;
using System.IO;
using System.Threading;
using System.Threading.Tasks;
using System.Collections.Generic;

namespace ParentChildTasksExceptionHandling
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Case 2

         Task parentTask = null;
         Task childTask = null;
         ManualResetEvent mre = new ManualResetEvent(false);

         parentTask = Task.Factory.StartNew(() =>
            {
               childTask = Task.Factory.StartNew(() =>
                  {
                     throw new Exception("Child Task için Exception");
                  }, TaskCreationOptions.AttachedToParent);
               mre.Set();
               throw new Exception("Parent Task için Exception");
            }
          );

         mre.WaitOne();

         try
         {
            //Task.WaitAll(parentTask, childTask);
            parentTask.Wait();
```

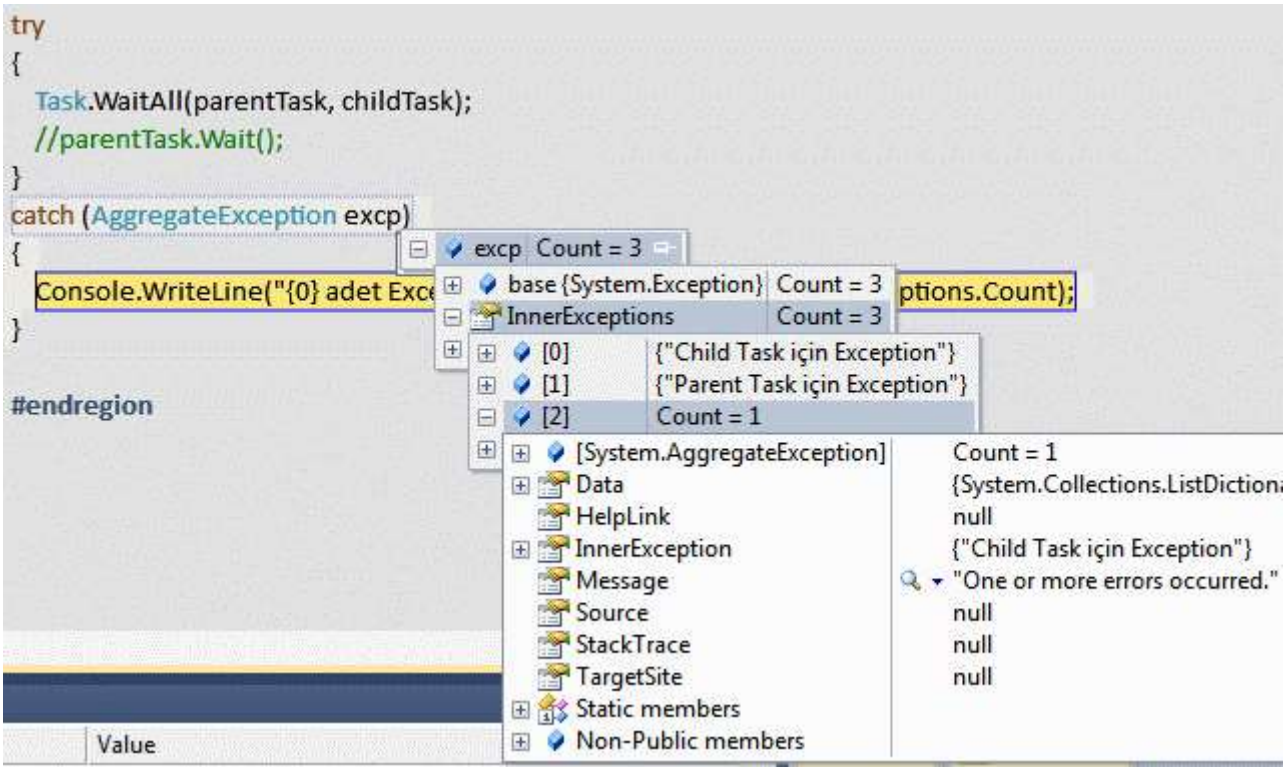
```
        }
        catch (AggregateException excp)
        {
           Console.WriteLine("{0} adet Exception söz
konusudur",excp.InnerExceptions.Count);
        }

        #endregion
      }
   }
}
```

Bu kod parçasında iki adet **Task** örneği bulunmaktadır ve aralarında **Parent-Child** ilişki söz konusudur. Dikkat edilmesi gereken nokta ise, kodun bu haliyle **debug** edilmesi veya çalıştırılması sonrasında **AggregateException** nesne örneği üzerinden yaklanan dahili istisna örnekleri ve toplam sayısıdır. **parentTask.Wait();** metod çağrısı kullanıldığında **debug**modda iken aşağıdaki sonuçlar ulaşılabildiğim gözlemlenecektir.



Aslında beklediğimiz gibi bir sonuç söz konusudur. **parentTask** nesne örneği üzerinden **Wait** metodu kullanıldığı için **AggregateException** nesne örneğinin **InnerExceptions**koleksiyonunun ilk elemanı **Parent Task** örneğinin çalıştırdığı metod içerisinden fırlatılan istisna bilgisini içermektedir. **InnerExceptions** özelliğinin 1 numaralı indis değeri içinse,**Child Task** içerisinden üretilen **Exception** söz konusudur. Ancak hem **Parent Task** hemde içerdiği **Child Task** nesne örnekleri için **Wait** işlemi gerçekleştirilirse? Yani yorum satırı

açılıp **Task.WaitAll(parentTask,childTask)** kullanılırsa, bu durumda **Debug** modda aşağıdaki sonuçlar ile karşılaşırız.



Dikkat edileceği üzere **AggregateException, 3** adet **InnerException** içerdiğini bildirmektedir. Her ne kadar **throw new** satırlarının sayısı iki olsa da **3** sonuç döndürülmüştür. Yakından bakıldığında ise durum aslında şu şekilde özetlenebilir; **AggregateException, WaitAll** çağrısı nedeniyle ne kadar **Task** örneği varsa bunların tamamının ürettiği**Exception** bilgilerini*(parentTask' in ürettiği ve yukarıya gönderdiği Exception dahil)* **InnerExceptions** altında toplamıştır.

Bunlara ilaveten **Parent Task** örneği içerisindeki **Child Task** örneğinden fırlatılan **Exception,** ayrıca **InnerExceptions** içerisindeki **2** numaralı indise atanmıştır. Dolayısıyla birden fazla**Task** örneğinin **WaitAll** ile beklenmesi halinde **AggregateException** nesnesinin istisna toplama yaklaşımı, sadece **Parent Task** örneğine uygulanan **Wait** metodu söz konusu olduğundakinden farklıdır. Bu çok tabi olarak üst tarafta Exception örneklerinin yakalanıp değerlendirildiği konumlarda önem kazanan küçük bir farktır.

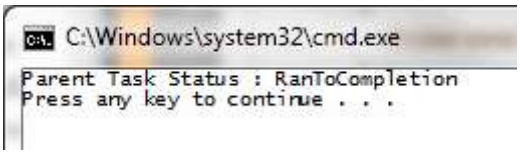
Şu ana kadar yaptıklarımızı değerlendirdiğimizde **Child Task** tamamlandığında eğer bir **Exception** içeriyorsa bunu **Parent Task**' e bildirdiği yönündedir.

Exception nesnelerinin ele alınması ile ilişkili bir diğer yaklaşımı ise aşağıdaki kod parçasından devam ederek değerlendirebiliriz.

```
using System;
using System.Threading;
```

```
using System.Threading.Tasks;

namespace ParentChildTasksExceptionHandling
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Case 3

         Task parentTask = null;
         Task childTask = null;

         parentTask = Task.Factory.StartNew(() =>
         {
            childTask = Task.Factory.StartNew(() =>
            {
               throw new Exception("Child Task için Exception");
            }, TaskCreationOptions.AttachedToParent);

            try
            {
               childTask.Wait();
            }
            catch
            {
            }
         }
          );

         try
         {
            parentTask.Wait();
            Console.WriteLine("Parent Task Status : {0}",parentTask.Status);
         }
         catch (AggregateException excp)
         {
            Console.WriteLine("{0} adet Exception söz konusudur",
excp.InnerExceptions.Count);
         }

         #endregion
      }
   }
}
```

Bu sefer **Parent Task** için açılan kod içerisinde **Child Task** örneği için **Wait** metodu çağrısı yapılmış ve söz konusu çağrı sırasında bir **Exception** oluşrsa, boş bir catch bloğunda yakalanmıştır. Dikkat çekici nokta da zaten burasıdır. Boş **catch** bloğu içerisinde herhangibir şekilde **Exception** nesne örneği fırlatılmadığından, **Child Task** tarafından üretilen istisna, **Parent Task** örneğine bildirilmemektedir. Bir başka deyişle Child Task örneğinin ürettiği istisna kendi içerisinde ele alınarak süpürülmüştür. Dolayısıyla bu örnek kod parçasının çalışma zamanı çıktısı aşağıdaki gibi olacak, bir başka deyişle, **Parent Task** nesne örneğinin **Final State** durumu **RanToCompletion** olarak belirlenecektir.

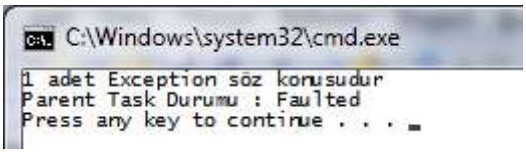


Şimdi burada durup önemli bir noktayı vurgulamak gerektiği düşüncesindeyim. **WaitAll** metodunu kullandığımız örnekte **Child Task** ve **Parent Task** örneklerinin durumları **Faulted** olarak set edilmektedir. Ancak son senaryoda **Parent Task** örneğine **Child Task** içerisinde fırlatılan **Exception** bildirilmediğinden sadece **Child Task**, **Faulted** durumuna düşecek ve **Parent Task**,**RanToCompletion** modunda olacaktır. Bunun belirgin olan sebebi az öncede belirttiğimiz üzere, **Child Task** içerisinden fırlatılan istisnanın **Parent Task**' e çıkartılmadan bir **try...catch** bloğu ile kontrol altına alınmasıdır. Ancak yine son senaryoda aşağıdaki gibi bir kullanım ile **Parent Task**' e **Child Task**' ten fırlatılan **Exception** durumunun bildirilmesi sağlanabilir.

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace ParentChildTasksExceptionHandling
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Case 4

         Task parentTask = null;
         Task childTask = null;

         parentTask = Task.Factory.StartNew(() =>
         {
            childTask = Task.Factory.StartNew(() =>
            {
               throw new Exception("Child Task için Exception");
            }, TaskCreationOptions.AttachedToParent);
```

```
            childTask.ContinueWith(_ =>
            {
               try
               {
                  childTask.Wait();
               }
               catch(AggregateException excp)
               {
               }
            }
            );
         }
         );

         try
         {
            parentTask.Wait();
            Console.WriteLine("Parent Task Status : {0}", parentTask.Status);
         }
         catch (AggregateException excp)
         {
            Console.WriteLine("{0} adet Exception söz konusudur\nParent Task Durumu :
{1}", excp.InnerExceptions.Count,parentTask.Status);
         }

         #endregion
      }
   }
}
```

Bu kod parçasında **Child Task** nesne örneği üzerinden **Continue** metodu kullanılmış ve **try...catch** bloğu ile alt task örneğinin ürettiği istisnanın yine aynı blok içerisinde ele alınması sağlanmıştır. Bu durumda **Parent Task** yine **Child Task** içerisinden üretilen **Exception** nesnesi için bilgilendirilecektir, üstelik **Continue** bloğu içerisindeki **catch** bloğundan dışarıya doğru bir **Exception** fırlatımı açık bir şekilde yapılmasa bile. Tabi bu durumda **Parent Task** nesne örneğinin **Final State** durumu yine **Faulted** olacaktır. İşte çalışma zamanı çıktısı.



Görüldüğü gibi **Parent-Child Task** diyerek geçmemek lazım Ele alınması gereken bir kaç durum söz konusu ki bunlardan birisi de iptal **işlemleri(Cancellation)**. Bu konuyu da bir sonraki yazımızda ele almaya çalışıyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ParentChildTasksExceptionHandling.rar (25,96 kb) **[örnek Visual Studio 2010 Ultimate sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Workflow Foundation Öğreniyorum - Ders 13 - Workflow Service için İstemci Geliştirmek (2010-07-30T13:30:00)

workflow foundation 4.0,workflow foundation,workflow services,iis,visual studio 2010,

Merhaba Arkadaşlar,

NedirTv?com desteğinde sürdürdüğümüz **"Workflow Foundation 4.0 öğreniyorum"** görsel eğitim serimizin **14ncü dersi ile(13+1)** karşınızdayız. Bir önceki dersimizde **Workflow Service** örneklerinin nasıl geliştirilebileceğini incelemeye çalışmıştık. Bu dersimizde ise ilk olarak, bir**Workflow Service** uygulamasını **Internet Information Services(IIS)** altına **Publish** ediyor olacağız. Sonrasında ise, **.Net Framework 4.0** tabanlı **uygulama havuzu(Application Pool)** altında **Host** edilen bu servis için basit bir **WinForms** istemcisi geliştireceğiz. Bu sayede **Workflow Service** içerisinde yer alan iş akışı mantıklarının, servis bazlı olarak bir istemci uygulama tarafından nasıl kullanılabileceğini de anlayacağız. Geliştireceğimiz istemci tarafında otomatik olarak üretilen **proxy** tipini kullanacak ve ilgili servis operasyonları için hem senkron hemde asenkron çağrıların nasıl tasarlanabileceğini öğreneceğiz. İyi seyirler dilerim.

Ders 13 - Workflow Service için İstemci Geliştirmek

Süre : 23:55

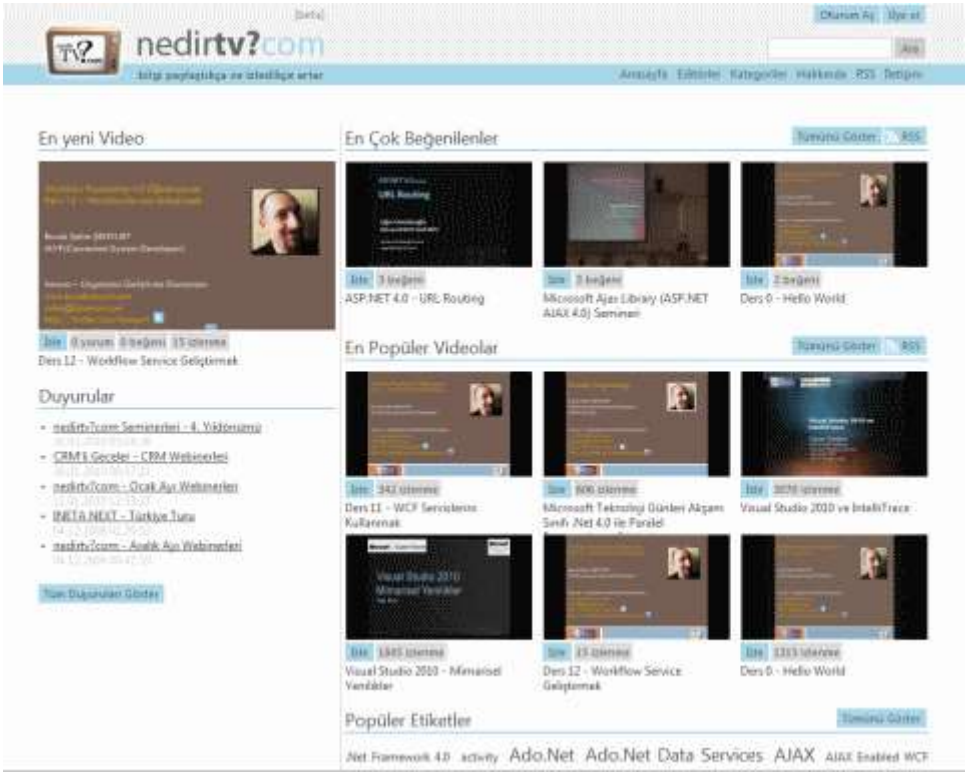
Dosya Boyutu : 32.3 Mb

örnek : Lesson13.rar (84,97 kb)

Mp4 Formatında İndirmek İçin Tıklayın

NedirTv?com Yeni Arayüzü ile Yayında (2010-07-23T20:45:00)

nedirtv?com,nedirtv,



Merhaba Arkadaşlar,

Yaklaşık 2 hafta önce NedirTv?com' u yeni arayüzü ile yayına aldık. Daha kullanışlı ve hızlı arayüzümüzle daha fazla yazılımcıya ulaşmayı umuyoruz.

Yeni arayüzü hazırlayıp kodlayan ve bu iş için büyük emek harcayan **Alper özçetin** ile **Alp çoker**' e de teşekkürlerimizi iletiyoruz. Ayrıca **Daron Yöndem** ve **Muammer Benzeş**' e de yardımlarından dolayı müteşekkiriz.

NedirTv?com ailesi ve değerli takipçilerine hayırlı olması dileğiyle hepinize mutlu seyirler.

Workflow Foundation Öğreniyorum - Ders 12 - Workflow Service Geliştirmek (2010-07-23T09:30:00)

workflow foundation 4.0,workflow foundation,visual studio 2010,wcf,workflow service,wcftestclient,windows communication foundation,

Merhaba Arkadaşlar,

NedirTv?com sponsorluğunda devam ettirdiğimiz **"Workflow Foundation 4.0 öğreniyorum"** serimizin **on üçüncü(12+1)** dersi ile karşınızdayız. Bu dersimizde Workflow örneklerinin servis bazlı olarak dış dünyaya nasıl sunulabileceklerini incelemeye çalışıyor olacağız. Bir başka deyişle**Workflow Service** örneklerini irdeleyeceğiz.

WCF Eco System' in de önemli bir parçası olan ve doğal olarak **Windows Communication Foundation** alt yapısı üzerinde konuşlandırılan **Workflow Service**' ler sayesinde, iş akışlarını servis olarak istemcilere açmamız mümkün hale gelmektedir. İlk dersimizde **Asp.Net Development Server**üzerinden **WcfTestClient** aracı yardımıyla test edeceğimiz **Workflow Service**' lerini, ilerleyen derslerimizde **IIS(Internet Information Services)** ortamı üzerinden yayınlayarak daha geniş bir dünyaya adım atmaya çalışıyor olacağız. Şimdilik ilk bebek adımımız atalım da sonradan gerisi gelir

Ders 12 - Workflow Service Geliştirmek

Süre : 24:41

Dosya Boyutu : 32.8 Mb

örnek : Lesson12.rar (15,41 kb)

Mp4 Formatında İndirmek İçin Tıklayın

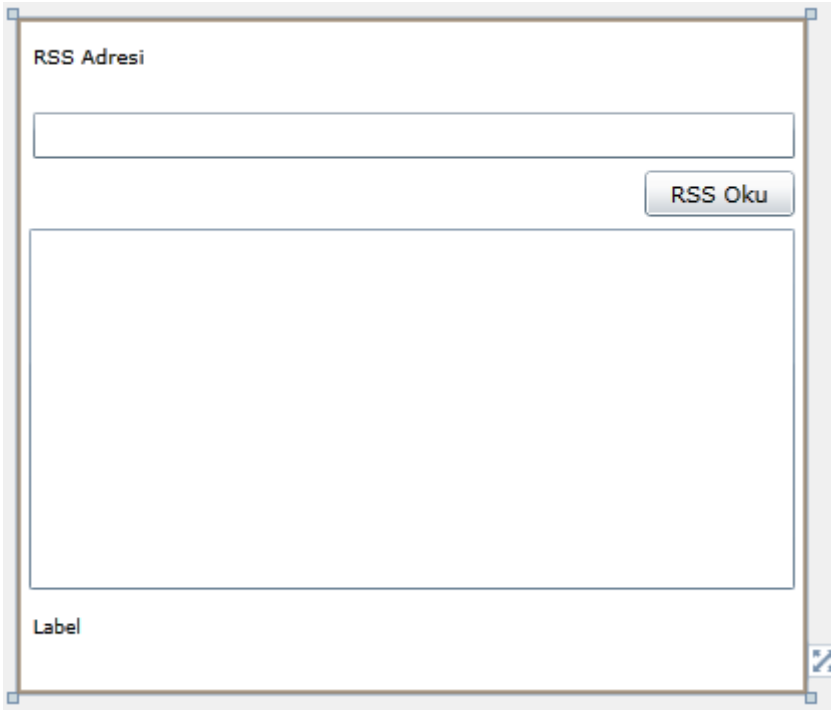
Silverlight Tarafından Feed Okumak (2010-07-22T18:05:00)

silverlight,wcf,syndication,wcf eco system,

Merhaba Arkadaşlar,

Yeni bir maceraya hazır mısınız? Hureyyy dediğinizi duyar gibiyim. Bildiğiniz üzere **Internet** kaynaklarının takibinin kolay bir şekilde yapılabilmesi adına **RSS** veya **Atom** formatındaki **Feed** içeriklerinden sıklıkla yararlanmaktayız. **Blog, Community, News Group** ve benzeri pek çok internet kaynağı, güncel içeriklerini yayınlamak amacıyla global olarak standart hale getirilmiş olan bu formatları kullanmaktalar. Pek tabi yayınlanan bu içeriklerin takip edilebilmesi içinde çeşitli istemci programlar söz konusu. **FeedReader** bu uygulamalara örnek olarak verilebilecek **Windows** tabanlı iddialı programlardan birisi. **Feed** içerikleri zaman zaman **internet** siteleri üzerinde kontrol şeklinde de barındırılmaktadır. Söz gelimi pek çok **blog**içerisinde bu durum söz konusudur ve hatta hazır **Widget**' lar yardımıyla entegrasyonları son derece kolaydır. Peki maceramız nerede başlıyor? özellikle ambulans resminin bu konu ile alakası nedir? 😬

Doğruyu söylemek gerekirse sıkıldığım bir ara ne yapayım diye düşünürken **Silverlight 4.0** tabanlı olarak geliştirilen bir uygulamadan**RSS** içeriklerini nasıl okuyabileceğimi düşünmeye başladım. Daha önceden **HTTP** bazlı **Get,Post,Put, Delete** metodlarınaa cevap veren **WCF** tabanlı servislerin tüketilmesi için**WebClient** tipinden nasıl yararlanıldığını incelemiştim(Silverlight Tarafında HTTP Bazli Servisleri Kullanmak isimli yazıyı incelemenizi öneririm) Yine aynı şekilde devam ederek herhangibir**RSS** içeriğini örnek **Silverlight** uygulamama taşıyabileceğimi düşünerek kolları sıvadım ve heyecanlı bir şekilde aşağıdaki ekran görüntüsü ve **XAML** içeriğine sahip kontrolü oluşturdum.



XAML içeriği;

```
<UserControl x:Class="RSSReaderim.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="337" d:DesignWidth="394"
xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk">

    <Grid x:Name="LayoutRoot" Background="White">
        <Button Content="RSS Oku" Height="23" HorizontalAlignment="Left"
Margin="313,76,0,0" Name="ReadRSSButton" VerticalAlignment="Top" Width="75"
Click="ReadRSSButton_Click" />
        <sdk:Label Height="29" HorizontalAlignment="Left" Margin="8,12,0,0"
Name="label1" VerticalAlignment="Top" Width="69" Content="RSS Adresi"
FontSize="10" />
        <ListBox Height="180" HorizontalAlignment="Left" Margin="6,105,0,0"
Name="RSSListBox" VerticalAlignment="Top"
Width="382" ItemsSource="{Binding}">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Title.Text}" Foreground="BlueViolet" />
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
```

```
      <TextBox Height="23" HorizontalAlignment="Left" Margin="8,47,0,0"
Name="RSSTextBox" VerticalAlignment="Top" Width="380" />
      <sdk:Label Height="40" HorizontalAlignment="Left" Margin="8,297,0,0"
Name="RSSInfoLabel" VerticalAlignment="Top" Width="380" FontSize="9" />
   </Grid>
</UserControl>
```

Aslında teori son derece basitti. Kullanıcı **TextBox** kontrolü üzerinden bir **RSS** adresi girecekti. Sonra düğmeye basarak içeriğin **ListBox** kontrolüne dolmasını seyredecekti. Son derece basit ve masumane bir talep öyle değil mi? Tabi bu işlemler için kod tarafını da, heyecanlı bir şekilde aşağıdaki gibi geliştirmeye çalıştım.

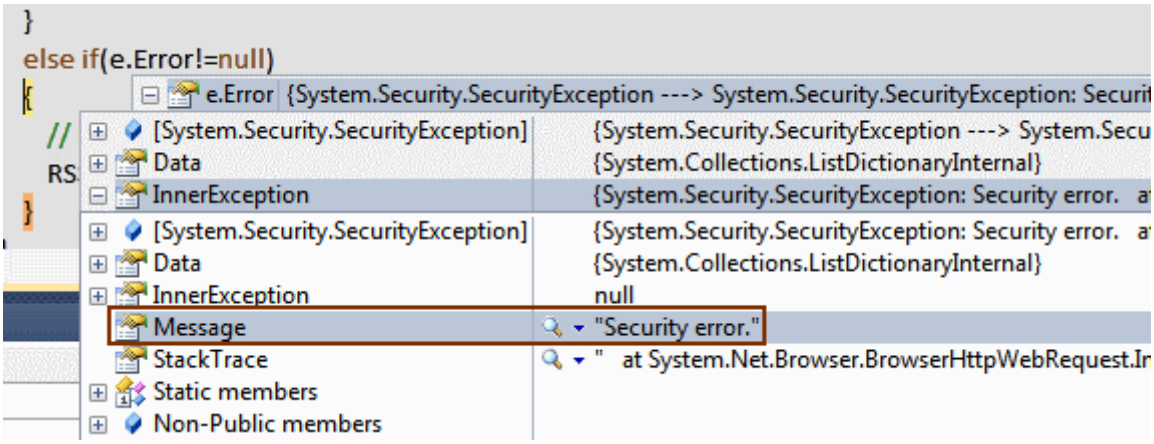
```
using System;
using System.Net;
using System.ServiceModel.Syndication;
using System.Windows;
using System.Windows.Controls;
using System.Xml;

namespace RSSReaderim
{
   public partial class MainPage : UserControl
   {
      WebClient client;

      public MainPage()
      {
         InitializeComponent();
         // WebClient nesnesi örneklenir
         client = new WebClient();
         // RSS Adresinden okuma işlemi tamamlanınca devreye girecek olan olay metodu
yüklenir
         client.OpenReadCompleted += new
OpenReadCompletedEventHandler(client_OpenReadCompleted);
      }

      void client_OpenReadCompleted(object sender, OpenReadCompletedEventArgs e)
      {
         if (e.Error == null) // Eğer okuma işlemi sırasında bir hata oluşmadıysa
         {
            // RSS bilgisi e.Result üzerinden Stream şeklinde elde edilir ve XmlReader
nesnesinin örneklenmesi için kullanılır. Bu gereklidir nitekim SyndicationFeed.Load
metodu XmlReader tipi ile çalışmaktadır.
            XmlReader xReader = XmlReader.Create(e.Result);
            // System.ServiceModel.Syndication.dll Assembly' ının projeye referans edilmesi
```

gerekmektedir.

```
            SyndicationFeed feed = SyndicationFeed.Load(xReader);
            // Items koleksiyonu ListBox bileşenine veri kaynağı olarak bağlanır
            RSSListBox.ItemsSource = feed.Items;
        }
        else if(e.Error!=null)
        {
            // Bir hata oluştuysa istisna mesajını Label kontrolünde göster
            RSSInfoLabel.Content = String.Format("Bir Sorun oluştu. {0}", e.Error);
        }
    }

    private void ReadRSSButton_Click(object sender, RoutedEventArgs e)
    {
        // Okuma işlemini başlat.
        // örnek RSS Adresi : http://www.buraksenyurt.com/syndication.axd?format=rss
        if (!String.IsNullOrEmpty(RSSTextBox.Text))
            client.OpenReadAsync(new Uri(RSSTextBox.Text));
        else
            RSSInfoLabel.Content = "Lütfen bir RSS Adresi giriniz";
    }
  }
}
```

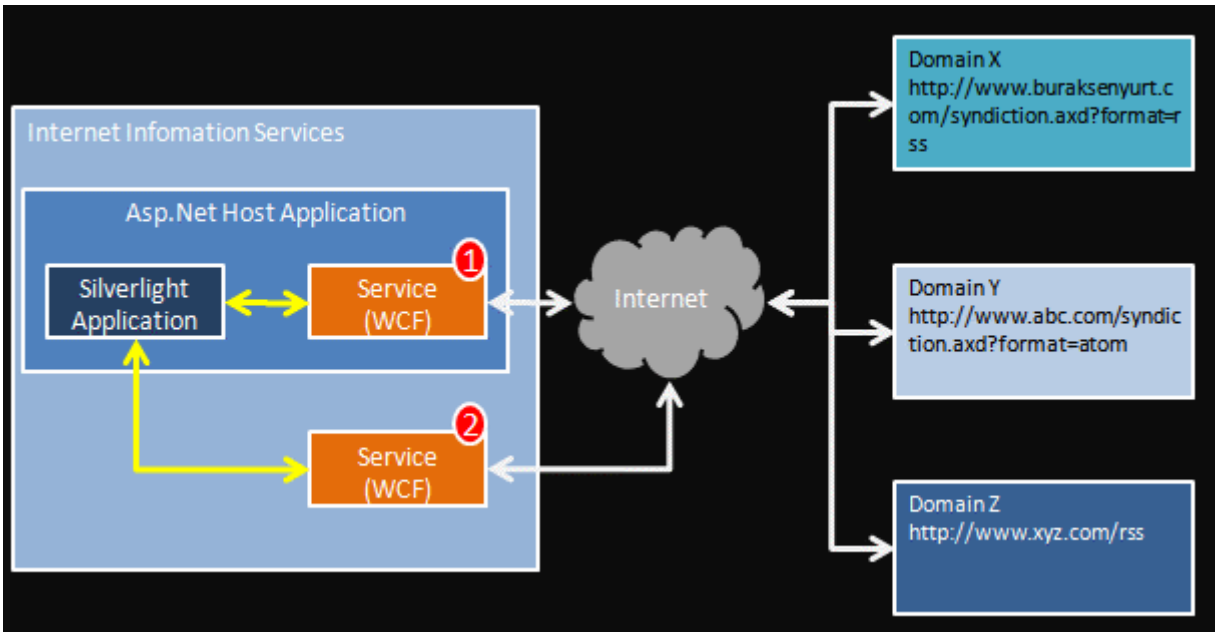
Kod parçasından da görüldüğü üzere **WebClient** tipini kullanarak **TextBox** kontrolüne girilen adres için bir talepte bulunulmaktadır. Söz konusu talepin sonucu elde edildiğinde devreye giren olay metodu içerisinde ise, öncelikli olarak bir hata kontrolü yapılmaktadır. Eğer herhangibir hata söz konusu değilse **SyndicationFeed** tipinden yararlanılarak elde edilen **Stream** referansınınFeed olarak ele alınabilmesi amacıyla gerekli işlemler yapılmaktadır. Son olarak söz konusu içerik nesnesi üzerinden ulaşılan **Items** koleksiyonu, **ListBox** kontrolüne bağlanır.

Şimdi **blog** girdimizin başında yer alan resmi açıklayalım. Bu kadar süratli araba kullanırsanız duvara toslamanız an meselesi olabilir. Aynen örneğimizde şu an tosladığımız gibi 😕 İşte duvara tosladığımız anda saniyenin milyonda birinde şişen hava yastığı içinden fırlayan **Exception** mesajımız.



Hayda breeeee!!! İşte hızlı gitmenin doğal sonucu.

Aslında gözden kaçırdığımız çok önemli bir durum söz konusu. O da **Silverlight** tarafında önem arz eden konuların başında gelen **Cross-Domain Policy** vakası. Sonuç itibariyle**RSS** çıktısı için talepte bulunduğumuz **Domain** adresi ile örneği geliştirmekte olduğumuz **Asp.Net Development Server**' ın **port** numarası eşliğine açtığı **Domain** adresleri birbirlerinden farklı. Bu sebepten sunucu tarafının bir **ClientAccessPolicy.xml** dosyasına sahip olması ve içerisinde söz konusu talepler için gerekli garanti haklarını belirtmiş olması şart. Ancak bu senaryoya göre **Silverlight** istemcileri için **Cross-Domain Policy** desteği vermeyen hiç bir sunucudan **RSS** içeriğini okumamız mümkün değil. Peki öyleyse ne yapacağız? çözüm olarak biraz dolambaçlı bir yol olsa da, aşağıdaki şekilde görülen planı izleyebiliriz.



Biliyoruz ki, **Silverlight** uygulamaları **Asp.Net** gibi Web uygulamaları içerisinde host edilebilmektedir. Planımıza göre **Cross-Domain Policy** sorunu ile karşılaşmayacak olan **WCF Service**' lerinin,**Silverlight** istemcilerinin talep edeceği **RSS** içeriklerini ele alması söz konusudur. Buna göre **Silverlight** istemcileri, **RSS** çıktılarına doğrudan talepte

bulunmak yerine söz konusu taleplerini önce arada **Proxy** görevini üstlenen bir **WCF** servisine iletecektir. Bu **WCF** servisi, ilgili adres bilgisini alarak **Feed** çıktısını talep edecek ve elde ettiği içeriği tekrardan **Silverlight** tarafına gönderecektir. Şekildeki plana göre **1** ve **2** numaralı iki adet **WCF** servisi söz konusudur. Bunlardan hangisinin seçileceği tamamen tercihe bağlıdır. İstersek **Silverlight** uygulaması ile aynı**Domain** içerisinde yer alan bir **WCF** servisini, istersek **IIS** üzerinde konuşlandırılan ayrı bir **WCF** servisini kullanabiliriz. Tabi **IIS** üzerinde host edilen bir **WCF** Servisi söz konusu ise, **Silverlight**istemcisi ile olan iletişiminin güvenlik sorununa takılmaması için **ClientAccessPolicy.xml** kullanılması gerekecektir. Tercih tamamen geliştiriciye bağlıdır. Ancak aynı Web sunucusu üzerinde yer alan birden fazla **Silverlight** istemcisi söz konusu ise ilgili servisin IIS altında konuşlandırılması daha çok tercih edilebilir.

Not : *Bu noktada hazır olarak* ***Siverlight*** *istemcilerine hizmette bulunabilen* ***Feed*** *servislerinden de yararlanabileceğimizi belirtmek isterim.* *Reading data and RSS with Silverlight and no cross-domain policy* *başlıklı yazıda* ***Tim Heuer*** *söz konusu servislerden bahsetmektedir.*

Ben örneğimizde hız kesemeden devam edebilmek adına, aynı uygulamaya **Silverlight** destekli bir **WCF** servisini ekleyerek ilerlemeyi tercih ettim. İşte **FeedReaderService** isimli **Silverlight** servisinin kod içeriği.

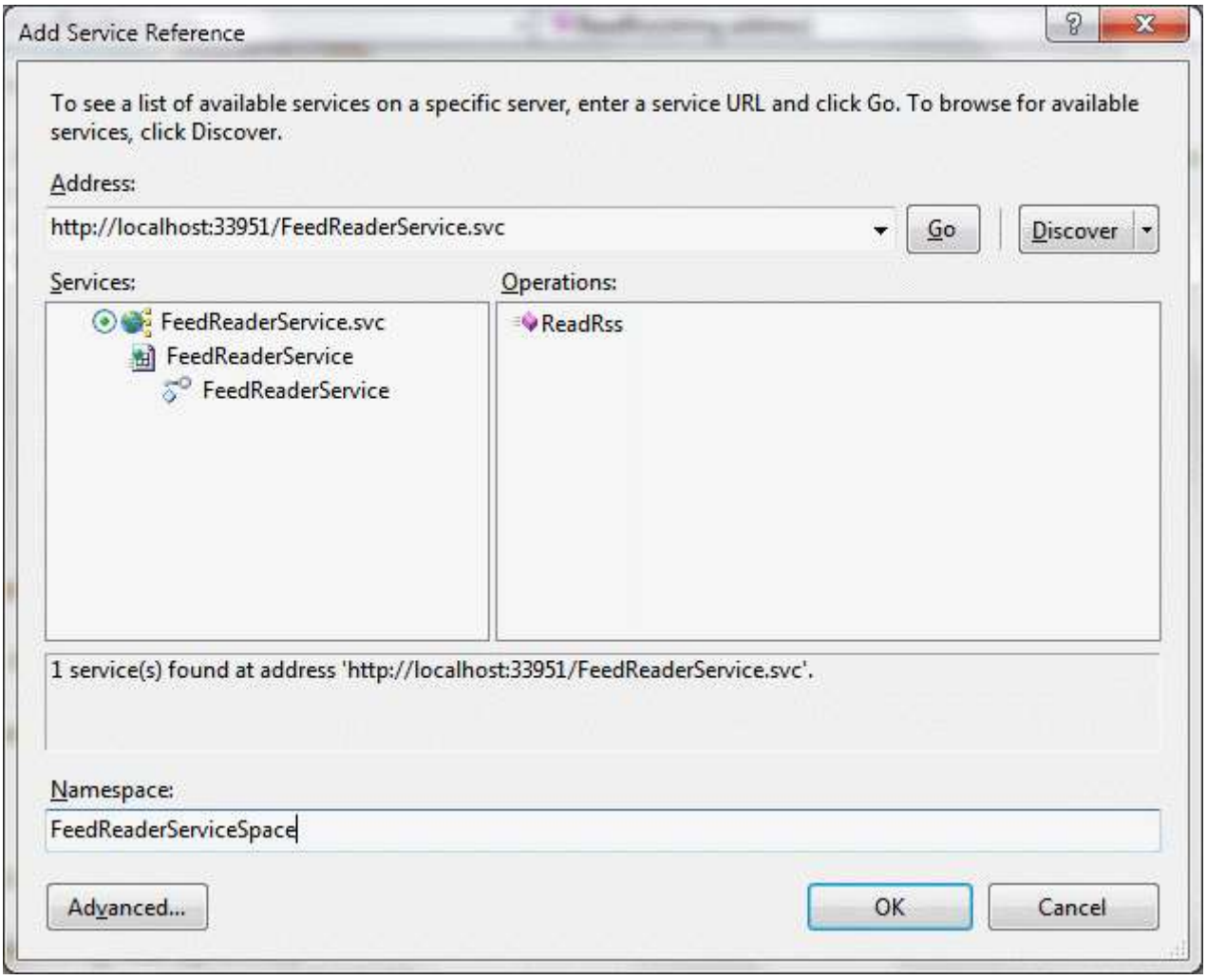
Not : Silverlight *destekli* ***WCF*** *Servicelerinin nasıl geliştirileceğini* *Screencast - Silverlight Enabled WCF Services* *isimli görsel dersten takip edebilirsiniz.*

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Syndication;
using System.Web.Services.Protocols;
using System.Xml;

namespace RSSReaderim.Web
{
   [ServiceContract(Namespace = "")]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   public class FeedReaderService
   {
      [OperationContract]
      public List<SyndItem> ReadRss(string address)
      {
         List<SyndItem> feedItems = null;
         try
```

```
        {
            // XmlReader.Create metodu parametre olarak address bilgisini almaktadır. Elde edilen Xml içeriği Load metodu yardımıyla çekilir ve SyndicationItem tipinden olan Items koleksiyonu çekilir.
            var syndicationItems =SyndicationFeed.Load(XmlReader.Create(address)).Items;
            // SyndicationItem örneklerinin her biri ele alınıp yeni bir SyndItem örneklenmesinde kullanılır.
            feedItems = (from syndicationItem in syndicationItems
                         select new SyndItem
                         {
                            Title = syndicationItem.Title.Text,
                            PublishDate = syndicationItem.PublishDate.DateTime,
                            Summary = syndicationItem.Summary.Text,
                            Link=syndicationItem.Links[0].Uri
                         }
                        ).ToList();
        }
        catch(Exception excp)
        {
            throw new SoapException("Bir hata oluştu", new XmlQualifiedName("RssReadError"), excp);
        }
        return feedItems;
    }
}

// SyndicationItem tipi serileştirme sorununa neden olduğundan araya bir Surrogate tip alınmıştır. Bu tip içerisinde Silverlight tarafı için gerekli temel Feed bilgileri yer almaktadır.
public class SyndItem
{
    public string Title { get; set; }
    public DateTime PublishDate { get; set; }
    public string Summary { get; set; }
    public Uri Link { get; set; }
    //TODO: Diğer bilgilerde getirilmelidir. örneği yazar bilgisi, son güncellenme tarihi veya kategoriler.
}
}
```

Servis kodunda dikkat edilmesi gereken en önemli noktalardan birisi, **ReadRss** metodunun geriye **SyndItem** tipinden generic bir **List** koleksiyonu döndürmesidir. Bu noktada akla şu soru gelebilir. Neden**List<SyndicationItem>** gibi bir koleksiyon döndürmüyoruz? 🙂 Aslında buradaki sorun **SyndicationItem** tipinin serileştirme işlemi

sırasında çalışma zamanı hatasına neden olmasıdır. Serileştirmedeki bu sıkıntı bizi alternatif bir yola itmiştir. Bu sebepten örnekte bir **Surrogate** tip kullanılmaktadır. Bu işlemin ardından artık **Silverlight** tarafı için gerekli geliştirmeler yapılabilir. İlk etapta aynı **Domain** içerisindeki*(bir başka deyişle aynı Solution içerisindeki)* **WCF Servisinin Silverlight** projesine eklenmesi gerekmektedir.



Sonrasında ise istemci tarafı için gerekli kodlar yazılabilir. Yeni örnekte **XAML** içeriği de aşağıdaki gibi düzenlenmiştir.

```
<UserControl x:Class="RSSReaderim.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="337" d:DesignWidth="394"
xmlns:sdk="http://schemas.microsoft.com/winfx/2006/xaml/presentation/sdk">

    <Grid x:Name="LayoutRoot" Background="White">
```

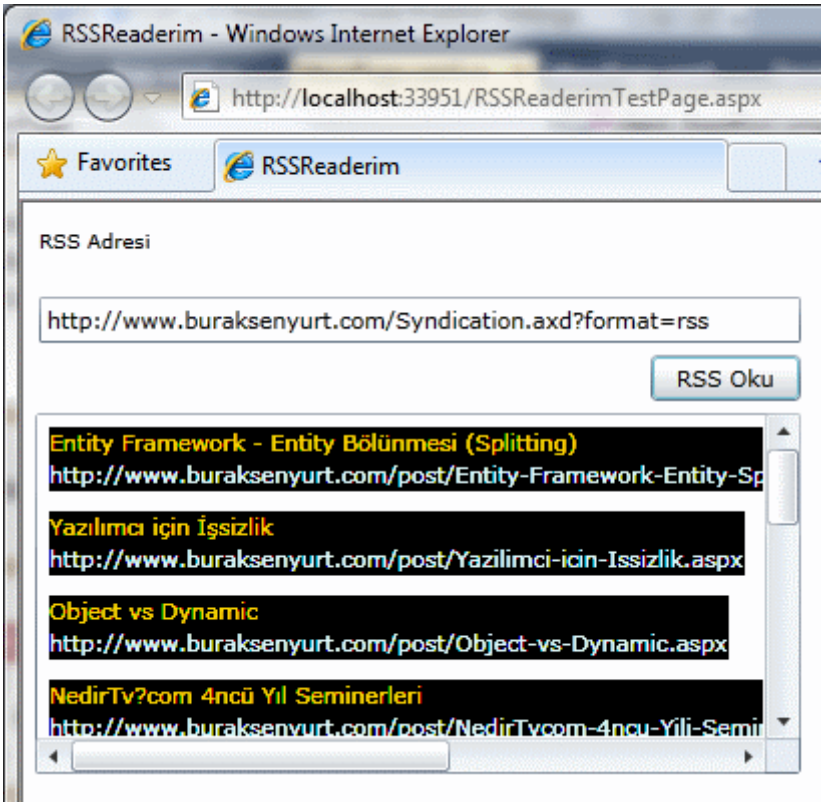
```
        <Button Content="RSS Oku" Height="23" HorizontalAlignment="Left"
Margin="313,76,0,0" Name="ReadRSSButton" VerticalAlignment="Top" Width="75"
Click="ReadRSSButton_Click" />
        <sdk:Label Height="29" HorizontalAlignment="Left" Margin="8,12,0,0"
Name="label1" VerticalAlignment="Top" Width="69" Content="RSS Adresi"
FontSize="10" />
        <ListBox Height="180" HorizontalAlignment="Left" Margin="6,105,0,0"
Name="RSSListBox" VerticalAlignment="Top"
Width="382" ItemsSource="{Binding}">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Vertical" Margin="2" Background="Black">
                        <TextBlock Text="{Binding Title}" Foreground="Gold" />
                        <TextBlock Text="{Binding Link}" Foreground="LightCyan" />
                    </StackPanel>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
        <TextBox Height="23" HorizontalAlignment="Left" Margin="8,47,0,0"
Name="RSSTextBox" VerticalAlignment="Top" Width="380" />
        <sdk:Label Height="40" HorizontalAlignment="Left" Margin="8,297,0,0"
Name="RSSInfoLabel" VerticalAlignment="Top" Width="380" FontSize="9" />
    </Grid>
</UserControl>
```

Bu kez **ListBox.ItemTemplate** içerisinde hem **Title** hemde **Link** bilgilerinin gösterilmesi sağlanmıştır. Yeni örnekte **SnydItem** isimli bir **Surrogate** tip söz konusu olduğundan ve bu tipin **Title**özelliği **String** tipten tanımlandığından, bir önceki **XAML** kodunda yer alan **{Binding Title.Text}** eşitlemesi kullanılmamalıdır. Gelelim kodlarımıza;

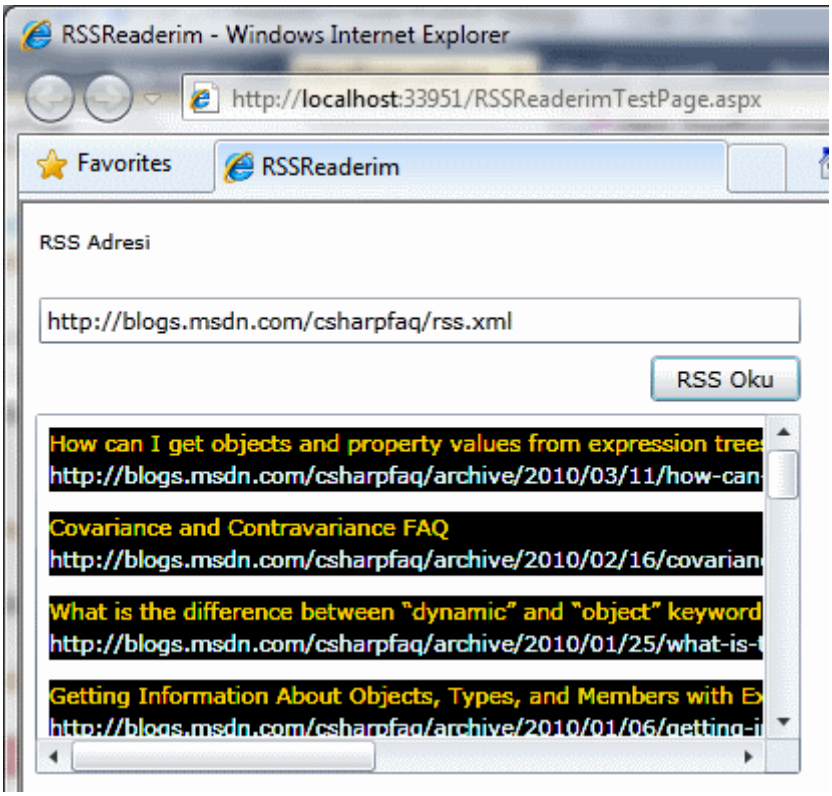
```
using System;
using System.Windows;
using System.Windows.Controls;
using RSSReaderim.FeedReaderServiceSpace;

namespace RSSReaderim
{
    public partial class MainPage : UserControl
    {
        FeedReaderServiceClient client = null;
        public MainPage()
        {
            InitializeComponent();
            client = new FeedReaderServiceClient();
            client.ReadRssCompleted += new
```

```
EventHandler<ReadRssCompletedEventArgs>(client_ReadRssCompleted);
        }

        private void ReadRSSButton_Click(object sender, RoutedEventArgs e)
        {
            client.ReadRssAsync(RSSTextBox.Text);
        }
        void client_ReadRssCompleted(object sender, ReadRssCompletedEventArgs e)
        {
            if (e.Error != null)
            {
                RSSInfoLabel.Content = e.Error;
            }
            else if (e.Cancelled)
            {
                RSSInfoLabel.Content = "İşlem iptal edildi";
            }
            else
            {
                RSSListBox.ItemsSource = e.Result;
            }
        }
    }
}
```

Kod içeriğinden de görüldüğü üzere **WCF** servisine ait **Proxy** tipinden yararlanılarak **Feed** içeriğinin asenkron olarak ortama çekilmesi işlemi gerçekleştirilmektedir. İşte örnek çalışma zamanı çıktılarından birisi.



ve diğer bir örnek;



Görüldüğü üzere **RSS** içerikleri başarılı bir şekilde getirilebilmektedir. Elbetteki örnekte eksik olan bir çok kısım vardır. Söz gelimi **RSS** ile ilişkili olarak daha çok verinin getirilmesi daha iyi olacaktır. Söz gelimi **Feed**' in sahibi olan siteye ait bilgiler. Diğer

yandan eksik kalan önemli noktalardan biriside **ListBox**' ta bir öğe seçildiğinde ilgili **Feed** adresine nasıl gidileceğidir. Sonuç itibariyle Silverlight uygulaması tarayıcı üzerinde çalışmaktadır ve ilgili **Feed** içeriğinin **Content** verisinin gösterilmesini herkes isteyecektir. İşte size güzel bir araştırma konusu ve ödev Benden buraya kadar. Bir süre dinlenmeye çalışacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

RSSReaderim_RTM.rar (1,48 mb)**[örnek Visual Studio 2010 Ultimate RTM sürümünde geliştirilmiş ve test edilmiştir]**

Microsoft Teknoloji Günleri Akşam Sınıfı - Gün 3 - WCF ile Servis Yaklaşımı Eğitimi Tamamlandı (2010-07-21T02:45:00)

microsoft teknoloji günleri,microsoft,bizspark,gelişim atölyesi,wcf,wcf 4.0,soa,

Merhaba Arkadaşlar,

25 Mayıs 2010 Tarihinde C# 4.0 ile birlikte gelen yenilikleri anlatarak başladığımız **Microsoft Teknoloji Günleri Akşam Sınıfının** üçüncü gününü de kazasız belasız tamamladık. öncelikli olarak tüm katılımcılarımızın ayağına sağlık.

Eğitimin video kayıtlarının alınması noktasında yardımcı olan **Mustafa Demirhan**'a, kameranın kapasitesi dolunca yardıma koşarak kendi kamerası ile çekim yapan **Microsoft Student Partner(MSP) adayı Burak özkan**'a, şu anda okumakta olduğunuz blog yazısındaki resimler gibi kaliteli, yüksek çözünürlüklü sayısız fotoğraf çekerek bizle paylaşan**Tuba çebi**' ye, Akşam Sınıfı fikrinin olgunlaşmasında en büyük yardımcım olan ve özellikle katılımcıları aşağıdaki resimde olduğu gibi hediyeler ile mutlu etmemizi sağlayan **Buket Şerefli**' ye**(Microsoft Türkiye İş Ortağı Yöneticisi)** canı gönülden teşekkür etmek istiyorum.

Buket Şerefli *tarafından temin edilen hediyeler arasında bulunan* ***Yeşil Vista Logolu Havlu****' sunu, kazanan değerli katılımcımıza hediye ederken.*

Microsoft Certified Partner *logolu duvar saatlerinden birisini, kazanan değerli katılımcımıza hediye ederken.*

Eğitimde çok güzel bir sürprizi de ben yaşadım. **Imagine Cup Türkiye** birincisi olan ve ülkemizi **Polonya**' da düzenlenen finallerde başarılı bir şekilde temsil eden ekipten iki arkadaşımız da aramızdaydı.

***Imagine Cup Türkiye birincisi** olan ekipten **İbrahim Kıvanç**(Imagine Cup 2010 Polonya Dünya Finalleri Ardından) ve **Yasemin çelik**, yarıştıkları **BabyRC** projesi hakkında kısaca bilgi verirken. (Ekibin bu akşamki eğitime katılamayan diğer üyeleri ise Burak Kanmaz ve Fatih Coşkun' dur)*

Eğitimimiz boyunca öncelikli olarak **Service Oriented Architecture(SOA)** kavramını kısaca tanımaya/anlamaya çalıştık. Sonrasında ise **Microsoft**' un uzun süredir vizyonda olan **SOA**implemantasyonu **WCF**' e**(Windows Communication Foundation)** giriş yaptık. **WCF** üzerinden uygulanabilen **SOA** desenelerini görüp, mimarisini ve çalışma zamanını kavradık. **WCF**' in**ABC'** sini de**(AddressBindingContract)** işin içerisine katmayı unutmadık elbette. Son olarak basit bir **Hello World** uygulaması geliştirip, **WCF 4.0** ile birlikte gelen bazı yenilikleri**(Discovery, Routing, Simplified Configurations)** örnek demolar üzerinden anlamaya çalıştık.

Yaklaşık olarak 3 saatten fazla bir zamanımızı **Microsoft İstanbul Ofisi Jupiter 1** salonunda geçirdik. Tabi akşam saati ve bir sonraki gününde mesai olması nedeniyle, özellikle uzakta oturan veya çok erken saatte işe giden arkadaşlarımızın bir kısmı mecburen erken çıkmak zorunda kaldılar. Ancak üzülmesinler nitekim bir şey kaçırmadılar. Tüm eğitimin video kaydını gerçekleştirdik. En kısa sürede render işlemini tamamlayıp Nedirtv?Com topluluğu üzerinden sizlerle paylaşıyor olacağım.

Bunlara ek olarak bir önceki eğitimimizde vaat ettiğimiz **Pro .Net 4.0 Parallel Programming** kitabını da, kazanan arkadaşımıza hediye ettik.

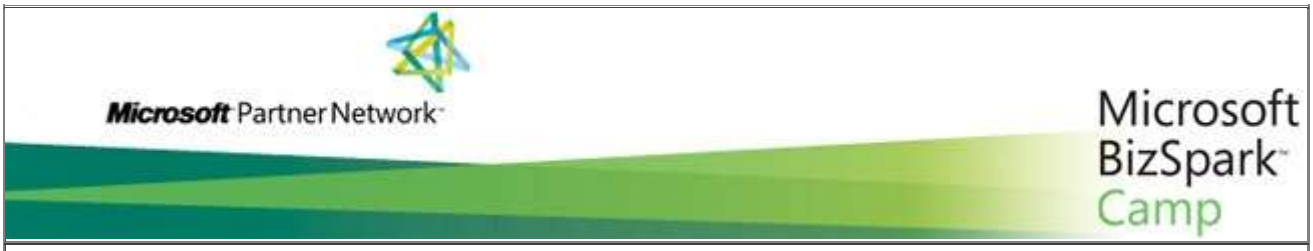
Bir sonraki eğitimimizin konusu **WCF Eco System**. **20 Ağustos 2010 Cuma** günü yapılacak bu eğitimi kaçırmamanızı öneririm. Eğitimin duyurusu ve kayıt linki 15 Ağustos' tan sonra duyuruluyor olacaktır.

Eğitimde Kullanılan örnekler : TeknolojiGunleri_WCF.rar (304,80 kb)

Eğitime ait Powerpoint Sunumu : WCF 4.0 - Introduction - Microsoft.pptx (483,87 kb)

[Microsoft Teknoloji Günleri Akşam Sınıfı - Gün 3 - WCF ile Servis Yaklaşımı (2010-07-14T14:12:00)]

microsoft teknoloji günleri,microsoft,bizspark,gelişim atölyesi,wcf,wcf 4.0,soa,



Tarih
20 Temmuz 2010 Salı

Saat
19.00 - 21.30

Yer
Microsoft İstanbul Ofisi

Eğitmen:
Burak Selim Şenyurt
Microsoft MVP

Hemen Kayıt Olun >

Microsoft Teknoloji Günleri -
Akşam Sınıfı

Her ay düzenli olarak gerçekleştireceğimiz ve bir seri olarak birbirini takip edecek sınıf etkinliklerimizle 9 ay boyunca siz yazılım geliştiren ve tasarım yapan iş ortaklarımızla birlikte olacağız.

Aşağıda detaylarını paylaştığımız ve sizler için hayli faydalı olacağına inandığımız Microsoft Teknoloji Günleri Akşam Sınıfı Etkinliğimize kaydınızı hemen yaptırabilirsiniz.

Ders İçerikleri:

1. **25.Mayıs.2010/Salı: C# 4.0 ile Gelen Yenilikler : (2 Saat)** Bu eğitimde C# programlama dilinin 4.0 versiyonu ile birlikte gelen yenilikleri tanıtmaya çalışılmakta ve özellikle, Dynamic diller ile Office API vb yapılar ile olan etkileşim üzerine örnekler geliştirilmektedir. Eğitimde temel olarak dynamic keyword, optional and named parameters, PIA, Co-Contra Variance Generics konularına değinilmektedir.

2. **22.Haziran.2010/Salı: .Net 4.0 ile Paralel Programlama : (2 Saat)** Uzun süredir ev bilgisayarlarımız dahil en az iki çekirdekli sistemler üzerinde çalışabiliyoruz. Son zamanlarda 8 çekirdeğe kadar çıkabilen yeni nesil işlemciler üzerinde paralel programlama yeteneklerini sonuna kadar kullanıp daha

performanslı, hızlı, verimli ve ölçeklenebilir uygulamalar geliştirmeye ne dersiniz? İşte tam size göre bir eğitim. Bu eğitim ile .Net 4.0 kütüphanelerini kullanarak kolay bir şekilde nasıl paralel programlama yapabileceğinizi göreceksiniz.

*3. **20.Temmuz.2010/Salı: WCF ile Servis Yaklaşımı : (2 Saat)** .Net 3.0 ile birlikte duyurulan ve tek bir servis geliştirme metodolojisi sunarak daha önceki dağıtık mimari geliştirme tiplerini (Xml Web Services, .Net Remoting, MSMQ vb...) bir çatı altında birleştiren Windows Communication Foundation konulu eğitimdir. Eğitimde özellikle .Net 4.0 ile birlikte gelen yeniliklere de değinilmekte olup asıl amaç WCF ile uygulama geliştirme şekillerinden bir kaçını göstermek ve tanıtmaktır.*

4. **20.Ağustos.2010/Cuma: WCF Eco System : (3 Saat)** WCF alt yapısı üzerine kurulu olan WCF Eco System içerisinde Data Services, Workflow Services, RIA Services, WebHttp Services ve Core Services tipleri yer almaktadır. Bu eğitimde Data Services, Workflow Services, RIA Services ve WebHttp Services konulu örnekler geliştirilmekte olup, söz konusu alt yapı ile değerlendirilebilecek hazır servis modelleri irdelenmektedir.

5. **20.Eylül.2010/Pazartesi: Windows Server AppFabric : (2 Saat)** Bu eğitimde WCF ve Workflow servis örneklerinin izlenmesi, sorunların teşhis edilmesi, örneklerin yaşam döngülerinin takibi, konfigurasyon ayarlarının belirlenmesi ve pek çok yönetimsel konuda geliştiriciler ile IT yöneticilerinin daha kolay anlaşabilmelerini de sağlayan Dublin kod adlı Windows Server AppFabric ürün ailesi incelenmektedir. özellikle IIS üzerine gelen eklentiler ile söz konusu yönetsel işlemlerin nasıl yapılabildiği derinlemesine incelenmektedir.

6. **20.Ekim.2010/çarşamba: Worfklow Foundation 4.0 : (3 Saat)** WF 4.0 beraberinde pek çok köklü yenilik ile gelmektedir. Geliştirilen Base Activity Library, paralel programlama desteği, veri akışı için gelen Argument, Variable gibi kavramlar ve daha pek

çoğunun ele alındığı eğitimde basit örnekler ile WF modelin tanıtılmaya çalışılmaktadır.

7. **23.Kasım.2010/Salı: Asp.Net 4.0 : (3 Saat)** Web Programlama' nın .Net 4.0 ile birlikte gelen yeni yüzünü görmeye hazır mısınız? Pek çok yeni özellik ile birlikte gelen Asp.Net 4.0' ın anlatıldığı bu eğitimde, temelden orta seviyeye kadar basit bir web uygulaması tasarlanmakta ve konunun daha iyi kavranabilmesi amaçlanmaktadır.

8. **20.Aralık.2010/Pazartesi: Visual Basic 2010 : (2 Saat)**Bu eğitimde Visual Basic 2010 programlama dili ile birlikte gelen pek çok yeni özellik üzerinde durulmakta ve geliştirilen örnekler ile bu kavramlar pekiştirilmeye çalışılmaktadır. Bu noktada AutoImplemented Properties, Collection Initializers, Implicit Line Continuation, Mutlipe Lambda Expressions, Dynamic keyword, Type Equivalance Support gibi konular üzerinde durulmaktadır.

9. **20.Ocak.2011/Perşembe: WPF 4.0 ile Windows Programlama : (3 Saat)** .Net 3.0 ile birlikte duyurulan Windows Presentation Foundation modeli ile zengin kullanıcı deneyimine sahip windows uygulamaları tasarlanabilmektedir. özellikle Windows 7 üzerinde en iyi kullanıcı deneyimini sunan WPF 4.0 ile birlikte gelen yenilikleri öğrenmeye ne dersiniz?

Workflow Foundation Öğreniyorum - Ders 11 - WCF Servislerini Kullanmak (2010-07-13T00:41:00)

workflow foundation 4.0,workflow foundation,visual studio 2010,

Merhaba Arkadaşlar,

NedirTv?com sponsorluğunda sürdürdüğümüz **"Workflow Foundation 4.0 öğreniyorum"** serimizin **on ikinci(11+1)** dersi ile karşınızdayız. Servisler, yazılım geliştirme dünyasının olmazsa olmaz parçalarından birisidir. özellikle son yıllarda servis geliştirme yaklaşımının parlayan yıldızı **WCF** Servisleri olarak düşünülebilir. önceki dağıtık mimari geliştirme modellerini tek bir çatı altında birleştirmeyi başaran bu Microsoft yaklaşımı, **Workflow Foundation** ile de yakın ilişki içerisindedir. Bilindiği üzere servisleri kullanan istemciler çeşitli tipte olabilirler. Workflow örnekleri de bu anlamda birer istemci olarak görülebilirler. Bu durumda, bir **Workflow** örneği içerisinden **WCF Servis Operasyonlarının** çağırılması da söz konusudur. İşte bu görsel dersimizde söz konusu çağrı işlemlerini en basit haliyle ele almaya çalışıyoruz. öyle ki; neredeyse **Toolbox->Messaging** sekmesinde yer alan ve servisler ile olan haberleşme işlemleri için kullanılan bileşenlerinden hiç birine dokunmadan. Nasıl mı? Haydi gelin öğrenelim.

Ders 11 - WCF Servislerini Kullanmak

Süre : 24:34

Dosya Boyutu : 34.7 Mb

örnek : Lesson11.rar (110,13 kb)

Mp4 Formatında İndirmek İçin Tıklayın

Silverlight Tarafında HTTP Bazli Servisleri Kullanmak (2010-07-12T09:55:00)

wcf webhttp services,silverlight 4.0,silverlight,wcf,http,get,post,put,delete,wcf eco system,

Merhaba Arkadaşlar,

Eğitmenlik yaptığım yıllarda **Microsoft**' un ders kitaplarında yer alan **LAB** çalışmalarını mümkün mertebe yapmaya ve yaptırmaya çalışırdım. Hatta çoğu zaman eğitimlere hazırlanırken sık sık bu lab çalışmalarını kendim yapar ve hatta ek ilaveler ile daha da eğlenceli hale getirmeye çalışırdım. Tabi bazen elimizde lab yapacağımız kitaplarımız olmazdı ki o ayrı bir hikaye. Lab çalışmaları öğrencinin adım adım yapması gerekenleri söylerek, konunun en yalın haliyle anlaşılmasını sağlamakta önemli rol oynamaktadır. Lab çalışmalarındakine benzer konu anlatımları benimde özümsediğim ve faydalı bulduğum öğrenme tekniklerinden birisidir. İşte bu yazımızda da bu kültüre uymaya çalışarak ilerlemeye çalışıyor olacağız. Hedefimiz **Silverlight** uygulamalarından, **HTTP** tabanlı taleplere göre operasyonel hizmetlerde bulunan servisleri nasıl kullanabileceğimizi, en yalın haliyle görmek. Haydi o zaman lab için gerekli materyalleri değerlendirerek yola koyulalım.

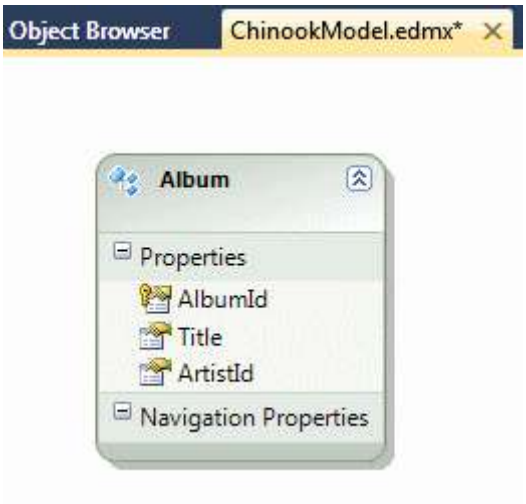
Adım 0 : Mevzumuz

Bilindiği üzere bazı servisler **HTTP** protokolü üzerinden **GET, POST, PUT** veya **DELETE** metod çağrıları ile kullanılabilmektedir. Bu anlamda **WCF Eco System** içerisinde yer alanWebHTTP servisleri, söz konusu tipteki hizmetleri sunmak üzere WCF alt yapısı üzerine oturmuş bir model sunmaktadır. çok doğal olarak **Silverlight** tabanlı istemciler de bu servislerin tüketicileri olabilirler. Bu tip servislerin kullanıldığı senaryolarda istemci tarafında herhangibir **Proxy** tipi söz konusu olmadığı için, **HTTP GET,POST,PUT** veya **DELETE** metodlarının manuel olarak hazırlanması ve gönderilmesi gerekmektedir. **Silverlight** tarafında bu işlemler için **WebClient** veya **HttpWebRequest** tiplerinden yararlanılabilmektedir. Biz bu yazımızda **WebClient**tipinden yararlanarak, **IIS(Internet Information Services)** üzerinde konuşlandırılmış basit bir **WebHttp Service** örneğinin nasıl kullanılabileceğini incelemeye çalışıyor olacağız.

Adım 1 : WCF Rest Application Uygulaması ve Entity Data Model' in Oluşturulması

İşe ilk olarak **WCF Rest Service Application** şablonunda bir proje oluşturarak başlayabiliriz. Bildiğiniz üzere bu proje şablonu**(Project Template)** hali hazırda yüklü değilse **Online Template**' ler arasından **install** etmeniz gerekmektedir. Söz konusu

örnekte **Chinook** veritabanında yer alan ve çok basit olarak ilerlemek istediğimizden sadece **Album** tablosunu içeren bir **Entity Data Model** kullanabiliriz. Aşağıdaki şekilde örneğimizde kullanmakta olduğumuz Entity Data Model yer almaktadır.



Adım 2 : WebHttp Service örneğinin Geliştirilmesi

Entities isimli **WCF WebHttp Service** sınıfımızın içeriğini ise aşağıdaki gibi düzenlediğimizi düşünebiliriz.

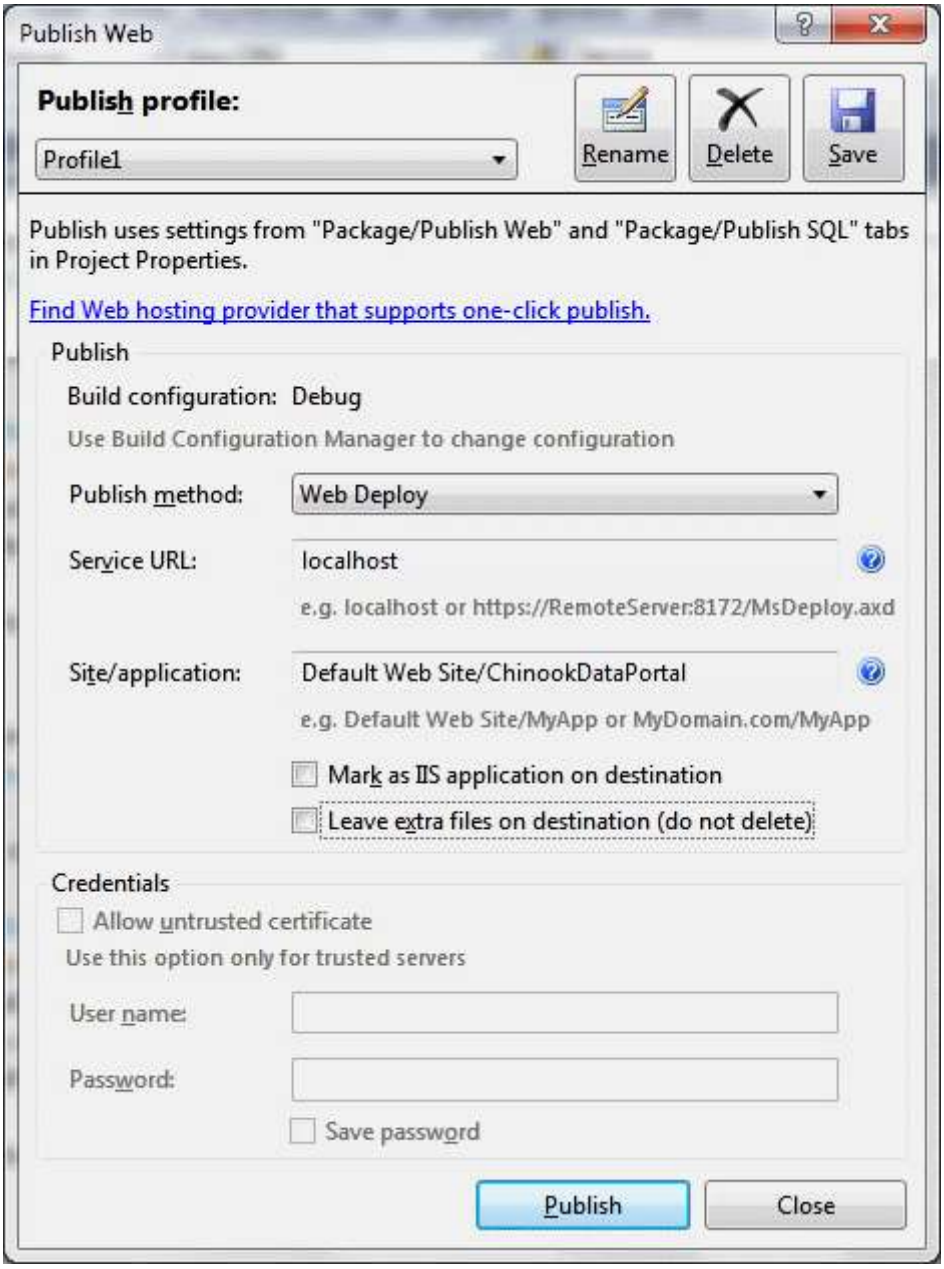
```
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;

namespace ChinookDataPortal
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class Entities
   {
      [WebGet(UriTemplate = "Albums/All")]
      public List<Album> GetAlbums()
      {
         List<Album> albums = null;

         ChinookEntities entities = new ChinookEntities();
         albums=(from albm in entities.Albums
```

```
                orderby albm.Title
                select albm).ToList();

            return albums;
        }

        [WebGet(UriTemplate="Albums/{firstLetter}")]
        public List<Album> GetAlbumsByFirstLetter(string firstLetter)
        {
            List<Album> albums = null;

            ChinookEntities entities = new ChinookEntities();
            albums = (from albm in entities.Albums
                      where albm.Title.ToLower().StartsWith(firstLetter.ToLower())
                      orderby albm.Title
                      select albm).ToList();

            return albums;
        }
    }
}
```

Servis tipimiz iki operasyon içermekte olup her ikiside **HTTP Get** çağrılarına cevap verecek şekilde düzenlenmişlerdir. **GetAlbums** metoduna yapılan çağrılarda servis **URL** adresine**Albums/All** takısı eklenmelidir. Diğer yandan ilk harflerine göre albümleri
listeleyen **GetAlbumsByFirstLetter** metodu, **URL** adresine **Albums/{firstLetter}** bilgisin in eklenmesini beklemektedir. Her iki metod**ChinookEntities** tipini kullanmakta ve basit **LINQ** sorguları ile sonuç üretmektedir. Servisimizi bu şekilde geliştirdikten sonra IIS altına Publish ederek devam edebiliriz.

Adım 3 : IIS Publish

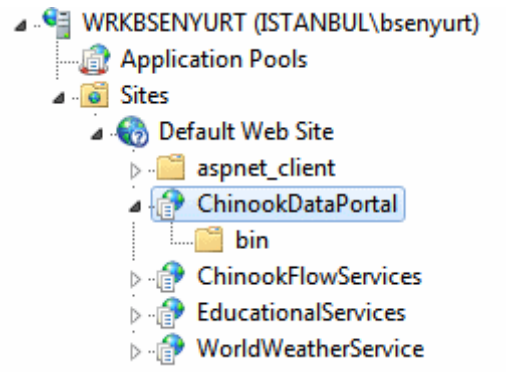
Publish işlemleri için aşağıdaki şekilde görülen Profile ayarlarını kullanabilirsiniz.



Bu ayarlara göre servisimizin **IIS** üzerinde yer alan **Default Web Site** isimli **Application Pool** altına dağıtılacağı belirtilmiş olunur.

***Not: IIS** üzerinden **Convert To Application** işlemini yapmanız gerekebilir.*

Sonuç olarak **IIS** içerisinde aşağıdaki gibi servisin üretilmiş olması gerekmektedir. Bu arada örneği geliştirdiğimiz makinede **Windows 7 Enterprise** işletim sisteminin ve **IIS 7.5.7600.16385**sürümünün olduğunu belirtelim.

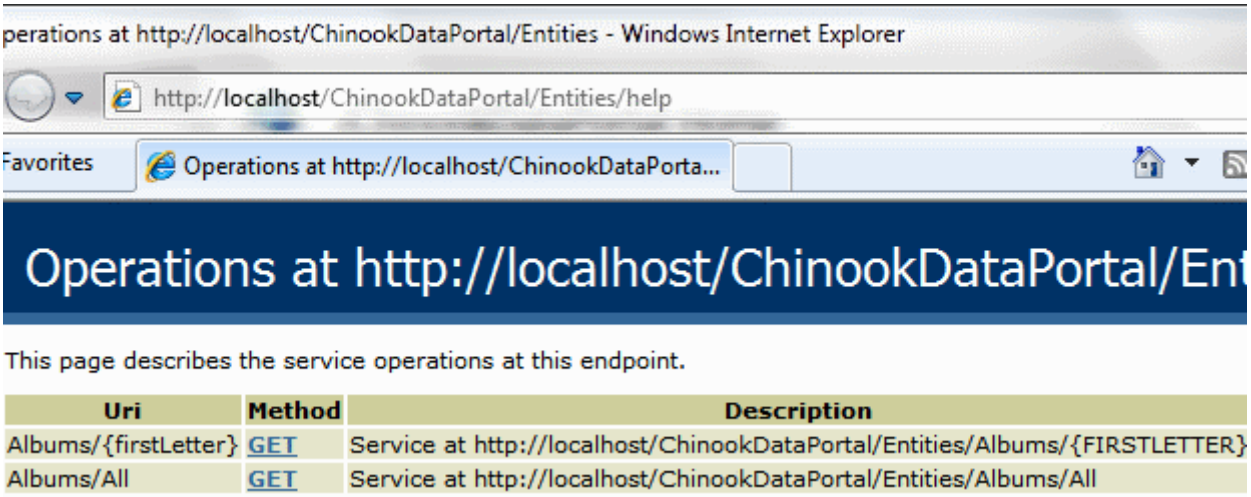


Bu noktadan sonra Silverlight uygulamasının geliştirilmesi aşamına geçilecektir. Ancak öncelikle gerekli testleri yapılmasında yarar vardır.

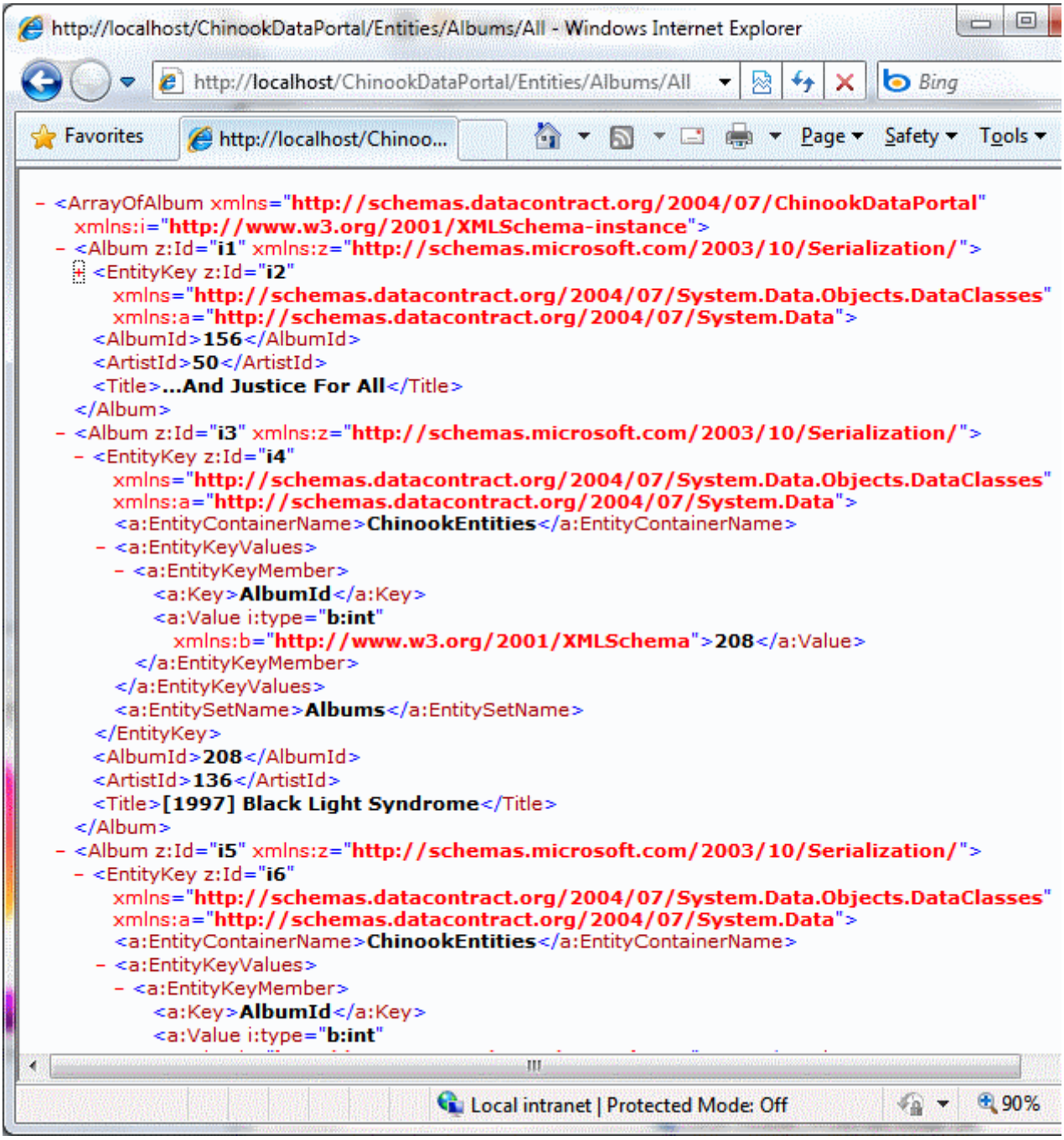
ChinookDataPortal.rar (46,54 kb) **[örnek Visual Studio 2010 Ultimate RC ortamında geliştirilmiş ve test edilmiştir]**

Adım 4 : WebHttp Service Test

Silverlight tarafındaki uygulamamızı geliştirmeden önce servisimizi **IIS** üzerinden test etmemizde ve çalıştığından emin olmamızda yarar olacağı kanısındayım. İlk olarak yardım sayfasına ulaşıp ulaşamadığımızı öğrenelim. Bilindiği üzere WebHttp Service örnekleri aksi belirtilmedikçe hazır bir yardım sayfası sunmaktadır. Bu amaçla tarayıcı uygulamadan **http://localhost/ChinookDataPortal/Entities/help** şeklinde bir talepte bulunduğumuzda, aşağıdaki ekran çıktısı ile karşılaşmış olmalıyız.

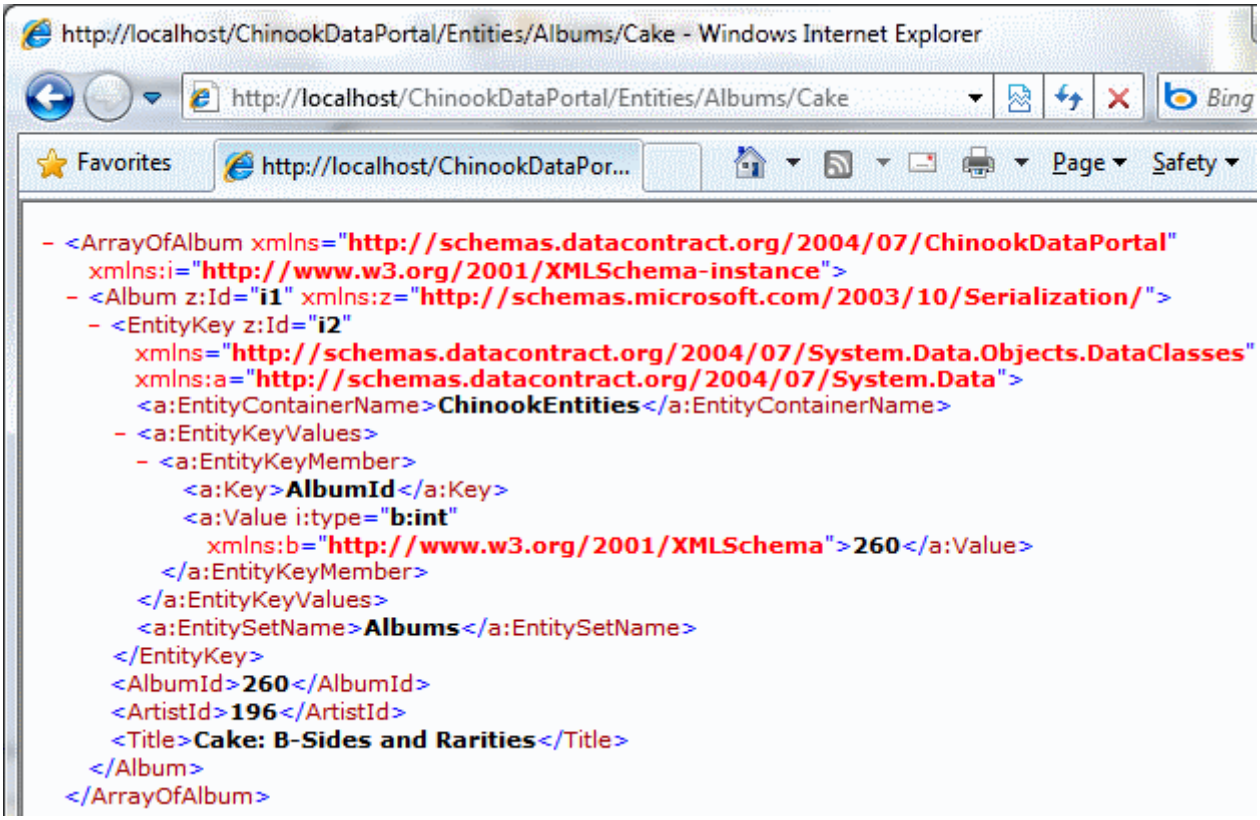


Yardım sayfasının çalışıyor olması dışında servis tarafında yer alan operasyonel metodların da test edilmesinde yarar vardır. örneğin tüm albümleri elde etmek için**http://localhost/ChinookDataPortal/Entities/Albums/All** şeklinde talepte bulunduğumuzda, aşağıdaki ekran görüntüsünde yer alan sonuçları elde etmiş olmamız gerekmektedir. Tabi veri içeriklerinde değişiklikler söz konusu olabilir. Ancak XML çıktısının şematik yapısının benzer olması gerekmektedir.




```
- <ArrayOfAlbum xmlns="http://schemas.datacontract.org/2004/07/ChinookDataPortal"
    xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  - <Album z:Id="i1" xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/">
    + <EntityKey z:Id="i2"
        xmlns="http://schemas.datacontract.org/2004/07/System.Data.Objects.DataClasses"
        xmlns:a="http://schemas.datacontract.org/2004/07/System.Data">
      <AlbumId>156</AlbumId>
      <ArtistId>50</ArtistId>
      <Title>...And Justice For All</Title>
    </Album>
  - <Album z:Id="i3" xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/">
    - <EntityKey z:Id="i4"
        xmlns="http://schemas.datacontract.org/2004/07/System.Data.Objects.DataClasses"
        xmlns:a="http://schemas.datacontract.org/2004/07/System.Data">
        <a:EntityContainerName>ChinookEntities</a:EntityContainerName>
      - <a:EntityKeyValues>
        - <a:EntityKeyMember>
            <a:Key>AlbumId</a:Key>
            <a:Value i:type="b:int"
              xmlns:b="http://www.w3.org/2001/XMLSchema">208</a:Value>
          </a:EntityKeyMember>
        </a:EntityKeyValues>
        <a:EntitySetName>Albums</a:EntitySetName>
      </EntityKey>
      <AlbumId>208</AlbumId>
      <ArtistId>136</ArtistId>
      <Title>[1997] Black Light Syndrome</Title>
    </Album>
  - <Album z:Id="i5" xmlns:z="http://schemas.microsoft.com/2003/10/Serialization/">
    - <EntityKey z:Id="i6"
        xmlns="http://schemas.datacontract.org/2004/07/System.Data.Objects.DataClasses"
        xmlns:a="http://schemas.datacontract.org/2004/07/System.Data">
        <a:EntityContainerName>ChinookEntities</a:EntityContainerName>
      - <a:EntityKeyValues>
        - <a:EntityKeyMember>
            <a:Key>AlbumId</a:Key>
            <a:Value i:type="b:int"
```

Son olarak örneğin **Cake** adı ile başlayan albümleri çekmek istediğimizi ve bu amaçla **URL** satırından **http://localhost/ChinookDataPortal/Entities/Albums/Cake** şekli nde bir talep gönderdiğimizi düşünelim. Bu durumda ekran çıktısının aşağıdakine benzer olması gerekmektedir.



Eğer bu sonuçları elde edebiliyorsak servisimizin çalıştığını ve **Sliverlight** tarafı için kullanılabilir olduğunu söyleyebiliriz. Lakin dikkat etmemiz gereken bir nokta daha vardır.

Adım 5 : Client Access Policy Ayarları

Silverlight uygulamamızın farklı bir **Domain** içerisinde **host** edilmesine karşılık, **IIS** üzerinde gerekli **Client Access Policy** ayarlarının bulunması gerekmektedir. Bu nedenle **IIS root**klasörü altında yer alması gereken **ClientAccessPolicy.xml** dosyasının içeriğini aşağıdaki gibi düzenleyebiliriz.

```
<?xml version="1.0" encoding ="utf-8"?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from>
        <domain uri="*" />
      </allow-from>
      <grant-to>
        <resource path="/WorldWeatherService" include-subpaths="true"/>
        <resource path="/ChinookDataPortal" include-subpaths="true"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

Burada görüleceği üzere ChinookDataPortal ve alt yollarına erişim izni verilmiştir. Artık **Silverlight** tarafını geliştirmeye başlayabiliriz.

Adım 6: Silverlight Application Projesinin Oluşturulması

Bu amaçla **Visual Studio 2010** ortamında **ConsumingHTTPBasedServices** isimi ve **Silverlight 4.0** tabanlı bir **Application** oluşturduğumuzu düşünelim. Söz konusu uygulamada RIA Service kullanılmayacağı için bu seçeneği pasif olarak bırakabiliriz. Bu işlem sonucu oluşturulan **MainPage** sayfasına ait **XAML** içeriğini ise aşağıdaki gibi geliştirebiliriz.

Adım 7 : MainPage.Xaml içeriği ve Kodun Yazılması

```
<UserControl x:Class="ConsumingHTTPBasedServices.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   mc:Ignorable="d"
   d:DesignHeight="300" d:DesignWidth="517">

   <Grid x:Name="LayoutRoot" Background="White">
      <ListBox Height="203" HorizontalAlignment="Left" Margin="8,70,0,0"
Name="AlbumListBox" VerticalAlignment="Top" Width="497" />
      <StackPanel Height="54" HorizontalAlignment="Left" Margin="9,10,0,0"
Name="ButtonsStackPanel" VerticalAlignment="Top" Width="496"
Orientation="Horizontal" />
   </Grid>
</UserControl>
```

MainPage içerisinde yer alan **ListBox** kontrolü içeriği **A' dan Z' ye harfler** ile doldurulacaktır. Herhangibir harfe basıldığında, **WebHttp Service**' imiz için bir **HTTP Get** talebi oluşturulacak ve sonuçların **ListBox** içerisinde gösterilmesi sağlanacaktır. Bu amaçla kod içeriğini aşağıdaki gibi geliştirmemiz yeterlidir.

```
using System;
using System.Linq;
using System.Net;
using System.Windows.Controls;
using System.Xml;
using System.Xml.Linq;

namespace ConsumingHTTPBasedServices
{
   public partial class MainPage
```

```
    : UserControl
  {
    // WebHttp Servisine basit HTTP metodları ile talepte bulunabilmemizi sağlayan
WebClient nesnesi tanımlanır
    WebClient client;

    public MainPage()
    {
      InitializeComponent();

      // WebClient nesnesi örneklenir
      client=new WebClient();
      // Belirtilen URL adresine yapılan talep sonucu gerçekleşecek okuma işlemi
tamamlandığında(bir başka deyişle veri istemci tarafında indirildiğinde) devreye girecek
olan olay metodu tanımlanır.
      client.OpenReadCompleted += new
OpenReadCompletedEventHandler(client_OpenReadCompleted);

      // A...Z Button üretimleri gerçekleştirilir
      for (int i = 65; i < 91; i++)
      {
        Button btn = new Button();
        btn.Width = 18;
        btn.Height = 18;
        btn.FontSize = 10;
        btn.Content = ((char)i).ToString();
        ButtonsStackPanel.Children.Add(btn);
        // Herhangibir Button tıklandığında
        btn.Click += (o, e) =>
        {
          // önce WebHttp Service' ne doğur yapılacak HTTP Get talebi için gerekli URI
oluşturulur
          Uri address = new
Uri(String.Format("http://localhost/ChinookDataPortal/Entities/Albums/{0}",
((Button)o).Content));
          // Belirtilen URI talebi asenkron olarak çalışan OpenReadAsycn metodu ile
gönderilir
          client.OpenReadAsync(address);
        };
      }
    }

    // URI ile belirtilen adres talebi gerçekleştirilip ilgili veri içeriği istemci tarafına
indirildikten sonra devreye giren olay metodudur
    void client_OpenReadCompleted(object sender, OpenReadCompletedEventArgs
```

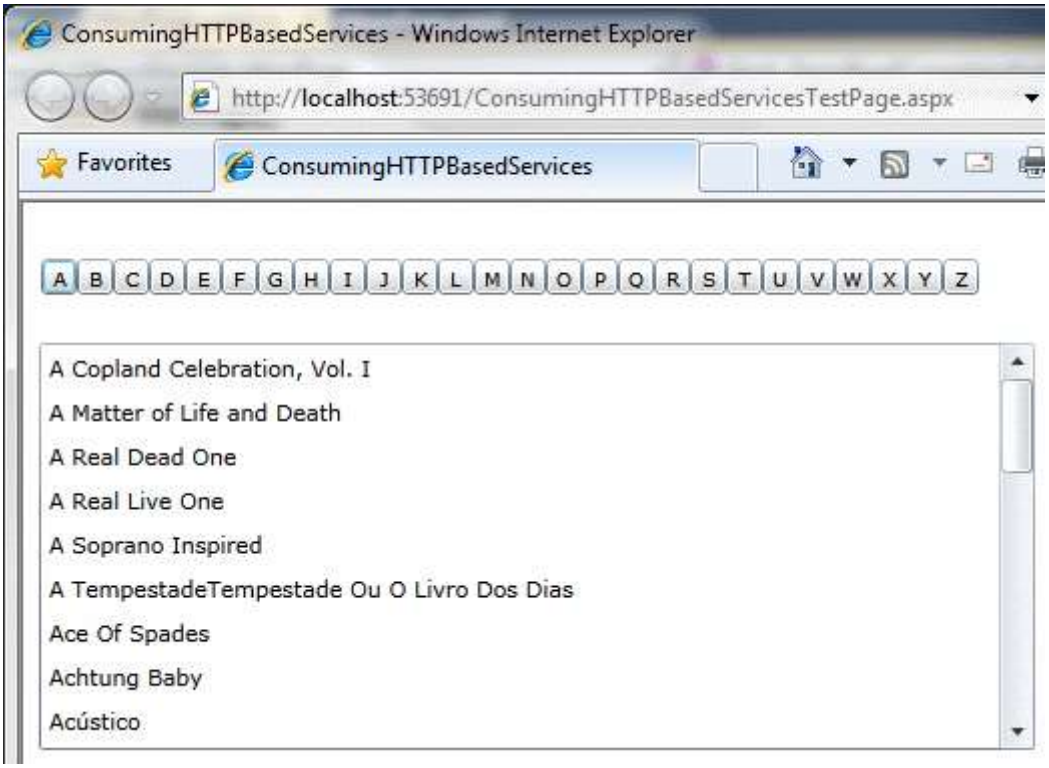
e)

```
        {
            // İçerik bir Stream olarak gelmektedir ve tasarlanan ChinookDataPortal WebHttp Servisi varsayılan olarak XML içerik göndermektedir.
            // Bu sebepten Stream XmlReader ile okunur
            XmlReader xReader = XmlReader.Create(e.Result);
            // XLINQ sorgusunun yapılabilmesi için XElement.Load metodu parametre olarak Stream' i kullanan XmlReader nesne örneğini alır
            XElement xElement = XElement.Load(xReader);
            // XLINQ sorgusu ile Title elementleri çekilir. XName.Get metodunun ikinci parametre XML Namespace' inin adıdır.
            var titles = from x in xElement.Elements().Elements(XName.Get("Title", "http://schemas.datacontract.org/2004/07/ChinookDataPortal"))
                         select x.Value;
            // çekilen veri içeriği ListBox kontrolünün ItemsSource özelliğine bağlanır
            AlbumListBox.ItemsSource = titles;
        }
    }
}
```

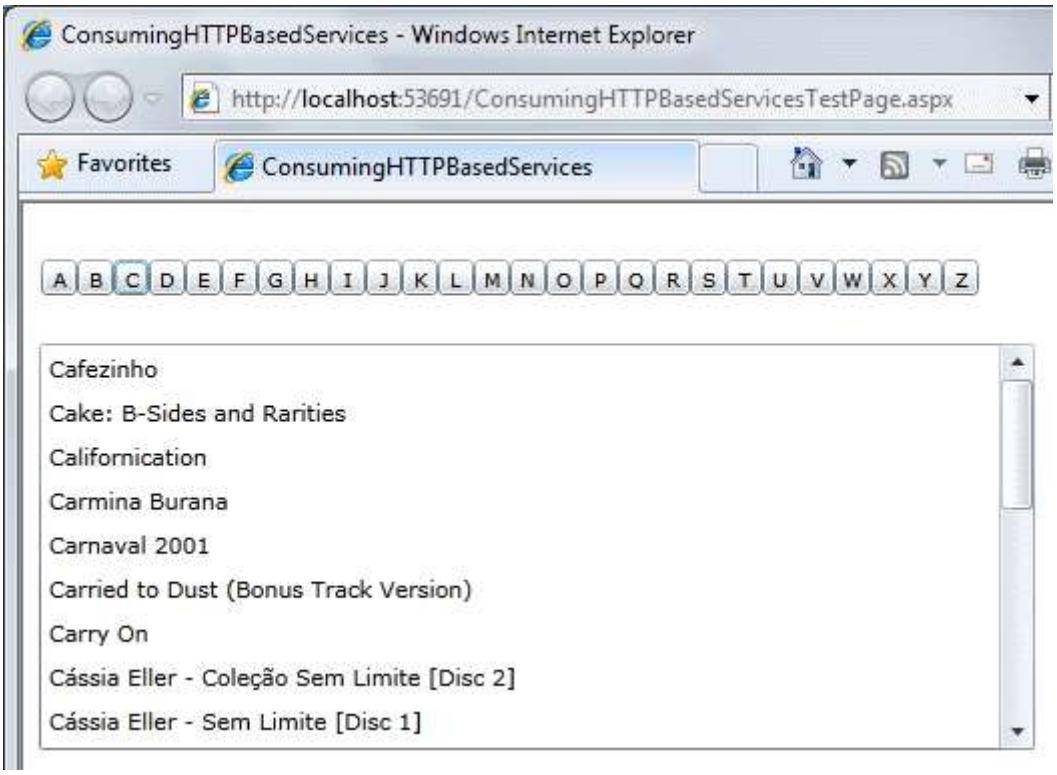
***Not: XElement** tiplerini kullanabilmek ve **XLINQ** sorgularını yazabilmek için, **Silverlight** uygulamasına(**ConsumingHTTPBasedServices.Web** uygulamasına değil) **System.Xml.Linq.dll Assembly**' ının referans edilmesi gerekmektedir.*

Adım 8 : Silverlight Uygulamasının Test Edilmesi

Dilerseniz uygulamanın çalışma zamanı sonuçlarına hemen bakalım. Böylece çalışma zamanı testlerini yapmış oluruz. örneğin **A** başlıklı **Button** kontrolüne bastığımızda, aşağıdaki ekran görüntüsündekine benzer sonuçları almış olmalıyız. Yani Title alanındakilerden A harfi ile başlayanların listesinin elde edilebiliyor olması gerekmektedir.



Görüldüğü üzere **ListBox** içeriği baş harfi A olan albüm adları ile doldurulmuştur. Hemen bu işlemin arkasından örneğin **C** başlıklı **Button** kontrolüne basarsak aşağıdaki sonuçlar ile karşılaştığımız görürüz.



Süper değil mi? 🙂

özet

Tabi bu örnekte dikkat edilmesi gereken noktalardan birisi de, istemci tarafında herhangibir **Proxy** tipinin olmayışıdır. Bunun yerine **HTTP Get** metodu ile talepte bulunulmuş ve elde edilen **Stream**üzerindeki **XML** içeriği değerlendirilmiştir. Diğer yandan çok doğal olarak **Servis** tarafında kullanılan **Entity Data Model** içerisindeki tiplerin istemci tarafındaki karşılıklarıbulunmamaktadır. Eğer bu tiplerin istemci tarafında ele alınması arzu edilirse açık bir şekilde oluşturulmaları gerekecektir. Tabi böyle bir senaryoda gelen **XML** veya **JSON** tipindeki içeriğinde ilgili tiplere dönüştürülmesi gibi bir işlem söz konusu olacaktır.

ödev

1. Servisin **XML** yerine **JSON(JavaScript Object Notation)** formatında bir çıktı vermesi halinde, **Silverlight** tarafında gerekli olan kod düzenlemelerini yapınız.
2. Servis üzerinden **HTTP Put** metod ile güncelleme işlemi yapabilmenizi sağlayacak bir geliştirmeyi aynı örnek üzerinden yapmaya çalışınız.

Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

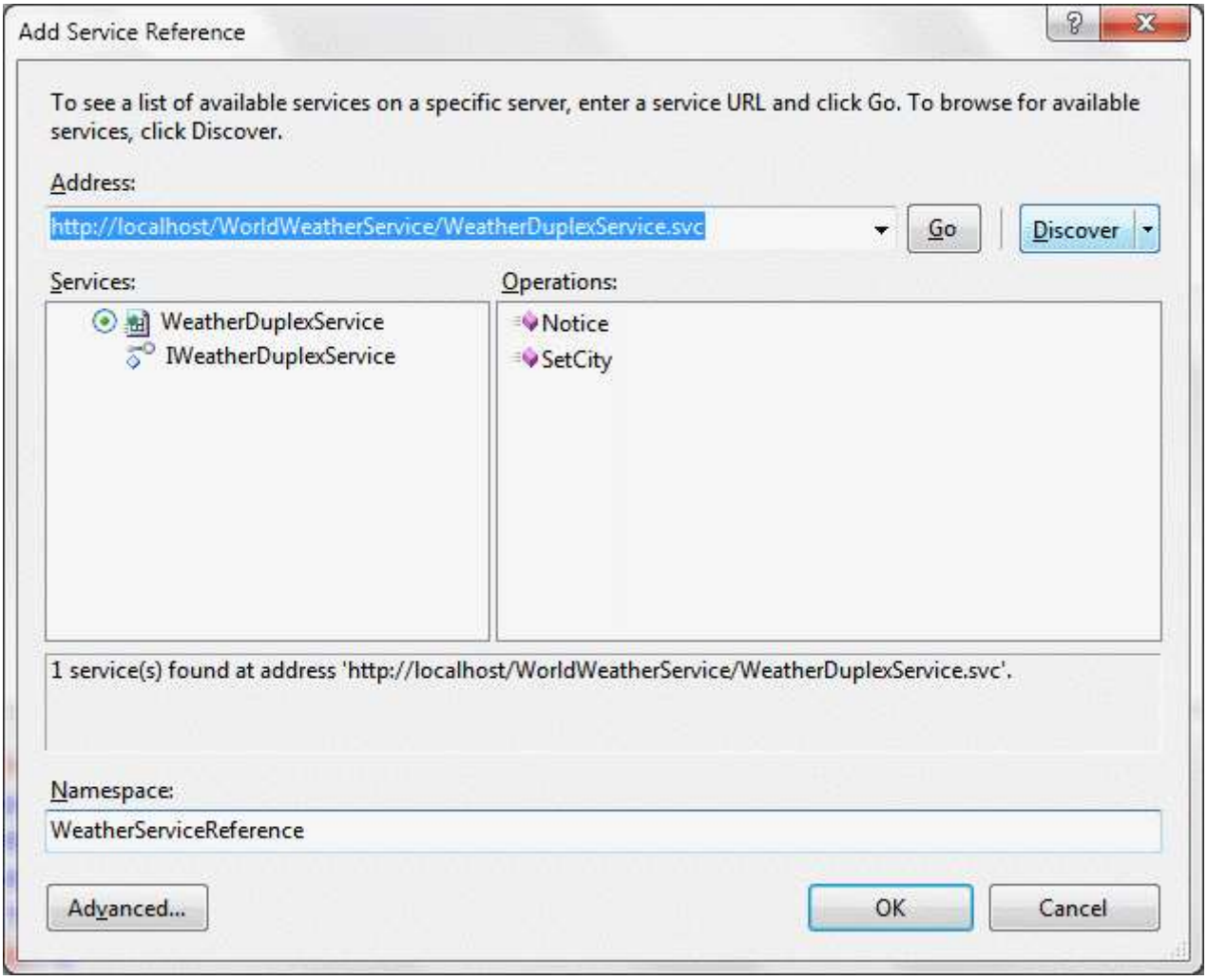
ConsumingHTTPBasedServices_RTM.rar (273,75 kb)**[örnek Visual Studio 2010 Ultimate RC Sürümü üzerinde geliştirişmiş ve RTM sürümü üzerinde test edilmiştir]**

Duplex Service için Silverlight İstemcisi Geliştirmek (2010-07-05T10:00:00)

wcf,silverlight,wcf service,duplex service,duplex communication,push,

Merhaba Arkadaşlar,

Hatırlayacağınız üzere bir önceki yazımızda **Silverlight** istemcilerinin kullanabileceği **Duplex WCF Service** uygulamalarının nasıl yazılabileceğini incelemeye çalışmıştık. çok doğal olarak bu işin bir de istemci tarafı bulunmaktadır. İşte bu yazımızda söz konusu istemciyi geliştirmeye çalışacak ve bir önceki yazının yorgunluğunu üzerimizden atarcasına, basit bir şekilde ilerliyor olacağız. İlk olarak **Visual Studio 2010 Ultimate RC** ortamında **Silverlight 4.0** tabanlı bir uygulama oluşturarak işe başlayabiliriz. Bu işlemin ardından**Proxy** tabanlı bir WCF servis kullanımı için **Add Service Reference** seçeneğine başvurmamız gerekecektir. Yine hatırlayacağınız üzere geliştirdiğimiz **WorldWeatherService** isimli servisi **IIS** üzerine **Publish** etmiştik. Bu sebepten ilgili servis referansına aşağıdaki şekilden de görüldüğü üzere **http://localhost/WorldWeatherService/WeatherDuplexService.svc**adresinden erişebiliriz.



İstemci tarafında çok basit olarak aşağıdaki **XAML** içeriğine sahip bir kontrol kullanıyor olacağız. Buna göre istemciler bir şehir adı girerek sunucudan anlık hava durumu bilgilerini alabilecekleri bir arayüze sahip olacaklar. Aslında alacaklar demek çok doğru bir tabir değil. Nitekim servisin kendisi, bağlı olan istemci üzerinde tetiklediği bir operasyona bu bilgileri parametre şeklinde gönderiyor olacak.

```
<UserControl x:Class="WeatherClientApp.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="230" d:DesignWidth="479">

    <Grid x:Name="LayoutRoot" Background="White">
        <Button Content="Start" Height="23" HorizontalAlignment="Left"
Margin="207,27,0,0" Name="StartButton" VerticalAlignment="Top" Width="75"
Click="StartButton_Click" />
        <TextBox Height="23" HorizontalAlignment="Left" Margin="25,27,0,0"
```

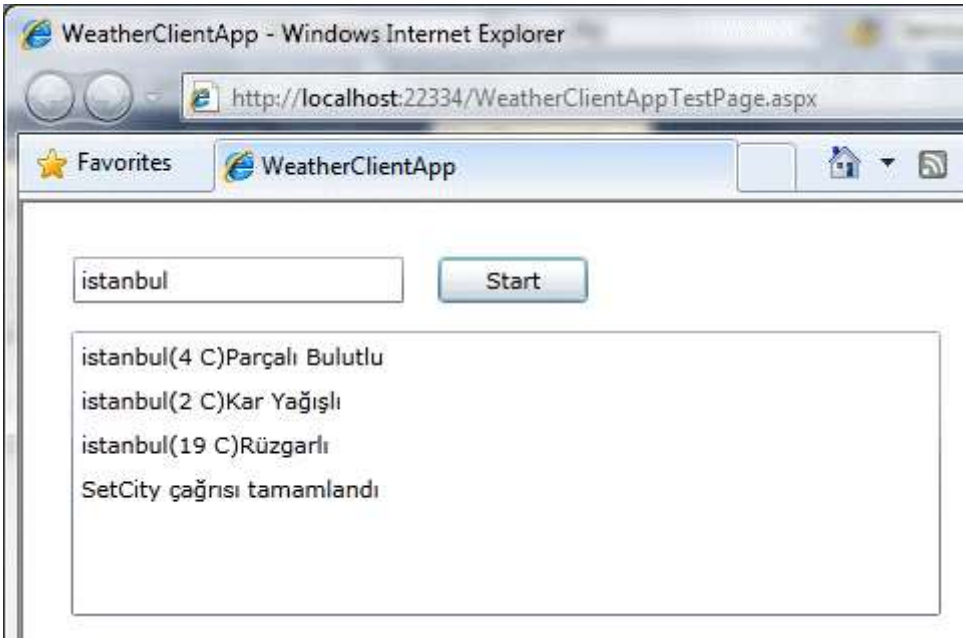
```
Name="CityTextBox" VerticalAlignment="Top" Width="165" />
      <ListBox Height="142" HorizontalAlignment="Left" Margin="24,64,0,0"
Name="WeatherStatusListBox" VerticalAlignment="Top" Width="434" />
   </Grid>
</UserControl>
```

İstemci tarafı kodlarına gelince;

```
using System;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.Windows;
using System.Windows.Controls;
using WeatherClientApp.WeatherServiceReference;

namespace WeatherClientApp
{
   public partial class MainPage : UserControl
   {
      WeatherDuplexServiceClient proxy = null;

      public MainPage()
      {
         InitializeComponent();

         EndpointAddress address = new
EndpointAddress("http://localhost/WorldWeatherService/WeatherDuplexService.svc
");
         PollingDuplexHttpBinding binding = new
PollingDuplexHttpBinding(PollingDuplexMode.MultipleMessagesPerPoll);
         proxy = new WeatherDuplexServiceClient(binding, address);

         proxy.SetCityCompleted += new
EventHandler<System.ComponentModel.AsyncCompletedEventArgs>(proxy_SetCit
yCompleted);
         proxy.NoticeReceived += new
EventHandler<NoticeReceivedEventArgs>(proxy_NoticeReceived);
      }

      void proxy_NoticeReceived(object sender, NoticeReceivedEventArgs e)
      {
         WeatherStatus wStatus = e.weather;
         WeatherStatusListBox.Items.Add(String.Format("{0}({1} C){2}",
wStatus.City, wStatus.Heat, wStatus.Summary));
      }
```

```
        void proxy_SetCityCompleted(object sender,
System.ComponentModel.AsyncCompletedEventArgs e)
        {
            WeatherStatusListBox.Items.Add("SetCity çağrısı tamamlandı");
        }

        private void StartButton_Click(object sender, RoutedEventArgs e)
        {
            WeatherStatusListBox.Items.Clear();
            proxy.SetCityAsync(CityTextBox.Text);
        }
    }

}
```

Dikkat edileceği üzere **WeatherDuplexServiceClient** tipinden olan **proxy** nesnesi örneklenirken iki önemli parametre bilgisi geçilmektedir. Bunlardan ilki **PollingDuplexHttpBinding** tipinden olan **bağlayıcı tiptir(Binding Type)**. Diğeri ise servise erişilecek olan **Endpoint** adresidir. İstemci tarafı asenkron olarak **SetCity** metoduna erişebilir. Bu nedenle **SetCityCompleted** olay metodu yüklenmiştir. Dikkat çekici noktalardan birisi de **NoticeReceived** isimli bir olayın söz konusu olmasıdır. Bilinen **Completed** son eki yerine **Received** son ekinin gelmesinin de bir anlamı vardır elbette. Bu, istemcinin servisten gelen **Notice** çağrısını takiben devreye girecek operasyon ile alakalıdır. Bir başka deyişle servis tarafı **Notice** metodunu çağırdıktan ve bu operasyon işleyişini tamamladıktan sonra istemci tarafında **proxy_NoticeReceived** olay metodu devreye girecektir. Ayrıca, bu olay metodunun **NoticeReceivedEventArgs** tipinden olan parametresi üzerinden yakalanan **weather** özelliği yardımıyla, servisin gönderdiği **WeatherStatus** nesnesine ulaşılabilir. Sonuç olarak uygulamayı test ettiğimizde örnek olarak aşağıdakine benzer bir sonuç elde ettiğimizi görebiliriz.



üç sonuç gelmesi tamamen servis tarafındaki zamanlama ayarları ile alakalı bir durumdur. Bu sürelerde oynayarak servisin istemci tarafına kaç kere bildirimde bulunacağını da ayarlayabilirsiniz. önemli olan nokta servisin istemci üzerinde bir operasyon tetiklemesidir. Bunu yazdığımız istemci ile test etmiş olduk.

Tüm bu çalışma sırasında dikkat edilmesi gereken bir husus da, önceki yazımızda da değinmiş olduğumuz **Client Access Policy** kullanımıdır. Eğer **IIS** root klasörü altında**ClientAccessPolicy.xml** dosyası ve gerekli içeriği olmassa çalışma zamanında aşağıdaki hata mesajı ile karşılaşılacaktır.

Oysaki geliştirdiğimiz örnek **Asp.Net Development Server** üzerinden yayınlanmaktadır*(http://localhost:22334/WeatherClientAppTestPage.aspx)* ve sorunsuz bir şekilde **IIS** üzerindeki**WorldWeatherService** uygulamasına erişebilmektedir. Dolayısıyla Silverlight uygulamalarında sıkça rastladığımız **Cross Domain** sorunu yaşanmamaktadır. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WeatherClientApp.rar (540,12 kb) **[örnek Visual Studio 2010 RC sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Workflow Foundation Öğreniyorum - Ders 10 - InvokeMethod (2010-07-02T11:00:00)

workflow foundation 4.0,workflow foundation,visual studio 2010,

Merhaba Arkadaşlar,

NedirTv?com sponsorluğunda sürdürdüğümüz "Workflow Foundation 4.0 öğreniyorum" serimizin **onbirinci(10+1)** dersi ile karşınızdayız. Bu dersimizde harici metodların, **InvokeMethod Activity** bileşeni yardımıyla nasıl çağırılabileceğini incelemeye çalışıyoruz. **InvokeMethod**bileşeni ile, **static sınıflar** içerisinde yer alan **static metodların** veya **örneklenebilir sınıflar** içerisinde yer alan **metodların** çağırılma şekillerini irdelediğimiz projemizde, yine **Ado.Net Entity Framework** tabanlı bir veri kaynağını kullanarak ilerliyoruz. E haydi o zaman. Keyifli seyirler dilerim.

Ders 10 - InvokeMethod

Süre : 12:48

Dosya Boyutu : 18.7 Mb

örnek : Lesson10.rar (98,86 kb)

Mp4 Formatında İndirmek İçin Tıklayın

Workflow Foundation Öğreniyorum - Ders 9 - Custom Activity Geliştirmek (2010-06-23T21:29:00)

workflow foundation 4.0,workflow foundation,visual studio 2010,custom activity,

Merhaba Arkadaşlar,

NedirTv?com sponsorluğunda sürdürdüğümüz **"Workflow Foundation 4.0 öğreniyorum"** serimizin **onuncu(9+1)** dersi ile karşınızdayız. Bu dersimizde var olan **built-in activity** bileşenlerinin bize yetmediğini düşünerek hareket ediyor ve **CodeActivity** sınıfı türevli **Custom Activity**bileşenlerinin nasıl yazılabileceğini incelemeye çalışıyoruz.

Hatırlayacağınız üzere önceki **Workflow Foundation** sürümünde, **Workflow** örnekleri içerisinde doğrudan kod tanımlaması yapılabilmesi mümkündür. Bu noktada **CodeActivity** bileşeninden yararlanılmaktadır. Oysaki yeni sürümde **CodeActivity** tipi **abstract** bir **sınıf** haline getirilmiş,**generic** bir versiyonu daha eklenmiş ve **Custom Activity** bileşeni geliştirilmesinin yollarından birisi olmak üzere sunulmuştur. Bu gelişmenin en büyük nedenlerinden birisi de, **WF 4.0** içerisindeki **Workflow Activity**' lerin herhangibir şekilde doğrudan kod tanımlaması içermelerinin mümkün olmamasıdır*(En fazla **Visual Basic Expression**)* İşte onuncu dersimizde çok basit bir örnek yardımıyla bu konuyu irdelemeye çalışıyor ve seviyemizi Level 125' e kadar çıkartıyoruz.

Ders 9 - Custom Activity Geliştirmek

Süre : 17:24

Dosya Boyutu : 21.5 Mb

örnek : Lesson9.rar (102,49 kb)

Mp4 Formatında İndirmek İçin Tıklayın

Microsoft Teknoloji Günleri Akşam Sınıfı - Gün 2 - Paralel Programlama Tamamlandı (2010-06-23T06:59:00)

microsoft teknoloji günleri,microsoft,bizspark,gelişim atölyesi,parallel programming,tpl,plinq,task parallel library,

Merhaba Arkadaşlar,

Microsoft Teknoloji Günleri Akşam Sınıfının ikinci eğitimi olan **Paralel Programlama** dersimizi, **22 Haziran 2010** günü **Microsoft İstanbul Ofisi Jupiter 1** salonunda gerçekleştirdik. öncelikle katılan tüm arkadaşlarımıza, ilgi ve alakaları için teşekkür ediyorum.

Bu sefer, seminerimizin video kaydını da gerçekleştirdik. ~~Gerekli video birleştirme işlemlerini en kısa sürede gerçekleştirip NedirTv?com üzerinden yayınlıyor olacağız.~~ Seminer kaydını http://www.nedirtv.com/video/Microsoft-Teknoloji-Gunleri-Aksam-Sinifi-Net-40-ile-Paralel-Programlama-Egitimi.aspx adresinden izleyebilirsiniz.

özellikle kamera ve organizasyon konusunda bize destek sunan **Buket Şerefli**' ye, çekimleri üstlenen **Mustafa Demirhan**' a ve seminer resimlerini çeken **Tuba çebi** arkadaşlarımıza katkılarından dolayı minnettarım.

Faydalı olduğunu düşündüğüm bu eğitime ait örnek kod ve sunum dosyalarını aşağıdaki linklerden tedarik edebilirsiniz.

Bu arada bir sonraki eğitimde verilmek üzere bir de ödülümüz var. Bu ödül için seminere katılan arkadaşlarımızın, sunum sırasında aktarılan soruyu geliştirip göndermesi gerekiyor. ödülümüz ise,

Bir sonraki eğitimimizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Sunum (6.37 Mb)

Solution (16.3 Mb) **[örnekler Visual Studio 2010 Ultimate üzerinde geliştirilmiş ve test edilmiştir]**

TPL - Göz Göre Göre Başımızı Belaya Sokmak (2010-06-21T18:08:00)

task parallel library,parallel programming,c# 4.0,.net framework 4.0,visual studio 2010 ultimate,deadlock,local variable evaluation,excessive spinning,

Merhaba Arkadaşlar,

Bazen göz göre göre başımıza bi ton dert açarız. Kimi zaman başlayacağımız iş bize çok eğlenceli gelebilir *(Yandaki resimde yüzü görünmeyen şahsın da bu heyacanla Hamburgere bindiğinden eminiz)* Ama işin sonuçlarını biliyorsak eğer, bunu yapmamızın nedeni büyük olasılıkla adrenalindir.

Tabi ki bir yazılımcı için adrenalin genellikle üst yöneticisi tarafından salgılanan bir hormondur. Nitekim yazılımcıların, ilerideki felaketleri kestirerek hareket etmesi ve geliştirmeleri buna göre yapması her zaman kolay olmayabilir. Bir başka deyişle bazı vakalara hazırlıklı olmak için önceden bunları çalışmak gerekmektedir.

İşte bu yazımızda biz de **Task Parallel Library** için söz konusu olan ve geliştiricilerin başını derde sokacak 2 vaka üzerinde duruyor olacağız. Haydi o zaman parmakları sıvayalım ve işe koyulalım.

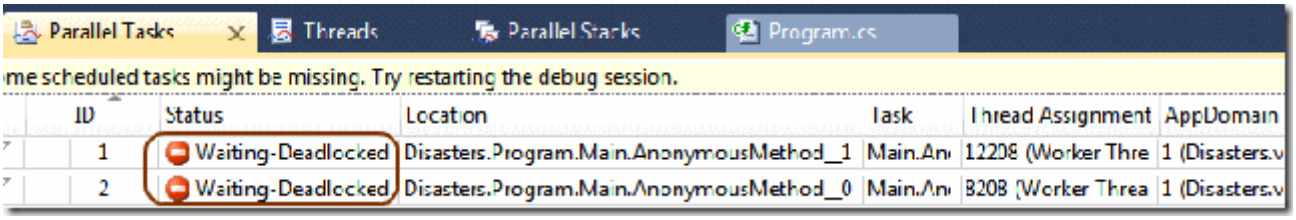
Deadlock Durumu

Bu kelime her zaman korkutucu olmuştur. Yazılım Geliştirme serüvenime ilk başladığım yıllarda çoğunlukla veritabanı tarafındaki kilitlenmelerden söz edildiğini çok net hatırlıyorum. Ancak birden fazla iş parçasının da **deadlock'** a düşmesi, bir başka deyişle birbirlerini beklemeleri nedeniyle, içinde çalıştıkları **Thread**' i*(çoğunlukla ana uygulama iş parçası-**Main Thread**)* kitlemeleri söz konusudur. Durumu daha net anlayabilmek için aşağıdaki kod parçasını göz önüne alalım.

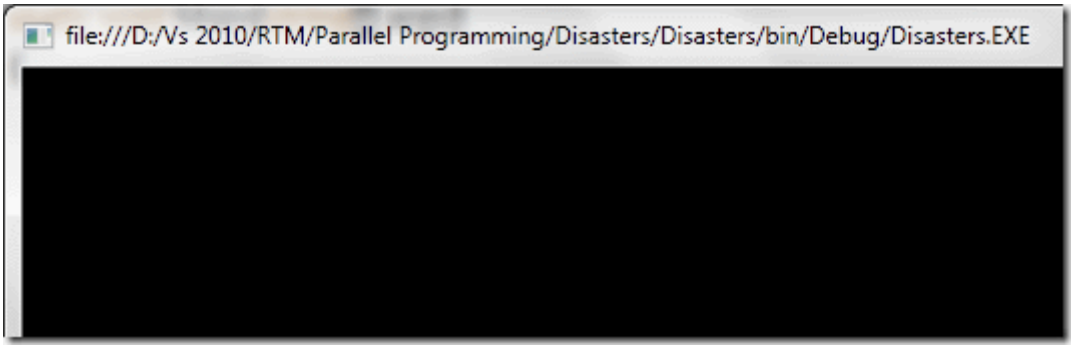
```
using System;
using System.Threading.Tasks;

namespace Disasters
{
   class Program
   {
      static void Main(string[] args)
```

```
        {
            Task<double> task1 = null;
            Task<int[]> task2 = null;

            task2 = Task.Factory.StartNew<int[]>(() =>
            {
              double task1Result=task1.Result;
               Random rnd = new Random();
               int[] numbers = new int[5];
               for (int i = 0; i < 5; i++)
               {
                  numbers[i] = rnd.Next(1, 250);
               }
               return numbers;
            }
            );

            task1 = Task.Factory.StartNew<double>(()=>
            {
               double totalValue = 0;
               foreach (int number in task2.Result)
               {
                  totalValue += number;
               }
               return totalValue;
            }
            );

            Task.WaitAll(task1, task2);

            Console.WriteLine("İşlemlerin sonu.Programdan çıkmak için bir tuşa basınız");
            Console.ReadLine();
        }
    }
}
```

Aslında kod çok fazla değerlendirilebilir veya anlamlı değildir.
Ancak **Deadlock** oluşumunu görmemiz açısından yeterlidir.
örnekte **task1** ve **task2** isimli **Task** nesne örneklerinin, birbirlerinin dönüş değerlerini kullanmaya çalıştığı ifade edilmektedir. İşte **Task** örneklerinin çalışma zamanında birbirlerini beklemeleri, kendi durumlarının **Deadlock** olarak set edilmesine neden olacaktır. Bu durum **Debug** modda aşağıdaki ekran görüntüsünde olduğu gibi görülebilir.



Görüldüğü üzere her iki **Task** birbirini bekler şekilde kalmıştır. çok doğal olarak çalışma zamanı çıktısı kapkara bir ekran olacaktır.



Gelelim diğer bir senaryoya.

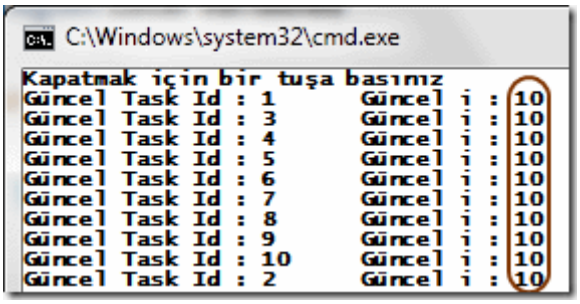
Döngü Değişkenlerine Dikkat

Bu aslında oldukça eğlenceli ve bir o kadarda beklenmedik sonuçları üreten vakalardandır. Olayı hızlı bir şekilde değerlendirmek adına aşağıdaki kod parçasını göz önüne alabiliriz.

```
using System;
using System.Threading.Tasks;

namespace Disasters
{
   class Program
   {
      static void Main(string[] args)
      {
         for (int i = 0; i < 10; i++)
         {
            Task.Factory.StartNew(() =>
               {
                  for (int j = 0; j < 125000000; j++)
                  {
                     j++;
                     j--;
                     j*=j;
                  }
                  Console.WriteLine("Güncel Task Id : {0}\tGüncel i :
```

```
{1}",Task.CurrentId,i.ToString());
                }
            );
        }

        Console.WriteLine("Kapatmak için bir tuşa basınız");
        Console.ReadLine();
    }
  }
}
```

örnek kod ile çalışma zamanında **10 Task** örneği başlatılmakta ve bunların **lambda ifadeleri(=> Expressions)** içerisinden o anki **i** değerleri ekrana yazdırılmaktadır. Normal şartlarda **i** değerlerinin her bir **Task** örneği için farklı olması beklenir. Ancak çalışma zamanına baktığımızda aşağıdaki enteresan sonuçlar ile karşılaştığımızı görebiliriz.

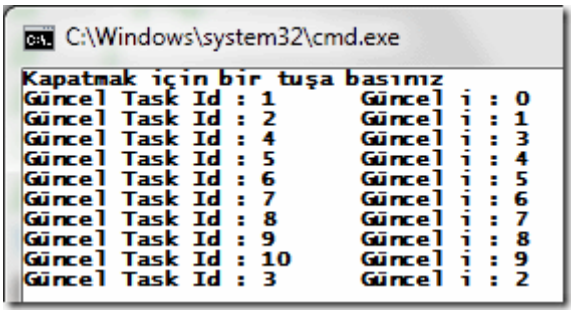


Görüldüğü üzere bütün **Task** örnekleri **for** döngüsü sayacının son değerini ekrana yazdırmaktadır. çok kolay bir şekilde gözden kaçabilecek bu vaka nedeniyle uzun süre ekrana baka kalabilir ve arkadaşlarımızın bize "Kal Gelmiş" demelerine neden olabiliriz. Oysa ki çözüm son derece basittir.

```
using System;
using System.Threading.Tasks;

namespace Disasters
{
   class Program
   {
      static void Main(string[] args)
      {
         for (int i = 0; i < 10; i++)
         {
            Task.Factory.StartNew((s) =>
            {
               for (int j = 0; j < 125000000; j++)
               {
```

```
                j++;
                j--;
                j *= j;
            }
            int currentI = (int)s;

            Console.WriteLine("Güncel Task Id : {0}\tGüncel i : {1}",
Task.CurrentId, currentI.ToString());
        },i
        );
    }

    Console.WriteLine("Kapatmak için bir tuşa basınız");
    Console.ReadLine();
  }
 }
}
```

Bu sefer **StartNew** metodunun farklı bir versiyonu kullanılmıştır. Dikkat edileceği üzere metodun ikinci parametresi olarak **i** değişkeni kullanılmıştır. Bu aslında **State Object** olarak düşünülebilir. Dolayısıyla başlatılan her **Task** örneğine parametre olarak o anki döngü değeri*(i değişkeninin değeri)* geçirilmektedir. **s** isimli değişken, **for** döngüsünden gelen **i** değişkenini **object** tipinden temsil ettiği için de, basit bir **Cast** işlemi yapılması yeterlidir. İşte çalışma zamanı sonuçları.



Görüldüğü gibi Task örnekleri kendilerine atanan i değerlerini ekrana basmıştır.

Elbette farklı vakalar ve felaket senaryoları da söz konusudur. Bu gibi durumları ilerleyen yazılarımızda ele almaya çalışıyor olacağız. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Disasters.rar (22,06 kb) **[örnek Visual Studio 2010 Ultimate Sürümünde Geliştirilmiş ve Test Edilmiştir]**

Silverlight İstemcileri için Duplex Service Geliştirmek (2010-06-18T13:50:00)

wcf,silverlight,wcf service,duplex service,duplex communication,push,

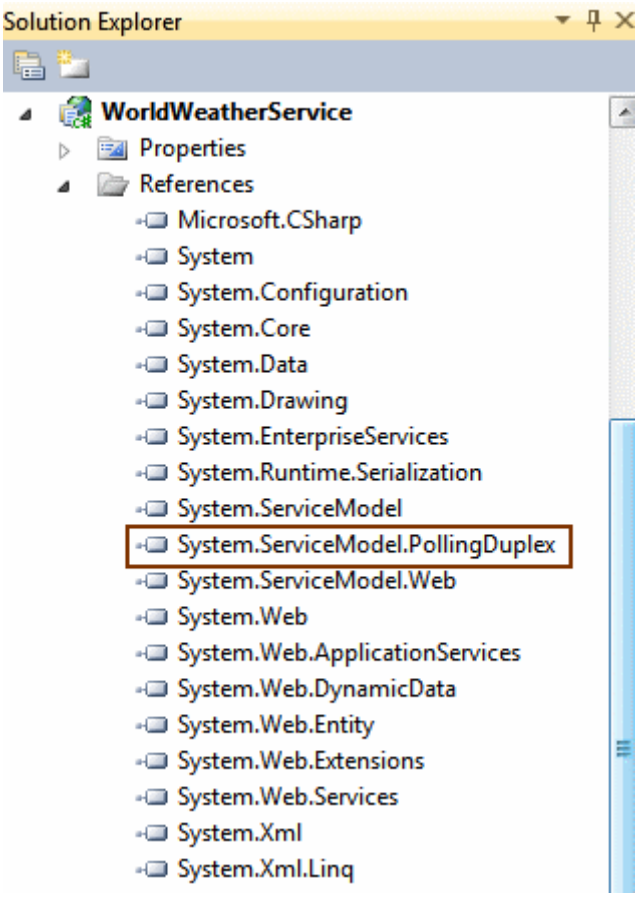
Merhaba Arkadaşlar,

Lost dizisinin müptelası olan arkadaşlar **"Push The Button"** repliğini bilirler. Hikayeye göre **DHARMA** girişimin labaratuvarında yer alan ve **108**dakikadan geriye doğru sayan bir numarator vardır. Zaman sayacı sıfırlanmadan önce toplamları **108** olan **4,8,15,16,23,42** sayı dizisinin klavyeden girilmesi ve **Enter** tuşuna basılması gerekmektedir. Tabi ben Lost dizisinin tüm sezonlarını takip etmemiş ve hatta sonunu getirememiş birisi olarak ne olduğunu tam olarak anlayabilmiş değilim. Lakin bu **Push The Button** mevzusunda düşündüğüm genelde, sunucu üzerindeki bir servisin **Pusher Service** olarak hizmet vermesi olmuştur. Buna göre, servis kendisine bağlı olan istemci modundaki lokasyonlara bir bildiri yapmaktadır diyerek konuyu bir şekilde bağlmaya çalışayım. Bu günkü yazımızda **Silverlight** istemcilerinin **Duplex** iletişim üzerinden hizmet verebilen servisler yardımıyla nasıl tetiklenebileceğini incelemeye çalışıyor olacağız. Sanıyorum **Siverlight** tabanlı **chat** programları geliştirmek isteyenlerin ilgisini çekecek en azından biraz bilgi verecek bir yazı olacaktır.

Bilindiği üzere **WCF(Windows Communication Foundation)** tarafında geliştirilen servislerin **Duplex** iletişimi kullanaraktan istemciler üzerinde operasyonlar gerçekleştirmesi, bir başka deyişle metod çağrılarında bulunabilmeleri mümkündür. Burada çift kanallı olarak gerçekleştirilen bir iletişim söz konusudur. Daha çok **chat**uygulamalarında veya istemcinin her hangibir durum değişikliğinde uyarılması gerektiği vakalarda bu tip servislerden yararlanılabilir. Söz gelimi bu yazımızda geliştireceğimiz **WCF Servis**örneği, istemcilerden aldığı şehir bilgisine göre anlık hava durumu bilgisini döndürecektir. Bu cümle ilk bakışta istemcinin yapacağı normal bir servis çağrısından ve sonucunun alınmasından farksız bir operasyonmuş gibi görünebilir. Ancak gözden kaçırılmaması gereken bir husus vardır; o da hava durumu bilgisinin bildirilme işleminin, servis tarafından istemci üzerindeki bir operasyon çağrısı ile yapılacağıdır.

Tabi yazımızın başlığından da anlayacağınız üzere söz konusu **WCF Servisini** bir **Silverlight** istemcisi üzerinden test etmeye çalışıyor olacağız. Duplex **WCF Servisining**eliştirilmesi başlı başına karmaşık bir süreç gerektiğinden yazımızı iki seriye

bölüyor olacağız. İlk bölümdeki hedefimiz **Duplex** iletişimi sağlayacak olan **WCF Servisini** geliştirmek olacak. İşe **Visual Studio 2010 Ultimate RC** ortamında **WorldWeatherService** isminde bir **WCF Service Application** uygulaması açarak ve hemen **C:\Program Files (x86)\Microsoft SDKs\Silverlight\v4.0\Libraries\Server** adresinde yer alan **System.ServiceModel.PollingDuplex.dll assembly**' ını referans ederek başlayabiliriz. Nitekim bu referans içerisindeki tiplere sunucu tarafında ihtiyacımız olacaktır.



Şimdi servis için gerekli sözleşmeleri yazabiliriz. **WCF Duplex** servisleri iki **sözleşme(Contract)** içermektedir. Bunlardan birisi istemci tarafından çağırılacak olan operasyonları içeren sözleşmedir. Ancak diğer sözleşme **geri bildirim(Callback)** sırasında kullanılacak sözleşmedir. Aşağıda söz konusu sözleşmelere ait **arayüz(Interface)** içerikleri yer almaktadır.

```
using System.ServiceModel;

namespace WorldWeatherService
{
    // Servis sözleşmesinin kullanacağı geri bildirim sözleşmesi CallbackContract özelliği ile bildirilir.
    [ServiceContract(CallbackContract=typeof(IWeatherDuplexClient))]
    public interface IWeatherDuplexService
```

```
    {
        [OperationContract]
        void SetCity(string cityName);
    }

    [ServiceContract]
    public interface IWeatherDuplexClient
    {
        // Servisin istemci tarafında tetikleyeceği Notice operasyonunun geriye bir şey
döndürmeyeceği bir başka deyişle tek yönlü çalışan bir metod olduğu IsOneWay niteliği
sayesinde bildirilir.
        [OperationContract(IsOneWay=true)]
        void Notice(WeatherStatus weather);
    }
}
```

Dikkat edileceği üzere **IWeatherDuplexService** sözleşmesi için kullanılan **ServiceContract** niteliğinde, geri bildirim sözleşmesi **CallbackContract** özelliği yardımıyla bildirilmiştir. Bu sayede servis sözleşmesini uygulayan tip, çalışma zamanında gerekli geri bildirim nesne örneğini değerlendirebilecektir. Diğer yandan servisin istemci üzerinde tetiklemede bulunacağı sözleşme, **IWeatherDuplexClient** adlı **interface** tipi olarak tanımlanmıştır. Bu sözleşme için önem arz eden konu ise **Notice** metodunun **OperationContract** niteliğinde yer alan **IsOneWay**özelliğinin **true** değere sahip olmasıdır. Nitekim servisin yaptığı istemci bazlı geri bildirimlerden bir sonuç beklenmemelidir. Bu da ilgili operasyonun tek yönlü çalışacak şekilde işaretlenmesi ile mümkün olacaktır. Gelelim servis sözleşmesinin uygulandığı tipe.

```
using System;
using System.ServiceModel;
using System.Threading;

namespace WorldWeatherService
{
    public class WeatherDuplexService
        : IWeatherDuplexService
    {
        #region Variables

        IWeatherDuplexClient client;
        string[] posibilities ={"Güneşli",
                                "Yağmurlu",
                                "Parçalı Bulutlu",
                                "Kar Yağışlı",
                                "Tipi",
```

```
                    "Fırtına",
                    "Bulutlu",
                    "Rüzgarlı",
                    "Zaman zaman yağışlı"
                };

        #endregion

        #region IWeatherDuplexService Members

        public void SetCity(string cityName)
        {
            // İstemci kanalı yakalanıyor ki geri bildirim sırasında hangi kanal üzerinden gidileceği bilinsin.
            client = OperationContext.Current.GetCallbackChannel<IWeatherDuplexClient>();
            // Geri bildirimlerde devreye girecek metod bloğunu işaret edecek TimerCallback tipinden bir temsilci kullanılır.

            Random rnd = new Random();
            TimerCallback tCallback = o => {
                // Sembolik olarak bir hava durumu bilgisi oluşturulur
                string summary=posibilities[rnd.Next(0,posibilities.Length-1)];
                int heat = rnd.Next(0, 30);

                // İstemci tarafındaki Notice metodu çağırılır ve söz konusu şehir için anlık hava durumu bilgisi gönderilir(Tabiki hayali olarak)
                client.Notice(new WeatherStatus { City = cityName, Heat = heat, Summary = summary });
            };
            // Olayı simule etmek için,
            // Belirli süre duraksatma yapılır ve sonunda tCallback isimli temsilcinin işaret ettiği metod bloğunun çalıştırılması sağlanır.
            using (Timer tmr = new Timer(tCallback, null, 500, 500))
            {
                Thread.Sleep(2000);
            }
        }

        #endregion
    }
}
```

WeatherDuplexService sınıfı **IWeatherDuplexService arayüzünü(Interfaca)** uygulamaktadır. Bu uyarlamaya göre **SetCity** metodunu

ezmektedir. **SetCity** metodu içerisinde en can alıcı nokta ise, **Callback Channel** nesne referansının yakalanmasıdır. Dikkat edileceği üzere **GetCallbackChannel** metodu **generic** olarak **IWeatherDuplexClient** arayüzünü kullanmaktadır. Buna göre, çalışma zamanında istemcinin servise gönderdiği çağrıya göre yakalayacağı kanalın, **IWeatherDuplexClient** arayüzünü uygulamış bir tip olacağı ortadadır. E haliyle bu arayüzün içerisinde tanımlanmış bir de operasyonumuz bulunmaktadır. **Notice** isimli metod. Dolayısıyla yakalanan kanal üzerinden yapılabilecek olan bir **Notice** operasyon çağrısı mevcuttur ve bu çağrı servis tarafından istemci üzerinde gerçekleştirilecektir. Kod içerisinde sembolik olarak bir gecikme işlemi uygulanmış ve bunun sonucunda **TimerCallback temsilci tipinin(Delegate)** işaret ettiği bir metod gövdesinin de devreye girmesi sağlanmıştır. Bu metod bloğu içerisindeyse **Notice** metod çağrısı gerçekleştirilmekte ve aşağıdaki içeriğe sahip olan **WeatherStatus** tipinden bir nesne örneği üretilmektedir.

```
namespace WorldWeatherService
{
   public class WeatherStatus
   {
      public int Heat { get; set; }
      public string Summary { get; set; }
      public string City { get; set; }
   }
}
```

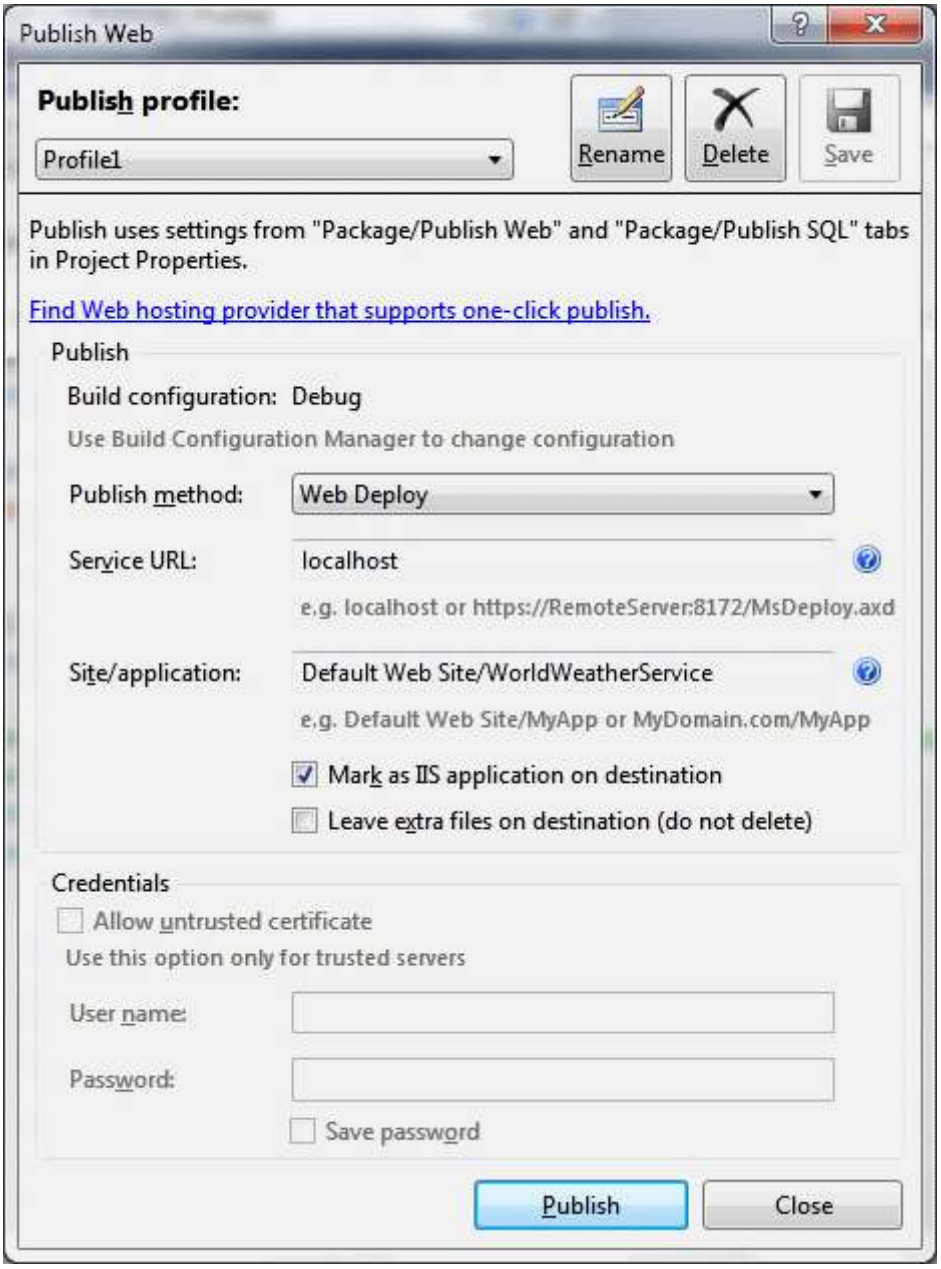
Sırada servis tarafının çalışma zamanını ilgilendiren konfigurasyon ayarlarının yapılması yer almakta. Burada işler biraz karışıyor. Neyseki **MSDN** üzerinden konu ile ilişkili yardımcı dökümanların fazlasıyla yararı olduğunu ifade edebilirim. İşte sunucu uygulamamıza ait **web.config** içeriğimiz.

```
<?xml version="1.0"?>
<configuration>
   <system.serviceModel>
      <extensions>
         <bindingExtensions>
            <add name="duplexHttpBinding"
type="System.ServiceModel.Configuration.PollingDuplexHttpBindingCollectionElem
ent, System.ServiceModel.PollingDuplex, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"/>
         </bindingExtensions>
      </extensions>
      <bindings>
         <duplexHttpBinding>
            <binding name="duplexHttpBindingConfiguration"
duplexMode="MultipleMessagesPerPoll" maxOutputDelay="00:00:05"/>
         </duplexHttpBinding>
```

```
      </bindings>
      <behaviors>
        <serviceBehaviors>
          <behavior name="">
            <serviceMetadata httpGetEnabled="true"/>
            <serviceDebug includeExceptionDetailInFaults="false"/>
          </behavior>
        </serviceBehaviors>
      </behaviors>
      <serviceHostingEnvironment multipleSiteBindingsEnabled="true"/>
      <services>
        <service name="WorldWeatherService.WeatherDuplexService">
          <endpoint address="" binding="duplexHttpBinding"
bindingConfiguration="duplexHttpBindingConfiguration"
contract="WorldWeatherService.IWeatherDuplexService"/>
          <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange"/>
        </service>
      </services>
    </system.serviceModel>
    <system.web>
      <compilation debug="true"/>
  </system.web>
</configuration>
```

Anneciğimmm!!! Sakın korkmayın. Ezberlemeye gerek yok. Ancak bir özet geçmemizde yarar olduğu kanısındayım. öncelikli olarak **Silverlight** tarafı için gerekli bir takım işlemler yapıldığını söyleyebiliriz. Bunlardan ilki bir **bindingExtension** bildirimidir. Sanırım projeyi oluşturduktan sonra neden **System.ServiceModel.PollingDuplex.dll assembly**' ını referans ettiğimizi anlamışsınızdır. Söz konusu **extension** ile yeni bir **bağlayıcı tip(Binding Type)** tanımlayarak kullanıma sunuyoruz. **duplexHttpBinding** olarak isimlendirdiğimiz bağlayıcı tipin bir takım özellikleri de*(duplexMode, maxOutputDelay)* belirtilmiş durumdadır.

Peki ya bundan sonrası? örneğimizi **Asp.Net Development Server** yerine **IIS** altında konuşlandıracak şekilde tesis edebiliriz. Aslında projeyi doğrudan **IIS** altına **Publish** ettikten sonra servisi bir tarayıcı uygulama ile açarak sorunsuz bir şekilde çağırılıp çağırılmadığını görmekte yarar olacağı kanısındayım. **Publish** seçenekleri aşağıdakine resimde görüldüğü gibi yapılabilir.

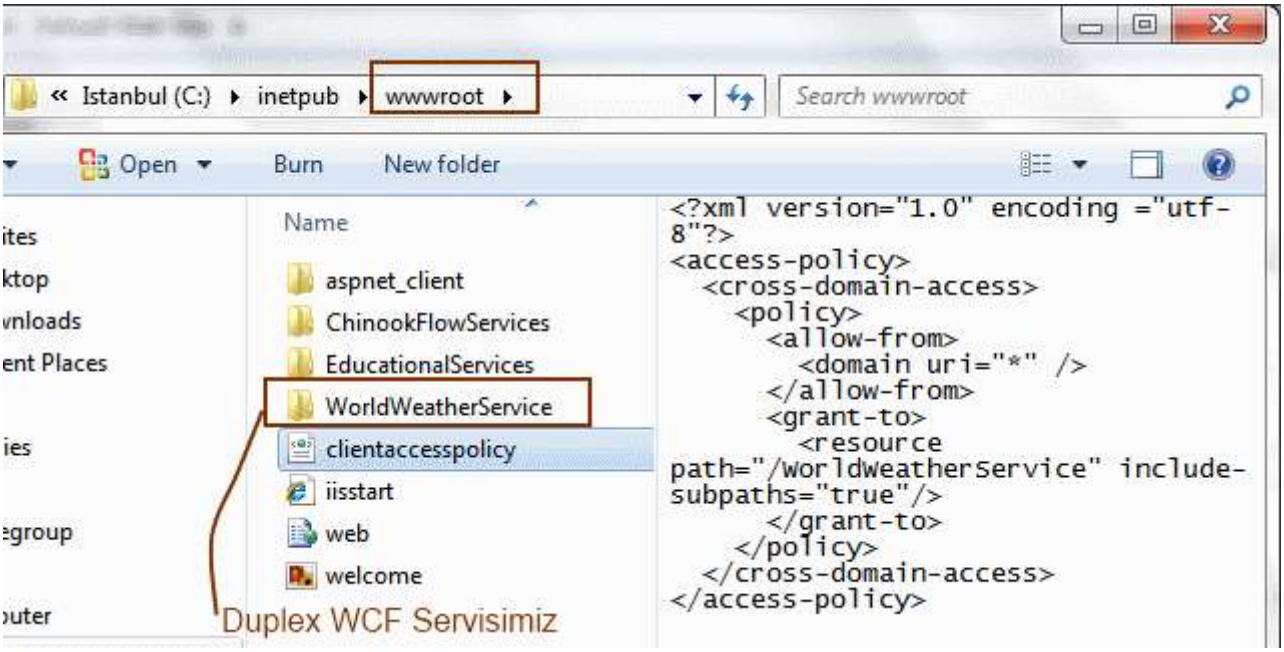


örneğimizi test ettiğimiz aşağıdaki gibi bir sonuç ile karşılaşırsak her şey yoldundadır diyebiliriz. Diyebiliriz çünkü asıl test **Silverlight** istemcisi tarafından servise erişmeye ve kullanmaya çalıştığımızda oluşacaktır.

Servis tarafında dikkat edilmesi gereken noktalardan birisi de **Client Access Policy** konusudur. Servisimizi **IIS** altına **Publish** etsek bile herhangibir **Silverlight** istemcisinin kullanabilmesi için**ClientAccessPolicy.xml** içeriğinin **Domain Root** altında yer alması gerekmektedir. Söz konusu dosyasının içeriğini etkileyen pek çok faktör vardır ve açıkçası bu kadar detaya girmemize şimdilik gerek yoktur. Ancak en geçerli kaynaklardan birisi olarak **Time Heuer**' in blog girdisinden ve tabiki **Microsoft**' un **Network Security Access Restricions in Silverlight** yazısından yararlanabilirsiniz. örneğimiz için aşağıdaki gibi bir içerik yeterli olacaktır.

```
<?xml version="1.0" encoding ="utf-8"?>
<access-policy>
  <cross-domain-access>
    <policy>
      <allow-from>
        <domain uri="*" />
      </allow-from>
      <grant-to>
        <resource path="/WorldWeatherService" include-subpaths="true"/>
      </grant-to>
    </policy>
  </cross-domain-access>
</access-policy>
```

Bu içeriğe sahip olan **ClientAccessPolicy.xml** dosyasının ise **WCF** Servisimizi **Publish** ettiğimiz **IIS** sunucusundaki ilgili **Domain**' e ait **root** klasörde yer alması gerekmektedir. Aşağıdaki şekilde görüldüğü gibi.



Artık farklı bir **domainde** yer alan herhangibir **Silverlight** istemcisi **WorldWeatherService**' ini kullanabilecektir. Artık geride istemci tarafının yazılması ve test edilmesinden başka bir şey kalmamıştır. Ancak biraz nefes alalım ve bunu bir sonraki yazımıza bırakalım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WorldWeatherService.rar (157,92 kb) **[örnek Visual Studio 2010 Ultimate RC sürümü üzerinde geliştirilmiş ve test edilmiştir.]**

Microsoft Teknoloji Günleri Akşam Sınıfı - Gün 2 - .Net 4.0 ile Paralel Programlama (2010-06-17T17:20:00)

microsoft teknoloji günleri,microsoft,bizspark,gelişim atölyesi,c# 4.0,parallel programming,wcf,wcf eco system,windows server appfabric,wf 4.0,asp.net 4.0,visual basic 2010,wpf 4.0,

Tarih
22 Haziran 2010 Salı

Saat
19.00 - 21.30

Yer
Microsoft İstanbul Ofisi

Eğitmen:
Burak Selim Şenyurt
Microsoft MVP

ay boyunca siz yazılım geliştiren ve tasarım yapan iş ortaklarımızla birlikte olacağız.

Aşağıda detaylarını paylaştığımız ve sizler için hayli faydalı olacağına inandığımız Microsoft Teknoloji Günleri Akşam Sınıfı Etkinliğimize kaydınızı hemen yaptırabilirsiniz.

Ders İçerikleri:

1. **25.Mayıs.2010/Salı: C# 4.0 ile Gelen Yenilikler : (2 Saat)** Bu eğitimde C# programlama dilinin 4.0 versiyonu ile birlikte gelen yenilikleri tanıtmaya çalışılmakta ve özellikle, Dynamic diller ile Office API vb yapılar ile olan etkileşim üzerine örnekler geliştirilmektedir. Eğitimde temel olarak dynamic keyword, optional and named parameters, PIA, Co-Contra Variance Generics konularına değinilmektedir.

*2. **22.Haziran.2010/Salı: .Net 4.0 ile Paralel Programlama : (2 Saat)** Uzun süredir ev bilgisayarlarımız dahil en az iki çekirdekli sistemler üzerinde çalışabiliyoruz. Son zamanlarda 8 çekirdeğe kadar çıkabilen yeni nesil işlemciler üzerinde paralel programlama yeteneklerini sonuna kadar kullanıp daha performanslı, hızlı, verimli ve ölçeklenebilir uygulamalar geliştirmeye ne dersiniz? İşte tam size göre bir eğitim. Bu eğitim ile .Net 4.0 kütüphanelerini kullanarak kolay bir şekilde nasıl paralel programlama yapabileceğinizi göreceksiniz.*

3. **20.Temmuz.2010/Salı: WCF ile Servis Yaklaşımı : (2 Saat)** .Net 3.0 ile birlikte duyurulan ve tek bir servis geliştirme metodolojisi sunarak daha önceki dağıtık mimari geliştirme tiplerini (Xml Web Services, .Net Remoting, MSMQ vb...) bir çatı altında birleştiren Windows Communication Foundation konulu eğitimdir. Eğitimde özellikle .Net 4.0 ile birlikte gelen yeniliklere de değinilmekte olup asıl amaç WCF ile uygulama geliştirme şekillerinden bir kaçını göstermek ve tanıtmaktır.

4. **20.Ağustos.2010/Cuma: WCF Eco System : (3**

Saat)WCF alt yapısı üzerine kurulu olan WCF Eco System içerisinde Data Services, Workflow Services, RIA Services, WebHttp Services ve Core Services tipleri yer almaktadır. Bu eğitimde Data Services, Workflow Services, RIA Services ve WebHttp Services konulu örnekler geliştirilmekte olup, söz konusu alt yapı ile değerlendirilebilecek hazır servis modelleri irdelenmektedir.

5. **20.Eylül.2010/Pazartesi: Windows Server AppFabric : (2 Saat)** Bu eğitimde WCF ve Workflow servis örneklerinin izlenmesi, sorunların teşhis edilmesi, örneklerin yaşam döngülerinin takibi, konfigurasyon ayarlarının belirlenmesi ve pek çok yönetimsel konuda geliştiriciler ile IT yöneticilerinin daha kolay anlaşabilmelerini de sağlayan Dublin kod adlı Windows Server AppFabric ürün ailesi incelenmektedir. özellikle IIS üzerine gelen eklentiler ile söz konusu yönetsel işlemlerin nasıl yapılabildiği derinlemesine incelenmektedir.

6. **20.Ekim.2010/çarşamba: Worfklow Foundation 4.0 : (3 Saat)** WF 4.0 beraberinde pek çok köklü yenilik ile gelmektedir. Geliştirilen Base Activity Library, paralel programlama desteği, veri akışı için gelen Argument, Variable gibi kavramlar ve daha pek çoğunun ele alındığı eğitimde basit örnekler ile WF modelin tanıtılmaya çalışılmaktadır.

7. **23.Kasım.2010/Salı: Asp.Net 4.0 : (3 Saat)** Web Programlama' nın .Net 4.0 ile birlikte gelen yeni yüzünü görmeye hazır mısınız? Pek çok yeni özellik ile birlikte gelen Asp.Net 4.0' ın anlatıldığı bu eğitimde, temelden orta seviyeye kadar basit bir web uygulaması tasarlanmakta ve konunun daha iyi kavranabilmesi amaçlanmaktadır.

8. **20.Aralık.2010/Pazartesi: Visual Basic 2010 : (2 Saat)**Bu eğitimde Visual Basic 2010 programlama dili ile birlikte gelen pek çok yeni özellik üzerinde durulmakta ve geliştirilen örnekler ile bu kavramlar pekiştirilmeye çalışılmaktadır. Bu noktada AutoImplemented Properties, Collection Initializers, Implicit Line Continuation, Mutlipe Lambda

Expressions, Dynamic keyword, Type Equivalance Support gibi konular üzerinde durulmaktadır.

9. **20.Ocak.2011/Perşembe: WPF 4.0 ile Windows Programlama : (3 Saat)** .Net 3.0 ile birlikte duyurulan Windows Presentation Foundation modeli ile zengin kullanıcı deneyimine sahip windows uygulamaları tasarlanabilmektedir. özellikle Windows 7 üzerinde en iyi kullanıcı deneyimini sunan WPF 4.0 ile birlikte gelen yenilikleri öğrenmeye ne dersiniz?

Türkiyeâ€™ nin Açık Kaynak Topluluğu Birliktegeliştir Yeni Versiyonu İle Yayında (2010-06-15T23:14:00)

birlikte gelistir,açık kaynak kod,

Açık kaynak kodlu proje geliştirme yaklaşımının giderek yaygınlaştığı günümüz yazılım dünyasında **Microsoft** platform ve araçları kullanılarak birçok açık kaynak kodlu uygulama geliştirilmekte ve **Microsoft**'un kendisi de bu konuda çalışmalar yapmaktadır.

Açık kaynak kodlu bir içerik yönetim sistemi olan **Umbraco** üzerine kurulan **Birliktegeliştir**, **Türkiye**'de de **Microsoft** platformlarında açık kaynaklı projeler geliştirilmesi ve mevcut açık kaynak kodlu projelerin kullanımının yaygınlaştırılmasını amaçlamaktadır.

Zaman içinde yaygınlaşan açık kaynak kod topluluğuna daha iyi içerik sağlamak amacıyla **Birliktegeliştir** yeni arayüzü ve yeni özellikleriyle yayında.

Sizlerin de kendi projelerinizi ve kendi içeriğinizi rahatlıkla paylaşabileceğiniz **Birliktegeliştir**'de ayrıca **Türkiye**'de geliştirilmiş açık kaynak kodlu projelere, bugüne kadar dünyada yapılmış açık kaynak kodlu uygulamaların kullanımı ile ilgili makale, video gibi **Türkçe** kaynaklara ulaşabilirsiniz.

Bunun yanısıra uygulama geliştiricilerin kodlama sırasında çoğunlukla kullandığı kod bloklarına **Kod Parçaları** kısmından ulaşabilirsiniz.

Birliktegeliştir topluluğuna www.birliktegelistir.com adresinden ulaşabilirsiniz.

E-Mail: birliktegelistir@birliktegelistir.com
Twitter: www.twitter.com/bgelistir

Workflow Foundation Öğreniyorum - Ders 8 - Exception Handling (2010-06-15T09:20:00)

workflow foundation 4.0,workflow foundation,visual studio 2010,exception handling,error handling activities,

Merhaba Arkadaşlar,

NedirTv?com sponsorluğunda sürdürdüğümüz **"Workflow Foundation 4.0 öğreniyorum"** serimizin **dokuzuncu(8+1)** dersi ile karşınızdayız. Bu dersimizde, **çalışma Zamanında(Runtime)** oluşabilecek istisnai durumların kontrollü bir şekilde ele alınmasında önem arz eden **Exception Handling**konusunu ele alıyoruz. Bu amaçla geliştirilen örnekte, **Workflow Foundation 4.0** modelinde istisna yönetiminin, kod tarafında **try...catch...finally** bloğu kullanmak kadar basit olduğuna şahit olacağız. **Workflow Console Appplication** tipinden geliştireceğimiz projede, **TryCatch** ve **Throw Activity** bileşenleri üzerinde duruyor ve ayrıca geliştirici tanımlı bir istisna tipini nasıl kullanabileceğimize de bakıyoruz. İyi seyirler dilerim.

Ders 8 - Exception Handling

Süre : 20:04

Dosya Boyutu : 28.7 Mb

örnek : Lesson8.rar (53,38 kb)

Mp4 Formatında İndirmek İçin Tıklayın

Parent-Child Tasks Kavramı (2010-06-11T14:00:00)

tpl,task parallel library,parallel programming,parallel computing,

 Merhaba Arkadaşlar,

Planlama gerçek hayatta her zaman karşımıza çıkan ve yaşamımızın, işlerimizin düzenli devam edebilmesi için gereken olmazsa olmazlar arasında yer alan bir kavramdır. Toplantıların planlanmasından tutun da, işlerin hangi sırada yapılacağına karar verilmesine kadar pek çok yerde planlamanın önemini görürüz. Aslında başarılı sistemlerin tasarlanması, çalışması ve istenen sonuçları üretmesi iyi planlamayla ilişkilidir. Tasarımın planlanması, kaynakların planlanması, sistemin önceliklerinin planlanması, müşteri toplantılarının planlanması vs...Bazen kafamızda hayatımızın ilerleyişini planlarız ve bazende yazdığımız kodun fonksiyonelliklerini.

Dolayısıyla planlamanın yazılımın pek çok noktasında da son derece önemli bir rol üstlendiğine şahit oluruz. Aslında planlı olmak, neyin ne zaman nasıl ve ne şekilde yapılacağının bilinmesi, ortaya çıkacak sonuçlarda nasıl hareket edileceğinin tespit edilmesi noktasında son derece hayatidir.

Planlama, bu yazımızda ele alacağımız gibi **Task Parallel Library** içerisinde de ele alınan bir konudur. özellikle **Parallel.ForEach, Parallel.For, Parallel.Invoke** gibi metodlar zaten paralel çalışma için gerekli planlamaları*(yapılandırmaları ki neden yapılandırma dediğimi biraz sonra anlayacağız)* kendi iç yapılarında gerçekleştiren fonksiyonellikler sunmaktadır.

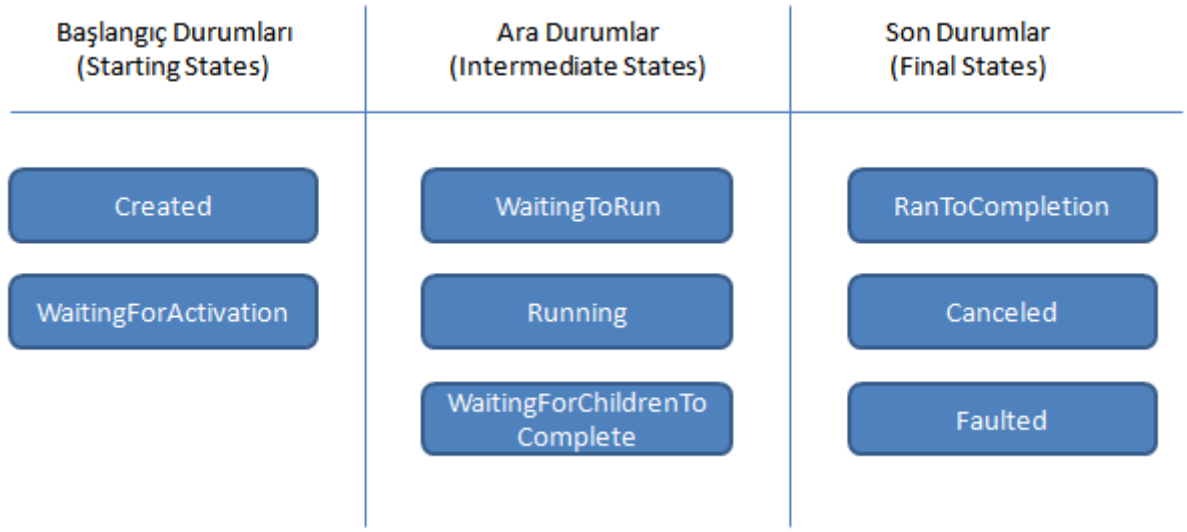
Ancak **Task** nesne örneklerinin devreye girdiği noktada, **Parent-Child** ilişkiler kurularak **Planlanmış/Yapılandırılmış görevlerin(Structured Tasks)** oluşturulması da mümkündür.

Kişisel Not : *Gerçi **Structured** kelimesini **Yapılandırılmış, Denetim Altında Olan** anlamlarında da kullanabiliriz ancak burada mevzu bahis olan konu bana göre **Task** nesne örneklerinin planlı bir şekilde **Parent-Child** ilişki içerisine alınması ve çalıştırılmasıdır. Tabi ki evdeki hesap her zaman için çarşıya uymaz, uymayabilir. Bu nedenle **Parent Task** nesne örneklerinin **n** sayıda **Child Task** örneğini içeriyor olması da, bir **yapının-Structure** kurulması olarak düşünülebilir.*

İşte bu yazımızda **Task** nesne örnekleri arasında **Parent-Child** ilişkiyi incelemeye çalışıyor olacağız. İşe ortam gereklilikleri ile başlamakta yarar olduğu kanısındayım.

Task nesne örnekleri arasında **Parent-Child** ilişki oluşturulabilmesi için gerekli iki şart vardır. İlk olarak **Child** olacak **Task** örneğinin, **Parent Task** örneğinin çalıştığı **yaşam döngüsü(Life Cycle)** içerisindeyken **oluşturulması(Create)** gerekmektedir. İkincil olarak **Child Task** örneklerinin oluşturulma işlemleri sırasında, **TaskCreationOptions enum** sabitlerinden**AttachedToParent** değeri ile üretilmesi gerekmektedir. Bir başka deyişle oluşturulan **Task** nesne örneğinin, içerisinde çalıştığı **Task** nesne örneğinin **Child**' ı olacağının bilinçli bir şekilde bildirilmesi gerekir. Nitekim normal şartlarda varsayılan olarak tüm **Task** nesne örnekleri **Detached** pozisyondadır. çok doğal olarak **Parent-Child Task** ilişkisi için en az iki**Task** nesne örneğinin olması gerektiği ortadadır.

Child Task örnekleri, **Parent Task** örneklerine **dönüş durumlarının(Return States)** değerlerine göre etkide bulunabilirler. **TaskStatus enum** sabiti için olası değerler bu noktada önem taşımaktadır. Aşağıdaki şemada olası durumların hangi zaman aralıklarında anlam kazandığı gösterilmeye çalışılmaktadır.

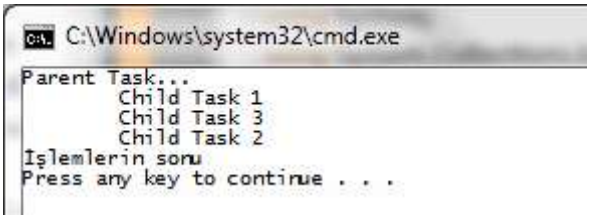


Bir başka deyişe, **Child Task** nesne örnekleri dönüş değerlerine göre **Parent** örneklerin **yaşam döngülerine(Life Cycle)** etkilde bulunurlar diyebiliriz. Peki en basit haliyle **Parent-Child Task**ilişkisini canlandırmak istersek nasıl bir kod deseni oluşturmamız gerekir? Aşağıda **Visual Studio 2010 Ultimate RC** ortamında buna istinaden oluşturulmuş bir kod örneği görülmektedir.

```
using System;
using System.Threading.Tasks;

namespace StructuredTasking
{
   class Program
   {
      static void Main(string[] args)
      {
```

```
        Task parentTask = Task.Factory.StartNew(
            () =>
            {
                Console.WriteLine("Parent Task...");
                Task childTaskOne = Task.Factory.StartNew(() => {
Console.WriteLine("\tChild Task 1"); }, TaskCreationOptions.AttachedToParent);
                Task childTaskTwo = Task.Factory.StartNew(() => {
Console.WriteLine("\tChild Task 2"); }, TaskCreationOptions.AttachedToParent);
                Task childTaskThree = Task.Factory.StartNew(() => {
Console.WriteLine("\tChild Task 3"); }, TaskCreationOptions.AttachedToParent);
            }
        );
        parentTask.Wait();
        Console.WriteLine("İşlemlerin sonu");
    }
  }
}
```

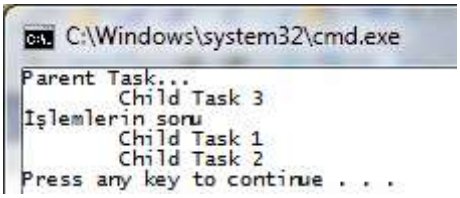
Kod parçasında dikkat edileceği üzere **childTaskOne**, **childTaskTwo** ve **childTaskThree** isimli **Task** nesne örnekleri **parentTask** nesne örneğinin oluşturulduğu metod içerisinde üretilmiş ve başlatışmıştır. Buna göre **çalışma zamanında(Runtime)** aşağıdakine benzer bir sonuç ile karşılaşılabilir.



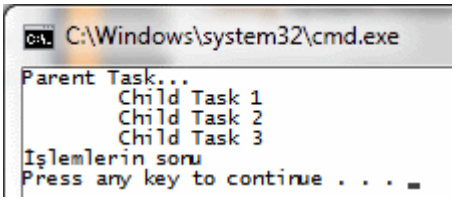
Karşılaşılabilir diyoruz, nitekim **Parent Task** nesne örneğine dahil olan **Child Task** nesne örnekleri farklı sıralarda çalıştırılabilirler. örneğimizde **Task** sırası **1,3,2** şeklindedir ama bu sıra sabit ve kesin değildir. Yani **3,2,1** veya **3,1,2** vb sonuçlar elde edilebilir.*(Toplamda 6 farklı sonuç olabileceğini de ifade edebiliriz)*

Biraz önce **Child Task** nesne örneklerinin **Parent Task** örneğine dahil olmaları için **AttachedToParent** sabit değerini belirtmeleri gerektiğini söylemiştik. Aksi durumda varsayılan olarak **Detached**olduklarını ifade etmiştik. Yukarıdaki kod parçasında **AttachedToParent** değerlerini kullanmassak aşağıdaki ekran görüntülerinde yer alanlara benzer sonuçlar ile karşılaşırız.

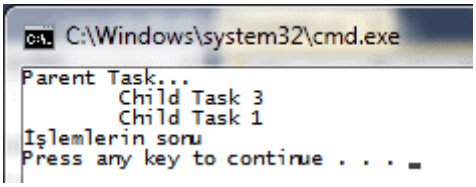
İlk deneme;



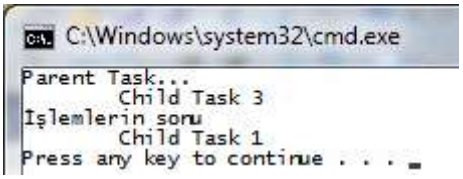
İkinci deneme;



üçüncü deneme;



Dördüncü deneme;



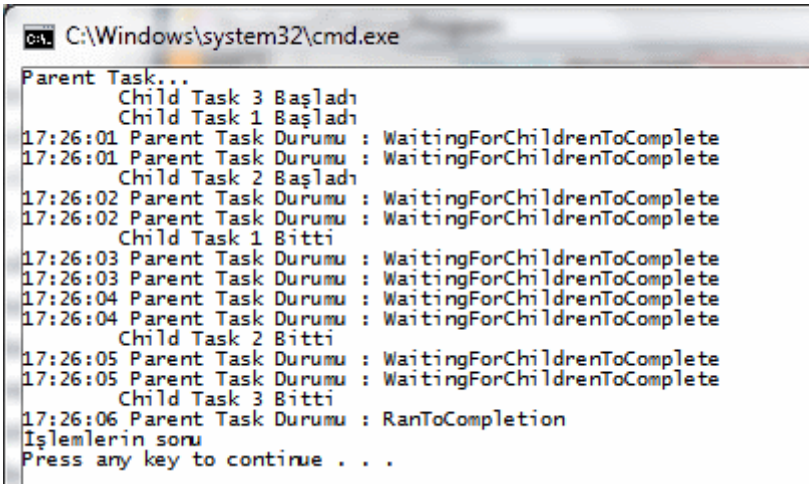
Görüldüğü gibi bilinçli olarak bir planlama belirtilmediğinden **Task** örneklerinin yapılandırılmasında sorunlar oluşmuş, bazı çalışmalarda bazı **Task** örneklerinin sonuçlarıalınamamıştır.*(3 ve 4 teki durumlar)*

Parent-Child Task senaryolarında dikkat edilmesi gereken hususlardan birisi de, **Parent task** nesne örneğinin **Final State** zaman dilimine girebilmesi için, içerdiği **Child Task** nesne örneklerinin çalışmalarını tamamlamış olmaları gerektiğidir. Bu durumu analiz etmek için aşağıdaki kod parçasını göz önüne alabiliriz.

```
using System;
using System.Threading.Tasks;
using System.Threading;

namespace StructuredTasking
{
   class Program
   {
      static void Main(string[] args)
```

```
      {
         Task parentTask = Task.Factory.StartNew(
            () =>
            {
               Console.WriteLine("Parent Task...");

               #region IsCompleted

               Task childTaskOne = Task.Factory.StartNew(() => {
                  Console.WriteLine("\tChild Task 1 Başladı");
                  Thread.Sleep(2000);
                  Console.WriteLine("\tChild Task 1 Bitti");
               }, TaskCreationOptions.AttachedToParent);
               Task childTaskTwo = Task.Factory.StartNew(() => {
                  Console.WriteLine("\tChild Task 2 Başladı");
                  Thread.Sleep(3000);
                  Console.WriteLine("\tChild Task 2 Bitti");
               }, TaskCreationOptions.AttachedToParent);
               Task childTaskThree = Task.Factory.StartNew(() => {
                  Console.WriteLine("\tChild Task 3 Başladı");
                  Thread.Sleep(5000);
                  Console.WriteLine("\tChild Task 3 Bitti");
               }, TaskCreationOptions.AttachedToParent);

               #endregion
            }
         );

         while (!parentTask.IsCompleted)
         {
            Console.WriteLine("{0} Parent Task Durumu :
{1}",DateTime.Now.ToLongTimeString(),parentTask.Status);
            Thread.Sleep(500);
         }
         Console.WriteLine("{0} Parent Task Durumu : {1}",
DateTime.Now.ToLongTimeString(), parentTask.Status);
         Console.WriteLine("İşlemlerin sonu");
      }
   }
}
```

Kodu çalıştırdığımızda aşağıdakine benzer sonuçlar ile karşılaşırız. **Child Task**' ler içerisinde bilinçli olarak **Thread.Sleep** metodu kullanılmış ve uygulamanın belirli süreler boyunca duraksatılması sağlanmıştır. while döngüsünde ise **Parent Task** nesne örneğinin tamamlanıp tamamlanmadığı sürekli olarak denetlenmektedir.



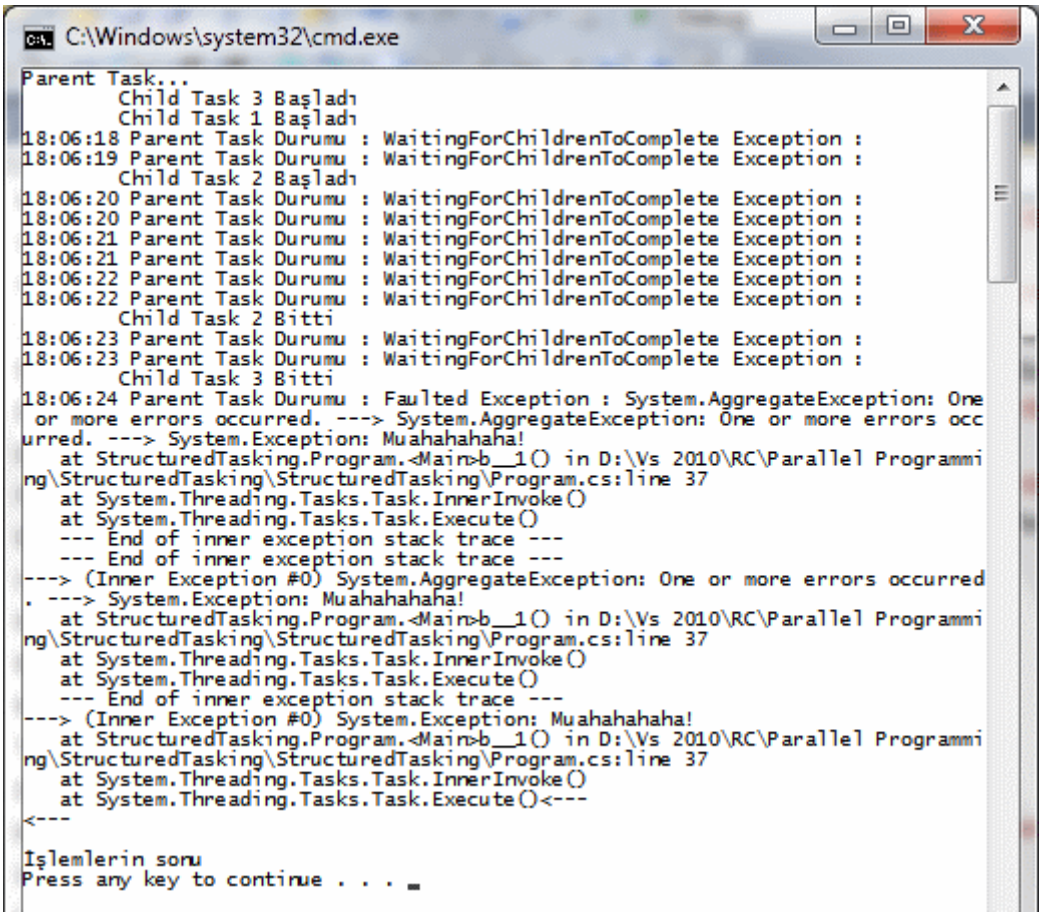
Dikkat edileceği üzere **parentTask** nesne örneğinin **RanToCompletion** moduna, bir başka deyişle **Final State**' e geçmesi için **Child Task** nesne örneklerinin tamamının bitmesi beklenmiştir. üstelik **Child Task** nesne örnekleri çalışmalarını sürdürürken **Parent Task** nesne örneğine kendisini beklemesini bildirmektedir ki bu durumda **Parent Task** ara zaman diliminde durmaktadır ve bu nedenle **WaitingForChildrenToComplete** modundadır.

Parent Task nesne örneğinin **Final State** zaman dilimine girmesi anında olası 3 **Status** değeri bulunmaktadır. Şemamızdan hatırlayacağınız üzere bunlar **RanToCompletion, Canceled** ve**Faulted** olarak belirlenmiştir. **Child Task** örneklerinin başarılı bir şekilde tamamlanmış olmaları, **Parent Task** örneğinin **RanToCompletion** değeri üretmesi için yeterlidir. Peki ya **Child Task** nesne örneklerinden herhangibirinin başlattığı kod içerisinden çalışma zamanına bir **Exception** fırlatılırsa? 😉 Söz gelimi bir önceki örnek kodumuzda yer alan **Child Task**örneklerinden birisinden bir **Exception** fırlattığımızı ve bunu yakalamak istediğimizi düşünelim.

```
Task childTaskOne = Task.Factory.StartNew(() => {
                    Console.WriteLine("\tChild Task 1 Başladı");
                    Thread.Sleep(2000);
                    throw new Exception("Muahahahaha!");
                }, TaskCreationOptions.AttachedToParent);

...

while (!parentTask.IsCompleted)
        {
            Console.WriteLine("{0} Parent Task Durumu : {1} Exception : {2}",
DateTime.Now.ToLongTimeString(), parentTask.Status, parentTask.Exception);
            Thread.Sleep(500);
        }
```

Console.WriteLine("{0} Parent Task Durumu : {1} **Exception : {2}**", DateTime.Now.ToLongTimeString(), parentTask.Status,**parentTask.Exception**);
Console.WriteLine("İşlemlerin sonu");

Bu durumda uygulama aşağıdaki örnek çıktıyı verecektir.



Dikkat edilmesi gereken 3 önemli nokta vardır. **Parent Task** nesne örneği **Faulted Status** değerini üreterek sonlanmıştır. Diğer yandan ilk **Child Task,** bir **Exception** ile sonlandırılsa bile diğer **Task**' lerin başlattığı işler yürütülmeyi sürdürmüş ve tamamlanmıştır. Bu dikkat edilmesi gereken bir durumdur. Nitekim **Child Task**' ler ortak bir takım verileri etkiliyor olabilirler ki bu durumda herhangibir istisna, diğer **Task**' lerin yanlış veriler üzerinden işlem yapmaya devam etmesine neden olabilir. üçüncü nokta ise **Parent Task** nesne örneğinin **Exception**özelliğinin değeridir. Yukarıdaki senaryoya göre bu özellik, **Parent Task** nesne örneği **WaitingForChildrenToComplete** durumundayken **null** değer döndürmektedir. Bir başka deyişle **Child Task** içerisinden fırlatılan **Exception** nesnesi aslında bu zaman dilimi içerisinde **Parent Task** örneğine bildirilmemektedir. Ancak **Faulted** durumuna düşüldükten sonra söz konusu **Exception**nesnesi yakalanmaktadır.

Son olarak **Parent-Child** ilişki ile ilgili olarak şu notu düşebiliriz; **Parent Task** nesne örnekleri, kaç adet **Child Task** örneği içerdiğini bilmektedir. Bir başka deyişle kendisine eklenen **Child Task**' lerin sayısını tutar. Dolayısıyla **Child Task** nesne örneklerinin, dahil

oldukları **Parent Task** örneğine bir referans bildiriminde bulunduğunu ve hatta tamamlanma durumlarını ilettiklerini ifade edebiliriz. Parent-Child Task' ler arasındaki ilişkiyi fırsat buldukça incelemeye devam ediyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

StructuredTasking_RTM.rar (23,32 kb) **[örnek Visual Studio 2010 Ultimate RTM Sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Workflow Foundation Öğreniyorum Ödülünü Sahibi Belirledi ve Nihayet Kitabına Kavuştu (2010-06-10T10:15:00)

workflow foundation 4.0,head first design patterns,book,

Merhaba Arkadaşlar,

Bir süredir NedirTv?com sponsorluğunda devam ettirdiğimiz "Workflow Foundation 4.0 öğreniyorum" görsel eğitim serimize devam etmekteyiz. Bu serinin 5+1=6ncı dersinde Argument kavramına değinmiş ve sunumun sonunda bir de **ödüllü soru** sormuştuk.

Sorunun doğru cevabını verenler arasından yapılan çekilişte **Ergün Yücel** arkadaşımız ödülü kazanan kişi oldu. ödülümüz ise **Amazon**sitesinden 30 dolara kadarlık bir kitap.

Ergün Yücel arkadaşımız bu konuda oldukça başarılı bir tercih yaptı ve tasarım kalıplarını gerçek hayat örnekleri ile son derece sağlam anlatanHead First Design Patterns kitabını seçti.

Kendisine **Workflow Foundation öğreniyorum** serisine gösterdiği ilgi için teşekkür ediyorum. Ayrıca kendisini verdiği doğru cevap için de kutluyorum.

Tebrikler Ergün Bey. çalışmalarınızda başarılar.

Ve işte sonunda kargo Ergün Bey' in eline ulaştı!!! 😀

Kendisi sağolsun ödülle birlikte bir fotoğrafını da bizimle paylaştı. Aslında kargonun kendisine ulaşması beklediğimden uzun sürdü. Her zamanki gibi standart olarak **Amazon.co.uk** üzerinden **Priority Express** ile ürünü istedim. Genelde **UPS** firması

kargoları getirip elden teslim etmekte. üstelik sabah verdiğim bir siparişin ertesi gün öğle saatlerinde elimde olduğuna da şahit oldum. Lakin bu sefer devreye bir Fransız kargo firması girdi. **Cronopost**. Firmanın sitesinde gönderi takibini yapmak istediğimde Fransızca bir portal ile karşılaştım. Ekrana tam bir Fransız gibi baktığımdan hiç bir şey anlamadım. Kargo neredeydi, ne yapmaktaydı, gecikme söz konusu muydu?

Halbuki bu tip uluslararası bir kargo şirketinin ana sayfasında dil desteği olması gerekirdi diye düşünürken Ergün Bey ile olan mailleşmelerimiz sonrası kendisinin ingilizce sayfasını da bulduğunu öğrendim. Hiç olmassa ana sayfaya bir kaç ülke bayrağı koysalar da, insanlar kolayca **Multi - Language** desteğini alabilse. Fransız milliyetçiliği dedikleri bu mu acaba? Hatta kargo firmaları gönderilerini harita üstünde gösterseler daha güzel olmaz mı? örneğin kargo uçakta iken harita üzerindeki minik bir uçak ikonu da eş zamanlı hareket etse. Ne manzara olurdu ama değil mi?

Tabi ben Ergün Bey ile mailleşirken boş durmadım ve şirkette yan masamda oturan Fransa doğumlu Abdullah arkadaşımdan yardım istedim. Abdullah, **Sharepoint** konusunda şirketimizin en önemli danışmanlarındandır. üstelik sayısız proje tecrübesi bulunmaktadır. Hatta yandaki resimde biraz yorgun gözükmesinin sebebi de şu anda birden fazla proje ile ilgileniyor olmasıdır. Bu özelliklerinin yanında ana dili gibi **Fransızca** ve **İngilizce'** de bilmektedir.

Kendisinden yardım istediğimde önce sayfaya baktı ve sonrasında yazılanları akıcı ve düzgün bir şekilde **Fransızca** okudu*(Fransızca çok güzel bir dil aslında)* Sonra şöyle dedi "Aaaa...Bu kargo bizim orada!" Sonunda **Abdullah** arkadaşımdan kargo ile ilişkili detayları öğrendim. Kargo gideceği uçağı kaçırmıştı ve bir sonraki uçuş için beklemedeydi. İşte tam bir skandal. Daha da garibi, Ergün bey ile haberleştiğimde gönderinin **Yurt İçi Kargo** ile ulaştırıldığını öğrendim. İlginç bir durum. Nitekim Priority Express seçeneğinde gönderi en başından teslim edilen ana kadar **UPS** firması tarafından taşınmaktaydı. Ancak sonuç itibariyle kitap Ergün Bey' e ulaştı.

İlerleyen zamanlarda yeni ödüllerimizde olacak. Tabi bunlar birer sürpriz. Bizi izlemeye devam edin.

Workflow Foundation Öğreniyorum - Ders 7 - Homework (2010-06-08T11:10:00)

workflow foundation 4.0,workflow foundation,visual studio 2010,

Merhaba Arkadaşlar,

NedirTv?com sponsorluğunda yürüttüğümüz **"Workflow Foundation 4.0 öğreniyorum"** serimizin **sekizinci(7+1)** dersi ile karşınızdayız. Bu bölümde daha önceki görsel derslerimizde ele aldığımız konuları pekiştirmek amacıyla basit bir ev ödevi geliştiriyor olacağız. Bu ödevde,**kullanıcı tanımlı tiplerin(Class ve Enum sabiti)** değerlendirilmesini, **Argument** ve **Variable** kavramlarını, **Sequence, ForEach<T>, Switch<T>, If, Assign, WriteLine** gibi temel aktivite bileşenlerini, çalışma zamanında Workflow üzerinden **veri akışı için Argument' ların kullanımını** tekrar ederek, öğrendiklerimizin bir kısmını pekiştirmeye gayret edeceğiz. Yine bebek adımları attığımızdan bir **Workflow Console Application** projesi üzerinde çalışacağız. Bakalım nasıl bir sonuç ortaya çıkacak. İyi seyirler dilerim.

Ders 7 - Homework

Süre : 20:18

Dosya Boyutu : 27.3 Mb

örnek : Lesson7.rar (59,96 kb)

Mp4 Formatında İndirmek İçin Tıklayın

WCF Service' lerine Silverlight İstemcilerinden Channel Bazlı Erişim (2010-06-02T10:55:00)

 Merhaba Arkadaşlar,

Bir zamanlar*(aslında çok uzun zaman olmadı ayrılalı)* özel bir eğitim firmasında **Yazılım Eğitmeni** olarak görev yapmaktaydım. **Freelance**olarak başladığım ilk dönemlerde kurumda en çok dikkatimi çeken nokta, şirketin gece koruyuculuğunu yapan köpekleri olmuştu. Aslında herkes yandaki resimde görülen ki kadar etkili ve caydırıcı olmasını bekleyebilir ancak son derece sakin ve kendi halinde sevimli bir köpekti. Bilişim sektöründe sistem eğitimleri de veren bu şirketin, geceleri koruma görevi ile barındırdığı köpeğinin adı ise, şirketin yaptığı işle ilintili olarak **Proxy** olarak verlimişti. Bilenler bilir...Neredeyse gıkı bile*(pardon havı bile)* çıkmayan bu sevimli köpeğin yersiz serzenişte bulunmamasının da bir nedeni vardı elbette. **Proxy**...Gelen veriyi süzüp ona göre aksiyon veriyordu çünkü Şaka bir yana tesadüfe bakın ki bu günkü konumuzda **Proxy** kavramı ile alakalı.

WCF Servislerinin herhangibir istemci uygulama tarafından kullanılmasını sağlamak için tercih ettiğimiz yollardan birisi de **Proxy** tiplerinden faydalanmaktır. Genellikle **Add Service Reference** veya**Svcutil.exe** ya da **SlSvcUtil.exe(Silverlight versiyonu)** gibi araçlar yardımıyla **Proxy** üretimi kolayca gerçekleştirilebilir. **Proxy** tipleri, servislere erişilmesi sırasında istemci tarafında yazılan kodu hafifletmekle kalmaz aynı zamanda çalışma zamanının ayağa kaldırılması gereken ya da iletişim sırasında oluşturulması gereken pek çok nesnenin iş yükünü de üzerine alır. Ancak bazı durumlarda istemci tarafında **Proxy** tipi kullanımı yerine, servis ile olan iletişimde gerekli olan **kanal(Channel)** yapısının manuel kod ile oluşturulması ve diğer hazırlıkların yapılarak iletişim kurulması istenebilir.

Bu istek özellikle servis tarafındaki özel **serileştirilebilir tiplerin(Serializable Types)** çeşitli yardımcı fonksiyonelliklere sahip olduğu durumlarda önem kazanmaktadır. örneğin sunucu tarafındaki bu tiplerde **doğrulama işlemleri(Validation)** için yazılmış bazı özel fonksiyonlar yer alabilir ve bunların istemci tarafında üretilen tiplere de alınması istenebilir. Böylece bu fonksiyonelliklerin içerdiği bazı iş kurallarının istemci tarafında da yüklenilmesi istenebilir ki normal **Proxy** üretiminde bu metodların istemci tarafına taşınmadığı bilinmektedir. İşte bu teoriden yola çıkarak yazdığımız bu blog girdimizde, söz konusu durumun **Silverlight** uygulamalarında nasıl çözümlenebileceğini incelemeye

çalışıyor olacağız. *(Aslında çok eskiden **.Net Remoting** ile uğraşmış birisi olarak, dağıtık uygulama geliştirirken servis tarafında çalışan bileşenlerin, istemci tarafına da referans edildikleri bir yöntem olduğunu ifade edebilirim)*

***Not : Silverlight** tarafında **Proxy** tabanlı olarak **WCF** servislerinin nasıl kullanılabileceğini Screencast - Silverlight Enabled WCF Services görsel dersinden izleyebilirsiniz.*

Dilerseniz vakit kaybetmeden örneğimize başlayalım. **Visual Studio 2010 RC** sürümünde oluşturduğumuz **Silverlight 4.0** uygulamamızın içerisinde
aşağıdaki **IAlbumProducer** isimli**arayüzün(Interface)** olduğunu düşünelim.

```
using System.Runtime.Serialization;
using System.ServiceModel;

namespace WithChannelBased.Web
{
   [ServiceContract]
   public interface IAlbumProducer
   {
      // Silverlight istemcilerin asenkron çağrı yapmalarını zorlamak için aşağıdaki ön işlemci direktifi(Pre Proccesor Directive) eklenmiştir.
#if SILVERLIGHT

      [OperationContract(AsyncPattern=true)]
      IAsyncResult BeginGetAlbum(int albumId, AsyncCallback callback, object state);

      Album EndGetAlbum(IAsyncResult result);

#else

      [OperationContract]
      Album GetAlbum(int albumId);

#endif
   }

   [DataContract]
   public class Album
   {
      [DataMember]
      public int Id { get; set; }
      [DataMember]
      public string Title { get; set; }
      [DataMember]
```

```
        public string Genre { get; set; }
    }
}
```

Dikkat edileceği üzere **Interface** tipi aslında bir **servis sözleşmesi(Service Contract)** tanımlamaktadır. Bu sözleşmenin **Siverlight** istemcilerinde asenkron çağrıları zorunlu kılması içinse bir ön işlemci direktifi kullanılmıştır. Yanlız bu ön işlemci direktifinin büyük harfler ile yazılması önemlidir. Peki neden böyle bir gereksinimimiz olmuştur?

öncelikli olarak sunucu tarafında **BeginGetAlbum** ve **EndGetAlbum** metodlarının, arayüz zorlaması nedeniyle servis sınıfı içerisinde uygulanması istenmemektedir. Diğer yandan aynı dosyayı istemci tarafına alıyor olacağız ki bu durumda **Silverlight** istemcisinin **asnekron** çağrılar için söz konusu **BeginGetAlbum** ve **EndGetAlbum** metodlarını da kullanabilmesi gerekmektedir. İşte bu sebepten bir ön işlemci direktifi kullanılması tercih edilmiştir.

Diğer yandan sözleşmenin bulunduğu dosya içerisinde **Album** isimli serileştirilebilir bir tipin de yer aldığı görülmektedir. **Album** sınıfı bir **veri sözleşmesi(Data Contract)** şeklinde tanımlanmıştır. önemli olan noktalardan birisi servis sözleşmesi ve veri sözleşmesinin aynı fiziki dosya içerisinde tutulmuş olmalarıdır. Nitekim bu dosya **Silverlight** uygulamasına doğrudan link olarak bağlanacaktır. Servis sözleşmesini uygulayacak tipin aslında **Silverlight** destekli bir **WCF Servisi** olduğu düşünüldüğünde projeye **Silverlight-enabled WCF Service** şablonunda **AlbumProducer** isimli bir öğenin eklenmesi **web.config** dosyası içerisinde bazı ön hazırlıkların yapılmasını sağlayacaktır. Elbette eklenen bu tipin yukarıda tanımlı olan servis sözleşmesini uygulaması gerekmektedir. Aşağıdaki kod parçasında görüldüğü gibi;

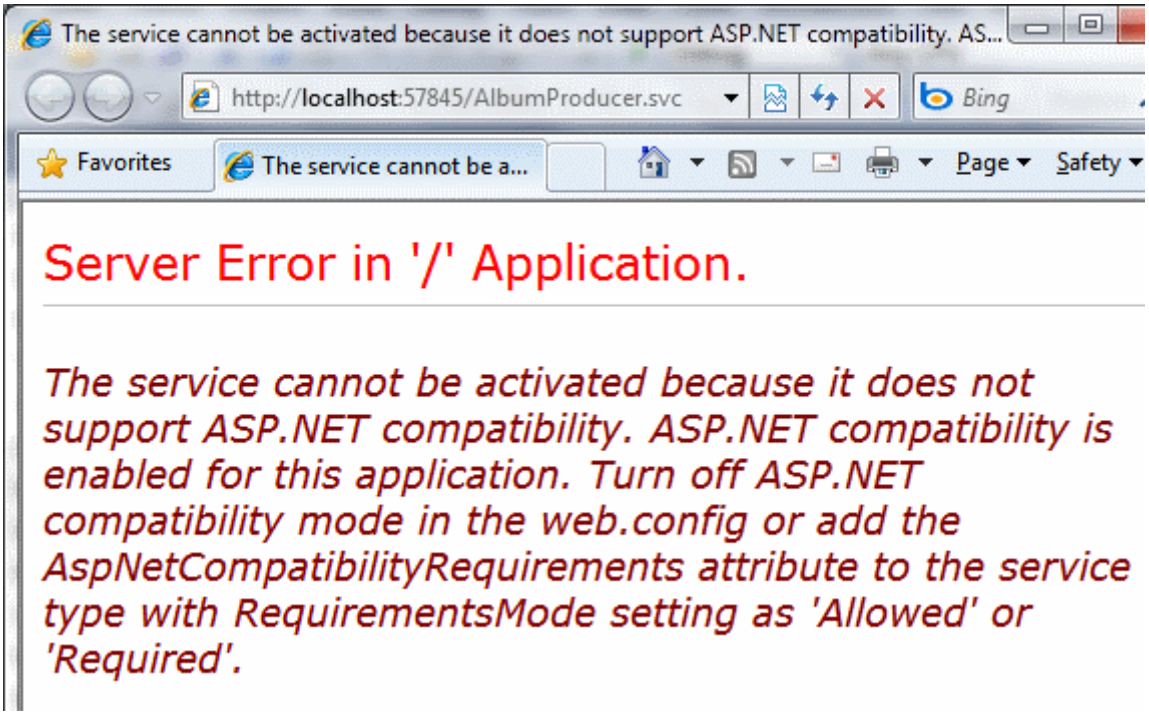
```
using System.Runtime.Serialization;
using System.ServiceModel;

namespace WithChannelBased.Web
{
    public class AlbumProducer
        :IAlbumProducer
    {
        #region IAlbumProducer Members

        public Album GetAlbum(int albumId)
        {
            return new Album { Id = 1000, Title = "Benim Şarkılarım", Genre="Türkçe Pop" };
        }

        #endregion
```

```
    }
}
```

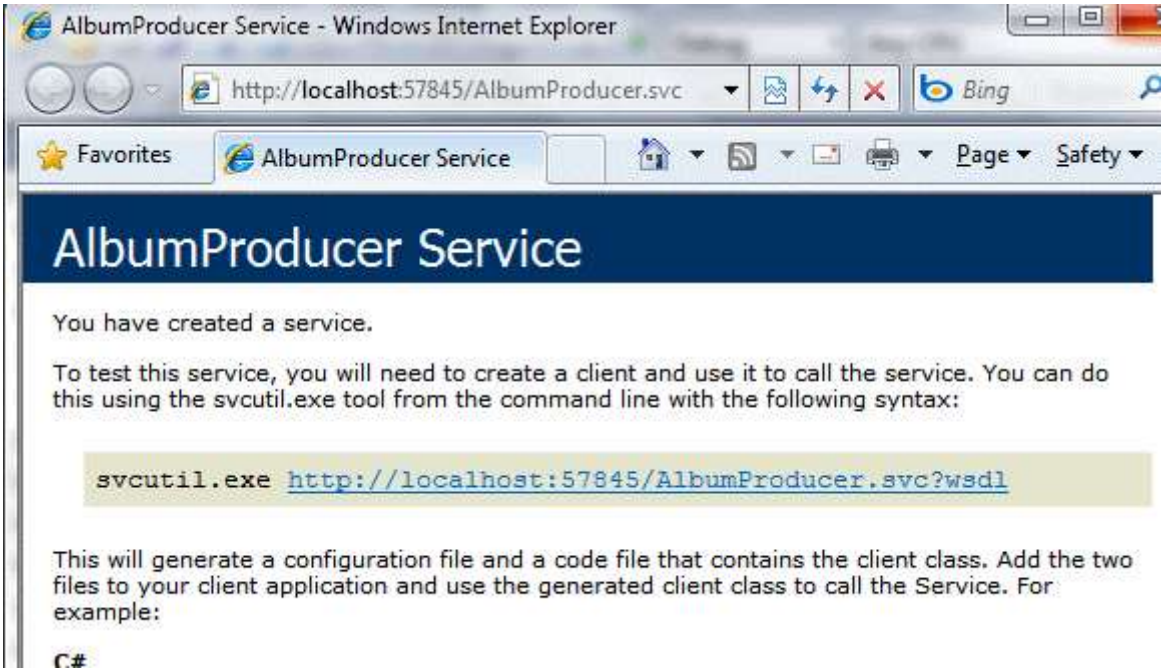
GetAlbum metodunun ne yaptığı çok fazla önemli değildir. Sadece test amacıyla kullanacağımız bir içerik döndürmektedir. Bu işlemlerin ardından servisin bulunduğu sunucu tarafındaki **web.config** dosyasının içeriğinde bazı düzenlemeler yapılması gerekmektedir.

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="false" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <bindings>
      <customBinding>
        <binding name="WithChannelBased.Web.AlbumProducer.customBinding0">
          <binaryMessageEncoding />
          <httpTransport />
        </binding>
      </customBinding>
    </bindings>
    <!--<serviceHostingEnvironment
aspNetCompatibilityEnabled="true" multipleSiteBindingsEnabled="true" />-->
    <services>
      <service name="WithChannelBased.Web.AlbumProducer">
        <endpoint address="" binding="customBinding"
bindingConfiguration="WithChannelBased.Web.AlbumProducer.customBinding0"
          contract="WithChannelBased.Web.IAlbumProducer" />
        <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Aslında **Siverlight-enabled WCF Service**' i **AlbumProcuder.svc** adıyla eklediğimizden **web.config** dosyasında yukarıdaki **XML** içeriği otomatik olarak oluşturulmaktadır. Bir kaç küçük fark ile...İlk olarak **Contract** tipinin aslında **IAlbumProcuder** arayüzü olarak belirtilmesi gerekmektedir. Diğer yandan **Asp.Net Compatibility Enabled** opsiyonunun pasif olması gerekmektedir. Aksi durumda servisi, herhangibir tarayıcı uygulama üzerinden görüntülemek istediğimizde aşağıda yer alan hata ile karşılaşırız.

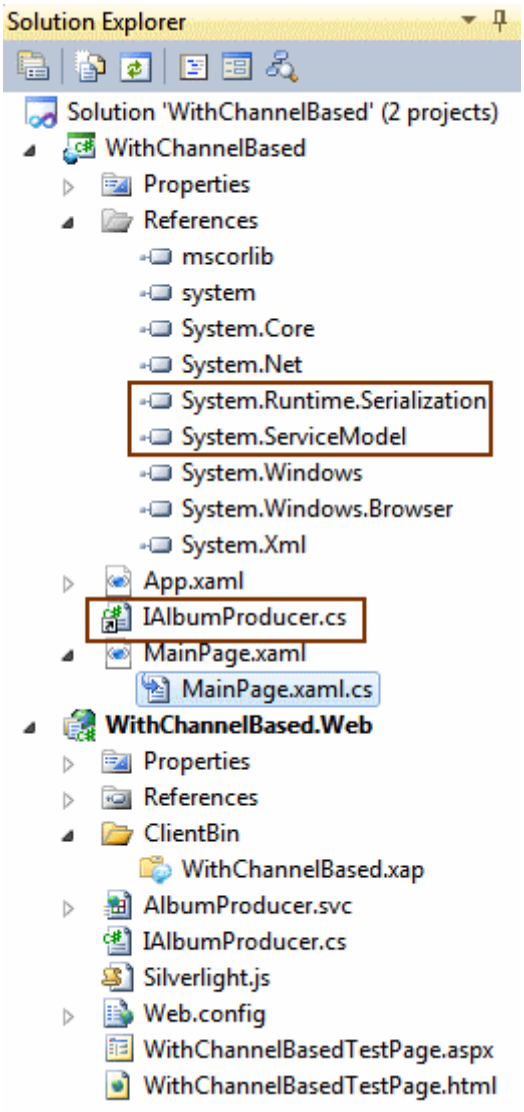


Şu aşamdan sunucu tarafındaki hazırlıklar tamamlanmıştır. Buna göre **AlbumProducer.svc** sayfasının tarayıcı uygulamadan talep edilmesi halinde aşağıdaki görüntü ile karşılaşılması işlerin iyi gittiğinin habercisidir.



Hemen bir hatırlatmada bulunalım. Servisi çalıştıran **Asp.Net Development Server** uygulamasının açtığı **port** numarası önemlidir. Nitekim istemci tarafında yazılacak olan kod içerisinde bu port numarası değerlendirilecektir 🙂

İşin belkide kodlama açısından en sıkıcı noktası ise istemci tarafını geliştirmektir. Her şeyden önce istemci tarafının, sunucu tarafında yer alan **IAlbumProducer.cs** dosyasını referans etmesi gerekmektedir ki bunu ilgili dosyayı ilave ederken **Add As Link** seçeneğinin kullanılmasında yarar vardır. Böylece dosyanın tek bir noktada durması garanti edilmiş olur. Diğer yandan **System.ServiceModel.dll***(Servis çalışma zamanının tesisi için gerekli tipleri kullanabilmek için)* ve **System.Runtime.Serialization.dll***(Veri sözleşme nitelikleri için)* **Assembly**' larının **Silverlight** uygulamasının olduğu projeye referans edilmesi şarttır.



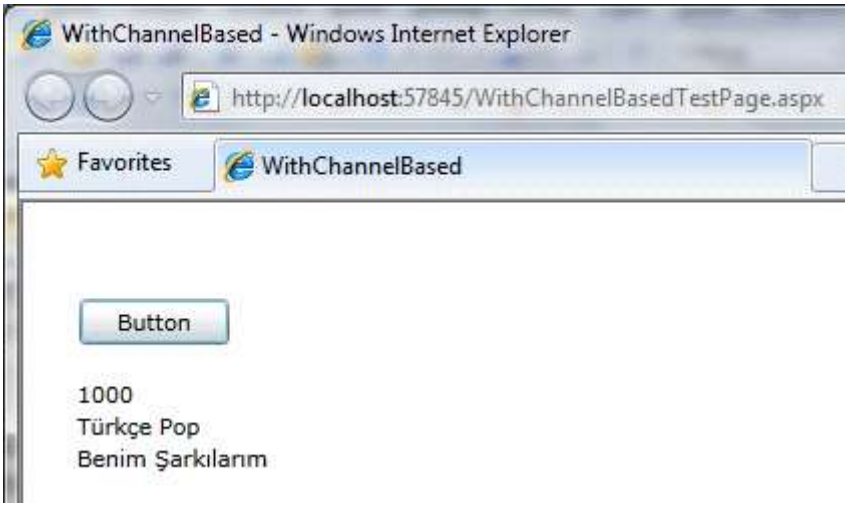
MainPage.xaml içeriğimizde basit olarak bir **Button** bileşenine basıldığında **TextBlock** kontrolünün içeriğinin sunucundan gelen **Album** bilgisi ile doldurulması planlanmaktadır. Buna göre kod içeriğini aşağıdaki gibi oluşturmamız yeterli olacaktır.

```
using System;
using System.Collections.Generic;
using System.ServiceModel;
using System.ServiceModel.Channels;
using System.Threading;
using System.Windows;
using System.Windows.Controls;
using WithChannelBased.Web;

namespace WithChannelBased
{
   public partial class MainPage : UserControl
```

```
{
    // User Interface için ayrı bir Thread' in değerlendirileceği nesne tanımlanır
    SynchronizationContext syncContext;

    public MainPage()
    {
        InitializeComponent();
    }

    private void GetAlbumButton_Click(object sender, RoutedEventArgs e)
    {
        // BindingElement listesi tanımlanır
        List<BindingElement> bindings = new List<BindingElement>();
        // Sunucu tarafındaki web.config dosyasından hatırlanacağı üzere
BinaryMessageEncoding tipinden bir Binding tipi mevcuttur. öncelikle bu bağlayıcı
listesine eklenir.
        bindings.Add(new BinaryMessageEncodingBindingElement());
        // Yine sunucu tarafındaki Binding listesine bakıldığında HttpTransport tipinden bir
bağlayıcının da olduğu görülmektedir. Dolayısıyla bu tipten bir nesne örneğide oluşturulur.
        bindings.Add(new HttpTransportBindingElement());

        // Aynen Web.config dosyasında olduğu gibi, yukarıda tanımlanan Binding nesne
örnekleri bir CustomBinding nesne örneği içerisinde toplanır. Bu nedenle parametre olarak
bindings isimli liste verilmiştir.
        CustomBinding cBinding = new CustomBinding(bindings);

        // WCF çalışma zamanının bir kanal oluşturması için gerekli fabrika tipi tanımlanır.
        // İlk parametre kanalın kullanacağı bağlayıcı listesidir. İkinci parametre ise
EndPoint için gerekli adres bilgisini içermektedir. (Port numarasını saklayın demiştim :) )
        var channelFactory = new ChannelFactory<IAlbumProducer>(
            cBinding,
            new EndpointAddress("http://localhost:57845/AlbumProducer.svc")
            );

        syncContext = SynchronizationContext.Current;

        // Kanal oluşturulu ve dolayısla açılır
        IAlbumProducer cnl = channelFactory.CreateChannel();

        // Servis tarafındaki GetAlbum metodu asenkron olarak çağırılır.
        cnl.BeginGetAlbum(
            1102
            , iar => {
                Album albm=((IAlbumProducer)iar.AsyncState).EndGetAlbum(iar);
                syncContext.Post(
```

```
                obj =>
                {
                    AlbumInfoTextBlock.Text = String.Format("{0}\n{1}\n{2}", albm.Id,
albm.Genre, albm.Title);
                }
                , albm);
            }
            , cnl
            );
        }
    }
}
```

Volaaa!!! Uygulamayı test ettiğimizde aşağıdaki çalışma zamanı çıktısını almamız beklenmektedir.



Dikkat edileceği üzere istemci tarafında bir konfigurasyon dosyası içeriği hazırlanmamıştır. Bir başka deyişe **WCF çalışma Zamanı(WCF Runtime)** için gerekli **Endpoint, CustomBinding** bildirimlerinin tamamı kod içerisinde gerçekleştirilmiştir. özet olarak **Silverlight** istemcisinin **Proxy** tipine ihtiyaç duymadan çalışması sağlanabilmiştir. Böylece geldik bir görsel dersimizin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WithChannelBased_RTM.rar (72,10 kb) **[örnek Visual Studio 2010 RC sürümü üzerinde geliştirilmiş ve RTM sürümü üzerinden de test edilmiştir. Son sürümle birlikte test etmenizde yarar bulunmaktadır.]**

Workflow Foundation Öğreniyorum - Ders 6 - Expression Activities (2010-06-01T10:35:00)

workflow foundation,workflow foundation 4.0,wf,wf 4.0,workflow foundation öğreniyorum,

Merhaba Arkadaşlar,

NedirTv?com sponsorluğunda hazırladığımız **"Workflow Foundation 4.0 öğreniyorum"** görsel eğitim serimizin **7nci** dersi ile birlikteyiz.

Bu dersimizde **System.Activities.Expression** isim alanı(Namespace) altında yer alan ve her biri **Activity** bileşeni olan tiplerden bir kaçını kavramaya çalışıyoruz. özellikle üzerinde durduğumuz nokta ise, Workflow tasarım ortamına ait **Component** sekmesinde görünmeyen bu bileşenleri kullanabilmek için,**XAML(eXtensible Application Markup Language)** tarafında geliştirme yapmak. Haydi bakalım parmakları sıvayalım.

Ders 6 - Expression Activities

Süre : 13:41

Dosya Boyutu : 17.7 Mb

örnek : Lesson6.rar (35,50 kb)

Mp4 Formatında İndirmek İçin Tıklayın

LINQ Sorgusu mu? ForEach mi? Bir Türlü Karar Veremedim (2010-05-28T11:40:00)

c#,linq,foreach,linq vs foreach,


Merhaba Arkadaşlar,

Bilim Kurgu fanatiklerinin kafasında her zaman hayranı oldukları filmlerden kesitler, sahneler kalır. Matrix filmini izleyenler eminimki Neo' ya uzatılan kırmızı ve mavi hap serenatını gayet iyi hatırlayacaktır. Morpheus haplardan birisinde Alice Harikalar Diyarının kapılarını ardına kadar açabileceğini ifade ederken, diğer hapı yuttuğunda, Neo' nun yatağında hiç bir şey olmamış gibi uyanacağını ve tüm bunların bir hayalden ibaret olduğunu düşüneceğini belirtir. Tabi Neo amacına ulaşmak için zaten hangi hapı içmesi gerektiğini biliyordur ki son bölümde aslında gerçekten hapı yutmaktadır 😆

Bizde yazılımcılar olarak bazen karar verirken tabir yerinde ise sürüncemede kalabiliriz. Böyle durumlarda ufak tefek gözüken noktaların aslında çok büyük riskler taşıdığını da düşünmemiz gerekmektedir. çünkü karar vermek için basit bir kaç test kodu çok işimize yarayacaktır. İşte bu yazımızda böyle bir konuya değiniyor olacağız.

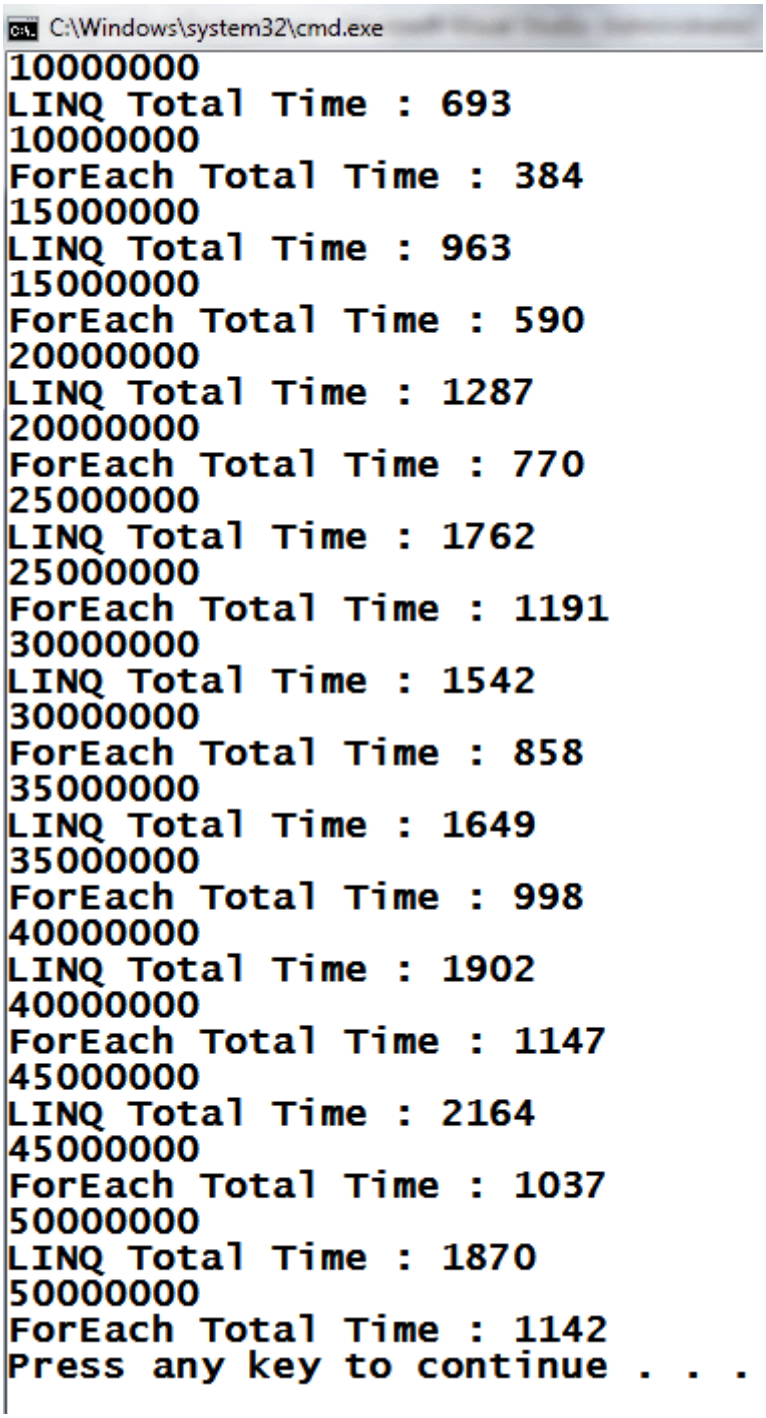
Aslında konunun çıkış noktası Microsoft Teknoloji Günleri Akşam Sınıfındaki bir meslektaşımın sorusu oldu. Değerli meslektaşım uygulama kodunda koleksiyon bazlı sorgulamaları gerçekleştirirken pek çok vakada foreach döngülerini tercih ettiğini söyledi. Tabi her durumda değil. Bende bu noktada aynı amaca hizmet eden bir **LINQ** sorgusu ile **ForEach** çalışması arasındaki performans farklılıklarını irdelemeye karar verdim. Nitekim performans her zaman için karar vermeden önem arz eden kriterlerden birisidir. Anlayacağınız basit bir test ve sonuçlarını irdeliyor olacağız bu kısa yazımızda.

örnek uygulamamızda **Enumerable.Range** metodu yardımıyla elde edilen bir **int** sayı dizisi içerisinde **2** ile tam bölünebilen sayıların adedini hesap ettirmekteyiz. Tahmin edeceğiniz üzere bu tip bir işlemi **LINQ** sorgusu yardımıyla anlamlı bir kod ifadesi ile yerine getirebiliriz. Ayrıca bunu bir **foreach** döngüsü ile de gerçekleştirebiliriz. İşte test kodlarımız.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Diagnostics;

namespace LINQForEachPerformance
{
   class Program
   {
      static void Main(string[] args)
      {
         for (int i = 1; i < 10; i++)
         {
            IEnumerable<int> range = Enumerable.Range(i,(i+1)*10000000);
            WithLinq(range);
            WithForeach(range);
         }
```

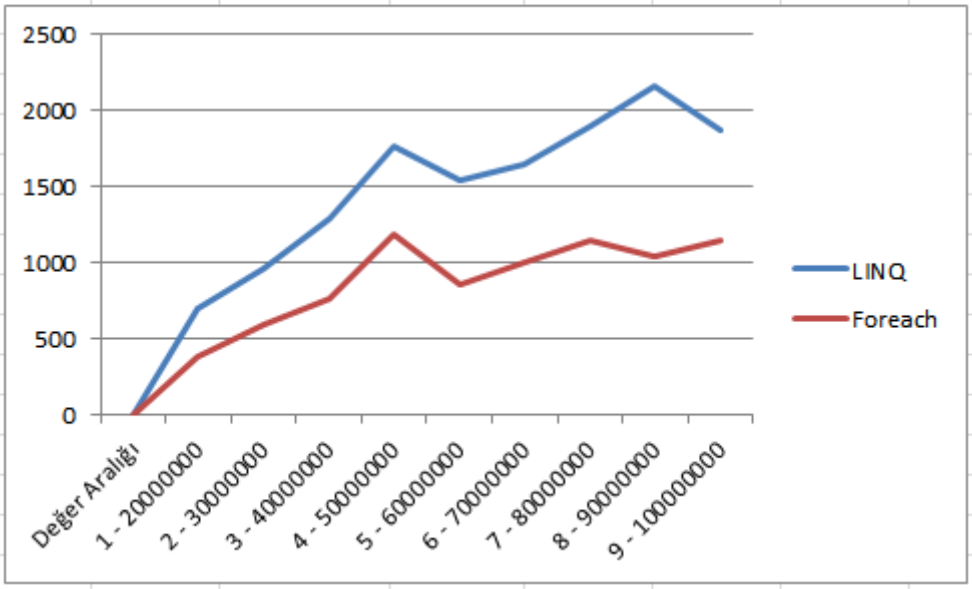
```
        }

        static void WithLinq(IEnumerable<int> range)
        {
            Stopwatch sWatch = new Stopwatch();
            sWatch.Start();

            int count = (from i in range
                         where i % 2 == 0
                         select i).Count<int>();
            Console.WriteLine(count.ToString());

            sWatch.Stop();
            Console.WriteLine("LINQ Total Time :
{0}",sWatch.ElapsedMilliseconds.ToString());
        }

        static void WithForeach(IEnumerable<int> range)
        {
            Stopwatch sWatch = new Stopwatch();
            sWatch.Start();

            int count = 0;
            foreach (int i in range)
            {
                if (i % 2 == 0)
                    count++;
            }
            Console.WriteLine("{0}",count.ToString());

            sWatch.Stop();
            Console.WriteLine("ForEach Total Time : {0}",
sWatch.ElapsedMilliseconds.ToString());
        }
    }
}
```

örnekte **arka arkaya 10 deneme** yapılmaktadır. **WithLinq** metodu **LINQ** sorgusunu kullanarak ikiye tam bölünen sayıların adedini vermektedir. **WithForeach** metodu ise aynı işlemi **foreach**döngüsü yardımıyla gerçekleştirmektedir. **Stopwatch** tipi yardımıyla her hesaplamanın toplam süresi bulunmaktadır. Uygulamanın **Intel çift çekirdek işlemcili, 4Gb Ram'** i olan makinemdeki çalışma zamanı sonuçlarından bir tanesi aşağıdaki ekran görütüsündeki gibidir.



Aslında her zaman için süreler farklı olacaktır ancak grafiksel eğriler benzer olacaktır. Durumun daha net bir şekilde görülmesi için değerlerin **Excel** üzerinde **Chart** olarak gösterilmesi yeterlidir. İşte sonuçlar.

Değer Aralığı	LINQ	ForEach
1 - 20000000	693	384
2 - 30000000	963	590
3 - 40000000	1287	770
4 - 50000000	1762	1191
5 - 60000000	1542	858
6 - 70000000	1649	998
7 - 80000000	1902	1147
8 - 90000000	2164	1037
9 - 100000000	1870	1142



Görüldüğü üzere **foreach** döngüsü ile yapılan hesaplamalar değer aralığı büyüse dahi **LINQ** sorgusuna göre daha kısa sürede icra edilmektedir. Buna göre **foreach'** in daha hızlı olduğunu söyleyebilir miyiz? Bu senaryo için evet. Ama bildiğiniz üzere **LINQ** daha karmaşık sorgular yazılması noktasında elbetteki iç içe geçecek sayısız **foreach** kullanımından çok daha etkili bir yöntemdir. Ancak başta da belirttiğimiz gibi insan bir an için hangi hapı yutacağına karar veremiyor. Tabi farklı sorgulama senaryoları ile farklı denemeler yaparak karşılaştırmalara devam etmekte yarar olabilir. Bu kutsal görevi de siz değerli okurlarıma bırakıyorum. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

LINQForEachPerformance.rar (23,30 kb) **[örnek Visual Studio 2010 Ultimate ile geliştirilmiş ve test edilmiştir]**

Microsoft Teknoloji Günleri Akşam Sınıfı Başladı (2010-05-25T23:00:00)

microsoft teknoloji günleri,microsoft,bizspark,gelişim atölyesi,c# 4.0,dynamic,optional parameters,named parameters,com interop features,covariance generic,contravariance generic,visual studio 2010,

Merhaba Arkadaşlar,

Bildiğiniz üzere bir süre önce **Microsoft Türkiye** ile birlikte **Teknoloji Günleri Akşam Sınıfınını** duyurmuştuk. **Workshop** tadında olan eğtimlerde ki amacımız, **.Net Framework 4.0** tarafında gelen yenilikleri basit bir şekilde aktarmak ve tanıtmak. Bu gün başlayan servüvenimizin ilk dersinde, **C# 4.0 ile Birlikte Gelen Yeniliklere** değindik. 4 basit örnekle tanıdığımız yeni özelliklerde ilk olarak **Reflection** yerine **dynamic** kullanımını ve **IronPython** ile yazılmış bir kod içeriğinin çağırılmasını inceledik. Bu sayede **dynamic diller** ile olan etkileşimi de irdelemiş olduk. Ardından **COM Interop** tarafında gelen yenilikleri ve **Optional, Named Parameters** ve **Ommiting Ref** gibi konuları **Office** etkileşimi olan bir örnek üzerinden değerlendirdik. Son olarak **Generic Covariance** ve **Contravariance** konusuna giriş yaparak basit bir örnekle konuyu pekiştirmeye çalıştık.

Hafta içi gerçekleştirdiğimiz bu etkinliğe kayıt yaptırıp, özellikle iş yoğunluğu arasında zaman ayıran tüm katılımcılara canı gönülden teşekkür etmek istiyorum. Bir sonraki eğitimimiz **22 Haziran 2010 Salı** günü yapılacak. Bu eğitimde ise **.Net 4.0 ile Paralel Programlama** konusuna giriş yapıyor olacağız. Eğlenceli ve ödüllü bir eğitim olacağını şimdiden belirtmek isterim.

Bu günkü etkinliğimize ait sunum dosyası ve örneklere aşağıdaki linklerden ulaşabilirsiniz.

C# 4.0 - New Features.pptx (474,28 kb)

Microsoft Yaz Okulu Gun 1.rar (1,18 mb)

Workflow Foundation Öğreniyorum - Ders 5 - Argument Kavramı ile Tanışalım (2010-05-25T14:25:00)

workflow foundation 4.0,workflow foundation,visual studio 2010,argument,inargument,outargument,variable,

Merhaba Arkadaşlar,

NedirTv?com sponsorluğunda hazırladığımız "Workflow Foundation 4.0 öğreniyorum" görsel eğitim serimizin **6ncı** dersi ile birlikteyiz. Bu dersimizde**Workflow** örneklerinde içeriye ve dışarıya doğru veri akışlarında kullanılan **Argument** kavramını tanımaya çalışıyoruz. Bu anlamda**Argument** ile **Variable** arasındaki temel farklılıklara da değiniyoruz.

Geliştireceğimiz örnekte, **Argument** tipi olarak kendi tasarldığımız bir sınıfı işin içerisine katarak olaya farklı bir boyut kazandırıyor ve bu sayede **geliştirici tanımlı bir tipin**, bir **Workflow** örneği içerisinde nasıl ele alınabileceğini de görmüş oluyoruz. örneğimizde **Direction** değeri **In**ve **Out** tipinden olan **Argument**' ları ele alıyoruz. Ayrıca, **Workflow** örneklerini çalıştırırken **In** tipinden olan **Argument**' lara nasıl değer atayabileceğimizi, ek olarak, **Workflow**' un yürütülmesi sonucu çalışma zamanı ortamına veri taşıyan **Argument**' ları nasıl okuyabileceğimizi öğreniyoruz.

İyi seyirler dilerim.

Ders 5 - Argument Kavramı ile Tanışalım

Süre : 12:33

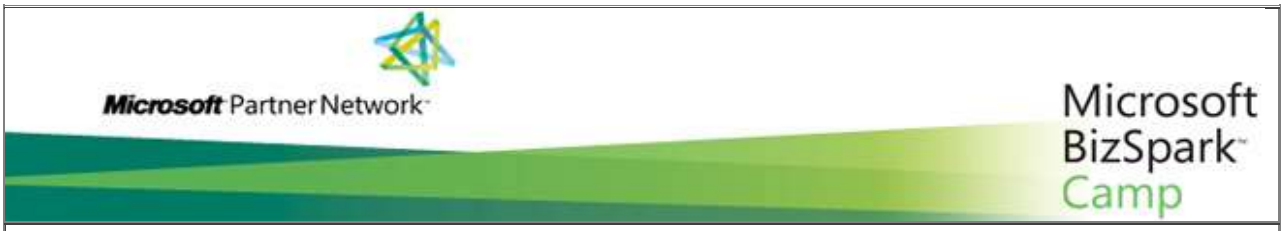
Dosya Boyutu : 14.1 Mb

örnek : Lesson5.rar (47,56 kb)

Mp4 Formatında İndirmek İçin Tıklayın

[Microsoft Teknoloji Günleri Akşam Sınıfında Buluşalım (2010-05-22T15:40:00)]

microsoft teknoloji günleri,microsoft,bizspark,gelişim atölyesi,c# 4.0,parallel programming,wcf,wcf eco system,windows server appfabric,wf 4.0,asp.net 4.0,visual basic 2010,wpf 4.0,



Tarih
25 Mayıs 2010 Salı

Saat
19.00 - 21.30

Yer
Microsoft İstanbul Ofisi

Eğitmen:
Burak Selim Şenyurt
Microsoft MVP

Microsoft Teknoloji Günleri -
Akşam Sınıfı

Her ay düzenli olarak gerçekleştireceğimiz ve bir seri olarak birbirini takip edecek sınıf etkinliklerimizle 9 ay boyunca siz yazılım geliştiren ve tasarım yapan iş ortaklarımızla birlikte olacağız.

Aşağıda detaylarını paylaştığımız ve sizler için hayli faydalı olacağına inandığımız Microsoft Teknoloji Günleri Akşam Sınıfı Etkinliğimize kaydınızı hemen yaptırabilirsiniz.

Ders İçerikleri:
1. **25.Mayıs.2010/Salı: C# 4.0 ile Gelen Yenilikler : (2 Saat)** Bu eğitimde C# programlama dilinin 4.0 versiyonu ile birlikte gelen yenilikleri tanıtmaya çalışılmakta ve özellikle, Dynamic diller ile Office API vb yapılar ile olan etkileşim üzerine örnekler geliştirilmektedir. Eğitimde temel olarak dynamic keyword, optional and named parameters, PIA, Co-Contra Variance Generics konularına değinilmektedir.

2. **22.Haziran.2010/Salı: .Net 4.0 ile Paralel Programlama : (2 Saat)** Uzun süredir ev bilgisayarlarımız dahil en az iki çekirdekli sistemler üzerinde çalışabiliyoruz. Son zamanlarda 8 çekirdeğe kadar çıkabilen yeni nesil işlemciler üzerinde paralel programlama yeteneklerini sonuna kadar kullanıp daha performanslı, hızlı, verimli ve ölçeklenebilir uygulamalar geliştirmeye ne dersiniz? İşte tam size

göre bir eğitim. Bu eğitim ile .Net 4.0 kütüphanelerini kullanarak kolay bir şekilde nasıl paralel programlama yapabileceğinizi göreceksiniz.

3. **20.Temmuz.2010/Salı: WCF ile Servis Yaklaşımı : (2 Saat)** .Net 3.0 ile birlikte duyurulan ve tek bir servis geliştirme metodolojisi sunarak daha önceki dağıtık mimari geliştirme tiplerini (Xml Web Services, .Net Remoting, MSMQ vb...) bir çatı altında birleştiren Windows Communication Foundation konulu eğitimdir. Eğitimde özellikle .Net 4.0 ile birlikte gelen yeniliklere de değinilmekte olup asıl amaç WCF ile uygulama geliştirme şekillerinden bir kaçını göstermek ve tanıtmaktır.

4. **20.Ağustos.2010/Cuma: WCF Eco System : (3 Saat)**WCF alt yapısı üzerine kurulu olan WCF Eco System içerisinde Data Services, Workflow Services, RIA Services, WebHttp Services ve Core Services tipleri yer almaktadır. Bu eğitimde Data Services, Workflow Services, RIA Services ve WebHttp Services konulu örnekler geliştirilmekte olup, söz konusu alt yapı ile değerlendirilebilecek hazır servis modelleri irdelenmektedir.

5. **20.Eylül.2010/Pazartesi: Windows Server AppFabric : (2 Saat)** Bu eğitimde WCF ve Workflow servis örneklerinin izlenmesi, sorunların teşhis edilmesi, örneklerin yaşam döngülerinin takibi, konfigurasyon ayarlarının belirlenmesi ve pek çok yönetimsel konuda geliştiriciler ile IT yöneticilerinin daha kolay anlaşabilmelerini de sağlayan Dublin kod adlı Windows Server AppFabric ürün ailesi incelenmektedir. özellikle IIS üzerine gelen eklentiler ile söz konusu yönetsel işlemlerin nasıl yapılabildiği derinlemesine incelenmektedir.

6. **20.Ekim.2010/çarşamba: Worfklow Foundation 4.0 : (3 Saat)** WF 4.0 beraberinde pek çok köklü yenilik ile gelmektedir. Geliştirilen Base Activity Library, paralel programlama desteği, veri akışı için gelen Argument, Variable gibi kavramlar ve daha pek çoğunun ele alındığı eğitimde basit örnekler ile WF modelin tanıtılmaya çalışılmaktadır.

7. **23.Kasım.2010/Salı: Asp.Net 4.0 : (3 Saat)** Web Programlama' nın .Net 4.0 ile birlikte gelen yeni yüzünü görmeye hazır mısınız? Pek çok yeni özellik ile birlikte gelen Asp.Net 4.0' ın anlatıldığı bu eğitimde, temelden orta seviyeye kadar basit bir web uygulaması tasarlanmakta ve konunun daha iyi kavranabilmesi amaçlanmaktadır.

8. **20.Aralık.2010/Pazartesi: Visual Basic 2010 : (2 Saat)**Bu eğitimde Visual Basic 2010 programlama dili ile birlikte gelen pek çok yeni özellik üzerinde durulmakta ve geliştirilen örnekler ile bu kavramlar pekiştirilmeye çalışılmaktadır. Bu noktada AutoImplemented Properties, Collection Initializers, Implicit Line Continuation, Mutlipe Lambda Expressions, Dynamic keyword, Type Equivalance Support gibi konular üzerinde durulmaktadır.

9. **20.Ocak.2011/Perşembe: WPF 4.0 ile Windows Programlama : (3 Saat)** .Net 3.0 ile birlikte duyurulan Windows Presentation Foundation modeli ile zengin kullanıcı deneyimine sahip windows uygulamaları tasarlanabilmektedir. özellikle Windows 7 üzerinde en iyi kullanıcı deneyimini sunan WPF 4.0 ile birlikte gelen yenilikleri öğrenmeye ne dersiniz?

Workflow Foundation Öğreniyorum - Ders 4 - Flowchart için Ek İşlemler (2010-05-18T14:05:00)

workflow foundation 4.0,workflow foundation,visual studio 2010,

Merhaba Arkadaşlar,

"Workflow Foundation 4.0 öğreniyorum" görsel eğitim serimizin **5nci dersi ile(4+1)** karşınızdayız. Daha önceki dersimizde **Flowchart** aktivite bileşenini tanımaya başlamıştık. Bu dersimizde ise **Flowchart** aktivitesi içerisinde **Flowswitch** ve **Parallel** bileşenlerinin nasıl kullanılabileceğini incelemeye çalışıyor olacağız.

Flowswitch aktivite bileşeni **C#** tarafındaki **switch...case** ifadelerinin bir benzeri olarak düşünülebilir. **Flowswitch** bileşeni **generic** türlendirilmiş bir tiptir. Genellikle bir değişkenin veya bir **Visual Basic ifadesinin(Expression)** çalıştırılması sonucu elde edilen sonucun farklı değerlerine göre, akışın farklı yerlere dallanması gerektiği durumlarda kullanılır.

Parallel aktivite bileşeni ise, birbirleri ile ilişkisi olmayan, dolayısıyla belirli bir sırada çalıştırılması gerekmeyen bileşenlerin paralel olarak yürütülmesi amacıyla ele alınmaktadır. İyi seyirler dilerim.

Ders 4 - Flowchart için Ek İşlemler

Süre : 15:05

Dosya Boyutu : 16.7 Mb

örnek : Lesson4.rar (38,14 kb)

Mp4 Formatında İndirmek İçin Tıklayın

Int32 ve Int64 Haricindekiler için Parallel.ForEach (2010-05-17T09:46:00)

parallel programming,c#,c# 3.0,c# 4.0,

 Merhaba Arkadaşlar,

Bir kaç yıl öncesine kadar **Bizitek** firmasına **Junior Developer** olarak görev almaktaydım. Bu şirkette çalıştığım süre boyunca pek çok projede görev alma fırsatım oldu. Ancak genellikle şirketin iş akışları üzerine geliştirdiği bir ürünün kurulması ve ihtiyaçlara göre düzenlenmesi ile ilgilenmekteydim. Söz konusu uygulamanın belki de en önemli özelliklerinden birisi, kurulduğu firmanın organizasyon ağacını içermesi ve buna göre akış içi adımların kolayca tesis edilebilmesiydi. İşte zaten bazı sıkıntılar da burada başlıyordu. Nitekim bazı firmaların organizasyonel yapıları düzgün değildi. En sık rastlanan vakalardan birisi, herhangibir çalışanın aslında birden fazla görev üstlenmesi nedeniyle organizasyon ağacında birden fazla yerde var olabilmesiydi.

Dolayısıyla bazı durumlarda en tepeden aşağıya doğru inen ve **bağlı liste(Linked List)** benzeri bir oluşumun sağlanması zorlaşmaktaydı. Her neyse...Eminim bu sorunlar çoktan aşılmıştır. Ancak bir önceki cümlede yer alan bağlı liste tarzı yapıların başında dolaşan bir kara bulut daha mevcuttur. Sorunun kaynağında paralel programlama amacıyla **.Net** ortamına kazandırılan **Parallel.ForEach** döngüsü yer almaktadır. Dilerseniz öncelikle sorunu masaya yatıralım. Bu amaçla aşağıdaki kod içeriğine sahip **Employee** isimli basit bir sınıfımız olduğunu düşünelim.

```
class Employee
   {
      public string Profession { get; set; }
      public string Name { get; set; }
      public Employee Parent { get; set; }
      public Employee Child { get; set; }
   }
```

Employee sınıfı ile bir şirketin belirli organizasyonel pozisyonlarını ifade etmek istediğimizi düşünebiliriz. İçeriğinde yer alan **Parent** ve **Child** isimli özellikler dikkat edileceği üzere**Employee** tipindendir. Buna göre bir **Employee** nesne örneğinin altına ve üstüne başka bir **Employee** referansının atanması mümkündür. Dolayısıyla bir ağaç yapısının kolayca oluşturulması mümkündür. Elbetteki sembolik olarak. Söz gelimi;

Director
--->Project Manager
------>Technical Project Manager

--------->Senior Developer
---------->Junior Developer

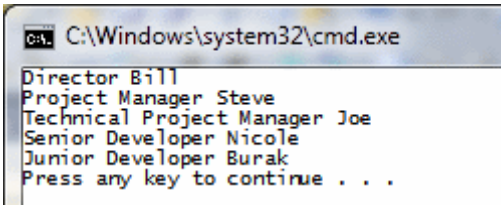
gibi.

Yukarıdaki gibi bir yapıyı oluşturduğumuzda **Parent** ve **Child** özellikleri sayesinde organizasyon içerisinde aşağı ve yukarı doğru kolayca hareket edebileceğimizi görebiliriz. Bu durum aşağıdaki **Console** uygulamasında ele alınmaktadır.

```
using System;

namespace ParallelForNonIntegralRanges
{
   class Program
   {
      static void Main(string[] args)
      {
         Employee root = Fill();

         #region Case 1 - Seri for döngüsü

         for (Employee emp = root; emp !=null ; emp=emp.Child)
         {
            Console.WriteLine("{0} {1}",emp.Profession,emp.Name);
         }

         #endregion
      }

      static Employee Fill()
      {
         Employee d = new Employee { Profession = "Director",Name="Bill" };
         Employee pm = new Employee { Profession = "Project Manager",Name="Steve" };
         Employee tpm = new Employee { Profession = "Technical Project
Manager",Name="Joe" };
         Employee sd = new Employee { Profession = "Senior Developer",Name="Nicole"
};
         Employee jd = new Employee { Profession = "Junior Developer",Name="Burak" };

         d.Parent = null;
         d.Child = pm;

         pm.Parent = d;
         pm.Child = tpm;
```

```
            tpm.Parent = pm;
            tpm.Child = sd;

            sd.Parent = tpm;
            sd.Child = jd;

            jd.Parent = sd;
            jd.Child = null;

            return d;
        }
    }

    class Employee
    {
        public string Profession { get; set; }
        public string Name { get; set; }
        public Employee Parent { get; set; }
        public Employee Child { get; set; }
    }
}
```

Dikkat edileceği üzere **for** döngüsünden yararlanılarak **Director**' den en alt kademede yer alan **Junior Developer**' a kadar ilerlenilmesi sağlanılmaktadır. İşte bu örnek kod parçasının çalışma zamanı çıktısı;



Peki ya **Employee** gibi bir tipin çalışma zamanındaki örneği ve içeriğindeki bağlı referanslar arasında **Parallel.For** döngüsü ile ilerlenilmek istenirse? Nitekim elimizin alında binlerce ve hatta daha fazla elemandan oluşan bir ağaç yapısı olabilir ve bu yapı üzerindeki elemanlarda bazı işlemlerin yapılması istenebilir. Bu durumda işlemlerin daha hızlı gerçekleştirilebilmesi için paralel programlama yetenekleri göz önüne alınabilir. Ancak ortada önemli bir sorun vardır. **Parallel.For** metodunun versiyonlarına bakıldığında **int(Int32)** ve **long(Int64)** tipleri ile çalıştığı görülecektir. Bu durumda **Employee** nesne örnekleri için **Parallel.For** döngüsünü kullanmamız mümkün değildir. O zaman belki **Parallel.ForEach** döngüsü tercih edilebilir. Edilebilir mi acaba? Bunu denediğimizde derleme zamanında aşağıdaki sonuçlar ile karşılaşmamız kaçınılmazdır.

```
#region Case 2 - Parallel ForEach

Parallel.ForEach<Employee>(root,
  emp => emp => Console.WriteLine("{0} {1}"
    , emp.Profession, emp.Name));
```

Error List

3 Errors | 0 Warnings | 0 Messages

	Description
1	The best overloaded method match for 'System.Threading.Tasks.Parallel.ForEach<ParallelForNonIntegralRanges.Employee> (System.Collections.Concurrent.OrderablePartitioner<ParallelForNonIntegralRanges.Employee>, System.Action<ParallelForNonIntegralRanges.Employee,System.Threading.Tasks.ParallelLoopState,long>)' has some invalid arguments
2	Argument 1: cannot convert from 'ParallelForNonIntegralRanges.Employee' to 'System.Collections.Concurrent.OrderablePartitioner<ParallelForNonIntegralRanges.Employee>'
3	Argument 2: cannot convert from 'lambda expression' to 'System.Action<ParallelForNonIntegralRanges.Employee,System.Threading.Tasks.ParallelLoopState,long>'

Bu son derece doğaldır. Nitekim **Employee** tipinin **IEnumerable<T>** gibi bir arayüz implemantasyonu yapmadığı ortadadır. Kaldı ki **foreach** döngüleri üzerinde hareket edecekleri tipler için bu arayüz implemantasyonunun yapılmasını beklemektedir. O halde farklı bir yardımcı metoddan yararlanalılabilir. Aşağıdaki gibi;

```
static IEnumerable<Employee> Iterate(Employee root)
{
  for (Employee emp = root; emp != null; emp = emp.Child)
  {
    yield return emp;
  }
}
```

Yaşananlardan : *çok eskiden **C#Nedir?** adına düzenlenen bir etkinlikte **C# 2.0** ile birlikte gelen yenilikleri anlatmaktaydım. **C# 2.0** ile birlikte gelen önemli yeniliklerden birisi de **yield** anahtar kelimesi ile bilinen iterasyon deseninin çok daha kolay uygulanabilmesiydi.Hatta bu konu ile ilişkili olaraktan amatör olduğum dönemlere ait bir yazım dahi bulunmaktadır. **C# 2.0 İçin İterasyon Yenilikleri(5.7.2005)** Şimdi o yeniliğin buradaki örnekte işimizi ne kadar kolaylaştırdığını bir kere daha görebiliyoruz. Hey gidi günler.* 😀

Iterate isimli metod **yield return** kullanarak **IEnumerable<Employee>** tipinden bir sonuç kümesi döndürmektedir ve bu aslında **Parallel.ForEach** döngüsünün tamda istediği referanstır. Dolayısıyla artık aşağıdaki gibi bir kod parçası derlenip çalıştırılabilir.

```
Parallel.ForEach<Employee>(
  Iterate(root),
  emp => Console.WriteLine("{0} {1}", emp.Profession, emp.Name)
);
```

Dikkat edileceği üzere ilk parametre ile **ForEach** döngüsünün beklediği **IEnumerable<Employee>** içeriği verilmektedir. Bu durumda kod sorunsuz bir şekilde çalışacaktır. Tabiki ekrana yazdırılma sırası değişmeyecektir. Nitekim buradaki tip içeriği saniyenin çok küçük bir diliminde tamamlanacağından farklı **Thread**' lerin işi ele almasına zaman kalmayacaktır. Sakın sırası karışık bir organizasyon ağacı yazdırılmasını beklemeyin. Yaptığımız sadece ve sadece **Int32/Int64** dışında kalan ve **IEnumerable<T>** gibi bir arayüzü implemente etmeyen bir tipe ait çalışma zamanı nesne örneği ve bağlı referanslarının, **Parallel.ForEach** döngüsü tarafından dolaşılabilmesini sağlamaktır.

Ancak yine de dikkat edilmesi gereken önemli bir durum vardır. **IEnumerable<T> thread safe** olmadığından döngünün kullandığı veriye olan erişimler sırasında kilitleme tekniklerinden yararlanılması gerekebilir ki bu da aslında performansı olumsuz yönde etkileyebilecek bir durumdur. Burada performansı arttırmaya yönelik olarak belkide **Iterate** metodunun sonucunun **ToArray** gibi bir metod yardımıyla diziye çevrilmesi düşünülebilir. Aşağıdaki kod parçasında görüldüğü gibi.

```
Parallel.ForEach<Employee>(
  Iterate(root).ToArray(),
  emp => Console.WriteLine("{0} {1}", emp.Profession, emp.Name)
);
```

Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ParallelForNonIntegralRanges_RTM.rar (27,05 kb) **[örnek uygulama Visual Studio 2010 Ultimate RTM sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Workflow Foundation Öğreniyorum - Ders 3 - Yeni Bir Yüz - Flowchart (2010-05-11T00:20:00)

workflow foundation 4.0,workflow foundation,visual studio 2010,

Merhaba Arkadaşlar,

NedirTv?com sponsorluğunda hazırladığımız **"Workflow Foundation 4.0 öğreniyorum"** görsel eğitim serimizin **4ncü** dersi ile karşınızdayız. Hatırlayacağınız üzere bir önceki görsel dersimizde çıtayı biraz yükseltip tamamen kod yardımıyla bir **Workflow** içeriğinin nasıl oluşturulabileceğini incelemeye çalışmıştık. Bu yorucu bölümden sonra daha sakin ilerlemenin ve hafif bir konu ile devam etmenin yararlı olacağı kanısındayım. Bu amaçla sıradaki dersimizde **Workflow Foundation 4.0** ile gelen yeni aktivite bileşenlerinden birisi olan **Flowchart** tipine bir **Merhaba** demeye çalışıyor olacağız. örneği geliştirirken temel olarak **Flowchart** tipinin ne işe yaradığını görecek ve ayrıca **FlowDecision** aktivite bileşenini tanıyor olacağız. Keyifli seyirler dilerim.

Ders 3 - Yeni Bir Yüz - Flowchart

Süre : 13:11

Dosya Boyutu : 14.9 Mb

örnek : Lesson3.rar (37,26 kb)

MP4 Formatında İndirmek için Tıklayın

Entity Framework - POCO ve Lazy Loading (2010-05-10T09:55:00)

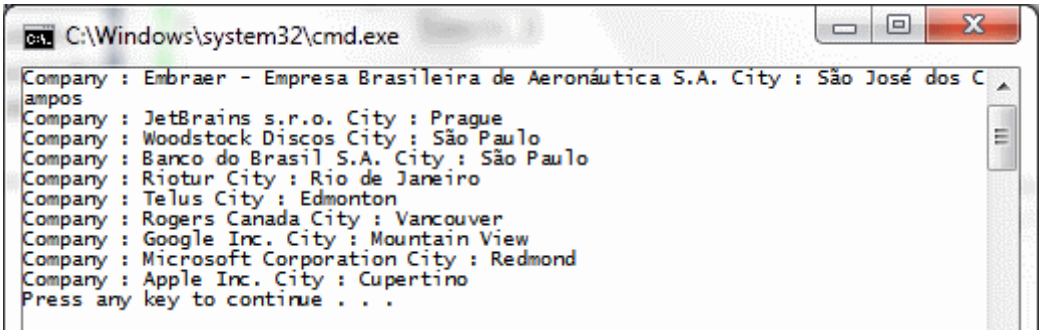
ado.net entity framework 4.0,ado.net entity framework,

Merhaba Arkadaşlar,

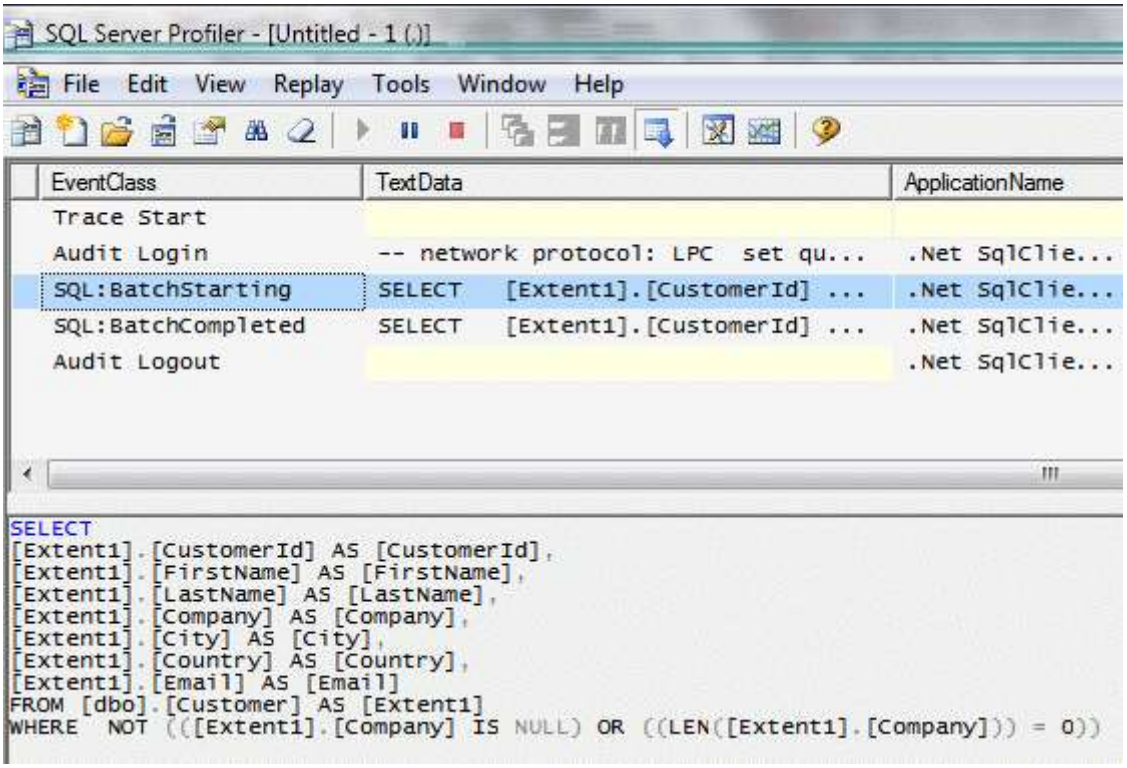
Hatırlayacağınız üzere bir önceki yazımızda **Ado.Net Entity Framework** içerisinde **POCO(Plain Old CLR Object)** nesnelerinin kullanımını incelemeye çalışmıştık. örneğimizde kullanmış olduğumuz **LINQ** sorgusu basit bir **Join** işlemini gerçekleştirmekteydi. Tabi **Join** sorgusu kullandığımız için gözden kaçırdığımız ufak ama bir o kadar da önemli bir vaka oluşmaktadır. Bu vakayı ele almak için program kodunu biraz daha değiştirdiğimizi ve aşağıdaki hale getirdiğimizi düşünelim.

```
using System;
using System.Linq;
using ChinookModel;

namespace POCODans
{
   class Program
   {
      static void Main(string[] args)
      {
         using (ChinookEntities entities = new ChinookEntities())
         {
            #region Sample 2

            // Şirket bilgisi null veya boş olmayan müşteriler
            var customers = from c in entities.Customers
                            where !String.IsNullOrEmpty(c.Company)
                            select c;

            foreach (var customer in customers)
            {
               Console.WriteLine("Company : {0} City :
{1}",customer.Company,customer.City);

               // ve bu müşterilere ait fatura bilgileri
               foreach (var invoice in customer.Invoices)
               {
                  Console.WriteLine("\tInvoice Date : {0} ,Total :
{1}",invoice.InvoiceDate,invoice.Total.ToString("C2"));
               }
            }

            #endregion
         }
      }
   }
}
```

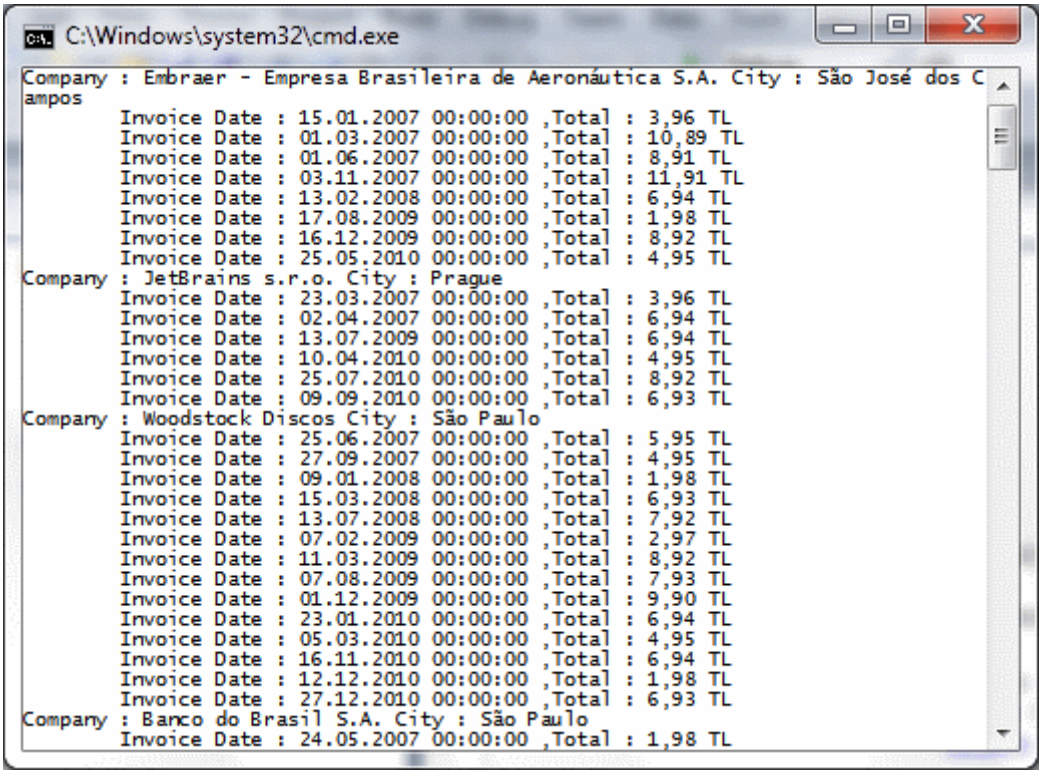
örneğimizin bu yeni haline göre **Company** alanı dolu olan*(Null veya Empty olmayan)* müşterilerin şirket adları ile bulundukları şehir bilgileri, ayrıca bu firmaların faturalarına ait tarih ve tutarları ekrana yazdırılmaktadır. Bu kodun çalışması sırasındaki beklentimiz ise **Customer** ve bunlara bağlı olan **Invoice** bilgilerinin getirilmesidir. Ancak uygulamayı çalıştırdığımızda aşağıdaki ekran görüntüsünde yer alan sonuçları elde ettiğimizi görürüz.



Dikkat edileceği üzere sadece **Customer** bilgileri çekilebilmiş ancak iç **foreach** döngüsü tarafından o anki **Customer** nesnesine ait **Invoice** bilgileri yazdırılmamıştır. Aslında bunun sebebini anlamak için **SQL Server Profiler** aracı yardımıyla arka planda çalıştırılan **SQL** sorgularına bakmamız yeterli olacaktır. Nitekim aşağıdaki ekran görüntüsünde yer alan sonuçlar ile karşılaştığımızı görürüz.

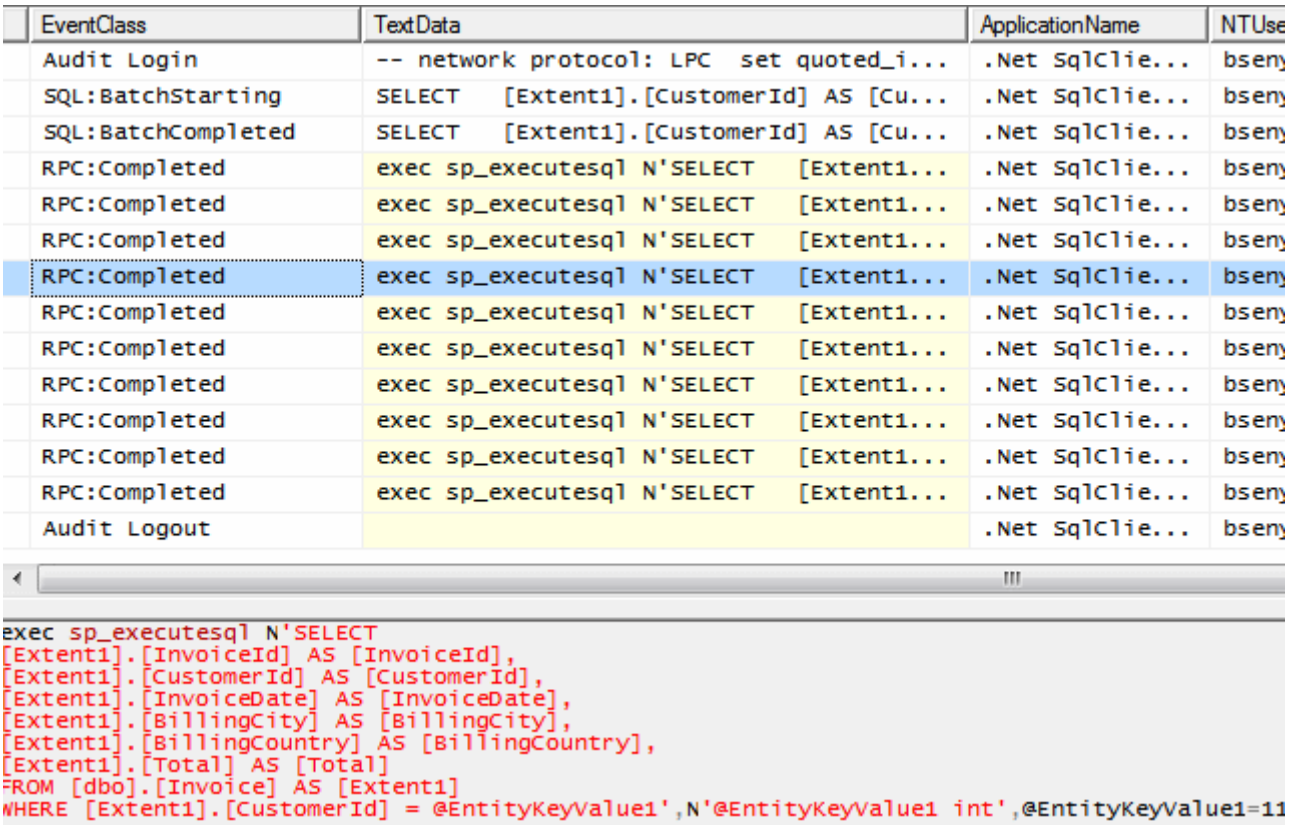


Dikkat edileceği üzere kodun çalışması sonrasında sadece **Customer** bilgilerinin çekildiği görülmektedir. Tabi bu noktada örneğimizin **POCO** nesnelerini kullandığını unutmayalım. Eğer örneğimize ait **Entity Model**' in otomatik olarak üretildiğini düşünürsek aynı kodun çalışma zamanında aşağıdaki sonuçları ürettiğini görebiliriz.



```
Company : Embraer - Empresa Brasileira de Aeronáutica S.A. City : São José dos C
ampos
        Invoice Date : 15.01.2007 00:00:00 ,Total : 3,96 TL
        Invoice Date : 01.03.2007 00:00:00 ,Total : 10,89 TL
        Invoice Date : 01.06.2007 00:00:00 ,Total : 8,91 TL
        Invoice Date : 03.11.2007 00:00:00 ,Total : 11,91 TL
        Invoice Date : 13.02.2008 00:00:00 ,Total : 6,94 TL
        Invoice Date : 17.08.2009 00:00:00 ,Total : 1,98 TL
        Invoice Date : 16.12.2009 00:00:00 ,Total : 8,92 TL
        Invoice Date : 25.05.2010 00:00:00 ,Total : 4,95 TL
Company : JetBrains s.r.o. City : Prague
        Invoice Date : 23.03.2007 00:00:00 ,Total : 3,96 TL
        Invoice Date : 02.04.2007 00:00:00 ,Total : 6,94 TL
        Invoice Date : 13.07.2009 00:00:00 ,Total : 6,94 TL
        Invoice Date : 10.04.2010 00:00:00 ,Total : 4,95 TL
        Invoice Date : 25.07.2010 00:00:00 ,Total : 8,92 TL
        Invoice Date : 09.09.2010 00:00:00 ,Total : 6,93 TL
Company : Woodstock Discos City : São Paulo
        Invoice Date : 25.06.2007 00:00:00 ,Total : 5,95 TL
        Invoice Date : 27.09.2007 00:00:00 ,Total : 4,95 TL
        Invoice Date : 09.01.2008 00:00:00 ,Total : 1,98 TL
        Invoice Date : 15.03.2008 00:00:00 ,Total : 6,93 TL
        Invoice Date : 13.07.2008 00:00:00 ,Total : 7,92 TL
        Invoice Date : 07.02.2009 00:00:00 ,Total : 2,97 TL
        Invoice Date : 11.03.2009 00:00:00 ,Total : 8,92 TL
        Invoice Date : 07.08.2009 00:00:00 ,Total : 7,93 TL
        Invoice Date : 01.12.2009 00:00:00 ,Total : 9,90 TL
        Invoice Date : 23.01.2010 00:00:00 ,Total : 6,94 TL
        Invoice Date : 05.03.2010 00:00:00 ,Total : 4,95 TL
        Invoice Date : 16.11.2010 00:00:00 ,Total : 6,94 TL
        Invoice Date : 12.12.2010 00:00:00 ,Total : 1,98 TL
        Invoice Date : 27.12.2010 00:00:00 ,Total : 6,93 TL
Company : Banco do Brasil S.A. City : São Paulo
        Invoice Date : 24.05.2007 00:00:00 ,Total : 1,98 TL
```

Dikkat edileceği üzere fatura bilgileri ekrana yazdırılmaktadır. Tabi **SQL Server Profiler** aracının üzerinden çalıştırılan **SQL** sorgularına baktığımızda aşağıdaki ifadelerin yer aldığını da görebiliriz.



EventClass	TextData	ApplicationName	NTUse
Audit Login	-- network protocol: LPC set quoted_i...	.Net SqlClie...	bseny
SQL:BatchStarting	SELECT [Extent1].[CustomerId] AS [Cu...	.Net SqlClie...	bseny
SQL:BatchCompleted	SELECT [Extent1].[CustomerId] AS [Cu...	.Net SqlClie...	bseny
RPC:Completed	exec sp_executesql N'SELECT [Extent1...	.Net SqlClie...	bseny
RPC:Completed	exec sp_executesql N'SELECT [Extent1...	.Net SqlClie...	bseny
RPC:Completed	exec sp_executesql N'SELECT [Extent1...	.Net SqlClie...	bseny
RPC:Completed	exec sp_executesql N'SELECT [Extent1...	.Net SqlClie...	bseny
RPC:Completed	exec sp_executesql N'SELECT [Extent1...	.Net SqlClie...	bseny
RPC:Completed	exec sp_executesql N'SELECT [Extent1...	.Net SqlClie...	bseny
RPC:Completed	exec sp_executesql N'SELECT [Extent1...	.Net SqlClie...	bseny
RPC:Completed	exec sp_executesql N'SELECT [Extent1...	.Net SqlClie...	bseny
RPC:Completed	exec sp_executesql N'SELECT [Extent1...	.Net SqlClie...	bseny
RPC:Completed	exec sp_executesql N'SELECT [Extent1...	.Net SqlClie...	bseny
Audit Logout		.Net SqlClie...	bseny

```
exec sp_executesql N'SELECT
[Extent1].[InvoiceId] AS [InvoiceId],
[Extent1].[CustomerId] AS [CustomerId],
[Extent1].[InvoiceDate] AS [InvoiceDate],
[Extent1].[BillingCity] AS [BillingCity],
[Extent1].[BillingCountry] AS [BillingCountry],
[Extent1].[Total] AS [Total]
FROM [dbo].[Invoice] AS [Extent1]
WHERE [Extent1].[CustomerId] = @EntityKeyValue1',N'@EntityKeyValue1 int',@EntityKeyValue1=11
```

Dikkat edileceği üzere kaç tane **Customer** bilgisi çekilmişse her biri için **Invoice** tablosundan bilgi çekmek amacıyla birer **SQL** sorgusu icra edilmiştir. çok doğal olarak burada **Ado.Net Entity Framework**' ün **4.0** versiyonu ile birlikte otomatik olarak kazandığı **Lazy Loading** kabiliyetinin devreye girdiğini söyleyebiliriz.
Peki **POCO** nesneleri için **Lazy Loading** işlevselliğini nasıl kazandırabiliriz? Burada biraz dikkali olunması gerekmektedir. Nitekim **Lazy Loading** yeteneği için yapılacak eklentiler, **POCO** nesnelerinden **Framework**' e doğru bir bağımlılık oluşturmamalıdır. Normal şartlarda **Lazy Loading** işlemi için akla gelen ilk yöntem **Customer** tipi içerisinde yer alan **Invoices** özelliğinin **get** bloğunda gerekli kodlamaları yapmaktır. Oysaki bu hamle sonucunda **Framework**' e bir bağımlılık oluşturulması kaçınılmazdır ki **POCO** nesnelerinin ilkesine aykırı bir durumdur. Bu nedenle **dinamik olarak üretilen proxy tiplerinden(Dynamically Generated Proxies)** yararlanılmaktadır. Bu proxy tipleri aslında **POCO** nesnesinden türeyen ve ilgili özellikleri **override** ederek **Lazy Loading** gibi yetenekleri içeriye enjekte eden sınıflar olarak düşünülebilir. Aslında **Ado.Net Team Blog**' da bu tip **Proxy** sınıflarının nasıl geliştirildiği ve kullanıldığından bahsedilmektedir. Ancak biz çok daha basit bir yol izliyor olacağız.

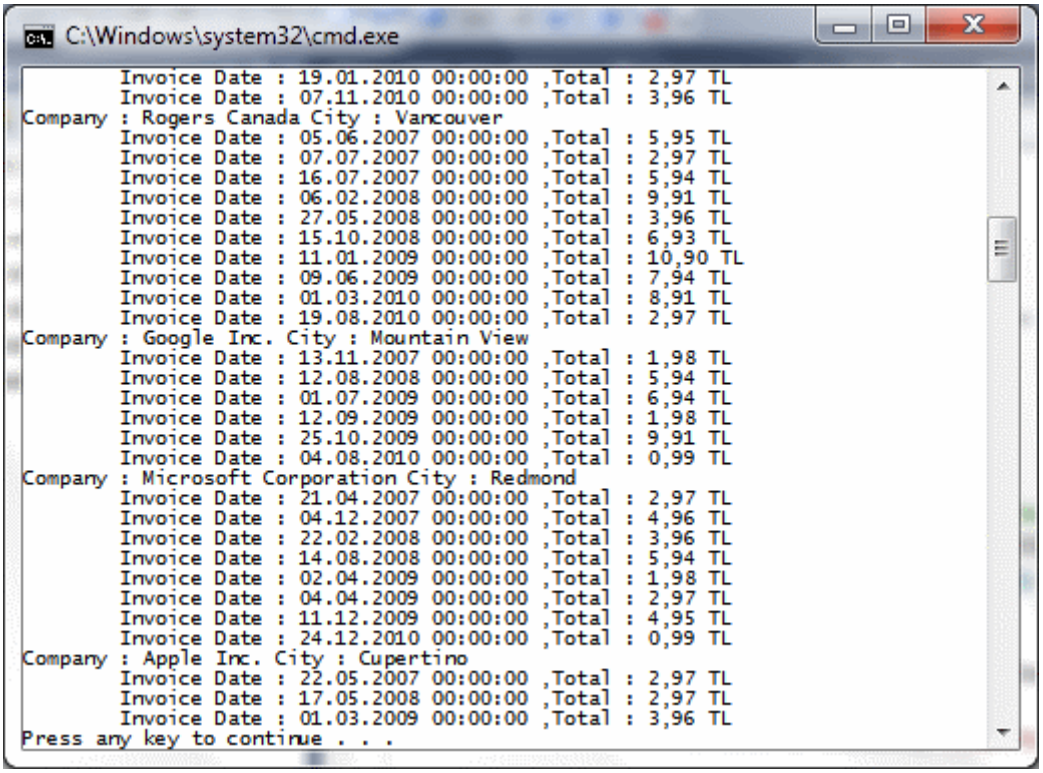
POCO nesnelerinde **Lazy Loading** işlemini gerçekleştirebilmemiz için üç basit kurala dikkat etmemiz yeterlidir. Buna göre **Context** tipine ait **yapıcı metod(Constructor)** içerisinde gerekli bazı üst özelliklerin etkinleştirilmesi, **Navigation Property**' nin **virtual** olarak tanımlanması ve **POCO** tipinin **public** ve **sealed olmaması** yeterlidir. örneğimizin kod içeriğinin yeni hali aşağıdaki gibidir.

```
using System;
using System.Collections.Generic;
using System.Data.Objects;

namespace ChinookModel
{
   public class ChinookEntities
      :ObjectContext
   {
      private ObjectSet<Customer> _customers;
      private ObjectSet<Invoice> _invoices;

      public ChinookEntities()
         :base("name=ChinookEntities")
      {
         // KURAL 1:
         // Dinamik proxy üretimi etkinleştirilir. Varsayılan değeri true dur
         this.ContextOptions.ProxyCreationEnabled = true;
         // çalışma zamanında Navigation Property' lere erişildiğinde Lazy Loading işleminin otomatik olarak gerçekleştirilmesi için true değeri atanır.
```

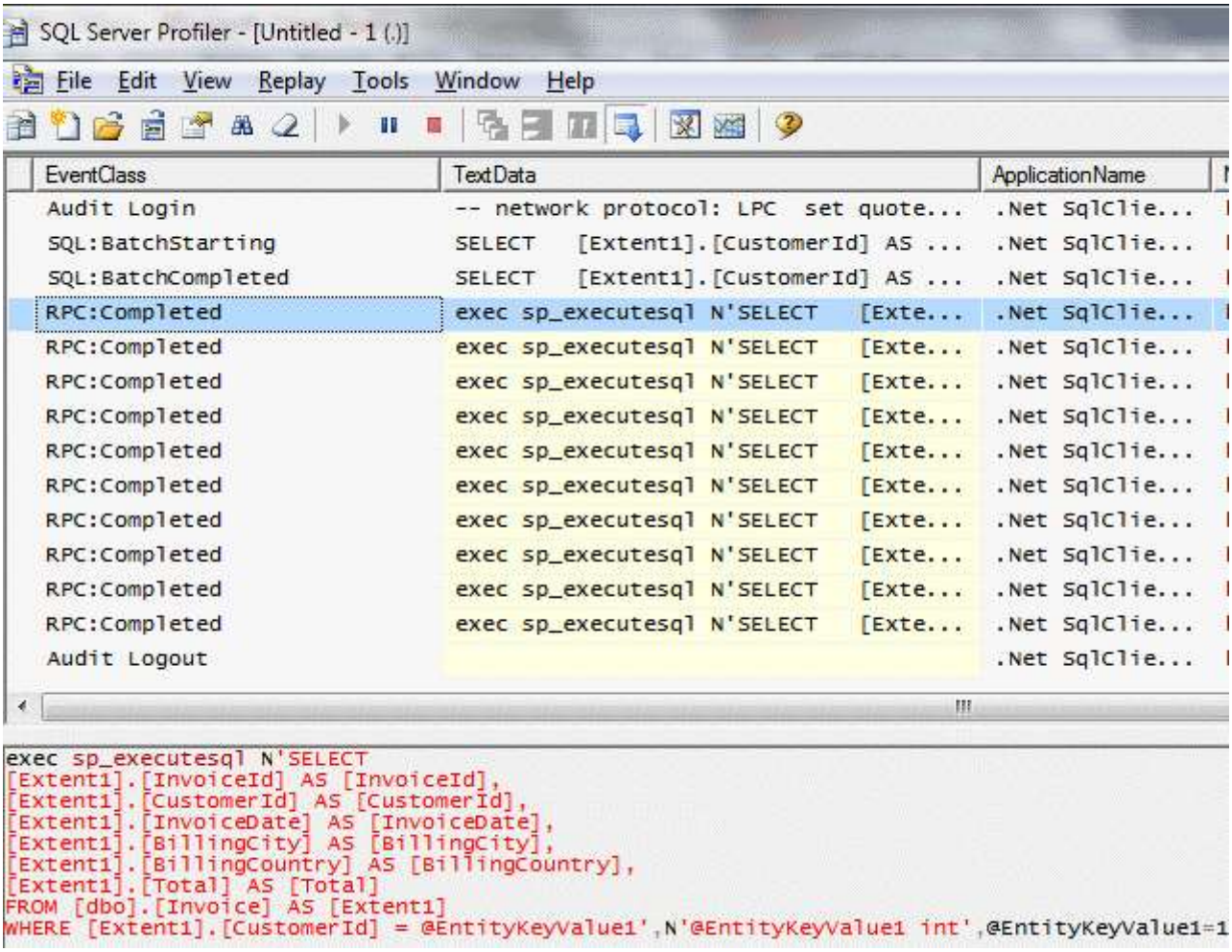
```
        this.ContextOptions.LazyLoadingEnabled = true;
    }

    public ObjectSet<Customer> Customers
    {
        get
        {
            if (_customers == null)
            {
                _customers = base.CreateObjectSet<Customer>();
            }
            return _customers;
        }
    }

    public ObjectSet<Invoice> Invoices
    {
        get
        {
            if (_invoices == null)
            {
                _invoices = base.CreateObjectSet<Invoice>();
            }
            return _invoices;
        }
    }
}

// KURAL 2 : POCO sınıfı public olmalı ve sealed olarak işaretlenmemelidir.
public class Customer
{
    public int CustomerId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Company { get; set; }
    public string City { get; set; }
    public string Country { get; set; }
    public string Email { get; set; }

    private List<Invoice> _invoices = new List<Invoice>();

    // KURAL 3 : Navigation Property özelliğinin virtual olması gerekir.
    public virtual List<Invoice> Invoices
    {
        get { return _invoices; }
```

```
            set { _invoices = value; }
        }
    }

    public class Invoice
    {
        public int InvoiceId { get; set; }
        public int CustomerId { get; set; }
        public DateTime InvoiceDate{ get; set; }
        public string BillingCity { get; set; }
        public string BillingCountry { get; set; }
        public decimal Total { get; set; }
        public Customer Customer { get; set; }
    }
}
```

Ve örneğimizin çalışma zamanı hali;



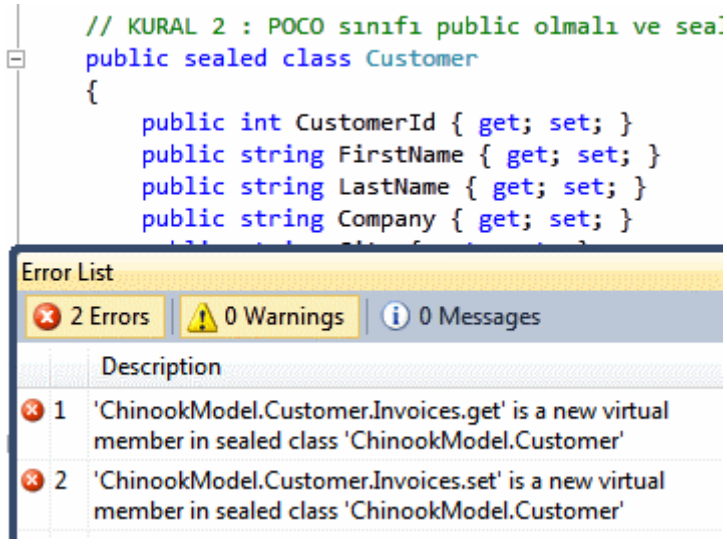
Volaaaa!!! 😃 üstelik **SQL Server Profiler** aracında da tam istediğimiz şekilde sorguların çalıştırıldığını görebiliriz.



EventClass	TextData	ApplicationName
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...
SQL:BatchStarting	SELECT [Extent1].[CustomerId] AS ...	.Net SqlClie...
SQL:BatchCompleted	SELECT [Extent1].[CustomerId] AS ...	.Net SqlClie...
RPC:Completed	exec sp_executesql N'SELECT [Exte...	.Net SqlClie...
RPC:Completed	exec sp_executesql N'SELECT [Exte...	.Net SqlClie...
RPC:Completed	exec sp_executesql N'SELECT [Exte...	.Net SqlClie...
RPC:Completed	exec sp_executesql N'SELECT [Exte...	.Net SqlClie...
RPC:Completed	exec sp_executesql N'SELECT [Exte...	.Net SqlClie...
RPC:Completed	exec sp_executesql N'SELECT [Exte...	.Net SqlClie...
RPC:Completed	exec sp_executesql N'SELECT [Exte...	.Net SqlClie...
RPC:Completed	exec sp_executesql N'SELECT [Exte...	.Net SqlClie...
RPC:Completed	exec sp_executesql N'SELECT [Exte...	.Net SqlClie...
RPC:Completed	exec sp_executesql N'SELECT [Exte...	.Net SqlClie...
Audit Logout		.Net SqlClie...

```
exec sp_executesql N'SELECT
[Extent1].[InvoiceId] AS [InvoiceId],
[Extent1].[CustomerId] AS [CustomerId],
[Extent1].[InvoiceDate] AS [InvoiceDate],
[Extent1].[BillingCity] AS [BillingCity],
[Extent1].[BillingCountry] AS [BillingCountry],
[Extent1].[Total] AS [Total]
FROM [dbo].[Invoice] AS [Extent1]
WHERE [Extent1].[CustomerId] = @EntityKeyValue1',N'@EntityKeyValue1 int',@EntityKeyValue1=1
```

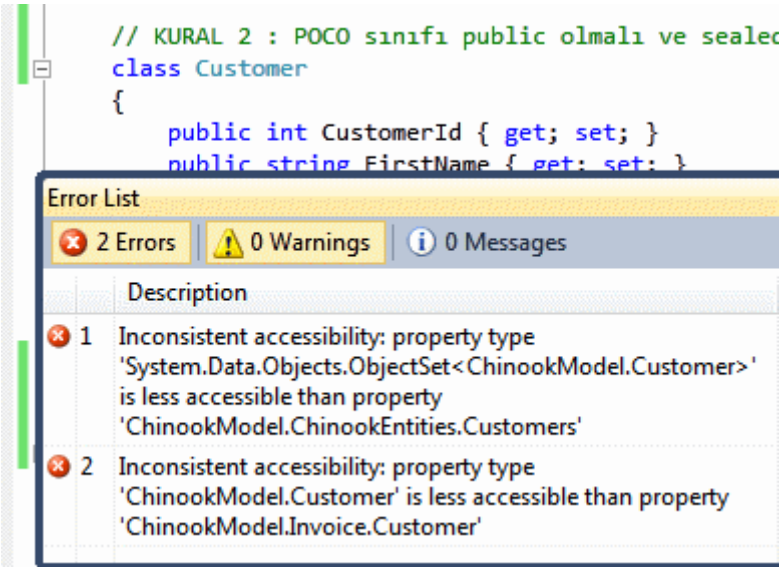
Tabi buradaki kuralların neden var olduğuna dikkat edilmelidir. Söz gelimi **LazyLoadingEnabled**, **ProxyCreationEnabled** değerlerinin **true** olması çalışma zamanındaki **Entity** motoru için gereklidir. Bu değerlerin **false** olması halinde bir hata mesajı alınmaz ancak istenen çalışma zamanı çıktıları da elde edilmez. Diğer taraftan **POCO** tipinin **sealed** olması halinde zaten**virtual** eleman içermesi ile ilişkili olarak derleme zamanı hatası alınacaktır.



```
// KURAL 2 : POCO sınıfı public olmalı ve sea
public sealed class Customer
{
    public int CustomerId { get; set; }
    public string FirstName { get; set; }
    public string LastName { get; set; }
    public string Company { get; set; }
```

	Description
1	'ChinookModel.Customer.Invoices.get' is a new virtual member in sealed class 'ChinookModel.Customer'
2	'ChinookModel.Customer.Invoices.set' is a new virtual member in sealed class 'ChinookModel.Customer'

Buna ek olarak derleme zamanı **POCO** nesnesinin **public** olmasını beklemektedir. **Public** olarak tanımlanmadığı takdirde yine bir hata mesajı üretilecektir.



Navigation Property' nin **virtual** olarak tanımlanmaması derleme veya çalışma zamanında bir hata mesajı üretmezken **Lazy Loading** işleminin de çalışmamasına neden olmaktadır. İşte bu kadar. Bu yazımızda **POCO** nesnelerinin kullanıldığı vakalarda **Lazy Loading** işlemlerinin nasıl ektinleştirilebileceğini incelemeye çalıştık. Unuttuğumuz bir şey var mı? Aslında olabilir. Eğer öyleyse de ilerleyen yazılarımızda ele almaya çalışacağız. Bir sonraki yazımızda görüşünceye dek hepinize mutlu günler dilerim.

POCOandLazyLoading_RTM.rar (90,73 kb) **[örnek uygulama Visual Studio 2010 Ultimate RTM sürümü üzerinde geliştirilmiş test edilmiştir]**

Workflow Foundation Öğreniyorum - Ders 2 - Kodla Başbaşayız (2010-05-04T15:25:00)

workflow foundation 4.0,workflow foundation,visual studio 2010,

Merhaba Arkadaşlar,

"Workflow Foundation 4.0 öğreniyorum" serimizin üçüncü dersi ile karşınızdayız. Bu kez bir sıçrama yaparak **Level** değerimizi **101**' den **110**' a çekiyoruz. Bunu yaparken de kendimize biraz eziyet ediyoruz. Eziyet etmek içinde, bir önceki dersimizde ele aldığımız **Visual Studio 2010 IDE**' sinin sunduğu**WPF** tabanlı güzelim **Workflow Designer** ortamını bırakarak, bir **Workflow** örneğinin ve tüm içeriğinin tamamen kod bazlı olarak nasıl geliştirilebileceğini görmeye çalışıyoruz. Ancak bu eziyet sayesinde, yeri geldiğinde söz konusu akışların kod yardımıyla dinamik olaraktan üretilebileceğini öğreniyor ve bunun bazı vakaları karşılayacağını farkediyoruz. örnek akışta bir önceki akışın aynısının kod tarafına geliştirilmesini gerçekleştiriyoruz. Bu sırada **Variable, InArgument, OutArgument, ActivityContext, ExpressionServices** gibi önemli tipleri de öğreniyor ve nasıl kullanıldıklarını görüyoruz. Bakalım eziyetimize değecek mi? İyi seyirler dilerim.

Ders 2 - Kodla Başbaşayız

Süre : 17:15

Dosya Boyutu : 18.3 Mb

örnek : Lesson2_KodlaBasbasayiz.rar (40,12 kb)

Mp4 Formatında İndirmek İçin Tıklayın

WF Ado.Net Entity Pack - Hello World (2010-05-03T10:00:00)

workflow foundation 4.0,wf ado.net activity pack,wf state machine activity pack,wf,wf 4.0,activity,

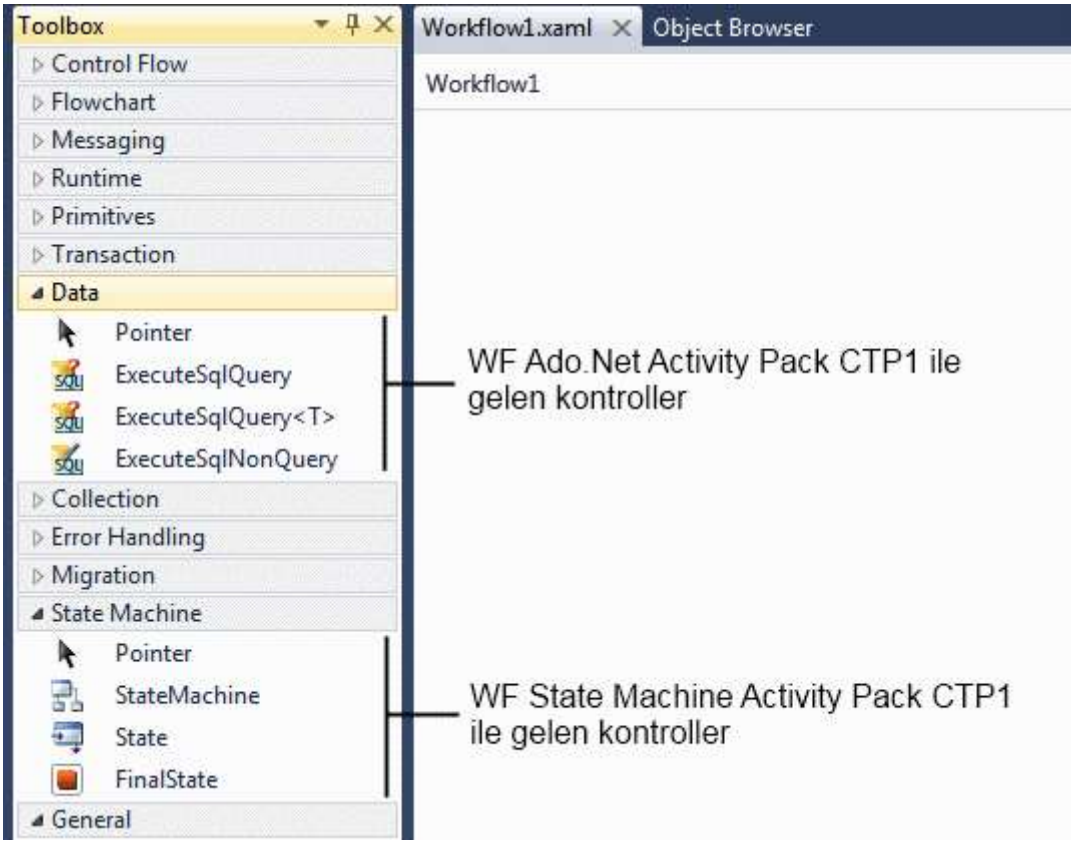
Merhaba Arkadaşlar,

2008 yılının son çeyreğinde **Microsoft** tarafından düzenlenen **Yazılım Geliştiriciler Zirvesinde**, **WCF & WF 4.0** konulu bir sunumum olmuştu.**Derinlere Dalıyoruz** mesajını içeren etkinliğin sunum dosyalarının arka plan resminde, dipteki bilgisayara ulaşmaya çalışan bir balık adam motifi yer almaktaydı. Sevgili **Mehmet Emre**' nin beni davet ettiği bu oturuma hazırlanırken, **Microsoft PDC 2008**' de dağıtılan **VHD**' ler üzerinde çalışmıştım. O zamanlarda **WF Designer** üzerinde dikkatimi çeken noktaların

başında, **WF** aktivitelerindeki çeşitlilik yer almaktaydı. özellike **DbQuery, DbUpdate** isimli **SQL** odaklı aktivite bileşenleri dikkatimi çekmişti.

Ne var ki ilerleyen zamanlarda çıkan **PreBeta, Beta 1, Beta 2, RC** ve nihayet **RTM** sürümlerinde yer alan **Activity Component** setinde bu tip bileşenlerin yer almadığına da şahit olduk. Hatta **WF 4.0** öncesinde aşina olduğumuz **State Machine** tipinden şablonlarında kaldırıldığını gördük. Geçtiğimiz günlerde ise **Codeplex** üzerinden iki **WF 4.0 Activity Pack** yayınlandı. Bunlardan birisi WF Ado.Net Activity Pack CTP 1 iken diğer ise WF State Machine Activity Pack 1 isimli paketti.

Söz konusu paketleri indirip kurduğumuzda **Visual Studio 2010 Workflow Designer**' a aşağıdaki ekran görüntüsünde yer alan kontrollerin eklendiğini fark edebiliriz.

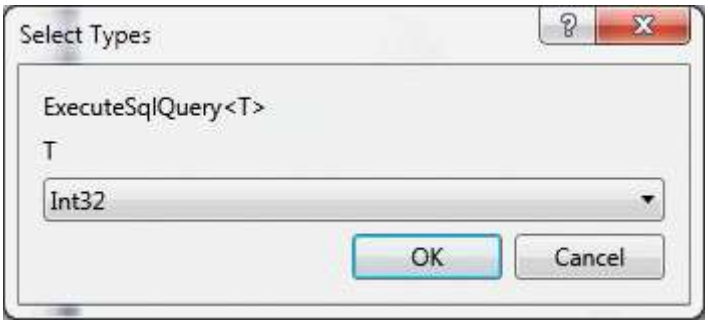


İşte bu yazımızda çok basit olarak **WF Ado.Net Activity Pack** içerisindeki kontrollerden birisini tanımaya çalışacak ve basit bir örnek geliştiriyor olacağız. Bu tip **Pack**' leri işlerimizi son derece kolaylaştıracak ve **Activity** bileşenlerinin daha zengin bir çerçevede değerlendirilmesine olanak sağlayacak genişletmeler olarak düşünmemizde yarar vardır. özellikle **Codeplex** gibi açık kaynak kodlu projeleri yayınlayan ve **Microsoft** tarafından desteklenen bu tip bileşen paketleri, kişisel düşünceme göre WF tasarımlarında önemli bir rol üstlenecekler gibi görünmekte. Haydi gelin parmaklarımızı sıvayalım ve basit bir **Workflow Console Application** projesi oluşturarak yola koyulalım. İlk olarak senaryomuzdan bahsetmemizde yarar olacağı kanısındayım.

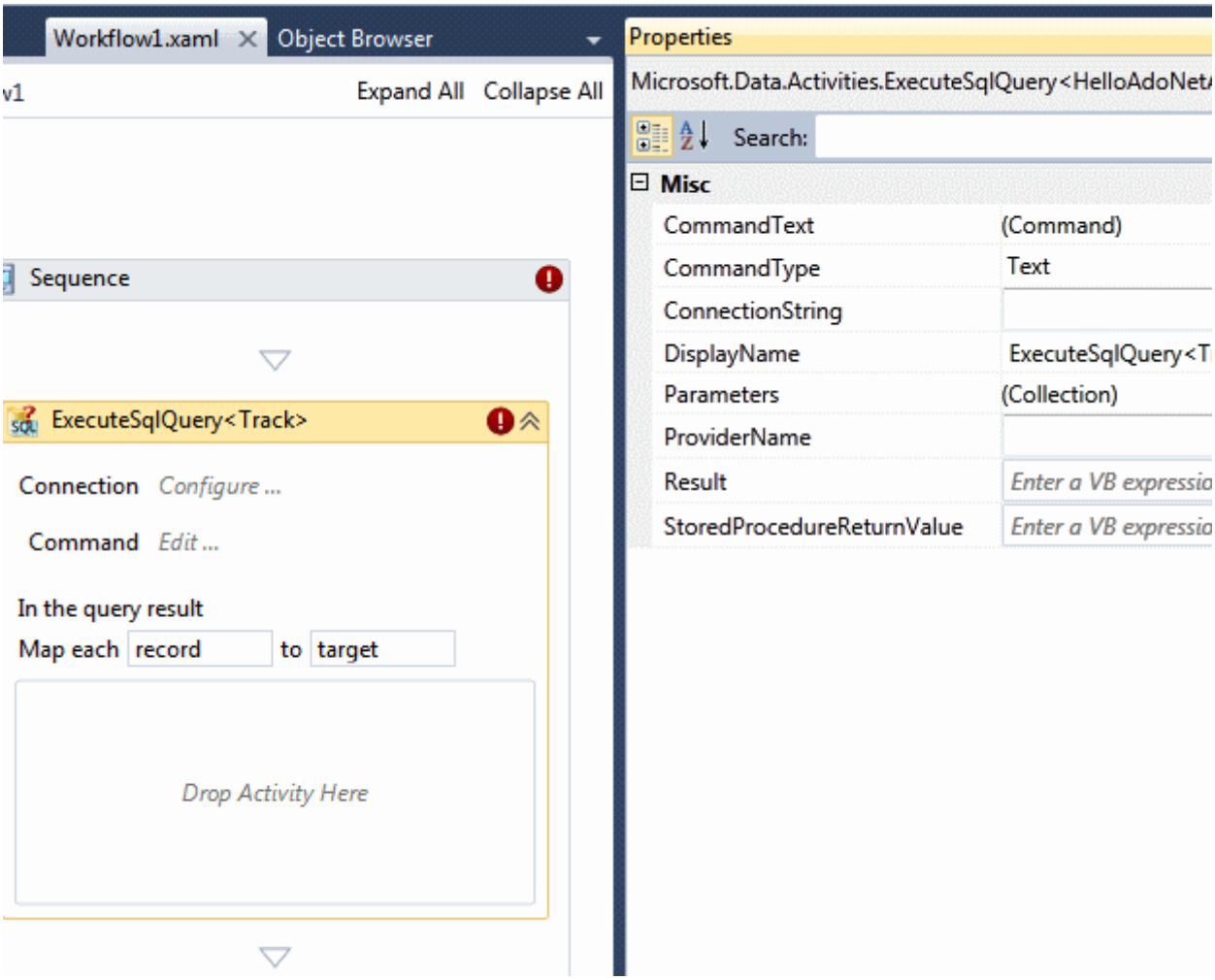
örneğimizde bir **Select SQL** sorgusunu, **SQL 2008** sunucusu üzerinde konuşlandırılmış ve yine **Codeplex** sitesinden indirip kurabileceğiniz **Chinook** veritabanı üzerindeki **Track** tablosunda çalıştırıyor olacağız. Bu sorgunun çalıştırılması sonucu elde edilen **veri kümesi(Result Set)** içerisindeki her bir satırı ise, aşağıda tasarımı görülen **Track** isimli sınıfa ait nesne örneklerinin oluşturulmasında kullanacağız.

```
namespace HelloAdoNetActivityPack
{
   public class Track
   {
      public int TrackId { get; set; }
      public string Name { get; set; }
      public string Composer { get; set; }
      public int Milliseconds { get; set; }

      public override string ToString()
      {
         return string.Format("{0}  Şarkı : {1} Besteci : {2} Süre : {3} ",
TrackId.ToString(), Name, Composer, Milliseconds.ToString());
      }
   }

}
```

Şimdi **Workflow** ortamımızda **Sequence** diagramı içerisine bir adet **ExecuteSqlQuery<T>** tipinden bileşeni sürükleyelip bırakalım. Sürükleme işlemi sonrasında **T** tipi için aşağıdaki ekran görüntüsünde yer alan soru ile karşılaşırız.

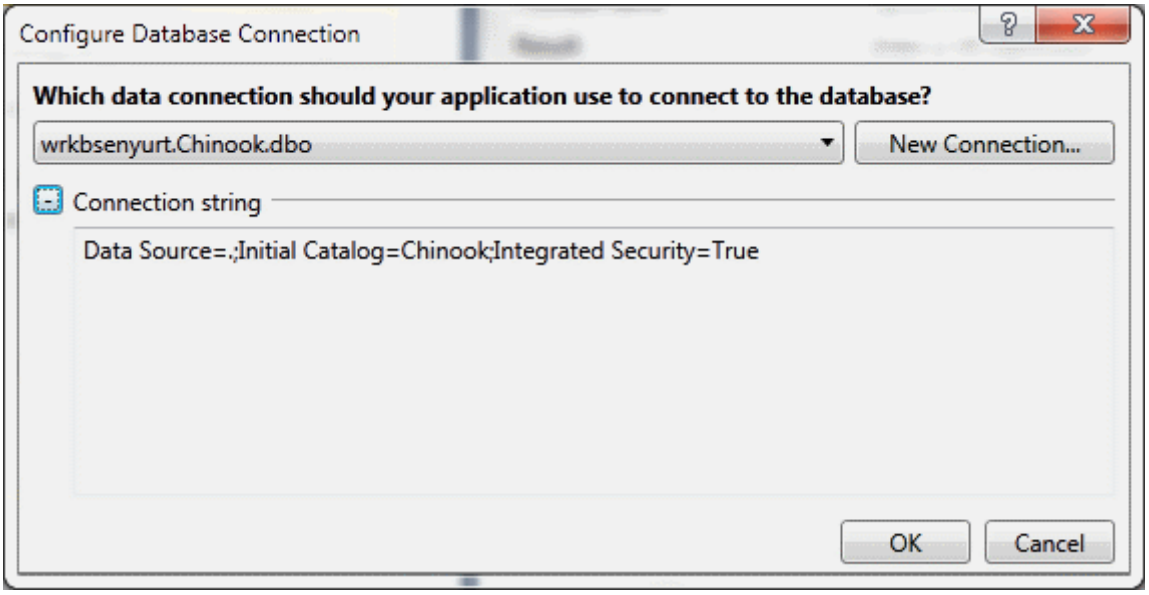


çok doğal olarak sorgu sonuçlarının **Track** tipinden nesne örnekleri içerisinde toplanmasını planladığımızdan bu sınıfı seçmemiz gerekmektedir. **Browse for Types** kısmından**Track** sınıfını işaretledikten sonra kontrolümüzün **WF Designer** ortamına aşağıdaki gibi eklendiğini görebiliriz.

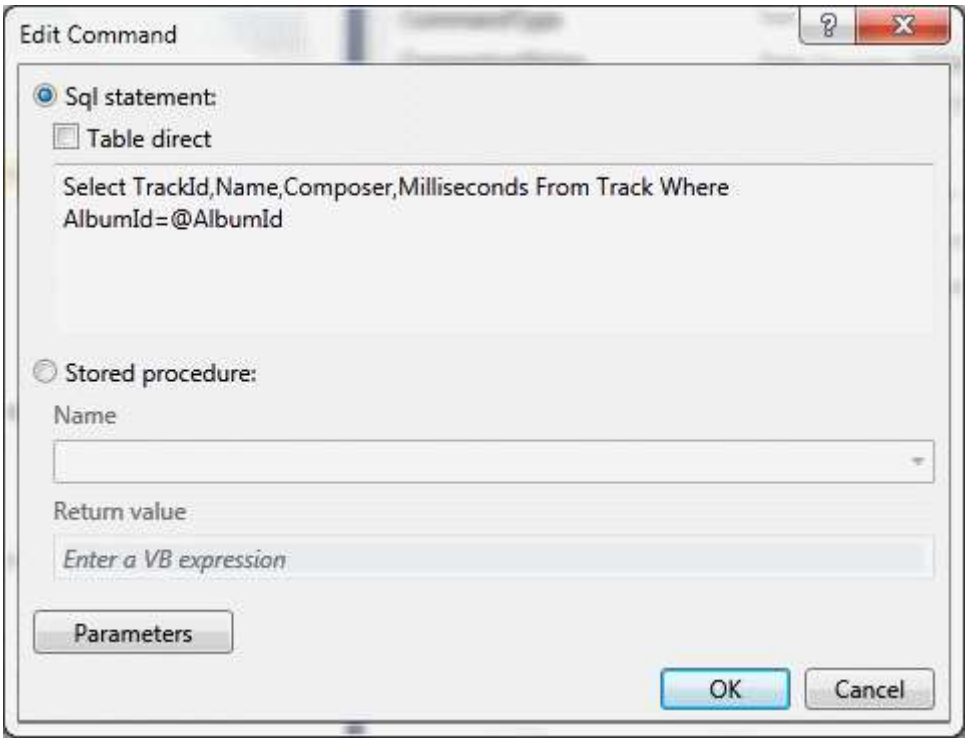


Eğer bu tip bir **Activity** bileşenini kendimiz tasarlıyor olsaydık, çok doğal olarak sorgunun çalıştırılması için gerekli bağlantıyı ve sorgunun kendisini birer **özellik(Property)** olarak sunmayı planlardık. Benzer durum **ExecuteSqlQuery<T>** bileşeni için de geçerlidir. Bir başka deyişle **ConnectionString, CommandText** ve **CommandType** özelliklerinin bildirilmesi gerekmektedir. Tam bu noktada söz konusu bileşenin **SqlCommand** nesne örneğinin görsel bir formatı olduğunu da ifade edebiliriz. özellikle **CommandType, ProviderName** özellikleri aldığı değerler ile bunu ispat eder niteliktedir. **CommandType** özelliğine **Text, StoredProcedure, TableDirect** değerlerinden birisini verebiliriz. Buna göre bir **Stored Procedure**'ün, **Tablo** adının veya bizim tarafımızdan yazılacak bir **SQL Sorgusunun** kullanılabilmesi mümkündür. Ayrıca parametreli sorgulamalar için **Parameters** özelliği göz önüne alınabilmektedir. Standart olarak **SqlClient Veri Sağlayıcısı(Data Provider)** kullanılmaktadır. Ancak dilerseniz diğer **Provider** tiplerini de seçebilir ve farklı veri kaynakları için gerekli sorguları yürütebilirsiniz. **ConnectionString, Parameters** ve**CommandText** vb özelliklerinin yanında bulunan **üç nokta düğmeleri**, sağladıkları arabirimler ile ilgili özelliklerin kolayca belirlenebilmesini sağlamaktadır.

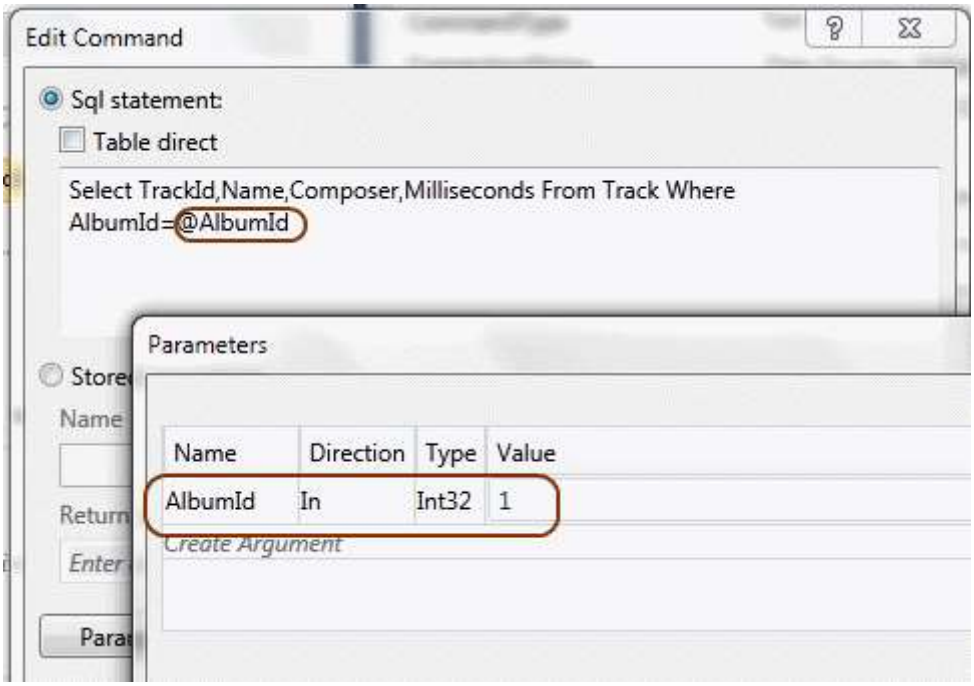
örneğimizde **ConnectionString** için aşağıdaki ayarlar kullanılmaktadır.



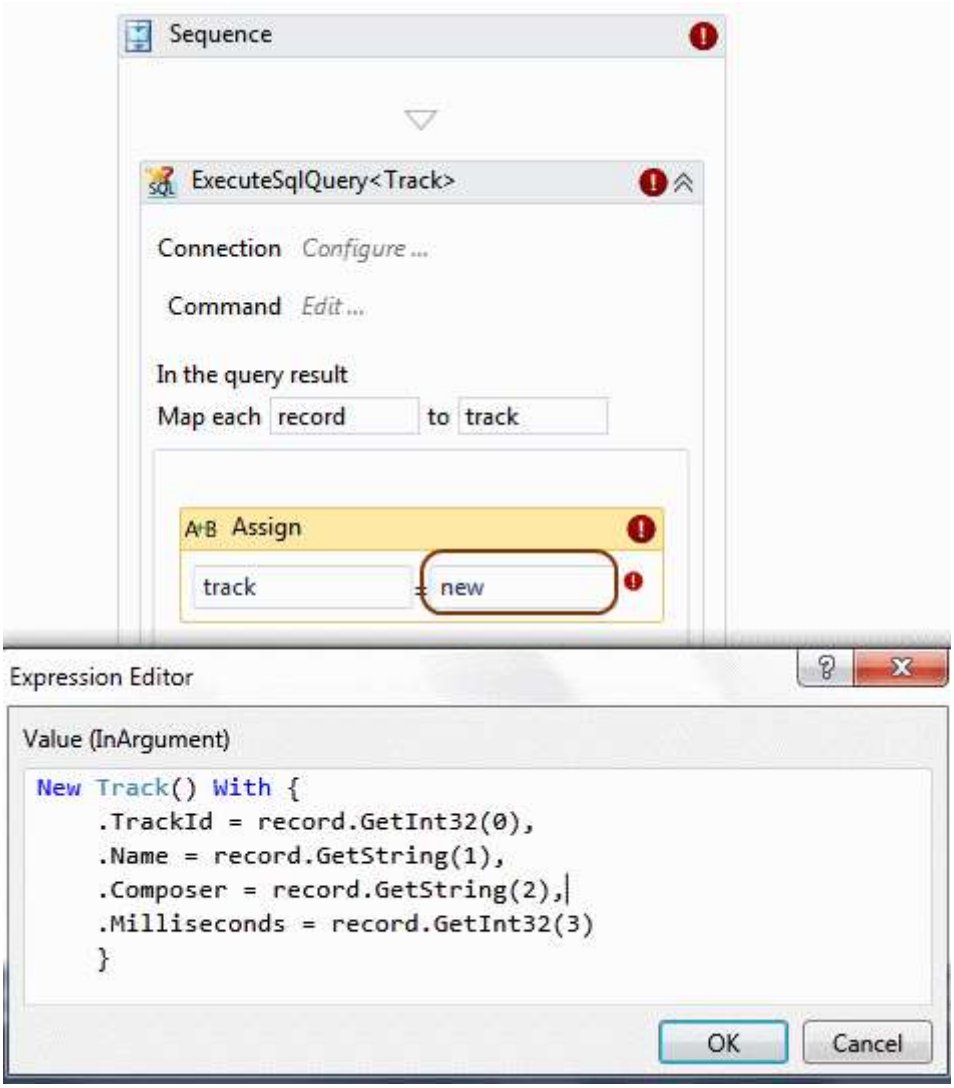
CommandText özelliğini ise aşağıdaki gibi belirlediğimizi düşünebiliriz.



Buna göre **AlbumId** değerine göre **Track** bilgilerini getiren bir SQL sorgusunu çalıştırıyor olacağız. **SQL** sorgusunda **@AlbumId** isimli bir parametre kullandığımızdan bunun değerini ortama alabilmek amacıyla bir **Argument** örneğinden yararlanıyor olacağız. Söz konusu parametre bildirimini ise aşağıdaki gibi yaptığımızı düşünebiliriz.



Buraya kadar her şey güzel. Peki elde ettiğimiz veri kümesini nasıl değerlendireceğiz? Aslında **ExecuteSqlQuery<T>** kontrolü sorgu sonuçlarının **SqlDataReader** yardımıyla dolaşılmasına olanak sağlamaktadır. Bu nedenle **Map Each record to target** gibi bir alt içeriğe sahiptir. Biz de buradaki alanda örnek olarak bir **Assign** aktivitesi kullanacak ve **target** nesne örneklerini oluşturacağız. Burada söz konusu olan **target** değişkeni **T** tipindendir. Dolayısıyla **Track** sınıfını işaret etmektedir. Diğer yandan **record** değişkeni ise, sorgu sonucu okunmakta olan her bir satırı ifade etmektedir. Bu bilgiler ışığında kontrolün içeriğini ve **Assign** aktivitesinin **To** özelliğini aşağıdaki **Visual Basic Expression** ile tamamlayabiliriz. *(Bildiğiniz üzere**WF** tarafındaki tüm **Expression**' lar **Visual Basic Syntax** ' ında yazılmaktadır)*

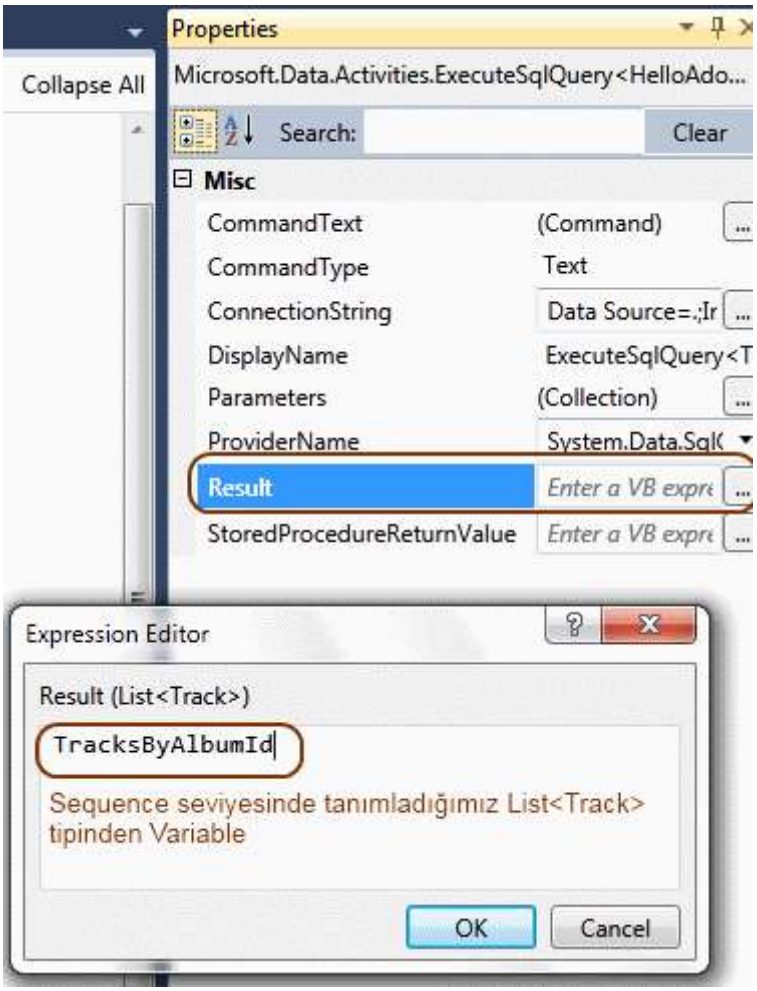


Görüldüğü üzere **Track** nesne örneğini oluştururken **record** değişkeni üzerinden ilgili **alan değerleri(Field Values)** belirlenmiş ve ilgili **özelliklere(Property)** atanmıştır.

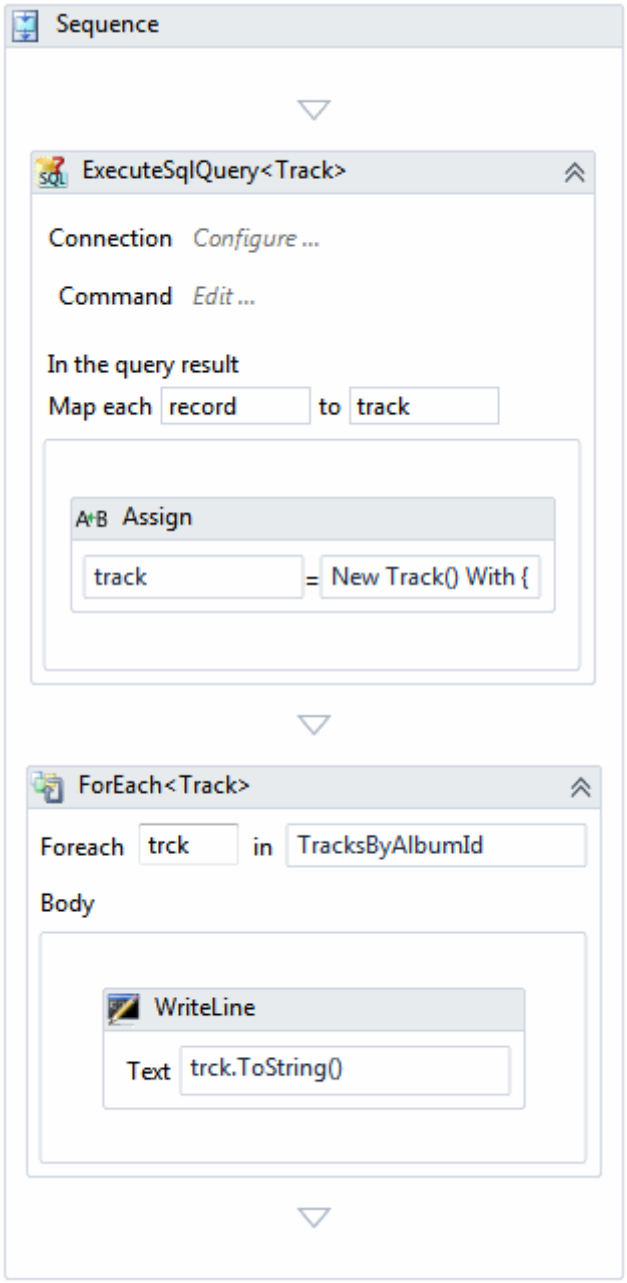
*Not: örneği geliştirirken **C1020: Build error occurred in the XAML MSBuild task: ''xml:space' is a duplicate attribute name** içerikli bir derleme zamanı hatası ile karşılaştım. Bunun üzerine aşağıdaki **XAML** içeriğinde yer alan **xml:space="preserve"** kısmını kaldırarak hatanın önüne geçtim. Söz konusu hatanın sebebinin araştırmaya devam ediyorum.*

<InArgument x:TypeArguments="local:Track" **xml:space="preserve"**>[New Track()
With {
.TrackId = record.GetInt32(0),
.Name = record.GetString(1),
.Composer = record.GetString(2),
.Milliseconds = record.GetInt32(3)
}]</InArgument>

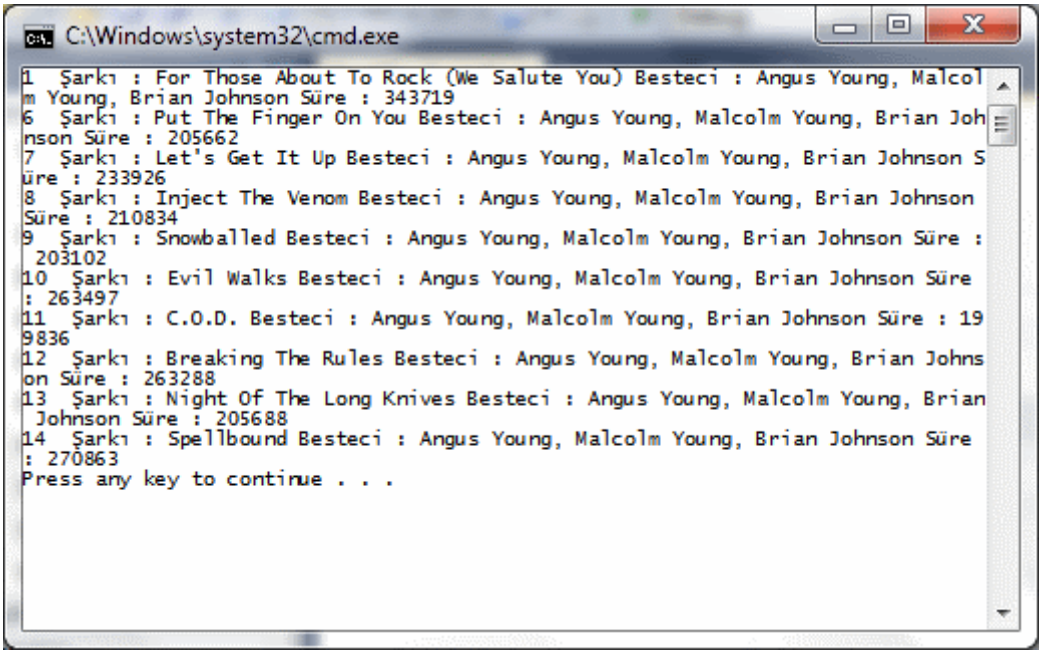
ExecuteSqlQuery<T> kontrolünün yürüttüğü sorgu sonuçlarını aslında **List<T>** tipinden bir koleksiyonda toplanmaktadır. Dolayısıyla sorgu sonuçları örneğimize göre**List<Track>** tipinden bir koleksiyon içerisine de aktarılmaktadır. Bu sorgu sonucu da **List<Track>** tipinden bir **Variable** veya **Argument**' a atanarak kullanılabilir. Tabi bunun için **ExecuteSqlQuery<T>**bileşeninin **Result** özelliğinin uygun değişkene atanarak kontrolün dış ortamına sunulması gerekmektedir.



Artık basit bir **ForEach<T>** aktivitesini kullanarak elde edilen veri kümesi üzerinden hareket edebiliriz. Buna göre **Workflow1** örneğinin son hali aşağıdaki şekilde görüldüğü gibi tamamlanabilir.



ForEach<T> aktivite bileşeni de **Track** tipi ile çalışacak şekilde tesis edilmiş ve bu sayede **ExecuteSqlQuery<T>** aktivitesinin ürettiği **List<Track>** tipinden olan koleksiyon üzerinde dolaşmak üzere ayarlanmıştır. Olayı çok basit bir şekilde ele almak için sadece bir **WriteLine** aktivitesi kullanılmış ve varsayılan değeri **1** olan **AlbumId** değerli Track örneklerinin içeriği ekrana yazdırılmıştır. İşte çalışma zamanı sonuçları.



Süper

Görüldüğü üzere **SQL** sorgu cümlelerinin icra edilmesi ve özellikle üretilen veri kümesinin **Workflow** tarafında ele alınması oldukça kolay bir hale getirilmiştir. örneğimizin **Sequence**bileşenine ait **XAML** çıktısı aşağıdaki gibidir.

```
<Sequence sad:XamlDebuggerXmlReader.FileName="D:\Vs 2010\RTM\Workflow
Foundation\HelloAdoNetActivityPack\HelloAdoNetActivityPack\Workflow1.xaml"
sap:VirtualizedContainerService.HintSize="309,637">
  <Sequence.Variables>
    <Variable x:TypeArguments="scg3:List(local:Track)" Name="TracksByAlbumId" />
  </Sequence.Variables>
  <sap:WorkflowViewStateService.ViewState>
    <scg3:Dictionary x:TypeArguments="x:String, x:Object">
      <x:Boolean x:Key="IsExpanded">True</x:Boolean>
    </scg3:Dictionary>
  </sap:WorkflowViewStateService.ViewState>
  <mda:ExecuteSqlQuery x:TypeArguments="local:Track" CommandText="Select
TrackId,Name,Composer,Milliseconds From Track Where AlbumId=@AlbumId"
ConnectionString="Data Source=.;Initial Catalog=Chinook;Integrated
Security=True" sap:VirtualizedContainerService.HintSize="287,267" ProviderName="S
ystem.Data.SqlClient" Result="[TracksByAlbumId]">
    <mda:ExecuteSqlQuery.RecordProcessor>
      <ActivityFunc x:TypeArguments="sd:IDataRecord, local:Track">
        <ActivityFunc.Argument>
          <DelegateInArgument x:TypeArguments="sd:IDataRecord" Name="record"
/>
        </ActivityFunc.Argument>
```

```
        <ActivityFunc.Result>
          <DelegateOutArgument x:TypeArguments="local:Track" Name="track" />
        </ActivityFunc.Result>
        <Assign sap:VirtualizedContainerService.HintSize="252,100">
          <Assign.To>
            <OutArgument x:TypeArguments="local:Track">[track]</OutArgument>
          </Assign.To>
          <Assign.Value>
            <InArgument x:TypeArguments="local:Track">[New Track() With {
.TrackId = record.GetInt32(0), .Name = record.GetString(1), .Composer =
record.GetString(2), .Milliseconds = record.GetInt32(3) }]</InArgument>
          </Assign.Value>
        </Assign>
      </ActivityFunc>
    </mda:ExecuteSqlQuery.RecordProcessor>
    <InArgument x:TypeArguments="x:Int32" x:Key="AlbumId">1</InArgument>
  </mda:ExecuteSqlQuery>
  <ForEach x:TypeArguments="local:Track" DisplayName="ForEach&lt;Track&gt;"
sap:VirtualizedContainerService.HintSize="287,206" Values="[TracksByAlbumId]">
    <ActivityAction x:TypeArguments="local:Track">
      <ActivityAction.Argument>
        <DelegateInArgument x:TypeArguments="local:Track" Name="trck" />
      </ActivityAction.Argument>
      <WriteLine sap:VirtualizedContainerService.HintSize="257,100"
Text="[trck.ToString()]" />
    </ActivityAction>
  </ForEach>
 </Sequence>
```

özellikle **ExecuteSqlQuery<T>** tipinin **bold** olarak işaretlenmiş nitelikleri ve değerleri bizim için dikkat edilmesi gereken noktalar arasında yer almaktadır. Bundan sonraki kısımda size düşenleri ise şu şekilde özetleyebiliriz.

- Aynı örneği bir **Stored Procedure** kullanacak ve parametre değerlerini çalışma zamanında(Runtime) kullanıcıdan alacak hale getirmeyi deneyebilirsiniz. Ayrıca sorgu sonuçlarını**Workflow** dışına çıkartmak için gerekli hamleleri yapmayı deneyebilirsiniz(Argument kullanımı)
- Bu yazımızda değinmediğimiz **ExecuteSqlNonQuery** Activity bileşeninin ne amaçla kullanıldığını incelemeye çalışabilirsiniz.
- **ExecuteSqlQuery<T>** bileşeninin **CommandType** özelliğinden yararlaran doğrudan Tablo adını belirtmek suretiyle ilgili sonuç kümesini değerlendirmeye çalışabilirsiniz.

Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HelloAdoNetActivityPack.rar (73,63 kb)

Entity Framework - POCO(Plain Old CLR Objects) (2010-05-01T00:01:00)

ado.net entity framework,ado.net entity framework 4.0,

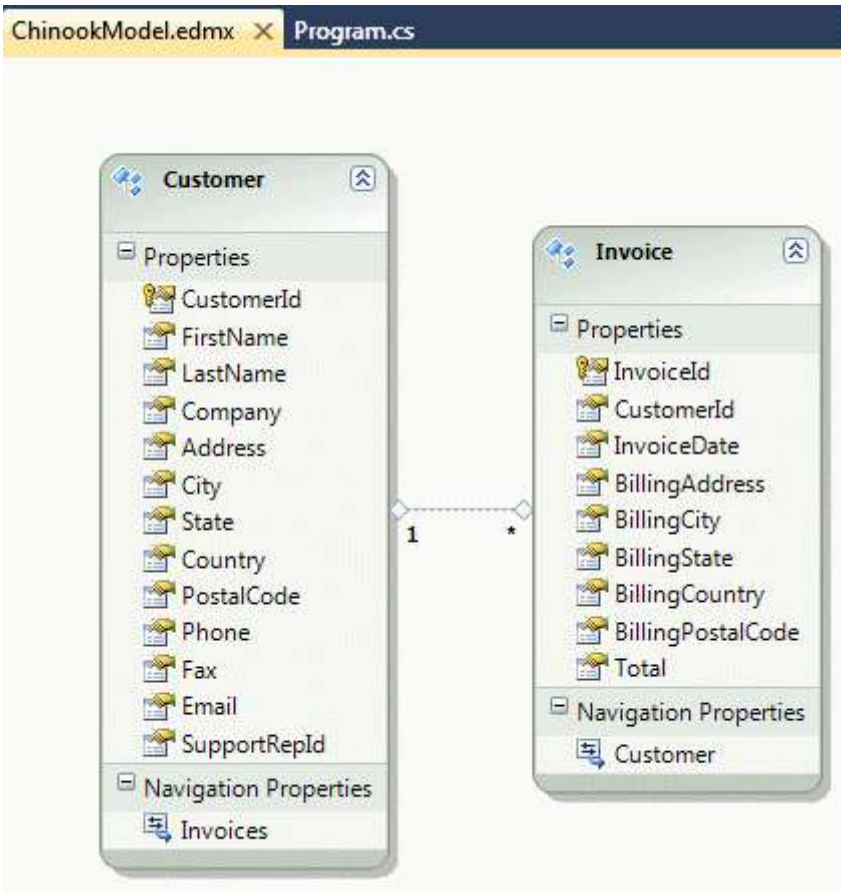
 Merhaba Arkadaşlar,

Yandaki resimde **Seychelles Blue Pigeon** olarak adlandırılan ve **Hint Okyanusundaki 115** adadan birisi olan **Republic of Seychelles** kolonisine has bir güvercin resmi yer almaktadır. Aslında bu kuş oldukça meşhurdur. Nitekim çeşitli hayvan türlerini genellikle kitap kapaklarında kullanan **O'Reilly** yayınlarının uzun zaman önce çıkarttığı ve **Julia Lerman** tarafından yazılmış olan Programming Entity Framework baskısında bu kuşa yer verilmiştir. Kitaptan bahsetmişken...Bendeki baskısı **Ado.Net Entity Framework 3.5** sürümünü içermekteydi. Doğal olarak köprünün altından çok sular geçti ve artık **4.0** sürümü ile karşı karşıyayız. Kitabın **Aralık 2009** baskısında **4.0** vesiyonu içinde ek bilgiler yer almakta. Ancak sanıyorum ki yakın zamanda son sürümüne kavuşacak olan **.Net Framework 4.0** ile birlikte yeni bir baskısı daha çıkacaktır.

Aslında **Ado.Net Entity Framework 4.0** tarafında yer alan önemli kavramlardan birisi de **POCO** nesnelerinin kullanımı. **Ado.Net Entity Framework 3.5(Service Pack 1)** sürümünde net bir şekilde ele alınmayan **POCO** nesneleri için, **4.0** versiyonunda tam destek söz konusu. **POCO(Plain Old CLR Objects), .Net Framework** üzerinde herhangibir bağımlılığı olmadan tanımlanabilen nesneler olarak düşünülebilir. **POCO** nesneleri herhangibir sınıftan türemez*(Class Inheritance)*, çeşitli arayüzleri uygulamaz*(Interface Implementation)*veya özel nitelikler*(Attributes)* ile işaretlenmezler. Sadece verinin üzerlerinden akmasını sağlayan **hafif siklet(Lightweight)** nesnelerdir. Oysaki **Entity Framework** tarafında üretilen tipler düşünüldüğünde, türetmelerin, nitelik işaretlemelerinin ve **IPOCO** olarak adlandırabileceğimiz bazı arayüz uyarlamalarının yapıldığı görülür. Bu nedenle **Entity**nesnelerinin, **Persistence** kıstasını göz ardı etmesi veya test senaryoları düşünülerek **Entity Framework** yapısı içerisinde ele alınmaları zorlaşmaktadır. Dolayısıyla **Entity Framework** tarafına getirilen **POCO** desteğinin önemi büyüktür.

Visual Studio 2010 ortamında geliştirilen **Ado.Net Entity Framework** bazlı uygulamalarda **POCO** nesnelerinin oluşturulması ve kullanımı son derece kolaydır. İşte bu

yazımızda **POCO**nesnelerinin ne işe yaradığını, nasıl tanımlandıklarını ve kullanıldıklarını incelemeye çalışıyor olacağız. İşe başlamadan önce **POCO'** suz bir hayatın nasıl olacağına bakmamızda yarar olduğu kanısındayım. Bu amaçla örnek
bir **Console** uygulamasını **Visual Studio 2010 Ultimate RC** sürümü üzerinden geliştirerek devam edebiliriz. örneğimizde **Chinook** veritabanını baz alan ve aşağıdaki **Entity Model** diagramında görülen tiplere sahip olduğumuzu düşünelim.



çok basit anlamda **Customer** ve **Invoice** tablolarına karşılık gelen **Entitiy** tipleri söz konusudur. Bu tipler arasında **bire çok(one-to-many)** ilişki mevcuttur. **Entity Model** tarafında üretilen sınıf kodlarına baktığımızda aşağıdaki içeriklerin oluşturulduğunu görürüz.

Customer sınıfı;

```
[EdmEntityTypeAttribute(NamespaceName="ChinookModel", Name="Customer")]
[Serializable()]
[DataContractAttribute(IsReference=true)]
public partial class Customer : EntityObject
{
```

Invoice sınıfı;

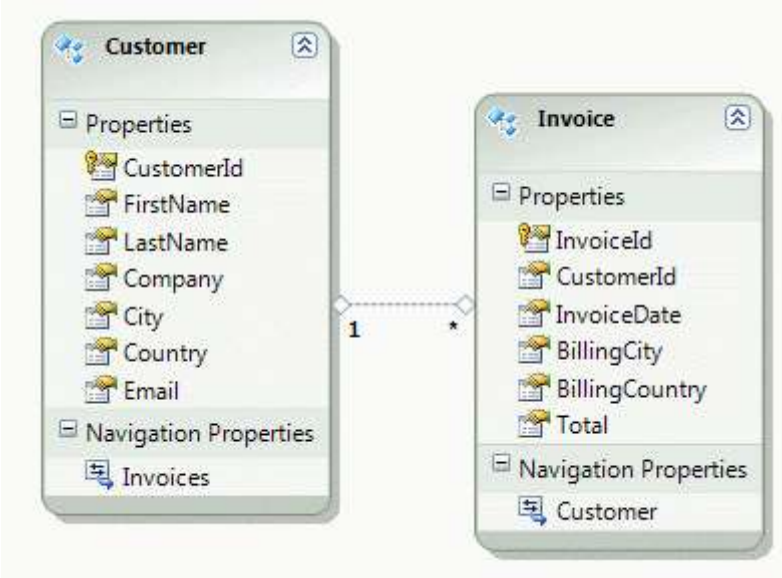
```
[EdmEntityTypeAttribute(NamespaceName="ChinookModel", Name="Invoice")]
[Serializable()]
[DataContractAttribute(IsReference=true)]
public partial class Invoice : EntityObject
{
```

Aslında her iki **Entity** sınıfı içinde dikkat edilmesi gereken nokta, **EntityObject** tipinden türemeleri ve bazı **nitelikler(Attribute)** tarafından imzalanmış olmalarıdır. üstelik sınıflar içerisinde yer alan **özelliklere(Properties)** de bazı **niteliklerin(Attribute)** uygulandığı görülür. örneğin;

```
[EdmScalarPropertyAttribute(EntityKeyProperty=true, IsNullable=false)]
[DataMemberAttribute()]
public global::System.Int32 CustomerId
{
    get
    {
        return _CustomerId;
    }
    set
    {
        if (_CustomerId != value)
        {
            OnCustomerIdChanging(value);
            ReportPropertyChanging("CustomerId");
            _CustomerId = StructuralObject.SetValidValue(value);
            ReportPropertyChanged("CustomerId");
            OnCustomerIdChanged();
        }
    }
}
private global::System.Int32 _CustomerId;
```

gibi.

Bu noktada söz konusu tiplerin bazı bağımlıklar taşıdığını düşünebiliriz.
Ancak **POCO** nesnelerinde, yazımızın başında da belirttiğimiz üzere bu tip bağımlılıklar söz konusu değildir. *(Buna göre POCO nesneleri için şu tip bir tanımlama da yapılabilir; buradaki Entity tiplerinde yer alan bağımlılıklara ihtiyaç duymayan tiplere olan ihtiyaçlarda göz önüne alınan tiplerdir)* Şimdi örneğimizde ilerlemek için, her iki tipten sadece belirli özellikleri projemizde kullanmak istediğimizi düşünelim. Gerçekten de kod tarafında söz konusu **Entity** tipleri içerisinde yer alan tüm özelliklere ihtiyacımız olmayabilir. Bu amaçla sembolik olarak, Entity tiplerinden bazı özellikleri*(**Nullable** değeri **true** olanlar seçilirse iyi olur)*silip diagramı aşağıdaki şekilde görülen hale getirelim.

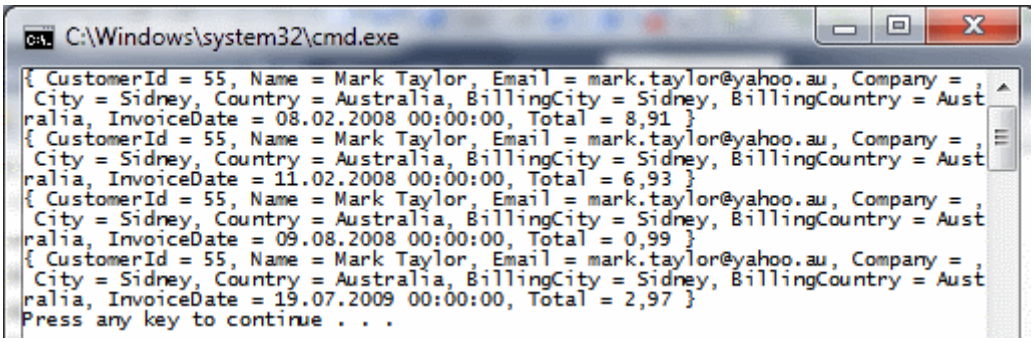


Senaryomuz gereği sadece bu özellikler ile ilgilendiğimiz bir vakamız olduğunu düşünüyoruz. Buna göre **Console** uygulamamızda çok basit bir **LINQ** sorgusu yazarak Entity tiplerin kullanılabilir olduğundan emin olmamızda yarar vardır. İşte örnek kod parçamız ve çalışma zamanı çıktısı...

```
using System;
using System.Linq;

namespace POCODans
{
   class Program
   {
      static void Main(string[] args)
      {
         using (ChinookEntities entities = new ChinookEntities())
         {
            var results = from c in entities.Customers
                          join i in entities.Invoices on c.CustomerId equals i.CustomerId
                          where c.City=="Sidney"
                          select new
                          {
                             c.CustomerId,
                             Name=c.FirstName+" "+c.LastName,
                             c.Email,
                             c.Company,
                             c.City,
                             c.Country,
                             i.BillingCity,
                             i.BillingCountry,
                             i.InvoiceDate,
```

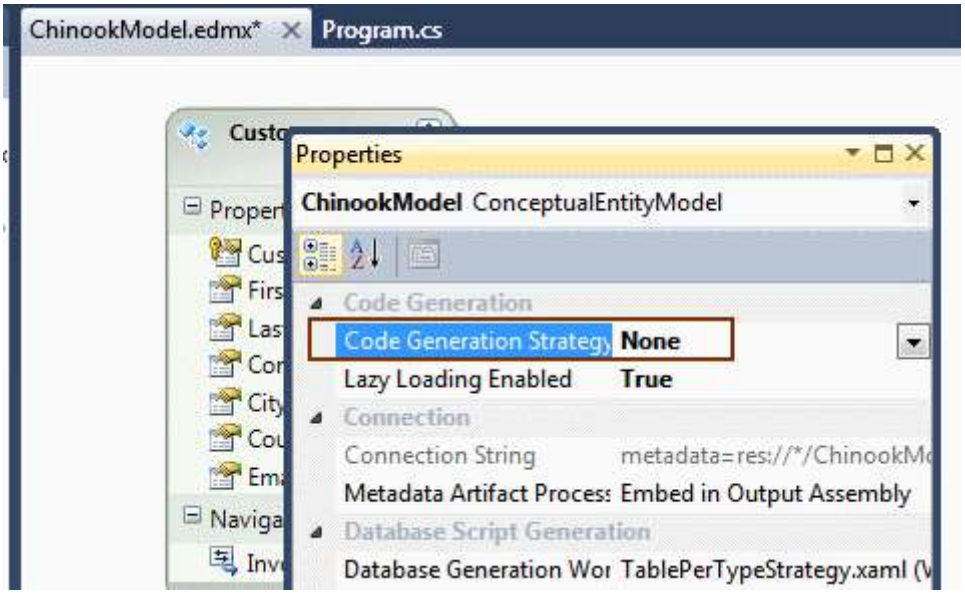
```
                        i.Total
                    };

            foreach (var r in results)
            {
                Console.WriteLine(r.ToString());
            }
        }
    }
  }
}
```

örnek **LINQ** sorgusuna göre **Customer** ve **Invoice** verileri **CustomerId** alanı üzerinden birleştirilerek, Sidney' de yaşayan müşteri ve fatura bilgileri **isimsiz tip(Anonymous Type)** içerisinde toplamakta ve ekrana yazdırmaktadır. Modelimizi test ettiğimizde aşağıdaki sonuçları aldığımızı dolayısıyla çalıştığını görürüz.

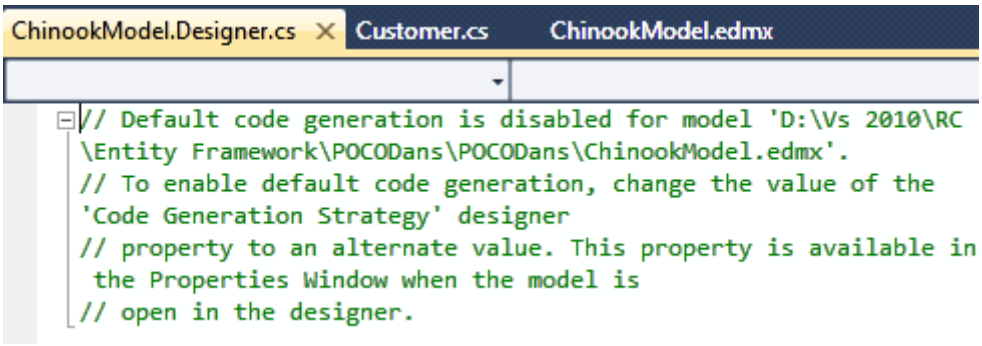


Buraya kadar yaptıklarımızı özetlediğimizde, aslında **Wizard** yardımıyla bir modelin oluşturulduğunu fark edebiliriz. Bu otomatik süreç sonrasında **Storage, Conceptual** ve **Mapping**modellerinin üretildiğini ve bunların **edmx** dosyası içerisine aktarıldığını da anlayabiliriz. Peki bu otomatik üretim hattı göz önüne alındığında, herhangibir bağımlılığı bulunmayan, dolayısıyla otomatik olarak üretilmemesi gereken ve sadece veri aktarımı amacıyla kullanılacak olan bir nesnenin(POCO) entegrasyonunu nasıl gerçekleştirebiliriz?

öncelikli olarak otomatik kod üretimi seçeneğini pasifleştirmeliyiz. Bu amaçla aşağıdaki şekilden de görüleceği üzere, **EDMX'** in özelliklerinden **Code Generation Strategy** değerini**None** olarak set etmeliyiz.



Bu işlemin sonuçlarını aslında hemen görebiliriz. Uygulamanın build edilmesinin ardından **ChinookModel.Designer.cs** içeriğine baktığımızda aşağıdaki ekran görüntüsünde yer alan çıktı ile karşılaşırız. Dikkat edileceği üzere normalde var olması gereken tiplerin hiç birisi mevcut değildir.



Şimdi işin en önemli kısımlarından birisine geldik. **Entity Model** üzerinde görülen **Customer** ve **Invoice** tiplerinin **POCO** versiyonlarını eklemek. Bunun için yapacağımız tek şey birer sınıf oluşturup içerisine gerekli özellikleri koymak olacaktır. Bunu zaten yapmamız gerekmektedir nitekim otomatik üretimi kapattığımız için elimizde artık Customer ve Invoice isimli Entity tipleri de bulunamamaktadır. 🙂 Bu amaçla uygulamamıza aşağıdaki sınıfları eklediğimizi düşünelim.

```
using System;
using System.Collections.Generic;

// Namespace adının ChinookModel olması önemlidir.
namespace ChinookModel
{
   public class Customer
   {
      public int CustomerId { get; set; }
```

```
        public string FirstName { get; set; }
        public string LastName { get; set; }
        public string Company { get; set; }
        public string City { get; set; }
        public string Country { get; set; }
        public string Email { get; set; }
        private List<Invoice> _invoices = new List<Invoice>();// Bir Customer' ın birden
fazla faturası olabilir

        public List<Invoice> Invoices
        {
            get { return _invoices; }
            set { _invoices = value; }
        }
    }

    public class Invoice
    {
        public int InvoiceId { get; set; }
        public int CustomerId { get; set; }
        public DateTime InvoiceDate{ get; set; }
        public string BillingCity { get; set; }
        public string BillingCountry { get; set; }
        public decimal Total { get; set; }
        public Customer Customer { get; set; } // Birden fazla fatura tek bir Customer ile
ilişkilidir.
    }
}
```

Bir anlamda otomatik olarak üretilen **Entitiy** sınıflarını yazdığımızı düşünebiliriz ancak çok önemli bir fark bulunmaktadır. **Bağımlılıklar**. Hiç bir sınıf türetmesi, arayüz uygulaması veya nitelik işaretlemesi mevcut değildir dikkat edeceğiniz üzere.
Elbette **Customer** ve **Invoice** sınıflarının yazılmış olmaları yeterli değildir. Birde bu tiplerin çalışma zamanındaki yönetimini üstelenebilecek bir **içerik(Context)** tipi olmalıdır. Bu amaçla uygulamaya aşağıdaki sınıfı eklememiz yeterli olacaktır.

```
using System;
using System.Collections.Generic;
using System.Data.Objects;

namespace ChinookModel
{
    public class ChinookEntities
        :ObjectContext
    {
```

```
        private ObjectSet<Customer> _customers;
        private ObjectSet<Invoice> _invoices;

        public ChinookEntities()
            :base("name=ChinookEntities")
        {
        }

        public ObjectSet<Customer> Customers
        {
            get
            {
                if (_customers == null)
                {
                    _customers = base.CreateObjectSet<Customer>();
                }
                return _customers;
            }
        }

        public ObjectSet<Invoice> Invoices
        {
            get
            {
                if (_invoices == null)
                {
                    _invoices = base.CreateObjectSet<Invoice>();
                }
                return _invoices;
            }
        }
    }
}
```

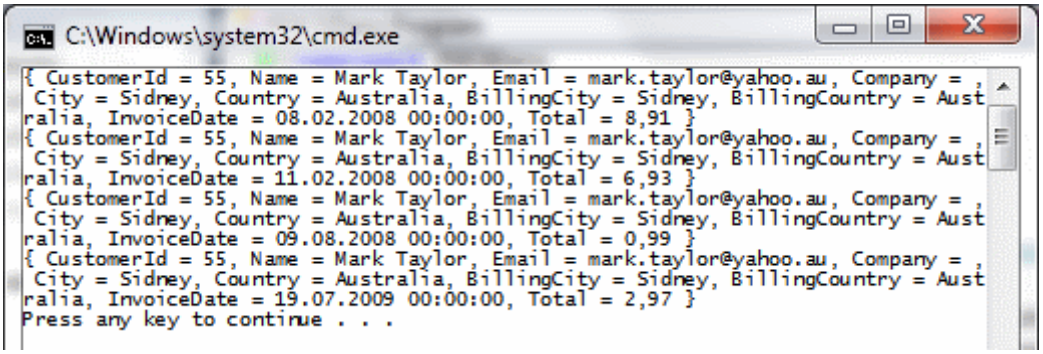
ObjectContext tipinden türeyen **ChinookEntities** sınıfı içerisinde **Customer** ve **Invoice** sınıflarını kullanan **ObjectSet** tipli özellikler yer almaktadır. Dikkat edilmesi gereken noktalardan birisi de **yapıcı metodun(Constructor) base** çağrısı yardımıyla **ObjectContext** tipinin yapıcısına **App.config** dosyasında yer alan **Connection String** bilgisini gönderiyor olmasıdır.

ChinookModel.edmx | App.Config | POCOTypes.cs | Program.cs

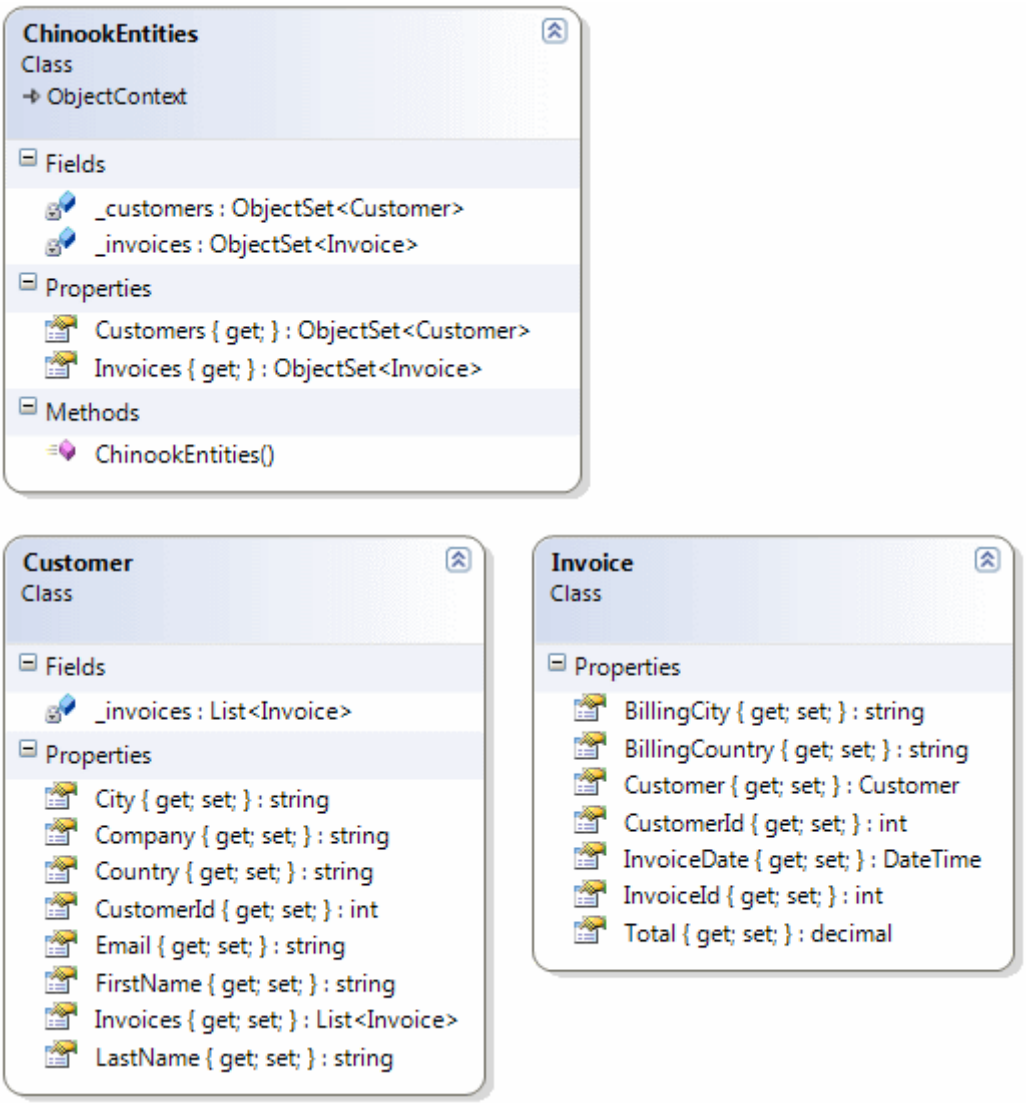
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <connectionStrings>
    <add name="ChinookEntities" connectionString="metadata=res://*/
ChinookModel.csdl|res://*/ChinookModel.ssdl|res://*/
ChinookModel.msl;provider=System.Data.SqlClient;provider connection
string=&quot;Data Source=.;Initial Catalog=Chinook;Integrated
Security=True;MultipleActiveResultSets=True&quot;"
providerName="System.Data.EntityClient" />
  </connectionStrings>
</configuration>
```

ChinookEntities isimli sınıfı **ChinookModel isim alanı(Namespace)** içerisinde tasarladığımızdan, **Main** metodunda gerekli bildirimi yapmamız, biraz önceki kodun değiştirilmeden çalışması için yeterli olacaktır.

```
using System;
using System.Linq;
using ChinookModel;

namespace POCODans
{
   class Program
   {
      static void Main(string[] args)
      {
         using (ChinookEntities entities = new ChinookEntities())
         {
            ...
```



Volaaaa!!! Dikkat edecek olursanız var olan kodu bozmadan ama bu kez POCO nesnelerinden yararlanarak çalıştırmayı başardık. Bu işlemlerin ardından uygulamamızın tip yapısını **Class Diagram** üzerinden incelediğimizde, aşağıdaki gibi bir oluşumun söz konusu olduğunu görebiliriz.



Bu yazımızda **POCO(Plain Old CLR Objects)** nesnelerinin ne olduğunu kısaca tanımaya çalışırken, çok basit bir örnek geliştirerek yolumuza devam ettik. Diğer yandan **POCO**nesnelerinin, **Entity Framework'** ün otomatik üretim aracı tarafından oluşturulan **Entitiy** tipleri ile olan farklarını anlamaya çalıştık. Artık **Ado.Net Entity Framework** tabanlı projelerimizde **POCO** nesnelerini kullanma ihtiyacını duyduğumuzda, nasıl hareket etmemiz gerektiğini az çok öğrenmiş olduğumuzu düşünüyorum.
Tabi **POCO** nesnelerini kullanmanın getirdiği bazı dezavantajlar da yok değil. Herşeyden önce **EntityObject** içerisindeki elemanların türetme olmayışı nedeniyle kullanılamayışı, veritabanı tablolarındaki alan adları ile bir mapping işleminin yapılamayışı başlıca dezavantajlar olarak sayılabilir.

POCO ile ilişkili yeni bilgiler öğrendikçe sizlerle paylaşıyor olacağım. özellikle bu yazıda henüz değerlendirmediğimiz bir durum var o da **Lazy Loading** durumlarında **POCO** nesnelerinin nasıl hazırlanması gerektiği? Bunu bir sonraki yazımızda aynı örnek üzerinden test ederek incelemeye çalışacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

POCO_RTM.rar (44,97 kb) **[örnek Visual Studio 2010 Ultimate RTM sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Workflow Foundation Öğreniyorum - Ders 1 - Biraz Daha Bileşen (2010-04-27T09:29:00)

workflow foundation 4.0,workflow foundation,visual studio 2010,

Merhaba Arkadaşlar,

"Workflow Foundation 4.0 öğreniyorum" serimizin ikinci dersi*(Ders 1 sizi aldatmasın, hatırlarsanız Ders 0 diyerek başladık)* ile karşınızdayız. Bir önceki görsel dersimizde **Hello World** demek için sadece **WriteLine** aktivite bileşeninden yararlanmıştık.
Ancak **Workflow Foundation 4.0**, pek çok aktivite bileşeni içermektedir ve bunları zaman içerisinde öğrenmemiz çok önemlidir. İşte bu dersimizde **Sequence, Assign, ForEach<T>, If** gibi aktivite bileşenlerini işin içerisine katıyor ve **Workflow Foundation** konusunda ilerlemeye devam ediyoruz. Geliştireceğimiz örnekte, bir klasör içerisinde yer alan dosyalardan belirli bir harf ile başlayanların*(örneğin A harfi)* ekrana yazdırılmasını sağlayan bir akış oluşturuyoruz. Bu sırada **Visual Studio 2010**ortamının sunduğu **WPF** tabanlı **Workflow Designer** ortamından yararlanıyor ve **Variable** kavramını da değerlendiriyoruz. İyi seyirler dilerim.

Ders 1 - Biraz Daha Bileşen

Süre : 09:56

Dosya Boyutu : 12.1 Mb

örnek Lesson1_BirazDahaBilesen.rar (36,13 kb)

Mp4 Formatında İndirmek İçin Tıklayın

Microsoft Yazılım Geliştiriciler Teknoloji Günleri Ankara' da C# 4.0 Anlattım (2010-04-26T11:45:00)

seminer,microsoft,mvp,c# 4.0,

Merhaba Arkadaşlar,

Hatırlayacağınız üzere bu gün **Microsoft Ankara**' da düzenlenen Yazılım Geliştiriciler Teknoloji Günleri' nde **C# 4.0 ile Gelen Yenilikler** konulu bir oturumum vardı. Sunumu az önce tamamladım. Katılan tüm arkadaşlarımıza ilgileri için, **Microsoft**'a başarılı organizasyonları için**(özellikle Buket Şerefli' ye)** çok teşekkür ederim. Oturumun benim için en keyifli anlarından birisi de **Eskişehir**' den katılan arkadaşlarımızın **Meşhur İnanMet Helvası** hediye etmesiydi. Değerli konuşmacı arkadaşlarımın söylediğine göre hediyenin verilişi sırasında biraz afallayıp kalmışım. 😲 çok doğal, çünkü ilk kez başıma böyle bir şey geliyor.

Şu anda **Muammer Benzeş(Windows üzerinde PHP), Ekin özçiçekçiler(Windows Mobile 6.5 ile Web Widget Geliştirme), Cengiz Han(Visual Studio 2010 ve Takım Geliştirme özellikleri), Erbuğ Kaya(Expression Studio' nun Tasarımcılara Sunduğu Artılar) ve Barış Atalay(SQL Server 2008 ile Yazılım Geliştirme)** arkadaşlarımızda sunumları ile etkinliğe renk katmaya devam etmekteler. Etkinlikte aktardığım sunum dosyası ve örnekleri aşağıda bulabilirsiniz.

C# 4.0 - New Features.pptx (472,86 kb)

C# 4.0.rar (1,21 mb)

NedirTv?com 4ncü Yıl Seminer Videoları (2010-04-22T10:15:00)

seminer,nedirtv?com,wcf eco system,webhttp services,data services,ria services,workflow services,windows server appfabric,dublin,velocity,

Merhaba Arkadaşlar,

Hatırlayacağınız üzere **10 Nisan 2010 Cumartesi** günü **NedirTv?com** topluluğunun **4ncü kuruluş yıldönümünü** değerli konuşmacı arkadaşlarımız ve katılımcılarımız ile birlikte kutladık. Bende **WCF Eco System** ve **Windows Server AppFabric** sunumlarımla etklikte

yer alarak bilgilerimi paylaşmaya fırsatını buldum. Seminerler sırasında adet olunduğu üzere **Video** kayıtlarımızda alındı. *(Her ne kadar dikkatsizlik edip **font** ayarlarını büyütmeyi unutmuş olsam da)* Açıkçası bu işte emeği geçen tüm arkadaşlarımıza teşekkür etmeliyiz. Geçtiğimiz günlerde videoların **ftp** üzerine de yüklendiğini öğrendim. Seminerleri kaçıran arkadaşlarımız en azından seminer kayıtlarına bakarak anlatılanları dinleyebilir ve yeni gelen özellikler hakkında fikir sahibi olabilirler. Keyifli seyirler dilerim.

Windows Server AppFabric Seminer Videosu (Süre -> 00:38:31 Dosya Boyutu -> 325 Mb)

WCF Eco System Seminer Videosu (Süre -> 01:27:24 Dosya Boyutu -> 724 Mb)

Workflow Foundation Öğreniyorum Başladı - Ders 0 - Hello World (2010-04-20T09:55:00)

workflow foundation 4.0,workflow foundation,visual studio 2010,

Merhaba Arkadaşlar,

Bundan bir kaç yıl(tam olarak **16 Mart 2004**) önce **Java** programlama dilini öğrenmeye çalışırken, bunu bir yazı dizisi ile eş zamanlı hale getirmeye çalışmış ve editörlüğünü yaptığım C#Nedir? topluluğunda yayınlamaya başlamıştım. "Java ile 24 Kahve Molası" isimli bu seride yer alan her makalede, başımdan geçenleri anlatıyor ve tecrübelerimi paylaşıyordum.

Artık buna benzer bir seriye daha başlamanın zamanı geldi. Bu kez, NedirTv?com sponsorluğunda ve kurucusu Uğur Umutluoğlu' nun **(MVP - Asp.Net)**desteğinde, kısa görsel derslerden oluşan yeni bir seri ile karşınızdayız. İlk serimizin adı **"Workflow Foundation 4.0 öğreniyorum"**. Bu da şu anlama geliyor. Başka serilerde olacak

Workflow Foundation öğreniyorum isimli serideki amacımız, **Workflow Foundation** konusunu **başlangıç seviyesinden, orta seviyeye kadar** incelemek ve öğrenmek. Bu amaçla, haftada bir yayınlanacak görsel derslerimiz ile hedefe ulaşmaya çalışıyor olacağız. Hedefimiz 24 derse ulaşabilmek ancak en az 18 dersin olacağını

şimdiden ifade edebilirim. Tabi ki ilk bölüm çok basit bir Hello World uygulamasını içeriyor olacak. Ancak ilerleyen zamanlarda daha da derinlere inmeye başlıyor olacağız*(İçeriğimizde çok sıkı konuların yer aldığını şimdiden ifade etmek isterim)*. **Workflow Foundation** tarafında uygulama geliştirmeye başlamak isteyenler için faydalı bir çalışma olacağını ümit ediyorum.

Elbette çekimler sırasında dilimizin sürçtüğü noktalar olabilir. **Her ne kadar sürçü lisan ettiysek affola** diyerek sizleri ilk görsel dersimizle baş başa bırakıyorum.

Ders 0 - Hello World

Süre : 09:55

Dosya Boyutu : 12.1 Mb

örnek Lesson0.rar (34,74 kb)

Microsoft Yazılım Geliştiriciler Teknoloji Günleri Ankara' da C# 4.0 Anlatıyorum (2010-04-16T22:30:00)

microsoft biz spark,c# 4.0,ankara teknoloji günleri,gelişim atölyesi,microsoft,

Microsoft Yazılım Geliştiriciler Teknoloji Günleri Ankara'da!

26 Nisan 2010 tarihinde **Microsoft Türkiye Ankara** ofisinde gerçekleştirilecek olan **Microsoft Yazılım Geliştiriciler Teknoloji Günlerinde, C# 4.0** ile birlikte gelen yenilikleri anlatıyor olacağım.

Benimle birlikte pek çok değerli arkadaşımın konuşma olarak katılacağı etkinlikte, **Windows Mobile 6.5** ile **Web Widget Geliştirme, Expression Studio, Visual Studio 2010 ve Takım Geliştirme özellikleri, SQL Server 2008 R2 ile Yazılım Geliştirme, Windows üzerinde PHP** konularına da değiniliyor olacak.

Kayıt yaptırmak için lütfen tıklayınız

Gündem

09:00 - 09:30 Kayıt
09:30 - 09:45 ISV'lerin Yararlanabilecekleri Program, Kampanya ve Etkinlikler
09:45 - 10:45 C# 4.0 ile Gelen Yenilikler
11:00 - 12:00 Windows Mobile 6.5 ile Web Gadget Geliştirme
12:15 - 13:15 Expression Studio'nun Tasarımcılara Sunduğu Artılar
13:15 - 14:00 öğle Yemeği
14:00 - 15:00 Visual Studio 2010 ve Takım Geliştirme özellikleri
15:15 - 16:15 SQL Server 2008 R2 ile Yazılım Geliştirme
16:30 - 17:30 Windows üzerinde PHP

Tarih : 26 Nisan 2010 Pazartesi Saat 09:00 - 17:30 **Yer :** Microsoft Ankara Ofisi

RTM Yayınlandı (2010-04-13T23:29:00)

visual studio,visual studio 2010,visual studio 2010 ultimate,rtm,

Merhaba Arkadaşlar,

2008 yılının son çeyreğiydi. **Out Source** olarak çalıştığım bankada **.Net 2.0** tabanlı olarak geliştirilen **WinForms** uygulamasının yabancı sistemler ile olan entegrasyonu üzerinde çalışıyordum. Bir yandan da **.Net Framework 4.0** ile ilişkili ilk bilgileri tedarik etmeye çalışıyordum. **WCF 4.0** ve **WF 4.0** tarafında gelecek olan yeniliklerin ne olacağını çok merak ediyorum. Tabi ki ortada çok fazla kaynak yoktu.

Bir gün iş arkadaşlarımla birlikte yemekteyken telefonum çaldı ve o zaman ki **MVP Turkey Lead**' imiz **Mehmet Emre** benden, **Microsoft Yazılım Zirvesinde WCF & WF 4.0** anlatıp anlatamayacağımı sordum. Büyük bir onur ve sevinçle kabul ettim. Telefonu kapattığımdaysa biraz panik yaşadığımı itiraf ediyorum. çünkü çok az kaynak vardır. çok şükür ki yeniliklerin ilk haberleri veya planlananlar **Microsoft PDC 2008**' de tanıtılmıştı.

O zaman gerçekleştirdiğim sunumda **WCF & WF 4.0** dışında, **Dublin, Oslo, Quadrant** kod adlı konulara değindiğimi de hatırlıyorum. Elimde**Visual Studio**' nun yeni sürümü yoktu ancak **PDC 2008**' de yayınlanan ve **Virtual PC** üzerinden çalıştırılan bir **PreBuild** versiyonu bulunmaktaydı. O sürüm üzerinden denemelerimi yapmış ve sunumuma hazırlanmıştım.

Zaman ilerledikçe **Beta 1** sürümü ile karşılaştık. **WF 4.0, C# 4.0, Visual Studio 2010 IDE**' si ile ve diğer pek çok alanda gelen yenilikler çok heyecan vericiydi. Artık indirilip kurulabilir bir sürüm mevcuttu ve bloğumdaki yazılarım ve görsel eğitim materyallerim için bolca sebep bulmayı başarmıştım.

Derken **Beta 2** sürümü geldi. **Beta 1** ile yaptığım ve henüz blogda yayınlamadığım bazı örneklerin, köklü değişiklikler nedeni ile **Beta 2**' de çalışmadığını gördüğümde hiç üzülmemiştim. Her zaman ki gibi **Microsoft**' un haklı sebeplerden dolayı bu değişiklileri yaptığını ve iyi de ettiğini biliyordum.

Ve sonrasında **RC(Release Candidate)** sürümüne kavuştuk. Parçalar daha da yerine oturmaya başlamıştı.

Nihayet dün itibariyle **RTM** sürümü ile karşı karşıyayız. Dün akşam saat **Türkiye** saati ile **20:00**' da **MSDN** abonelerine de açılan **Visual Studio 2010**' un **Ultimate**sürümünü indirdim ve ilk bisikletini almış bir çocuk heyecanı ile kurulumu gerçekleştirdim. çok yakında **Release** sürüme kavuşacağımızı bildiğimiz **Visual Studio 2010**yeni özellikleri ile gerçekten baştan çıkarıcı bir **Developer** deneyimini vaat ediyor. özellikle mimari taraf için getirdiği yeni **IDE** kabiliyetleri ve yetenekleri harika. Bu deneyimi kaçırırsanız yazık edersiniz...

C# 4.0 - Metod Overloading ve Dynamic Tipler (2010-04-13T13:40:00)

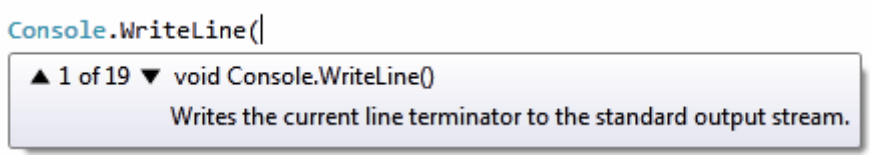
c# 4.0,dynamic,object,method overloading,

Merhaba Arkadaşlar,

Eminim hepimiz çocukluğumuzda en az bir kere olmak üzere yediklerimizi, elimize yüzümüze bulaştırmış ve kirlenmişizdir. Her ne kadar bazı şirketler reklam

kapmanyalarında kirlenmek güzeldir diyerek annelerin yüreğine su serpseler de, bu aslında pek gerçeği yansıtmamaktadır. Nitekim anneler, çocuklarının ellerini yüzlerini yediklerine bulayıp etraftaki eşyalara dokunmalarını pek hoş karşılamayabilirler. Ben şahsen bunu çocukken bir kaç kez tecrübe etmiş bir insanımdır. Yine de, yaşım hayatın yarısına merdiven dayamış olsa da, bazı zamanlarda o çok sevdiğim kayısı marmelatlı ve üstü pudralı olan Berliner tatlısını elime yüzüme(hatta burnuma) bulaştırarak yemeyi severim*(Tabi evde ve en fazla eşimin yanında)* Şimdi diyeceksiniz ki Burak Hoca gene başladı bir yiyecek ile... Yok. Aslında odaklanacağımız nokta herşeyi ele yüze bulaştırmak. Şimdi öyle bir konuya dalacağız ki herşeyi karıştırıp allak bullak edip yüzümüze gözümüze bulaştıracağız. öyleyse gelin hiç vakit kaybetmeden üstümüzü biraz kirletelim

Bu gün kü yazımızda **Dynamic** tiplerin, metodların **aşırı yüklenmesi(overload)** durumunda nasıl bir duruma neden olduklarını incelemeye çalışıyor olacağız. Bildiğiniz üzere bir metodun aynı isme ait birden fazla versiyonu yazılabilmektedir. Bu durum kısaca metodların **aşırı yüklenmesi(Overloading)** olarak adlandırılmaktadır. Metodların aşırı yüklenmesindeki en büyük gayelerden birisi de, aynı amaca hizmet eden ama bunu farklı sayıda veya tipte parametre ile yerine getirebilen fonksiyonların farklı isimler ile yazılmasını engellemek ve böylece anlamsal bütünlüğü korumaktır. **.Net Framework,** ilk versiyonundan itibaren bu özelliği içermektedir. çok eskiden eğitmenlik yaptığım dönemlerde, metodların aşırı yüklenmesi konusu ile ilişkili olarak verdiğim ilk örnek her zaman için **Console** sınıfının **static WriteLine** metodu olmuştur.



Şekilden de görüleceği üzere, **WriteLine** metodunun farklı tipte veya sayıda parametre ile çalışabilen **19 farklı versiyonu** bulunmaktadır. Burada derleme zamanı açısından önem arz eden konulardan birisi de, metodların hangisinin çağırıldığının ayırt edilmesidir. İşte bu noktada **metodun imzası(Signature)** adı verilen kavram devreye girmektedir. Metod imzası, parametre sayısı veya tiplerini kapsamaktadır. Buna göre aynı tipten ama farklı sayıda parametre veya farklı tipten ama aynı sayıda parametre, çoğunlukla geçerlidir. Upsss...çoğunlukla mı? Neden böyle söylediğimi ispat etmek için **C# 4** ile birlikte gelen **dynamic** tiplerin, metodların aşırı yüklenmesi sırasındaki kullanımlarına göz atmamız yeterlidir. öncelikli olarak aşağıdaki kod parçasını göz önüne alalım.

```
namespace DynamicAndOverloading
{
   class Program
   {
      static void Topla(int x) { }
```

```
        static void Topla(int x, dynamic y) { }
        static void Topla(dynamic x, int y) { }

        static void Main(string[] args)
        {
            Topla(4);
            Topla(2,3);
        }
    }
}
```

Topla metodunun 3 farklı versiyonunun yazıldığı görülmektedir. Teorimize göre, tüm metodlar birbirlerinde farklıdır. Nitekim metod imzası kriterleri sağlanmaktadır. İlk Topla metodu tek parametre aldığı için iki parametre alan diğer versiyonları ile otomatik olarak ayrışmaktadır. Diğer yandan iki parametre alan versiyonlarda da, parametrelerin tipleri farklıdır. Farklıdır, nitekim sıraları aynı değildir. Dolayısıyla herhangibir sorun görünmemektedir. Oysaki daha Toplam metodunun iki parametreli versiyonunu yazarken, aşağıdaki ekran görüntüsünde yer alan hata mesajı ile karşılaşılır.

```
    class Program
    {
        static void Topla(int x) { }
        static void Topla(int x, dynamic y) { }
        static void Topla(dynamic x, int y) { }

        static void Main(string[] args)
        {
            Topla(4);
            Topla(2,3);
        }
    }
}
```

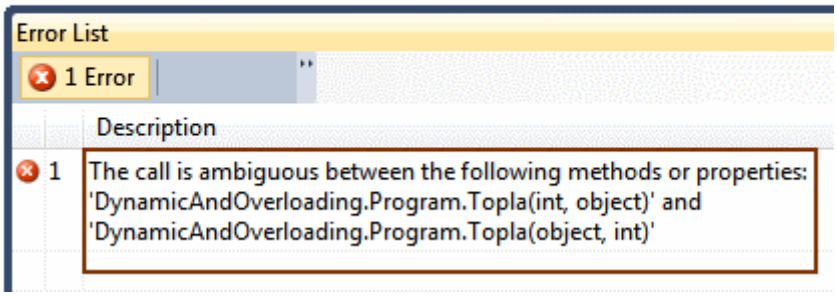
Error List

1 Error | 0 Warnings | 0 Messages

	Description
1	The call is ambiguous between the following methods or properties: 'DynamicAndOverloading.Program.Topla(int, dynamic)' and 'DynamicAndOverloading.Program.Topla(dynamic, int)'

Mesaja göre derleyici, **Topla(int,dynamic)** ile **Topla(dynamic,int)** çağrıları arasında kararsız kalmıştır. Açıkçası ortada tam bir belirsizlik söz konusudur. Peki bu durum size tanıdık geldi mi? Aslında **dynamic** tip yerine **object** tipini kullandığımızda da benzer bir sorunla karşılaşırız. Aynı örnek kod parçasında bu sefer **dynamic** yerine **object** tipini kullandığımızı düşünelim.

```
class Program
{
    static void Topla(int x) { }
    static void Topla(int x, object y) { }
    static void Topla(object x, int y) { }

    static void Main(string[] args)
    {
        Topla(4);
        Topla(2,3);

    }
}
}
```

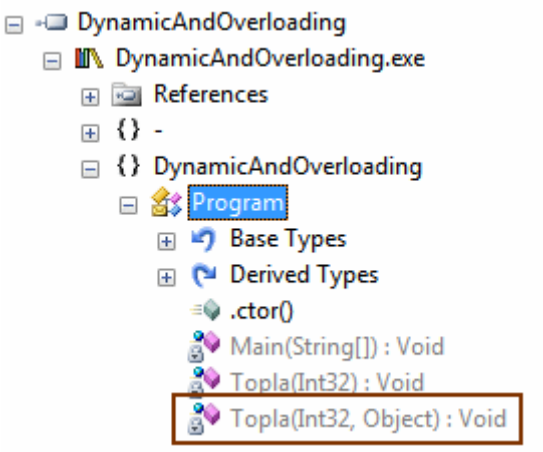


Durum değişmemiştir. Derleyici yine hangi metodu çağıracağı konusunda kararsız kalmış ve hata mesajı üretilmesine sebebiyet vermiştir. Dolayısıyla program çalışmamaktadır.

Bu noktada metodların aşırı yüklenmesinde **object** tipi ile **dynamic** tiplerin benzer bir davranışa neden olduklarını düşünebiliriz. Ancak bunun ispatını da yaparsak ballı kaymaklı tap taze beyaz ekmek yemiş kadar oluruz. Şimdi durumu net bir şekilde ispatlamak adına kodu aşağıdaki gibi değiştirelim.

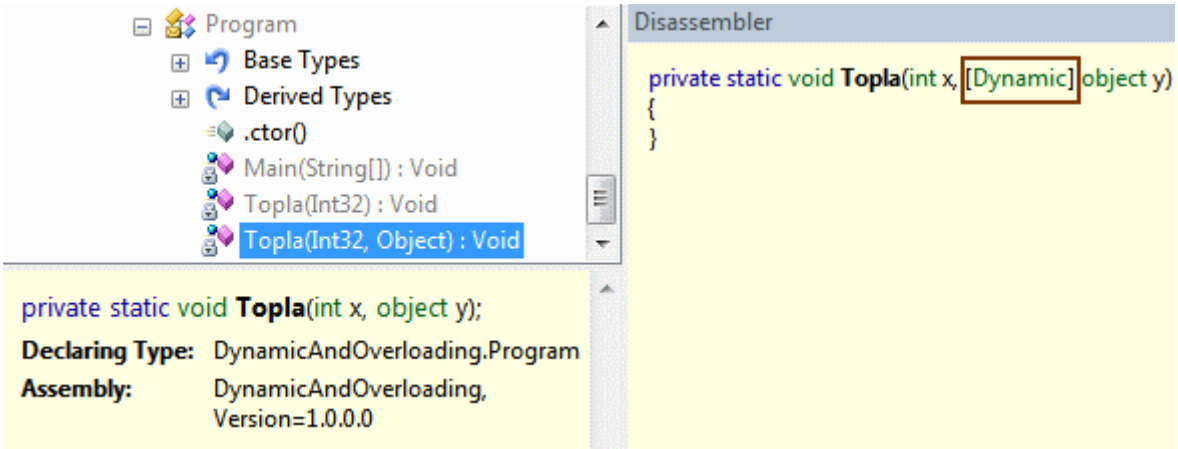
```
namespace DynamicAndOverloading
{
   class Program
   {
      static void Topla(int x) { }
      static void Topla(int x, dynamic y) { }
      //static void Topla(dynamic x, int y) { }  // Yorum satırı yaptık

      static void Main(string[] args)
      {
         Topla(4);
         Topla(2,3);
      }
   }
}
```

Bu durumda program kodu başarılı bir şekilde derlenecektir. Şimdi belki de uzun zamandır yanına uğramadığımız Red Gates .Net Reflector aracını açalım ve program kodumuzun içeriğine bir bakalım. Aşağıdaki ekran görüntüsünde yer alan sonuçlar ile karşılaşırız.



Dikkatinizi çeken bir şey oldu mu? 🙂

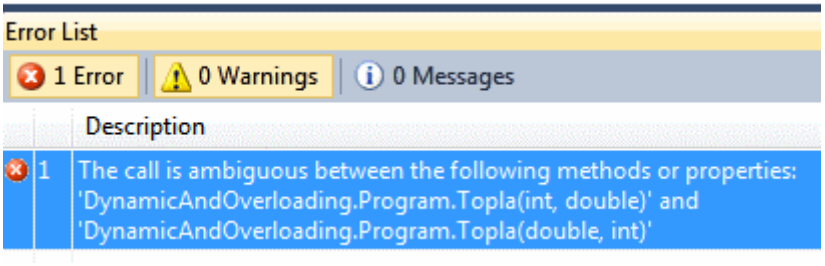
Topla metodunun iki parametre alan versiyonunun **IL(Intermediate Language)** tarafına olan aktarımına göre **dynamic** tip olarak tanımladığımız **y** değişkeni, aslında **Object** tipi olarak değerlendirilmektedir. Bir dakika... Peki çalışma zamanı bunun aslında **dynamic** bir tip olduğunu nasıl anlayacaktır? Bunu **Topla(Int32,Object): Void** metodunun **C#** kodu çıktısına bakarak görebiliriz. İşte **.Net Reflector** çıktısı.



Görüldüğü üzere **object y** değişkeni **[Dynamic]** niteliği ile imzalanmıştır. Dolayısıyla çalışma zamanı tarafından aslında **dynamic** tip olarak yorumlanacaktır.

Buraya kadar her şey netleşmiş gibi düşünülebilir ve hatta yazımızın artık bitmemesi için bir neden olmadığı da düşünülebilir. Ancak yazımızı sonlandırmadan önce, **object** tipinin metodların aşırı yüklenmesinde yol açtığı sorunun, diğer tipler içinde geçerli olabileceğini gösterek ilerlememizde yarar vardır. Söz gelimi aşağıdaki şekilde yer alan kod parçasında da benzer durumun oluştuğu görülebilir.

```
{
    static void Topla(int x) { }
    static void Topla(int x, double y) { }
    static void Topla(double x, int y) { }

    static void Main(string[] args)
    {
        Topla(4);
        Topla(2,3);
```



Dikkat edilecek olursa derleyici, **double** tipinin kullanıldığı **Topla** metodlarından hangisinin kullanılacağı konusunda kararsız kalmaktadır. Yoksa metod imzası kavramı çatlamakta mı? Aslında bir çözüm söz konusudur. O da, doğru değerler ile ilgili metodların çağırılmasıdır. Yani gerçekten **double** tip ile çağrı yapılmasıdır. Bu durumda aşağıdaki kod parçası derleme zamanı hatasına yol açmayacaktır.

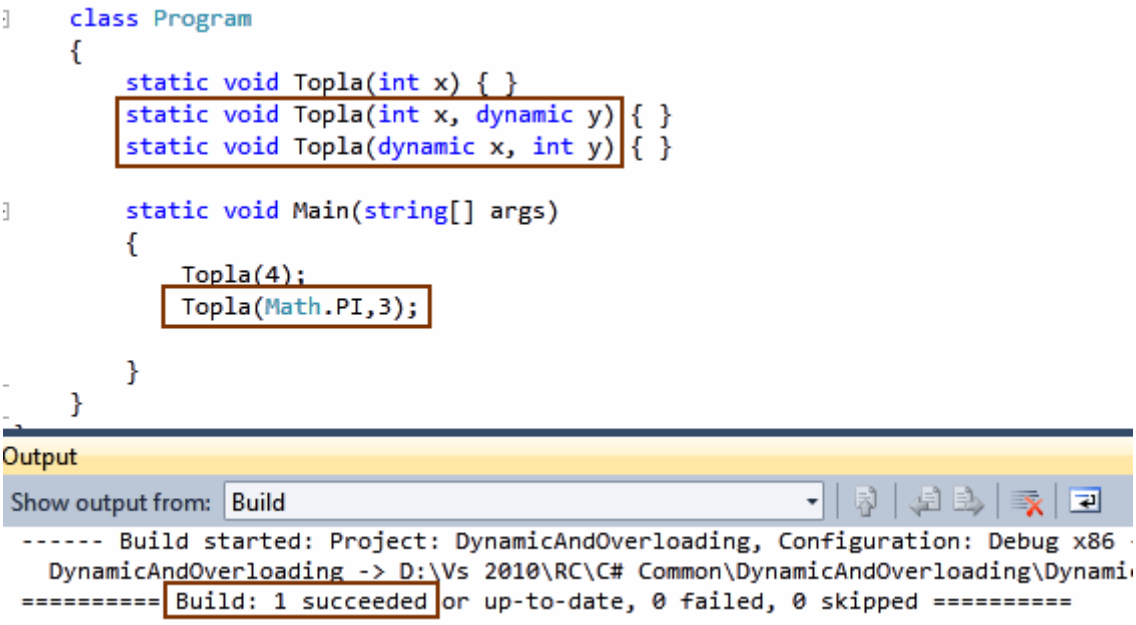
```
namespace DynamicAndOverloading
{
   class Program
   {
      static void Topla(int x) { }
      static void Topla(int x, double y) { }
      static void Topla(double x, int y) { }

      static void Main(string[] args)
      {
         Topla(4);
         Topla(Math.PI,3);
      }
   }
}
```

Derleme hatası olmaması son derece doğaldır. Nitekim **Math.PI** değişkeninin kullanılması, **Topla(double x,int y)** metodunun tespit edilmesini sağlamaktadır. Şimdi olay biraz daha ilginç bir hal almaya başlayacaktır. Bunun için program kodunu aşağıdaki şekilde güncelleyip devam ettiğimizi düşünelim.

```
namespace DynamicAndOverloading
{
   class Program
   {
```

```
        static void Topla(int x) { }
        static void Topla(int x, dynamic y) { }
        static void Topla(dynamic x, int y) { }

        static void Main(string[] args)
        {
            Topla(4);
            Topla(Math.PI,3);
        }
    }
}
```

Uygulamayı derlediğimizde her hangibir hata mesajı ile karşılaşmadığımızı görürüz.



Oysa ki az önce **dynamic** tipin kullanıldığı senaryoda kararsızlık yaşandığına gözümüzle şahit olmuştuk. Peki ya şimdi ne oldu da sorun çözüldü? Aslında sorun değerlerin farklılaştırılması ile çözüm bulmuştur. Dikkat edilecek olursa **Topla(2,3)** çağrısı derleyicinin kafasının karışmasına neden olurken, **Topla(Math.PI,3)** çağrısında bu sorun oluşmamıştır. Tahmin edeceğiniz üzere **object** tipi için yaşanan sorunda farklı tipteki değerlerin kullanılması halinde çözülecektir. Ve çok doğal **double** tipini kullandığımız vaka için de çözüm olacaktır. 🙂

Tabi bu noktada şunu da belirtmekte yarar vardır. Söz konusu metodların ayrı bir kütüphane içerisinde tanımlanması halinde, bu kütüphaneyi kullanan tiplerin yazıda ele aldığımız hatalara düşme olasılıkları bulunabilir ki bu da istenen bir durum değildir. Dolayısıyla **C#**' ın temel kavramlarından birisi olarak ele alınana **metodların aşırı yüklenmesi(Overload)**aslında derinlerine inildiğinde dikkatli olunmayı gerektirecek

vakaları içermektedir. Aynen yazımızda ele aldığımız üzere. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

[örnek kodlar Visual Studio 2010 Ultimate RTM sürümü üzerinde geliştirilmiş ve test edilmiştir]

Değişken Atamalarında Bir Efsane (2010-04-12T11:30:00)

c#,

 Merhaba Arkadaşlar,

Yandaki resimde görülen kahramanları tanıyanınız var mı? Biraz düşünün isterseniz...**Jamie Hyneman** ve **Adam Savage** ikilisi tarafından sunulan ve **Wikipedia'** daki verilere göre **Discovery Channel** aracılığıyla ilk yayınını **23 Ocak 2003** tarihinde gerçekleştiren **MythBusters** isimli bu belgesel dizide, bilimsel metodlardan yararlanılarak bazı şehir efsanelerinin gerçeklikleri ispatlanmaya çalışılmaktadır.

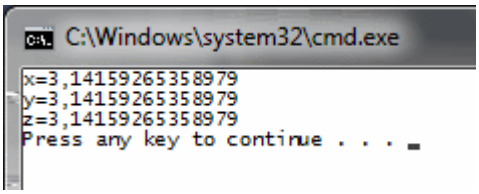
Aslında film sanayisinin ne kadar çok geliştiğini ispat eden bir dizidir. Nitekim dizinin sunucuları **Hollywood'** un tanınan film efekti teknisyenleridir *(ki bu nedenle bilimsel anlamda gayet donanımlıdırlar)* ve son derece ilginç efsaneleri araştırırlar. örneğin **"Yağmurda koşan mı yoksa yürüyen mi daha çok ıslanır?"**, **"Benzin istasyonunda cep telefonu kullanmak patlmaya neden olur mu?"**, **"Piercing yapan insanları yıldırım çarpar mı?"**,**"Emprise State binasının tepesinden yere serbest düşüşle bırakılan metal bir para betona saplanır mı?"** ve daha nice ilginç efsaneyi bilimsel taktikler ile çözmüşlerdir ve çözmeye devam etmektedirler. Açıkçası bende evdeki afacan müsade ettiği sürece bu tip belgeselleri kaçırmamaya çalışıyorum ve size de izlemenizi şiddetle tavsiye ediyorum.

Gelelim bu dizinin bu yazımızdaki konuyla ilgisinin ne olduğuna. Her zamanki gibi önce güzel bir giriş yapalım istedim. Ama asıl mesele **C#** tarafında da bazı efsanelerin olabileceği. üstelik bunların çoğundan habersisiz. Ancak Internet üzerinde siz de benim gibi yeteri kadar araştırma yaparsanız bu konulara ilişkin son derece güzel yazıların olduğunu keşfedebilirsiniz. Ben bu yazımızda değişken atamaları ile ilişkili bir konuyu incelemeye çalışacağım. Olayın çıkış noktası ise aşağıdaki kod parçamız olacak.

Vaka 1;

```
using System;

namespace AssignMyth
{
   class Program
   {
      static void Main(string[] args)
      {
         double x, y, z;

         x = y = z = Math.PI;

         Console.WriteLine("x={0}\ny={1}\nz={2}",x,y,z);
      }
   }
}
```

Uygulamanın çalışma zamanı çıktısı aşağıdaki gibidir.



Bu kod parçasında **double** tipinden olan **x, y** ve **z** değişkenlerine tek satırda **Pi** değerinin atanması söz konusudur. Bu son derece doğaldır nitekim eşitliğin sağından başlayan bir atama sırası mevcuttur. Hatta buna göre aşağıdaki ifade de doğrudur.

x = (y = (z = Math.PI));

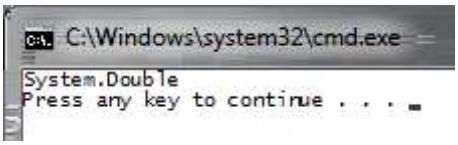
Nitekim parantezlerin olaya kattığı her hangibir öncelik bulunmamaktadır. Fakat aşağıdaki kod parçasını göz önüne aldığımızda eşitliğin en sağ tarafındaki değerden başlayarak en soldaki değişkene doğru yapılan atamaların her zaman sanıldığı gibi olmadığı izlenimine varmamız söz konusudur.

Vaka 2;

```
using System;

namespace AssignMyth
{
   class Program
   {
      static void Main(string[] args)
      {
```

```
            object x;
            double y;
            const float z = 3.14f;

            x = y = z;

            Console.WriteLine(x.GetType().ToString());
        }
    }
}
```

Ekran çıktısı aşağıdaki gibi olacaktır.



Hımmm...Enteresan bir durum söz konusu sanırım. Eşitliğin en sağında yer alan **z** isimli değişken aslında **float** tipinden tanımlanmıştır. Ardından hemen solunda yer alan **double**tipinden değişkene aktarılmıştır. **y** isimli değişken **double** tipindendir. Son olarak eşitliğin en solunda yer alan **x** isimli **object** tipinden değişkene bir atama yapılarak **3.14** değeri en sağdan en soldaki değişkene doğru taşınmıştır. Lakin değişkenin tipi eşitliğin en sağından en soluna kadar korunamamıştır. Ekran çıktısına dikkat edilecek olursa,**x** değişkeni gelen değeri **float** tipi yerine **double** tipi olarak ele almıştır. Yani **x=y=z** atamasında en soldaki x değişkeninin tipi **y'** nin tipine göre belirlenmektedir. Bu durumda eşitliğin en sağındaki değişkenin tipinin en soldaki **object** tipine taşınmasında bir anlamda bozulma olduğunu düşünebiliriz. Konuyu biraz daha ileri götürelim ve bu kez aşağıdaki kod parçasını göz önüne alalım.

Vaka 3;

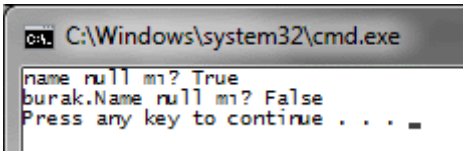
```
using System;

namespace AssignMyth
{
   class Program
   {
      static void Main(string[] args)
      {
         Person burak = new Person();

         object name = burak.Name = null;
         Console.WriteLine("name null mı? {0}",name==null);
         Console.WriteLine("burak.Name null mı? {0}",burak.Name==null);
```

```
    }
  }

  public class Person
  {
    private string _name;

    public string Name
    {
      get { return _name==null?"":_name; }
      set { _name = value; }
    }
  }
}
```

Dilerseniz kodun çalışma sonrası üretilen çıktıyı görmeden önce neler olduğuna bir bakalım. **Person** sınıfı içerisinde **Name** isimli bir **özellik(Property)** yer almaktadır. Bu özelliğe ait **get** bloğunda dikkat edilecek olursa **null** kontrolü yapılmaktadır. Bu kontrole göre **String** tipinden olan **_name** alanının değerinin **null** olması halinde geriye boş bir **string**döndürülmesi tercih edilmiştir. Aksi durumda ise **_name** değerinin kendisi döndürülmektedir. Buna göre aslında **Person** tipinin **Name** özellliği ya boş **string** ya da bir içeriğe sahiptir.

Main metodu içerisindeki kod parçasına baktığımızda ise ilgi çekici nokta **null** değer atamasının yapıldığı satırıdr. Eşitliğin en sağında yer alan **burak** isimli değişkenin **Name** özelliğine **null** değer atanmaktadır. Sonrasında ise bu değer **object** tipinden olan **name** değişkenine taşınmaktadır. İzleyen iki satırda ise **object** tipinden olan **name**değişkeni ile **burak** nesne örneğinin **Name** özelliklerinin değerlerinin **null** olup olmadığı kontrol edilmekte ve sonuçlar ekrana yazdırılmaktadır.

Normal şartlarda düşündüğümüzde **burak.Name** için geriye **null** değer dönmesi söz konusu değildir. Nitekim **get** bloğunda bunun için bir kontrol yapılmaktadır. Bu durumda son satırın sonucunda **false** değer dönmesi beklenmektedir. Diğer yandan en soldaki **name** değişkenine yapılan atamaya göre de **null** değer yerine **""** değeri taşınmış olmalıdır ki buna göre de **name** değerinin **null** olması sonucunun **false** dönmesi gerekmektedir. Ama uygulamayı çalıştırdığımızda sonuçların aşağıdaki gibi olduğu görülecektir.



Oda ne? **burak.Name== null** için **False** değer dönmüştür ve bu beklediğimiz sonuçtur. Ancak **name==null** kontrolünün değeri **true** olarak gelmiştir. Oysaki atamaya

göre **name**değişkenine **""** değerinin gelmesi ve bu nedenle **null** olmaması gerekmektedir. İlginç değil mi?

Sonuç olarak bu yazıda bahsettiğimiz şekliyle gerçekleştirilen atamalarda, eşitliğin en sağındaki değerin sola doğru taşındığı efsanesinin tam olarak doğru olmadığı ispatlanmış bulunmaktadır. Nitekim ilk vakada eşitliğin en sağından soluna aynı değer başarılı bir şekilde atanmaktadır. Ancak ikinci vakaya göre aslında en soldaki değişkenin bir sağındakinin tipine büründüğü de görülmektedir. üstelik **Vaka 3**' e göre en soldaki değişken en sağdan atanan değere bürünmüş ve bir sağındakini kaale bile almamıştır...Kafanız karıştı mı? Bakalım başka ne gibi efsaneler var. İlerleyen yazılarda değinmeye çalışıyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

AssignMyth_RC.rar (20,03 kb)

NedirTv?com 4ncü Yıl Etkinlikleri Tamamlandı (2010-04-11T02:10:00)

seminer,nedirtv?com,wcf eco system,webhttp services,data services,ria services,workflow services,windows server appfabric,dublin,velocity,

Merhaba Arkadaşlar,

Uzun süredir etkinliklerde konuşmacı olarak yer alamıyordum. **.Net Framework 4.0** ve **Visual Studio 2010** ürünlerinin **RTM** sürümlerinin çıkmak üzere olduğu bu günlerde, editörlüğünü yaptığım **NedirTV?com** bünyesinde bir etkinlikte yer almak beni çok mutlu etti. Etkinlik boyunca katılımcıların gösterdiği ilgiden çok memnun kaldığımı itiraf etmek isterim. özellikle görselliğin ön planda olmadığı servis bazlı konuların ele alındığı oturumlarımda, beni bıkmadan, sıkılmadan dinleyen tüm arkadaşlarıma canı gönülden teşekkür ediyorum.

NedirTv?com topluluğunun **4ncü** kuruluş yıldönümü vesilesiyle gerçekleştirilen etkinlikte, çok değerli isimler de oturumları ile yer aldılar. Kurucu **Uğur Umutluoğlu(MVP - Asp.Net)**,**Daron Yöndem(Microsoft RD, Silverlight MVP)** ve **Selçuk Yavuz** arkadaşlarımız, etkili ve bir o kadarda bilgilendirici sunumları ile güne anlam kattılar. Ben de **WCF Eco System** ve **Windows Server AppFabric** konuları üzerine iki sunumla bu etkinliğe elimden geldiğince katkıda bulundum. Sunumlarımı ve **WCF Eco System** konusu üzerine geliştirdiğimiz örnekleri aşağıda bulabilirsiniz. Bir dahaki etkinlikte görüşmek dileğiyle hepinize mutlu günler dilerim.

Sunum Dosyaları [örnekler Visual Studio 2010 Ultimate RC sürümü üzerinde geliştirilmiş ve test edilmiştir]

WCF Eco System.pptx (475,44 kb)

Windows Server AppFabric - Introduction.pptx (688,28 kb)

WCF Eco System Demoları

NedirTv.rar (2,63 mb)

Seminer Resimleri

WCF Web Http Services - ETags (2010-04-09T14:05:00)

wcf 4.0,wcf,wcf webhttp services,rest,

 Merhaba Arkadaşlar,

WCF WebHttp Service' leri ile ilişkili yazılarımıza kaldığımız yerden devam ediyoruz. Bu yazımızda **ETag(Entity Tag)** kullanarak sunucu ile istemci arasındaki veri trafiğini nasıl azaltabileceğimizi incelemeye çalışacağız. öncelikle istemci ile servis arasındaki iletişimi düşünerek ilerlemeye çalışalım. İstemci, sunucu üzerinde yer alan bir operasyon için talepte bulunduğunda bir cevap üretilecek ve buna bağlı bir içerik verisi istemci tarafına indirilecektir. Bu süreç tipik olarak **Request-Response** senaryosundan farklı bir işleyiş değildir. İstemci sonraki bir zaman diliminde aynı operasyona yeni bir talepte bulunduğunda ise, üretilecek olan sunucu **cevabının(Response)** bir öncekine göre hiç değişmemiş olma ihtimalide bulunmaktadır. Eğer istemci tarafı bir şekilde gönderdiği talebin karşılığı olan cevabın değişmediğini anlayabilirse ve kendisinde bu içerik zaten tampon alanda duruyorsa, aynı içeriğin sunucudan istemci tarafına bir kere daha indirilmesine gerek yoktur. İşte **ETag** takısının devreye girdiği nokta burasıdır.

Peki **ETag(Entity Tag)** tam olarak nedir? **Entity Tag**, sunucudan istemci tarafına gönderilen **Response** paketlerinin **Header** kısmında kullanılabilen bir takıdır. Bu takı yardımıyla bilginin değişikliğe uğrayıp uğramadığı kolayca anlaşılabilir. Bu ayrım bize

performans açısından bir kazanım sağlayabilir. öyleki, sunucu aynı **ETag** verisine sahip iki **Response** ürettiğinde, aslında istemcinin talebinin karşılığının bir öncekisi ile aynı olduğu sonucuna varılabilir. Bu noktada istemcinin içeriği tampon bir bölgede tuttuğu düşünüldüğünde, bir önceki ile aynı olan veri içeriğini sunucudan indirmesine gerek kalmayacaktır. Böyle bir durumda sunucun istemci tarafına **HTTP 304 Not Modified** bilgisi göndermesi söz konusudur. Tabi burada **ETag** içerisine yazılacak verinin nasıl üretileceği de önemlidir. Genellikle **Entity** ile alakalı olaraktan son güncelleme zamanı veya**checksum** kullanılabilir. Hatta **SQL** veritabanında kullanılabilen **Timestamp** tipide **ETag** verisi olarak ciddi anlamda düşünülebilir. Sonuç itibariyle **ETag** kavramının **Caching** modeli için çok önemli bir kriter olduğunu söyleyebiliriz.

Şimdi konuyu aşağıdaki içeriğe sahip bir **WebHttp Service** örneği üzerinden değerlendirmeye çalışalım.

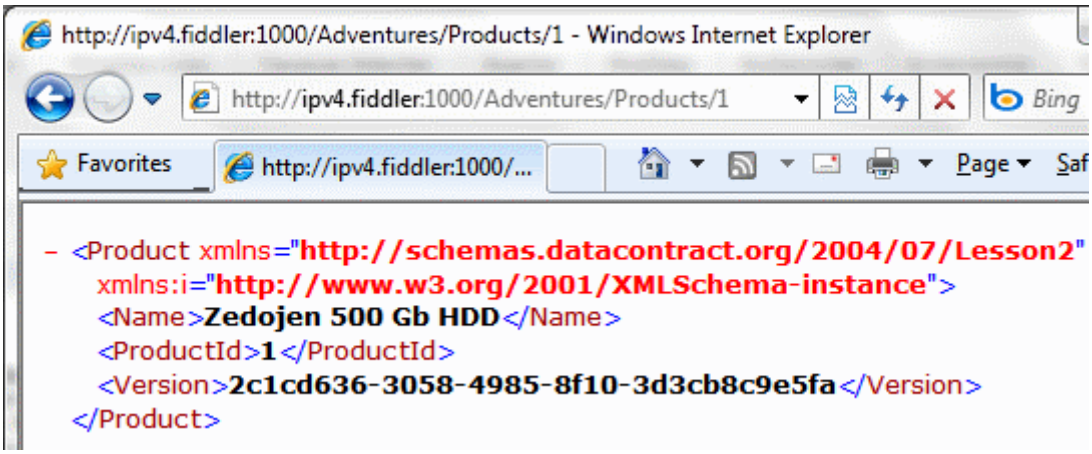
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;

namespace Lesson2
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class ProductService
   {
      static List<Product> products = new List<Product>
      {
         new Product{ ProductId="1",Name="Zedojen 500 Gb
HDD", Version=Guid.NewGuid()},
         new Product{ ProductId="2",Name="Zedojen 750 Gb
HDD", Version=Guid.NewGuid()},
         new Product{ ProductId="3",Name="Leveno Laptop XK
5301", Version=Guid.NewGuid()}
      };

      [WebGet(UriTemplate = "Products/{productId}")]
      public Product GetProduct(string productId)
      {
         var product = (from p in products
                        where p.ProductId == productId
```

```
                    select p).FirstOrDefault();
            // Eğer bir Product nesne örneği mevcutsa...
            if (product != null)
            {
                // İstemciden gelen paket Header' ındaki If-None-Match değeri alınır ve sunucu tarafında bulunan Product nesne örneğinin Version özelliğinin değeri ile kıyaslanır. Eğer aynı ise bu istemci tarafına HTTP 304 Not Modified döndürüleceği anlamına gelir.
                WebOperationContext.Current.IncomingRequest.CheckConditionalRetrieve(product.Version);
                // Response içerisinde yer alan HTTP Header içerisindeki ETag değeri sunucu tarafından bulunan Product nesne örneğinin Version özelliği ile set edilir.
                WebOperationContext.Current.OutgoingResponse.SetETag(product.Version);
            }

            return product;
        }
    }

    public class Product
    {
        public string ProductId { get; set; }
        public string Name { get; set; }
        public Guid Version { get; set; }
    }
}
```

Servisimizde yer alan **GetProduct** isimli operasyon geriye **Product** tipinden bir nesne içeriği döndürmektedir. **Product** tipinin en önemli özelliklerinden birisi ise **Guid** tipinden olan **Version**' dur. Burada veritabanında yer alan bir ürünün **ETag** için kullanılabilecek veri tipi simüle edilmeye çalışılmaktadır. Yazımızın başında da belirttiğimiz gibi tablo bazlı kaynağın söz konusu olması halinde, **Guid** yerine **Timestamp** tipinden bir alan da tercih edilebilir.

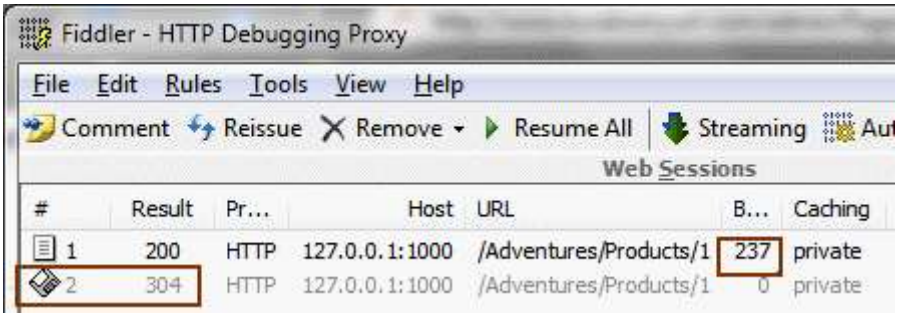
GetProduct metodu içerisinde **Exception** kontrolü yapılmamaktadır. Daha çok üzerinde durmak istediğimiz nokta **ETag** veri kontrolü ve üretimidir. Bu nedenle **WebOperationContext.Current**üzerinden çağırılan **CheckConditionalRetrieve** ve **SetETag** metodlarına konsantre olmamızda yarar vardır. **CheckConditionalRetrieve** metodu istemciden gelen talebe ait içerikteki **If-None-Match** değeri kontrolünü yapmaktadır. Eğer istemci aynı içeriği bir kere daha talep etmişse, bu durumda istediği **Product** tipinin **Version** değeri ile gönderdiği **If-None-Match** değeri aynı olmalıdır. Tabiki buradaki örnek senaryomuzda şu an için veri tarafında yer alan **Version** alanının değişmediğini düşünüyoruz. özellikle tarih bazlı olarak tutulan veri içeriklerinde ve örneğin son güncelleme tarihinin **ETag** olarak kullanıldığı durumlarda ya da herhangibir değişiklik sonrası ilgili versiyon kontrolü alanlarının değerlerinin

değiştirildiği hallerde, sunucu tarafından istemciye doğru **veri indirilmesi(Download)** işlemi yinelenecektir. **SetETag** metodu ise ilk gelen talep sonrası veya içeriğin istemciye indirilmesi gerektiği talep sonrası, **Response**' a ait içeriğe o anki **Product** nesne örneğinin **Version** değerini atayacaktır. Her iki metodunda farklı tipte **aşırı yüklenmiş(Overload)** versiyonları bulunmaktadır. Bu versiyonlardan birisi de örneğimizde ele aldığımız **Guid** veri tipi ile çalışanıdır. Dilerseniz durumu daha iyi anlamak için hemen testlerimize başlayalım. örneğimizi tarayıcı uygulama üzerinden talep ettiğimizde ilk etapta aşağıdaki örnek sonuçlar ile karşılaşırız.



Not : Burada ipv4.fiddler:1000 şeklindeki kök adres kullanımı mutlaka dikkatinizi çekmiştir. Bunu Fiddler üzerinden örneğimize ait HTTP paketlerini debug edebilmek için kullandığımızı belirtmek isterim.

örnekte **ProductId** değeri **1** olan ürüne ait bilgilerin elde edildiği görülmektedir. Bundan sonra **1 numaralı ürünü tekrardan talep edecek olursak Fiddler** tarafından aşağıdaki **HTTP**hareketliliklerinin yakalandığını görebiliriz.



Dikkatinizi çeken bir şey var mı? 🙂 İlk talep sonrasında sunucudan istemiye **HTTP 200 Ok** bilgisi dönmüş ve **237 Byte**' lık bir **Body** içeriği indirilmiştir. Diğer yandan aynı talebin ikinci kez yapılması sonrasında istemci tarafına **HTTP 304 Not Modified** mesajının döndürüldüğü görülmektedir. üstelik ikinci talep sonrası **Body** içeriği **0 byte** uzunluğundadır. Volaaa!!! 😃 Yani ikinci talebin ilki ile aynı veri üretimine sahip olduğu anlaşılmış ve bu sebepten üretilen paketin istemci tarafına yeniden indirilmesine gerek kalınmamıştır. örneğimize göre **byte** seviyesinde bu çok önemli bir performans kazanımına neden olmamaktadır. Ne varki **video, resim, müzik** gibi büyük

boyutlu **binary** içeriklerin yer aldığı paketlerde **304**döndürülmesinin büyük önemi vardır. Nitekim az önce üretilip istemci tarafına indirilen büyük boyutlu içeriğin, ikinci talep sonrası zaten istemci tarafındaki tamponda duran versiyonu ile aynı olması nedeniyle, yeniden gönderilmesi durumu ortadan kaldırılmakta ve böylece istemci ile sunucu arasındaki ağ trafiğinden akan veri boyutu minimize edilmektedir.

Şimdi **Fiddler** aracı yardımıyla paketlerin içeriklerine biraz daha yakından bakalım. İlk talep sonrası istemcinin gönderdiği içerik aşağıdaki gibidir.

GET http://127.0.0.1:1000/Adventures/Products/1 HTTP/1.1
Accept: application/x-ms-application, image/jpeg, application/xaml+xml, image/gif, image/pjpeg, application/x-ms-xbap, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwave-flash, */*
Accept-Language: tr
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.2; OfficeLiveConnector.1.4; OfficeLivePatch.1.3; .NET4.0C; .NET4.0E; MS-RTC LM 8)
Accept-Encoding: gzip, deflate
Connection: Keep-Alive
Host: 127.0.0.1:1000

Bu talebe karşılık sunucunu cevabı ise aşağıdaki gibi olacaktır.

HTTP/1.1 200 OK
Server: ASP.NET Development Server/10.0.0.0
Date: Fri, 26 Feb 2010 09:43:00 GMT
X-AspNet-Version: 4.0.30128
Content-Length: 237
ETag: "2c1cd636-3058-4985-8f10-3d3cb8c9e5fa"
Cache-Control: private
Content-Type: application/xml; charset=utf-8
Connection: Close

<Product xmlns="http://schemas.datacontract.org/2004/07/Lesson2" xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><Name>Zedojen 500 Gb HDD</Name><ProductId>1</ProductId><Version>2c1cd636-3058-4985-8f10-3d3cb8c9e5fa</Version></Product>

Dikkat edileceği üzere **Response** içerisinde bir **ETag** değeri olduğu görülmektedir. Sizce bu değerin **1** numaralı **Product**' ın güncel **Version** değeri ile aynı olması bir tesadüf müdür? 🙂 Ayrıca içeriğin uzunluğu **237 byte'** tır.

Gelelim ikinci talebe. İstemci tarafından sunucuya gönderilen ikinci talebin içeriği aşağıdaki gibidir.

GET http://127.0.0.1:1000/Adventures/Products/1 HTTP/1.1
Accept: application/x-ms-application, image/jpeg, application/xaml+xml, image/gif, image/pjpeg, application/x-ms-xbap, application/vnd.ms-excel, application/vnd.ms-powerpoint, application/msword, application/x-shockwave-flash, */*
Accept-Language: tr
User-Agent: Mozilla/4.0 (compatible; MSIE 8.0; Windows NT 6.1; WOW64; Trident/4.0; SLCC2; .NET CLR 2.0.50727; .NET CLR 3.5.30729; .NET CLR 3.0.30729; Media Center PC 6.0; InfoPath.2; OfficeLiveConnector.1.4; OfficeLivePatch.1.3; .NET4.0C; .NET4.0E; MS-RTC LM 8)
Accept-Encoding: gzip, deflate
If-None-Match: "2c1cd636-3058-4985-8f10-3d3cb8c9e5fa"
Host: 127.0.0.1:1000
Connection: Keep-Alive

Dikkat edileceği üzere **Guid** değerine sahip olan **If-None-Match** isimli element bulunmaktadır. Bu değer biraz önceki talep sonucu istemciye gönderilen **ETag** değeridir aslında. Şimdi bu noktada sunucu üzerinde yer alan **1** numaralı ürünün içeriğinin değiştirilmediği ve bu nedenle **Guid** değerinin de aynı olduğu düşünülmektedir. Bu sebepten sunucu tarafından istemciye gönderilen cevabın içeriği aşağıdaki gibi olacaktır.

HTTP/1.1 304 Not Modified
Server: ASP.NET Development Server/10.0.0.0
Date: Fri, 26 Feb 2010 09:43:05 GMT
X-AspNet-Version: 4.0.30128
ETag: "2c1cd636-3058-4985-8f10-3d3cb8c9e5fa"
Cache-Control: private
Connection: Close

Her hangibir **içerik(Content)** yoktur. Hatta **0 byte** uzunluğunda bir **Content** mevcuttur. Ama daha önemlisi yine **ETag** elementi vardır ve **Guid** değerini içermektedir. Ayrıca **HTTP 304 Not Modified**bilgisinin döndürüldüğü dikkatlerden kaçmamalıdır.

Peki sunucu tarafındaki **Product** içeriğinde ve dolayısıyla **Version** değerinde bir değişme olursa? Bir veritabanı örneği geliştirmediğimiz için bu durumu şu şekilde simüle edebiliriz ; **1** numaralı **ProductId** değerine sahip ürünün adını **Visual Studio 2010** ortamında değiştirip örneği tekrardan **build** ederek. Yeniden build işlemi sonucu **static** olarak tanımlanan**List<Product>** koleksiyon içeriğinin üretimi yinelenecektir. Bu da yeni **Guid** değerlerinin üretimi anlamına gelmektedir. Bu durumda servis operasyonuna yeniden talepte bulunursak aşağıdaki cevabı aldığımız görürüz.

HTTP/1.1 200 OK
Server: ASP.NET Development Server/10.0.0.0
Date: Fri, 26 Feb 2010 11:52:38 GMT
X-AspNet-Version: 4.0.30128
Content-Length: 237

ETag: "2fadf1be-d5c3-4fe0-a9c6-ecf20437ffe4"
Cache-Control: private
Content-Type: application/xml; charset=utf-8
Connection: Close

<Product xmlns="http://schemas.datacontract.org/2004/07/Lesson2" xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><Name>Zedojen 250 Gb HDD</Name><ProductId>1</ProductId><Version>2fadf1be-d5c3-4fe0-a9c6-ecf20437ffe4</Version></Product>

Dikkat edileceği üzere **Guid** değeri bir öncekinden farklıdır ve istemciye **HTTP 200 Ok** koduyla **Product** içeriği tekrardan gönderilmiştir. Tabi bunun sonrasında 1 numaralı ürünü yeninden talep edersek yine HTTP 304 Not Modified durumu ile karşılaşırız.

Buraya kadar her şey iyi gitti. Ancak testlerimizi farkettiğiniz üzere **Internet Explorer** gibi tarayıcı uygulamalar üzerinden gerçekleştirdik. Oysaki istemci uygulamayı biz yazıyorsa **ETag** kullanımı için de yapmamız gereken ekstra işlemler söz konusudur. Bu amaçla az önce geliştirdiğimiz servis uygulamasını test edeceğimiz basit bir **Console Application** geliştirdiğimizi düşünelim. Kod içeriğini aşağıdaki gibi yazmamız **ETag** desteği için yeterli olacaktır.

```
using System;
using Microsoft.Http;
using Microsoft.Http.Headers;

namespace Client
{
   class Program
   {
      static void Main(string[] args)
      {
         Uri serviceAddress = new Uri(@"http://ipv4.fiddler:1000/Adventures/");
         using (HttpClient client = new HttpClient(serviceAddress))
         {
            EntityTag eTag=null;
            Process(client, "1",ref eTag);
            Process(client, "1",ref eTag);
         }
      }

      static void Process(HttpClient client, string ProductId,ref EntityTag ETag)
      {
         Console.WriteLine("***{0} için Talep***\n",ProductId);
         // Talebin hazırlanması ve gönderilmesi
         using (HttpRequestMessage request = new HttpRequestMessage("GET",
```

```
"Products/"+ProductId))
        {
          // Metoda referans olarak gelen EntityTag tipinden olan ETag değerine bakılır.
Eğer null değil ise ki ilk talep sonrası sunucu tarafından ürünün Version değeri ile
doldurulacaktır; bu durumda Header kısmına If-None-Match değerinin eklenmesi sağlanır.
          if (ETag != null)
            request.Headers.IfNoneMatch.Add(ETag);

          // If-None-Match değeri içeren talep gönderilir
          using (HttpResponseMessage response = client.Send(request))
          {
            // ETag değeri gelen cevaptan alınır ve ref tipinden olan metod parametresine
aktarılır. Böylece Process metoduna yapılacak olan sonraki çağrılarda aynı ETag değerinin
taşınması kolaylaşmaktadır.
            ETag = response.Headers.ETag;
            // Sonuçlar ekran yazdırılır.
            Console.WriteLine("StatusCode : {0}\n", response.StatusCode);
            Console.WriteLine("Content : {0}\n",response.Content.ReadAsString());
            Console.WriteLine("ETag Değeri : {0}\n", ETag.Tag);
          }
        }
        Console.WriteLine("*******");
      }
    }
}
```

Uygulamamızı çalıştırdığımızd aşağıdaki sonuçlar ile karşılaşırız.

Görüldüğü gibi, ikinci talep sonrasında istemci tarafına **HTTP 304 Not Modified** bilgisi ve **0 Byte** uzunluğunda içerik gönderilmiştir. Böylece **WCF WebHttp Service**' leri ile ilişkili bir yazımızın daha sonuna geldik. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Lesson9_RC.rar (175,04 kb) **[örnek Visual Studio 2010 Ultimate RC sürümü üzerinde geliştirilmiş ve test edilmiştir]**

WCF WebHttp Services - Özel Formatta Mesaj Döndürmek (2010-04-08T16:39:00)

wcf,wcf 4.0,webhttp services,restful,non soap,wcf webhttp services,

 Merhaba Arkadaşlar,

Bu yazımızda bizleri uzun, zorlu ve yorucu bir macera bekliyor. Şimdiden söylemek isterim ki yanınızda tatlı*(Mesela kolalı jelibon olabilir)*, tuzlu yiyecek bir şeyler, boğaz kuruluğunuzu giderecek içecekler veya daha fazla oksijen çekmenizi sağlayacak sakızlar olsun. Unutmadan birde aspirin. Baş ağrısı için Gelelim bu günkü konumuza.

Bu yazımızda, son günlerde sıklıkla üzerinde durduğumuz **WCF WebHttp Service'** lerinde, istemciden gelen root adres bazlı taleplerin nasıl karşılanacağını ve özel formatta mesajların nasıl döndürüleceğini incelemeye çalışıyor olacağız. Ancak işe başlamadan önce ihtiyacın ne olduğundan bahsetmemizde yarar var. Bu amaçla bir **Web** uygulaması üzerinden **host** edilen birden fazla **WebHttp** servisimiz olduğunu düşünerek ilerleyelim. Bu servisler içerisinde de örneğin **HTTP Get** taleplerinin karışılığında çeşitli tipte koleksiyonları döndüren operasyonlarımız olduğunu farz edelim. Bu durumda **global.asax** dosyasındaki kodlarda yönlendirme tablosuna ekleyeceğimiz adres bilgilerine göre, gelen talepleri uygun olan servislere yöndermemiz mümkün olacaktır. Bunu zaten daha önceki bir yazımızda incelemiştik.

Söz gelimi **http://makineadı:port numarası/CompanyServices/AdventureWorks/Products** ile **http://makineadı:port numarası/CompanyServices/Chinook/Albums** gibi iki talep gönderildiğini düşünelim. Bu taleplerin aynı web uygulamasından host edilen iki farklı servis tipi tarafından değerlendirildiği bir durumda, doğru yönlendirme tekniği ile uygun olan servis ve operasyonunun çağırılması mümkündür. Oysaki

istemciler **http://makineadı:port/CompanyServices/** adresine de talepte bulunulabilir. Böyle bir durumda ne olur? Gelin bunu açıklamak için aşağıdaki örnek servis sınıflarını içeren bir**WCF REST Service Application** projemiz olduğunu düşünelim.

AdventureWorksService sınıfı;

```
using System.Collections.Generic;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;

namespace Lesson8
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class AdventureWorksService
   {
      [WebGet(UriTemplate = "Products")]
      public List<Product> GetProducts()
      {
         return new List<Product>
         {
            new Product{ Id=1, Name="Büyüteç", ListPrice=1.24M},
            new Product{ Id=2, Name="Stabilo Pen 68", ListPrice=2.35M},
            new Product{ Id=3, Name="Temizleme Spreyi", ListPrice=4.19M}
         };
      }
   }
}
```

Söz konusu sınıf içerisinde yer alan **GetProducts** isimli operasyon geriye **Product** tipinden bir ürün listesi döndürmekle görevlendirilmiştir. Tahmin edeceğiniz üzere şu anda kendi kendimizi yetiştirmeye çalıştığımızdan sadece anlamsız bir liste üretimi söz konusudur.

ChinookService sınıfı;

```
using System.Collections.Generic;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;

namespace Lesson8
{
```

```
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class ChinookService
   {
      [WebGet(UriTemplate = "Artists")]
      public List<Artist> GetArtists()
      {
         return new List<Artist>
         {
            new Artist{ Id=1, Name="Ayrosimit",IsGroup=true},
            new Artist{ Id=2, Name="Megadet",IsGroup=true},
            new Artist{ Id=3, Name="Metalika",IsGroup=true},
            new Artist{ Id=4, Name="Co Satriani",IsGroup=false}
         };
      }
   }
}
```

ChinookService sınıfıda, **AdventureWorksService** tipine benzer bir şekilde ama bu kez **Artist** tipinden liste döndüren tek bir operasyon içermektedir. Buraya kadar zaten bir sorun bulunmamaktadır. Ancak aynı web uygulamasında birden fazla servisi host etmek istediğimizde, **RouteTable** nesnesinin **Routes** koleksiyonu içerisinde gerekli düzenlemelerin de yapılması gerekmektedir. Bu nedenle**global.asax.cs** içeriğinin aşağıdaki gibi olduğunu düşünebiliriz.

```
using System;
using System.ServiceModel.Activation;
using System.Web;
using System.Web.Routing;

namespace Lesson8
{
   public class Global
      : HttpApplication
   {
      void Application_Start(object sender, EventArgs e)
      {
         RegisterRoutes();
      }

      private void RegisterRoutes()
      {
         RouteTable.Routes.Add(new ServiceRoute("AdventureWorks", new
```

```
WebServiceHostFactory(), typeof(AdventureWorksService)));
            RouteTable.Routes.Add(new ServiceRoute("Chinook", new
WebServiceHostFactory(), typeof(ChinookService)));
        }
    }
}
```

Buna göre istemciden gelecek
olan **http://localhost:10843/CompanyServices/AdventureWorks/Products** ve **http://localhost:10843/CompanyServices/Chinook/Artists** talepleri sorunsuz bir şekilde karşılanacaktır. Ancak
doğrudan **Web** uygulamasının **Root** adresine yapılan **http://localhost:10843/CompanyServices/** gibi bir talepte aşağıdaki ekran görüntüsü ile karşılaşılacaktır.

Elbette **Help** sayfalarına gidilerek servislere nasıl talepte bulunulabileceği öğrenilebilir. Ancak elimizde iki servis olduğundan söz konusu yardım sayfalarına gitmek için**http://localhost:10843/CompanyServices/AdventureWorks/help** veya **http://localhost:10843/CompanyServices/Chinook/help** gibi taleplerinin gönderilmesi gerekmektedir.

Sanıyorum ki nihayet ne yapmak istediğimize gelebildik. İstediğimiz
şey **http://localhost:10843/CompanyServices/** adresine yapılan talep ile Web uygulamasından sunulan servisleri bildirmek olacak. üstelik bu talebe karşılık dönecek mesajın içeriğini kendimiz tasarlayacağız. Yapabilir miyiz? Evet yapabiliriz. çünkü gerekli

tüm tipler **Framework** içerisinde çoktandır mevcutlar. özetle talebe uygun bir formatta*(XML, JSON, ATOM gibi)* kendi veri yayınımızı yapacağımızı ifade edebiliriz.

İşe ilk olarak yeni bir servis sınıfını geliştirerek başlamamız gerekiyor. Bu sınıf içerisinde yer alan operasyonumuz geriye, **System.ServiceModel.Channels** isim alanında yer alan**Message** tipinden bir değer döndürüyor olacak. İşte **EntranceService** isimli yeni sınıfımızın içeriği;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Channels;
using System.ServiceModel.Syndication;
using System.ServiceModel.Web;

namespace Lesson8
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class EntranceService
   {
      // İstemci tarafına sunulacak olan kaynakların listesi Resource tipinden generic bir
koleksiyonda tutulur.
      public static readonly List<Resource> Resources = new List<Resource>
      {
         new Resource{ Name="Chinook List", Description="Chinook nesneleri listesi",
RequestUri=new Uri("Chinook",UriKind.Relative),HelpUri=new
Uri("Chinook/help",UriKind.Relative)},
         new Resource{ Name="Adventure Works List", Description="Adventure Works
nesneleri listesi", RequestUri=new
Uri("AdventureWorks",UriKind.Relative),HelpUri=new
Uri("AdventureWorks/help",UriKind.Relative)}
      };

      [WebGet(UriTemplate="")]
      public Message GetResources()
      {
         WebOperationContext optContext = WebOperationContext.Current;
         IncomingWebRequestContext incomingRequest =
optContext.IncomingRequest;
```

```
            string mType=string.Empty;
            foreach (var acceptType in incomingRequest.GetAcceptHeaderElements())
            {
               // Gelen talebin Header bilgisinde yer alan MediaType değerine göre geriye Xml, Json veya Atom formatında bir içerik döndürülmesi sağlanır.
                mType= acceptType.MediaType.ToLower();

                if (mType == "application/xml" || mType == "text/xml")
                   return optContext.CreateXmlResponse(Resources); // Xml formatında dönüş
                else if (mType == "application/json")
                   return optContext.CreateJsonResponse(Resources);
                else if (mType == "application/atom+xml") // Json formatında dönüş
                {
                   // Atom formatında dönüş için SyndicationFeed nesnesinin örneklenmesi gerekmektedir. Resource değişkeninin işaret ettiği koleksiyon bu nesne örneği içerisindeki Items koleksiyonuna atanır
                   return optContext.CreateAtom10Response(
                      new SyndicationFeed(
                         "Company Services Resources",
                         "Adventure Works & Chinook Kaynakları",
                         new Uri("", UriKind.Relative),
                         Resources.Select(r => new SyndicationItem(r.Name, r.Description, r.RequestUri)
                         )));
                }
            }
            // Varsayılan olarak çıktı XML formatında verilir
            return optContext.CreateXmlResponse(Resources);
        }
    }

    // Servisten sunulan servislerin birer kaynak olduğu düşünüldüğünde bu servislere ait ad, açıklama, root Uri ve help page uri bilgilerinin saklandığı tip
    public class Resource
    {
        public string Name { get; set; }
        public string Description { get; set; }
        public Uri RequestUri { get; set; }
        public Uri HelpUri { get; set; }
    }
}
```

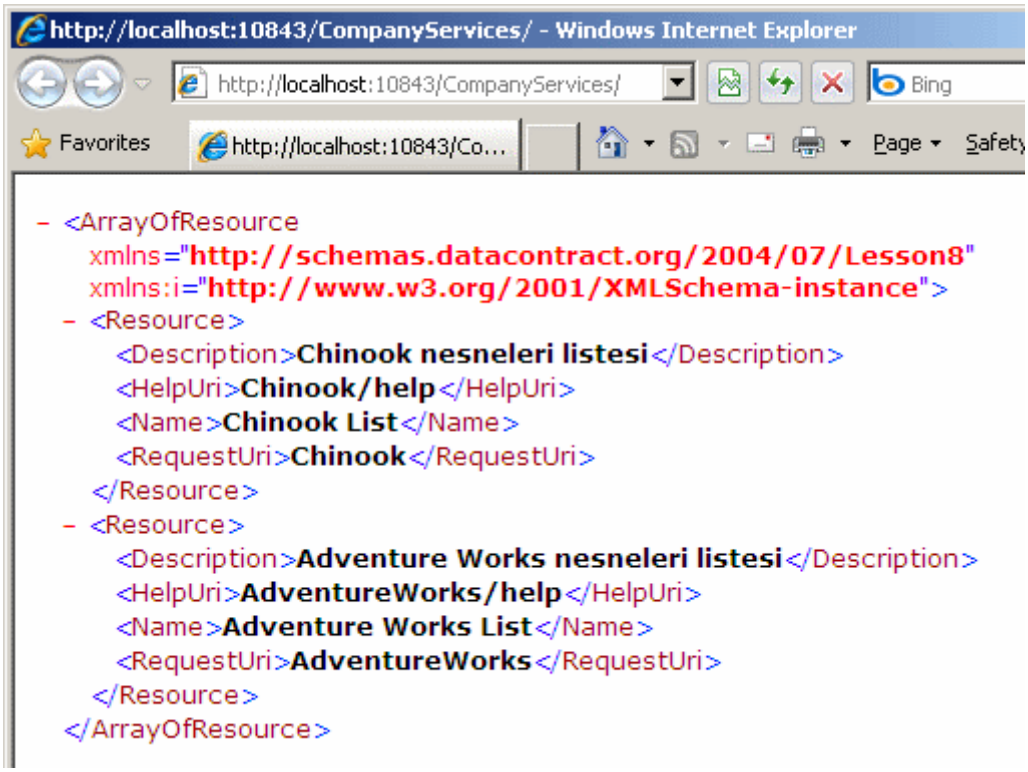
GetResources isimli servis operasyonunun en önemli özelliği **WebGet** niteliğinde boş bir template kullanılması ve tabiki geriye **Message** tipinden bir değer döndürmesidir. Servis

operasyonu dikkatlice incelendiğinde, istemciden gelecek olan taleplere ait **Header** kısımlarında yer alan mesaj formatı bilgisine göre bir çıktı üretildiği görülebilir. Buna göre standart olarak **XML, JSON** ve **ATOM** formatlarında bir üretim söz konusudur. **ATOM** formatındaki çıktının hazırlanması sırasında bir **SyndicationFeed** nesnesinin örneklendiğine dikkat edilmelidir. Tüm bu formatlama işlemlerinde o anki **Web** içeriği referansını taşıyan **WebOperationContext** tipine ait **Create...** metodlarından yararlanılmaktadır.

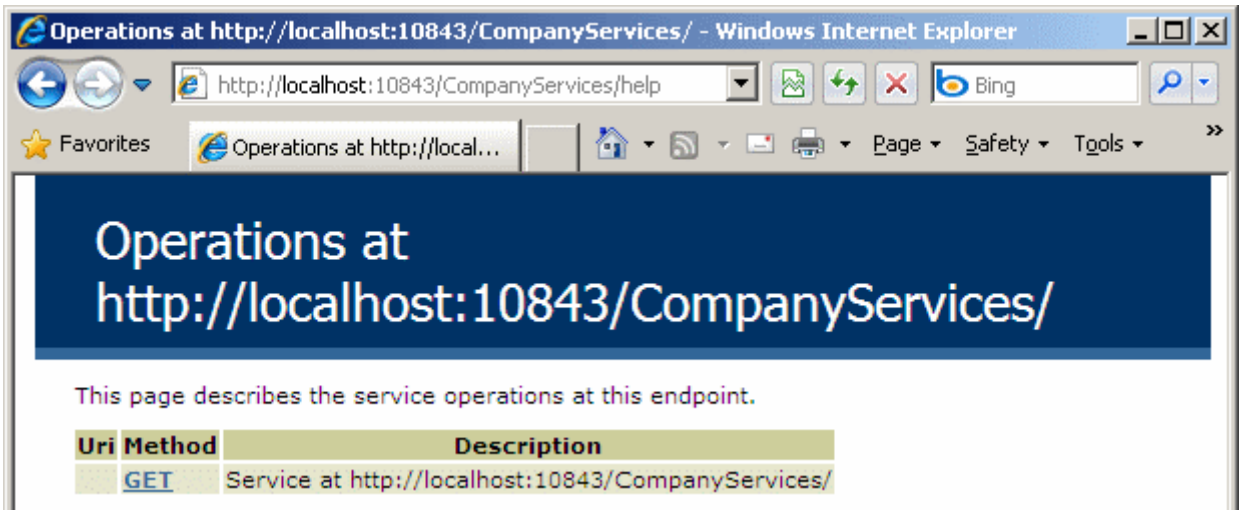
Tabi işimiz bu servis sınıfını yazmakla bitmiş değil. Fark ettiğiniz üzere üçüncü bir servis sınıfımız oldu ve bu sınıfa gelen talepler **UriTemplate** bilgisine göre **Web**uygulamasının **Root** adresine yapılmakta. Dolayısıyla boş template için gerekli yönlendirme bilgisinin **global.asax.cs** içerisinde bildirilmesi gerekiyor. Aynen aşağıda olduğu gibi.

```
using System;
using System.ServiceModel.Activation;
using System.Web;
using System.Web.Routing;

namespace Lesson8
{
   public class Global
      : HttpApplication
   {
      void Application_Start(object sender, EventArgs e)
      {
         RegisterRoutes();
      }

      private void RegisterRoutes()
      {
         RouteTable.Routes.Add(new ServiceRoute("", new WebServiceHostFactory(), 
typeof(EntranceService)));
         RouteTable.Routes.Add(new ServiceRoute("AdventureWorks", new 
WebServiceHostFactory(), typeof(AdventureWorksService)));
         RouteTable.Routes.Add(new ServiceRoute("Chinook", new 
WebServiceHostFactory(), typeof(ChinookService)));
      }
   }
}
```

Buna göre örnek bir tarayıcı uygulama üzerinden **http://localhost:10843/CompanyServices/** adresine yapacağımız bir talebin sonucu aşağıdaki gibi olacaktır.



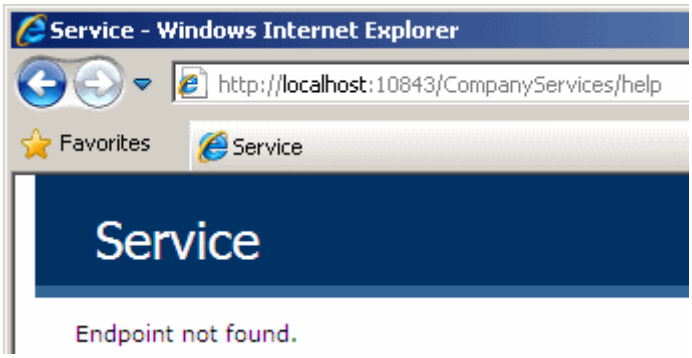
Her şey yolunda görünüyor. Ancak ufak bir pürüz var. **EntranceService** tipine boş **Uri** bilgisi üzerinden bir başka deyişle **Web** uygulamasına ait **Root** adresten gidilebildiği için,**http://localhost:10843/CompanyServices/help** şeklinde gönderilen bir talepte aşağıdaki ekran görüntüsü ile karşılaşılacaktır.



Oysaki bu **Help** sayfasının çıkmasına pekte gerek yoktur. Bu bir zorunluluk değildir ama olmasının da bir anlamı yoktur. Dolayısıyla pasif hale getirmemiz gerekmektedir. Bu amaçla **WCF 4.0**ile birlikte konfigurasyon dosyasına getirilen yeniliklerden yararlanarak gerekli sonucu elde edebiliriz. Tek yapmamız gereken **Web.config** dosyasını aşağıdaki gibi düzenlemek.

```
<?xml version="1.0"?>
<configuration>

  <system.web>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>

  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true">
      <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule,
System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
    </modules>
  </system.webServer>

  <system.serviceModel>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>
    <standardEndpoints>
      <webHttpEndpoint>
        <!-- Diğer servislerin help sayfalarının disable olmaması için ilk standartEndpoint
elementine dokunulmamalıdır-->
        <standardEndpoint name="" helpEnabled="true"
automaticFormatSelectionEnabled="true"/>
        <standardEndpoint name="EntranceServiceEndPoint"/><!-- Varsayılan olarak
helpEnabled özelliği false değere sahiptir-->
      </webHttpEndpoint>
    </standardEndpoints>
    <services>
      <service name="Lesson8.EntranceService">
        <endpoint contract="Lesson8.EntranceService" kind="webHttpEndpoint"
endpointConfiguration="EntranceServiceEndPoint"/>
      </service>
    </services>
  </system.serviceModel>

</configuration>
```

Zaten varsayılan **web.config** dosyası içeriğine göre, tüm servis talepleri otomatik olarak standart bir **Endpoint** tipine yönlendirilir. Ancak senaryomuza göre ana adres üzerinden yapılan yardım sayfası talebi geçersiz olmalı, diğerleri ile kullanılabilir durumda kalmalıdır. Bu nedenle **EntranceService** isimli hizmet için de bir **Endpoint** tanımlaması yapılmış ve **webHttpEndpoint**içerisindeki farklı bir ayara yönlendirilmiştir. Buradaki düzenlemeye göre **EntranceService** dışındaki tüm servislerin help sayfalarına ulaşılabilmektedir. Ancak **EntranceService** için help sayfası gösterilmemektedir. Buna göre **http://localhost:10843/CompanyServices/help** adresine bir talepte bulunulduğunda aşağıdaki ekran görüntüsü ile karşılaşılacaktır.

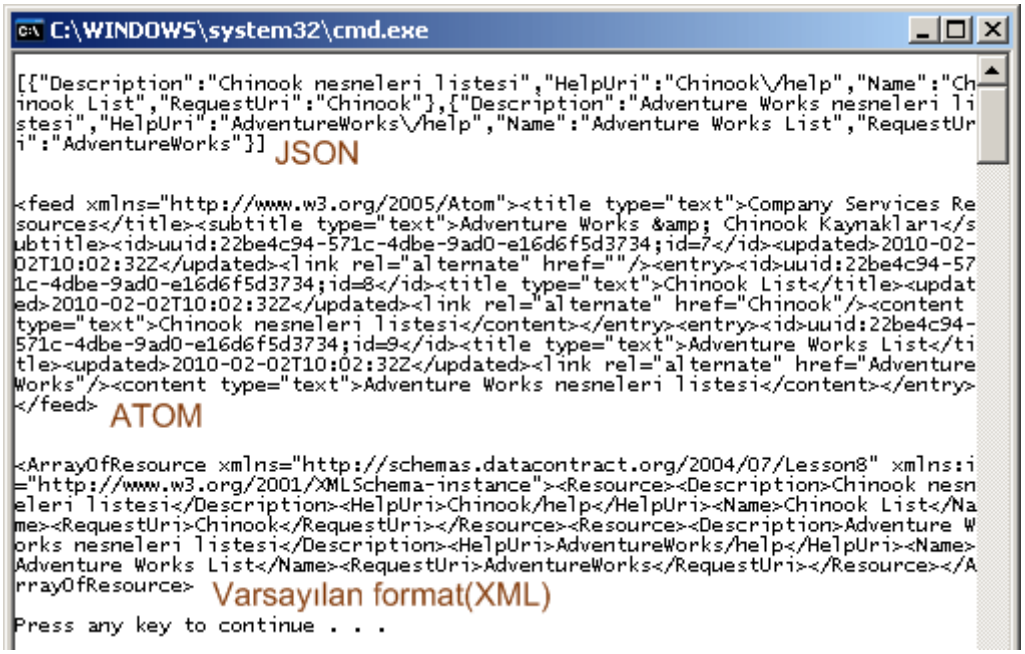


Servis tarafında pek çok işimizi hallettik. Ancak test etmemiz gereken bir husus daha var. İstemcinin, **XML** dışında **ATOM** veya **JSON** formatlı mesaj taleplerine karşılık olarak nasıl sonuçlar alacağı. Nitekim tarayıcı üzerinden yaptığımız taleplerde standart olarak **XML** çıktısı aldığımızı gördük ve biliyoruz. Peki ya diğer formatlar? Bu durumu test etmek için yine basit bir **Console** uygulaması geliştiriyor olacağız. Her zamanki gibi **HttpClient** tipinden yararlanacağız. Bu sebepten **REST Starter Kit** ile gelen **Microsoft.Http** ve**Microsoft.Http.Extensions Assembly** referanslarını eklemeyi unutmayalım. İşte **Console** uygulamamıza ait kodlarımız.

```
using System;
using Microsoft.Http;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         using (HttpClient client = new
HttpClient("http://localhost:10843/CompanyServices/"))
         {
            Execute(client,"application/json"); // Json formatında talep gönderilir
            Execute(client, "application/atom+xml"); // atom formatında talep gönderilir
            Execute(client, "noFormat"); // Olmayan bir format için talep gönderilir
         }
      }

      private static void Execute(HttpClient client,string acceptFormat)
      {
         // HttpRequestMessage nesnesi örneklenirken kullanılan ikinci parametreye göre
EntranceService tarafından karşılanacak bir talep oluşturulur
         using (HttpRequestMessage requestMessage = new
HttpRequestMessage("GET", String.Empty))
         {
            // Accept özelliğinin Add metodu yardımıyla Header' a eklenen bilgiye göre
hangi formatta mesaj istendiğin belirtilir.
```

```
            requestMessage.Headers.Accept.Add(acceptFormat);
            using (HttpResponseMessage responseMesssage = client.Send(requestMessage))
            {
               Console.WriteLine("\n{0}\n", responseMesssage.Content.ReadAsString());
            }
         }
      }
   }
}
```

Kodlarımızı çalıştırdığımızda aşağıdaki ekran görüntüsünde yer alan sonuçları elde ederiz.



Vuuuvvvv!!! Şu anda masamdaki şekerlerin oranına bakıyorum da...Baya bir tüketmişim. Artık dinlenmeye çekilmenin vakti geldi sanırım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Lesson8_RC.rar (181,52 kb) **[örnek Visual Studio 2010 Ultimate Beta 2 Sürümünde geliştirilmiş ancak RC sürümü üzerinde de test edilmiştir]**

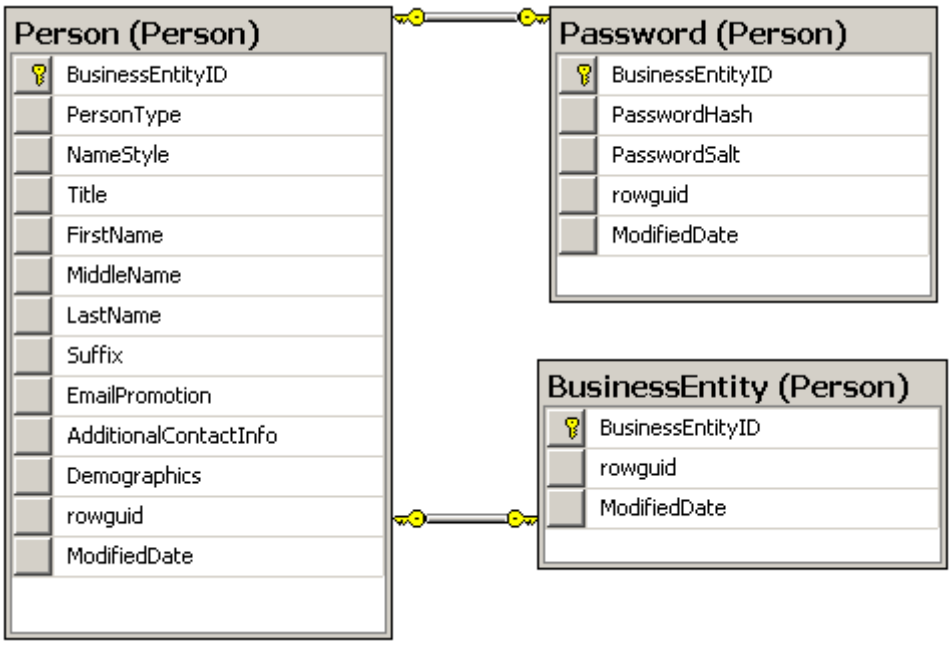
Entity Framework - Entity Bölünmesi (Splitting) (2010-04-05T11:15:00)

ado.net entity framework,entity,

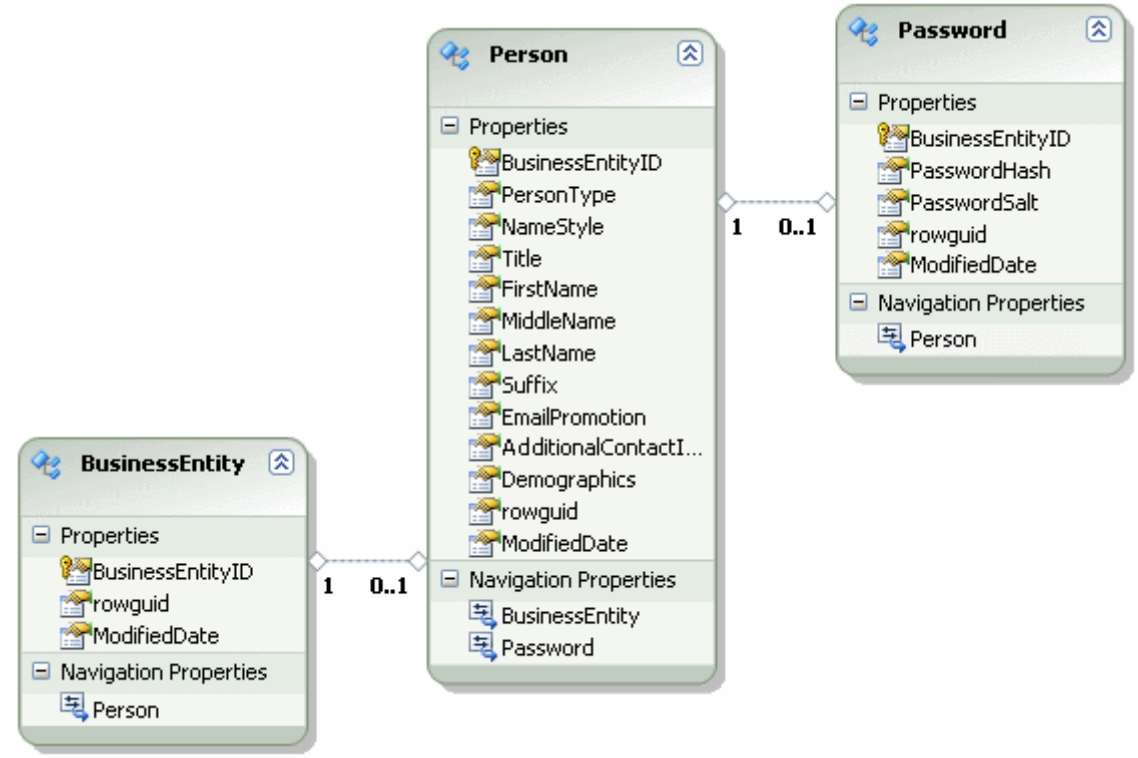
Merhaba Arkadaşlar,

Hız kesmeden **Ado.Net Entity Framework** ile ilişkili araştırmalarımızı paylaşmaya devam ediyoruz. Bu günkü yazımızda bir **Entity** tipinin, veritabanı tarafında birden fazla **Tablo'** yu işaret edebilmesi kabiliyetini incelemeye çalışacağız. Söz konusu duruma

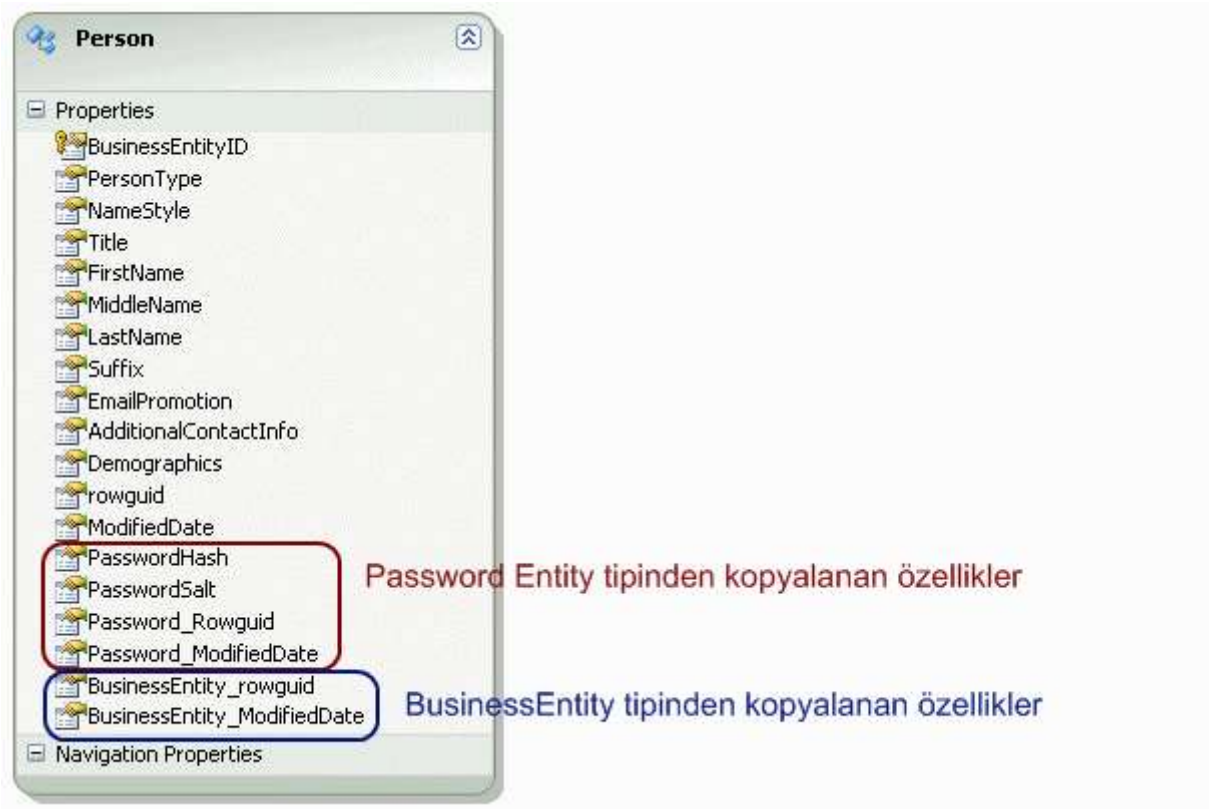
çok sık rastlanılmasa dahi, zaman zaman gereksinim duyulmaktadır. Konuyu daha kolay bir şekilde kavrayabilmek adına, yazıyı yazdığım tarih itibariyle sistemimde kurulu olan **Adventure Works 2008** veritabanında yer alan ve aşağıdaki diagramda görülen tablolarımızın olduğunu varsayalım*(Tabiki kendi sisteminizde yer alan örnek veritabanında benzer ilişikilere sahip başka tabloları da aynı teori içerisinde değerlendirebilirsiniz)*



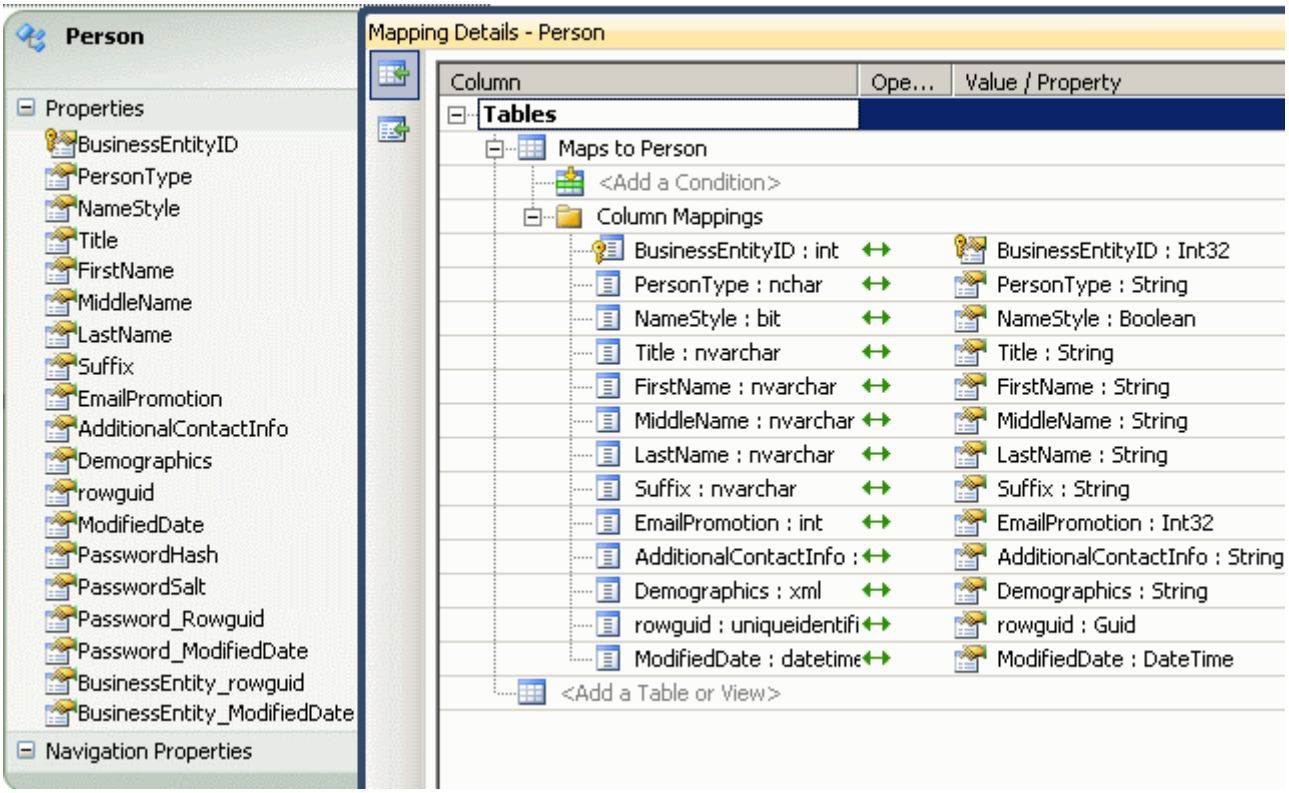
Person, Password ve **BusinessEntity** tabloları arasındaki ilişkileri değerlendirdiğimizde, **1 Person' a ait 1 Password satırı** ve yine **1 Person' a ait 1 BusinessEntity** satırı olabileceğini görmekteyiz. Aslında bu tabloların tamamı tek bir **Person** verisini ifade etmekte. Tabi veritabanı tarafında kavramsal olarak bu veri birden fazla tabloya ayrıştırılmış durumda. çok doğal olarak kod tarafına geçtiğimizde bu tabloların her biri için ayrı birer **Entity** tipi üretileceğine şahit olacağız. Aynen aşağıdaki **Entity Model Diagram'** da görüldüğü gibi.



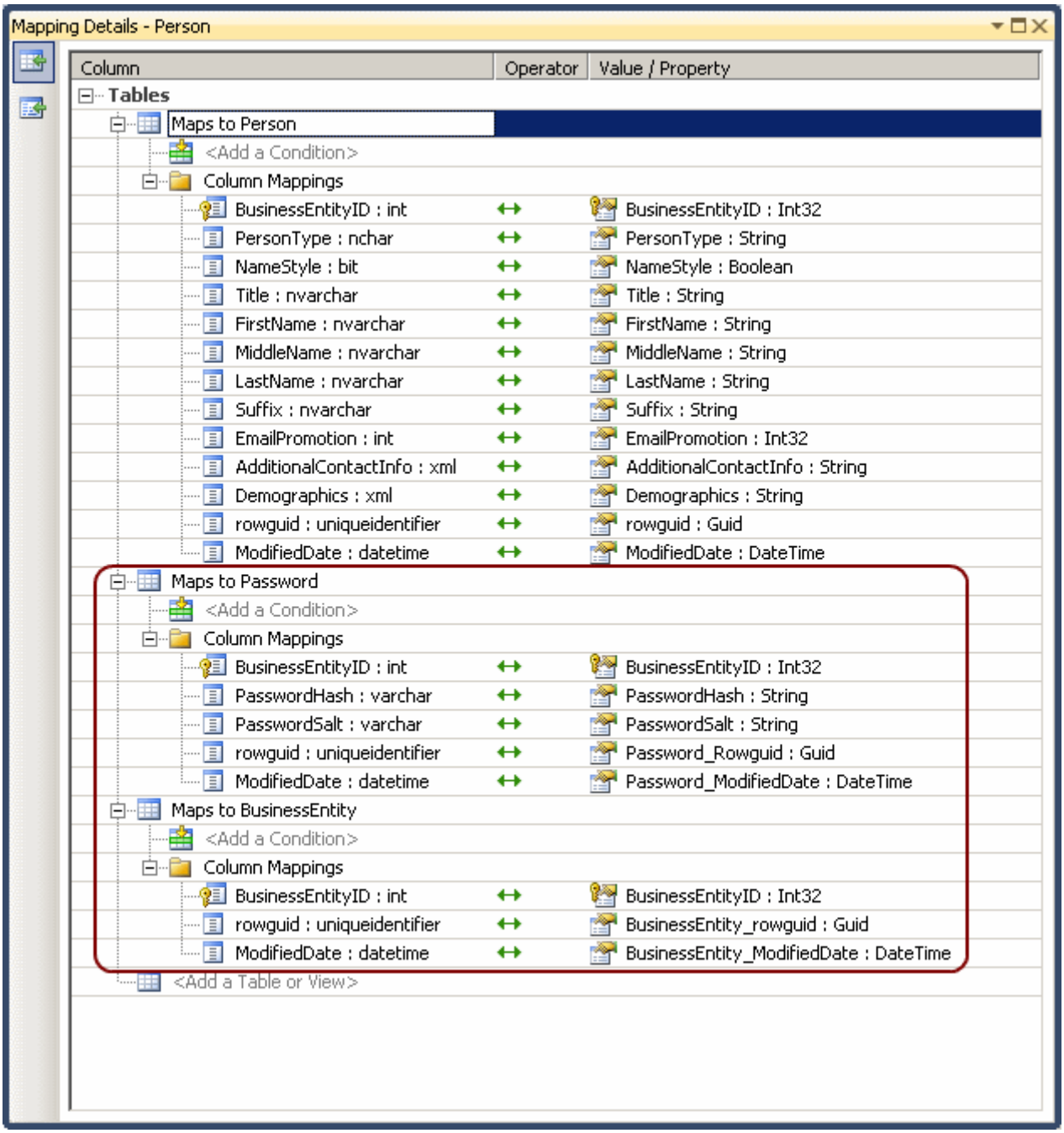
Tabi kod tarafında **Person** içeriğini değerlendirirken **Password** veya **BusinessEntity** içeriklerine de kolay bir şekilde erişebilmek gibi bir amacımız varsa eğer, **3 farklı Entity yerine 1 Entity**kullanmak daha avantajlı olabilir. özellikle kodlama zamanında yazılan **LINQ** sorgularında veya **CRUD** işlemlerinde. Peki bu 3 Entity' nin tek bir Entity olarak ele alınması için ne yapmak gerekmektedir? Son derece basit. **Cut-Paste** özelliklerinden faydalanılmalıdır 🙂 Bu anlamda, **BusinessEntity** ve **Password Entity** içeriklerine baktığımızda bizim için gerekli olan alanları kesip**Person Entity** tipi içerisine yapıştırmamız yeterlidir. Ne yazık ki bu örnekte yer alan **BusinessEntity** tablosunun içeriğinde çok faydalı bilgiler bulunmamaktadır. Ama yinede konunun kavranabilmesi açısından değerlendirilmiştir. Bu işlemleri gerçekleştirirken dikkat edilmesi gereken noktalaradan biriside, aynı isimli **Property'** lerin kopyalanması sonrasında, son ek olarak **_1** gibi sayısal bir isimlendirmenin söz konusu olmasıdır. Tabiki bu tip özellikleri tekrardan isimlendirmekte yarar vardır. Gerekli kesme ve yapıştırma işlemleri tamamlandıktan sonra Password ve BusinessEntity tipleri diagramdan silinebilir. Yine bu işlem sırasında dikkat edilmesi gereken bir noktada bize sorulan soruya No cevabını vermektir.*(Sorunun ne olduğunu söylemeyeceğim, örneği yaparken görmenizi istiyorum.* 🙂 *)* Sonuç olarak diagramın yeni hali aşağıdaki gibi olacaktır.



Yeterli mi? Elbette değil. **Designer'** da yapmış olduğumuz taşıma işlemlerine rağmen, **Mapping Details** içeriğinde halen daha eksiklikler bulunmaktadır. Aşağıdaki şekilde bu durum net bir şekilde görülebilir.



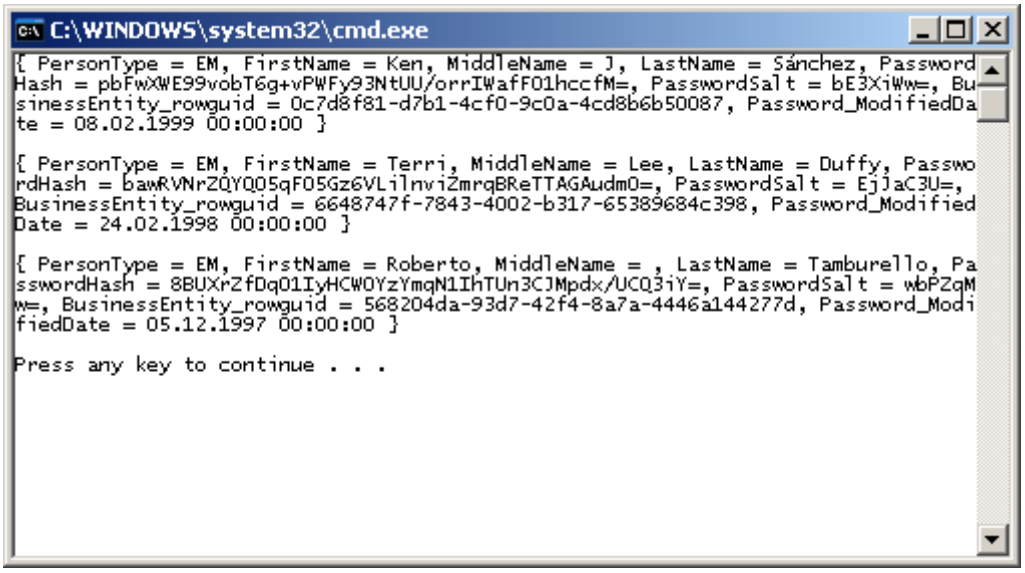
Eksik olan kısım şudur; kopyalanan özelliklerin hangi tablolardaki hangi alanları işaret ettiği belli değildir. Dolayısıyla **Mapping Details** kısmında gerekli düzenlemeleri yaparak aşağıdaki şekilde görülen eşleştirmeleri tamamlamamız gerekir.



Buna göre **Person Entity** tipinin **Person** tablosu dışında **BusinessEntity** ve **Password** tablolarını da işaret ettiği belirtilmiş olur. Gerekli **Field-Property** eşleştirmeleri yapıldıktan sonra*(örneğin rowguid' in BusinessEntity_RowGuid ile eşleşmesi gibi)* uygulama derlenip örnek bir kodun denenmesi için gerekli işlemlere başlanabilir. Bu amaçla **Main** metodu içerisinde aşağıdaki test kodunu yazdığımızı düşünelim.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Splitting
{
   class Program
   {
      static void Main(string[] args)
      {
         using (AdventureWorks2008Entities entities = new AdventureWorks2008Entities())
         {
            var result = (from p in entities.People
                          select new
                          {
                             p.PersonType,
                             p.FirstName,
                             p.MiddleName,
                             p.LastName,
                             p.PasswordHash,
                             p.PasswordSalt,
                             p.BusinessEntity_rowguid,
                             p.Password_ModifiedDate
                          }).Take(3);

            foreach (var r in result)
            {
               Console.WriteLine(r.ToString()+"\n");
            }
         }
      }
   }
}
```

örneğimizde ilk 3 satıra ait alanlardan bazılarının çekildiği **isimsiz tip(Anonymous Type)** kullanımı söz konusudur. Dikkat edilmesi gereken nokta ise tek bir **Entity** nesnesi üzerinden, **Person, Password** ve **BusinessEntity** tablolarının ilgili alanlarına ulaşılabiliyor olmasıdır ki bu **Splitting** özelliğinin bir sonucudur. Buna göre örneğimizin çalışma zamanı çıktısı aşağıadaki gibi olacaktır.



Bu kabiliyetin uygulanışı sırasında en çok dikkat edilmesi gereken nokta arka planda çalıştırılan **SQL** sorgularıdır. Nitekim birden fazla Tablonun tek bir **Entity** içerisinde birleştirilmesi arka planda **Inner Join** sorgularının atılmasına neden olacaktır. Yukarıdaki örneğin çalışma zamanı sırasında **SQL** tarafında icra edilen sorgudan bu durum net bir şekilde görülebilir.

```
SELECT TOP (3)
[Extent1].[BusinessEntityID] AS [BusinessEntityID],
[Extent3].[PersonType] AS [PersonType],
[Extent3].[FirstName] AS [FirstName],
[Extent3].[MiddleName] AS [MiddleName],
[Extent3].[LastName] AS [LastName],
[Extent2].[PasswordHash] AS [PasswordHash],
[Extent2].[PasswordSalt] AS [PasswordSalt],
[Extent1].[rowguid] AS [rowguid],
[Extent2].[ModifiedDate] AS [ModifiedDate]
FROM   [Person].[BusinessEntity] AS [Extent1]
INNER JOIN [Person].[Password] AS [Extent2] ON [Extent1].[BusinessEntityID] = [Extent2].[BusinessEntityID]
INNER JOIN [Person].[Person] AS [Extent3] ON [Extent1].[BusinessEntityID] = [Extent3].[BusinessEntityID]
```

Dolayısıyla bu çalışma şekli dikkate alınarak gerçekten gereksinim var ise **Entity** birleştirmesi yolu tercih edilmelidir. öyleki **Entity** tiplerinin ayrık olarak kalmaları tercih edilebilir. Söz gelimi geliştirdiğimiz örnekte **BusinessEntity** tablosundan alınan alanların program kodu içerisinde hiç bir yerde kullanılmadığı bir durumda, bu tablo alanlarının **Person Entity** tipi içerisine alınması anlamsızdır. Nitekim, performans açısından gereksiz **Inner Join** sorgusu atılmasına neden olmaktadır.

Sonuç olarak **Splitting** özelliği yardımıyla bir **Entity** içerisinde yer alan **özelliklerin(Property),** veritabanı tarafında birden fazla **Tablo' nun alanlarına(Fields)** ait olması sağlanabilmektedir. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Splitting_RC.rar (49,90 kb) **[örnek Visual Studio 2010 Ultimate RC sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Yazılımcı için İşsizlik (2010-04-02T10:00:00)

Merhaba Arkadaşlar,

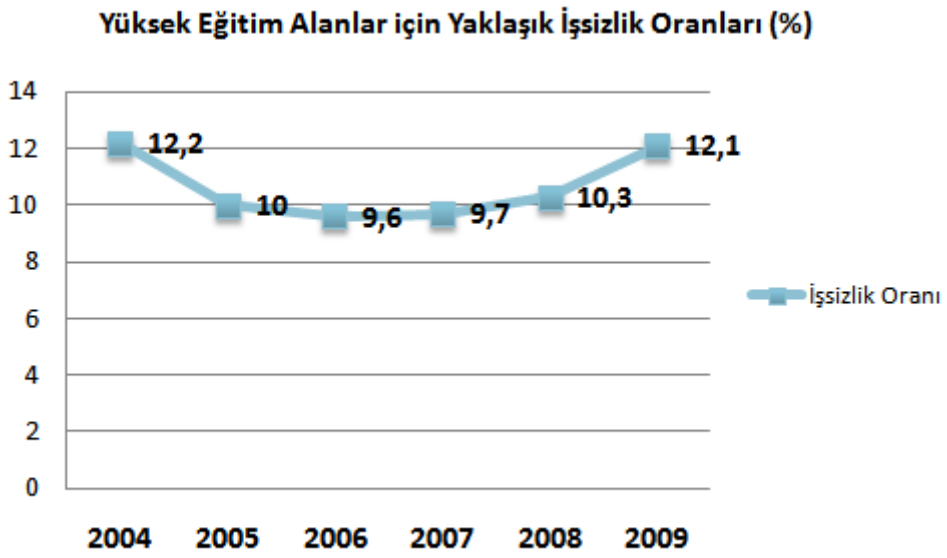
üniversiteden yeni mezun veya bitirmekte olan ve kariyerinde bilgisayar yazılım teknolojileri üzerine ilerlemek isteyen pek çok genç arkadaşımız için en büyük kabuslardan birisi de iş bulmaktır. Ne yazık ki global kriz, ülkemizin içinde bulunduğu şartlar ve daha pek çok nedenden ötürü ilk kaygımız **"doğru işi bulmak"** yerine, **"bir iş bulabilmek"** olmaktadır. Hatta çoğu zaman herhangibir iş bulmak kadar kötümser bir senaryo da söz konusu olabilir ki ben bu yazımızda bu konuya değinmeyi düşünmüyorum. Şu andan itibaren yazılım alanında ilerlemek istediğimiz konusunda hem fikir olduğumuzu ve buna göre bir iş bulma kaygısını taşıdığımızı düşünerek ilerlememiz daha doğru olacaktır.

Matematik Mühendisliği mezunu bir birey olarak ispatlara hayatımın büyük bir bölümünü adamış olsam da, her zaman için istatistiki verilerin gerçek hayat problemlerinin çözümünde önemli rol aldığını düşünmüşümdür. Amerikalıların spor organizasyonlarını seyredenler *(özellikle benim gibi NBA maçlarını takip edenler)* olur olmadık istatistik veriler yayınladıklarını bilirler. örneğin **"Takımının geriden gelerek öne geçtiği bölümlerde, bench' ten gelen oyunculardan boyalı bölgeden yaptığı iki sayılık atışlarda en yüksek başarı yüzdesi gösteren power forward' lerin bir sezonluk veya o an oynanamakta olan maç için anlık istatistiki verisini almak"** gibi...

Aslında Amerikalılar bu tip istatistikleri çıkartmakta çok ta haksız sayılmazlar. çünkü meseleyi kavrayabilmenin en iyi yollarından birisi sayısal değerlere dayanan istatistiki

verileri incelemekle mümkün olacaktır. Bu şekilde yanlış veya doğru giden pek çok faktörü görme ve tedbir alma şansı bulunmaktadır.

İtiraf etmeliyim ki **MBA Yüksek Lisans** projemi verdiğim dönemden bu yana yıllardır istatistik verileri incelememiştim. özellikle o zamanki kaynaklara kolayca ulaşma sıkıntısı göz önüne alındığında **Google** gibi arama motorlarının ve istatistiki rapor sağlayan veri ambarlarının ne kadar önemli olduklarını bir kere daha görmüş oldum. Şu an için sizlere çok detaylı istatistik veri sunmayacağım. Ancak basit bir istatistiğinde durumumuzu görmek açısından önemli bir etken olacağı kanısındayım. Bu amaçla **Türkiye İstatistik Kurumundan**aldığım son bir kaç yılın işsizlik verilerini incelediğimi ve aşağıdaki grafikte görülen tablonun oluştuğunu söyleyebilirim.



Not : *Türkiye İstatistik Kurumunun diğer istatistikleri için http://www.tuik.gov.tr*

Buna göre örneğin **2009** yılında Yüksek öğrenim gören **her 100 kişiden 12,1**' i *(ki biz buna iyimser olarak 12 kişi diyebiliriz)* işsizdir. Bu aslında çok yüksek bir oranmış gibi görünmeyebilir ancak bu istatistiki veride mesleki bir ayrımın olmadığı belirtmek isterim. Peki biz bu yazımızda böyle istatistiklerle mi boğuşacağız? Hayır...Ancak durum ortada. İşsizlik problemi var ve bu, genç bir yazılımcı adayı için çağın hastalıkları arasında sayılan Stres' ten daha büyük bir sorun olarak görülebilir. Dolayısıyla bu gün yazımızdaki konumuz şu olacaktır;

"İş arayan ve henüz bulamayan yeni mezun genç bir yazılımcı adayı, vaktini nasıl değerlendirebilir/değerlendirmelidir?"

İşe ilk olarak işsizliğin ne gibi problemlere yol açabileceğinden bahsederek başlamakta yarar vardır. Kendimizi en kötü senaryoya hazırlamanın yararı olabilir. Tabi geleceği umutla bakmamızı, son raddeye kadar mücadele etmemizi engellememelidir. Dilerseniz bir kaç madde ile içinde bulunabileceğimiz durumu irdelemeye çalışalım.

Donanımsal ve Yazılımsal Gereksinimler

Açıkça itiraf etmek gerekirse bilgisayar pahalı bir icattır. Hele ki **Apple** tarafında iseniz el yakan fiyatlara hazırlıklı olmalısınız. Her ne kadar günümüzde sayısız kredi kartı kampanyası ve taksitli alış imkanı olsa da, yazılımcıların bazı bilgileri öğrenebilmesi için iyi donanımsal özelliklere sahip bir bilgisayarı tercih etmeleri duruma göre gerekbilir. Şirketler bu ihtiyaçları çoğunlukla karşılarlar. Ancak sorun henüz iş bulamamış olmamızdır. örneğin şu sıralarda **.Net Framework** platformu ile paralel programlama konusuna eğilen bir yazılımcının çalışacağı bir bilgisayarın en az çift çekirdek işlemcili olması şarttır. Kaldıki, yeni mezun olan veya son sınıfını okuyan bir öğrencinin biriktirdiği para ile iyi donanımsal özelliklere sahip bir bilgisayar alması her zaman mümkün olmayabilir. Ancak bu noktada üniversitelerin öğrencilere sağladığı laboratuvar ortamlarının önemi ortaya çıkmaktadır. üniversitenin bizlere sunduğu bu ortamları sonuna kadar kullanmak gerekir.

Yaşananlardan : Y.T.ü. Matematik Mühendisliği *bölümünü* ***1993*** *yılında kazandım. O yıllarda üniversite laboratuvarları yenilenmek üzereydi. Bilgisayar konulu ilk dersimizde elimizde* ***3.5" lik HD*** *disketler olduğunu hatırlıyorum.* ***Ms DOS*** *işletim sistemi yüklü bilgisayarlar söz konusuydu. Hevesli bir öğrenci olarak* ***286****'dan* ***368*** *tabanlı işlemcilere geçirilen bilgisayarlarda vakit harcamak için fırsat kolluyordum. Ne varki o yıllarda labarotuvarlarda çalışmanın belirli saatleri ve kuralları olurdu. öyle saatlerce kalmak, istediğiniz zaman girip çıkmak veya yaptığınız çalışmaları* ***USB*** *gibi mucizevi aygıtlar ile eve götürmeniz de mümkün değildi. Her şeyden önce sistemi açmak için diskete yüklü olan işletim sistemini kullanmalıydınız. Nitekim çoğu bilgisayarda* ***Hard Disk*** *yoktu.Ancak artık durumun değiştiğini biliyorum.*

Sakın o bilgisayarları küçük görerek **"Bunlarda WPF çalışmaz, Visual Studio yok, 3D uygulama geliştiremem, az Ram var, yavaş bunlar vs..."**demeyiniz. **Commodore** bilgisayarlarda **64 Kb** alan içerisinde o herkesin başından kalkmadan saatlerce oynadığı oyunların geliştirildiğini hatırlayınız. **.Net Framework** uygulamarı geliştirmek için mutlaka **Visual Studio** ya sahip olmanız gerekmediğini biliniz. **SharpDevelop** gibi 3ncü parti araçların olduğunu, en kötü ihtimalle **.Net Framework** ve **Notepad**' in yeterli olacağını düşününüz. Yetmedi mi? çok

mu kötü bir konfigurasyon var? Neden **C, C++** kullanarak **driver**yazmayı öğrenmeye çalışmıyorsunuz? Şöyle bir düşünün. Etrafınızda tanıdığınız kaç kişi var; donanımlar için driver yazacak derecede iyi bilgisi olan veya işletim sistemlerini yamayacak kadar **C++** döktüren...Buna göre şu sonuca varabiliriz; üniversiteki ortamın sizlere sağladığı bilgisayarların konfigurasyonu kendinizi geliştirmeniz ve işe hazırlanmanız açısından sorun değildir. Eldeki imkanları etkin bir şekilde kullanabilme yeteneğini kazanma bu düşük konfigurasyonlu sistemlerden çıkartabileceğiniz en önemli derstir.

 **Eğitim Materyalleri**

İşsiz kalan birisinin veya halen daha iş aramakta olan birisinin yeni bir şeyler öğrenme isteğini çeşitli eğitim materyalleri ile desteklemesi önemlidir. Ne yazık ki ülkemizde yazılan yerli kitapların çoğu kişisel görüşüme göre yetersiz ve vasat. Her şeyden önce büyük puntolar ile yazılıp sayfa sayısı fazla tutulan kitapların aslında boş yere ağaç kesimine neden olduğunu düşünüyorum. Bazı kitaplar ne yazık ki dış kaynaklı kitapların bire bir çevirisi olurken, bazıları **MSDN**' i Türkçeleştirmekten başka bir içerik sunmuyor. Dolayısıyla her zamanki gibi dış kaynakların çok daha iyi olduklarını ifade etmek istiyorum. Bu noktada yazılım sevdalılarının vazgeçilmeleri arasında yer alan **korsan PDF** kitapları kesinlikle uygun bulmuyorum. Bunun yerine online olarak okunabilen **PDF** sunan ve bunun için çok yüksek ücretler talep etmeyen **Wrox, APress, O'Reilly, Addison Wesley** vb... yayın evlerini tercih etmenizi öneriyorum.

çok doğal olarak **Amazon** gibi siteler üzerinden ülkemize kitap getirtmenin yüklü maliyetleri olmakta. özellikle en hızlı gönderi seçeneğini işaretlerseniz. Bu nedenle belki bir kaç arkadaş bir araya gelerek birden fazla kitap sipariş edip, en azından taşıma ücretini eritebilirsiniz kanısındayım. Tabi mali duruma göre en isabetli kaynağın seçilmesi çok ama çok önemli. öyleki,**Amazon** sitesine girip belirli bir konu üzerine yazılım kitaplarını aradığınızda onlarca sayfa ile karşılaşılmakta. Bu nedenle seçim yapmak için işaretlenmiş yıldız değerlerine ve kitaplar ile ilişkili yorumlara bakmak, kitap içeriklerini iyi bir şekilde etüt etmek gerekmekte.

Tabi şunu unutmamakta yarar var. Halen daha öğrenciysek, üniversitelerin kütüphanelerini mutlaka değerlendirmeliyiz. Her ne kadar akademik ağırlıklı yayınlar çoğunlukta olsa da en azından nesne yönelimli bir dilin anlatıldığı kaynaklara mutlaka ulaşabilirsiniz. Hatta diğer üniversitelerin kütüphanelerini misafir öğrenci olarak ziyaret dahi edebilirisiniz.

Yaşananlardan : *Bizitek firmasına çalışmaya başladığım sırada iş arkadaşlarımdan birisinin elinde* ***Visual C#*** *ile ilişkili* ***Microsoft Press'*** *in bir yayınını gördüğümü hatırlıyorum. Şaşırtıcı ama bu yayın* ***İstanbul Teknik üniversitesi*** *kütüphanesine aitti.*

üstelik orjinal basımdı. ***Amazon****' dan kitap almaya gerek var mıydı? Kaynaklar zaten kütüphanedeydi.*

Tabi eğitim materyalleri arasında sayabileceğimiz en önemli malzemelerden birisi de görsel içerikli olanlarıdır. Bildiğiniz üzere bir resim onlarca kelimeden oluşan bir felsefeyi tek seferde ifade edebilir. çok doğal olarak interaktif içerikli bir yayının zihinde daha kalıcı olması söz konusudur. Bu anlamda **Channel9, NedirTv?** ve benzeri toplulukların sunduğu kaliteli ve öğretici içeriklerden yararlanılabilir. özellikle **How To** ile başlayan **Screencast**' ler başlangıç için idealdir. Diğer yandan **Webcast(Webiner)** yayınları da işsizlik dönemlerinde oldukça yararlıdır. Bu tip oturumlarda dünyanın öteki ucundaki uzmanları dinleme ve en önemlisi canlı olarak soru sorma şansınız bulunmaktadır. **Microsoft,** özellikle yazılım tarafındaki **Webcast, Screencast** gibi sayısız interaktif materyali ile bizlere çok önemli bir veri evrenini, ücretsiz olarak vermektedir.

 **Yazılım Firmalarının öğrenciler için Sundukları**

Staj dönemleri çok ama çok önemlidir. Tabi eğer öğrenci olarak staj yeri ayarlanabilirse. Nitekim günümüzde iş bulmak ne kadar zor ise, staj yeri ayarlamakta o derece zor. Ancak staj yeri ayarladıysanız kıymetini bilmenizde ve gittiğiniz yerde sizinle ilgilenen kimse yoksa dahi sonuna kadar durmanızda ve hatta kendinize iş çıkartmanızda yarar var. Mutlaka bulunduğunuz ortamda dikkatinizi çeken bir eksiklik olacaktır. Pek çok çalışanın vakit ayıramadığı/ayırmadığı için boşta kalan ama yapılırsa o şirkete katma değer sunan bir şeyler...Belki şirketin kullandığı yazılım için küçük bir ek modül...Yazamasanız dahi fikrinizi sahiplenip, olgunlaştırmanız, sunmanız ve karşınızdakileri kabullenmeleri için ikna edebilmeniz yeterlidir. Yeterliden de ötedir. Tarih boyunca çoğu staj hepimize anlamsız gelmiştir. Oysaki yıllar sonra bu staj hikayeleri **CV**' lerde gayet hoş duracaktır. Heleki yaptıklarınızı anlatacağınız ilk iş görüşmenizde. Bu nedenle öğrencilik zamanında yapılan stajların göz ardı edilmemesi gerekir. özellikle bir yazılım firmasında yapılıyorsa **part-time** çalışmak için yalvarılması bile düşünülebilir. Ancak asla ve asla stajınızı naylon torba haline getirmeyin. Geleceğinizin söz konusu olduğunu unutmayın ve işsizlik oranlarını hatırlayın.

 **Internet**

Internet' in son on yıllık zaman dilimi içerisinde geldiği nokta düşünüldüğünde bilgiye ulaşmanın en kolay ve hızlı yolu olduğu sonucuna varabiliyoruz. Ancak bir yazılımcı için **Internet** çok tehlikeli bir yerde olabilir. Bunun en büyük nedenlerinden birisi çok fazla bilgi olmasıdır. Hatta zaman zaman bu durum, **"Bilgi çöplüğü"** olarakta yorumlanabilir, yorumlanmaktadır. Dolayısıyla ilerlenmek istenen yola ilişkin olarak doğru ve güvenilir adreslere ulaşmakta ve bu noktalardaki kaynakları takip etmekte yarar vardır. İşsiz kalan bir yazılımcı, bilgisayarı olmasa dahi herhangibir **Internet Cafe**' den dünyanın diğer ucundaki sunucuda duran makaleye ulaşabilir, okuyabilir ve gerekli bilgileri alabilir. Eğer öğrencilik devam ediyorsa elbetteki **Internet Cafe**' lerde para harcamak yerine*(ki bunun pek çok yolu vardır, Chat yapmak, Facebook' ta dolanmak, Twittlemek, Oyun oynamak vs...)* üniversitenin laboratuvar ortamını kullanmak düşünülebilir.

Tabi internet üzerinden kendinize **Freelance** işlerde bulabilirsiniz. Bu konu ile ilişkili pek çok site mevcuttur. Söz konusu sitelerde proje sahipleri, taleplerini ve nasıl bir yazılımcı aradıklarını, işin ne kadar sürede yapılması gerektiğini ücreti ile birlikte açık bir şekilde ifade ederler. önce bir işle başlarsınız. Ancak başarılı oldukça alacağınız proje sayısı artar. Hatta kendinize ait bir fikri geliştirmek için gerekli maddi desteği dahi bulabilir istediğini yazılım geliştirme araçlarını satın alabilir veya yazdırtabilirsiniz.

Yaşananlardan : *çok yakın bir meslektaşım dışarıdan* ***Freelance*** *iş yaparak bir iş yerine bağlı olmadan uzun yıllar çalışabilmeyi ve iyi kazanmayı başarmıştır. üstelik yüzün üzerinde başarılı projeye de imza atmıştır. Arkadaşım o kadar çok projede görev almıştı ki neredeyse her konuşmamızda ağzından yeni bir fikir çıkar veya benim önerdiğim fikirlerin neden olamayacağını ya da olması için ne gerektiğini, nelere dikkat edileceğini söylerdi.*

Proje tecrübesini iş başvurularında gösterebilmek çok önemlidir. İş verenler tecrübeye, proje çeşitlerine, karşılaşılan zorluklara bakarak daha iyi karar verebilir. Ayrıca adayın şansı proje tecrübesine bağlı olaraktan çok daha fazla olur. İşsiz olan birisi evde miskin miskin otururken, **"İyi de nasıl bir tecrübe sunacağım ki?"** dememeli, az önce bahsettiğim**Freelance** sitelerden yararlanarak başlangıç için gerekli proje tecrübesini tamamlamaya uğraşmalıdır.

Takip Edilmesi Gereken Blog Listesi(Sizin OPML içeriğiniz)

Günümüzün bilgi sunumu açısından en önemli kaynakları arasında **Blog'** lar yer almaktadır. Bundan yıllar önce **Community**' ler ne kadar ön plana çıktıysa, son yıllarda **Blog** siteleri aynı önem derecesinde ön plana çıkmaktadır. özellikle **Blog**' ların **RSS, Atom** gibi formatlarda **Feed** içeriği sunmaları nedeniyle, kişisel bilgisayarlarımızdan, **PDA** cihazlardan ve cep telefonlarından kolayca takip edilmeleri söz konusudur. İş arayan ve işsiz bir halde evde oturan birisinin her gün düzenli olarak yapması gerekenler arasında, günde en az iki kere diş fırçalamak, üç öğün yemek yemek, aşırı tatlı/tuzlu yiyeceklerden uzak durmak, gazete okumak, uyumak ve liste olarak tuttuğu Blog' ları takip etmek yer alır. Tabi blog deyince yazılım sektörünün öncüleri arasında yer alan isimlerin takip edilmesinde yarar vardır. Sadece teknik konuların ele alındığı bloglar ile yetinmemek gerekir. ürün grupları, pazarlama stratejileri, gelecek vizyonları ile ilişkili olarak yazılarını yayınlayan bireylerin Blog girdilerini de takip etmek önemlidir. Bu aşamada **Feedreader** gibi araçlardan yararlanarak n sayıda **Blog**' u tek bir uygulama içerisinden takip edebileceğinizi belirtmek isterim. Başlangıç için size bir **OPML(Outline Processor Markup Language)** listesi vermemi ister misiniz?

feedreader.opml (17,38 kb)

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Object vs Dynamic (2010-04-01T00:10:00)

dynamic language runtime,c# 4.0,c# 3.0,c#,

 Merhaba Arkadaşlar,

Ayrıntılar detaylarda saklıdır. Bu cümleyi çok severim. Sevdiğim özlü sözler arasındadır. Gerçel bir nesnenin ne kadar kaliteli olduğunu anlamak için detaylarına bakmak gerekir. İşçiliğine, kullanılan malzemeye, malzemelerin uyumuna vs...Hatta benzer diğerleri ile olan kalite farkını anlamak için bile. çok doğal olarak yazılım dünyasında da bir takım konuların anlaşılabilmesi, kavranabilmesi, benzerleri ile olan farklarının irdelenebilmesi için mutlaka detaylara bakmak, ama sıkılmadan bakmak gerekir. Aynen bu günkü yazımızda yapacağımız gibi.

Bu yazımızda **Dynamic Language Runtime** kullanımında büyük öneme sahip olan **dynamic** ile **.Net Framework'** ün ilk çıktığı zamandan beri var olan **Object** tipi arasındaki farklılıkları görmeye çalışacağız. Bunun için kod tarafında biraz daha detaya

girmemiz gerekecek. çok derin değil belki de ama aradaki farklılıkları çıkartabilmek adına önemli detaylar. Başlamadan önce örneklerimizi **Visual Studio 2010 Ultimate RC** sürümünde geliştirdiğimizi ve ilerleyen sürümlerde farklılıklar olabileceğini hatırlatmak isterim.

öncelikli olarak işe aşağıdaki basit kod parçası ve **Object** tipini ele alarak başlayalım.

Case 1;

```
object pi = Math.PI;
Console.WriteLine(pi.GetType().ToString());
```

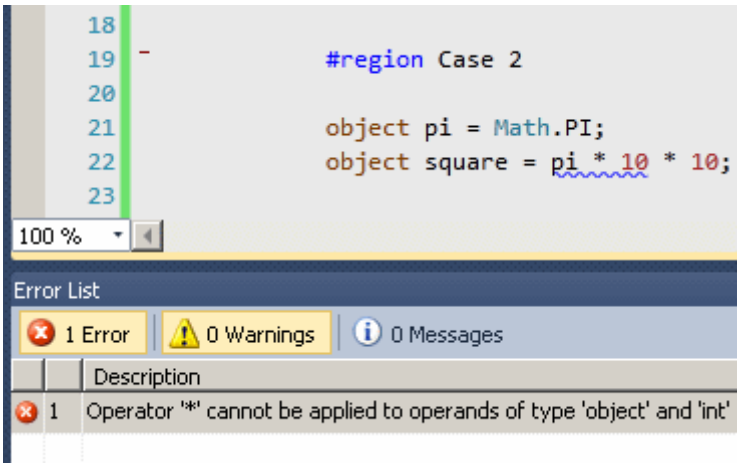
İlk satırda **Math.PI** sabit değerinin **object** tipine atandığını görüyoruz. Aslında bilinçsiz olarak bir tür dönüşümü söz konusu**(Implicitly Type Casting)**. Burada herhangibir sıkıntı yok. **pi.GetType** satırının çalışma zamanı çıktısı ise **System.Double** olmalıdır. Nitekim eşitliğin sağ tarafından gelen **Math.PI, double** tipinden olduğu için **pi**isimli **object** tipi üzerinden ele alınsa da kendi tipini taşımış olmaktadır.

Şimdi kodumuzu aşağıdaki gibi değiştirdiğimizi düşünelim.

Case 2;

```
object pi = Math.PI;
object square = pi * 10 * 10;
```

Bu kez **object** tipinden olan **pi** ile iki sayısal değeri(ki bunlarda int tipindendir) matematiksel bir işleme tabi tutup sonucu yine bir **object** tipine atamaya çalışıyoruz. Ancak kodu derlediğimizde **Compiler'**ın bir derleme zamanı hata mesajı ürettiğini görürüz.



Hımmm...Aslında bu son derece doğal bir sonuç. Nitekim derleme zamanında **pi** değişkeninin tipi **object** ve biz **object** tipi ile **int** tiplerini işleme sokmaya çalışıyoruz. Burada çözüm, dönüştürme işlemini bilinçli olarak yapmaktan ibaret**(Explicitly Type Casting)**. Yani kodu aşağıdaki gibi düzenlemekten;

Case 3;

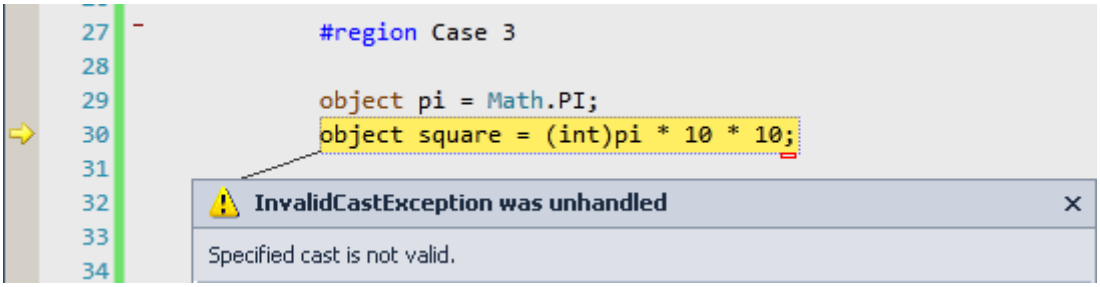
```
object pi = Math.PI;
object square = (double)pi * 10 * 10;
```

Dikkat edileceği üzere **pi** değişkeni bilinçli olarak **double** tipine dönüştürülmüş durumdadır. Aslında az önceki senaryomuzda **pi'** nin taşıdığı tipin **double** olduğunu görmüştük. Yine de başka tipler ile işleme tabi tuttuğumuzda derleme zamanı hatası ile karşılaşmaktan kurtulamadık. çünkü **pi** değişkeni **double** tipten değer taşıyan bir**object** idi aslında.

Şimdi durumu biraz daha entersan bir hale getireceğiz. Aşağıdaki kod parçasını göz önüne alın ve kodu kafanızda derleyip hata olup olmadığını söyleyin. Sonrasında ise çalışma zamanında bir hata oluşup oluşmayacağını düşünün ve bir karar daha verin. En sonunda ise bunu, konu ile ilgili yakın arkadaşlarınızla tartışın

```
object pi = Math.PI;
object square = (int)pi * 10 * 10;
```

Derleme zamanında herhangibir hata mesajı alınmayacaktır. Ancak **çalışma zamanına(Runtime)** geçtiğimizde aşağıdaki ekran görüntüsünde yer alan **istisna(Exception)** ile karşılaşırız.



Upsss!!!

Sorun şudur; **pi** değişkeni **object** tipinden tanımlanmıştır ve eşitliğin sağ tarafına göre double tipinden bir değer taşımaktadır. Ancak bu tip **cast** operatörüne göre **int** tipine dönüştürülmeye çalışılmaktadır. Oysaki çalışma zamanının burada beklediği tam olarak **double** tipine dönüştürme işlemidir.

*Dolayısıyla **object** tipini kullandığımız bu senaryoda matematiksel işlemlerin yapılabilmesi için, mutlaka doğru tipe dönüşüm gerçekleştirilmelidir. Eğer bir dönüştürme işlemi yapmassak, derleme zamanında hata mesajı alırız. Ancak tip dönüşümü yaparkende*

koltuğumuzda rahat edemeyiz, çünkü yanlış tipe dönüştürme işlemleri çalışma zamanı istisnaları ile cezalandırılır. Buna göre gerçekten beklenen tipe dönüşüm işlemi sağlanmalıdır.

Gelelim yeni gözdemiz olan **Dynamic** tipe. Yukarıdaki senaryoları bire bir, ama bu kez **dynamic** tipini kullanarak değerlendireceğiz.

```
dynamic pi = Math.PI;
Console.WriteLine(pi.GetType().ToString());
```

Bu kez eşitliğin sol tarafında **dynamic** tipi vardır. **object** tipinin kullanıldığı örnektekine**(Case 1)** benzer olaraktan **pi** değişkeni yine **double** tipinden bir değer taşımaktadır. çalışma zamanının üreteceği sonuçta aynı olacaktır. Ancak aşağıdaki kod parçasını göz önüne aldığımızda,

```
dynamic pi = Math.PI;
dynamic square = pi * 10 * 10;
```

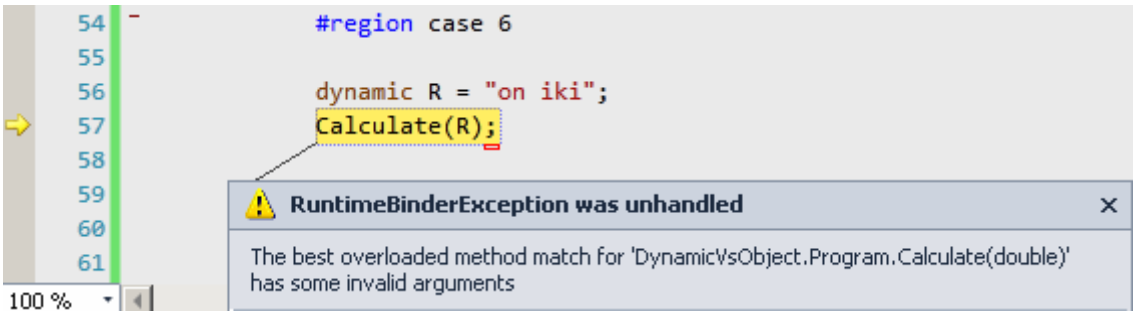
object tipinin kullandığımız örneğe**(Case 2)** baktığımızda derleme zamanında bir hata aldığımızı hatırlıyoruzdur sanırım. Oysaki **dynamic** tipi derleme zamanı ile ilgilenmemektedir. çalışma zamanında ise **pi**' yi gereken tipte ele almaktadır. Zaten **object** tipini kullandığımız üçüncü senaryomuzu ele almamıza da gerek kalmamıştır. 😉

*Buna göre şöyle bir sonuca varabiliriz. **Dynamic** tipin kullanıldığı hallerde **derleyicinin(Compiler)** derleme zamanında tip tahmini yapmasına gerek yoktur. Bu çözümleme işi çalışma zamanında yapılmaktadır.*

Tabi bu sonuçlara göre "her yerde **dynamic** tip kullanalım mı?" sorusu da gündeme gelir. Ancak "metodlara parametre aktarımlarında **dynamic** kullanımının kodun kırılmasına neden olması söz konusu olabilir mi?" sorusu daha da önemlidir. Şimdi bu durumu ele almaya çalışalım. İşte kodlarımız;

```
        dynamic R = "on iki";
        Calculate(R);

        #endregion

        #endregion

    }

    static double Calculate(double r)
    {
        return Math.PI * r * r;
    }
```

Bu kod parçasında kodu kırmaya yönelik olarak bir hamle yapıldığı düşünülebilir. **Calculate** metodu **double** tipinden bir değer beklemektedir. Biz ise **dynamic** olarak tanımladığımız **R** değişkenine **string** bir değer atayarak parametre gönderme işlemini gerçekleştirmekteyiz. **Dynamic** kullanımına göre derleme zamanında bir hata mesajı alınmaması normaldir. Benzer şekilde çalışma zamanında gelen **string** tipinin, **double** tipe otomatik olarak dönüştürüleceğini de düşünebiliriz. Eğer böyle olsaydı, kodun **dynamic** tipi yardımıyla kolayca kırılabileceği sonuçlarına varabilirdik. Şükür ki çalışma zamanında aşağıdaki hata mesajı ile cezalandırılırız.



Ancak,

dynamic R=12;

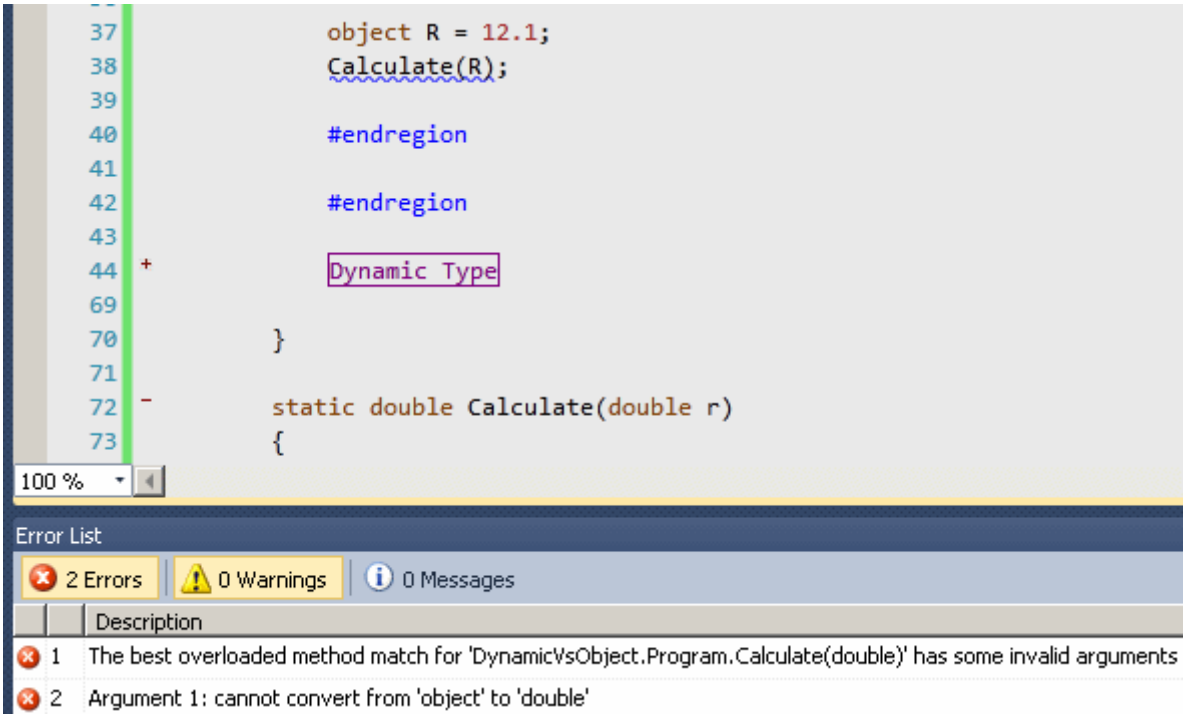
atamasını yaparsak herhangibir sorun ile karşılaşmayız. çünkü **Calculate** metodunun beklediği(veya taşıyabildiği) tipte bir değişkenin gönderilmesi sağlanmaktadır.

*Sonuç olarak bir metoda **dynamic** tipte veri atayabiliyor olsakta bu, metoda her tipten değeri aktarabileceğimiz anlamına gelmemektedir. Gerçekten metodun beklediği veya kabul edebileceği türden bir değişkenin atanması şarttır.*

Son senaryomuzu **object** tipi ile düşündüğümüzdeyse yine başta açıklanan 3 vakanın gerçekleşeceği görülecektir. Gelin bu durumları bir kere daha inceleyelim. Kodu ilk etapta aşağıdaki gibi düzenleyelim.

object R = 12.1;
Calculate(R);

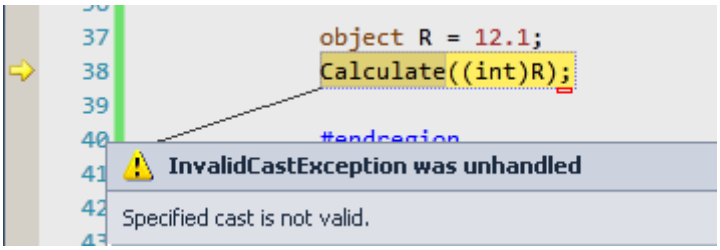
Bu durumda daha derleme zamanında hata mesajı alırız. Aşağıdaki şekilde olduğu gibi.



Dolayısıyla tip dönüşümü yapmamız şarttır. öyleyse yapalım.

```
object R = 12.1;
Calculate((int)R);
```

Derleme zamanında hata yok. Süper...Ama oda ne? çalışma zamanında yine hata aldık.



Tahmin edileceği üzere **object** tipi kullanıldığından çalışma zamanında tam olarak **double** tipinden bir değer taşınması beklenmektedir. **12.1 double** olarak ifade edilmesine rağmen **int**tipine yapılan dönüşüm geçersizdir*(**Dynamic** tipin kullanımının tam aksine)*. Dolayısıyla kodun aşağıdaki gibi düzenlenmesi gerekir.

```
object R = 12.1;
Calculate((double)R);
```

Bu durumda çalışma zamanında her hangibir hata mesajı alınmadan ilerlenebilecektir.

Görüldüğü üzere derinlerde, **dynamic** ve **object** kullanımları arasında belirgin farklılıklar bulunmaktadır. Bunların sebepleri aşikardır. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek, hepinize mutlu günler dilerim.

DynamicVsObject_RC.rar (20,73 kb) **[örnek Visual Studio 2010 Ultimate RC sürümü üzerinde geliştirilmiş ve test edilmiştir]**

NedirTv?com 4ncü Yıl Seminerleri (2010-03-31T17:00:00)

nedirtv,seminer,

2006 Mart ayında yayın hayatına başlayan nedirtv?com, 10 Nisan Cumartesi günü İstanbul' da düzenleyeceği etkinliklerle 4. yılını kutluyor. Konularında uzman nedirtv?com editörleri tarafından, beş farklı oturumda gerçekleştirilecek etkinliğimizde sizleri de aramızda görmeyi istiyoruz. Seminer programı ile ilgili detaylı bilgiler aşağıda yer almaktadır. Seminere katılmak için kayıt formunu doldurmanız gerekmektedir.

Not: Seminerlere katılım ücretsizdir. Kontenjan 100 kişi ile sınırlıdır.

09:45 Açılış
10:00 HTML 5 - Daron Yöndem
11:00 ASP.NET MVC 2 - Selçuk Yavuz

12:30 Ara

14:00 Microsoft Ajax Library(ASP.NET AJAX 4) - Uğur Umutluoğlu
15:00 WCF Eco System - Burak Selim Şenyurt
16:30 Windows Server AppFabric - Burak Selim Şenyurt
17:00 Hediye çekilişi ve Kapanış

Tarih: 10 Nisan 2010 Cumartesi
Yer: Microsoft İstanbul Ofisi Bellevue Residence Levent Mahallesi, Aydın Sokak. No:7 Levent İstanbul
Kroki için tıklayın

Kayıt formuna bu linkten ulaşabilirsiniz.

Screencast - Ajax Enabled WCF Serviceâ€™lerin Silverlight ile Kullanılması (2010-03-31T14:35:00)

screencast,ajax,ajax enabled wcf services,wcf,silverlight,

Merhaba Arkadaşlar,

Daha önceki görsel derslerimizde **WCF RIA Service** ve **Proxy** bazlı **WCF Service** örneklerinin, **Silverlight** tarafında nasıl kullanılabildiklerini yeri geldikçe irdelemeye çalışmıştık. **Silverlight** uygulamalarının **ASP.NET** gibi Web ortamlarında sunulduğu göz önüne alındığında, kullanabileceği servis tiplerinden birisi de **AJAX Enabled WCF Service**' leridir ki özellikle Web Form' lar üzerinde bu tip servisler sıklıkla kullanılmaktadır. Söz gelimi otomatik metin tamamlama işlemlerinde...*(Aslında **AJAX** tabanlı **WCF Service**' leri dışında **ASMX** tipindeki **AJAX Service**' leri de söz konusudur ama bu görsel dersimizde ele alınmamaktadır)*

İşte bu görsel dersimizde öncelikli olarak bir **AJAX Enabled WCF Service** geliştirip, bunu örnek bir **Silverlight** uygulamasında nasıl kullanabileceğimizi görmeye çalışıyor olacağız. Geliştireceğimiz örnekte servis ve **Silverlight** uygulamasının aynı domain içerisinde yer aldıklarını farz ediyor olacağız ki ilerleyen görsel derslerimizde ayrı domain' lerdeki servisleri nasıl değerlendirebileceğimizi de incelemeye çalışacağız. E haydi o zaman, ne duruyorsunuz? **NedirTv?** sponsorluğunda hazırladığımız görsel dersimizi izleyin.

***Not :** Dersin ilerleyen kısımlarında, arka planda bizim ufaklığın ve onu memnun etmeye çalışan ev ahalisinin derinden gelen seslerini duyabilirsiniz. Ama dikkati dağıtacak derecede olmayacaktır.*

Süre : 14:27

Dosya Boyutu : 24 Mb

Download Etmek veya İndirmek İçin

SilverlightApplication6.rar (100,34 kb) **[örnek Visual Studio 2010 RC Sürümü üzerinde geliştirimiş ve test edilmiştir]**

WCF WebHttp Services - Client Bazlı Cache (2010-03-30T16:30:00)

wcf,wcf 4.0,webhttp services,restful,non soap,wcf webhttp services,

Merhaba Arkadaşlar,

Bir önceki yazımızda(WCF WebHttp Services - Server Bazlı Cache) hatırlayacağınız üzere **WCF WebHttp Service'** lerinde **sunucu taraflı ön belleklemeyi(Server-Based Caching)**incelemeye çalışmış ve bu işin birde istemci taraflı olanından bahsetmiştik. Aslında sunucu ve istemci taraflı ön bellekleme işleyişleri birbirlerinden tamamen farklıdır. Sunucu taraflı ön bellekleme işleyişinde, tamponlanan veriyi üreten operasyonun duration süresi dolana kadar çalıştırılmaması söz konusudur. Yani istemciden gelen ilk talebin sonucunun ön belleğe alınmasını takiben gelen taleplerde, sunucu tarafındaki operasyon kodları icra edilmemektedir. Ne varki istemci taraflı ön belleklemenin işleyişine göre sunucu tarafındaki kodlar icra edilir ve ön bellekleme yapılacağı, **HTTP Cache-Control** bilgisinin istemciye gönderilen **cevabın(Response) Header** kısmına eklenmesi ile anlaşılır. Bir başka deyişle ilk talepten sonra gelecek taleplerde yine servis kodunun çalıştırılması gündemdedir. Dolayısıyla ispatı ve analizi pekte kolay olmayan bir konu ile karşı karşıyayız. Bu yüzden en azından nasıl hayata geçirilebileceğini görmeye çalışacağız. Elbette bir örnek geliştirerek. Gelin tembellik etmeyerek bir önceki uygulamamızdan devam etmek yerine yeni bir örnek üzerinden istemci taraflı ön belleklemenin nasıl yapılacağını araştıralım. öncelikle aşağıdaki servis kodlarına sahip olan bir **WCF REST Service Application** projemiz olduğunu düşünelim.

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;
using System.Web;

namespace Lesson7
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class FunnyService
   {
      List<Wallpaper> wallpapers = new List<Wallpaper>
         {
            new Wallpaper{ Id=1000, Name="Ahhh!",
Image=File.ReadAllBytes(HttpContext.Current.Server.MapPath("~/Images/Ahhh.jpg"))},
            new Wallpaper{ Id=1001, Name="Blue Light",
Image=File.ReadAllBytes(HttpContext.Current.Server.MapPath("~/Images/BlueLight.jpg"
))},
            new Wallpaper{ Id=1002, Name="My Dear Manager",
Image=File.ReadAllBytes(HttpContext.Current.Server.MapPath("~/Images/Manager.jpg"))
},
            new Wallpaper{ Id=1003, Name="No Sacrifice No Victory",
```

```
Image=File.ReadAllBytes(HttpContext.Current.Server.MapPath("~/Images/Sacrifice.jpg"))
}
        };

    [AspNetCacheProfile("WallpaperCache")] //web.config dosyasında
WallpaperCache ismi ile output cache konfigurasyonu yapılmaktadır.
    [WebGet(UriTemplate = "Wallpapers/{Name}")]
    public List<Wallpaper> GetWallpapers(string Name)
    {
        var result = (from w in wallpapers
                      where w.Name.ToLower().Contains(Name.ToLower())
                      select w).ToList();
        return result;
    }
  }

  public class Wallpaper
  {
    public int Id { get; set; }
    public string Name { get; set; }
    public byte[] Image{ get; set; }
  }
}
```

GetWallpapers isimli operasyonun çalıştırılması sonucunda istemci tarafına, **Images** klasörü altında yer alan **jpg** uzantılı resim dosyalarına ait **byte[]** içeriklerini taşıyan ve **Wallpaper** tipinden nesne örneklerinden oluşan generic List<T> koleksiyonu döndürülmektedir*(Elbette varsayılan mesaj formatına göre **XML** olarak)*. **AspNetCacheProfile** niteliği yardımıyla söz konusu operasyon için Caching işleminin icra edileceği bildirilir. Bu ön belleklemenin hangi kriterlere göre yapılacağına ait tanımlamalar ise **web.config** dosyası içerisinde aşağıdaki gibi belirlenir.

```
<?xml version="1.0"?>
<configuration>

  <system.web>
    <compilation debug="true" targetFramework="4.0" />
    <caching>
      <outputCache enableOutputCache="true"/>
      <outputCacheSettings>
        <outputCacheProfiles>
          <!-- İstemci taraflı ön bellekleme query string parametresi veya Header kısımlarını
dikkate almaz. Bu nedenle varyByParam niteliğine none değerini atanmış ve varyByHeader
niteliği kullanılmamıştır. Dikkat edileceği üzere ön belleklemenin istemci taraflı olacağı
location niteliği ile belirlenmiştir.-->
```

```
        <add name="WallpaperCache" duration="120" varyByParam="none"
location="Client"/>
      </outputCacheProfiles>
    </outputCacheSettings>
  </caching>
 </system.web>
```

Konfigurasyon dosyasının devamı...

OutputCacheProfiles içerisine eklenen **WallpaperCache** profiline göre **Cache** süresi **120** saniyedir. Diğer taraftan istemci taraflı ön bellekleme işlemlerinde **Querystring** kullanımı bir faktör olmadığından **varyByParam** niteliğine **none** değeri atanmalıdır. Bu profilde belkide en önemli nokta **location** niteliğinin değerinin **Client** olarak belirlenmesidir. Böylece ön bellekleme işleyişinin istemci taraflı yapılacağı tayin edilimiş olur.

Tabi bu durumu incelemek için istemci tarafının da geliştirilmesi gerekmektedir. Nitekim istemci tarafına gönderilen cevabın **Header** kısmında, **Caching** ile ilgili bilgilerin bulunması gerekmektedir. Bu amaçla çok basit olarak aşağıdaki kodlara sahip bir **Console** uygulaması yazdığımızı düşünebiliriz *(**HttpClient** ve diğer yardımcı tiplerin kullanımı söz konusu olduğundan **REST Starter Kit'** in parçası olan **Microsoft.Http** ve **Microsoft.Http.Extension assmebly'** larının referans edilmesi gerektiğini unutmayalım)*

```
using System;
using Microsoft.Http;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         // HttpClient nesnesi url adresine göre oluşturulur.
         HttpClient client = new HttpClient("http://localhost:1000/");

         Send(client);

         Console.WriteLine("Kapatmak için bir tuşa basınız");
         Console.ReadLine();

      }
      static void Send(HttpClient client)
      {
         Console.WriteLine("\nAradığınız kelimeyi girin\n");
```

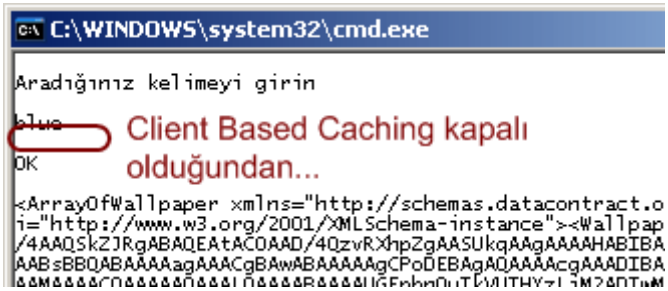
```
            string word = Console.ReadLine();
            // Wallpapers için gerekli request string oluşturulur. word bilgisi kullanıcı
tarafından girilmektedir.
            string requestString = String.Format("Wallpapers/{0}", word);

            // Get metodundan yararlanılarak sunucu tarafına ilgili talep gönderilir
            using(HttpResponseMessage responseMessage = client.Get(requestString))
              {
                // Sunucudan gelen cevap içerisinde yer alan Cache bilgisi ve ayrıca HTTP Statü
koduna bakılır
                Console.WriteLine("{0}\n{1}\n",responseMessage.Headers.CacheControl.Ma
xAge,responseMessage.StatusCode);

                // İçerik okunur
                Console.WriteLine(responseMessage.Content.ReadAsString());
              }
        }
    }
}
```

İstemci uygulamadan kullanıcının girdiği kelimeye göre **HTTP Get** formatında bir talebinin gönderilmesi sağlanmaktadır. Sunucu tarafından çağırılan operasyona ait ön bellekleme işleminin istemci bazlı olduğu, istemci tarafına gönderilen **cevap(Response)** içerisinden öğrenilebilir. Bu amaçla örnek olarak **HttpResponseMessage** nesnesinin **Header.CacheControl** özelliği üzerinden yakalanan **MaxAge** değerine bakılmıştır. Bu değer dikkat edileceği üzere sunucu üzerinde belirtilen **duration** süresinin **Timespan** karşılığıdır*(örneğimize göre 2 dakika)*.

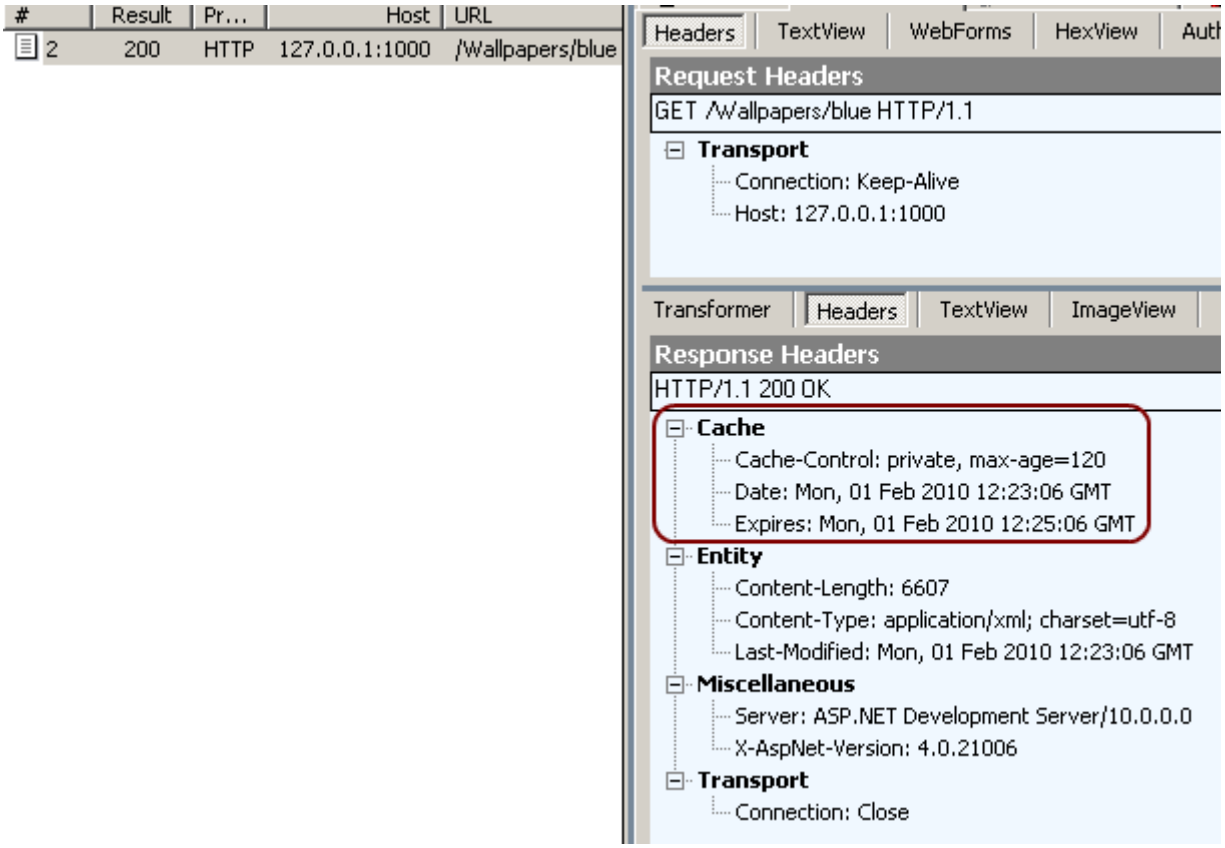
Tabiki Caching özelliğini sunucu tarafında kapatırsak ki bunun için **web.config** dosyasında ilgili **cache profile'** in **enabled** niteliğine **false** değeri atamamız yeterlidir,

<add name="WallpaperCache" duration="120" varyByParam="none" location="Client" **enabled="false"**/>

ve aynı örneği yeniden test edersek aşağıdaki sonuçlar ile karşılaşırız.



çok doğal olarak bir ön bellekleme süresi olmayacaktır. örneği **Fiddler** gibi bir araç yardımıyla incelediğimizde ise istemci tarafına dönen cevaba ait **Cache** bilgisini daha detaylı bir şekilde görebiliriz.



WCF WebHttp Service' ler ile ilişkili serimizde yer alan bir yazımızın daha sonuna geldik. Bu yazımızda istemci bazlı ön bellekleme işlemleri için ne gibi hazırlıklar yapılması gerektiğini gördük. En önemli noktanın **web.config** dosyası içerisinde belirtilen **location** niteliği olduğunu özetleyebiliriz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Lesson7_RC.rar (192,46 kb) **[örnek Visual Studio 2010 Ultimate Beta 2 Sürümünde geliştirilmiş ancak RC sürümü üzerinde de test edilmiştir]**

WCF WebHttp Services - Server Bazlı Cache (2010-03-30T00:05:00)

wcf,wcf 4.0,webhttp services,restful,non soap,wcf webhttp services,

 Merhaba Arkadaşlar,

Şirketimi çok seviyorum. Nazar değmesin ama araştırma yapmam, yeni bir şeyler öğrenmem ve bunları ekip arkadaşlarımla paylaşmam için beni özellikle teşvik eden bir şirkette bulunmaktayım. çalıştığım şirketin en güzel özelliklerinden biriside, **Cuma** günleri yapılan minik ikramları. Her cuma değişik bir yiyecek ile karşılaşıyoruz. Geçtiğimiz Cuma' lardan birisinde ise yanda çektiğim resimde görülen gülen kurabiyelerimiz vardı. E böylesine güler yüzlü kurabiyeler ile gerekli glikozu aldıktan sonra içimden hemen eve gitmek gelmedi. Bunun yerine mesai sonrasında çalışma masamda oturup, etrafın sakinleşmesi ve sessizliğin artması ile birlikte bloğuma bir şeyler yazmaya karar verdim. Bir süredir **WCF Eco System**' in parçaları üzerinde yazmakta olduğum bir seri bulunmaktaydı. Bunu devam ettirmek ile **Cuma** gecesini güzelce tamamlayabileceğimi düşündüm. İşte bu günkü konumuz...**WCF WebHttp Service**' lerinde **ön bellekleme(Output Caching)**.

WCF WebHttp Service' leri bildiğiniz üzere **Web** ortamı üzerinden sunulan hizmetlerdir. Bu sebepten **Web** tarafının sunucu ve istemci bazlı bazı yeteneklerini kullanabilirler. örneğin **Asp.Net Compatibility Mode** ile çalıştırıldıklarında **Asp.Net** dünyasının **Output Cache** yeteneğine sahip olurlar. Bildiğiniz üzere **Output Cache** mekanizması sayesinde **Web** içeriklerine gelen taleplerin ön bellekten karşılanması ve bu sayede arka planda ilgili **HTML** çıktılarının üretilmesi için gerekli işlemlerin otomatikman atlanılması sağlanabilmektedir. Bu, özellikle üretim maliyeti yüksek olan ama belirli bir süre zarfı içerisinde değişmeyen sayfa içeriklerinin üretiminde oldukça performans arttırıcı bir tekniktir. Madem **Web** tarafında böyle bir yeteneğimiz bulunmaktadır, o halde neden bu kabiliyeti **WCF** Servislerinde de kullanamayalım? İşte bu yazımızda **Asp.Net** tarafında hazır olan bu alt yapının **WebHttp Service'** lerinde nasıl kullanıldığını incelemeye çalışıyor olacağız. İlk olarak konuyu **sunucu bazlı ön bellekleme(Server Side Caching)** olarak değerlendireceğiz. Serinin sonraki bölümünde ise istemci taraflı ön belleklemeyi ele alacağız. Dilerseniz vakit kaybetmeden kodlamaya başlayalım. öncelikli

olarak aşağıdaki **WebHttp Service** içeriğine sahip bir **WCF REST Service Application** geliştirdiğimizi düşünelim.

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;
using System.Web;

namespace Lesson6
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class EinsteinService
   {
      // GetCategories isimli metodun çıktısının ön bellekleneceği AspNetCacheProfile
niteliği ile belirtilir. Nitelikte parametre olarak kullanılan string bilgi ile web.config
dosyasında bu metod için nasıl bir ön bellekleme yapılacağı set edilir. Ne kadar süre ile
tutulacağı, parametre bazlı olup olmayacağı gibi...
      [AspNetCacheProfile("CategoriesCache")]
      [WebGet(UriTemplate = "Categories")]
      public List<string> GetCategories()
      {
         string[]
categories=File.ReadAllLines(HttpContext.Current.Server.MapPath("~/Kategoriler.txt"));
         return new List<string>(categories);
      }

      [AspNetCacheProfile("CategoriesByFirstLetterCache")]
      [WebGet(UriTemplate = "Categories/{firstLetter}")]
      public List<string> GetCategoriesByFirstLetter(string firstLetter)
      {
         string[] categories =
File.ReadAllLines(HttpContext.Current.Server.MapPath("~/Kategoriler.txt"));
         return (from category in categories
                 where category.StartsWith(firstLetter,true,null)
                 select category).ToList();
      }
   }
}
```

örneğimizde yer alan servisimizde iki operasyon yer almaktadır. Her iki operasyonda aşağıda örnek çıktısı olan **Kategoriler.txt** isimli text tabanlı dosyayı kullanmaktadır.

GetCategories operasyonu **text** dosyası içerisindeki tüm satırları **string** tipinden **generic List** koleksiyonu olarak geriye döndürmektedir. **GetCategoriesByFirstLetter** operasyonu ise aynı çıktıyı baş harflere göre üretmektedir. Bizim odaklanmamız gereken nokta ise her iki operasyon başında uygulanan **AspNetCacheProfile niteliğidir(Attribute)**. Bu niteliklerin uygulanması ile söz konusu operasyonların çıktılarının ön bellekleneceği, çalışma zamanı ortamına bildirilmektedir. Her iki nitelikte birbirlerinden benzersiz olan takma adlar(Alias) ile işaret edilmektedir. Peki bu isimleri nerede değerlendireceğiz? Bu sorunun cevabı **Web.config** dosyasında yer alan **Asp.Net Output Cache** ayarlarında gizlidir... Buradaki ayarlar ile hangi operasyon için nasıl bir ön bellekleme işleminin uygulanacağını belirtebiliriz. örneğin operasyonların sonuçlarının ne kadar süreyle ön bellekte tutulacaklarını farklılaştırabiliriz. Yada parametre bazlı olanları...örneğin **GetCategoriesByFirstLetter** operasyonunun çalışma zamanı **HTML** çıktılarının **firstLetter** bilgisine göre ön belleklenebileceğini belirtebiliriz. Tüm bu ayarlamalar için**web.config** dosyasına aşağıdaki eklemeleri yapmamız yeterlidir.

```
<?xml version="1.0"?>
<configuration>

  <system.web>
    <compilation debug="true" targetFramework="4.0" />

    <caching>
      <outputCache enableOutputCache="true"/>
      <outputCacheSettings>
```

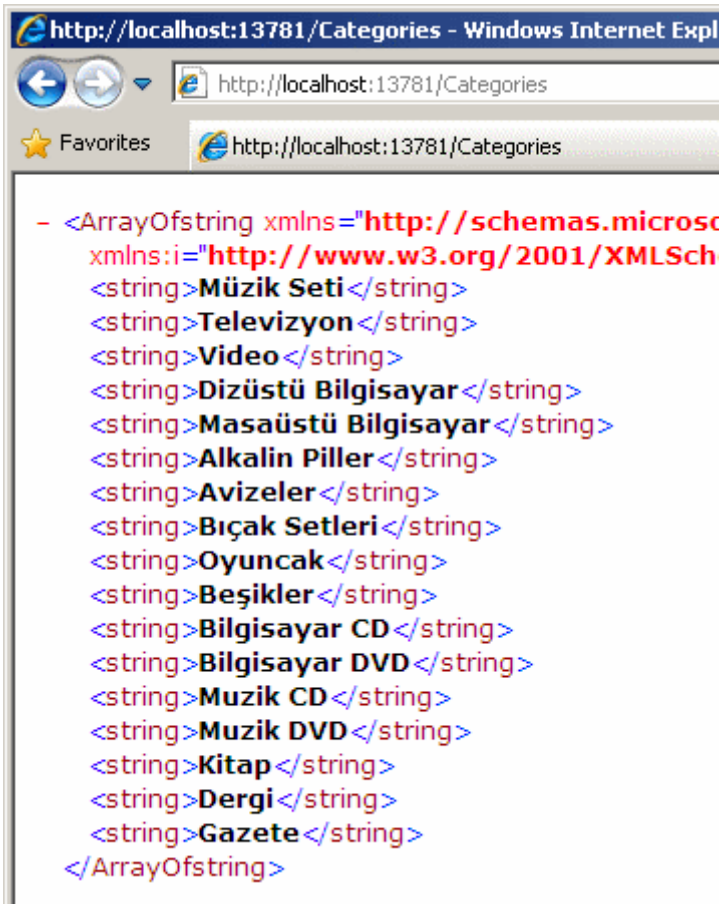
```
      <outputCacheProfiles>
        <add name="CategoriesCache" duration="120" location="Server"
varyByParam="none" varyByHeader="Accept"/>
        <add name="CategoriesByFirstLetterCache" duration="300"
location="Server" varyByParam="firstLetter" varyByHeader="Accept"/>
      </outputCacheProfiles>
    </outputCacheSettings>
  </caching>
 </system.web>

 <system.webServer>
  <modules runAllManagedModulesForAllRequests="true">
   <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule,
System.Web, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" />
  </modules>
 </system.webServer>

 <system.serviceModel>
  <!--Asp.Net Output Caching alt yapısını ve yeteneklerini kullanmak istediğimiz için,
AspNet Compatibility modun açık olması önemlidir.-->
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>
  <standardEndpoints>
   <webHttpEndpoint>
    <!--
       Configure the WCF REST service base address via the global.asax.cs file and the
default endpoint
       via the attributes on the <standardEndpoint> element below
    -->
    <standardEndpoint name="" helpEnabled="true"
automaticFormatSelectionEnabled="true"/>
   </webHttpEndpoint>
  </standardEndpoints>
 </system.serviceModel>

</configuration>
```

Buradaki ayarlamalara göre **CategoriesCache** adı ile belirlenen operasyonlar **120** saniye süreyle ön bellekten getirilecektir. ön bellekleme ortamının sunucu taraflı olduğu **location** niteliği ile belirtilmektedir. Bu ön bellek ayarında parametre kullanılmadığı*(varyByParam="none")* ve **talebin(request) Header** kısmına göre **Accept** tipinde olanların göz önüne alındığı ifade edilmektedir. Diğer yandan **CategoriesByFirstLetterCache** bildirimine bakıldığında sürenin **300** saniye olduğu ama bir öncekinden farklı olarak **firstLetter** parametresine göre ön bellekte ayrı ayrı görüntüler tutulacağı belirtilmektedir. Buna göre **Categories/A** gibi bir talep ile **Categories/s** gibi bir taleb için üretilen **HTML** çıktıları ön bellekte ayrı ayrı

tutulacaktır. Tabi birden fazla parametrenin değerlendirilmesi gerektiği durumlarda **;** işareti kullanılarak bildirilmeleri gerekmektedir. örneğin **varyByParam="firstLetter;productCount"** vb.

Tabi yapımış olduğumuz bu anlatımın sonuçlarını test ederek görmemiz gerekiyor. Burada **debug** modu kullanarak talepler sırasında operasyon metodlarının içerisine ne zaman girilip ne zaman girilmediğini tespit ederek gerekli kontrolleri yapabiliriz. İşte size örnek test senaryoları;

1 - önce **URL** üzerinden **Categories** talebini gönderin. Bu durumda tüm kategorilerin aşağıdaki şekilde olduğu gibi geldiğini göreceksiniz.



Şimdi **120** saniyelik süre dolmadan **Kategoriler.txt** üzerinde bir değişiklik yapın. örneğin **Mücehver** isimli yeni bir kategori ekleyin veya bir kaç kategorinin ismini değiştirin yada silin. **120**saniyelik zaman dilimi içerisinde yeniden **Categories** talebinden bulunursanız yapmış olduğunuz değişiklilerin tarayıcıya getirilmediğini görebilirsiniz. Ancak **120** saniyelik ön bellek süresi dolduktan sonra değişiklikleri görebileceksiniz ve hatta **Debug** moddaysanız ilgili operasyon kodu içerisine tekrardan girildiğini fark edeceksiniz. Bu zaten ön belleklemenin çalıştığının ispatıdır.

2 - İkinci olarak parametre bazlı ön bellekleme yapıldığını test edebilirsiniz. Bu amaçla tarayıcı üzerinden örneğin **Categories/A** ve **Categories/M** taleplerinde

bulunun. **Debug** modda ilerlerseniz testinizi daha başarılı bir şekilde yapabilirsiniz. özellikle **300** saniyelik ön bellekleme süreleri dolmadan farklı harfler ile denemeler yaptığınızda, ön belleklenenler için operasyon koduna girilmediğini ama daha önceden talep edilmeyenler veya **300** saniyelik ön bellekte kalma süresini dolduranlar için kodun tekrar çalıştırıldığını gözlemleyebilirsiniz.

Bu sayede geliştireceğimiz **WebHttp Service'** lerin operasyonlarının hızlı sonuçlar üreterek daha performanslı ve verimli olmasını sağlayabiliriz. Bu yazımızdaki örneğimizde sunucu tarafında ön bellekleme işlemlerini gerçekleştirdik. Ancak birde **istemci taraflı ön bellekleme(Client Based Caching)** işlemlerinin söz konusu olduğunu belirtelim. Bunu serinin sonraki yazısında incelemeye çalışacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Lesson6_RC.rar (20,41 kb) **[örnek Visual Studio 2010 Ultimate Beta 2 Sürümünde geliştirilmiş ancak RC sürümü üzerinde de test edilmiştir]**

Screencast - Silverlight Enabled WCF Services (2010-03-24T11:10:00)

screencast,wcf services,silverlight enabled wcf services,silverlight,

Merhaba Arkadaşlar,

Silverlight tarafında servis denildiğinde akla ilk gelen model **WCF RIA Services**' leridir. Esas itibariyle **WCF Eco System**' in bir parçası olan **WCF RIA Service**' ler**Silverlight** tarafında ele alınabilecek tek servis modeli değildir. Söz gelimi, aynı domain içerisinde yer alan bir **WCF Service**' i de, **Silverlight** istemcileri tarafından**Proxy** kullanımı ile **tüketilebilir(Consume)**. İşte NedirTv? sponsorluğunda hazırladığımız bu görsel dersimizde serileştirilebilir bir tipi geriye döndüren bir operasyonu içeren **Silverlight destekli bir WCF Service'** inin, **Proxy** tabanlı olarak nasıl kullanılabileceğini incelemeye çalışıyoruz. üstelik **Proxy** üretiminin bir faydası olarak ilgili servis operasyonu çağrısının asenkron olarak nasıl yapılabileceğini de göreceğiz.

Süre : 14:55

Boyut : 21.7 Mb

Download etmek veya izlemek için

SilverlightApplication4.rar (104,07 kb) **[örnek Visual Studio 2010 RC sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Entity Framework - Many To Many Relations - Link Tablosunda Israrcı Olmak (2010-03-24T09:35:00)

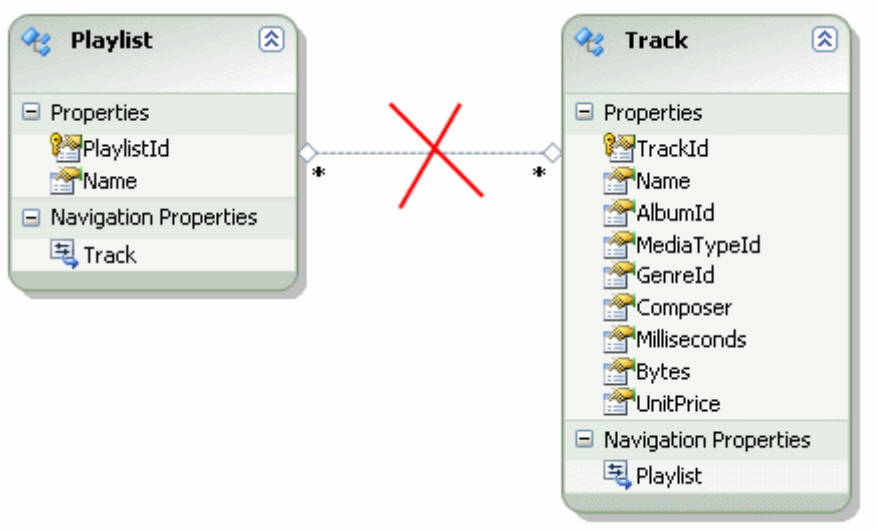
ado.net entity framework,

Merhaba Arkadaşlar,

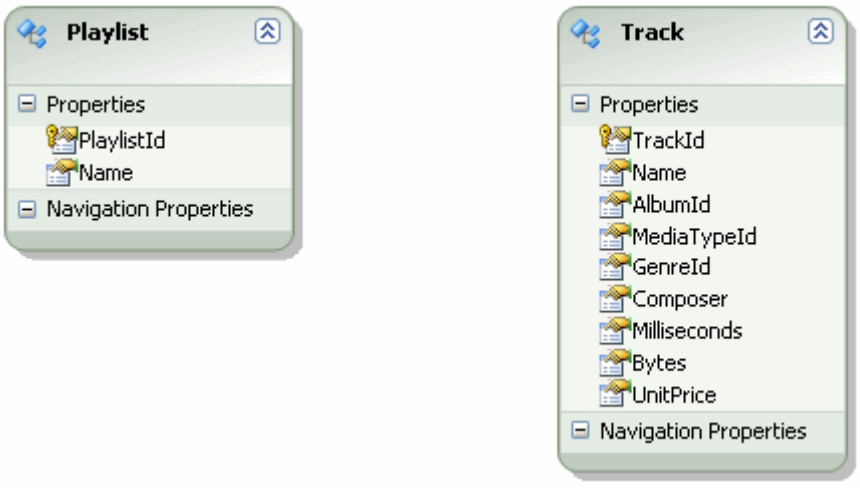
Bazen bebek adımları ile ilerlememiz gerekir. özellikle yazılım alanında bazı konuları öğrenirken işin teorisinden önce pratik bir örneği adım adım geliştirmek son derece faydalıdır. **Ado.Net Entity Framework** ile ilişkili inclemelerimize devam edeceğimiz bu yazımızda bebek adımları ile ilerleyeceğiz.

Hatırlayacağınız üzere son iki yazımızda **Many-To-Many** ilişkileri nasıl ele alabileceğimizi incelemiştik. **Many-To-Many** ilişkilerin **Entity Model'e** olan yansımasında belkide en önemli nokta, ara bağlantı tablosunun taşınmıyor olmasıydı. Bu genellikle ara bağlantı tablosu üzerinde diğer tablolara ait **Primary Key** alanlarının bulunduğu durumlar düşünülerek meydana gelen bir sonuçtur. Nitekim ara tablonun **Entity**tarafına taşınmayışının herhangibir olumsuz maliyeti bulunmamaktadır. Ancak bazı durumlarda söz konusu ara bağlantı tablosunun ilerleyen zamanlarda ek alanlar ile genişlemesi söz konusu olabilir. Bu durumda ara bağlantı tablosunun **Entity** olaraktan **Model Diagram** içerisinde yer almasının avantajı olacaktır. Nitekim ileride eklenecen kolonlar için **Entity Model** tarafında sadece **Conceptual Schema'** yı güncellemek yeterlidir.

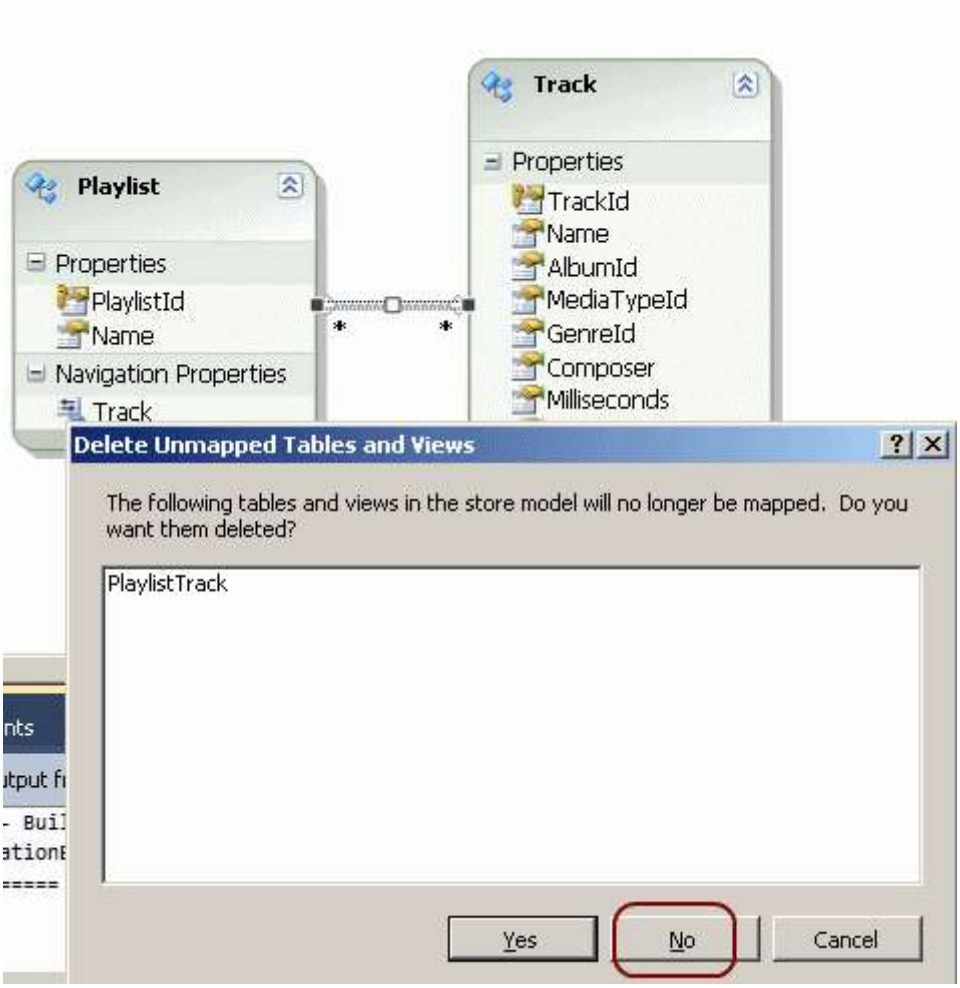
Dolayısıyla bu yazımızdaki amacımız, varsayılan olarak **Entity Model Diagram** üretilirken ortadan kaldırılan ara bağlantı tablosunun manuel olarak oluşturulmasını sağlama olacaktır. Başlangıçta **Visual Studio 2010 Ultimate RC** sürümü üzerinde oluşturduğumuz **Console** uygulaması içerisinde yer alan **Entity Model** diagramının aşağıdaki gibi olduğunu varsayıyoruz.



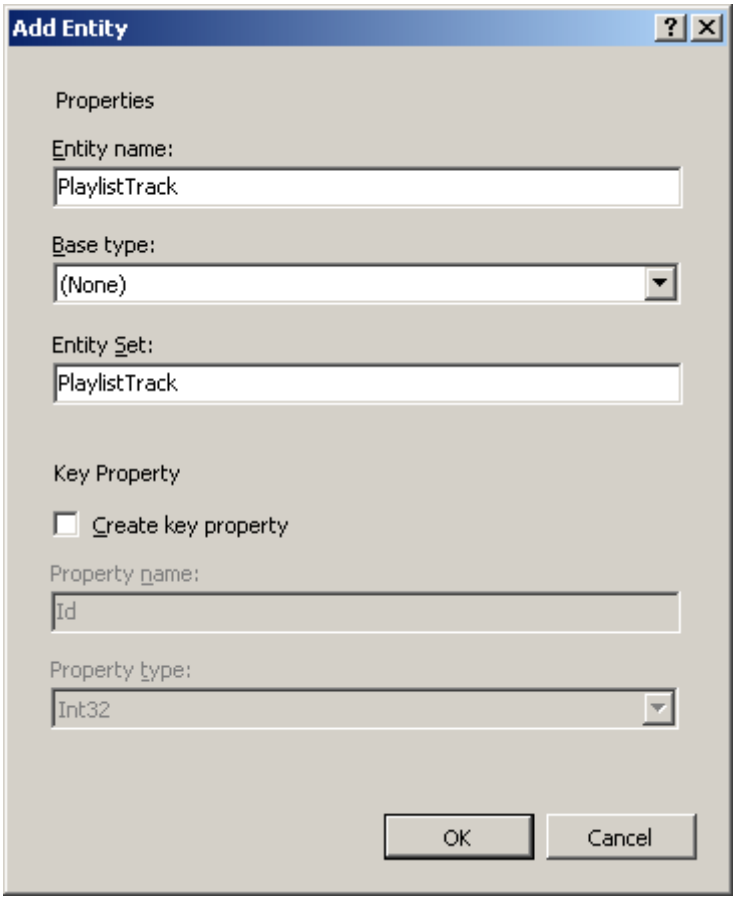
Tahmin edeceğiniz üzere **Chinook** veritabanı içerisinde yer alan **Playlist-Track** ilişkilerini değerlendirmekteyiz. Şekil üzerinde yer alan çarpı işareti mutlaka dikkatinizi çekmiştir. Aslında bu yapacağımız ilk hamle olacak. Aradaki **Association** nesnesini silmek. Bu durumda çok doğal olarak **Playlist** ve **Track** tablolarında yer alan ve birbirlerine olan geçişleri sağlayan **Navigation Property'** lerin de ortadan kaldırıldığını göreceğiz.



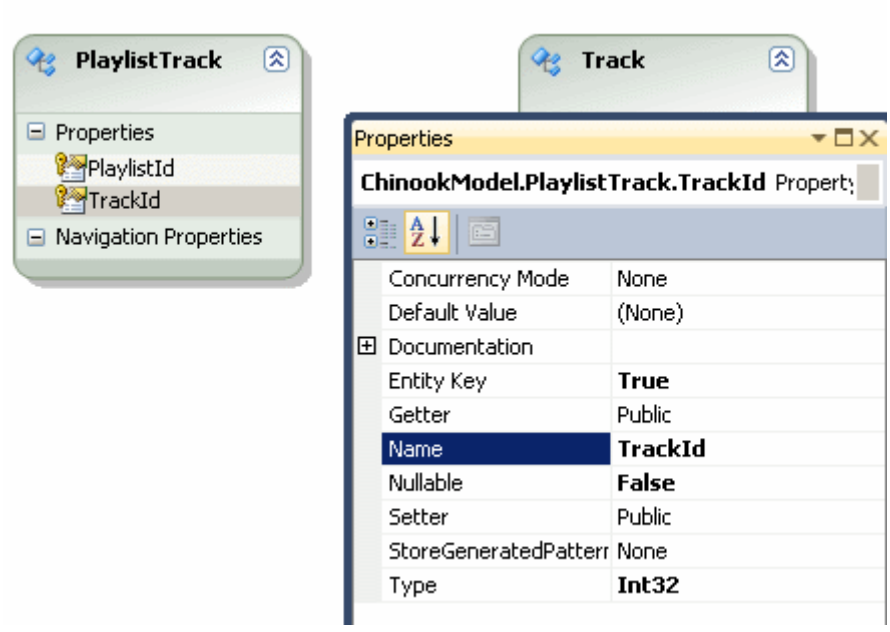
Ancak burada dikkat etmemiz gereken bir nokta bulunmaktadır. Silme işlemi sırasında Designer' ın aşağıdaki sorusu ile karşılaşacağız. **PlaylistTrack** tablosu her ne kadar **Entity** olarak ifade edilmese de, oluşturacağımız **Entity'** nin ileride veritabanında karşılık geldiği tablo ile ilişkilendirilmesi sırasında ele alınacaktır. Bu nedenle **No** seçimi yaparaktan silme işlemini icra etmemiz daha doğrudur.



Sıradaki adımımız ise veritabanında yer alan ve **Playlist** ile **Track** tabloları arasındaki **Many-To-Many** ilişkiyi gerçekleştiren tabloya karşılık gelen bir **Entity** tipinin oluşturulmasıdır. Yani veritabanında yer alan **PlaylistTrack** isimli tablonun karşılığı olan **Entity** tipinin üretilmesi. Bunun için **Model** diagramı üzerinden **Add->Entity** seçimini yapmamız yeterli olacaktır. Sonrasında ise **Entity** özelliklerini aşağıdaki gibi ayarlamamız gerekecektir.



Burada dikkat edilmesi gereken noktalardan birisi, **Key Property** özelliğinin kullanılmayışıdır. Nitekim bu alana senaryomuza göre gerek yoktur. Ara tablonun karşılığı olan **PlaylistTrackEntity** tipinin görevi, **Playlist** ve **Track** **Entity** tipleri arasındaki **Many-To-Many** ilişkiyi sağlamak olduğundan, bu **Entity'** lere ait **Key** özelliklerini barındırması yeterlidir. Buna göre yeni oluşturulan **Entity** içerisinde aşağıdaki şekilde görülen **Scalar Property'** lerin eklenmesi gerekir.

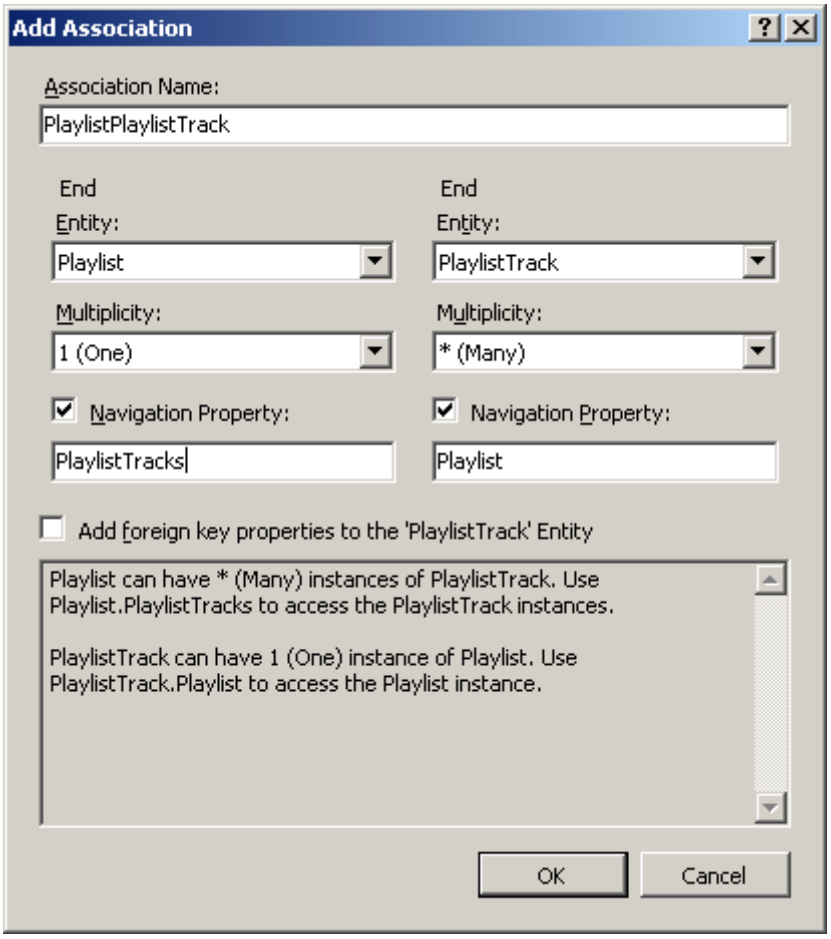


Yine dikkat edilmesi gereken önemli bir nokta vardır. **PlaylistTrack Entity** tipi içerisinde yer alan **PlaylistId** ve **TrackId** isimli alanlarının **Entity Key** özelliklerine **true** değer atanmıştır. Buna göre bu alanların **Playlist** ve **Track** tablosundaki karşılıkları ile ilişki kurabilmesi mümkün olacaktır. Dolayısıyla sıradaki adımımız tablolar arasındaki ilişkileri kurmaktır.

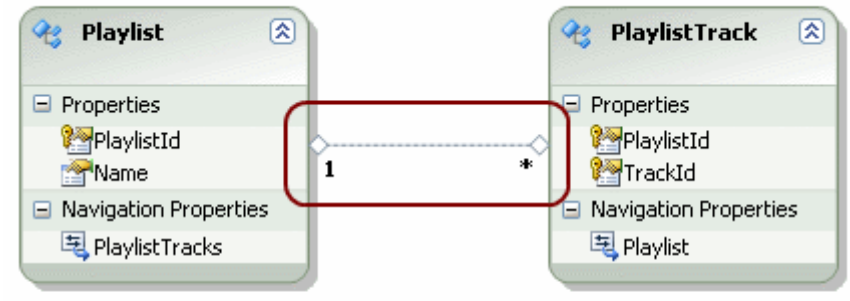
*Not : Aslında **PlaylistId** ve **TrackId** alanlarının oluşturulması şart değildir. **PlaylistTrack** tablosuna ait bilgileri silmediğimizden **Association'** ları oluşturduğumuz sırada söz konusu alanların otomatik olarak üretileceğini görebiliriz. Bu tekniğide tercih edebilirsiniz. Şu andaki ilerleyişimize göre **Association'** ların oluşturulması sırasında **Add foreign key properties to the...** seçeneğini kaldırmamız gerekmektedir. Ancak kolonların oluşturulmasını otomatikleştirmek istiyorsak, bu seçeneği işaretli bırakmamız gerekmektedir. Böyle yaptığımız takdirde kolon adlarının pekte istediğimiz gibi olmayacağını, yeniden isimlendirmemiz gerektiğinide belirtmek isterim. Tercih size kalmış.*

Şimdi bir düşünelim. Bir **Playlist'e** bağlı birden fazla **Track** olabilir. Benzer şekilde bir **Track** birden fazla **Playlist** içerisinde yer alabilir. Buna göre **Playlist** ve **Track** varlıkları arasında **Many-To-Many** ilişki söz konusudur. Bunu zaten biliyoruz Ancak bunun **Entity Model** diagram tarafında, şu andaki yapıya göre ifadesi nasıl olacaktır? Cevap; **Playlist Entity'** sinden **PlaylistTrack Entity'** sine ve benzer şekilde **Track Entity'** sinden yine **PlaylistTrack Entity'** sine doğru **One-To-Many** ilişki sağlayarak. Bunun için diagrama bir **Association** nesnesi eklenmesi yeterlidir**(Add->Association).**

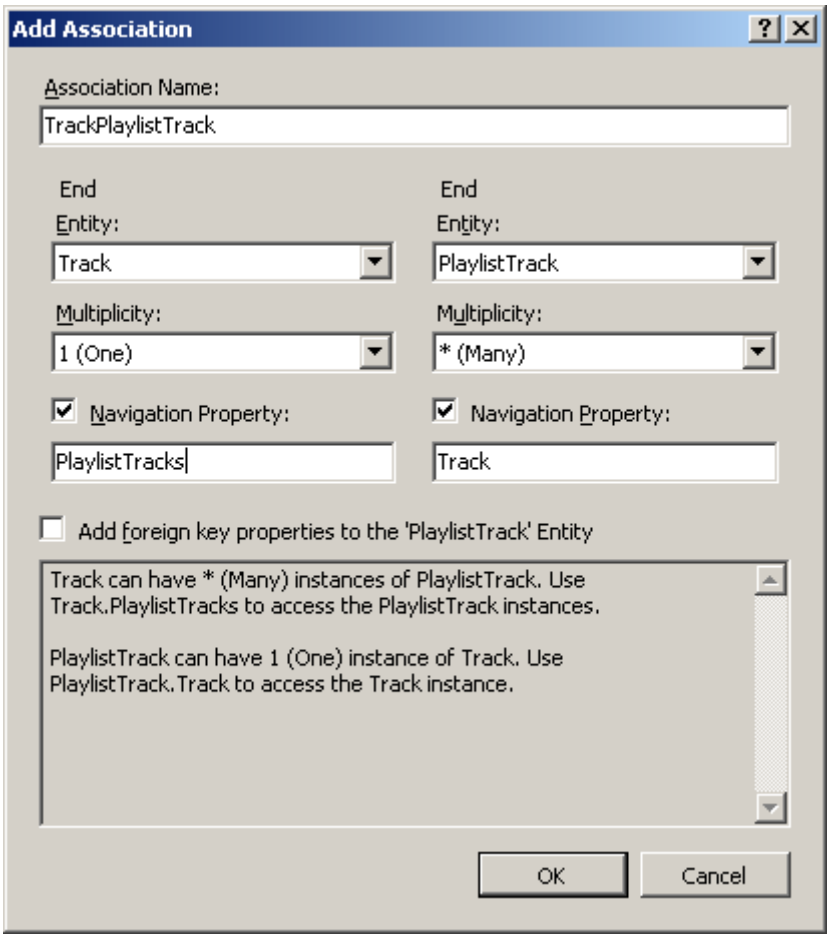
Playlist -> PlaylistTrack



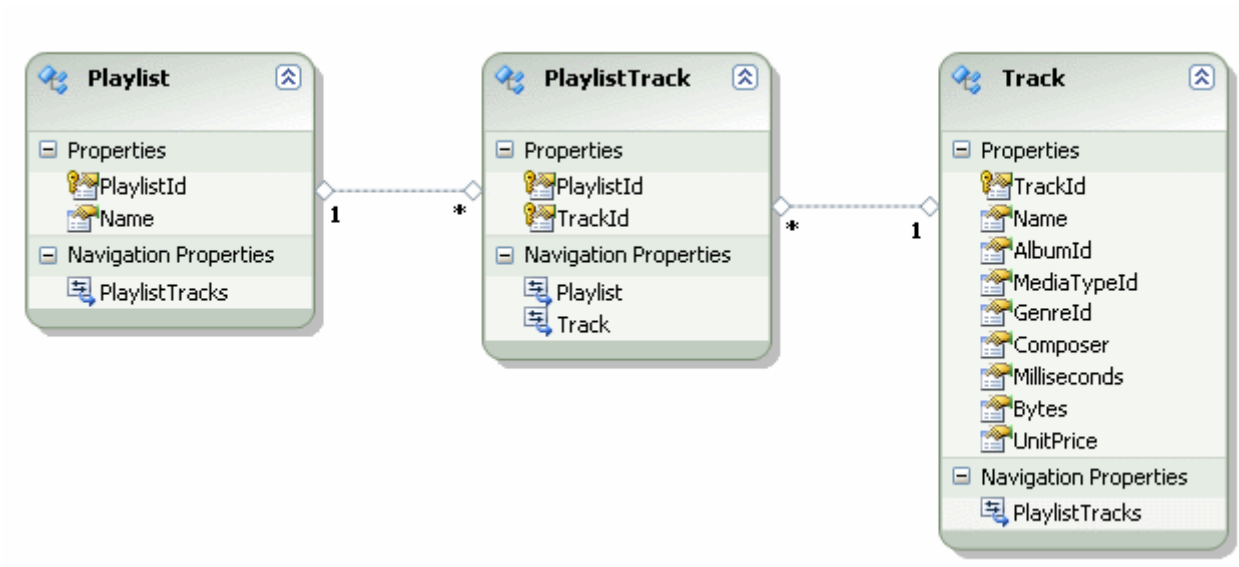
Dikkat edileceği üzere **End** özelliklerinden sol tarafta yer alanda **Playlist** bulunmaktadır. **Playlist Entity** tipi için **Multiplicity** özelliği **1(One)** iken, sağ tarafta yer alan **PlaylistTrack** için ***(Many)** dir. Buna göre **Playlist Entity** tipinden **PlaylistTrack Entity** tipine doğru **One-To-Many** ilişki kurulmuştur. üretilen **Navigation Property**' lerden **PlaylistTrack** olarak adlandırılanlar, çoğulluk nedeni ile PlaylistTrack**s** olarak yeniden isimlendirilmiştir. Bu işlemin sonrasında diagramın yeni hali aşağıdaki gibi olacaktır.



Track -> PlaylistTrack

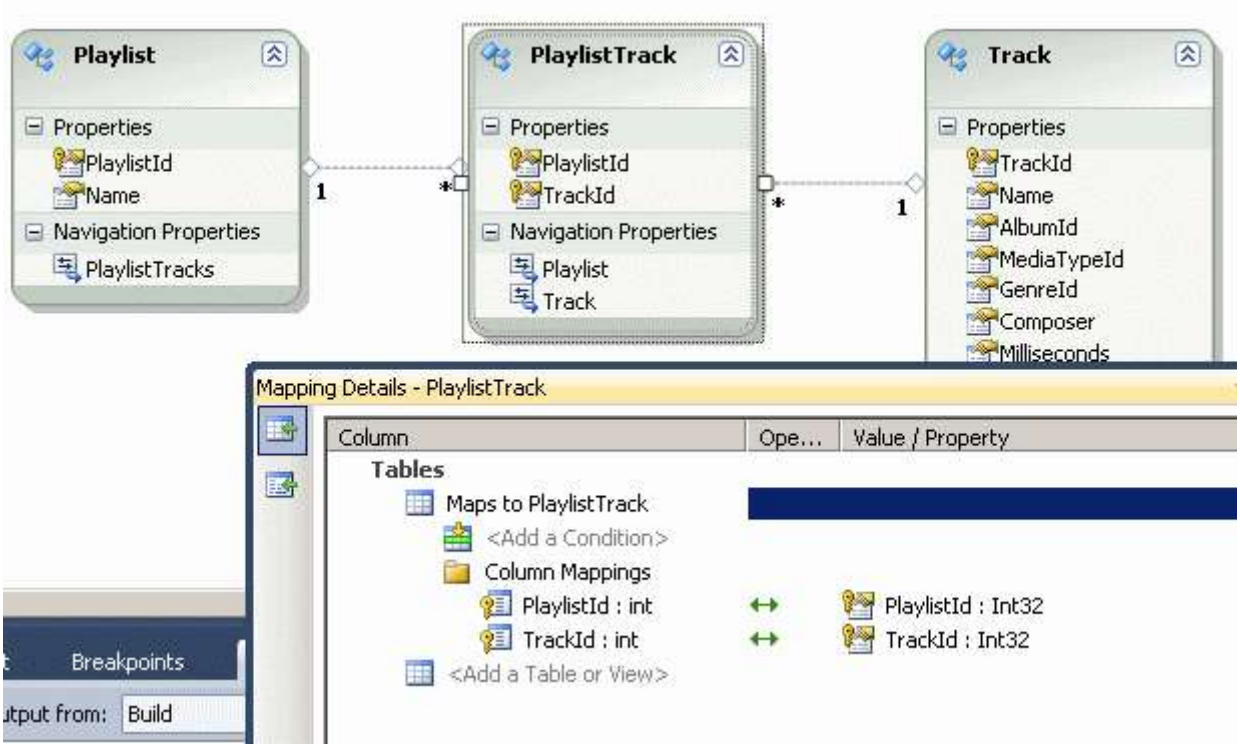


Bu kez **Track Entity** tipi üzerinden az önceki gibi **PlaylistTrack Entity** tipine doğru **One-To-Many** ilişki kurulmuştur. Sonuç olarak diagramın yeni hali aşağıdaki gibi olacaktır.

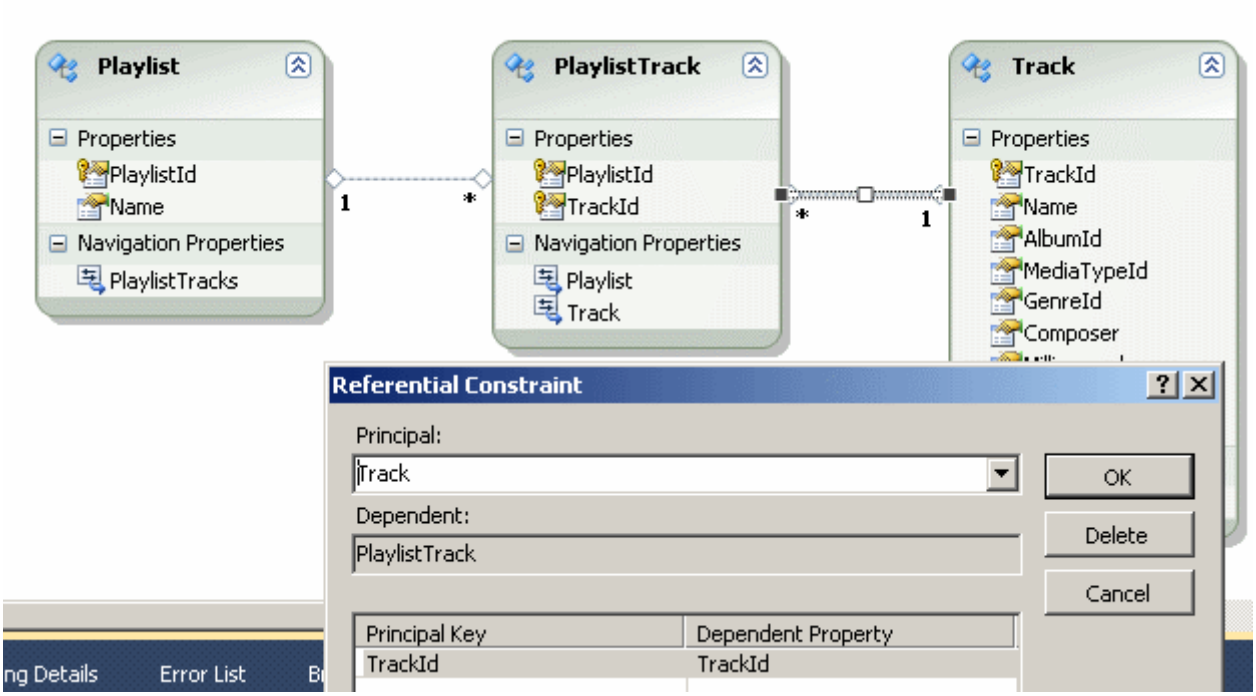


Sıradaki adımda oluşturulan yeni **PlaylistTrack Entitiy** tipinin, veritabanında yer alan **Tablo** karşılığı ile eşleştirilmesi gerekir. Bu

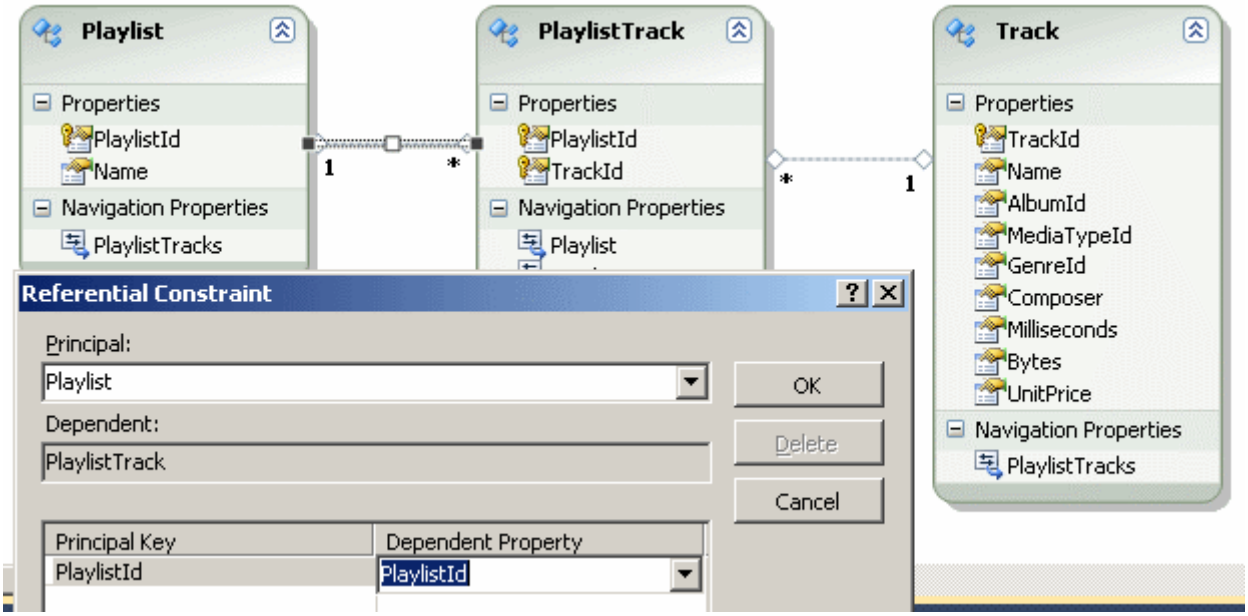
sebepten **PlaylistTrack Entity** tipinin **Mapping Details**özelliklerinden aşağıdaki gibi gerekli eşleştirmenin yapılması şarttır. Hatırlayacağınız üzere ilk adımda **Association** nesnesini silerken tablonun kaldırılmamasını belirtmiştik. Bu sayede **Table** eşleştirmesini kolayca gerçekleştirdik.



Ancak işlemlerimiz bir türlü bitmek bilmemektedir. Sabredin, çok az kaldı. Takip eden adımda **Referentail Constraint**' lerin tanımlanması gerekmektedir. Normal şartlarda eğer Association' lar oluşturulurken **Add foreign key properties to the...** işaretli olsaydı bunlarda ilgili alanlar ile birlikte otomatik oluşturulacaklardı. Bu nedenle izleyen şekillerde görüldüğü üzere ilgili eklemelerin yapılması gerekmektedir.



ve



Buraya kadar yaptıklarımızı özetleyecek olursak aşağıdaki adımları gerçekleştirdiğimizi ifade edebiliriz.

1- **Entity'** ler arasındaki **Many-To-Many** tipinden **Relation** silinir.

2- Veritabanında yer alan ama **Entity Model** tarafına aktarılmayan ara tablonun karşılığı olan tipin üretilmesi sağlanır.

3- Eklenen **Entity** tablosuna **Entity Key** özellikleri **true** olan ve diğer **Entity'** ler üzerinde karşılık düşen alanlar ile aynı tipten olan*(örneğimizde hepsi Int32)* **Scalar Property**' ler ilave edilir.

4- **Entity** tipleri arasındaki **One-To-Many ilişkiler(Association)** tesis edilir.

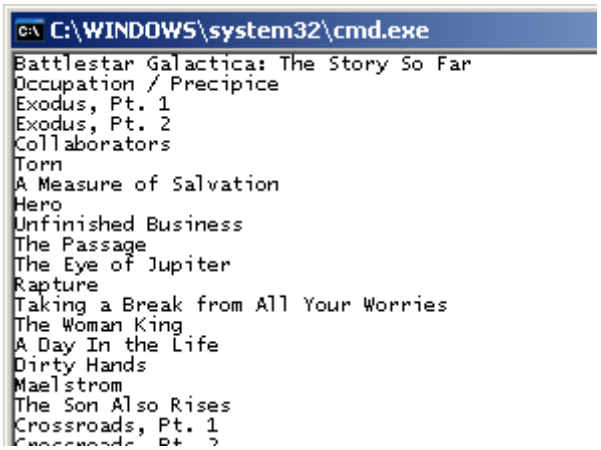
5- Eklenen **Entity** nesnesi, veritabanındaki tablo ile ilişkilendirilir***(Mapping)***.

6- **Association'** lar ile ilişkili olaraktan gerekli **Referencial Constraints'** ler tanınmlanır.

Artık konuyu basit bir örnek sonlandırabiliriz. Bu amaçla aşağıdaki program kodunu yazdığımızı düşünelim.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RelationEntityCreation
{
   class Program
   {
      static void Main(string[] args)
      {
         using (ChinookEntities entities = new ChinookEntities())
         {
            var result = (from p in entities.Playlists.Include("PlaylistTracks.Track")
                          where p.PlaylistId == 3
                          select p).First();


            foreach (var r in result.PlaylistTracks)
            {
               Console.WriteLine(r.Track.Name);
            }
         }
      }
   }
}
```

örnek kod parçasında **PlaylistId** değeri **3** olan satıra bağlı olan **Track'** lerin çekilmesi sağlanmaktadır. **Include** metodu içerisinde belirtilen tanımlama önemlidir.
Burada **PlaylistTracks**üzerinden **Track Entity'** sine ulaşılmaktadır. Bu sayede Playlist' e bağlı Track' lerinde yüklenmesi sağlanmış olur. Uygulama kodunun çalışmasının sonucu aşağıdaki gibidir.



Tabi bu kodun çalışması sonrasında SQL tarafında oldukça yoğun bir sorgunun üretilmesi söz konusudur.

```
SELECT
[Project1].[PlaylistId] AS [PlaylistId],
[Project1].[Name] AS [Name],
[Project1].[C1] AS [C1],
[Project1].[PlaylistId1] AS [PlaylistId1],
[Project1].[TrackId] AS [TrackId],
[Project1].[TrackId1] AS [TrackId1],
[Project1].[Name1] AS [Name1],
[Project1].[AlbumId] AS [AlbumId],
[Project1].[MediaTypeId] AS [MediaTypeId],
[Project1].[GenreId] AS [GenreId],
[Project1].[Composer] AS [Composer],
[Project1].[Milliseconds] AS [Milliseconds],
[Project1].[Bytes] AS [Bytes],
[Project1].[UnitPrice] AS [UnitPrice]
FROM ( SELECT
 [Limit1].[PlaylistId] AS [PlaylistId],
 [Limit1].[Name] AS [Name],
 [Join1].[PlaylistId] AS [PlaylistId1],
 [Join1].[TrackId1] AS [TrackId],
 [Join1].[TrackId2] AS [TrackId1],
 [Join1].[Name] AS [Name1],
 [Join1].[AlbumId] AS [AlbumId],
 [Join1].[MediaTypeId] AS [MediaTypeId],
 [Join1].[GenreId] AS [GenreId],
 [Join1].[Composer] AS [Composer],
 [Join1].[Milliseconds] AS [Milliseconds],
 [Join1].[Bytes] AS [Bytes],
 [Join1].[UnitPrice] AS [UnitPrice],
 CASE WHEN ([Join1].[PlaylistId] IS NULL) THEN CAST(NULL AS int) ELSE 1 END
```

```
AS [C1]
 FROM   (SELECT TOP (1) [Extent1].[PlaylistId] AS [PlaylistId], [Extent1].[Name] AS
[Name]
  FROM [dbo].[Playlist] AS [Extent1]
  WHERE 3 = [Extent1].[PlaylistId] ) AS [Limit1]
 LEFT OUTER JOIN  (SELECT [Extent2].[PlaylistId] AS [PlaylistId], [Extent2].[TrackId]
AS [TrackId1], [Extent3].[TrackId] AS [TrackId2], [Extent3].[Name] AS [Name],
[Extent3].[AlbumId] AS [AlbumId], [Extent3].[MediaTypeId] AS [MediaTypeId],
[Extent3].[GenreId] AS [GenreId], [Extent3].[Composer] AS [Composer],
[Extent3].[Milliseconds] AS [Milliseconds], [Extent3].[Bytes] AS [Bytes],
[Extent3].[UnitPrice] AS [UnitPrice]
  FROM  [dbo].[PlaylistTrack] AS [Extent2]
  INNER JOIN [dbo].[Track] AS [Extent3] ON [Extent2].[TrackId] = [Extent3].[TrackId] )
AS [Join1] ON [Limit1].[PlaylistId] = [Join1].[PlaylistId]
)  AS [Project1]
ORDER BY [Project1].[PlaylistId] ASC, [Project1].[C1] ASC
```

Buuuwvvvvv!!! Açıkçası ben bu tip bir tablonun eğer maliyet kaybı yoksa ısrarla eklenmesinden yana değilim. Yani Ado.Net Entity Framework gibi düşünüyorum. Fakat yazımızın başında da belirttiğimiz üzere ara tablonun, **One-To-Many** ilişkiler için gerekli olanlar dışında ek alanlar da içerebileceği durumlar söz konusu olabilir. Bu alanların zaman içerisinde tabloya eklenmesi ve **Entity** tarafına da taşınması gerektiği hallerde, **Entity Model** üzerinde söz konusu ara tablonun karşılığının olması, sadece **Conceptual Model** üzerinde müdahaleler yaparak işi kurtarmamızı sağlayabilir. Teorik olarak. Yine de anlattıklarımızın **RC sürümü** üzerinde yazıldığını belirtmek isterim. Yani ilerleyen sürümde farklılıklar söz konusu olabilir. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

RelationEntityCreation_RC.rar (52,15 kb) **[örnek Visual Studio 2010 Ultimate RC sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Screencast - WCF RIA Services, OData, Excel PowerPivot (2010-03-23T09:05:00)

screencast,wcf ria services,ado.net entity framework 4.0,ado.net entity framework,wcf,.net ria services,odata,powerpivot,excel 2010,wcf eco system,

Merhaba Arkadaşlar,

Yeni bir maceraya daha hazır mısınız? Evet dediğinizi duyar gibiyim. Bu sefer kobay haline getirdiğimiz **Chinook** veritabanına yine **Entity Framework 4.0**üzerinden bir **WCF RIA Service** yardımıyla erişiyoruz. Ancak **Domain Service** tipini üretirken **OData(Open Data Protocol)** desteği vereceğini belirtiyoruz. Bu durumda servisimizin çıktı olarak ürettiği **Feed** içeriğinin **OData** standartlarını destekleyen herhangibir ürün tarafından kullanılabileceğini belirtmiş oluyoruz. İstemci tarafında ise **Excel 2010 Beta** ürününe bir **AddIn** olarak gelen **PowerPivot** aracından yararlanıyoruz. Buna göre servisin sunduğu veri içeriğini, sadece bir iki hareketle **Excel** üzerinde kullanılabilir halde çekmiş oluyoruz. Söz konusu fonksiyonellik sayesinde veriyi ister ızgara görünümünde bırakabilir ister detaylı grafiklerini çıkartabiliriz. İşin güzel yanı, servisi dünyanın herhangibir noktasındaki bir sunucu üzerinde host edebilecek olmamız. Dolayısıyla **Excel PowerPivot** aracı söz konusu servise bağlanabildiği sürece verinin en güncel halini çekebiliyor olacak. E ne duruyorsunuz? **NedirTv?** sponsorluğundaki görsel dersimizi izlemeye buyrun.

Süre : 14:52

Boyut : 24.5 Mb

Download etmek veya izlemek için

Workflow Services - Custom Authorization (2010-03-22T13:30:00)

workflow services,wf 4.0,wcf 4.0,

 Merhaba Arkadaşlar,

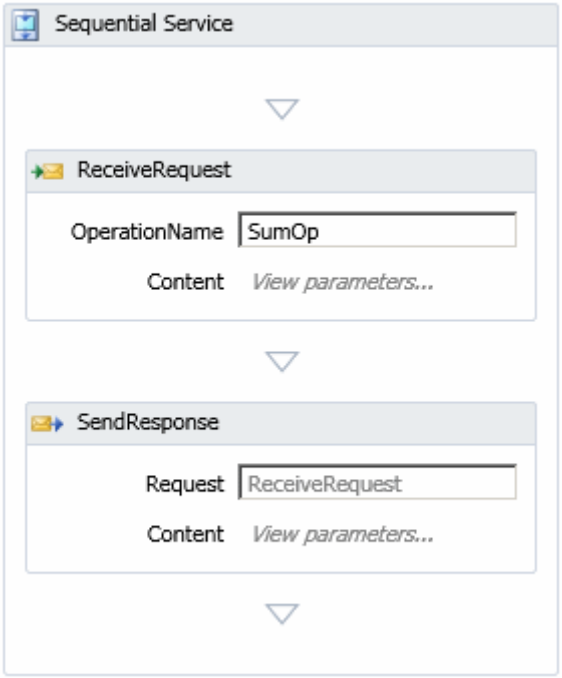
Aşçılık zevkli ama bir o kadarda zor bir zanaattır. Hatta bazen o kadar zor bir zanaat olur ki, aşcının aldığı maaşı pek çok yazılımcı iki yılda kazanamaz. Tabi bu tip aşçılar işin ehli olan kişilerdir. Heleki tek bir mutfak değilde dünya mutfağının seçkin olanlarına ait becerileri bulunanlara paha biçilemez. Türk Mutfağından Japon mutfağına, Meksika yemeklerinden İtalyan spesiyallerine, Fransız tatlılarından Okyanus deniz ürünlerine ve daha nicelerine...Tabi bir aşçı için olmassa olmazlardan biriside yemeği için gerekli olan malzemelerin kalitesidir. Kaliteli zeytinyağı, hamur ve baharat ile yapılan spagettinin, kalitesiz olanlar ile yapılanı arasında dağlar kadar fark olabilir. Spagetti demişken bu günkü yazımızda neler yapacağımıza da bir bakalım dilerseniz. Aşçı olarak bu gün elimizde zor bir tarif var. Malzemelerimiz belli ama pişecek olan yemeğin yapımı biraz zahmetli. Haydi gelin hiç vakit kaybetmeden önlüğümüzü takıp klavyenin başına geçelim.

Bu yazımızda **.Net Framework 4.0** tarafında geliştireceğimiz **Workflow Service'** lerde yetkilendirme işlemini nasıl sağlayabileceğimizi görmeye çalışacağız. Ne yazık ki **Authorization** işlemini kolaylaştırmak adına hazır bir yapı mevcut değil. Bu nedenle biraz kodlama yapmamız ve çalışma zamanının işleyişine bu şekilde müdahale etmemiz gerekiyor. Hatta yapacağımız özelleştirme öylesine etkili olacak ki, aradan **Doğrulamayı(Authentication)** bile çıkaracağız farkına varmadan. Ama önce yemek için gerekli malzemelerimizin neler olduğuna bir bakalım.

- Bir adet **Workflow Service.**
- Bir adet konfigurasyon dosyası**(Web.config).**
- **System.IdentityModel.dll** referansı.
- **ServiceAuthorizationManager** türevli bir sınıf.
- **Windows** üzerinde tanımlanmış roller ve bu roller içerisinde yer alan kullanıcılar.

Tabiki malzemeleri tedarik etmek yeterli değil. Birde tarifi bilmek lazım öncelikli olarak yetkilendirme işlemini üstelenen bir sınıf yazmamız gerekiyor. çok doğal olarak bu sınıfın**Workflow Service** çalışma zamanı tarafından değerlendirilebilmesi için

konfigurasyon dosyası üzerinde de gerekli düzenlemeleri yapmalıyız. Sonrasında ise işi istemci tarafından gelen taleplere bırakıyor olacağız. İşe ilk olarak aşağıdaki gibi bir **Workflow Service** projemiz olduğunu varsayarak başlayalım.



Calculus.xamlx içeriğimiz ise aşağıdaki gibidir;(*Sadece Sequence elementi içeriği verilmiştir)*

```
<p:Sequence DisplayName="Sequential Service"
sad:XamlDebuggerXmlReader.FileName="D:\Projects\Workflow Foundation
4.0\UsingCustomAuthorization\UsingCustomAuthorization\CalculusService.xamlx"
sap:VirtualizedContainerService.HintSize="277,336">
  <p:Sequence.Variables>
    <p:Variable x:TypeArguments="CorrelationHandle" Name="handle" />
    <p:Variable x:TypeArguments="x:Int32" Name="X" />
    <p:Variable x:TypeArguments="x:Int32" Name="Y" />
    <p:Variable x:TypeArguments="x:Int32" Name="Sum" />
  </p:Sequence.Variables>
  <sap:WorkflowViewStateService.ViewState>
    <scg3:Dictionary x:TypeArguments="x:String, x:Object">
      <x:Boolean x:Key="IsExpanded">True</x:Boolean>
    </scg3:Dictionary>
  </sap:WorkflowViewStateService.ViewState>
  <Receive x:Name="__ReferenceID0" CanCreateInstance="True"
DisplayName="ReceiveRequest" sap:VirtualizedContainerService.HintSize="255,86"
OperationName="SumOp" ServiceContractName="p1:IService">
    <Receive.CorrelatesOn>
      <MessageQuerySet />
```

```
    </Receive.CorrelatesOn>
    <Receive.CorrelationInitializers>
      <RequestReplyCorrelationInitializer CorrelationHandle="[handle]" />
    </Receive.CorrelationInitializers>
    <ReceiveParametersContent>
      <p:OutArgument x:TypeArguments="x:Int32"
x:Key="XValue">[X]</p:OutArgument>
      <p:OutArgument x:TypeArguments="x:Int32"
x:Key="YValue">[Y]</p:OutArgument>
    </ReceiveParametersContent>
  </Receive>
  <SendReply Request="{x:Reference __ReferenceID0}" DisplayName="SendResponse"
sap:VirtualizedContainerService.HintSize="255,86">
    <SendParametersContent>
      <p:InArgument x:TypeArguments="x:Int32" x:Key="SumResult">[X +
Y]</p:InArgument>
    </SendParametersContent>
  </SendReply>
 </p:Sequence>
```

Aslında **Calculus** isimli **Workflow Service** içerisinde yer alan **SumOp** isimli operasyonun görevi çok basit ve bellidir. **Int32** tipinden iki sayısal değerin toplanması ve sonucunun istemci tarafına geri döndürülmesi. Bizim hedefimiz sadece yetkisi olan kişilerin bu operasyonu çalıştırmasıdır. Bu durumda güvenlik ile ilişkili olarak yetki kontrolünün özel bir sınıf tarafından yapılması gerekmektedir. Söz konusu sınıf **System.ServiceModel isim alanı(Namespace)** altında yer alan **ServiceAuthorizationManager** tipinden türetilmelidir. İçeriğini ise çok sade olarak aşağıdaki gibi tasarlayabiliriz.

```
using System.Collections.Generic;
using System.Security.Principal;
using System.ServiceModel;

namespace UsingCustomAuthorization
{
  public class Authorizer
    : ServiceAuthorizationManager
  {
    protected override bool CheckAccessCore(OperationContext operationContext)
    {
      var authCtx = operationContext.ServiceSecurityContext.AuthorizationContext;
      var identities=(List<IIdentity>)authCtx.Properties["Identities"];

      foreach (var identity in identities)
      {
        var winIdentity = identity as WindowsIdentity;
```

```
            if (winIdentity != null)
            {
               var winPrincipal = new WindowsPrincipal(winIdentity);
               return winPrincipal.IsInRole("Administrators");
            }
         }

         return false;
      }
   }
}
```

Authorizer sınıfı içerisinde **CheckAccessCore** metodu ezilmiş ve parametre olarak gelen **operationContext** değişkeninden yararlanarak gerekli yetki kontrolü yapılmıştır. Buna göre**Workflow Service'** ten talepte bulunan bir **Windows** kullanıcısı, servisin host edildiği makinede tanımlı ise, **Administrators** rolünde olup olmadığı kontrol edilmekte ve buna göre geriye **true**veya **false** değeri döndürülmektedir. Tahmin edileceği üzere **CheckAccessCore** metodunun geriye **false** değer döndürmesi güvenlik hatasına yol açacaktır. Burada unutulmaması gereken bir noktayı da hatırlatmak yarar var. örneğimizde konuyu son derece basit bir şekilde ele almak istediğimizden, doğrudan **Administrator** rolünün kontrolünü yapıp işin içinden sıyrılmaktayız. Oysaki yetki kontrolü için harici bir listeden yararlanılabilir. Bu liste **web.config** dosyasında **appSettings** kısmında tutulabileceği gibi bir **Text** dosya içerisinde veya **veritabanı**üzerindeki bir **tabloda** konuşlandırılabilir. Yinede varmak istediğimiz noktayı anladığınızı düşünerek devam ediyorum.

Sırada çalışma zamanı için **Authorizer** sınıfının yetki kontrolü amacıyla kullanılacağını bildirmemiz gerekiyor. Bunun için **web.config** dosyasını aşağıdaki şekilde düzenlememiz yeterli olacaktır.*(Bu arada projemize **System.IdentityModel.dll assembly'** ını referans etmeyi unutmamalıyız. Aksi takdirde **authCtx** üzerinden hiç bir özelliğe erişemeyiz)*

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceAuthorization
serviceAuthorizationManagerType="UsingCustomAuthorization.Authorizer,
UsingCustomAuthorization"/>
          <serviceMetadata httpGetEnabled="true"/>
          <serviceDebug includeExceptionDetailInFaults="false"/>
```

```
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <protocolMapping>
    <add scheme="http" binding="wsHttpBinding"/>
  </protocolMapping>
 </system.serviceModel>
 <system.webServer>
  <modules runAllManagedModulesForAllRequests="true"/>
 </system.webServer>
</configuration>
```

Dikkat edileceği üzere **serviceAuthorization** elementi ile yetkilendirme davranışını ele alacak **Authorizer** tipi belirlenmiştir. Ayrıca iletişimin güvenli olmasını sağlamak adına**protocolMapping** sekmesinde **wsHttpBinding** bağlayıcı tipinin kullanılacağı bildirilmiştir. İşte bu kadar. Artık yemeğimizi orta ateşte 40 dakika kadar pişirip servis edebiliriz. Tabi servis etmeden önce tadına bakmak gerekmektedir. Nasıl mı?

öncelikli olarak **Administrator** rolünde olan ve olmayan iki test kullanıcımız olduğunu düşünelim. Ben, örneği geliştirmekte olduğum makinede bu amaçla **bsenyurt** ve **runi** isimli iki kullanıcı oluşturdum. Bu kullanıcılardan **bsenyurt Administrator** rolünde iken **runi User** rolü içerisinde yer almakta. Dolayısıyla test sonuçlarımıza göre **runi** isimli kullanıcı talebi karşılığında **Access Denied** hata mesajını almalı. Bakalım gerçektende böylemi oldu?

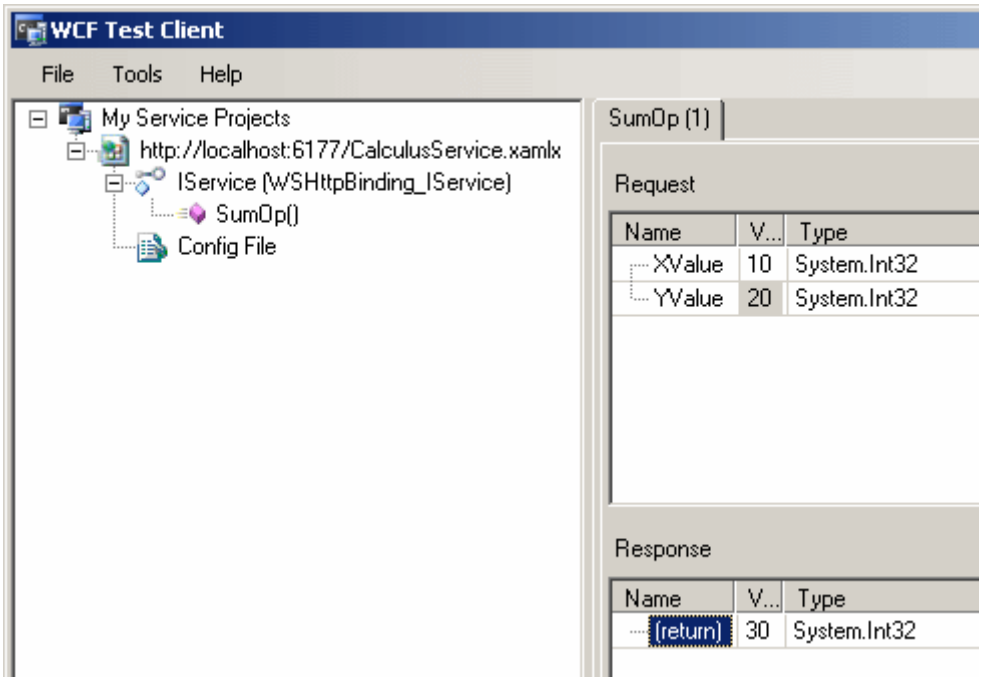
WcfTestClient uygulamamızı **Run As..** komutu yardımıyla önce **bsenyurt** kullanıcısı ile çalıştıralım. Eğer **Asp.Net Developement Server**' ı **Visual Studio** ortamına **Attach**'larsak **CheckAccessCore** metodu içeriğini de **debug** edebiliriz. Buna göre **debug** modunda aşağıdaki sonuçlar ile karşılaşırız.

```
protected override bool CheckAccessCore(OperationContext operationContext)
{
    var authCtx = operationContext.ServiceSecurityContext.AuthorizationContext;
    var identities=(List<IIdentity>)authCtx.Properties["Identities"];
    foreach (v
    {
        var winI
        if (winI
        {
            var winPrincipal = new WindowsPrincipal(winIdentity);
            return winPrincipal.IsInRole("Administrators");
        }
    }
```

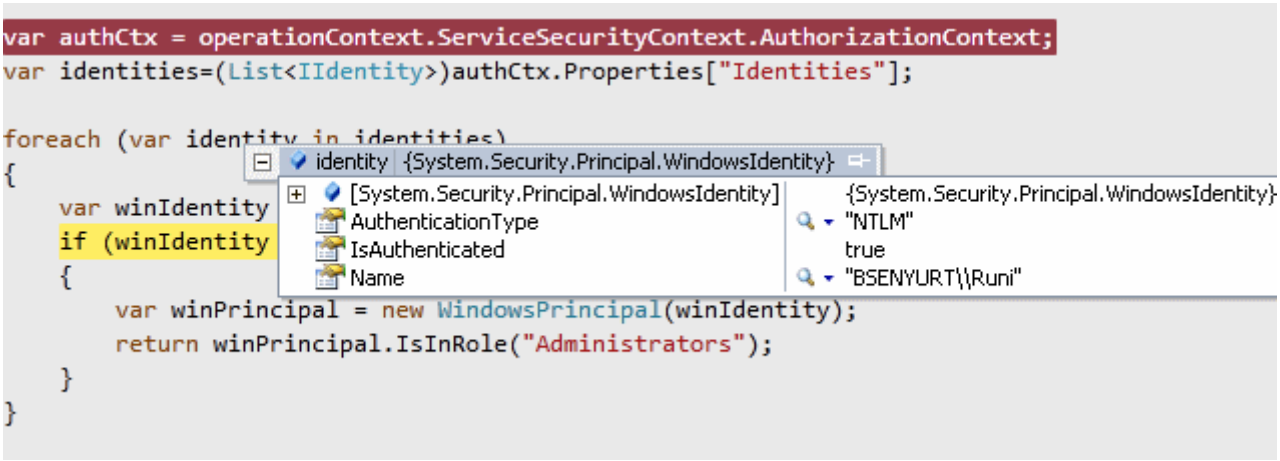
identities Count = 1
[0] {System.Security.Principal.WindowsIdentity}
[System.Security.Principal.WindowsIdentity] {System.Security.Principal.WindowsIdentity}
AuthenticationType "Kerberos"
IsAuthenticated true
Name "ISTANBUL\\bsenyurt"

Görüldüğü üzere **bsenyurt** kullanıcısı doğrulanmıştır. **IsAuthenticated** değerinin **true** olduğuna dikkat edelim.

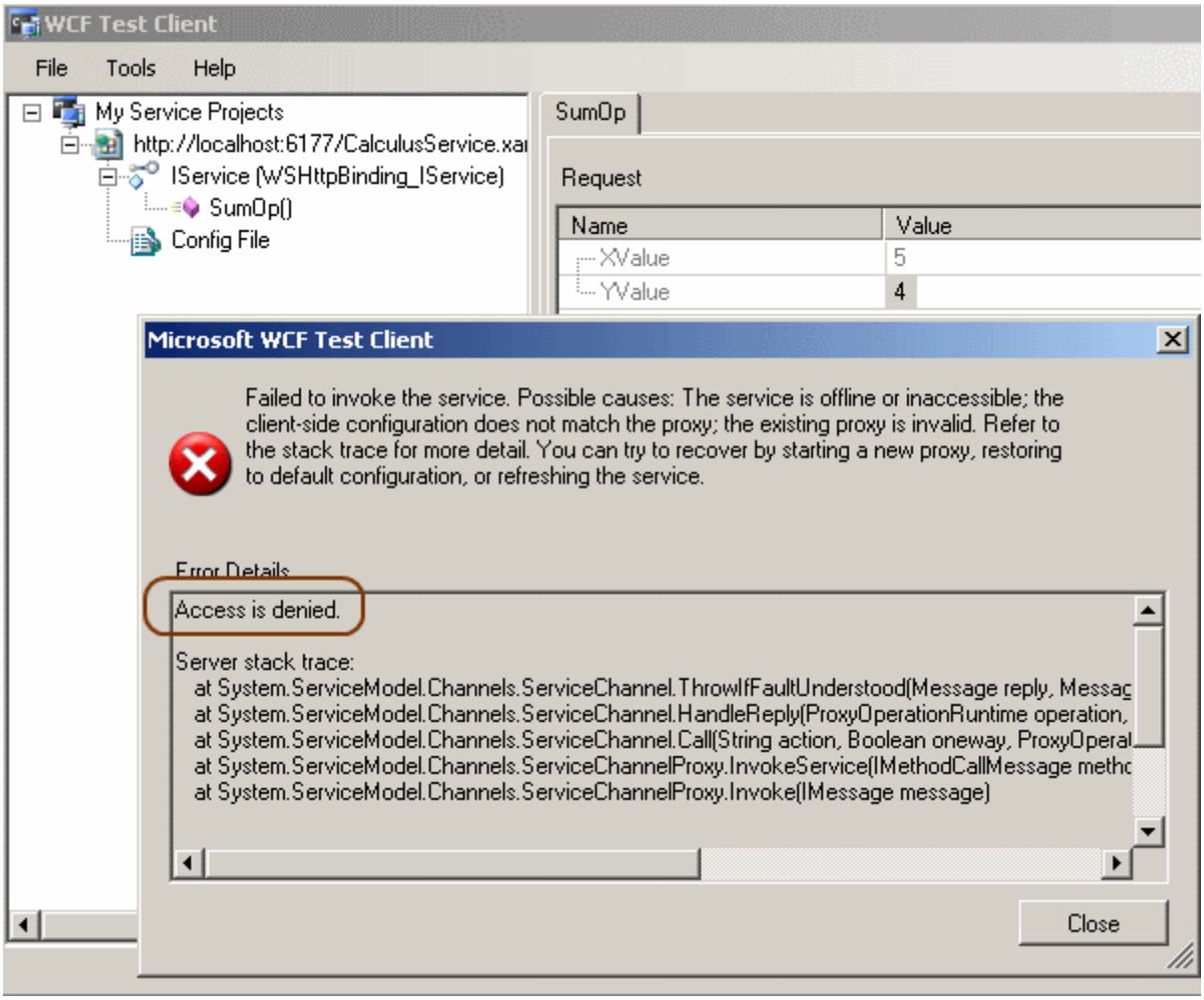
Şimdi **WcfTestClient** uygulamasının sonuç ekranına bakarsak toplama işleminin başarılı bir şekilde gerçekleştirildiğini görebiliriz.



Şimdi de **WcfTestClient** aracını **Runi** isimli kullanıcı ile çalıştıralım. **Debug** modda aşağıdaki sonuçlar ile karşılaşırız.



Tahmin edileceği üzere **Runi** isimli kullanıcı da doğrulanmıştır. Nitekim servisin çalıştırıldığı makine de tanımlı bir **Windows** kullanıcısıdır.
Ancak **WcfTestClient**uygulaması üzerinden bir toplama işlemi talebinde bulunulduğunda hata mesajı ile karşılaşılacaktır.



Volaaaaa!!! Bu çok doğaldır. Nitekim **Runi** isimli kullanıcı **Administrators** grubuna dahil değildir ve bu yüzden yetki kontrolünden geçememiştir.

Yapmış olduğumuz bu çalışmaya göre bir **Workflow Service'** ini host ettiğimiz sunucu üzerindeki **Windows** kullanıcılarından ve dahil oldukları grup bilgilerinden yararlanarak az bir kodlama ile doğrulama ve yetkilendirme işlemlerini gerçekleştirebiliriz. Afiyet olsun

UsingCustomAuthorization_RC.rar (17,67 kb) **[örnek Visual Studio 2010 Ultimate RC sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Screencast - Entity Framework, WCF RIA Services, Silverlight 4.0 (2010-03-19T22:56:00)

ado.net entity framework,wcf ria services,silverlight,wcf,silverlight 4.0,wcf eco system,

Merhaba Arkadaşlar,

Bildiğiniz üzere **2010** yılı **Microsoft** geliştiricileri açısından epey bir hareketli başladı. Aslında ayak sesleri **2008** yılındaki Profesyonel Geliştiriciler Konferansında**(Microsoft PDC)** duyulan pek çok ürünün artık nihai sürümlerinin çıkacağı günlere yaklaşıyoruz. Aldığımız duyumlar Nisan ayı içerisinde .Net Framework 4.0 ve Visual Studio 2010 tarafında Relase sürümlerinin en azından RTM sürümlerinin çıkacağı yönünde. 2008 ve belki de daha öncesinden beri süre gelen zaman içerisinde beni en çok şaşırtan ürünlerinden birisi de **Silverlight**. Daha dün gibi ilk versiyonunu hatırladığımız ürünün geçtiğimiz günlerde **MIX2010** ile birlikte **4.0** sürümünün çıktığı duyuruldu. **Visual Studio 2010 Beta 2** ile kullanıp bakabildiğimiz ama **RC** sürümünde kullanılamayan sürüm artık kullanılabilir halde Bunun için Silverlight resmi sitesinden gerekli kurulumları yapmanız yeterli olacaktır. Şu aşamada **Visual Studio 2010 RC** ve **Visual Studio 2008 SP1** sürümlerinde ele alabiliyorıuz.

Silverlight tarafında beni en çok ilgilendiren konuların başında ise sunucu kaynaklarının istemci tarafından kullanılabilmesinde önemli bir rol üstelenen **WCF RIA Service**' ler gelmekte. Eski adıyla **.NET RIA Service**' lerin **WCF Eco System** içerisinde **WCF RIA Service** olarak anılmaya başlandığını biliyoruz. Daha öncesinde WCF RIA Service' ler ile ilişkili çeşitli blog yazılarım oldu ancak görsel anlatım ne denli güçlü olduğunu hepimiz gayet iyi biliyoruz. İşte bu felsefe ile ve NedirTv? desteğiyle hazırladığımız bu görsel dersimizde **WCF RIA Service**' lerinin kullanımına dair basit bir **Hello World** uygulaması geliştiriyor olacağız. üstelik **Domain Service** tarafında **Entity Framework 4.0** sağlayıcısından yararlanarak veri sunumunu gerçekleştireceğiz. İyi seyirler dilerim.

Boyut : 23.2 Mb

Süre : 15:32

Download Etmek veya İzlemek İçin

SilverlightApplication2.rar (2,37 mb) **[örnek Visual Studio 2010 RC sürümü üzerinde geliştirilmiş ve test edilmiştir]**

[Screencast - Windows Server AppFabric - Application Import ve Export İşlemleri (2010-03-18T10:00:00)]

wcf,wcf 4.0,screencast,windows server app fabric,dublin,

Merhaba Arkadaşlar,

özellikle HTTP bazlı servis uygulamalarının geliştirilmesi sırasında göz önüne alınması gereken önemli noktaların başında **IIS(Internet Information Services)** üzerine yapılan dağıtım prosedürleri gelmektedir. Geliştirme sürecinde genellikle **Asp.Net Development Server** gibi sanal sunucular üzerinden çalıştırılan servislerin önce yerel **IIS** ortamından yayınlanıp test edilmeleri yaygındır. Sonrasında bir **UAT(User Acceptance Test)**makinesi üzerinde yer alan **IIS** sunucuna dağıtımları yapılarak söz konusu testlerin tekrar edilmesi sağlanır. Eğer herhangibir sorun yoksa, çoğunlukla UAT makinesi ile bire bir aynı olan **Production** makinesine taşınması işlemi gerçekleştirilir.

Günümüzde servis tabanlı uygulamalar pek çok büyük çaplı projede baş rolde yer almaktadır. Bu nedenle servislerin **IIS** gibi ortamlar üzerine yayınlanmalarının, izlenmelerinin öneminin daha da arttığı gözlemlenmektedir. özellikle **IT** çevresinde yer alan ekiplerin, **IIS** ortamına dağıtılacak servisler ile ilişkili olarak daha çok yönetimsel araca sahip olmaları önemlidir. Bu noktada bilinen kod adı **Dublin** olan **Windows Server AppFabric**' in getirdiği önemli yenilikler bulunmaktadır. İşte bu görsel dersimizde söz konusu yeniliklerden birisini daha incelemeye çalışıyor olacağız. çok basit olarak **WCF servislerinin IIS** üzerine bir **Deployment Package** haline dönüştürüldükten sonra **Import** edilmesi ve **IIS** üzerinde var olan bir **WCF Servis Uygulamasının** dış ortama paketlenerek sunulması**(Export Application)** işlemlerine göz atmaya çalışacağız. Keyifli seyirler dilerim.

Süre : 22:02

Boyut : 37.4 Mb

İzlemek veya download etmek için

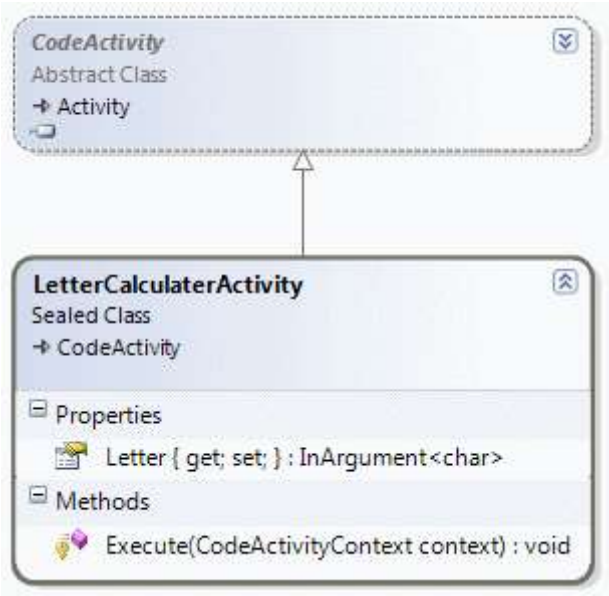
[Workflow Foundation 4.0 - Paralel Olmak ya da Olmamak (2010-03-16T11:15:00)]

wf,wf 4.0,parallel programming,.net framework 4.0,

 Merhaba Arkadaşlar,

Pek çoğumuz ünlü İngiliz şairi **Shakespeare**' in adını ve eserlerini bir şekilde duymuş, okumuş veya seyretmişizdir. Yaşadığı **1564-1616** yılları arasında yazdığı sayısız **Komedi, Trajedi** ve **Romanesk** bulunmaktadır. Bunlar yüz yıllar boyu Tiyatrolarda sergilenmiş ve edebi değeri yüksek eserlerdir. **Shakespeare** dendiğinde insanın aklına hemen **Romeo ve Juliet, Hamlet, Othello, Machbeth** gibi eserleri gelmektedir. Aslında **Edebiyat**' tan çok fazla anlamam. Büyük bir ihtimalle **Matematikçi** olduğum içindir.
Dolayısıyla **Shakespeare** gibi ünlü şarilerin eserlerini hayatım boyunca çok fazla dikkate almamışımdır. Tabi son zamanlarda bu tip kült klasiklerin **NTV Yayınlarından** çıkan çizgi serileri yer almakta. En azından çocuklarımızın okuması için bir hamlede bulunabiliriz. Her ne kadar **Shakespeare**' i çok fazla okumasam da, **Mel Gibson**' ın oyunculuğuyla parladığı **Hamlet**filmini seyretmişimdir. Her ne kadar **Shakespeare**' in eserlerindeki anlamı, derinliği tam olarak bilemesem de, şiirindeki şu meşhur mısrayı hiç unutmam;**"Olmak ya da olmamak; işte bütün mesele bu"** . Bu konuya nereden mi geldik? Kısaca hikayeyi anlatayım.

Geçtiğimiz günlerde **Workflow Foundation 4.0** içerisinde **NativeActivity** türevli bileşenlerde hata yönetiminin nasıl yapılabileceğini incelerken, ne olduysa kendimi **ParallelForEach<T>** aktivitesini çalıştırmaya uğraşırken buldum. Bir türlü istediğim gibi ayrı **Thread** parçaları oluşturulmuyor dolayısıyla aktivite içerisine aldığım işler paralel olarak yürütülmüyordu. O sırada şöyle mırıldandığımı çok net hatırlıyorum; **"Paralel olmak ya da olmamak. Sanırım tüm mesele bu..."** İşte bu yazımızda**ParallelForEach<T>** aktivitesinin örnek senaryoya göre neden çalıştırılamadığını ve buna karşın basit olan çözümün ne olduğu görmeye çalışacağız. öncelikli olarak sorunumuzu örnek bir senaryo üzerinden masaya yatıralım. Bu amaçla aşağıdaki sınıf diagramında görülen **CodeActivity** türevli bir bileşenimiz olduğunu düşünelim.

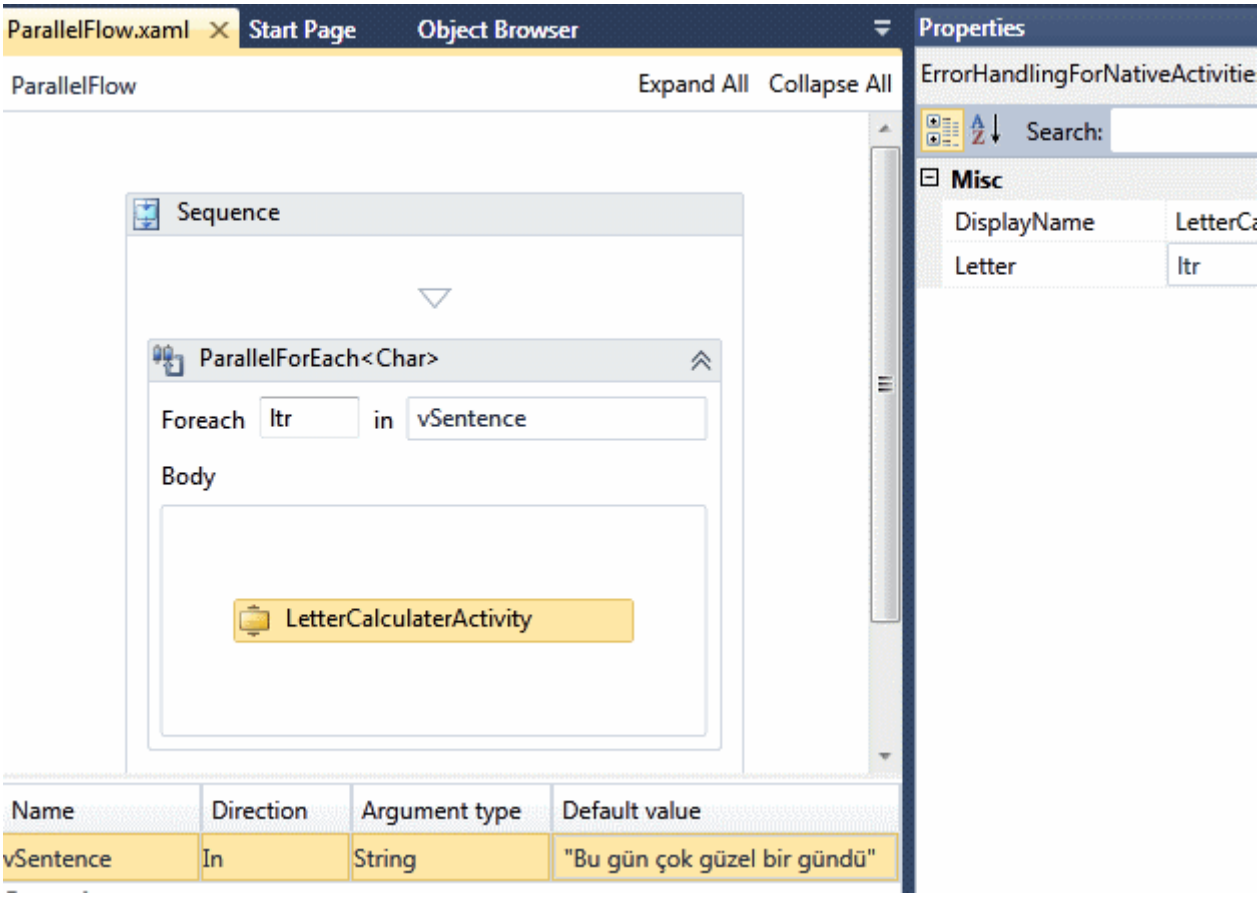


Kod içeriği;

```
using System;
using System.Activities;
using System.Threading;

namespace ErrorHandlingForNativeActivities
{
   public sealed class LetterCalculaterActivity
      : CodeActivity
   {
      public InArgument<char> Letter { get; set; }

      protected override void Execute(CodeActivityContext context)
      {
         Thread.Sleep(1000);
         Console.WriteLine(
            "{0} ASCII = {1} | Current Thread Id : {2}"
            , Letter.Get(context)
            ,((byte)Letter.Get(context)).ToString()
            , Thread.CurrentThread.ManagedThreadId.ToString());
      }
   }
}
```

CodeActivity türevli olan bu bileşenimiz **InArgument<char>** tipinden olan **Letter** isimli özelliğin çalışma zamanı değerini almakta ve **ASCII** kodu karşılığı ile o anki**yönetimli(Managed) Thread Id** değerlerini ekrana yazdırmaktadır. **Execute** metodunda dikkat edileceği üzere şakacıktan ana **Thread**' in 1 saniye süreyle duraksatılması söz konusudur. Peki bu **Activite** bileşeni ne işimize

yarayacak? Bu amaçla tasarım görünümü aşağıdaki gibi olan bir **Workflow** geliştirdiğimizi düşünelim.



Xaml içeriği;

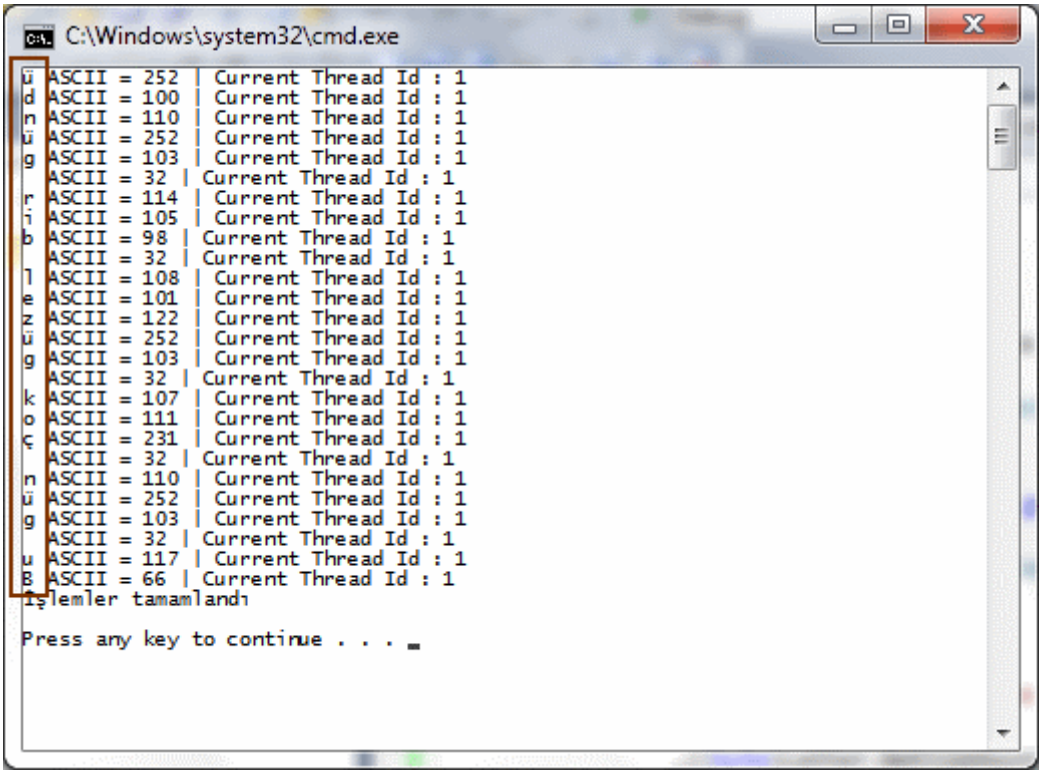
```
<Activity.....>
  <x:Members>
    <x:Property Name="vSentence" Type="InArgument(x:String)" />
  </x:Members>
  <sap:VirtualizedContainerService.HintSize>349,370</sap:VirtualizedContainerService.H
intSize>
  <mva:VisualBasic.Settings>Assembly references and imported namespaces for internal
implementation</mva:VisualBasic.Settings>
  <Sequence sad:XamlDebuggerXmlReader.FileName="D:\Vs 2010\RC\Workflow
Foundation\ErrorHandlingForNativeActivities\ErrorHandlingForNativeActivities\ParallelF
low.xaml" sap:VirtualizedContainerService.HintSize="309,330">
    <sap:WorkflowViewStateService.ViewState>
      <scg:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">True</x:Boolean>
      </scg:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <ParallelForEach x:TypeArguments="x:Char"
```

```
DisplayName="ParallelForEach&lt;Char&gt;"
sap:VirtualizedContainerService.HintSize="287,206" Values="[vSentence]">
    <ActivityAction x:TypeArguments="x:Char">
      <ActivityAction.Argument>
        <DelegateInArgument x:TypeArguments="x:Char" Name="ltr" />
      </ActivityAction.Argument>
      <local:LetterCalculaterActivity
sap:VirtualizedContainerService.HintSize="257,100" Letter="[ltr]" />
    </ActivityAction>
  </ParallelForEach>
 </Sequence>
</Activity>
```

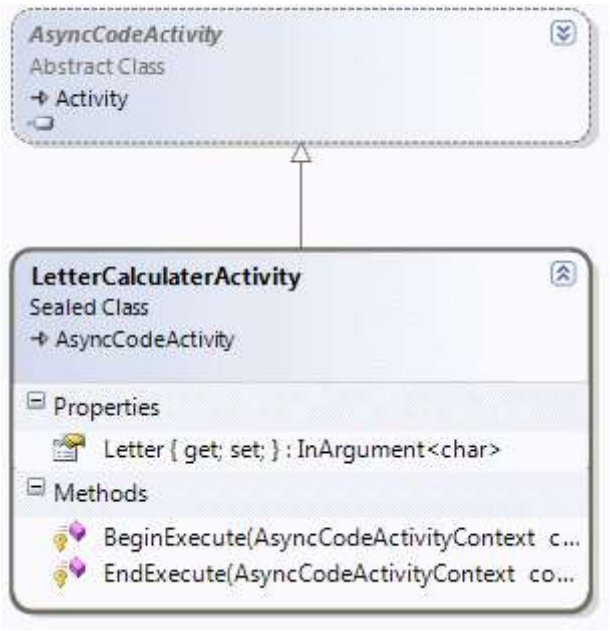
Tasarım zamanından da görüleceği üzere, **ParallelFlow.xaml** içerisinde **ParallelForEach<T>** aktivite bileşeni yer almaktadır. **ParallelForEach<T>**, **char** tipi ile çalışacak şekilde ayarlanmıştır ve **Workflow** için **vSentence** isimli argümanın değerini alıp, söz konusu cümledeki her bir harfi, içerdiği **LetterCalculaterActivity** bileşenine göndermektedir. Bu akıştan çalışma zamanındaki beklentimiz, söz konusu cümledeki harflerin paralel thread' lerin ele alacağı şekilde yorumlaması ve kullanılmasıdır. Bu arada unutmadan, **Main** metodununiçeriğinin aşağıdaki gibi olduğunu düşünelim.

```
using System;
using System.Activities;

namespace ErrorHandlingForNativeActivities
{

   class Program
   {
      static void Main(string[] args)
      {
         ParallelFlow pFlow = new ParallelFlow();
         WorkflowInvoker.Invoke(pFlow);
         Console.WriteLine("İşlemler tamamlandı");
         Console.ReadLine();
      }
   }
}
```

ve buna göre çalışma zamanı sonuçlarına kısaca bir bakalım.



Uppsss!!! Enteresan bir durum söz konusu. **"Bu gün çok güzel bir gündü"** cümlesi ters sırada işlenmiştir. Dahası tüm işlemler **1** numaralı **ThreadId**' ye bağlı olarak gerçekleştirilmektedir. Bir başka deyişle **ParallelForEach<T>** aktivitesi istediğimiz/beklediğimiz şekilde çalışmamıştır. Sorun ne olabilir acaba? Aslında sorundan ziyade yanlış bir çözüm yolu izlediğimizi ifade edebiliriz.
Esasında **ParallelForEach<T>** bileşeninin bu senaryoda işe yarayabilmesi için içerisinde yer alan **CodeActivity** türevli bileşenin de paralel çalışmaya destek vermesi bir başka deyişle asenkron olarak yürütülebiliyor olması gerekmektedir. Ahaaa!! İşte şimdi çözümü bulduk. Buna
göre **LetterCalculaterActivity** bileşeninin**CodeActivity** yerine **AsyncCodeActivity** tipind en türetilmesi ve kodlanması yeterlidir. O halde söz konusu bileşenimizi aşağıdaki şekilde değiştirelim.



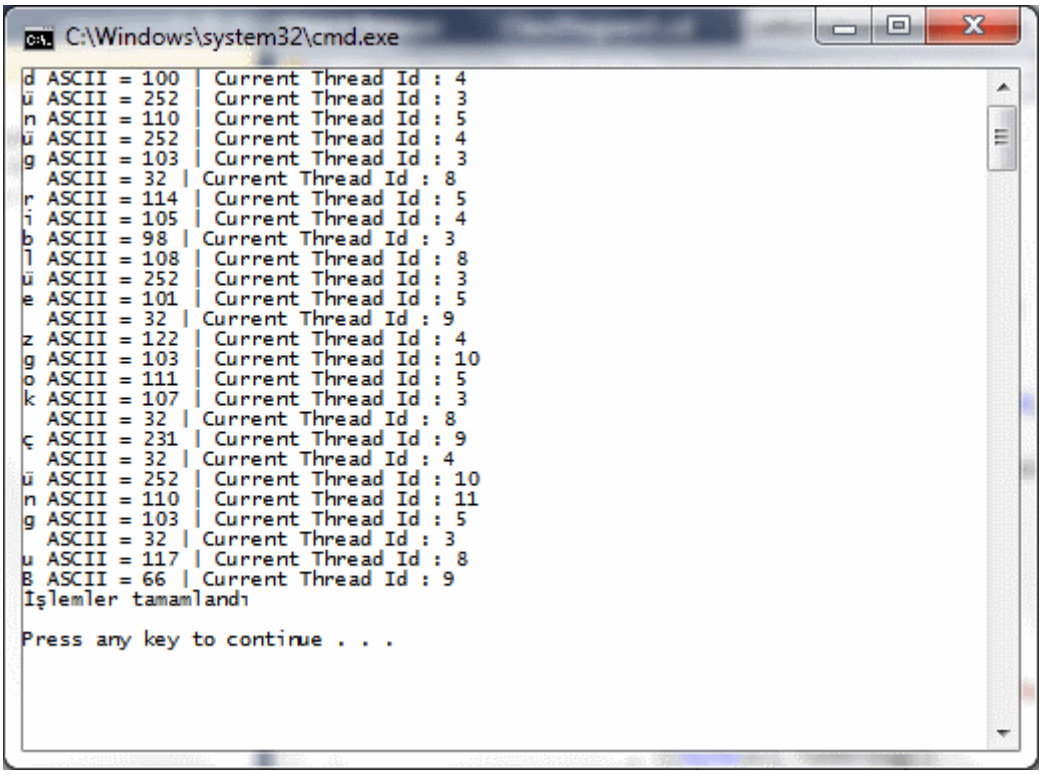

```
using System;
using System.Activities;
using System.Threading;

namespace ErrorHandlingForNativeActivities
{
   public sealed class LetterCalculaterActivity
         : AsyncCodeActivity
   {
      public InArgument<char> Letter { get; set; }

      protected override IAsyncResult BeginExecute(AsyncCodeActivityContext
context, AsyncCallback callback, object state)
      {
         Func<char, bool> dlg = c =>
         {
            Thread.Sleep(1000);
            Console.WriteLine("{0} için hesaplamalar| Current Thread Id : {1}",
c, Thread.CurrentThread.ManagedThreadId.ToString());
            return true;
         };

         context.UserState = dlg;
         return dlg.BeginInvoke(Letter.Get(context), callback, state);
      }

      protected override void EndExecute(AsyncCodeActivityContext context,
IAsyncResult result)
      {
```

```
            bool r = ((Func<char, bool>)context.UserState).EndInvoke(result);
            Console.WriteLine("\t{0}", r);
        }
    }
}
```

Kod parçasına göre, **temsilcilerin(Delegates) BeginInvoke** ve **EndInvoke** metodlarından yararlanılarak aktivite içerisinde asenkron bir işin yürütülmesinin sağlandığını özetleyebiliriz.

Not: AsyncCodeActivity *türevli bileşenlerin nasıl geliştirileceğini daha önceden [Workflow Foundation 4.0 - Custom Async Activity Geliştirmek [Beta 2]] isimli yazımızda değerlendirmiştik.*

Buna göre program kodumuzu yeniden test edersek, çalışma zamanında aşağıdakine benzer sonuçlar ile karşılaştığımızı görebiliriz.



çalışma sırası tam olarak şöyledir ; **"Bu gnü ç kogz eülbir günüd"** . Sakın bu cümleyi okumaya çalışmayın.

Görüldüğü üzere **farklı yönetimli Thread Id** değerleri üretilmiş, üstelik **"Bu gün çok güzel bir gündü"** cümlesi aynı harf sırasına göre ele alınmamıştır. Bir başka deyişle paralel çalışma sağlanmıştır. Tabi çalışma zamanı ve çevresel donanım şartlarına göre bu sıralama her seferinde farklı sonuçlanabilir veya aynı sonuçlar tekrar tekrar elde edilebilir. Aslında bütün mesele de budur zaten. **"Paralel olmak ya da olmamak"**. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ToBeOrNotToBe_RC.rar (48,75 kb) **[örnek uygulama Visual Studio 2010 Ultimate RC Sürümü üzerinde Geliştirilmiştir ve Test Edilmiştir]**

Screencast - WCF Servislerini Windows Server AppFabric Üzerinden İzlemek (2010-03-15T09:35:00)

wcf,wcf 4.0,screencast,windows server app fabric,dublin,

Merhaba Arkadaşlar,

2008 Yılında gerçekleştirilen **Microsoft PDC** konferansında değinilen önemli konulardan birisi de **Dublin**(Nostaljisi için tıklayın) kod adlı projeydi. Bu gün itibariyle **Windows Server AppFabric** adıyla anılan bu projede amaç, **IT** çevrelerinin **WCF** ve **Workflow Service**' lerini daha etkin bir şekilde yönetebilmelerini ve ayrıca izleyebilmelerini sağlamaktır. Buna göre, günümüz **Enterprise** seviyedeki ürünlerin önemli bir parçasını oluşturan servislerin, host edildikleri makineler üzerindeki davranışlarının analiz ve kontrolü, ilgili kitleye*(özellikle sistemciler)* hitap ederekten takip edilebilmektedir. Geliştirilen **WCF** veya **WF** Servislerinin **IIS** üzerinden **host** edilmeleri halinde, çalışma zamanı davranışlarının bilinmesi, sunucuların sağlığı ve performanslarının değerlendirilebilmesi açısından önem arz eden konuların başında gelmektedir. örneğin

- **Host** edilen servislere yapılan çağrıların öğrenilmesi,
- bu çağrılardan hangilerinin başarılı olduğunun veya **istisna(Exception)** ile sonuçlandığının görülmesi,
- **eş zamanlı(Concurrent)** olarak kaç servis operasyon çağrısının sunucu tarafından ele alınabileceğinin ayarlanması,
- servislerin kolay bir şekilde **import/export** edilmeleri
- özellikle **Long Running Process**' lere ait **Workflow Service**' lerinden hangilerinin **beklemede olduğunun(Idle),** hangilerinin tamamlandığının görülmesi,
- an itibariyle eş zamanlı kaç servis örneğinin üretilebileceğinin belirlenmesi,
- **monitoring** ve **persistence** işlemleri ile ilişkili bilgilerin hangi **veri sağlayıcıları(Providers)** üzerinden, nereye kayıt edileceğinin ayarlanabilmesi

ve daha pek çok özellik **AppFabric** bünyesinde yer almaktadır. Windows Server AppFabric' in, görsel dersi hazırladığım tarih itibariyle **Beta 2** sürümü yayınlanmış durumdadır. Web Platform Installer yardımıyla sistemimize yükleyebileceğimiz bu ürünü

tanımak amacıyla NedirTv? üzerinden yayınladığımız ilk görsel dersimizde, **IIS(Internet Information Services)** üzerine dağıtılan bir **WCF** servisinin çalışma zamanı davranışlarını nasıl **izleyebileceğimizi(Monitoring)** incelemeye çalışıyoruz. İyi seyirler dilerim.

Dosya Boyutu : 33.1 Mb

Süre : 23:30

İzlemek veya Download etmek için

EinsteinService.rar (53,12 kb)

.Net' e Nereden Başlamalıyım? (2010-03-15T09:00:00)

Merhaba Arkadaşlar,

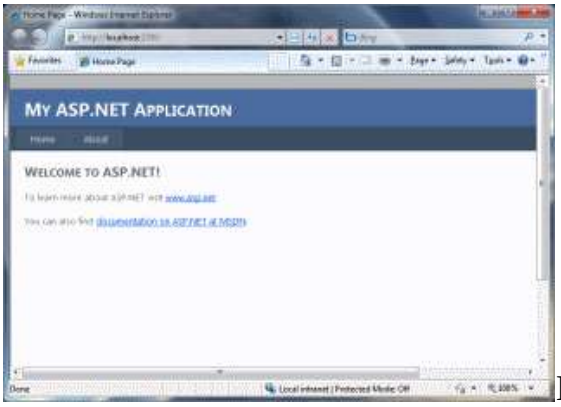
üniversiteden yeni mezun olmuş bir yazılımcı çoğu zaman kendisini, olimpiyat oyunlarındaki 10bin metre maraton yarışı eleme turunun başlangıç noktasındaymış gibi hisseder. Aslında pekte haksız sayılmaz. Bir kaç senelik üniversite eğitimi boyunca kazandığı nosyona rağmen, gerçek hayat çok daha acımasızdır. Sayısız rakip yetmezmiş gibi bir de iş bulma kaygısı söz konusudur. üstelik hangi teknoloji ile ilerleyeceği meçhuldur. Kulaktan dolma bilgiler ile hareket etmek zordur. Bir kulağınızda **.Net** diye bağıran **Bill** amcayı, diğer kulağınızda**iPhone development** diye haykıran **Steve'** i duymakla kalmaz, **Java** aromalı kahve çekirdeklerinin kokusunu da burun deliklerinizden taaa en derin beyin hücrelerinize ulaşıncaya kadar çekersiniz. Şanslısınız ki bu yazıyı okuyorsunuz. çünkü yazıyı okuduğunuz kişi uzun uzun zaman öncesinde **.Net** platformuna geçiş yapmış birisidir. Aslında **Anders Hejslberg'** in ayak izlerini takip ettiğimi ifade edebilirim. Bildiğiniz üzere **Anders**,**Delphi** takımındayken **Microsoft'** a geçmiş ve sonrasında .Net platformu doğmuştur. Kendisini **C#** programlama dilinin Babası olarakta tanımlayanlar vardır. Pekte haksız sayılmazlar. Peki biz yarı teknik dışı bu yazımızda neyi araştırıyor olacağız?

Uzun zamandır **.Net Framework** üzerinde geliştirme yapmaktayım. Bu zaman süresi içerisinde bu alanda ilerlemek isteyen pek çok genç arkadaşımızın bana yönelttiği soruların başında **"Nereden Başlayacağım?"** gelmekte. Bu cevaplanması çok zor olan, yoruma açık ve her profesyonelin farklı şekilde çözebileceği bir soru aslında. Aslında olaya destekli bir

yerden başlamakta yarar var. Bana göre ilk hedef **"hangi ürün?"** olmalı. Sonuç itibariyle .Net çok geniş bir yelpazeyi hedef almakta ve geliştirilebilecek ürün çeşitlerine bakıldığında "Nereden Başlayacağım" sorusunun aslında gerekli bir soru olduğu da ortaya çıkmakta. Herşeyden önce **.Net Framework** ile neler yapılabileceğini, bir başka deyişle neler geliştirilebileceğini bilmek önemli.

Bir kere en basit haliyle **Console** uygulamaları geliştirebilirsiniz ki pek çoğunuzun "Iykkk" dediğinizi duyar gibiyim. Oysaki **C#, VB.Net** gibi programlama dillerinin saf olan özelliklerini, kabiliyetlerini öğrenmek, gerektiğinde test etmek için bulunmaz bir uygulama çeşididir. Nitekim sizi **Windows Forms,WPF** veya **ASP.NET**'in ve benzerlerinin görselliğinden uzaklaştırır ve sadece programlama diline odaklanmanızı sağlar. Tabi şu önemli noktayı da unutmamak gerekir; örneğin **Sistem Yöneticileri** bazı işlerini **Console** ekranından yapmayı daha çok tercih ederler. Bu ilk işletim sistemlerinden gelen bir alışkanlık olarak düşünülebilir de.

İkinci bir ürün grubu olarak **Windows Forms** veya **.Net Framework 3.0** sonrasında çıkan **Windows Presentation Foundation** uygulamalarını düşünebiliriz. Bu tip uygulamalar görsel arayüzleri sayesinde kullanıcıların daha fazla ilgisini çekebilecek veya **kullanıcı deneyimi(User Experience)** olarakta bilinen tecrübelerin daha fazla yaşanabileceği ortamlar sunmaktadır. Genellikle **Intranet**tabanlı olan veya sadece masa üstünde yerel olarak çalışan pek çok uygulamanın **Windows Forms** veya **WPF** tabanlı olması söz konusudur. Ancak bu ürün grupları arasında keskin farklılıklar bulunmaktadır. özellikle **WPF** tarafında yazılımcının **XAML** gibi kavramlarla karşı karşıya kalması söz konusudur. **Windows Forms** uygulamaları geliştiren yazılımcılar bile, **WPF** tarafına ilk geçiş yaptıklarında kısa bir süre afallamaktadır. çünkü farklı bir dünyanın kapıları açılmaktadır.



Bir de, **"Ah şu Facebook, google gibi bir fikir bulsam, ya da Amazon gibi bir site geliştirsem de...köşeyi şuradan dönsem..."** diyerek hayaller kuranlar için **ASP.NET** dalı söz konusudur. Web tabanlı uygulamaları geliştirmek aslında sanıldığı kadar kolay değildir. Alt tarafta yatan pek çok temel kavramı bilmek gerekmektedir. **HTML(Hyper Text Markup Language)** den iyi anlamak, yeri geldiğinde **Javascript** ile müdahalelerde bulunmak şarttır. Ama daha ileri seviyelerde, bu alanda ilerlemek isteyen yazılımcıları **AJAX(Asynchron Javascript And Xml)**, **XML(eXtensible Markup Language)**,**JSON(JavaScript Object Notation), MVC(Model View Controller), SEO(Search Engine Optimization), CSS(Cascading Style Sheet)** gibi konular ağırlar. üstelik **Web'** in doğası **Windows/WPF/Console** gibi uygulamalardan çok daha farklıdır. **Stateless** olmak, performans için **Cache'** leme yapmak, kullanıcıyı doğrulamak ve yetkilendirmek, profilini yönetmek, **Web Part'** lar ile sayfayı kişiselleştirilebilir hale getirmek gibi kavramların havada uçuştuğu bir dünyadır.

Tabi **Web** uygulamaları diyince akla son yıllarda **Silverlight** gibi **RIA(Rich Internet Application)** uygulamaları da gelmektedir. Bu tip uygulamalar sayesinde, **Web** tarafındaki kullanıcı deneyiminin en üst seviyeye çıkartılması ve çok daha zengin içeriklerin sunulması söz konusudur. İlk zamanlarında emekleme aşamasında olan bu ürün, son zamanlarda çıkan versiyonları sayesinde çok daha iyi bir duruma gelmiştir. Tabi **Silverlight** tarafında da**WPF** kökleri bulunmaktadır. Bu nedenle **XAML(eXtensible Application Markup Language)** tarafına hakim olmak önemli olabilir. Diğer yandan yazılımcılar, tasarımcıyı geliştiriciyle ayırt eden konsept nedeniyle belki de sadece kod tarafıyla ilgilenmelidir. Nitekim tasarımı kolaylaştırmak adına **Expression** ürün ailesinin kullanılması önerilmektedir.

Yaşananlardan : *Web dünyasına adım attıktan bir süre sonra kendime bir site yapmananın gelişimim açısından çok önemli olacağını düşünmüştüm. www.bsenyurt.com' un ilk temellerini attığımda tasarım yönünden sınıfta kaldığım çok açık ve belliydi. Bir türlü doğru renkleri seçemiyor, sayfanın HTML giydirilmesi konusunda başarı*

sağlayamıyordum. Açıkçası içime sinmeyen bir tasarım oluşmaktaydı. Kod tarafında bir sorun yoktu. Ama zaten ilerleyen zamanlarda çalıştığım tüm yazılım şirketlerinde, web uygulamalarının tasarımı ile ilgilinen ayrı arkadaşların olduğunu görmüştüm. Halen daha da böyledir ve böyle olmasında yarar vardır. Kendi sitemi görsel açıdan toparlayamayacağımı anlayınca çocukluk arkadaşımdan yardım istedim. Kendisi ***Bilkent üniversitei Grafik Tasarım*** *bölümünü burslu okumuş ve yüksek başarı ile bitirmişti. Bir web sitesinde hangi font' ların kullanılacağını, kurumsal kimliği yansıtacak renklerin neler olması gerektiğini, sayfa tasarımında neyin nerede bulunabileceğini çok iyi bilen yetenekli bir arkadaşımdı. Sadece 3 renk kullanarak sitemin görüntüsünü bir anda değiştirmiş ve benim için bir sanat şahaseri gibi görünen arayüzü bir kaç dakika içerisinde oluşturmuştu. Sonra tabi o tasarımı hayata geçiremedim ama yine de bir tasarımcı ile geliştiricinin keskin çizgiler ile bir birlerinden ayrılan ikililer olduklarını da gördüm.*

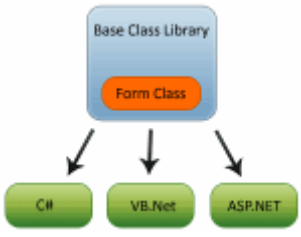
İşte bu keskin çizgiye rağmen her iki tarafın rahat çalışabilmesi adına yazılım şirketlerinin de yatırımlar yaptığını görmekteyiz. Söz gelimi **Microsoft'** un **Expression** ailesi buna verilebilecek en güzel örneklerden birisidir. **Visual Studio** geliştirme ortamı ile olan entegrasyonu sayesinde birlikte çalışılabilirlik hem ayrık hem de bir arada olmasının önemi ortaya çıkmaktadır.

Görsel tarafta sayılabilen ve aslında bilgisayarı olmayanların dahi kullanabileceği bir ürün grubu da söz konusudur. **Mobil** uygulamalar. **Smartphone, PDA**cihazları, kısacası **Windows Mobile** işletim sistemine sahip ve bir anlamda da üzerlerinde **.Net Compact Framework** yüklü olan aygıtlar için geliştirmeler yapılabilir.
Aslında **Mobil** dünyası çok farklıdır ve bu tarafta pek çok zorluk vardır. Cihazların bilgisayarlara göre daha sınırlı kapasitede donanıma sahip olması, ekran boyutlarının küçük olması gibi faktörler bunların arasında sayılabilir.

Yaşananlardan: *Eğitmen olarak yaşadığım ilk kurumsal tecrübem,* ***Boehringer-ingelheim*** *isimli ilaç firması için olmuştur. Tesadüfe bakın ki bu ilk kurumsal eğitimin konusu zayıf olduğum alanlardan birisi olan Mobil Uygulama geliştirmedir. Eğitimi verdiğim sırada* ***.Net Framework 2.0*** *henüz****Beta*** *aşamasındaydı. Eğitimde en çok üzerinde durulan konular, mobil cihazların donanım kapasitelerinin yetersiz olması, offline konuma sık sık düşmeleri nedeniyle sunucu üzerindeki kaynaklar ile olan senkronizasyonlarının zor teknikler ile gerçekleştirilmek zorunda kalışı vs gibi güçlüklerdi. Aslında teorik bilgimiz ne kadar iyi olursa olsun saha tecrübesini yapmadığımız takdirde yeteneklerimizin gerçek hayattaki sınamalarını asla öğrenemeyebiliriz. Hatta söz konusu alanlarla ilişkili olarak pek çok* ***Case Study*** *okusak bile fiziki saha tecrübesinin yerini tutmayacaktır.*

ürün ailesi içerisinde yer alan önemli parçalardan birisi de **Servis** uygulamalarıdır. Bu uygulamalar **.Net Framework 3.0** çıkana kadar çeşitlilik göstermiştir. **XML Web Service**' leri, **Windows Service**' leri, **COM+** veya **MSMQ** hizmetleri vb. **.Net Framework 3.0** sonrasında ise **WCF(Windows Communication Foundation)** ortaya çıkmış ve **Microsoft**' un servis tabanlı uygulama geliştirme modelinin yeni yüzü haline gelmiştir. İşe yeni başlayan bir yazılımcı için servis geliştirmek çok basit öğrenilebilecek bir süreçtir. Aslında bu bir yanılgıdır. Nitekim servislerin asıl hedef noktası **Servis Yönelimli Mimari(Service Oriented Architecture)** tarafıdır. Gerçekten de **SOA** için servis geliştirilecekse, olayların bambaşka bir hal aldığı ve sayısız **SOA** Tasarım Kalıbının bilinmesi gerektiği gibi korkutucu bir senaryo ile karşılaşılmaktadır. Hatta pek çok büyük çaplı projede servisler sadece entegrasyonu sağlamak amacıyla kullanılan birer piyondur.

Yaşananlardan: *Şu anda çalışmakta olduğum firmadan önce uluslar arası bir banka da yine dış kaynaklı yazılımcı olarak görev almaktaydım. Buradaki asli görevim, geliştirilmekte olan **WinForms** tabanlı uygulamanın, banka içerisindeki yabancı sistemler ile olan konuşmaları sırasındaki entegrasyonunu sağlamaktı. Yeri geldiğinde uzak sunucularda bulunan ve sadece soketler aracılığıyla erişilebine **C** tabanlı uygulamalar ile konuşulması, yeri geldiğinde **COM** nesnelerinin **Wrap**edilmiş halleri üzerinde ele alınan bileşenler(**RightFax API** gibi 😬) ile haberleşilmesi, yeri geldiğinde **Windows Service'** leri veya **XML Web Service'** lerinin yazılarak projeye entegre edilmesini gerçekleştirmeye çalıştım. İnanılması güç ama servisleri yazmak ne kadar kolay olduysa da, projenin çalıştığı banka ortamının kısıtları nedeniyle onları **dağıtmak(Deployment)** ve onlarla etkili bir şekilde konuşmak bir o kadar zor oldu. Dolayısıyla yine aynı noktaya geliyoruz. Saha tecrübesi bambaşka bir heyecan...*



Peki ya bir **Windows** uygulaması veya **Web Portalı** ya da basit bir **WCF Servisi** tek başına yeterli midir? çoğu zaman bu uygulamaların tabir yerinde ise button arkası olacak şekilde geliştirildiklerine şahit oluruz. Oysaki büyük çaplı projelerde yazılım mimarları sürekli olarak uygulamanın katmanlarını düşünür ve en uygun olanını tespit ederek tasarlarlar. Bu katmanlarda yer alan vazgeçilmezlerden birisi de **Sınıf Kütüphaneleridir(Class Libraries)** ve görsel bir arabirim sunmadıkları için pek çok yeni yazılımcı tarafından Console uygulaması muamelesine tabi tutulurlar*(Aslında bir Class*

*Library uygulaması içerisine **static Main** metodunu dahil ederek ve**Visual Studio** üzerinde çıktının **Console Application** olacağını söyleyerek Console uygulamasına çevirebileceğinizi de biliyor muydunuz?)* Oysaki **Enterprise Libary**, **.Net Framework**' ün kendisi pek çok sınıf kütüphanesini ihtiva eder. Açıkçası bir **Framework** geliştirilmesi gerektiğinde veya katmanlı mimari tercih edildiğinde her yazılımcı er ya da geç **Class Library** kavramı ile karşılacaktır.

Bitti mi? Aslında saymadığımız pek çok uygulama çeşidi ve kavram söz konusu. **Ado.Net Entity Framework, Workflow Foundation, Parallel Development, Component Development** vb...Yine de buraya kadarki düşüncelerimizi göz önüne alarak yapabileceklerimizi gördüğümüzü var sayalım. Bir yazılımcı için bu yine de yeterli olmayacaktır. **.Net Framework** tarafında yazılımcıların en az bir **.Net** dilini etkin bir şekilde kullanabilmesi şarttır. Bu çoğunlukla **C#** veya **VB.Net** dillerinden birisi olacaktır. Tercih yaparken daha önceden **C++,C** veya **Visual Basic**kökenli olmak karar vermeyi kolaylaştıracaktır. Fakat unutulmaması gereken önemli bir nokta da, gerçek hayat projelerinde başımıza ne gelebileceğinin bilinmemesidir. Bir bakmışsınız sıradaki projede **Iron Pyhton** veya **F#** kullanacaksınızdır. Ya da tamamen platform değiştirip **Java** tarafında ilaveler yapacaksınızdır...Hatta kendinizi **Java** tarafında yazılmış servisler ile konuşacak bir entegarasyonun içerisinde bulabilirsiniz. Kim bilebilir? Son sözü söyleyen yöneticiniz değil midir? 😉 Ama her ne olursa olsun **.Net** tarafında ilerlerdiğimizi varsayarsak, **C#** veya **VB.Net**dillerinin her ikisine de hakim olmak bir avantaj olabilir.

***Yaşananlardan:** Bir kaç yıl önce çalıştığım bir **.Net** projesinde Türkiye' nin önde gelen bir Holding' i için önemli bir **ASP.NET** uygulaması geliştirmekteydik. Ben her zamanki gibi dış kaynak elemanı olarak projeye dahil olmuş ekibin bir parçasıydım. Projede, Holding tarafında yer alan ekip arkadaşlarımız da mevcuttu. Gelin görünkü bizim sahip olduğumuz hazır alt yapı **C#** tabanlı iken, Holding tarafındaki arkadaşlarımız **VB.Net** kökenli idi. Bir arada çalışılacağına göre biz biraz **VB.Net** onlar biraz **C#** yazmak durumundaydı. Aynen de böyle oldu. Ben **ASP.NET** tarafındaki ekranların arka planlarını **VB.Net** ile geliştirirken **Business Layer** tarafı C# tabanlıyıdı. Dolayısıyla bir anlamda dilden bağımsız harkete edebilir formasyonda uçmaktaydık. Nitekim .Net destekli birden fazla dil aynı proje içerisinde kullanılabilirdi.*

Peki neler yapabileceğimizi görmek ve uzmanlaşmak için bir ürün ailesini seçmek, hangi dilleri öğrenmemiz gerektiğini bilmek yeterli midir? Ne yazık ki hayır. 😕 Başlamak için beynimizi paralel çalışabilen parçalara

bölmemiz de gerekecektir. Nitekim öğrenilmesi gereken kavramların sayısı çok fazladır ve bunlar zamanlar acımasız bir şekilde beynimizin her noktasına enjekte edilmeye başlanacaktır. Buna göre biraz programlama dillerini çalışırken biraz **.Net Framework** içeriği araştırılmalı, biraz **Tasarım Desenlerini(Design Patterns)** incelerken biraz mimari modellere bakılmalı, biraz uzmanlaşmak istediğimiz taraftaki*(Windows, Web, Mobile, Service Oriented vb...)* güvenilir makaleleri karıştırırken biraz yeni gelen teknolojilere bakmalı vb... şeklinde bir yol izlenilmelidir.

Ancak şu an için kendimizi bu kadar fazla korkutmaya da gerek yoktur. Nereden mi başlamanız gerekiyor? Bence **.Net Framework** üzerinde geliştirme yapabileceğiniz bir dili iyi bir şekilde öğrenerek. İyi bir şekilde öğrenmekten kastımız, tabiki tüm **anahtar kelimeleri(Keywords)** bilip **"button arkası"** kodlama yapmak demek değildir. Dile öylesine iyi bir şekilde hakim olmasınız ki bilerek, görerek, hissederek, hatasız kodlama yapabilmeli,**Nesne Yönelimli Programlama(Object Oriented Programming)** terimlerini su emen sünger misali yutarak anında uygulayabilmeli veya çözüm için gerekli tasarım kalbını o dil ile birlikte kullanabilmelisiniz.

Sonrası mı?... **.Net Framework**' ün kabiliyetlerini tanımak olabilir. öylesine iyi bilinmelilerdir ki, bir sınıf kütüphanesi içerisinde yazdığınız tipler yardımıyla gerçekleştireceğiniz**IO, Database, XML** gibi işlemleri yeri geldiğinde bir **Windows** uygulamasına, yeri geldiğinde çok az konfigurasyon değişikliği ile belki bir Web uygulamasına açabilecek kadar. Hatta bunlarda yetmeyecektir ki, geliştirdiğiniz projenin nasıl bir mimari modele oturduğunu görerek **.Net Framework** materyallerini kullanmayı bilmek, **AOP(Aspect Oriented Programming)** tadında bir yaklaşım olacaksa eğer, niteliklerin veya konfigurasyon dosyasının ne kadar etkili kullanılabileceğini görmek gerekecetir.

Sonrası mı? Aslında sizi daha da öteye taşıyacak materyaller hemen yanı başınızdadır. Yurt dışından bizzat veya şirketiniz aracılığıyla getirttiğiniz kitaplar, içerisinde yer almakta olduğunuz projeler ve tabiki en önemlisi takım arkadaşlarınız. Yeter mi? Elbette hayır. O yüzden yeni başlayan arkadaşlarım bu koşulları tamamladıklarında lütfen bana mail atsınlar... Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

YazılımcıyızBiz Yayında (2010-03-12T11:00:00)

yazilimciyiz biz,bizspark,websitespark,empower,mvp,microsoft,

Merhaba Arkadaşlar,

Yazılımcıların ortak buluşma noktası **YazılımcıyızBiz** yayında.

Yazılımla ilgili **güncel eğitim, seminer, duyuru, webiner** ve **haberlere** ulaşabileceğiniz sitede ürün odaklı pek çok kaynak ve bilgide yer almakta. Ayrıca tek bir noktadan **BizSpark, WebsiteSpark, Empower** ve diğer bazı programlara ulaşabilir, **Lisanlama** konusunda bilgiler edinebilir ve **Burak** ile **Emre**' nin neler yaptığına bakabilirsiniz.

Yazılım alanında ilerleyen veya ilerlemek isteyen pek çok meslektaşımın mutlaka uğrak noktalarından birisi olması gereken bu siteyi sizlere ısrarla öneririm. Ben **RSS** içeriğini kendi **FeedReader** programıma hemen ekledim.

Bu arada sitenin sağ alt tarafında yer alan(Gönder düğmesinin altı) **Silverlight** uygulamasında görünen ürünlere tıkladığınızda bu konuda ürün odaklı çalışan **MVP**' lerin sektörel görüşlerine yer verilen videolara ulaşabilir, ufkunuzu ve vizyonunuzu profesyoneller ile birlikte daha da ileri götürebilirsiniz. Denemeden geçmeyin derim.

Tesadüf eseri sitedeki iki yazılımcıdan birisi ile adaşız. Ortak pek çok zevkimiz olduğunu ifade edebilirim Pek bana benzemese de şu anda olmak istediğim ideal kiloda görünen bir karakter olduğunu itiraf etmeliyim. çizerin eline sağlık.

Tabi içerikte bazı kelime hataları veya eksiklikler söz konusu olabilir*(örneğin paylaşım yapmak için o kadar kasmama rağmen şu an itibariyle MVP Blog listesinde adımın ve blog adresimin olmayışı gibi)*. Ancak bunlarda siteyi tasarlayan ve içeriği hazırlayan ekip tarafından düzenlenecektir düşüncesindeyim.

Screencast - WCF Data Services - Projections (2010-03-08T11:10:00)

wcf,wcf data services,ado.net data services,screencast,

Merhaba Arkadaşlar,

Yazılımcı bile olsak bu arada sırada spor yapmadığımız anlamına gelmemeli. Bende çalışma arkadaşlarım ile sık sık spor aktivitelerinde bulunuyorum. Zaman zaman parke zeminde o muhteşem NBA yıldızları gibi giyinerek basketbol oynuyor, zaman zaman Ping Pong...Geçtiğimiz günlerde ise bir halı saha maçı organizasyonundaydım. Bu organizasyonun bitmesinden hemen sonra duşumu aldım, günlük kıyafetlerimi giydim ve akşamın sekizinde bilin bakalım nereye gittim...Şirkete Deli mi bu adam...Eve gitse ya...Amacı ne? Nitekim maç sırasında sürekli aklımda olan ve beni dürten bir mesele vardı. Uzun zamandır çekmek istediğim bir görsel ders için şu saatlerde sessiz olan şirket en ideal mekandı.

NedirTv? aracılığıyla hazırladığımız bu görsel dersimizde, **WCF Eco System**' in bir parçası olan **WCF Data Service**' lerinde **Projections** sorgularının nasıl kullanılabileceğini incelemeye çalışıyoruz. Aslında kod adı **Astoria** olan **Ado.Net Data Service**' lerin **1.5 CTP2** sürümünde de duyurulan bu yetenek, zaten **.Net Framework 4.0** içerisine gömülü olarak gelen **WCF Data Service**' ler için standartlaştırılmış bir özellik. Bu amaçla hazırladığımız ve **7,5**dakikayı aşmayan görsel dersimizin faydalı olacağını ümit ediyorum. İyi seyirler dilerim.

Dosya Boyutu : 14.3 Mb

Süre : 7:35

İzlemek veya Download Etmek için

WCF WebHttp Services - Routing (2010-03-08T08:45:00)

wcf,wcf 4.0,webhttp services,restful,non soap,wcf webhttp services,

 Merhaba Arkadaşlar,

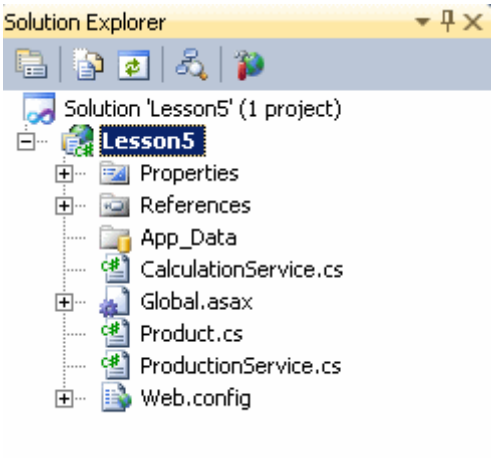
Geçtiğimiz aylarda yurdumuzun en güzel şehirlerinden birisi olan İstanbul' da, oldukça soğuk ve kar yağışlı günler geçirdik. Yandaki resimde görülen manzara çalıştığım **ARI 1** binasının yürüyüş yolu üzerinde çok da yeni olmayan, dokunmatik ekran, **Windows Mobile** gibi ileri özellikleri bulunmayan ancak **Carl Zeiss** mercekli **Sony Ericsson K810i** marka cep telefonum tarafından çekilmiştir. **3.1 Megapiksel** ölçekli çekilen fotoğrafı kaliteli yapan ise elbetteki **Carl Zeiss** mercek. İnsan böyle zamanlarda eğer şartlar yerindeyse*(çay, kahve, sıcak ve sessiz bir ortam, hızlı bir internet bağlantısı)* işiyle ilgili pek çok konuda araştırma fırsatı bulabiliyor. Malum böyle havalarda sizde benim gibi evden çıkmamayı veya mesai sonrası şirkette bir kaç saat durmayı tercih edinenlerdenseniz, yapılacaklar listesinin belkide en başında yenilikleri araştırmak vardır diye düşünüyorum. Malum bir süredir de **WCF Eco System'** in önemli parçalarından birisi olan **WebHttp Service'** lerini incelemekte olduğumuza göre yine bu alanda ilerleyerek devam edebiliriz. Bu günkü konumuz ise **WCF WebHttp** servislerinde **yönlendirme(Routing)** işlemleri olacak.

Pek tabi, bir **WCF REST Service Application** içerisinde birden fazla **WebHttp Service** sınıfı konuşlandırılabilir. Bu çoğunlukla tek bir sınıfın var olduğu hallerde zaman içerisinde artan fonksiyon sayısı nedeniyle kavramsal bütünün bozulmasını engellemek amacıyla alınan bir tedbir olarak düşünülebilir. Aslında gerçek hayatta söz konusu kavramsal bütünlüğün ayrıştırılması ile ilişkili pek çok başarılı örnek yer almaktadır. örneğin **Sql Server 2005** ile gelen meşhur **Adventure Works** veritabanını göz önüne alalım. Burada bildiğiniz üzere çok sayıda tablo çeşitli **şemalara(Schema)** ayrılmıştır. **Production, HumanResource, Sales** vb...Böylece veritabanı **alan modelinin(Domain Model)** kavramsal olarak kolay bir şekilde ayrıştırılması mümkün olmaktadır. İşte benzer durum servis sınıflarının içeriğini oluşturan fonksiyonlar için de geçerli olabilir. Buna göre operasyonların ayrı servis sınıfları altında toplanıyor olması **Adventure Works** örneğindekine benzer kavramsal bir ayrımın yapılabilmesi manasına gelmektedir. Peki bir servisin yürüttüğü operasyonları sınıf bazında ayrıştırmanın uygulanışında nelere dikkat edilmesi gerekmektedir?

Herşeyden önce **WebHttp Service'** leri **URI** üzerinden gelen **HTTP** taleplerini değerlendirmektedir. Yani servise gelen bir talep, servis sınıfı içerisindeki bir operasyon ile ilişkilendirilmektedir. Bu noktada **WCF WebHttp Service'** lerinin

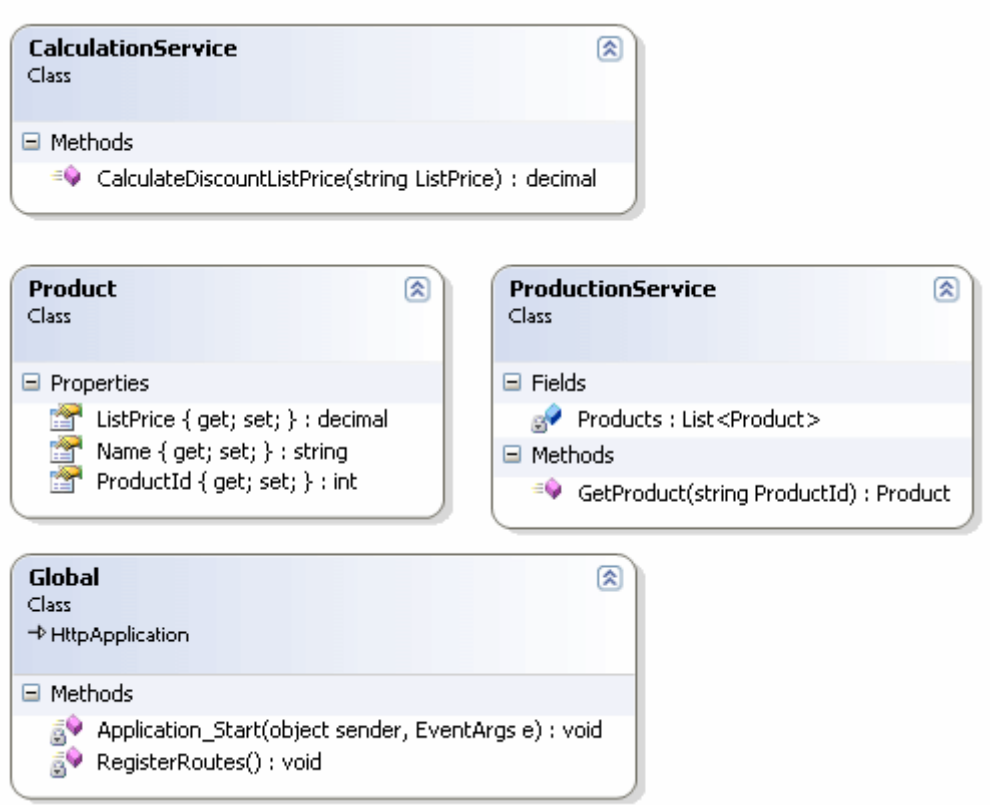
konuşlandırıldığı **Web** uygulamasının **global.asax** içeriğinin büyük önemi vardır. Nitekim bu dosya içerisinde gelen talep için bir yönlendirme yapılması mümkündür. Bu amaçla **RegisterRoutes** isimli metoddan yaralanılır. Bir başka deyişle servisleri barındıran Web uygulaması ayağa kaldırıldığında, hangi **URI** taleplerinin hangi servislere yönlendirileceği bellidir. **URI** içerisinde yer alan operasyonel yönlendirmeler ise ilgili servis sınıfının metodlarına ait **WebGet** veya **WebInvoke** nitelikleri yardımıyla belirlenmektedir. Dolayısıyla istemcilerden gelecek olan taleplerin hangi servise yönlendirileceği sorusunun cevabı **global.asax** dosyası içerisinde verilmektedir. Hemen küçük bir not düşelim; bildiğiniz üzere **Web** taleplerinin daha etkin bir şekilde yönlendirilebilmesi yeteneği **Asp.Net 4.0** ile birlikte gelmektedir. Bu sebepten geliştirilen **WCF WebHttp** servisinin **Asp.Netuyumluluğu modunda(Asp.Net Compatibility Mode)** çalıştırılıyor olması önemlidir.

Dilerseniz konuyu daha net kavrayabilmek adına basit bir örnek ile devam edelim. Bu amaçla **Visual Studio 2010 Ultimate RC** sürümü üzerinden geliştireceğimiz **WCF REST Service Application**'a ait **Solution** içeriğini aşağıdaki gibi tasarladığımızı düşünelim.



Görüldüğü üzere **CalculationService** ve **ProductionService** isimli iki sınıfımız bulunmaktadır. Bu servis sınıfları içerisinde çok basit iki operasyon yer almakta ve her ikiside **HTTP Get** metodlarına göre cevap vermektedir. **ProductionService** sınıfı içerisinde **Product** tipinden değer döndüren bir arama operasyonu yer almaktadır. Diğer yandan **CalculationService** sınıfı içerisinde bir ürün fiyatına hangi günde bulunulduğuna göre indirim yapan ve sonuç değerini gösteren bir operasyon yer almaktadır. Yönlendirme teorimize göre servis uygulamasına gelen
talepleri **CalculationService** ve **ProductionService** tipleri üzerine dağıtmak istiyoruz. Bunun için **global.asax** dosyasında gerekli kodlamaları yapmamız gerekecektir. Ama öncesinde**CalculationService, ProductionService** ve **Product** tipi içeriklerimize bir bakalım.

Sınıf diagramımız;



Product sınıfımız;

```
namespace Lesson5
{
   public class Product
   {
      public int ProductId { get; set; }
      public string Name { get; set; }
      public decimal ListPrice { get; set; }
   }
}
```

ProductionService sınıfımız;

```
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;

namespace Lesson5
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
```

```
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class ProductionService
   {
      private static List<Product> Products = new List<Product>
      {
         new Product{ ProductId=1023, Name="Microsoft Optical Mouse",
ListPrice=34.95M},
         new Product{ ProductId=1045, Name="Logitech Optical Mouse",
ListPrice=35.45M},
         new Product{ ProductId=1029, Name="KeyMaster Wireless Keyboard",
ListPrice=55.99M},
         new Product{ ProductId=9802, Name="Obi Wan Kneobi Jedi Light Saber",
ListPrice=334.45M},
      };

      [WebGet(UriTemplate = "Products/{ProductId}")]
      public Product GetProduct(string ProductId)
      {
         Product prd=null;

         try
         {
            prd = (from p in Products
                   where p.ProductId.ToString() == ProductId
                   select p).First();
         }
         catch
         {
            // Bir Product bulunamama olasılığına karşın HTTP statü kodu 404 döndürülür
            throw new WebFaultException(System.Net.HttpStatusCode.NotFound);
         }

         return prd;
      }
   }
}
```

CalculationService sınıfımız;

```
using System;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;
```

```
namespace Lesson5
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class CalculationService
   {
      [WebGet(UriTemplate = "Discount/{ListPrice}")]
      public decimal CalculateDiscountListPrice(string ListPrice)
      {
         decimal decreasedListPrice = 0;
         decimal currentListPrice = 0;

         // Eğer string olarak gönderilen ListPrice değeri decimal tipe dönüşütürülemesse
BadRequest tipinden bir statü kodu döndürülmesi sağlanır.
         if (!decimal.TryParse(ListPrice, out currentListPrice))
            throw new WebFaultException(System.Net.HttpStatusCode.BadRequest);

         switch (DateTime.Now.DayOfWeek)
         {
            case DayOfWeek.Friday:
               decreasedListPrice = currentListPrice-(currentListPrice * 0.1M);
               break;
            case DayOfWeek.Monday:
               decreasedListPrice = currentListPrice-(currentListPrice * 0.2M);
               break;
            case DayOfWeek.Saturday:
               decreasedListPrice = currentListPrice;
               break;
            case DayOfWeek.Sunday:
               decreasedListPrice = currentListPrice;
               break;
            case DayOfWeek.Thursday:
               decreasedListPrice = currentListPrice-(currentListPrice * 0.3M);
               break;
            case DayOfWeek.Tuesday:
               decreasedListPrice = currentListPrice-(currentListPrice * 0.5M);
               break;
            case DayOfWeek.Wednesday:
               decreasedListPrice = currentListPrice-(currentListPrice * 0.2M);
               break;
         }
```

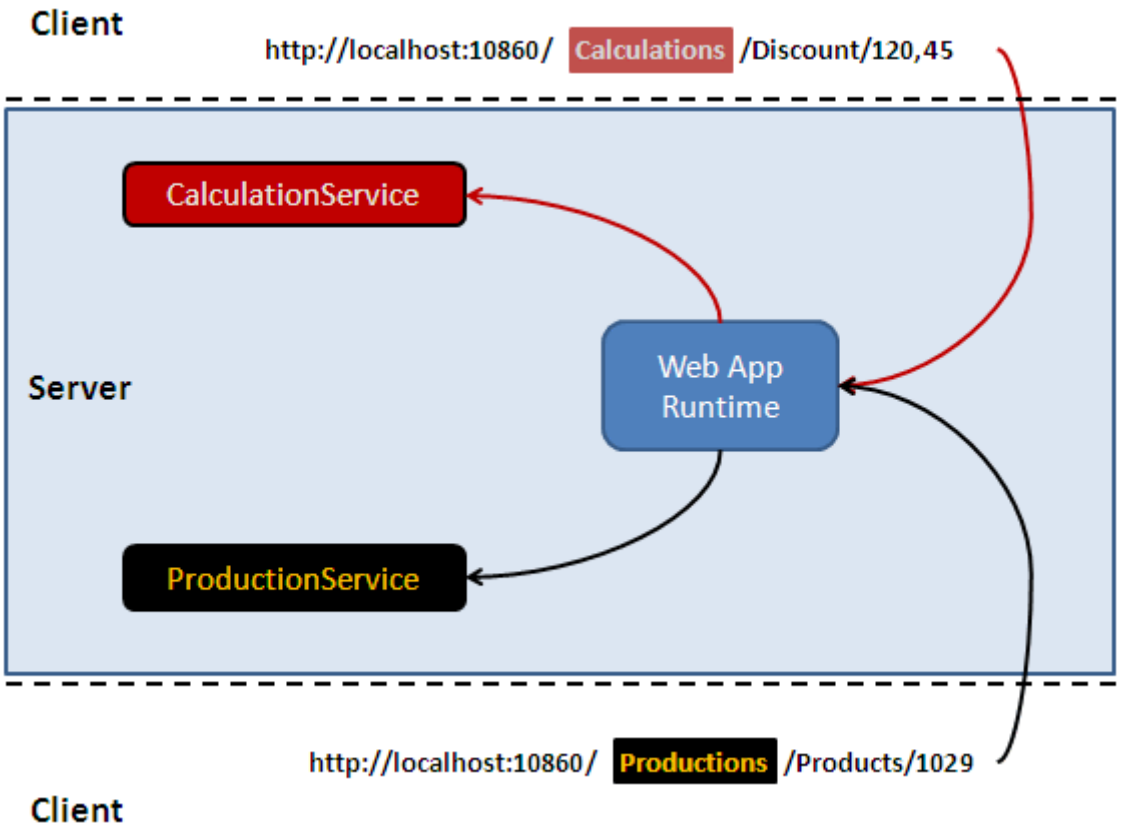
```
            return decreasedListPrice;
        }
    }
}
```

Ve gelelim projemizin en önemli kısmına. Yönlendirme ile ilişkili olarak **global.asax** dosyasında yer alan **RegisterRoutes** metodunun içeriğini aşağıdaki gibi değiştirmemiz yeterli olacaktır.

```
using System;
using System.ServiceModel.Activation;
using System.Web;
using System.Web.Routing;

namespace Lesson5
{
    public class Global
        : HttpApplication
    {
        void Application_Start(object sender, EventArgs e)
        {
            RegisterRoutes();
        }

        private void RegisterRoutes()
        {
            // WebServiceHostFactory nesnesi örneklenir.
            WebServiceHostFactory hostFactory = new WebServiceHostFactory();

            // Route tablosuna gerekli eşleştirme bilgileri eklenir.
            // Gelen talep Productions içinse ProductionService servis tipi ile ilişkilendirilir.
            // Gelen talep Calculations içinse CalculationService servis tipi ile ilişkilendirilir.
            RouteTable.Routes.Add(new ServiceRoute("Productions", hostFactory,
typeof(ProductionService)));
            RouteTable.Routes.Add(new ServiceRoute("Calculations", hostFactory,
typeof(CalculationService)));
        }
    }
}
```

Uygulama başlatıldığında **Application_Start** metoduna girilecektir. Bu metod içerisinde ise **RegisterRoutes** fonksiyonu çağırılmaktadır. **RegisterRoutes** metodu içerisinde dikkat edileceği üzere iki **ServiceRoute** tipinin örneklenmesi ve bunların **RouteTable** üzerindeki **Tables** koleksiyonuna eklenmesi sağlanmaktadır. Buna

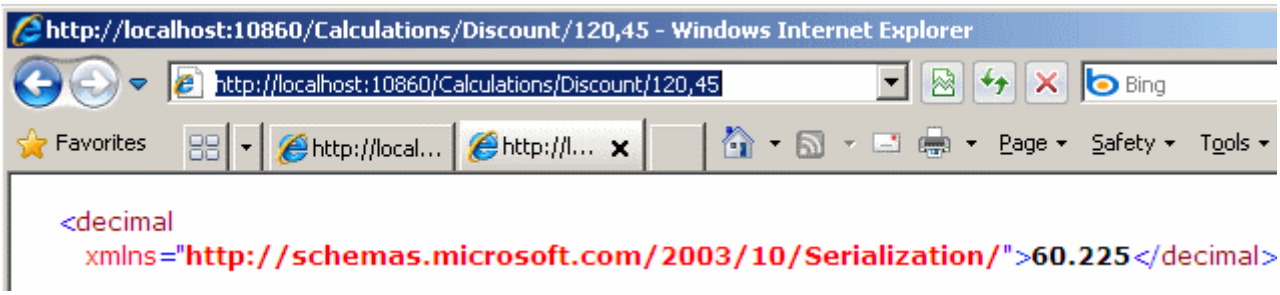
göre servise gelen talepler aşağıdaki şekilde görüldüğü üzere eşleşen tiplere yönlendirilecektir.



Hemen çalışma zamanı sonuçlarına bir bakalım.
örneğin **http://localhost:10860/Productions/Products/1029** talebinde bulunduğumuzda aşağıdaki sonuçlar ile karşılaşırız. Görüldüğü üzere talep **ProductionService'** e yönlendirilmiştir. *(Tabi var olmayan bir ProductID talebi girildiğinde HTTP Status Code 404 NotFound ile karşılaşırız. WebFaultException tipinin kullanımının anlatıldığı yazımızı hatırlayalım lütfen)*

Diğer yandan **http://localhost:10860/Calculations/Discount/120,45** şeklinde bir talepte bulunduğumuzda ise **CalculationService'** e yönlendirildiğimizi görebiliriz. *(Yine decimal*

tipe dönüştürülemeyen bir talep gönderildiğinde tahmin edileceği üzere HTTP Status Code 400 Bad Reuqest ile karşılaşırız.)



Sonuç olarak **UriTemplate'** lerin ilgili servis operasyonları ile eşleştirilmesi çok daha kolaylaştırılmıştır. öncelikle olarak gelen talebin hangi servis ile ilişkili olduğu noktasında **Route Table** devreye girmektedir. Ardından talep servise gelir. Servis ise **URI** içeriğine bakara uygun operasyonun çağırılması ile ilgilenir. Böylece geldik bir konumuzun daha sonuna. Bir sonraki yazımızda görüşünceye dek hepinize mutlu günler dilerim.

Lesson5_RC.rar (24,11 kb) **[örnek Visual Studio 2010 Ultimate Beta 2 Sürümünde geliştirilmiş ancak RC sürümü üzerinde de test edilmiştir]**

WCF WebHttp Services - Error Handling (2010-03-05T08:10:00)

wcf,wcf 4.0,webhttp services,restful,non soap,wcf webhttp services,

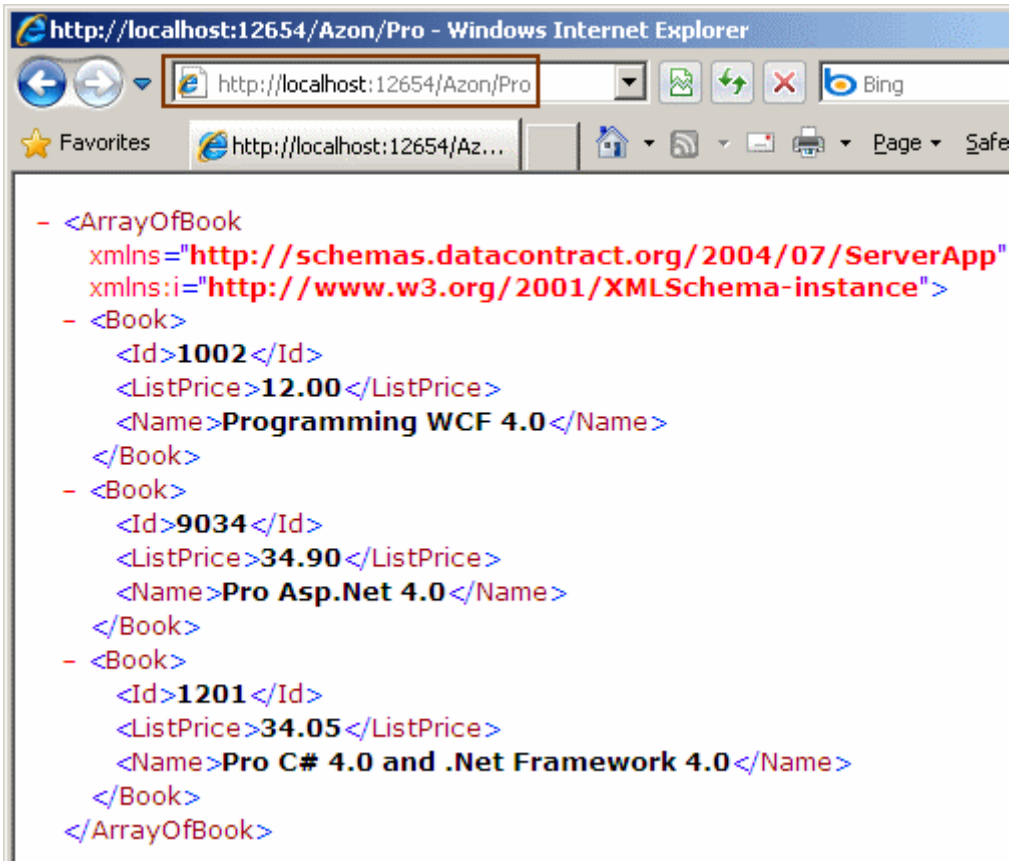
Merhaba Arkadaşlar,

Bu yazımızda **WCF Eco System'** in bir parçası olan **WebHttp Service'** lerinde **hata yönetimini(Error Management)** etkili bir şekilde nasıl ele alabileceğimizi incelemeye çalışıyor olacağız. **WCF WebHttp Service'** leri üzerinden çağırılan bir servis operasyonundan, istemci tarafına kendi insiyatifimizde hata mesajları gönderilmesini sağlayabiliriz. üstelik bu mesajları bilinen **HTTP durum kodları(HTTP Status Code)** çerçevesinde yayınlayabiliriz. Bu tip bir isteğin çeşitli sebepleri olabilir. Hemen bir gerçek hayat senaryosu üzerinden ilerleyerek bu basit konuyu pekiştirmeye çalışalım. Bu amaçla **Visual Studio 2010 Ultimate RC** sürümü üzerinde geliştirdiğimiz ve aşağıdaki kod içeriğine sahip bir **WCF REST Service Application** projemiz olduğunu düşünelim.

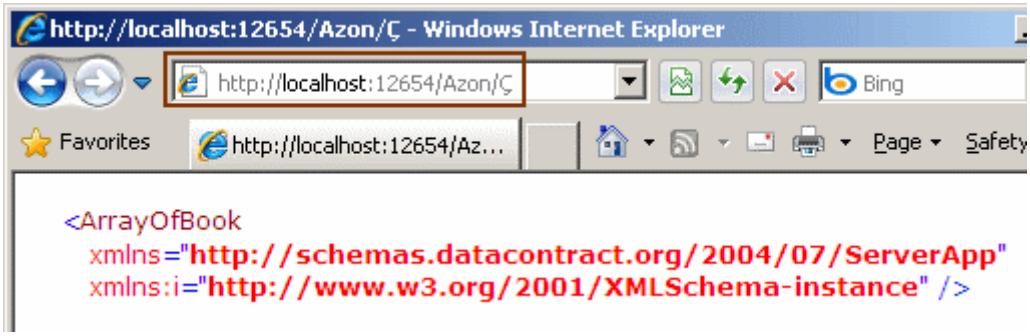
```
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;

namespace ServerApp
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class AzonBookService
   {
      static List<Book> books = new List<Book>
         {
            new Book{ Id=1002, Name="Programming WCF 4.0",ListPrice=12.00M},
            new Book{Id=9034,Name="Pro Asp.Net 4.0",ListPrice=34.90M},
            new Book{Id=4560,Name="Algebra",ListPrice=122.39M},
            new Book{Id=1200,Name="C# 3.5 Cookbook",ListPrice=14.45M},
            new Book{Id=1201,Name="Pro C# 4.0 and .Net Framework
4.0",ListPrice=34.05M},
            new Book{Id=1201,Name="Beginning Ado.Net Entity
Framework",ListPrice=14.55M}
         };

      [WebGet(UriTemplate = "/{firstLetter}")]
      public List<Book> GetBooks(string firstLetter)
      {
         return (from book in books
               where book.Name.StartsWith(firstLetter)
               select book).ToList();
      }
   }

   public class Book
   {
      public int Id { get; set; }
      public string Name { get; set; }
      public decimal ListPrice { get; set; }
   }
}
```

AzonBookService çok basit ve tek bir operasyona sahiptir. **GetBooks** isimli operasyon **HTTP Get** metoduna göre hizmet vermekte olup baş

harfleri **firstLetter** parametresin ile gelen değer ile başlayan bir**List<Book>** koleksiyonunu geriye döndürmektedir. Bu servisi bir tarayıcı uygulama üzerinden test ettiğimizde ve örneğin **Pro** kelimesi ile başlayan kitapları elde etmek istediğimizde aşağıdaki ekran görütüsündekine benzer sonuçlar ile karşılaşırız.

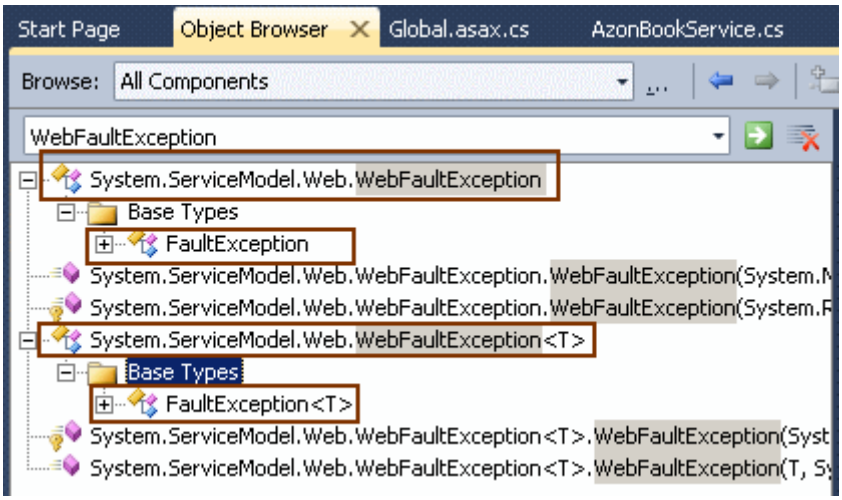


Bu noktaya kadar zaten herhangibir sorun bulunmamaktadır. Ancak örneğin **ç** harfi ile başlayan kitapların listesini elde etmek istediğimizde, aşağıdaki ekran görüntüsü ile karşılaşırız.



Şimdi bu noktada geliştirici olarak bir karar verebiliriz. İstemci tarafına bu şekilde boş bir **XML** içeriği döndürmemiz halinde, istemci tarafının **ç** harfi ile başlayan kitapların olmaması durumunu ele alması gerekmektedir. Yani istemci tarafına ek bir iş yükü getirmiş olabiliriz. Nitekim istemcinin dönen listenin eleman sayısına bakara bir karar vermesi gerekmektedir. Diğer yandan istemciyi bilinçli bir şekilde uyarabiliriz de. Nasıl mı? Servis

tarafındaki çalışma zamanında üreteceğimiz ve o anki vakaya uygun bir **istisna(Exception)** ile. Tabiki bu istisna mesajı servis bazlı bir yapı üzerinden ele alınacağı için istemci tarafına gönderilecek paketin içerisine gömülecektir. **WCF WebHttp Service'** lerinde bu tip istisnaları ele almak için **WCF** tarafından bildiğimiz **FaultException** tipinden türeyen **WebFaultException** sınıfından yararlanılır.



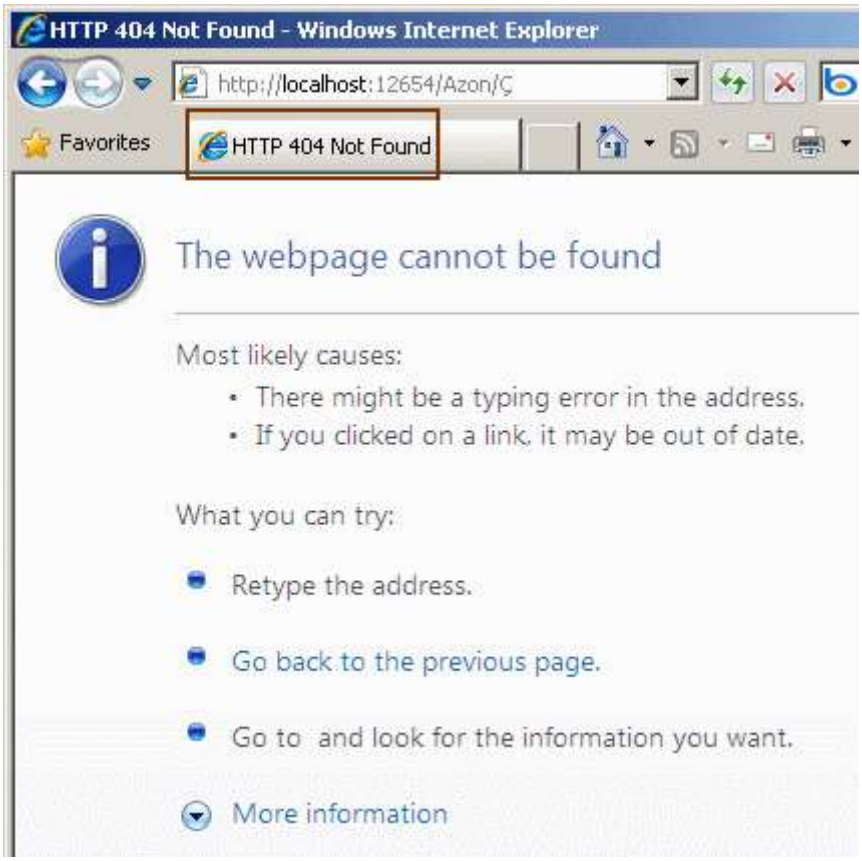
System.ServiceModel.Web isim alanı altında yer alan **WebFaultException** tipinin normal ve generic olan versiyonları bulunmaktadır. İlgili tipleri kullanımları son derece basittir. Yukarıdaki senaryomuza göre istemciye aradığı kriterlere uygun bir içerik bulunamadığını bildirmek amacıyla, **GetBooks** isimli servis operasyonunu aşağıdaki gibi değiştirebiliriz.

```
[WebGet(UriTemplate = "/{firstLetter}")]
      public List<Book> GetBooks(string firstLetter)
      {
         var result=(from book in books
               where book.Name.StartsWith(firstLetter)
               select book).ToList();

         if (result.Count == 0)
            throw new WebFaultException<string>("Talep edilen kelime ile başlayan
kitaplar sistemde mevcut değiller.", System.Net.HttpStatusCode.NotFound);

         return result;
      }
```

Burada dikkat edileceği üzere sonuç listesinin eleman sayısı kontrol edilmiş ve eğer **0** ise **WebFaultException** tipinden bir istisna mesajı fırlatılması sağlanmıştır. **WebFaultException**tipinin örneklenmesi sırasında dikkat edilmesi gereken hususlardan biriside **HttpStatusCode.NotFound Enum sabiti** değerinin verilmesidir. Bu şekilde istemci tarafına hangi **HTTP durum kodunun(Status Code)** gönderileceği belirlenmektedir. Tahmin edeceğiniz üzere pek çok **HTTP Status Code** değeri bulunmaktadır.

Aslında tam liste içeriği şudur Accepted, Ambiguous, BadGateway, BadRequest, Conflict, Continue, Created, Expectation Failed, Forbidden, Found, GatewayTimeout, Gone, HttpVersionNotSupported, InternalServerError, LengthRequired, MethodNotAllowed, Moved, MovedPermanently, MultipleChoices, NoContent, NonAuthoritativeInformation, NotAcceptable, NotFound, NotImplemented, NotModified, OK, PartialContent, PaymentRequired, PreconditionFailed, ProxyAuthenticationRequired, Redirect, RedirectKeepVerb, RedirectMethod, RequestedRangeNotSatisfiable, RequestEntityTooLarge, RequestTimeout, RequestUriTooLong, ResetContent, SeeOther, ServiceUnavailable, SwitchingProtocols, TemporaryRedirect, Unauthorized, UnsupportedMediaType, Unused, UseProxy

Operasyonumuzu bu yeni haliyle denediğimizde ise tarayıcı uygulama üzerinde aşağıdaki şekilde görülen çıktı ile karşılaşırız.



ki bu son derece doğaldır.

Nitekim zaten **HTTP 404** mesajının istemci tarafına gönderilmesi sağlanmaktadır. **WebFaultException<T>** sınıfının örneklenmesi sırasında **T** parametresi de önemlidir. örneğimizde basit bir string tipi kullanılmıştır. Peki ya kullanıcı tanımlı bir tip buraya dahil edilebilir mi? Gelin hem bu durumu hemde istemci uygulamanın geliştiriciler tarafından yazılması halinde söz konusu hata mesajlarının nasıl ele alındığı örneğimizi güncelleyerek irdelemeye çalışalım. Bu amaçla servisimizi aşağıdaki hale getirelim.

...Kodun diğer kısmı

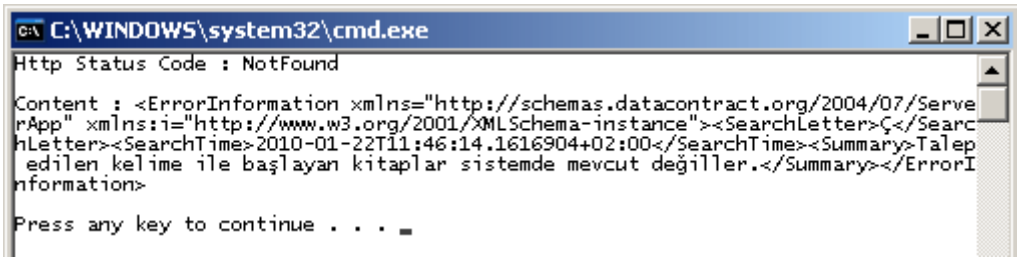
```
        [WebGet(UriTemplate = "/{firstLetter}")]
        public List<Book> GetBooks(string firstLetter)
        {
            var result=(from book in books
                  where book.Name.StartsWith(firstLetter)
                  select book).ToList();

            if (result.Count == 0)
               throw new WebFaultException<ErrorInformation>(
                  new ErrorInformation { SearchLetter = firstLetter, SearchTime =
DateTime.Now, Summary = "Talep edilen kelime ile başlayan kitaplar sistemde
mevcut değiller." }
                  , System.Net.HttpStatusCode.NotFound);

            return result;
        }
    }

    public class ErrorInformation
    {
        public string SearchLetter { get; set; }
        public DateTime SearchTime { get; set; }
        public string Summary { get; set; }
    }
```

Bu kod parçasında **WebHttpException<ErrorInformation>** tipinden bir nesne örneklenmiştir. Buna göre istemci tarafına bu içeriğin **XML** veya **JSON** formatında gönderilmesi mümkündür. Gelelim istemci tarafının kodlarına. Nitekim bir şekilde servis tarafından gelen içeriği kontrol atlına almamız gerekmekte.

```
using System;
using Microsoft.Http;

namespace ClientApp
{
    class Program
    {
        static void Main(string[] args)
        {
            using (HttpClient client = new HttpClient("http://localhost:12654/Azon/"))
            {
                HttpResponseMessage response=client.Get("ç");
```

```
            Console.WriteLine("Http Status Code :
{0}\n",response.StatusCode.ToString());
            Console.WriteLine("Content : {0}\n",response.Content.ReadAsString());

        }
    }
  }
}
```

ç harfi ile başlayan kitapların elde edilmesi için bir talep oluşturulmakta ve bu talep **Get** metodu yardımıyla servis tarafına gönderilmektedir. **Get** metodunun çıktısı **HttpResponseMessage**tipindendir. Elde edilen **HttpResponseMessage** nesne örneğinin **StatusCode** özelliği yardımıyla servis tarafında üretilen **HttpStatusCode Enum sabitinin** değeri elde edilir. Diğer yandan eğer istemci tarafından bir hata oluşuyorsa **WebFaultException** nesnesinin oluşturulması sırasında kullanılan **ErrorInformation** nesne örneğinin serileştirilmiş hali **Content** özelliği üzerinden elde edilebilmektedir. Buna göre çalışma zamanı çıktısı aşağıdaki gibi olacaktır.



Görüldüğü üzere servis operasyonu içerisinde üretilen **ErrorInformation** bilgisi istemci tarafına **XML** formatlı olacak şekilde aktarılmıştır. Sonuç olarak olası veya beklenen hatalar karşısında istemci tarafına uygun bir bildirim yapılması **HTTP** durum kodları bazında mümkündür. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Lesson4_RC.rar (174,57 kb) **[örnek Visual Studio 2010 Ultimate Beta 2 Sürümünde geliştirilmiş ancak RC sürümü üzerinde de test edilmiştir]**

Screencast - AJAX Enabled WCF Services (2010-03-04T10:45:00)

wcf,wcf 4.0,asp.net,asp.net 4.0,screencast,

Merhaba Arkadaşlar,

Görsel derslerimize kaldığımız yerden devam ediyoruz. Bu sefer elimizdeki materyaller bir **WCF Service**, **Asp.Net Web Uygulaması** ve **AJAX**. Bunları bir arada düşündüğümüzde ise karşımıza **AJAX Enabled WCF Service** kavramı çıkıyor. Bildiğiniz üzere **Asp.Net** uygulamalarında**AJAX** imkanları kullanılabilmekte ve bu sayede kısmi olarak post işlemleri gerçekleştirilebilmekte. çok basit anlamda bir sayfanın tamamını sunucuya göndermeden sadece istenilen parçaların gönderilmesi ve sonuçlarının ele alınabilmesi mümkündür. Tabi sonuçların istemci tarafında ele alınması gerekmekte*(örneğin **Javascript** ile)*. **AJAX Destekli WCF Servislerinde** ise herhangibir servis operasyonunun çağrısı sırasında, servisi çağıran web sayfasının tamamının sunucuya gönderilmemesi imkanı kazanılmaktadır. **AJAX Destekli WCF Servislerinin** kullanıldığı pek çok senaryo söz konusudur. En basit haliyle otomatik metin tamamlama kabiliyetine sahip kontroller için bu teknikten yararlanılabilir. Bizde görsel dersimizde konu olarak **bebek adlarını(Baby Names)** ele almaya çalışacağız. Upss!!! Bebek Adları mı? İzleyelim ve görelim.

Video Boyutu : 24 Mb

Süre : 14:06

Download Etmek veya İzlemek için

WebApplication2.rar (22,09 kb)

Screencast - Workflow Foundation 4.0 Switch Aktivite Bileşeni [RC] (2010-03-01T09:35:00)

screencast,wf 4.0,wf,

Merhaba Arkadaşlar,

Bildiğiniz üzere bir süre önce **.Net Framework 4.0 RC** sürümü yayınlandı. Bu da **RTM** ve **Release** sürüme çok yaklaştığımızı göstermekte. **RC(Release Candidate)** sürümünün önceki **Beta** sürümlerine göre daha tutarlı olduğunu söyleyebiliriz. Ancak yine de bu görsel dersimizde anlatacaklarımız ile ilişkili olarak **Bug Fix**' ler veya farklı güncelleştirmeler söz konusu olacaktır. Peki ya biz bu görsel dersimizde neyi ele almaktayız? 😉

Workflow Foundation 4.0 içerisinde yer alan **Built-In Activity** bileşenleri arasında bir programlama dilinin temel özelliklerinin de çoğu yer almakta aslında. Söz gelimi **ForEach<T>, While, If, DoWhile, Assign, InvokeMethod, WriteLine** ve benzerleri gibi aktiviteler göz önüne alındığında, değişken ataması, metod çağırılması, döngü kurulması veya koşullu kontrollerin yapılması son derece kolay. Bu anlamda, **WF 4.0** içerisinde yer alan aktivite bileşenlerinden bir tanesi de **Switch**. **C#** dilinden aşina olduğumuz bu koşul ifadesinin mantığını, **Workflow** tiplerinde de kullanabilmekteyiz. NedirTv? katkılarıyla gerçekleştirdiğimiz bu görsel dersimizde **Switch** aktivite bileşeninin özel **.Net** tipleri için nasıl değerlendirilebileceğini incelemeye çalışmaktayız. Hepinize keyifli seyirler dilerim.

[Görsel derste yer alan örneğimiz Visual Studio 2010 Ultimate Beta 2 sürümünde geliştirilmiş ve test edilmiştir.]

Dosya Boyutu : 24.5 Mb

Süre : 13:43

İzlemek veya Download etmek için

UsingSwitch.rar (57,95 kb)

Yazılımcının Kendi Kendini Eğitmesi (2010-03-01T08:00:00)

 Merhaba Arkadaşlar,

İster üniversite öğrencisi olun ister yeni mezun, yazılım sektörünün henüz başındaysanız eğer, kendinizi yandaki şekilde görülen yolun başında bekleyen birisine benzetebilirsiniz. Ne yolun uzunluğu bellidir ne de gidebileceğiniz yönler. Açıkçası buna baştan karar verebilmek çok zordur. Yaşantınızı ektileyen pek çok neden bu belirsizliğe sebebiyet verebilir. Hatta yazılım alanında hangi yöne doğru ilerleyeceğinize karar verdiğinizde aradan bir kaç yıl geçmiş olabilir. Tüm bu yaşam döngüsü içerisinde unutlmaması gereken bir noktada sizin öğrenme süreçlerinizdir. Pek çok yazılım sevdalısı arkadaşımız çabucak panikleyip bir yerlerden eğitim almaya çalışabilir. Bunun gerekli olduğu veya olmadığı durumlar vardır. Açıkçası Mühendislik nosynonunu üniversite öğrenimi sırasında kazanmış olanlar avantajlıdır. Ancak onlar içerisinde bile eğitim almak için parasını harcamaktan çekinmeyen kişiler olabilir. Bu yazımızda yine teknik olmayan bir konuya değiniyor olacağım. Sizin bir yazılımcı olarak kendi kendinizi nasıl eğitebileceğinizi anlatmaya çalışacağım. Uzun bir süre eğitimcilik yaptığım için yazıda yer alan ip uçlarını faydalı bulacağınızı düşünüyorum. Tavsiyelerime kulak verin.

Yaşananlardan : *90 lı yılların başlarında üniversite öğrencisiydim ve bilgisayar programcılığına olan merakım her geçen gün katlanarak artıyordu. Açıkçası 64 Kb sınırı altında yaşayan ve oyun yazan programcılara fazlasıyla imreniyordum. O yıllarda Türkçe yazılım kitaplarını bulmak zordu. Şimdi çok kitabımız var ancak aradan kaliteli olanları seçmek istediğinizde aslında çok az kitabımız var. Yabancı kitapları Türkiye' ye getirtmekte bir meseleydi. özellikle fiyatları düşünüldüğünde. Yurt dışı kanalı ile tedarik ettiğim ilk bilgisayar kitabım Delphi 2.0 Unleashed idi. (Eski bir Delphi programıcısı olduğumu ama Anders Hejslberg' in izinden gittiğimi belirtmek isterim) 1400 sayfaya yaklaşan bu kitabı Taksim' de bulunan Elit kitabevinden tedarik etmiştim. O günü çok net hatırlıyorum. Ahşap tabanda yürürken raflarda gördüğüm sayısız kitabın çoğunu satın almak istemiştim. Ama elimde o kadar çok param yoktu. Yine de isabetli bir atış yapmaya çalışmış ve gerçekten ilgilendiğim konuya ait bir tanesini tedarik edebilmiştim. Bu bulunmaz bir nimet gibiydi.*

Kitaplar

 Yazılımcıların en büyük dertlerinden birisi de çok hızlı gelişen ve sürekli olarak bizi ötelemeye zorlayan teknoloji. Programlama dilleri, Framework' ler, yazılım metodolojileri sürekli olarak değişmekte veya yenilenmekte. Bunun için uzağa gitmemize gerek yok. C#' ın bir kaç sene içerisinde 4ncü versiyonuna nasıl geldiğini gayet iyi biliyoruz. Dolayısıyla bir yazılımcının çok sık kitap okuması ve takip etmesi gerekiyor. Bu anlamda Amazon gibi Amerika veya İngiltere üzerinden kitap getirtebilmemizi sağlayan sitelerin sunduğu ingilizce kaynaklar hem profesyonellikleri hem de anlatımları yönünden son derece kuvvetli ve yeterli. Size tavsiyem bir kaç arkadaş birleşip birden fazla kitap getirterek taşıma ücretini mümkün olduğunca hafifletmeniz 🙂

Yaşananlardan : *üniversitedeki eğitimimi Türkçe almıştım. Mezun olduğumda 1 senelik ingilizce hazırlık eklenmişti. Aslında tüm eğitim hayatım boyunca devlet okullarında öğrenci olmuştum. Ne acıdır ki, orta okul yıllarında hoca yetersizliğinden veya anlaşmazlıklardan dolayı ingilizce derslerimizin yarısından çoğu boş geçmişti. Hal böyle olunca insan çok doğal olarak yabancı dil sorununu tek başına halletmek zorunda kalıyor. Az önce bahsettiğim Delphi 2.0 kitabı bir kaç ay içerisinde kara kalem çalışmasına dönmüştü. çünkü bilmediğim her kelimenin yanına kurşun kalem ile ne anlama geldiğini yazmıştım. Bu sonraki kitapta, ondan sonrakinde ve sonrakinde de devam etti. Ama bir süre sonra altı çizili kelimeler azalmaya başladı. Yani biraz azmetmem gerekmişti. Bu noktada pek çok rakibinizden geride kaldığınızı düşünebilirsiniz. Ama inanın bir noktadan sonra onları yakalayabilir ve hatta kazandığınız çalışma disiplini ile geçebilirsiniz. çünkü sıkılmadan yapacağınız ve sözlükten bulduğunuz kelimeleri yazarak harcayacağınız zaman içerisinde standart bir disipline sahip olursunuz. Bu disiplinin içeriği basittir. Sıkılmamak, konsantre olmak, hedefe ulaşmak. ödülü ise basittir; okuduğunuzu anlamak.* 😀

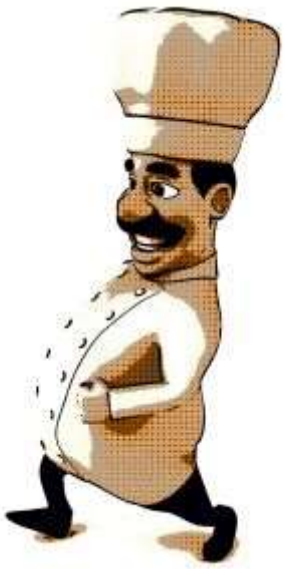
Tabi burada mutlaka dikkat edilmesi gereken bir nokta var. çok fazla kitap almaktan ziyade isabetli kitapları seçmek daha önemlidir. Yoksa yandaki resimde görüldüğü üzere kitap sayfaları arasında rahatlıkla kaybolabilir ve işi daha da karmaşık bir hale sokabilirsiniz. Bu noktada belkide daha tecrübeli kişilere danışmak uygun bir davranış olarak görülebilir. Ne de olsa artık günümüzde pek çok yetkin kişiye mailler aracılığıyla ulaşabilmekteyiz.

Eğitim

Kitaplar her zaman için faydalı olan kaynaklardandır ama tek başlarına yeterli olmayabilirler. çoğu zaman konuyu daha uzman birisinden dinlemek isteyebilirsiniz. Dinlediğiniz kişinin bu anlamda tecrübeli olması geçmiş proje deneyimlerini konu anlatımları üzerine serpiştirmesi de çok önemlidir. Hatta konu ile ilişkili olarak lab çalışması yapılabilmesi size çok fazla değer katacaktır.

Yaşananlardan : *Uzun süre bir eğitim firmasında yazılım eğitmeni olarak görev aldım. Eğitimler sırasında özellikle kitaplardan anlatılan konularda öğrencilerimin en çok yakındıkları veya sevmedikleri konu lab çalışmalarının olmasıydı. Aslında Microsoft tarafından sunulan kitaplarda yer alan lab çalışmalarının çoğu, konu ile ilişkili olarak baştan sona bir projenin tamamlanmasını, öğrenilenlerin pekiştirilmesini, hangi malzemeler ile neler yapılabileceğinin görülmesini sağlamaktaydı. üstelik bu sağlanırken gerçek hayat pratikleri de kazanılmaktaydı.Bu nedenle aldığınız eğitimler içerisinde lab çalışmaları var ise bunları görmezlikten gelmeyin.*

Açıkçası bu tip eğitmen bulduğunuzda yakasını eğitim süresi boyunca bırakmamanızda ve söylediklerini yapmanızda fayda vardır. Ne yazık ki günümüzde pek çok eğitim firması işlerini su istimal etmektedir. Açıkçası bu işi hafife alanlardan tutunda, tamamen ticari çıkarlara dönüştüren, rekabet etmek isterken eğitimin kalitesini düşüren firmalar söz konusudur. Ancak tüm bu firmalar her zaman için kendilerinin en iyi eğitmenlere sahip olduklarını ve son teknolojileri aktardıklarını ifade ederler. Aslında gerçek sadece şudur; siz eğitim kurumuna değil eğitmenine para vermelisiniz. Gerçekten o eğitim kurumunun almak istediğiniz eğitimle ilişkili olan konuda son derece saygın, size fayda sağlayacak, deneyimli bir eğitmeni varsa bu durumda eğitimden gerekli yararı sağlayabilirsiniz.



Uzmanlaşmak

Yazılımcının kendisini eğitmesi sırasında yaşadığı önemli güçlüklerden biriside uzmanlaşmadır. Yazılım çok geniş bir dünyadır hatta bir evrendir. Bu sebepten yazılımcıların galakside kendilerini kaybetmeleri son derece doğaldır. Dolayısıyla yazılım içerisinde de belirli konularda uzmanlaşmak gerekebilir. Bu artık bir mecburiyet halini almıştır. Söz gelimi .Net Framework cephesinde servis tabanlı çözümlerde, Web veya Windows tabanlı uygulamalarda ya da Mobil cihazlar üzerinde uzmanlaşılabilir. Aslında buradaki uzmanlaşma fikri bir aşçının uzmanlığına benzetilebilir. Sadece Fransız veya çin mutfağında ortalığı kasıp kavuran bir aşçı kendi istekleri doğrultusunda seçtikleri alanlarda uzmanlaşmıştır denilebilir. Ancak burada kaçınılmaz bir gerçek vardır. Birden fazla mutfakta uzmanlaşmış olan bir açşı diğerlerine göre çok daha fazla kazanacaktır. Aynı prensip yazılımcılar için de geçerlidir.

Yaşananlardan : *çok yakın zamanda başlayan ama bir süre sonra iptal edilmek zorunda kalınan bir projede, .Net tabanlı bir iş akışı geliştirme sisteminin yazılması istenmekteydi. Projenin en ilginç noktası sistemi kullanacak firma yöneticilerinin onay mekanizmasına Blackberry cihazları ile dahil olmak istemeleriydi. Bu noktada Blackberry cihaz üzerinde otomatik bildirimde bulunacak ve onaylama, red etme, geri gönderme gibi fonksiyonellikleri sunacak bir yazılımın olması gerekmekteydi. Dolayısıyla bizim Blackberry cihazlarında geliştirme konusunda bir şeyler öğrenmemiz bekleniyordu. Zaman azdı ve projede analiz ile çözüm tasarım süreçleri bitmek üzereydi. Güvendiğim tek şey geçmiş yazılım tecrübem, programlama disiplinlerine olan hakimiyetim, çözüm tasarımı yeteneğim, analizi iyi kavramış olmam ve öğrenme isteğimdi. Sizlerde zaman içerisinde bu gibi durumlara düşebilirsiniz. Burada kendi kendinizi eğitme ve yetiştirme tekniklerinizi ön plana çıkıp öğrenme sürecinizi belirleyecektir.*

Güvenilir Bilgi

Yazılımcının kendi kendisini geliştirmesinde internet tabanlı kaynaklarında yeri büyüktür. Bu alanda sayısız Community ve Blog mevcuttur. Ancak burada güvenilir olan bilgiye ulaşmak gerekmektedir. Bu sebepten dolayı yazılımcılar olarak sizlerin çok daha seçici davranması gerekmektedir. Ne yazık ki en büyük yanlış, yayınlanan teknik içeriklerin denetlenmemesidir. Eğer düzenli olarak yerli siteleri takip ediyorsanız teknik içerikli makale veya günlük girdilerine yabancı sitelerdekine oranla çok daha az sayıda yorum yapıldığını ve geri bildirimde bulunulduğunu görebilirsiniz. Oysaki pek çok güvenilir yabancı kaynaklı sitede, yayınlanan içerikten daha uzun yorumlar olduğunu ve geri dönüşler bulunduğunu fark edebilirsiniz. Bu da yazının kaliteli olması halinde sizin için çok daha doyurucu bir içerik oluşması anlamına gelmektedir. Nitekim o konu üzerinde farklı kişilerin görüşlerini de almış olursunuz.

Peki güvenilir sitelere nasıl ulaşabilirsiniz? Dikkat edebileceğiniz ilk nokta yazarın künyesi olacaktır. Microsoft içerisinden gelen, RD, MVP gibi ünvanlara sahip olan yabancı yazarların çoğusizlere tatmin edici içerikler sunacaktır. özellikle güvenilirliğinden emin olduğunuz kişilerin varsa bloklarını veya feed' lerini yardımıcı bir program aracılığıyla düzenli olarak takip edebilirsiniz. örneğin benim kullanmakta olduğum FeedReader programını bu anlamda değerlendirebilirsiniz.

Gördüğünüz üzere değerli meslektaşlarım yazılımcıların her daim kendilerini öğrenci olarak görmelerinde yarar vardır. Gerçekten yazılımı bir yaşam biçimi olarak düşündüğünüzde aslında uzun yıllar boyunca sürecek bir Go oyununa dahil olduğunuzu da görebilirsiniz. Go oyununda öğrenci olarak 30 kyu' dan başlayıp ilerlersiniz. Ancak DAN mertebesi alana kadar öğrencisinizdir ve bu seviyeye ulaşmak hayatınızın bir kaç yılına mal olabilir. Dan aldıktan sonra ise ulaşabileceğiniz en yüksek nokta 9ncu Dan' dır. Tabi oyunun sizin yazılım hayatınıza benzerliği açısından önemli olan katkısı şudur; hep kendinizden daha iyi birisi ile oynayarak ve dolayısıyla kendinizi

zorlayarak kademe atlayabilirsiniz. Dan'lara geçtikçe siz artık öğretmen olmaya başlarsınız. Bu da aslında çevrenizdeki yazılımcılara yardım edip onların size danışması anlamında düşünülebilir.

Yazının bu son noktasında halen daha nereden başlayacağınızı bilemeyebilir ve kafanızda çelişkiler yaşayabilirsiniz. öyleyse şunu deneyin. Yazılım alanında gelmek istediğiniz noktaya odaklanın. Odaklandığınız nokta şu andan 3 yıl sonrası olabilir. Görebildiğiniz kadar ilerisi.*(Görebildiğiniz kadar çünkü başta da belirttiğimiz üzere teknolojinin hızı neredeyse ışık hızına yakın)* Şimdi geriye doğru gelin. Nelere ihtiyacınız olduğunu not etmeye başlayın. Not ettikleriniz bahsettiğiniz hedefe ulaşmanız için gerekenlerdir. Şimdi bu notlarda yazılanları öğrenmek için neler yapmanız gerektiğini düşünün. Gerekli kitaplar, ilgili siteler, blog adresleri, kişiler, kurumlar...Şu anda en azından nelere ihtiyacınız olduğunu tespit ettiniz. Şimdi aradan uygun olanı seçip sıralı olarak devam edebilir ya da bir kaçına birden başlayarak paralel yürüyebilirsiniz. Kolay gelsin. Zorlu bir maratona başladınız...

1652 Sayfalık İçerik - Tüm Blog Girdilerim (2010-02-24T20:30:00)

Merhaba Arkadaşlar,

Sabırsızlanan arkadaşlarım yazının en altındaki download linklerine sıçrayabilirler.

Bir yazılımcı sıkıldığında ne yapar? İşte bu günün sorusu...Büyük ihtimalle işlerinden bunalmış veya bazı şeyleri kafaya taktığından içinden çıkılması güç bir psikoloji altına girmiş bir yazılımcının yapacağı şeyler aslında yine teknoloji merkezli olacaktır. Peki ya siz olsanız ne yaparsınız?

- Belki internet evreninde kafanızı dağıtacak sitelerde gezinirsiniz. Güncel haberler, spor, magazine bültenleri, sosyal ağlar veya chat.
- Arkadaşlarınız ile oyun oynarsınız ve büyük ihtimalle bu masa üstünde oynayacağınız Satranç ya da Go olmaz. Olsalar bile bilgisayar başında oynanabilenleri olur. Hatta bunun için sosyal ağınıza bir uygulama bile ekleyebilirsiniz. Kuvvetli ihtimalleden birisi PSP gibi oyun konsollarını değerlendireceğinizidir. Tabi şirketinizin içerisinde Turkcell' de olduğu gibi Playstation odası varsa harika.

- Kitap okuyabilirsiniz ki bu sıkıntınızı daha da arttırabilir çünkü elinizin altındaki kitapların belki de tamamı yazılım ile ilgilidir.
- Sinemaya gidebilirsiniz ama büyük ihtimalle teknolojinin sınırlarını zorlayan bir filmi seçersiniz. örneğin Avatar...
- Bunların haricinde belki MP3 çalarınız ile yürüyeşe çıkabilirsiniz.
- Ya da kız arkadaşınız veya eşinizle felekten bir gün/gece çalabilirsiniz.

Peki aranızda sıkılınca kod yazanınız var mı?

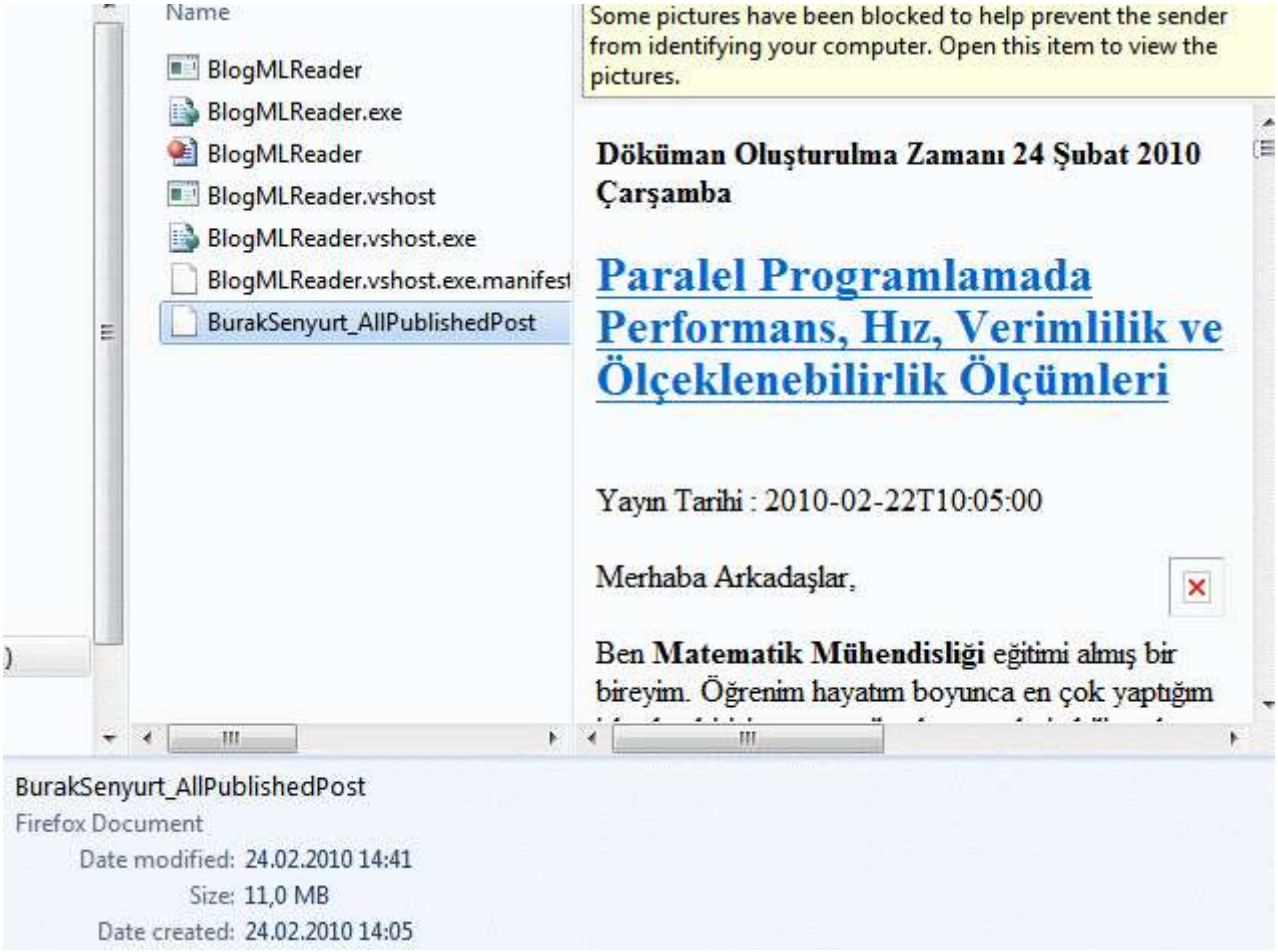
Bu gün öylesine çok sıkıldım ki...İçimden pek bir şey yapmak gelmedi. Standart yazılımcı buhranı diyerek ekranıma bakarken bari kod yazayım dedim. Yazacağım **kod benim için eğlenceli olsun, sonuçları ise herkes için faydalı olsun** istedim. İlk olarak **BlogeEngine** tabanlı sitemin yönetim paneline girdim ve tüm içeriği dışarıya aktaradım. Artık elimde bloğumda o ana kadar yayınlanmış/yayınlanmamış tüm girdilerin ve bilgilerinin bulunduğu bir **XML** dosyası vardı. Hımmm... öyleyse şöyle bir çalışma içerisine girebilirdim; **Tüm XML içeriğini alıp anlamlı bir şekilde tek bir HTML içerisinde birleştirmek.** öyle ahım şahım arayüzü olan bir programa da ihtiyacım yoktu. Basit bir **Console**uygulaması bile işimi görürdü. çala klavye yazmaya başladım. Tabi yazarken bol bol **debug** işlemini uygulamam ve içerikte neyin nerede olduğunu anlamam gerekti. Sonunda aşağıdaki kod ortaya çıktı.

```
using System;
using System.IO;
using System.Linq;
using System.Text;
using System.Xml.Linq;

namespace BlogMLReader
{
   class Program
   {
      static void Main(string[] args)
      {
          // önce indirdiğim BlogMl.xml dosyasını bir yükliyeyim
         XElement root = XElement.Load( "..\\..\\BlogML.xml");
         // Sonra içerisinden istediklerimi alayım.
         // Nedir bu istediklerim? Yazının başlığı, url adresi, yayın tarihi/oluşturulma tarihi, içeriği(en önemlisi de bu zaten), tag değerleri
         var allPosts = from post in root.Elements().Last().Elements()
                        where post.LastAttribute.Value == "True" // Hatta içeriği çekerkende yazdığım ama henüz yayınlamadığım yazıları da alması
                        select new // En güzeli anonymous tip kullanmak. çünkü her şeyi ele almak istemiyorum. Sadece istediklerimi bir tip içerisinde tutmak niyetindeyim.
                        {
                           Title=post.Elements().First().Value,
```

```
                    Url=post.Attribute("post-url").Value,
                    DateCreated = post.Attribute("date-created").Value,
                    Content=post.Elements().ElementAt(1).Value,
                    Tags=from tag in post.Elements().Last().Elements() select tag //
Birden fazla tag olduğu için...
                };

        // İçeriği string olarak ele alacaksam performanslı bir tip lazım. StringBuilder
mesela
        StringBuilder builder = new StringBuilder();

        builder.Append(String.Format("<b>Döküman Oluşturulma Zamanı {0}</b></br>",
DateTime.Now.ToLongDateString()));
        // Birde sayaç belirleyim. En azından ekrana kaç tanesinin yapıldığını yazdırabilirim
        int counter = 1;
        // LINQ sorgusu sonucu elde edilen tüm satırlarda dolaşayım
        foreach (var post in allPosts)
        {
           Console.Write("{0} ",counter.ToString());
           counter++;

           // Şurada bir boşluk bırakayım
           builder.AppendLine("</br>");
           // Yazının başlığını link şeklinde basmalıyım. Basit HTML tag' leri işimi görür
           builder.AppendLine(String.Format("<b><h2><a
href=\"http://www.buraksenyurt.com{0}\">{1}</a></h2></b>",post.Url,post.Title));
           // Bir boşluk daha bırakayım
           builder.AppendLine("</br>");
           builder.AppendLine(String.Format("Yayın Tarihi : {0}<br/>",
post.DateCreated));
           string newContent=String.Empty;
           // Post' taki image' lara ait resimlerin çekilmesi sırasında image.axd isimli bir
adrese yönlendirilme söz konusu. Bu nedenle başlarına duruma göre blog adresini
ekleyeyim.
           // Aslında bu image' ları request oluşturup yerel makineye indirip ekleyebilirim
ama bunu yapmaya üşeniyorum.
           if(!post.Content.Contains("src=\"image"))
              newContent = (post.Content.Replace("src=\"/",
"src=\"http://www.buraksenyurt.com/"));
           else
              newContent = (post.Content.Replace("src=\"image",
"src=\"http://www.buraksenyurt.com/image"));

           builder.AppendLine(newContent);
           // Sanırım bir boşluk burada iyi gider
```

```
            builder.AppendLine("</br>");

            #region Write Tags
            // Tag' lerini de yazdırayım da konunun hangi alanlara hitap ettiği anlaşılsın
            builder.Append("<b>Tags : </b>");
            foreach (var tag in post.Tags)
            {
               builder.Append(String.Format("{0}|",tag.FirstAttribute.Value));
            }
            // Bir boşluk daha iyi gider mi? Gider.
            builder.AppendLine("</br>");

            #endregion
         }
         // Nihayet! Şimdi tüm string' i html uzantılı bir dosyaya yazayım ki tarayıcılarda açabileyim. Hele ki internet bağlantısı da var ise tüm resimler görünür.
         File.WriteAllText("BurakSenyurt_AllPublishedPost.html",
builder.ToString(),Encoding.Unicode);
      }
   }
}
```

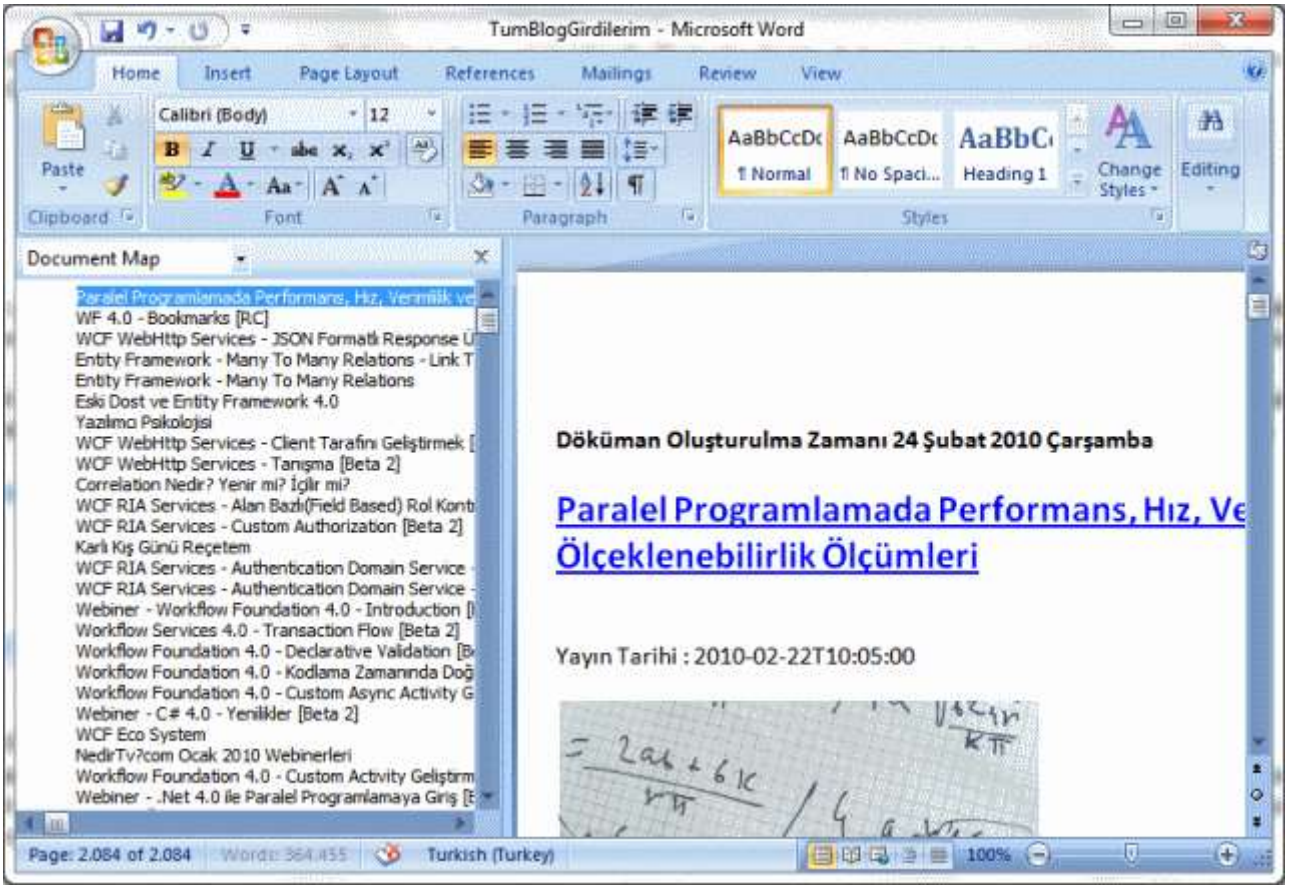
Kodu çok fazla açıklama gereği duymuyorum ama basit anlamda **XML** içeriğini **LINQ to XML** nimetlerinden yararlanarak ele aldığını ve **StringBuilder** tipini kullanarak bir **HTML** dosyasıürettiğini ifade edebilirim. İşte uygulamanın çalışması sonucu ilgili klasör altında oluşan HTML dosyası.



Gördüğünüz gibi **HTML** dosyası başarılı bir şekilde oluşturuldu. **11 Mb** civarında. Wovvv!!! Tabi kodun bu şekilde bir sonuç ürettiğini gördükten ve **HTML** içeriğinin tamamının yüklenmesinin tarayıcılara göre çok yavaş olmasından ötürü aklıma başka bir fikir daha geldi. Dedim ki, bu **HTML** içeriğini ben **Word** formatında kayıt edersem çok daha yararlı bir ürün ortaya çıkabilir. Böylece herkesin indirip istediği gibi kullanabileceği*(kötü niyetli olarak değil tabiki de)* bir döküman oluşabilirdi.

Artık bunun için de kod yazayım mı yazmıyayım mı diye düşünürken **Firefox** ile açtığım **HTML** içeriği üzerinde **Select All** yaptım. Aman Allahım. Yapmaz olaydım. **Firefox** kitlendi. E haklı tabi. **11 Mb**' lık Web sayfası içeriği ona fazla geldi. Peki dedim birde **Google Chrome**' u deniyeyim. Harika...Başarılı bir şekilde **Select All** ve **Copy** işlemi yapıldı. Ama **Word** içerisine **Paste** edilebilen hiç bir şey yoktu ortada. Uzun süre bekledim ama zaten sabırsız birisi olduğumdan onu da kapattım. En başta söylemem gerekeni en sonda söyledim...Dedim ki; **iki Microsoft ürünü birbirleriyle sorunsuz anlaşır**. **Internet Explorer 8** ile **HTML** dosyasını açtım. Dosya açıldı ama resimlerin yüklenmesi **Firefox**'a hatta**Chrome**'a göre çok daha yavaştı. Eyvah dedim...Ancak en azından sol alt köşede **1600** küsür resimden kaçıncısını yüklediğini söylüyordu. Buna göre tahmini olarak ne kadar bekleyeceğimi biliyordum. Resimler yüklendikten sonra **Select All** yaptım. Belki bir kaç on saniye bekledikten sonra tüm içerik seçili haldeydi. Hemen **Word 2007** tarafına geçtim ve **Paste** işlemini uyguladım. Volaaaaa!!! İşlem başarılıydı. Tabi **HTML** içeriğini oluştururken **Title**'

lara **Heading** takısını uygulamanın avantajlarını da hemen gördüm. Nitekim aşağıdaki resimden de görüldüğü üzere Document Map otomatik olarak oluşturulmuştu.



Tabi **Word** dökümanının boyutu **30 Mb**' ı geçmişti. 😕 Neyseki hosting firması **DiscountAsp.Net**' in sitem için ayırdığı **1 gb**' a varan alanı bu boyut için müsaitti. Tabi çok doğal olarak ilk yazılarda yer alan **Flash** nesneleri bu **Word** dosyası içerisinde oynamıyor. Ama **Title**' ların aynı zamanda makale linki olduğunu hatırlatmak isterim. Dolayısıyla internet bağlantınız mevcut ise doğrudan **buraksenyurt.com** adresine de gidebilirsiniz.

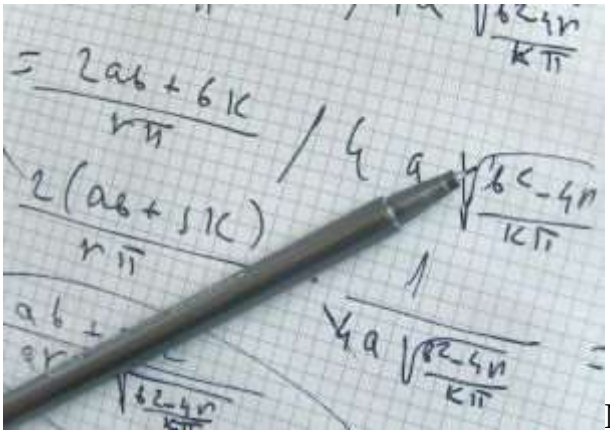
Not : 4 Gb Ram'e sahip çift çekirde intel işlemcili makinemde, Windows 7 işletim sisteminin yönetimi altındaki Word 2007 programı, söz konusu dökümanı yaklaşık olarak 40 saniye civarlarında açabilmekte.

Umarım işinize yarar. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

AllPosts.rar (30 Mb)

Paralel Programlamada Performans, Hız, Verimlilik ve Ölçeklenebilirlik Ölçümleri (2010-02-22T10:05:00)

parallel programming,task parallel library,.net framework 4.0,c# 4.0,



Merhaba Arkadaşlar,

Ben **Matematik Mühendisliği** eğitimi almış bir bireyim. öğrenim hayatım boyunca en çok yaptığım işlerden birisi, matematiksel teoremlerin bilimsel ispatlarını gerçekleştirmek olmuştur. Hemen hemen mühandisliğin her alanındaki farklı problemlerin modellenmesi ve ispatlarının yapılarak en uygun yol olduklarının gösterilmesi adına pek çok kağıt karalamış ve tüketmişimdir. Zaman zaman neden yaptığımızı anlamadığım ispatlardan tutunda, lastiğine konumuş olan sineğin bisikletin ileriye yönlü ama düz bir rotada olmayan hareketi boyunca çizdiği sarmalımsının denklemini çıkartmaya kadar matematiğin bir o kadar garip ama gizemli evreninde dolaşıp durduğumu hatırlıyorum. Hatta bir gün Matematik Analiz dersinin finalinde karşılaştığım bir soruda ne hikmetse **1=2** sonucuna ulaşmışımdır. Halbuki **1=1'** e ulaşmış olmam gerekirdi.

Tabi zaman ilerleyip iş para kazanmaya gelince kimsenin teorem ispatları ile uğraşmadığını acı olarak farketmiştim. İlgi duyduğum yazılım sektörüne gireli uzun yıllar olduğu için matematiksel teorem ispatlarında tam anlamıyla pas tutmuş durumdayım. Yine de zaman zaman yazılım içerisinde matematiği basit haliyle bile görebilmek, en azından bazı konuların ispatında kullanabiliyor olmak sevindirici.

Gelelim bu yazımızın konusuna. Bildiğiniz üzere bir süredir paralel programlama ile ilişkili konuları incelemeye çalışıyor ve öğrenebildiklerimi sizlerle paylaşıyorum. Yine bu vesile ile geçtiğimiz haftalar içerisinde internette dolaşırken gözüme ilişen kısa ve özlü bir yazı ile karşılaştım. Microsoft Paralel programlama takımı tarafından yayınlanan FAQ girdisinde, çeşitli kriterlere göre hangi kodun daha iyi olduğunun ölçümlenebilmesi için hangi kriterlere bakılabileceği özetlenmektedir. Kısaca elimizde aşağıdaki tabloda yer alan kriterler mevcut. Bu kriterlere göre seri ve paralel olarak yazılmış kod algoritmalarının kıyaslanması mümkün. özellikle yazdığımız paralel program kodlarının normal versiyonlarına göre daha iyi olup olmadıklarının tespit edilmesi noktasında son derece mühim kriterler olduklarını düşünmekteyim.

Kriter	**Açıklama**
Performance(Performans)	Genel olarak seri ve paralel yazılmış kodların performans ölçüm olarak bir algoritmanın yürütülme süresinin diğerine göre daha
SpeedUp(Hızlanma)	Hızlanma değerini hesap etmek için şu formül kullanılır;

	SpeedUp = Seri çalışma Süresi / Paralel çalışma Süresi Bu formülün sonucuna göre bir algoritmanın diğerinden kaç kat
Efficiency(Verimlilik/Etkinlik)	Tabiki yazılan algoritmanın çalıştırıldığı seri veya paralel işleyiş hangisinin daha verimli olduğudur. Verimliliği veya etkinliği he **Efficiency = Hızlanma / İşlemci çekirdek Sayısı**
Scalability(ölçeklenebilirlik)	ölçeklenme bilimsel alanda son derece yaygın kullanılan etkili b denemelerine ait **SpeedUp** değerlerinin farklı sayıda çekirdek/iş ile**SpeedUp** değerlerinin düşmesi bir başka deyişle daha hızlı sc işlemci/çekirdek sayısının artması dolayısıyla ortamın büyümes ***Kişisel Not :*** *Her ne kadar söz konusu yazıda sadece çekirdek sa algoritmanın seri veya paralel çalışması durumlarındaki ölçekl*

Haydi gelin buradaki ölçüm değerlerini örnek bir kod üzerinden incelemeye çalışalım. Bu amaçla **Visual Studio 2010 Ultimate RC** ve **.Net Framework 4.0 RC** üzerinde basit bir **Console**uygulaması geliştiriyor olacağız. örneğe ait senaryomuz ise şu şekilde olacaktır.

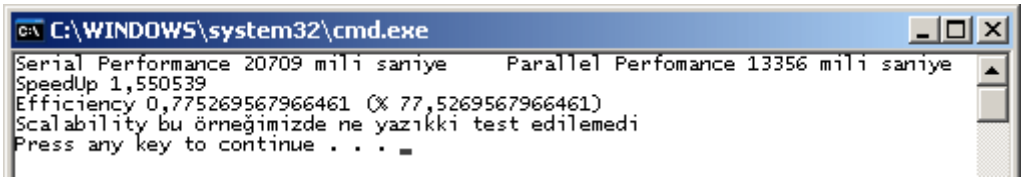
Bir klasör içerisinde yer alan **jpg** uzantılı resim dosyalarının boyutlarının arttırılmasını ele alan bir kod parçası geliştireceğiz. Resimlerin boyut arttırım işleminin zaman alan ve yorucu bir işlem olduğu düşünüldüğünde, seri ve paralel kodların ürettiği sonuçları yukarıdaki kriterler eşliğinde değerlendirmeye çalışacağız. Ne yazık ki elimde sadece çift çekirdekli iki makine olduğundan **Scalability** kriterini bu örnekte sizlere gösteremiyorum 😉 Ancak sizler uygun test ortamlarına sahipseniz, ölçeklenebilirlik kriterini değerlendirebilirsiniz ki değerlendirmenizi tavsiye ederim. İşte Console uygulaması kodlarımız...

```
using System;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Threading.Tasks;

namespace ProofOfConcept
{
   class Program
   {
      static void Main(string[] args)
      {
         string[] files = Directory.GetFiles("..\\..\\Images\\", "*.jpg");

         // Execution süreleri tüm kriterlerde önemlidir. Bu hesaplama için Diagnostics isim alanında yer alan Stopwatch tipinden yararlanılır
         Stopwatch watcher = new Stopwatch();
```

```
            watcher.Start();
            SerialExecution(files);
            watcher.Stop();
            // Seri çalışmanın toplam işlem süresi milisaniye cinsinden elde edilir
            float serialElapsed=Convert.ToSingle(watcher.ElapsedMilliseconds);

            watcher.Restart();
            ParallelExecution(files);
            watcher.Stop();
            // Paralel çalışmanın toplam işlem süresi milisaniye cinsinden elde edilir
            float parallelElapsed = Convert.ToSingle(watcher.ElapsedMilliseconds);

            // Kriterler hesaplanır
            Console.WriteLine("Serial Performance {0} mili saniye \t Parallel Perfomance {1}
mili saniye",serialElapsed,parallelElapsed);
            float SpeedUp = serialElapsed / parallelElapsed;
            Console.WriteLine("SpeedUp {0}",SpeedUp);
            double Efficiency = SpeedUp / Environment.ProcessorCount;
            Console.WriteLine("Efficiency {0} (% {1})", Efficiency,Efficiency*100);
            Console.WriteLine("Scalability bu örneğimizde ne yazıkki test edilemedi");
        }

        // Seri çalıştırma metodumuz
        static void SerialExecution(string[] fileList)
        {
            foreach (string file in fileList)
            {
                Resize(file);
            }
        }

        // Paralel çalıştırma metodumuz
        static void ParallelExecution(string[] fileList)
        {
            // Dosyaları Paralel ForEach döngüsüne göre ele almakta.
            Parallel.ForEach<string>(fileList, s => Resize(s));
        }

        // Resim dosyasını yeniden boyutlandırmak için kullandığımız metod
        static void Resize(string fileName)
        {
            // önce Image tipi elde edilir
            Image img = Image.FromFile(fileName);
            //Yeni genişlik ve yükseklik değerleri belirlenir. örnek olarak % 40 artım
yapılmıştır
```

```
            int newWidth = Convert.ToInt32(img.Width * 1.40);
            int newHeight = Convert.ToInt32(img.Height * 1.40);
            // Yeni boyutlarına göre bir Bitmap nesnesi örneklenir
            Bitmap btmp = new Bitmap(img, new Size(newWidth, newHeight));
        }
    }
}
```

Uygulamayı kendi makinemde (Intel Core2Duo, 2.5 Gb Ram), 45 resim(44.8 Mb) üzerinde test ettiğimde aşağıdaki sonuçları elde ettim. Burada seri ve paralel yürütmeler arasındaki farklılıklar kriterler bazında açık bir şekilde ortaya çıkmakta.



Buna göre paralel çalıştırmanın **performansı** seri olana göre daha yüksektir. Ayrıca paralel çalıştırma, seri olana göre **1,55 kata kadar daha hızlıdır**. İlaveten paralel olan çalıştırmanın seri olana göre **%77 daha verimli/etkili** olduğu sonucuna ulaşılabilir. Tabi bu kodu farklı sayıdaki işlemci veya çekirdek sayısına sahip sistemlerde defalarca test edip bir istatistik çıkartmak ve bunun sonuçlarına bakmak daha doğru olacaktır. İsterseniz bu kod parçasında farklı bir deneyimi tecrübe edebilirsiniz.
örneğin **SerialExecution** ve **ParallelExecution**metodlarını arka arkaya **10** kez çalıştırıp sonuçları **Excel** tablosunda istatistikleştirip gerekli analizleri yapabilir ve hangisini tercih edebileceğinize daha kolay karar verebilirsiniz. En azından ispatı daha güçlü kılarsınız. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ProofOfConcept_RC.rar (4,33 mb) **[örnek Visual Studio 2010 Ultimate RC sürümü üzerinde geliştirilmiş ve test edilmiştir]**

WF 4.0 - Bookmarks [RC] (2010-02-19T14:04:00)

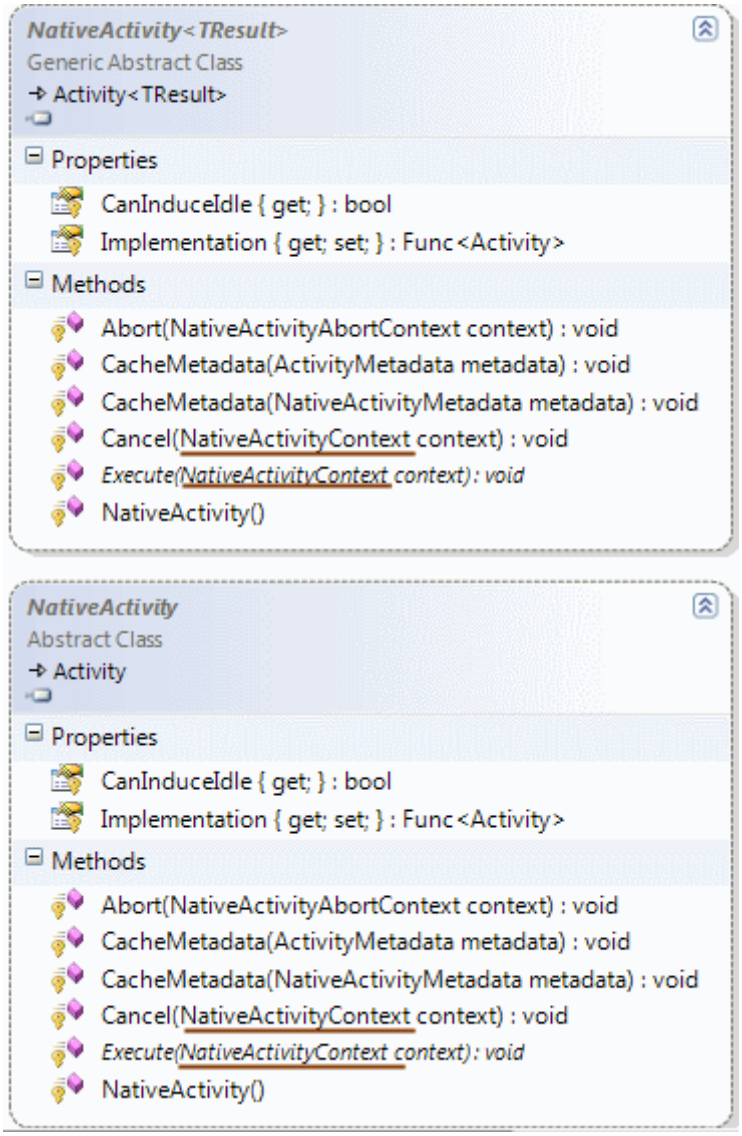
workflow foundation,wf 4.0,wf,

 Merhaba Arkadaşlar,

çalışmakta olduğum yazılım şirketinin çok yakınında kocaman bir alışveriş merkezi bulunmakta. Bazen öğle yemekleri için alışveriş merkezinin tahsis ettiği servisler ile oraya gidiyoruz. Alışveriş merkezi olduğu için tehlikeli bir yer olduğunu da söyleyebiliriz. Nitekim çok büyük bir yer ve A' dan Z'ye herşey bulunabilmekte. Arkadaşlarım ile sık uğradığım mekanlardan birisi de D&R kitap evi. çoğunlukla aylık dergilerimi almak için uğramaktayım *(Aslında Türkiye' de Amazon gibi bir kitap dağıtım evi olmadığı için çok şanslı olduğumu düşünüyorum. Her halde öyle bir yer açılsa kazancımın çok büyük bir kısmı meslek kitaplarına gider)* Geçen gün yine Bilim Teknik, NTV Bilim ve NG dergilerimi almak üzere oradaydım. Sırada beklerken kasada ücretsiz olarak verilen kitap ayraçlarını farkettim. Hep görürdüm ama bu gün biraz daha anlamlı geliyorlardı. üzelerinde çeşitli reklamlar veya faydalı bilgiler bulunan bu ayraçlar yardımıyla*(ki**Bookmark** diyebilir miyiz acaba?)*, okuduğumuz kitabın neresinde kaldığımızı kolayca hatırlayabildiğimizi düşünmeye başladım. Derken evde uzun süredir el değdirip kaldığım yerden devam edemediğim kitaplarım aklıma geldi. Hüzünlendim... Tesadüfe bakın ki bu kitapları okumak baya uzun sürmüştü. Zamanın neresinde okumaya başladığımı pek hatırlamamakla birlikte, neresinde kaldığımı da hatırlamadığım bir kaç kitap...Tesadüfe bakın ki bu uzun sürecin benzeri Workflow Uygulamalarında da söz konusu olabilmekteydi.

Aslında **.Net Framework 3.5** sürümünde **Uzun Süreli İşemlerin(Long Running Process)** için **ExternalDataExchangeService** veya **WorkflowQueue** tiplerinden yararlanılmaktadır. Ne varki**Workflow Foundation 4.0** sürümünde, geliştiricilerin kulağına daha hoş gelen **Bookmark** kavramı ile karşılaşmaktayız. Peki **Bookmark** nedir? Ne işe yaramaktadır? Nasıl kullanılmaktadır? Konuyu anlamanın belki de en kolay yolu her zaman ki gibi basit bir örnek üzerinden ilerlemekle olacaktır. Bu nedenle Bookmark kavramının tanımlamasını yazımızın sonunda yapmaya çalışacağız.

Bookmark kullanımında önemli olan noktalardan birisi, geçici olarak duraksatılabilecek(Pause) **Activity** bileşeninin **Workflow'** un çalışma zamanı içeriğine ulaşabiliyor olmasıdır. Bu nedenle en uygun aktivite bileşenleri, **NativeActivity(veya NativeActivity<T>)** türevli olanlardır. Bunu ilk gereksinimimiz olarak düşünebiliriz. Aşağıdaki diagramda **.Net Framework 4.0 RC**sürümü içerisinde yer alan **NativeActivity** tipleri ve üyeleri görülmektedir.

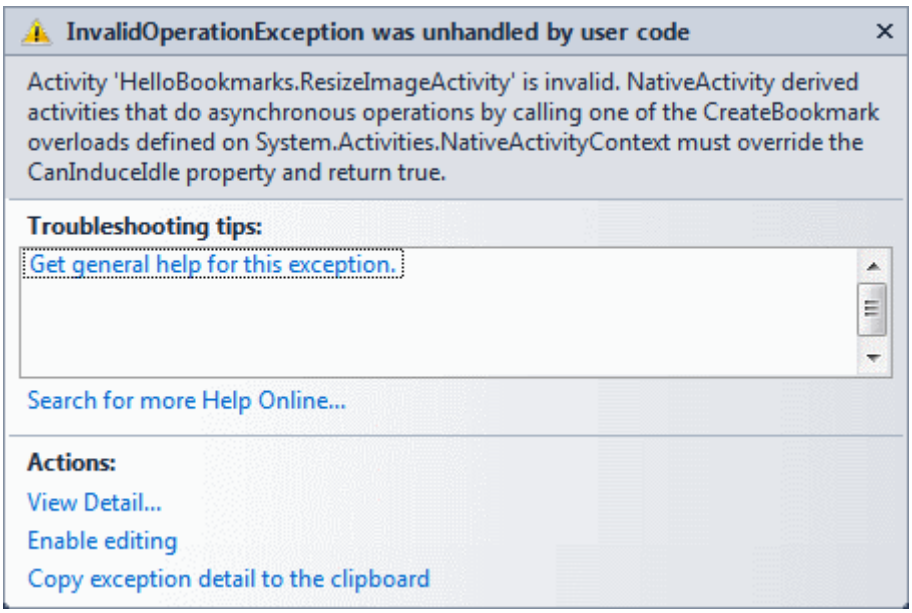


Şimdi bu türetmeyi kullanarak aşağıdaki kod parçasında görülen aktivite bileşenini geliştirdiğimizi düşünelim. İlgili örneğin bir **Workflow Console Application** üzerinden geliştirebiliriz.

```
using System;
using System.Activities;

namespace HelloBookmarks
{

   public sealed class ResizeImageActivity
      : NativeActivity
   {
      protected override bool CanInduceIdle
      {
         get
         {
```

```
            return true;
        }
    }

    protected override void Execute(NativeActivityContext context)
    {
        Console.WriteLine("Resize Image bir takım işlemler yapıyor");
        // Bazı işlemler
        // İkinci parametre BookmarkCallback temsilcisi tarafından işaret edilen bir
fonksiyondur.
        context.CreateBookmark("ResizeImageBookmark",
            (nac, b, obj) =>
            {
                Console.WriteLine("Resume edilen bookmark adı
{0}",b.Name);
            }
        );
    }
  }
}
```

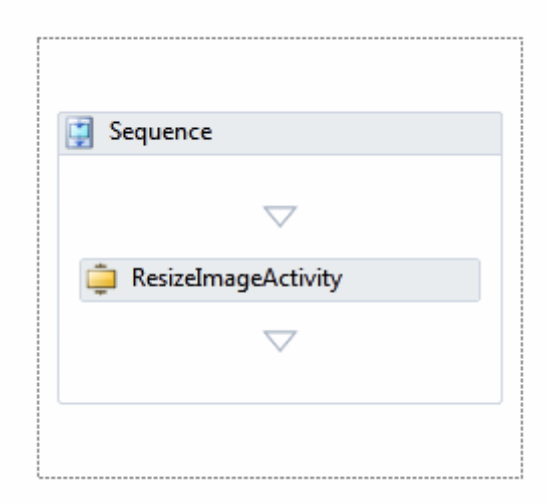
Bookmark işlemleri **Idle** konuma düşebilen **Workflow** aktivitelerinde işe yarayacak bir teknik olarak düşünülmelidir. Nitekim bir aktivite içerisinde herhangibir zamanda **Pause** etme ve sonraki bir anda **Resume** etme söz operasyonları konusudur. Bu sebepten **CanIncludeIdle** özelliğinin **override** edilmesi ve geriye **true** değer döndürmesi söz konusudur. Aksi durumda çalışma zamanında aşağıda görülen **InvalidOperationException** hata mesajı alınacaktır.



CreateBookmark metodunun ikinci parametresi **BookmarkCallback** tipinden bir **temsilcidir(Delegate)**. Bu temsilcinin yapısı ise aşağıdaki gibidir.

public delegate void BookmarkCallback(System.Activities.NativeActivityContext context, System.Activities.Bookmark bookmark, object value)

Buna göre örneğimizde yer alan **isimsiz metodun(Anonymous Method)** ilk parametresi ile **Activity**' nin çalışma zamanındaki çevresel içeriğine, ikinci parametre ilede **Bookmark** örneğineerişilebilir. Bu temsilci aslında bir **geri bildirim metodunu(Callback Method)** işaret etmektedir. Bir başka deyişle, **Idle** konumda kalan **Activity** örneğinin tekrar **Resume** edilmesi halinde devreye girecek olan metod olarak düşünülebilir. Dolayısıyla geri bildirim metodu içerisinde **CreateBookmark** tarafında saklanan bazı varlıkların tekrardan yüklenmesi, hazırlanması gibi operasyonlar ele alınabilir. **CreateBookmark** metodunun aslında 8 **aşırı yüklenmiş(Overload)** versiyonu bulunmaktadır. Diğer versiyonlar göz önüne alındığında dikkat çeken parametrelerden birisi **BookmarkOptions Enum** sabitidir. Bu enum sabiti **MultipleResume, NonBlocking** ve **None** değerlerinden birisini almaktadır. Varsayılan değer **None**' dır. **MultipleResume** olması halinde bir den fazla **Resume** işlemi yapılabileceği belirtilir. **NonBlocking** değerine göre ilgili aktivite bileşeni **Resume** edilmemiş olsa dahi **WF**' in çalışacağını belirtilir. Nitekim normal şartlar altında bir aktivite içerisinde oluşturulan **Bookmark**' ların tamamı **Resume** edilmediği sürece **WF**' in tamamlanması söz konusu değildir.
Dilerseniz **MultipleResume** ve **NonBlocking**değerlerini bir arada kullanabilirsiniz. Gelelim aktivitenin nasıl kullanılacağına. örneğimizdeki amacımız sadece **Bookmark** kullanımını görmek olduğundan aşağıdaki şekilde görülen **Workflow Activity** içeriğini değerlendirebiliriz.

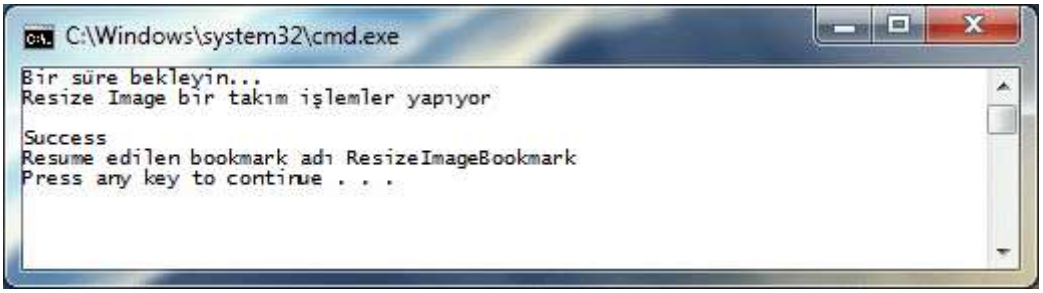


Bookmark kullanımı yazımızın başında da belirttiğimiz üzere **uzun süreli işlemler(Long Running Process)** için anlamlıdır. Bu sebepten **WorkflowApplication** tipinin kullanılması gerekmektedir. **Workflow Console Application** tipinden olan uygulamamızda, çalışma zamanındaki **Idle** durumları irdelememiz aslında son derece kolaydır. **Console.ReadLine** metodu burada çok işe yarayacaktır. İşte kod içeriğimiz;

```
using System;
using System.Activities;
using System.Threading;

namespace HelloBookmarks
{

   class Program
   {
      static void Main(string[] args)
      {
         AutoResetEvent rE = new AutoResetEvent(false);

         // Workflow örneği oluşturlur
         Workflow1 wf1 = new Workflow1();
         // Workflow Application örneği oluşturulur
         WorkflowApplication wfApp = new WorkflowApplication(wf1);
         // Workflow' un tamamlanması sonrası devreye girecek Completed olay metodu
         wfApp.Completed = (e) => { rE.Set(); }; // işlemlerin bittiğine dair bilgilendirme için AutoResetEvent örneğinin Set metodu çağırılır.
         // Workflow çalışma zamanı başlatılır dolayısıyla Workflow1 örneği yürütülür
         wfApp.Run();
         Console.WriteLine("Bir süre bekleyin...");
         Console.ReadLine(); // Bu noktada Workflow1 örneğinin Idle konuma geçmesi söz konusudur.

         /* Kullanıcı devam etmek istediğinde Bookmark' lanan Workflow1 örneğine tekrardan hayata geçirilir. ResumeBookmark metodunun ilk parametresi dikkat edileceği üzere ResizeImageActivity içerisinde kullanılan Bookmark adıdır. Bu adın aslında aktivite bileşeni içerisinden çalışma zamanı ortamına verilmesi(örneğin bir OutArgument) ile faydalı olabilir. İkinci parametre ise hangi Workflow örneğinin Resume edileceğidir. Buna göre Workflow1 içerisinde ResizeImageBookmark isimli Bookmark' ın yer aldığı aktivite bileşeninin Resume edilmesi söz konusudur. */
         BookmarkResumptionResult result = wfApp.ResumeBookmark("ResizeImageBookmark", wf1);
         // ResumeBookmark metodunun sonucu olan Enum sabitinin değerine göre bir işlem yapılabilir
         switch (result)
         {
            case BookmarkResumptionResult.NotFound:
               Console.WriteLine("Not Found");
               break;
            case BookmarkResumptionResult.NotReady:
               Console.WriteLine("Not Ready");
               break;
```

```
            case BookmarkResumptionResult.Success:
               Console.WriteLine("Success");
               break;
            default:
               break;
         }

         // Eğer Workflow Application' ın beklediği çalışan örnekler var ise bunların tamamlanması beklenir
         rE.WaitOne();
      }
   }
}
```

Aslında örnek kodumuz **Workflow1** tipinden bir nesne örneğini çalıştırmakta ve yaşamı içerisinde kullanıcından belirli süreliğine tuşa basmasını beklemektedir. Tuşa basmayı beklediği sırada ise **Idle** olabilen bileşenlerin bu konuma geçmesi söz konusudur. **ResumeBookmark** çağrısından sonra ise **Bookmark** ile Pause konumda duran aktivitenin ilgili geri bildirim metodunun çağırılması ve dolayısıyla yürütülmeye devam edilmesi sağlanır. İşte örnek program kodumuzun çalışma zamanı çıktısı.



Bu arada çalışma zamanında aktif olan **Bookmark** listesini de **WorkflowApplication** nesne örneğine ait **GetBookmarks** metodu üzerinden alabileceğinizi belirtmek isterim. Aşağıdaki kod parçasında bu durum örneklenmektedir.

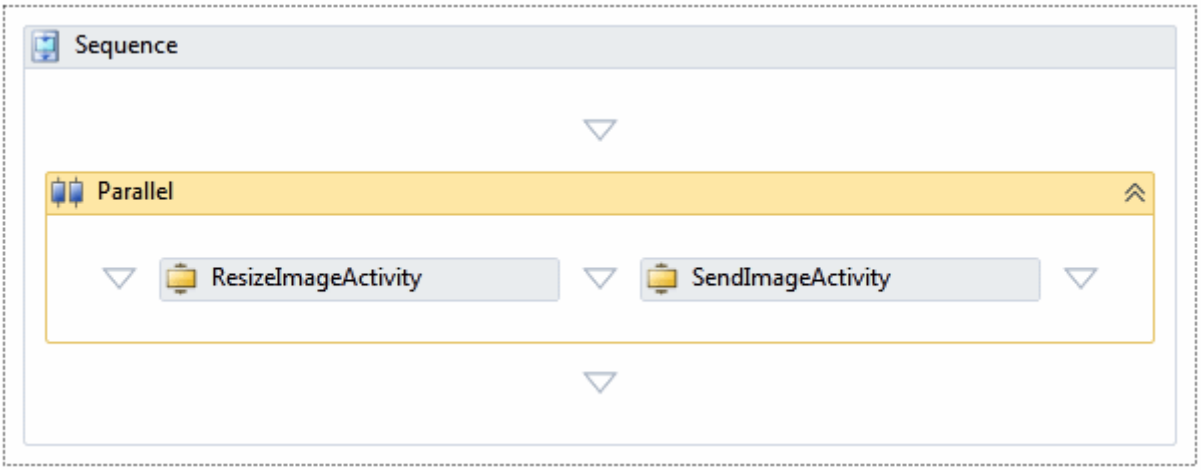
```
using System;
using System.Activities;
using System.Threading;
using System.Activities.Hosting;

namespace HelloBookmarks
{
   class Program
   {
      static void Main(string[] args)
      {
         AutoResetEvent rE = new AutoResetEvent(false);
```

```
        Workflow1 wf1 = new Workflow1();
        WorkflowApplication wfApp = new WorkflowApplication(wf1);
        wfApp.Completed = (e) => { rE.Set(); }; // işlemlerin bittiğine dair bilgilendirme
için AutoResetEvent örneğinin Set metodu çağırılır.
        wfApp.Run();
        Console.WriteLine("Bir süre bekleyin...");
        Console.ReadLine();
        foreach (BookmarkInfo bm in wfApp.GetBookmarks())
        {
           Console.WriteLine(bm.BookmarkName);
        }
...
```

Şimdi örneğimizi biraz daha ilginç bir hale getirelim. öncelikli olarak aşağıdaki kod içeriğine sahip **Custom Activity** sınıfını oluşturarak projemize ilave edelim.

```
using System;
using System.Activities;

namespace HelloBookmarks
{

   public sealed class SendImageActivity
      : NativeActivity
   {
      protected override bool CanInduceIdle
      {
         get
         {
            return true;
         }
      }

      protected override void Execute(NativeActivityContext context)
      {
         Console.WriteLine("Send Image bir takım işlemler yapıyor");
         context.CreateBookmark("SendImageBookmark",
            (nac, b, obj) =>
            {
               Console.WriteLine("Resume edilen bookmark adı {0}",b.Name);
            }
         );
      }
   }
}
```

ResizeImageActivity tipinin tıpkısının aynısı olan bu bileşeni de ele alarak **Workflow1** içeriğini aşağıdaki şekilde görüldüğü gibi değiştirelim.

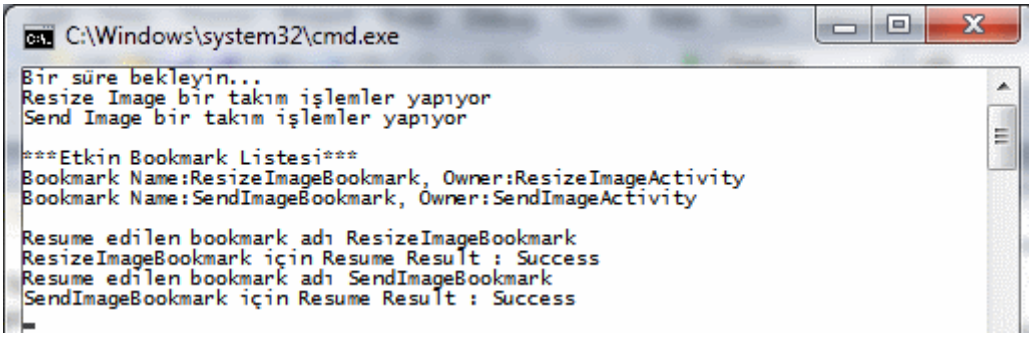


Bu sefer aynı anda birden fazla aktivitenin çalıştırılmasına izin veren **Parallel** bileşeninden yararlanmaktayız. İncelemek istediğimiz nokta ise şu; her iki aktivite de **Execute** metodları içerisinde birer **Bookmark** oluşturmaktadır. Buna göre program kodumuzda birden fazla **Bookmark**' ın nasıl ele alınacağına bakmak istiyoruz. Söz konusu vakayı analiz etmek için, **Main**metoduna ait kod içeriğini aşağıdaki gibi değiştirmemiz yeterli olacaktır.

```
using System;
using System.Activities;
using System.Threading;
using System.Activities.Hosting;

namespace HelloBookmarks
{
   class Program
   {
      static void Main(string[] args)
      {
         AutoResetEvent rE = new AutoResetEvent(false);
         Workflow1 wf1 = new Workflow1();
         WorkflowApplication wfApp = new WorkflowApplication(wf1);
         wfApp.Completed = (e) => { rE.Set(); };
         wfApp.Run();
         Console.WriteLine("Bir süre bekleyin...");
         Console.ReadLine();
         Console.WriteLine("***Etkin Bookmark Listesi***");
         foreach (BookmarkInfo bm in wfApp.GetBookmarks())
         {
            Console.WriteLine("Bookmark Name:{0},
Owner:{1}",bm.BookmarkName,bm.OwnerDisplayName.ToString());
```

```
            }
            Console.WriteLine();

            BookmarkResumptionResult result1 =
wfApp.ResumeBookmark("ResizeImageBookmark", wf1);
            BookmarkResumptionResult result2 =
wfApp.ResumeBookmark("SendImageBookmark", wf1);

            Console.WriteLine("ResizeImageBookmark için Resume Result :
{0}",result1.ToString());
            Console.WriteLine("SendImageBookmark için Resume Result : {0}",
result2.ToString());

            rE.WaitOne();
            Console.ReadLine();
        }
    }
}
```

Ve işte çalışma zamanı sonuçları;



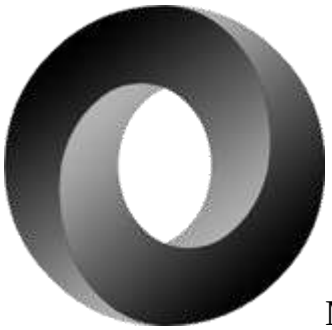
çıktımıza göre her iki akitivite bileşeni, eş zamanlı olarak **Execute** metodlarını icra etmiş ve birer **Bookmark** oluşturmuştur. Sonrasında kullanıcının tuşa basması ile devam eden süreçte ilk olarak yüklü olan **Bookmark**' lar listelenmiş ve ardından **ResumeBookmark** çağrıları nedeniyle **Pause** edilmiş olan aktivitelerin geri bildirim operasyonları devreye girmiştir. Her iki **Bookmark**' ında **Resume** edilmesinin sonucu olarak **Workflow1** örneğinin işlemlerini tamamladığı anlaşılmaktadır. Program kodu da buna göre sonlanır.

Görüldüğü gibi bir aktivitenin **Bookmark**' lanması aslında isimlendirilmiş duraksama noktalarına(Named Pause Points) sahip olması anlamına gelmektedir. öyleki, **ResumeBookmark** metodu sayesinde **Pause** edilen noktadan tekrar yürütülmeleri sağlanabilir. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HelloBookmarks.rar (54,52 kb) **[örnek Visual Studio 2010 Ultimate RC sürümü üzerinde geliştirilmiş ve test edilmiştir]**

WCF WebHttp Services - JSON Formatlı Response Üretmek (2010-02-19T07:50:00)

wcf,wcf 4.0,webhttp services,restful,non soap,wcf webhttp services,



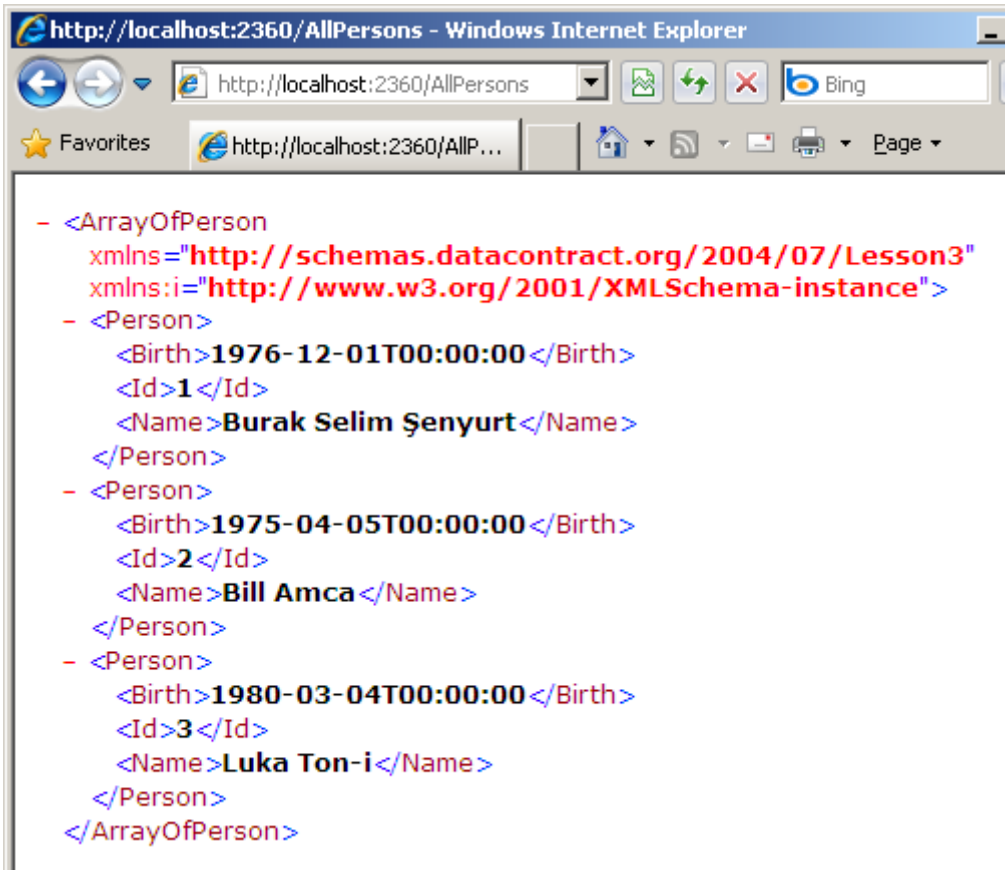
Merhaba Arkadaşlar,

Yandaki Logo size neyi çağırıştırıyor? Aslında bakarsanız çok meşhur olan hafif siklette bir veri değiş tokuş formatının logosunu ifade etmekte.**JSON(JavaScript Object Notation).** Hatırlayacağınız üzere bir süredir **WCF Eco System** içerisinde yer alan **WCF WebHttp Service** alt yapısını incelemeye çalışıyoruz. **WCF WebHttp Service**' leri eğer istemci tarafından aksi belirtilmezse varsayılan olarak **XML** formatında çıktı üretmektedir. Ancak istenirse**JSON(JavaScript Object Notation)** formatında çıktı üretmeside sağlanabilir. Söz konusu çıktı üretim işlemi iki yolla gerçekleştirilebilir. **Bilinçli olarak(Excplicitly)** veya otomatik olarak. Bu yazımızda söz konusu yolları inceleyerek **JSON** formatında çıktıları nasıl verebileceğimizi basit bir örnek üzerinden görmeye çalışıyor olacağız. örnek uygulamamızı bu kez **Visual Studio 2010 Ultimate RC** ortamı üzerinde geliştirdiğimizi belirtelim. Dolayısıyla ilerleyen sürümde bazı farklılıklar olabilir. **Lesson3** isimli **WCF REST Service Application** uygulamamız içerisinde yer alan servis sınıfı içeriğimiz çok basit olarak aşağıdaki kod parçasından oluşmaktadır.

```
using System;
using System.Collections.Generic;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;

namespace Lesson3
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class PersonalityService
```

```
    {
        [WebGet(UriTemplate = "AllPersons")]
        public List<Person> GetAllPersons()
        {
            return new List<Person>()
            {
                new Person() { Id = 1, Name = "Burak Selim Şenyurt",Birth=new
DateTime(1976,12,1) } ,
                new Person() { Id = 2, Name = "Bill Amca",Birth=new DateTime(1975,4,5) } ,
                new Person() { Id = 3, Name = "Luka Ton-i",Birth=new DateTime(1980,3,4) }
            };
        }
    }

    public class Person
    {
        public int Id { get; set; }
        public string Name { get; set; }
        public DateTime Birth { get; set; }
    }
}
```

Servisimizde yer alan **GetAllPersons** isimli operasyon istemci tarafına **Person** tipinden bir liste içeriği döndürmektedir. Söz konusu operasyon **HTTP** protokolünün **GET** metoduna ait talepleri kabul etmektedir. Varsayılan olarak **URL** üzerinden yapacağımız **AllPersons** çağrısının sonucu aşağıdaki gibi olacaktır.



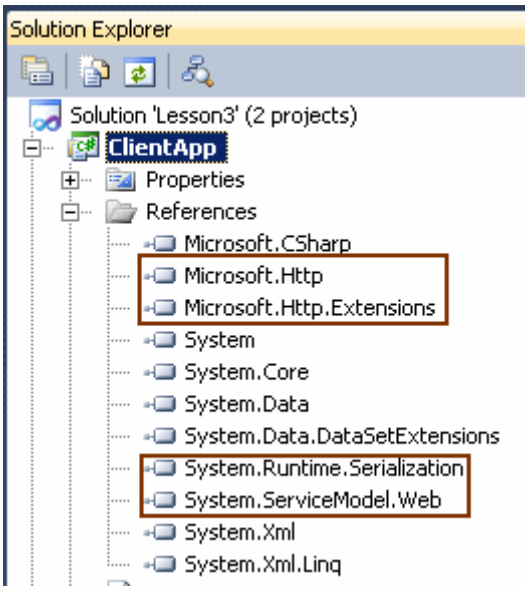
Ancak şimdiki hedefimiz bu **XML** çıktısı yerine **JSON** çıktısını vermektir. Bunu iki yol ile gerçekleştirebileceğimizden bahsetmiştik. öncelikle otomatik **JSON** çıkıtısı üretiminin nasıl gerçekleştirilebileceğine bakalım. Bu amaçla sunucu tarafındaki **web.config** dosyası içerisinde yer
alan **webHttpEndpoint** içerisindeki**standardEndpoint** elementinin **automaticFormatSelectionEnabled** niteliğinin **true** değere sahip olması yeterlidir. Aynen aşağıda görüldüğü gibi.

```
<system.serviceModel>
   <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>
   <standardEndpoints>
     <webHttpEndpoint>
       <standardEndpoint name=""
helpEnabled="true" automaticFormatSelectionEnabled="true"/>
     </webHttpEndpoint>
   </standardEndpoints>
 </system.serviceModel>
```

Peki bu otomatikliğin anlamı nedir? 😕 Nitekim herhangibir yerde JSON çıktısı vereceğimizi belirtmedik. Dolayısıyla birisinin bunu talep ediyor olması gerekmekte. Tahmin edeceğiniz üzere burada sorumluluk istemci tarafına ait. Bir başka deyişle istemci uygulama talebini gönderirken **JSON** formatında bir içerik istediğini servis tarafına

bildirmelidir. Dolayısıyla örneğimize aşağıdaki kodları içeren istemci uygulamayı yazarak devam etmeliyiz.

Not : *İstemci uygulama açısından önem arz eden konulardan biriside,* ***HttpClient*** *tipinin kullanımı için gerekli olan* ***WCF REST Starter Kit Preview 2 assmebly****' ları ile****ReadAsJsonDataContract genişletme metodunun(Extension Methods)*** *kullanımı için gerekli olan* ***System.ServiceModel.Web*** *ve* ***System.Runtime.Serialization assembly****' larını referans etmesidir.*

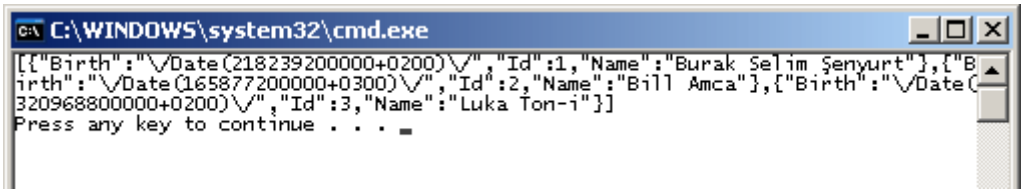


Buna göre istemci tarafının kodlarını aşağıdaki şekilde geliştirebiliriz.

```
using System;
using Microsoft.Http;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         using (HttpClient client = new HttpClient("http://localhost:2360/"))
         {
            HttpRequestMessage request = new HttpRequestMessage("GET",
"AllPersons");
            request.Headers.Accept.AddString("application/json");
            HttpResponseMessage response = client.Send(request);
            response.EnsureStatusIsSuccessful();
            HttpContent content=response.Content;
            Console.WriteLine(content.ReadAsString());
         }
```

```
        }
    }
}
```

Bu kod parçasında en çok dikkat edilmesi gereken nokta talep ile ilişkili **Header** kısmına eklenen **application/json** bilgisidir. Bu durumda **HTTP Get** metoduna göre yapılan servis çağrısı çıktısının **JSON** formatında olması istenmektedir. Servis tarafında da gelen isteğe göre bir çıktı üretildiğinden, **WCF** çalışma zamanı operasyon çıktısını **JSON**formatına dönüştürecektir. Kodun çalışması sonrasında aşağıdaki ekran çıktısı ile karşılaştığımızı görürüz.



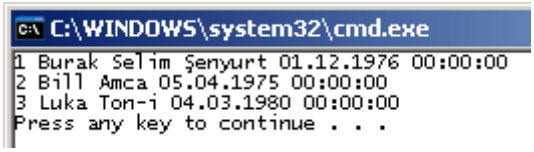
Dikkat edileceği üzere **JSON** formatında bir çıktı elde edilmiştir.

İstemci tarafına gelen bu çıktının **Person** tipini içeren bir koleksiyon şeklinde ele alınması istediğimizdeyse **HttpContent** tipi üzerinden **System.Runtime.Serialization.Json** isim alanında yer alan **ReadAsJsonDataContract** genişletme metodunu çağırabiliriz. Tabi burada istemci tarafında **Person** tipininde bir örneğinin yer aldığını varsayıyoruz ki bunu bildiğiniz üzere **WCF REST Starter Kit Preview 2** ile gelen **Paste XML As Types** seçeneği ile oluşturabiliriz. Eğer hatırlamıyorsanız biraz araştırmaya ne dersiniz? İşte istemci tarafındaki yeni kod içeriğimiz.

```
using System;
using System.Collections.Generic;
using System.Runtime.Serialization.Json;
using Microsoft.Http;

namespace ClientApp
{
    class Program
    {
        static void Main(string[] args)
        {
            using (HttpClient client = new HttpClient("http://localhost:2360/"))
            {
                HttpRequestMessage request = new HttpRequestMessage("GET",
"AllPersons");
                request.Headers.Accept.AddString("application/json");
                HttpResponseMessage response = client.Send(request);
                response.EnsureStatusIsSuccessful();
                HttpContent content=response.Content;
```

```
            //Console.WriteLine(content.ReadAsString());

            List<Person>
personList=response.Content.ReadAsJsonDataContract<List<Person>>();

            foreach (Person person in personList)
            {
               Console.WriteLine("{0} {1}
{2}",person.Id,person.Name,person.Birth.ToString());
            }
         }
      }
   }
}
```

Bu durumda çalışma zamanında aşağıdaki sonucu elde ederiz.

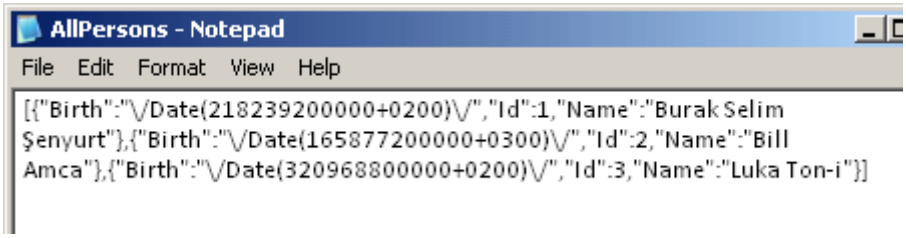


Gelelim bilinçli olarak çıktı formatının nasıl belirleneceğine. öncelikli olarak neden bilinçli bir şekilde format çıktısını söylememiz gerektiğini kavramamızda yarar olduğu kanısındayım. İstemci tarafının her zaman **HTTP** talebinin **Header** kısmına müdahale etmesi söz konusu olamayabilir. Böyle bir durumda istemcinin **JSON** formatında talepte bulunabilmesi de mümkün değildir. Dolayısıyla bu tip bir vakada **JSON** formatında çıktı verileceğinin bilinçli olarak bildirilmesi gerekmektedir. Peki ya nerede ve nasıl? Cevap: Servis tarafındaki ilgili operasyon içerisinde ve bir parça kod yardımıyla 😉 İşte GetAllPersons isimli servis operasyonumuzun bilinçli olarak JSON formatında çıktı veren yeni versiyonu.

```
using System;
using System.Collections.Generic;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;

namespace Lesson3
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class PersonalityService
   {
```

```
    [WebGet(UriTemplate = "AllPersons?whichFormat={format}")]
    public List<Person> GetAllPersons(string format)
    {
       if (format.ToLower().Equals("json"))
       {
          WebOperationContext.Current.OutgoingResponse.Format =
WebMessageFormat.Json;
       }
       return new List<Person>()
       {
          new Person() { Id = 1, Name = "Burak Selim Şenyurt",Birth=new
DateTime(1976,12,1) } ,
          new Person() { Id = 2, Name = "Bill Amca",Birth=new DateTime(1975,4,5) } ,
          new Person() { Id = 3, Name = "Luka Ton-i",Birth=new DateTime(1980,3,4) }
       };
    }
  }

  public class Person
  {
    public int Id { get; set; }
    public string Name { get; set; }
    public DateTime Birth { get; set; }
  }
}
```

İlk dikkat edilmesi gereken nokta, **WebGet** niteliğinde belirtilen **format** isimli parametre ile istemciden hangi formatta çıktı almak istendiğinin sorulmasıdır. Eğer **json** kelimesi yazılmışsa**WebOperationContext** üzerinden güncel çalışma zamanı içeriğine geçilerek cevap formatının **JSON** olacağı belirtilir ki buda dikkat edilmesi gereken ikinci noktadır. Tahmin edileceği üzere **json** dışında bir kelime girildiği takdirde varsayılan **XML** çıktısının üretilmesi söz konusu olacaktır. Servis operasyonumuzun bu son haline göre **Internet Explorer** üzerinden**http://localhost:2360/AllPersons?whichFormat=json** şeklinde bir talepte bulunursak, içeriği kaydetmemiz için bir iletişim penceresi ile karşılaşırız. İçeriği kaydettikten sonra Notepad programı ile açacak olursa aşağıdaki içeriğin üretildiğini görebiliriz.



ki buda tam anlamıyla **JSON** çıktısıdır. çıktının **JSON** veya **XML** harici formatlarda olması da söz konusudur aslında. Bu formatların nasıl ele alınacağını ise ilerleyen yazılarımızda değerlendirmeye çalışıyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Lesson3_RC.rar (173,29 kb) **[örnek Visual Studio 2010 Ultimate Beta 2 Sürümünde geliştirilmiş ancak RC sürümü üzerinde de test edilmiştir]**

Entity Framework - Many To Many Relations - Link Tablosunu Okumak (2010-02-16T09:30:00)

ado.net entity framework,

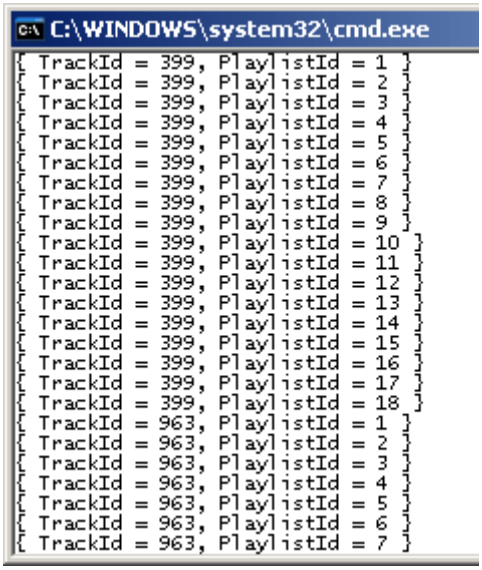
Merhaba Arkadaşlar,

Hatırlayacağınız üzere bir önceki yazımızda veritabanı tarafındaki **Many-To-Many Relation'** ların, **Entity Framework** tarafında nasıl değerlendirilebileceğine değinmeye çalışmıştık. Dikkat edilmesi gereken önemli noktalardan birisi, veritabanı tarafında yer alan ara bağlantı tablosunun **Entity Framework** üzerindeki modele alınmayışıydı. Ancak tam bu noktada ortaya bir soru işareti çıkmakta. Ya çalışma zamanında sadece ilişkili **primary key** alanlarının değerlerini elde etmek istersek. Bir önceki senaryomuza göre bu, hangi **Track** satırının hangi **Playlist'** ler ile ve hangi **Playlist** satırının hangi **Track'** ler ile ilişkili olduğunun görülmesi anlamına gelmektedir. Kısacası sadece **TrackId** ve **PlaylistId** değerlerinin eşleşmesine bakmak istediğimizi düşünebiliriz. Bu durumda aşağıdaki gibi bir kod parçası işimizi görecektir. Şükürler olsun **isimsiz tiplere(Anonymous Type)**

```
using System;
using System.Linq;

namespace ManyToMany
{
   class Program
   {
      static void Main(string[] args)
      {
         using (ChinookEntities entities = new ChinookEntities())
         {
            #region Only Related Columns

            var relations = from t in entities.Tracks
                            from p in entities.Playlists
                            where t.Name.StartsWith("Ab")
                            select new
```

```
                    {
                        t.TrackId,p.PlaylistId
                    };

            foreach (var relation in relations)
            {
                Console.WriteLine(relation.ToString());
            }

            #endregion
        }
    }
  }
}
```

Bu **LINQ** sorgusunda çoklu **Select** işlemi yapılmaktadır. Sorguya göre **Track.Name** değeri **Ab** ile başlayan parçaların **TrackId** değeleri ile **Playlist Entity** nesnesi içeriğinde yer alan **PlaylistId** değerleri arasındaki ilişkiler, yeni bir **isimsiz tip(Anonymous Type)** içerisine dahil edilmektedir. Kodun çalışması sonucu aşağıdakine benzer bir çalışma zamanı çıktısı elde edilecektir.



Arka planda çalışan **SQL Sorgusu** ise aşağıdaki gibidir.

```
SELECT
[Extent1].[TrackId] AS [TrackId],
[Extent2].[PlaylistId] AS [PlaylistId]
FROM  [dbo].[Track] AS [Extent1]
CROSS JOIN [dbo].[Playlist] AS [Extent2]
WHERE [Extent1].[Name] LIKE N'Ab%'
```

Görüldüğü üzere **Anonymous Type** kullanımı nedeniyle sadece istediğimiz **TrackId** ve **PlaylistId** alanları **Select** sorgusuna dahil edilmiştir. Buda arka planda çalışan **SQL**sorgusunda, tüm tablo alanlarının hesaba katılmaması anlamına gelmektedir ki bizim için bir avantajdır ve istediğimizde zaten budur Bir nevi veritabanında yer alan ama **Entity** olarak kod tarafına aktarılmayan ara tablo içeriğini, **LINQ** tarafında elde etmiş olduk. **Entity Framework** tarafında ipucu tadındaki başka konularlar devam ediyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Entity Framework - Many To Many Relations (2010-02-12T14:05:00)

ado.net entity framework,

Merhaba Arkadaşlar,

Bundan yıllar önce üniversiteden arkadaşım **Orkun Şentürk** ile birlikte **Altunizade Capitol** alışveriş merkezinde **Dreamcatcher** isimli bir bilim kurgu gerilim filmine gittiğimizi hatırlıyorum. *(Aslında bilim kurgu filmlerin tam bir hayranıyımdır. Ancak seyrettiklerimin hiç biri **Starwars** veya **Terminator** gibilerinin yerini tutmamakta)* Film **Stephen King'** in bir romanından uyarlanmıştı. Sevgili arkadaşım Orkun kitabını okuyarak geldiğinden, film üzerinde daha iyi yorumlar yapmıştı ancak ortak kanımız çok başarılı olmadığı yönündeydi. Açıkçası fazla etkilenmemiştik. özellikle filmden çıktıktan sonra yaptığımız muhabbet bunu doğrular nitelikteydi. **.Net Framework 1.1** ile yazmakta olduğumuz programda kullandığımız **SQL 2000** veritabanındaki **many-to-many** ilişkileri anlamaya çalışıyorduk. O zamanlar bizim için **SQL** veritabanına bağlanabilmek bile bir lüks iken bu tip alengirli konular korkutucu geliyordu. Filmden daha çok gerildiğimiz bir konuydu. Sonuçta bende sevgili **Orkun'** da evlerimizin yolunu tuttuğumuzda, odamızda bizi bekleyen masa üstü bilgisayarlar ile ne yapacağımızı gayet iyi biliyorduk. **Many-To-Many** olayını kavramak.

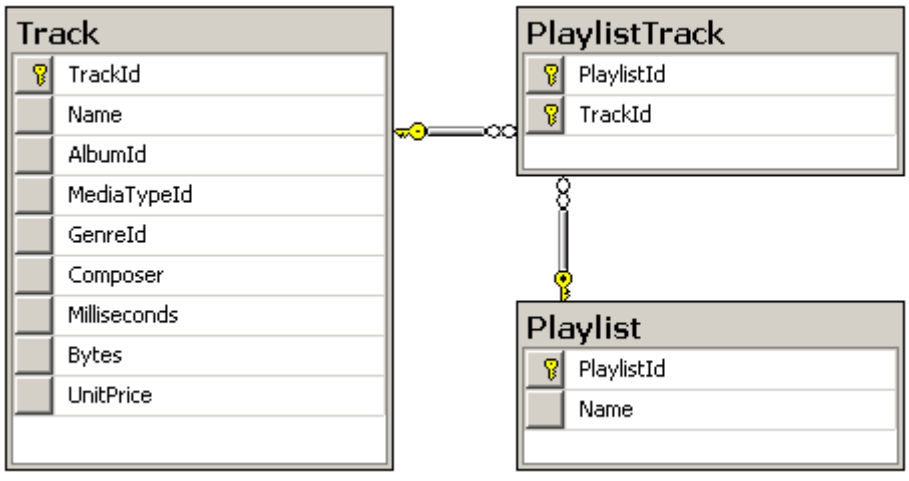
Gerçektende yazılımın ilk yıllarında hepimiz benzer durumları tecrübe etmekteyiz. Pek çok konuyu anlamakta, öğrenmekte güçlük çekiyoruz. Tabi içimizdeki öğrenme arzusunun gücüne bağlı olaraktan, ya çok çalışarak ya da önemli ve dikkat edilmesi gereken noktaları anında yakalayarak ilerleyişimizi sürdürüyoruz. Bir yazılımcı, yıllar sonra geldiği noktadan geriye doğru dönüp baktığındaysa, ilerleyişinin ne kadar hızlı olduğunu net bir şekilde görebilir aslında. Ancak bu yeterli değildir. Dilerseniz sözü fazla uzatmadan bu günkü

konumuza geçelim. Bu gün yine **Many-To-Many** ilişkileri inceliyor olacağız. Ancak bu kez olayı **Entity Framework** üzerinden değerlendireceğiz. Başlamadan önce veritabanı üzerindeki tablolar arası **ilişkilerden(Relations)** kısaca bahsetmekte yarar olduğu kanısındayım.

Genel olarak tablolar arasında **bire bir(one-to-one), bire çok(one-to-many), çoğa çok (many-to-many)** ve **self referencing** ilişkilerden bahsedilmektedir. Bire-bir ilişkilerde tabloda yer alan bir satırın diğer tabloda yine tek bir satır ile ilişkilendirilmesi söz konusudur. Yaygın olarak kullanılan bire-çok ilişkilerde ise, bir tablodaki satıra diğer bir tablodan n adet satırın bağlanabilmesi mümkündür. Söz gelimi ürün kategorilerinin tutulduğu tablodaki bir satırın, ürünlerin tutulduğu tablodaki n satırı referans etmesi gibi. Self-Referencing ilişkilerde ise bir tablonun herhangibir satırının/satırlarının, kendi içerisindeki bir satırı referans etmesi durumu söz konusudur. örneğin bir şirketin organizasyon ağacında yer alan bir personelin/personel topluluğunun kime bağlı olduğunun tutulduğu tablolarda, bu tip ilişkiler kullanılabilir.

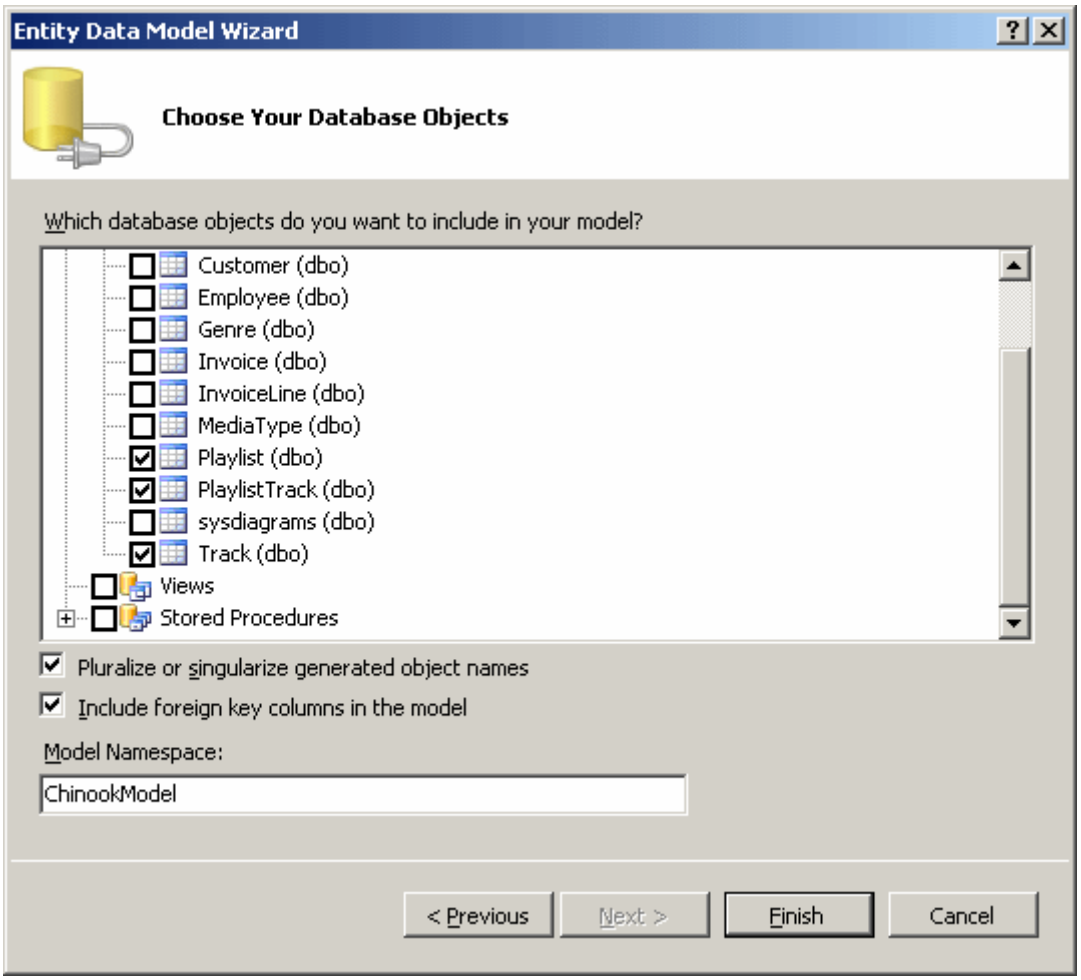
Gelelim çoğa-çok ilişkilere. örneğin filmler ve oyunculara ait tablolar olduğunu düşünelim. Burada bir oyuncunun birden fazla filmde rol alması veya bir film içerisinde birden fazla oyuncunun bulunması çok normal ve olasıdır. Bu sebepten her iki tablo birbirleri üzerinde n sayıda satırı referans edebilmektedir. Bu tip bir durumda tablolar arasındaki ilişkiyi ifade etmek için ek bir tablonun kullanılması söz konusudur. Bu tablo üzerinden sağlanan ilişkiler sayesinde çoğa-çok ilişkinin gerçeklenmesi mümkün olabilir. Peki **SQL** tarafında ek bir tablo yardımıyla ele alınan bu ilişkilerin **Entity Framework** tarafındaki yansıması nasıldır? Gelin bu durumu basit bir örnek yardımıyla incelemeye çalışalım.

İlk olarak **Chinook** veritabanında yer alan ve **SQL** tarafındaki diagramda aşağıdaki şekilde görülen ilişkilere sahip olan tabloları kullanacağımızı belirtelim.

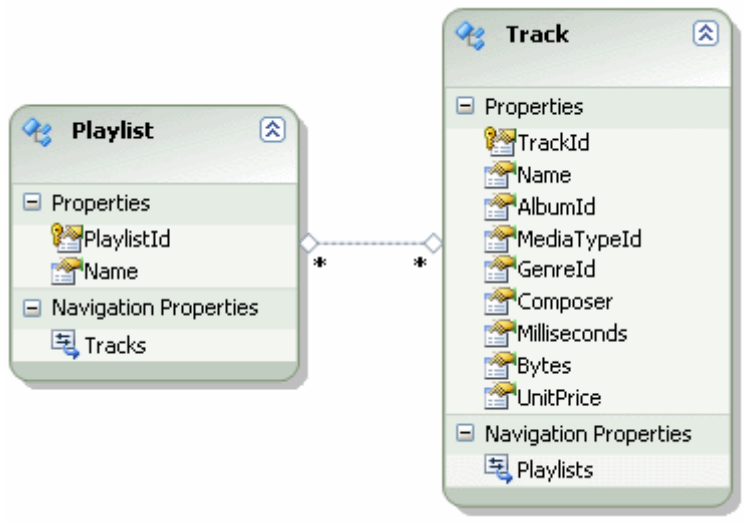


Buradaki senaryoda **Playlist** ve **Track** tabloları arasında çoğa-çok ilişki olduğu görülmektedir. Bir başka deyişle bir **Playlist** birden fazla **Track** satırına referans edebileceği gibi, bir**Track** satırı da birden fazla **Playlist** satırına referans edebilir. Şimdi bu ilişkilerin **Entity Framework** tarafındaki oluşumuna bir bakalım. *(Bu amaçla **Visual***

***Studio 2010 Ultimate RC** ve**Ado.Net Entity Framework 4.0** sürümlerini kullanıyor olacağız. Ancak bu konunun **Entity Framework'** ün önceki sürümünde de aynen geçerli olduğunu hatırlatalım.)* özellikle 3 tabloyu da seçtiğimizi düşünelim.

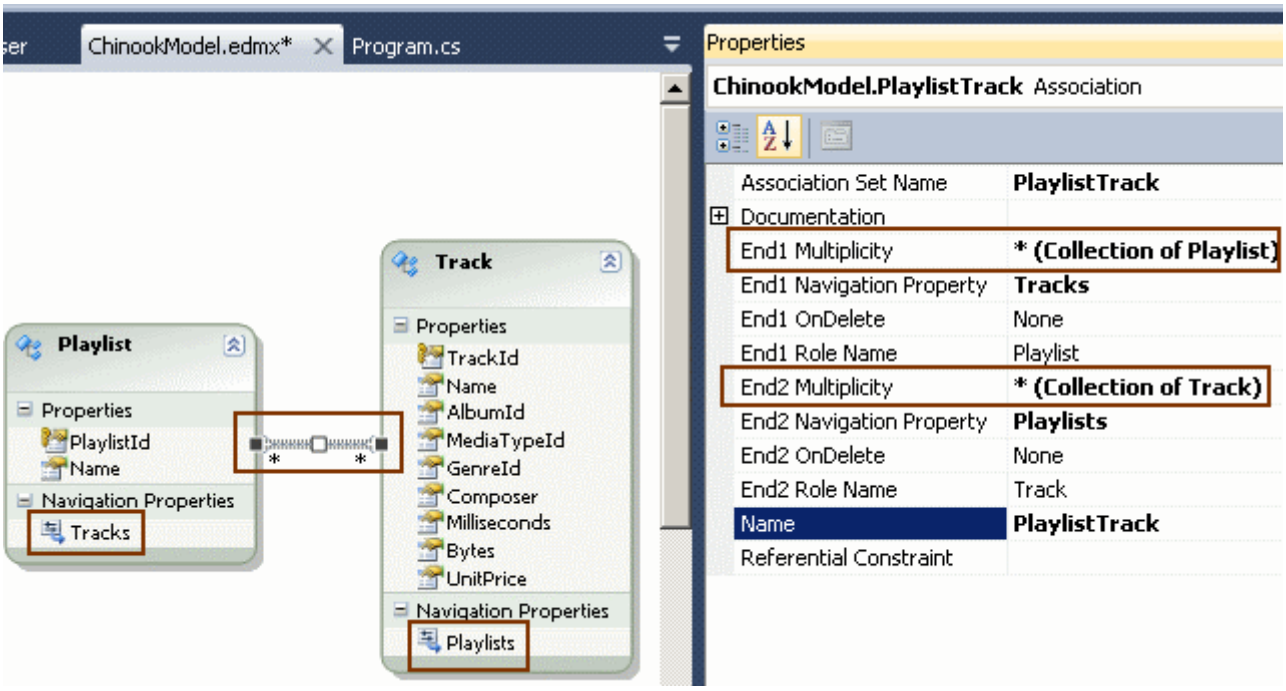


Dikkat edileceği üzere **Playlist, Track** ve **PlaylistTrack** tablolarının tamamı seçilmiştir. Sihirbaz adımlarını tamamladığımızda ise, aşağıdaki Model diagramının oluştuğunu görürüz.



Hımmm... Dikkat edileceği üzere **PlaylistTrack** tablosu model diagramına dahil edilmemiştir. Neden?

SQL tarafına bakıldığında **PlaylistTrack** tablosu üzerinde sadece **Playlist** ve **Track** tablolarına ait **Primary Key** alanları bulunmaktadır. Bunlar dışında ek bir alan yoktur. Bir başka deyişle veritabanı tarafında **Playlist** ve **Track** tablolarının **many-to-many** ilişkilerinin sağlandığı tablodur. Ancak **Entity Framework** tarafında tabloların sınıflar yardımıyla ve bu sınıfların referans ettikleri diğer sınıfların ise özellikler yardımıyla belirtildiği bilinmektedir. Dolayısıyla **Entity Framework** tarafında **PlaylistTrack** isimli bir sınıfın oluşturulmasının bir anlamı yoktur. Dahası olmamasının bir kaybı da yoktur. Bu yüzden **Playlist** ve **Track** sınıfları birbirlerine **EntityCollection<T>** tipinden olan navigasyon özellikleri yardımıyla*(Tracks ve Playlists)* doğrudan bağlanmışlardır. Bu durum ilişkiyi sağlayan bileşenin*(**Association** nesnesi)* özelliklerine bakıldığında da net bir şekilde görülebilir.

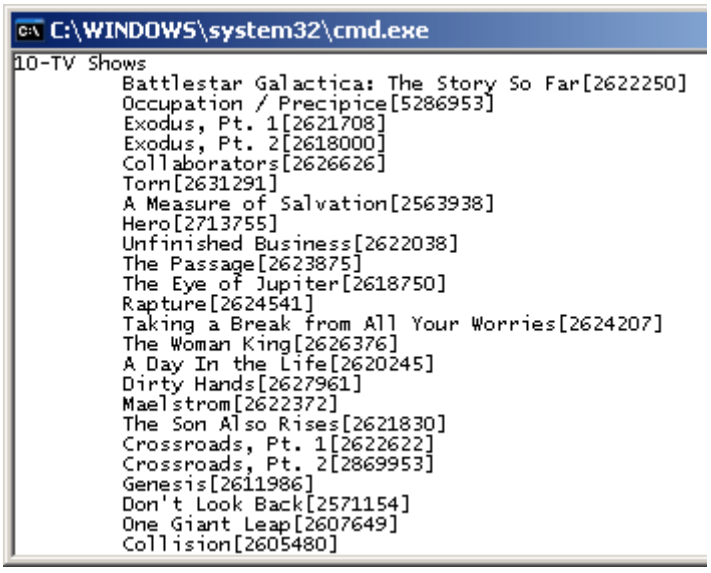


Şimdi kod tarafında **Many-To-Many** ilişkileri nasıl ele alacağımıza bakmaya çalışalım. örneğin **TV-Show** listesine ait **Track** bilgilerini getirmek istediğimizi düşünelim. Bunun için aşağıdaki gibi bir kodlama yapabiliriz.

```
using System;
using System.Linq;

namespace ManyToMany
{
   class Program
   {
      static void Main(string[] args)
```

```
        {
            using (ChinookEntities entities = new ChinookEntities())
            {
                Playlist result = (from pList in entities.Playlists
                                   where pList.PlaylistId == 10
                                   select pList).First();


                Console.WriteLine("{0}-{1}", result.PlaylistId, result.Name);
                foreach (var t in result.Tracks)
                {
                    Console.WriteLine("\t {0}[{1}]", t.Name, t.Milliseconds.ToString());
                }
            }
        }
    }
}
```

Buna göre aşağıdaki sonuçları elde ederiz.



Bu kod parçasının çalışması sonucunda **SQL** tarafında, aşağıdaki sorgunun oluşturulduğu gözlemlenecektir.

İlk önce Playlist bilgisinin çekilmesi,

```
SELECT TOP (1)
[Extent1].[PlaylistId] AS [PlaylistId],
[Extent1].[Name] AS [Name]
FROM [dbo].[Playlist] AS [Extent1]
WHERE 10 = [Extent1].[PlaylistId]
```

sonrasında ise ilgili **Playlist'** e bağlı **Track'** lerin **Inner Join** sorgusu ile elde edilmesi gerçekleşecektir.

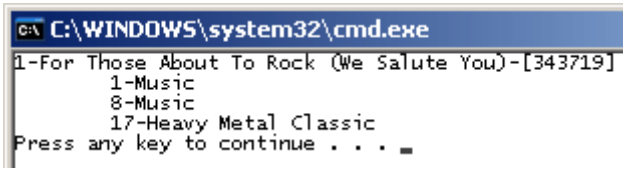
```
exec sp_executesql N'SELECT
[Extent2].[TrackId] AS [TrackId],
[Extent2].[Name] AS [Name],
[Extent2].[AlbumId] AS [AlbumId],
[Extent2].[MediaTypeId] AS [MediaTypeId],
[Extent2].[GenreId] AS [GenreId],
[Extent2].[Composer] AS [Composer],
[Extent2].[Milliseconds] AS [Milliseconds],
[Extent2].[Bytes] AS [Bytes],
[Extent2].[UnitPrice] AS [UnitPrice]
FROM  [dbo].[PlaylistTrack] AS [Extent1]
INNER JOIN [dbo].[Track] AS [Extent2] ON [Extent1].[TrackId] =
[Extent2].[TrackId]
WHERE [Extent1].[PlaylistId] = @EntityKeyValue1',N'@EntityKeyValue1
int',@EntityKeyValue1=10
```

Birde ters tarafan gitmeye çalışalım. örneğin bir **Track'** ın geçtiği **Playlist'** leri bulmak istediğimizi düşünelim. Bu durumda aşağıdaki gibi bir kod parçasını göz önüne alabiliriz.

```
using System;
using System.Linq;

namespace ManyToMany
{
   class Program
   {
      static void Main(string[] args)
      {
         using (ChinookEntities entities = new ChinookEntities())
         {
            Track track = (from t in entities.Tracks
                           where t.TrackId == 1
                           select t).First();

            Console.WriteLine("{0}-{1}-
[{2}]",track.TrackId.ToString(),track.Name,track.Milliseconds.ToString());

            foreach (var p in track.Playlists)
            {
               Console.WriteLine("\t{0}-{1}",p.PlaylistId.ToString(),p.Name);
            }
         }
```

```
        }
    }
}
```

Bu kod parasına göre, çalışma zamanında **TrackId** değeri **1** olan parçanın geçtiği **Playlist'** lerin listesini elde edebildiğimizi görürüz.



Bu kod parçası ise bir öncekine benzer olaraktan aşağıdaki **SQL** sorgularının çalıştırılmalarına neden olacaktır.

önce Track bilgileri çekilecek,

```
SELECT TOP (1)
[Extent1].[TrackId] AS [TrackId],
[Extent1].[Name] AS [Name],
[Extent1].[AlbumId] AS [AlbumId],
[Extent1].[MediaTypeId] AS [MediaTypeId],
[Extent1].[GenreId] AS [GenreId],
[Extent1].[Composer] AS [Composer],
[Extent1].[Milliseconds] AS [Milliseconds],
[Extent1].[Bytes] AS [Bytes],
[Extent1].[UnitPrice] AS [UnitPrice]
FROM [dbo].[Track] AS [Extent1]
WHERE 1 = [Extent1].[TrackId]
```

sonrasında ise ilgili **Track** satırının geçtiği **Playlist** satırlarının bulunması için gerekli **Inner Join** sorgusu çalıştırılacaktır.

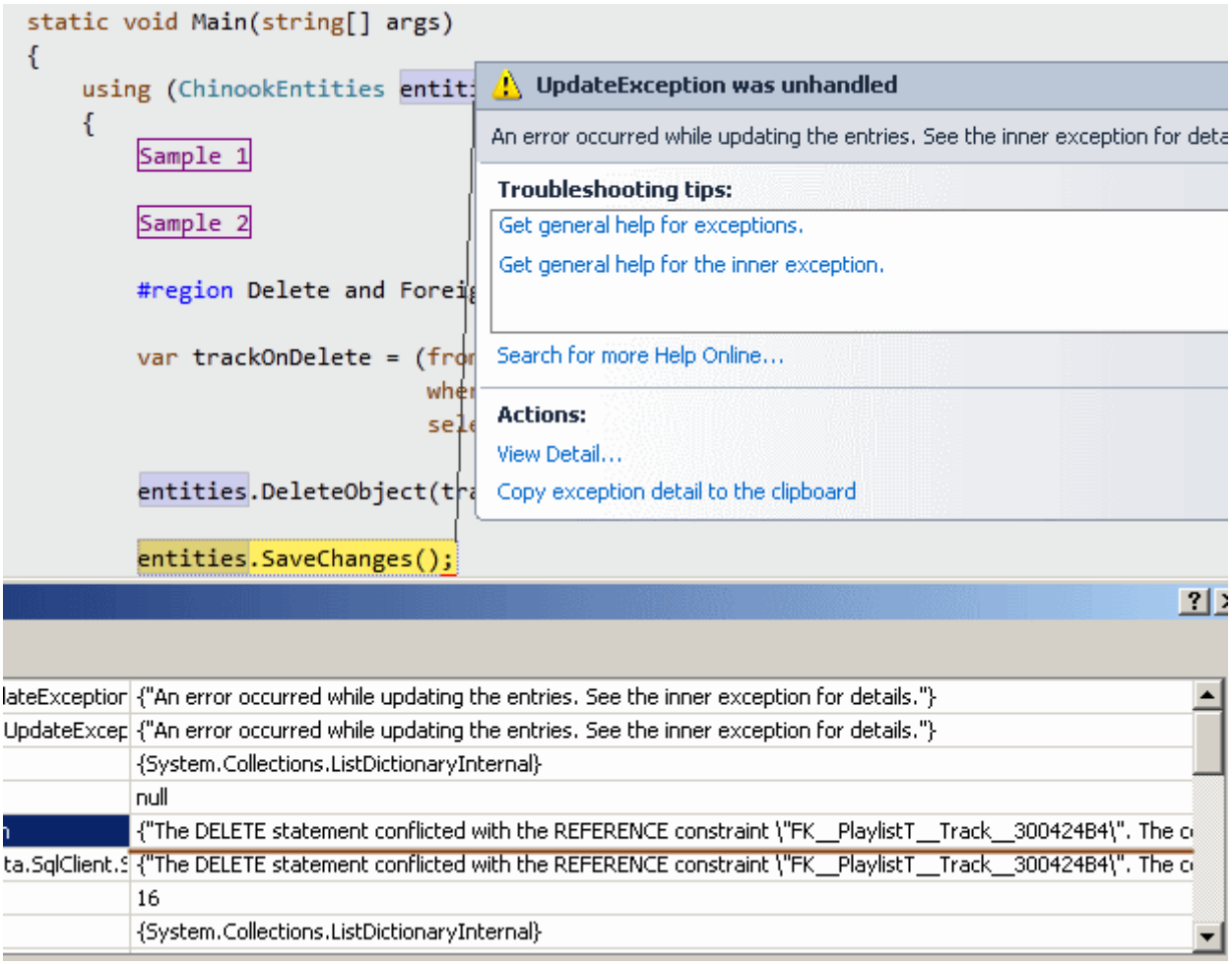
```
exec sp_executesql N'SELECT
[Extent2].[PlaylistId] AS [PlaylistId],
[Extent2].[Name] AS [Name]
FROM  [dbo].[PlaylistTrack] AS [Extent1]
INNER JOIN [dbo].[Playlist] AS [Extent2] ON [Extent1].[PlaylistId] =
[Extent2].[PlaylistId]
WHERE [Extent1].[TrackId] = @EntityKeyValue1',N'@EntityKeyValue1
int',@EntityKeyValue1=1
```

Pek tabi olarak **Many-to-Many** ilişki söz konusu olan tablolar üzerinden **Update, Delete** veya **Insert** işlemleri de yapılabilir. Burada, özellikle **Delete** operasyonlarında tarafların nasıl tepki göstereceği önemlidir. Normal şartlarda **Association** nesnesi üzerinde

yer alan **End1 OnDelete** ve **End2 OnDelete** isimli özelliklerin değerleri **none** olarak belirlenmiştir. Ancak istenirse bunlar **Cascade**değerine çekilebilir. Pek tabi ilişkinin nasıl bir tepki vereceğine bağlı olaraktan **SQL** tarafından **Foreign Key'** ler ile alakalı istisnaların alınması da muhtemeldir. Şimdi bu durumu incelemeye çalışalım.

```
using System;
using System.Linq;

namespace ManyToMany
{
   class Program
   {
      static void Main(string[] args)
      {
         using (ChinookEntities entities = new ChinookEntities())
         {
            #region Delete and Foreign Key Violation

            var trackOnDelete = (from t in entities.Tracks
                                 where t.TrackId == 3502
                                 select t).First();

            entities.DeleteObject(trackOnDelete);

            entities.SaveChanges();

            #endregion
         }
      }
   }
}
```

Bu kod parçasına göre **TrackId** değeri **3502** olan **Track** satırı silinmeye çalışılmaktadır. Ancak **Track'** lar çok doğal olarak **Playlist'** lere bağlıdır. Bu nedenle silme işlemi sırasında aşağıdaki **çalışma zamanı istisnası(Runtime Exception)** alınacaktır.



Burada sebep, söz konusu alan için bir **Relation'** ın var olmasıdır. Dolayısıyla öncelikle silinmek istenen **Track** satırı ile bağlı olduğu **Playlist** satırları arasındaki ilişkileri kaldırmak gerekmektedir. Silme operasyonunun gerçeklenmesi için, silinmek istenen **Track'** ın dahil olduğu **Playlist** nesnelerini bulup, herbirinin **Tracks** özelliği üzerinden **Remove** metodunun çalıştırılması yeterlidir. Mi acaba? 😕

Bu tip bir kod yazılmak istendiğinde elde edilen **Playlist.Tracks** özellikleri üzerinden yapılan silme hareketleri, koleksiyonunun değişmesine neden olacağından, çalışma zamanında **Concurrency** hatası alınacaktır. çok şükür ki artık elimizin altında **.Net Framework 4.0** içerisine gömülü olarak gelen **Concurrent** koleksiyonlar bulunmaktadır.(**Concurrent Collections (Eş Zamanlı Koleksiyonlar) [Beta 1]** , **Concurrent Collections : Macera BlockingCollection ile Devam Ediyor [Beta 1]**) Buna göre kodu aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.Linq;
using System.Collections.Concurrent;

namespace ManyToMany
{
```

```
class Program
{
   static void Main(string[] args)
   {
      using (ChinookEntities entities = new ChinookEntities())
      {
         #region Delete and Foreign Key Violation

         var trackOnDelete = (from t in entities.Tracks
                              where t.TrackId == 3503
                              select t).First();

         ConcurrentBag<Playlist> playList = new
ConcurrentBag<Playlist>(trackOnDelete.Playlists);
         foreach (var pl in playList)
         {
            pl.Tracks.Remove(trackOnDelete);
         }

         entities.DeleteObject(trackOnDelete);

         entities.SaveChanges();

         #endregion
      }
   }
}
```

Biraz performans kaybı söz konusu olabilir ancak silme işlemi başarılı bir şekilde gerçekleştirilebilecektir. özellikle **SQL** tarafında çalıştırılan sorgulara bakıldığında, aşağıdaki ifadelerin icra edildiği gözlemlenir.

önce **Track** ve **Playlist** tabloları arasındaki **çoğa-çok** ilişkiyi sağlayan **PlaylistTrack** tablosundaki ilgili satırlar silinir.

```
exec sp_executesql N'delete [dbo].[PlaylistTrack]
where (([PlaylistId] = @0) and ([TrackId] = @1))',N'@0 int,@1 int',@0=1,@1=3503

exec sp_executesql N'delete [dbo].[PlaylistTrack]
where (([PlaylistId] = @0) and ([TrackId] = @1))',N'@0 int,@1 int',@0=5,@1=3503

exec sp_executesql N'delete [dbo].[PlaylistTrack]
where (([PlaylistId] = @0) and ([TrackId] = @1))',N'@0 int,@1 int',@0=8,@1=3503
```

```
exec sp_executesql N'delete [dbo].[PlaylistTrack]
where (([PlaylistId] = @0) and ([TrackId] = @1))',N'@0 int,@1 int',@0=12,@1=3503

exec sp_executesql N'delete [dbo].[PlaylistTrack]
where (([PlaylistId] = @0) and ([TrackId] = @1))',N'@0 int,@1 int',@0=13,@1=3503
```

Artık sorun yoktur, nitekim **Track** tablosu ile **Playlist** arasındaki ilişkiler ortadan kalkmıştır. Buna göre son olarak, **Track** tablosundan ilgili satırın silinmesi işlemi gerçekleştirilir.

```
exec sp_executesql N'delete [dbo].[Track]
where ([TrackId] = @0)',N'@0 int',@0=3503
```

Insert ve **Update** işlemlerinin incelenmesini de siz değerli okurlarıma bırakıyorum. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ManyToMany_RC.rar (47,16 kb) **[örnek Visual Studio 2010 Ultimate RC sürümü üzerinde geliştirilmiş ve test edilmiştir]**

Eski Dost ve Entity Framework 4.0 (2010-02-09T18:28:00)

ado.net entity framework,

Merhaba Arkadaşlar,

Bildiğiniz üzere **.Net Framework 4.0** ve **Visual Studio 2010** sürümlerinin çıkmasına az bir zaman kaldı(Hatta yarın RC sürümü herkese açık olacak). Blog yazısını hazırladığım zaman itibariyle **Microsoft'** taki güvenilir kaynaklar ve **MSDN** yazarlarından edinilen bilgilere göre, sürümün **Nisan 2010** içerisinde çıkması bekleniyor. Tabi yeni bir sürüm denilince eski sürüm ile aradaki farklılıkları bilmek, nereden nereye gelindiği ve nelerin düzeltildiğini öğrenmek, beklentilerin karşılanıp karşılanmadığına bakmak oldukça önemli. Bende buna istinaden yazımızın ilerleyen kısımlarında, **Ado.Net Entity Framework 4.0** ile bir önceki versiyonu arasında, sorgu teknikleri açısından oluşan farkları aktarmaya çalışıyor olacağım. **Ado.Net** takımının **4.0** sürümündeki hedeflerinden biriside SQL sorgularının daha anlaşılır olmasını sağlamak. Buna ek olarak gereksiz **SQL** ifadelerinin kullanılmasından uzaklaşılmaya çalışılmış ve hatta önceki **Entity Framework** sürümünde **SQL** sorgusuna dönüştürülemeyen **LINQ** ifadeleri kullanılabilir hale getirilmiş. Tabi daha da fazla fark bulunabilir. Ancak **MSDN** ve **Ado.Net** takımının gerek blog yazılarından gerek Webcast' lerinden takip ettiğim kadarı ile aşağıda yer alan 4 önemli farklılık göze çarpmaktadır. çok doğal olarak söz konusu iyileştirmeler **LINQ** ifadelerinin **SQL** sorgularına dönüştürülmesinde devreye giren motor üzerinde yer almaktadır.

Not : *örneklerimizde yer alan LINQ sorguları .Net Framework 3.5 odaklı olan* ***Visual Studio 2008*** *ve .Net Framework 4.0 odaklı olan* ***Visual Studio 2010 Ultimate Beta***

*2 ürünleri üzerinde yazılmıştır. özellikle Ado.Net Entity Framework 4.0 üzerinden daha nihai bir sürüme ulaşılmadığı için ilereyen zamanlarda değişiklikler gözlemlenebilir. **LINQ** sorgularının icrası sırasında, **SQL Server Profiler** aracı kullanılarak arka planda yürütülen **SQL** sorgularının elde edilmesi sağlanmıştır. Ayrıca örnekte Codeplex üzerinden sunulan Chinook veritabanı kullanılmıştır.*

Vaka 1 : Sorgularda

Açıklama

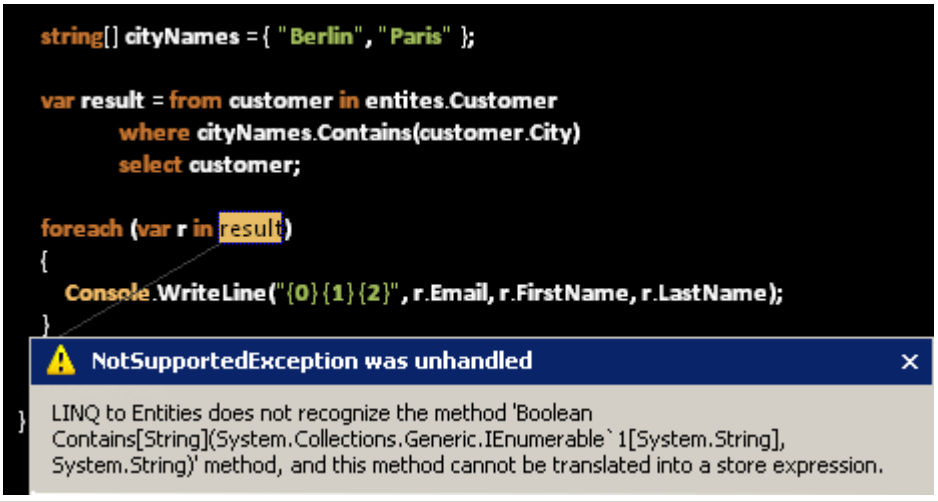
Ado.Net Entity Framework 4.0 öncesindeki sürümde **SQL** tarafında **In** anahtar kelimesine dönü
desteği belirli ölçüde bulunmaktadır. Aşağıdaki **LINQ** sorgusuna göre, Berlin ve Paris şehirlerinc
bir **LINQ** sorgusunda yer alan **Contains** metodunun **SQL** tarafına dönüştürülemediği görülür. Ar

önceki Versiyon - LINQ Sorgusu

```
string[] cityNames = { "Berlin", "Paris" };

var result = from customer in entites.Customer
             where cityNames.Contains(customer.City)
             select customer;

foreach (var r in result)
{
   Console.WriteLine("{0} {1} {2}", r.Email, r.FirstName, r.LastName);
}
```

önceki Versiyon - SQL Sorgusu

Bir **SQL** sorgusu yürütülememektedir nitekim çalışma zamanında aşağıdaki **Exception** mesajı al



4.0 - LINQ Sorgusu

```
string[] cityNames = { "Berlin", "Paris" };

var result = from customer in entites.Customers
             where cityNames.Contains(customer.City)
```

```
                select customer;

            foreach (var r in result)
            {
               Console.WriteLine("{0} {1} {2}", r.Email, r.FirstName, r.LastName);
            }
```

4.0 - SQL Sorgusu

```
SELECT
[Extent1].[CustomerId] AS [CustomerId],
[Extent1].[FirstName] AS [FirstName],
[Extent1].[LastName] AS [LastName],
[Extent1].[Company] AS [Company],
[Extent1].[Address] AS [Address],
[Extent1].[City] AS [City],
[Extent1].[State] AS [State],
[Extent1].[Country] AS [Country],
[Extent1].[PostalCode] AS [PostalCode],
[Extent1].[Phone] AS [Phone],
[Extent1].[Fax] AS [Fax],
[Extent1].[Email] AS [Email],
[Extent1].[SupportRepId] AS [SupportRepId]
FROM [dbo].[Customer] AS [Extent1]
WHERE [Extent1].[City] IN (N'Berlin',N'Paris')
```

Vaka 2 - Gruplamada

Açıklama

Gruplama uygulanmış olan bir **LINQ** sorgusunda **Count genişletme metodunun(Extension Me** dahil edildiği görülür. Ancak **4.0** versiyonunda bu gereksiz durum düzeltilmiştir.

önceki Versiyon - LINQ Sorgusu

```
            var result = from track in entites.Track
                         group track by track.Composer into trackGrp
                         select new
                         {
                            trackGrp.Key,
                            Count = trackGrp.Count(),
                            Sum = trackGrp.Sum<Track>(t => t.UnitPrice),
                            Max = trackGrp.Max<Track>(t => t.UnitPrice),
                            Min = trackGrp.Min<Track>(t => t.UnitPrice)
                         };

            foreach (var r in result)
            {
               Console.WriteLine(r.ToString());
```

```
        }
```

önceki Versiyon - SQL Sorgusu

```
SELECT
1 AS [C1],
[GroupBy1].[K1] AS [Composer],
[GroupBy1].[A1] AS [C2],
[GroupBy1].[A2] AS [C3],
[GroupBy1].[A3] AS [C4],
[GroupBy1].[A4] AS [C5]
FROM ( SELECT
 [Extent1].[Composer] AS [K1],
 COUNT( CAST( 1 AS bit)) AS [A1],
 SUM([Extent1].[UnitPrice]) AS [A2],
 MAX([Extent1].[UnitPrice]) AS [A3],
 MIN([Extent1].[UnitPrice]) AS [A4]
 FROM [dbo].[Track] AS [Extent1]
 GROUP BY [Extent1].[Composer]
)  AS [GroupBy1]
```

4.0 - LINQ Sorgusu

```
        var result = from track in entites.Tracks
                     group track by track.Composer into trackGrp
                     select new
                     {
                        trackGrp.Key,
                        Count = trackGrp.Count(),
                        Sum = trackGrp.Sum<Track>(t => t.UnitPrice),
                        Max = trackGrp.Max<Track>(t => t.UnitPrice),
                        Min = trackGrp.Min<Track>(t => t.UnitPrice)
                     };

        foreach (var r in result)
        {
           Console.WriteLine(r.ToString());
        }
```

4.0 - SQL Sorgusu

```
SELECT
1 AS [C1],
[GroupBy1].[K1] AS [Composer],
[GroupBy1].[A1] AS [C2],
[GroupBy1].[A2] AS [C3],
[GroupBy1].[A3] AS [C4],
[GroupBy1].[A4] AS [C5]
FROM ( SELECT
```

```
[Extent1].[Composer] AS [K1],
COUNT(1) AS [A1],
SUM([Extent1].[UnitPrice]) AS [A2],
MAX([Extent1].[UnitPrice]) AS [A3],
MIN([Extent1].[UnitPrice]) AS [A4]
FROM [dbo].[Track] AS [Extent1]
GROUP BY [Extent1].[Composer]
)  AS [GroupBy1]
```

Vaka 3 - Join sorgularında

Açıklama

LINQ tarafında icra edilen aşağıdaki gibi bir **Join** sorgusunda, bir önceki versiyonda üretilen **SQ** görülmektedir. **4.0**sürümünde ise gereksiz olan bu kontrol **SQL** tarafına aktarılmamaktadır.

önceki Versiyon - LINQ Sorgusu

```
var result = from artist in entites.Artist
             join album in entites.Album
             on artist.ArtistId equals album.Artist.ArtistId
             select new
             {
                ArtistName = artist.Name,
                AlbumTitle = album.Title
             };

foreach (var r in result)
{
   Console.WriteLine(r.ToString());
}
```

önceki Versiyon - SQL Sorgusu

```
SELECT
1 AS [C1],
[Extent1].[Name] AS [Name],
[Extent2].[Title] AS [Title]
FROM  [dbo].[Artist] AS [Extent1]
INNER JOIN [dbo].[Album] AS [Extent2] ON ([Extent1].[ArtistId] = [Extent2].[ArtistId]) OR (
```

4.0 - LINQ Sorgusu

```
var result = from artist in entites.Artists
             join album in entites.Albums
             //on artist.ArtistId equals album.Artist.ArtistId
             on artist.ArtistId equals album.ArtistId //(Zaten yeni sürümde yandaki gibi
             select new
             {
                ArtistName = artist.Name,
                AlbumTitle = album.Title
```

```
                };

            foreach (var r in result)
            {
               Console.WriteLine(r.ToString());
            }
```

4.0 - SQL Sorgusu

```
SELECT
[Extent1].[ArtistId] AS [ArtistId],
[Extent1].[Name] AS [Name],
[Extent2].[Title] AS [Title]
FROM  [dbo].[Artist] AS [Extent1]
INNER JOIN [dbo].[Album] AS [Extent2] ON [Extent1].[ArtistId] = [Extent2].[ArtistId]
```

Vaka 4 - Skip, Take gibi Metod Kullanımlarında

Açıklama

Aşağıdaki **LINQ** sorgusuna göre artistlerin tersten sıralanan adlar listesi ilk 10 satır atlanarak eld
halinde**SQL** tarafında oluşturulan Sub **SELECT** sorgusunda aslında **LINQ** sorgusuna dahil edilm
alanların **SELECT**sorgusuna alınmasının önüne geçilmiştir.

önceki Versiyon - LINQ Sorgusu

```
            var result = (from artist in entites.Artist
                          orderby artist.Name descending
                          select artist.Name).Skip(10);

            foreach (var r in result)
            {
               Console.WriteLine(r);
            }
```

önceki Versiyon - SQL Sorgusu

```
SELECT
[Extent1].[Name] AS [Name]
FROM ( SELECT [Extent1].[ArtistId] AS [ArtistId], [Extent1].[Name] AS [Name], row_nu
 FROM [dbo].[Artist] AS [Extent1]
)  AS [Extent1]
WHERE [Extent1].[row_number] > 10
ORDER BY [Extent1].[Name] DESC
```

4.0 - LINQ Sorgusu

```
            var result = (from artist in entites.Artists
                          orderby artist.Name descending
                          select artist.Name).Skip(10);

            foreach (var r in result)
```

```
        {
            Console.WriteLine(r);
        }
```

4.0 - SQL Sorgusu

```
SELECT
[Extent1].[Name] AS [Name]
FROM ( SELECT [Extent1].[Name] AS [Name], row_number() OVER (ORDER BY [Exten
 FROM [dbo].[Artist] AS [Extent1]
)  AS [Extent1]
WHERE [Extent1].[row_number] > 10
ORDER BY [Extent1].[Name] DESC
```

Görüldüğü üzere **Ado.Net Entity Framework 4.0** sürümünde özellikle **SQL** sorgularına dönüştürme işlemlerinde, motor üzerinde bazı yenilemelerin yapıldığı ve gereksiz işlemlerin önüne geçilmeye çalışıldığı gözlemlenebilmektedir. Açıkçası bu konu ile ilişkili olaraktan çıkacak kitapları merakla bekliyorum. özellikle **Julia Lerman'** ın daha önceki **Programming Entity Framework** kitabını okuyanlar eminim **4.0** için olan versiyonuda sabırsızlıkla ve merakla bekliyordur. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Differences.rar (183,69 kb) **[örnek Visual Studio 2010 Ultimate Beta 2 üzerinde geliştirilmiştir. Yarın public olarak yayınlanacak RC sürümü için test edilmemiştir]**

Yazılımcı Psikolojisi (2010-02-05T14:10:00)

Merhaba Arkadaşlar,

Beni tanıyanlar bloğumdan teknik konular dışında pek paylaşımda bulunmadığımı çok iyi bilirler. Ancak kalıpları biraz olsun esnetmekten kime zarar gelir ki? Belki hepimiz için faydası bile olabilir. Bu yazımda bahsedeceğim konuya başlamadan önce tüm psikologlardan affımı istemek zorundayım. Nitekim bir yazılımcının olağan psikolojisi üzerine gözlemlerimi aktarırken bilimsel hiç bir kurama uyamayacağımdan veya sonuçlara varamayacağımdan oldukça eminim. Aslında bu gibi konularda işi uzmanlarına bırakmak gerekiyor. Ben de aslında onbir seneyi bulan profesyonel iş ve yazılım hayatımdaki gözlemlerimi aktarmaya çalışacağım. İnanıyorum ki yazıyı okuyan her meslektaşım kendisine bir pay çıkartacak ve en azından bir kaç dakika da olsa düşünecektir. Aslında

anlatılanların gerçek hayat hikayeleri ile birleştirilmesi sonrasında dramatik bir durum mu oluşmaktadır yoksa traji komik bir dünya mı söz konusudur siz karar vereceksiniz.

öncelikle yandaki resmin size ne çağırıştırdığını düşünerek başlayalım. Bu resimde birden fazla kişinin olduğunu ama her birinin kendi cam fanusu içerisinde ayakta kaldığını görüyoruz. Ortada silik bir kişilik var ama onun konumuzla ilgisi bulunmamakta. Aslında çalışma ortamlarımızdaki oturma düzenlerimize, masamızın üstündeki malzemelere baktığımızda kendimize ait bir dünya kurduğumuzu hatta kimilerinin deyimiyle bir yaşam alanı oluşturduğumuzu kolayca fark edebiliriz.

Bu yaşam alanının belkide en önemli parçalarından birisi ses kalitesi yüksek olan kulaklıklardır. Şimdi oturduğunuz sandalyeden ayağa kalkın ve çevrenizde kulaklık ile müzik dinleyip bilgisayar ekranına bakan arkadaşlarınızı tespit etmeye çalışın. Sonra da şunu düşünün. Doğa da, seyahatlerde, deniz kenarında, parkta, bahçede, yürüyüşte, sağlık için yapılan koşuda, çevreye bakarken size çok iyi gelen, içinizi rahatlatan veya hissetmek istediğiniz duygularınızı daha çok ortaya çıkaran müzikler nasıl oluyor da 15.4 inch' lik bir dikdörgene bakarken size bir şeyler hissettirebiliyor. Gerçekte müziği mi dinliyorsunuz yoksa yaptığınız işe mi odaklanıyorsunuz.

Yaşananlardan : *çok eskiden çalıştığım bir yazılım şirketinde dış kaynak elemanı olarak bir projede görevlendirilmiştim. Yazılımı geliştirdiğimiz şirkette bize ayrılan yer İstanbul' un en güzel boğaz manzarasına sahip kesimlerinden birisiydi. çalışma ortamının kurulduğu binanın etrafında kocaman ve yemyeşil bir arazi yer alıyordu. Bu nedenle son derece sessiz ve işinize en yüksek seviyede konsantre olabileceğiniz bir ortam söz konusuydu. Ancak her yazılımcı gibi orada bile, kulaklıkla müzik dinlemeyi başarırdık. Arkadaşlarımızdan birisi ise dışarıdan çalışmasına etki eden en ufak sesten rahatsız olurdu. çoğu zaman kulaklıklarımdan dışarı çıkan ses nedeniyle ya MSN üzerinden, ya da mail araclığıyla beni uyarırdı. Ama hiç bir zaman arkadaşım yerinden kalkarak yanıma gelip bu uyarıyı yapmamıştı. İki masa arasında 3 metreyi bulmayan mesafeyi kat etmek yerine, online ortamdan haber uçururdu. İronik diyebilir miyiz?*

 çalışma ortamlarımızın oluşturulmasında şirketlerin büyük payı var bildiğiniz üzere. Günümüz yazılım projeleri ve ihtiyaçları düşünüldüğünde en popüler oturma düzenlerinden birisi yonca şeklinde olanlar. Ancak bazı şirketlerde sizi oturduğunuz masanın içerisine neredeyse hapsedecek şekilde düzenlerde söz konusu. örneğin sadece sağınızı ve solunuzu görebildiğiniz ama ön masanızda oturan şahsı göremediğiniz bir oturma düzeni de mevcut. *(Daha kötüleri de yok mu? Var tabiki de)* Tabi siz de ara sıra sağınızdaki veya solunuzdaki arkadaşınızla sohbet etmek, konuşmak ihtiyacı hissedebilirsiniz. Bu aslında son derece doğal bir insan tepkisidir. Ancak ne zamaki iletişiminiz sadece sabah **"Günaydın"** ve akşam **"İyi Akşamlar"** şeklinde olan üç basit ve çoğunlukla size söylemesi dahi zor gelen kelimelere düşer, o zaman bir şeylerin ters gittiğini kabul etmeli ve tedbirlerini almalısınız demektir. çünkü gittikçe ofis masanızın belirlediği sınırlar içerisinde yaşayan ve o dünyaya inanan birisi olmaya başlamışsınızdır. Aslında o dünya değildir. Size göre koca bir evrendir.

Buna itiraz edebilirsiniz ve hayır ben böyle bir insan değilim diyebilirsiniz. O zaman size bu durumu doğrulatacak bir deney önerebilirim. Portatif Web kameranızı alın ve masanın uygun bir yerine monte ederek sizin ofisteki bir günlük yaşantınızı çekmesini sağlayın. Sonrasında izleyin. çok şaşırabilirsiniz. özellikle çok vahim vakalarda, kaydı hızlı olarak ileri sardığınızda uzun aralıklarda aynı şekilde durduğunuzu fark edebilirsiniz. özellikle gözlerinizi bir noktaya uzun süre sabitleyebildiğinizi, göz doktorlarının sık sık vurguladığı ve göz kuruluğuna sebebiyet verdiği için göz yaşı damlası kullanmanıza neden olan duruma sık sık düştüğünüzü net bir şekilde izleyebilirsiniz. Eğer bu noktaya geldiyseniz bir de şunu deneyin. Bilgisayarınızı kapatın ve siyah ekranına sadece 15 dakika boyunca bakmaya çalışın. Sıkıldınız mı? Allah allah...Neden acaba? 🙂

Yaşananlardan : *Bulunduğum yazılım şirketinde pek çok arkadaşımız ne yazık ki bu şekilde yaşıyor. Günün normal olarak 8 saatlik mesai dilimi içerisinde zorunlu olmadıkça kimseyle konuşmayan, çoğunlukla kendi zevkine uygun müziği dinleyen, üzerine yıkılan onlarca iş içerisinde sadece nefes alıp vermekten sorumlu olduğunu sanan ve sanki bir mahkumiyeti acı çekmeden yaşamayan çalışan insanlar görüyorum. Hatta yazımı düzenlediğim şu sırada bile halen görmekteyim.*

Basit bir matematik hesabı yaptığımızda durumun vahimliği bir kere daha ortaya çıkıyor. çevrenizdekiler ile günde sadece 30 kelime konuştuğunuzu düşünün. Haftada 5 gün

çalıştığınızı...Ayda 20 gün...Senede 240 gün...240X30=7200 kelime. Bu sadece senelik olan çok kaba ve kötü bir tahmin. Peki ya ayda kaç satır kod yazıyorsunuz ve tahminen kaç kelime kod yazıyorsunuz. Bu şu anlama gelebilir. çevrenizdekiler ile konuşmaktan çok kelime harcayarak kodlama yapıyorsunuz. Bu çok doğal olarak işiniz bir parçası. Ama artık sizin sadece bilgisayarınız ile konuşmayı tercih ettiğinizin de bir göstergesi. Peki bu disiplini ne kadar devam ettirebilirsiniz. Aslında sizce bu bir disiplin olabilir mi?

Yaşananlardan: *Eski şirketimdeki patronum yazılımın bir yaşam tarzı olduğunu belirtmişti. Ancak bu yaşam tarzını devam ettirirken insan olmanın doğasında var olan bazı aktivitelerden de uzaklaşmamak gerektiğini sürekli ifade ederdi. Hatta bir keresinde başarılı yöneticilerin iş yaşantıları dışında spor, müzik gibi alanlarda amatör olarak bile olsa ilgilendiklerini söylemişti. İlgi derken fiili olarak katılmaktan bahsediyordu. örneğin toplanıp basketbol oynamak, masa tenisi müsabakalarına katılmak, gitar çalmak, flüt üflemek, saksafon çalmak vb...Şu anda çalıştığım şirkette bu profile uyan bir çok arkadaşım var. Tabiri yerinde ise tüm iş streslerini spor veya müzik ile uğraşarak atabiliyorlar. üstelik bu arkadaşlarımın bazıları(tamamı değil) gerçekten göz göze sosyalleşmeye inanan, sizinle konuşmaktan zevk alabilen ve sürekli masasında oturmayı sevmeyen kimseler. Tabi patronum bu tavsiyede bulunduğunda sene 1999 du ve ortalarda sosyal ağlar(Facebook, Twitter vb... gibi) diye bir gelişim bulunmamaktaydı.*

Sosyal ağlar demişken onlara da dokundurmadan geçmemek lazım. üniversite yıllarındayken ICQ, MIRC gibi programlar çok popüler idi. Hatta bir arkadaşımla BBS' ler üzerinden mesajlaştığımızı bile hatırlıyorum. Acaba kimse bu günlere gelebileceğimizi düşünüyor muydu? Artık Facebook, Twitter gibi sosyal ağlar söz konusu. Bir fayda olarak uzun zamandır görüşemediğiniz, belirli sebeplerden iletişiminizi kaybettiğiniz arkadaşlarınızı bulmak açısından oldukça etkili. Ya da ülkeler arası kendi mesleki gelişiminizle ilişkili olarak bir ağın parçası olmak istediğinizde son derece faydalı. Ancak şu durumu kim açıklayabilir; iki yan masanızdaki arkadaşınızın doğum gününü Facebook üzerinden kutlamak. 😕 Bunu ben dahil pek çoğumuz yapıyor. Şimdi yakın zamanda etrafınızda doğum günü olan ve Facebook listenizde yer alan bir arkadaşınızı düşünün. Eğer onunla küs değilseniz doğum gününde şunu yapın; oturduğunuz yerden kalkın, arkadaşınızın yanına yüzünüzde kocaman bir tebessüm ile gidin, elinizi uzatın tokalaşın ve yanaklarından öperek nice senelere demeyi bir deneyin. Bunu yapabilir misiniz? Yapın ve sonrasında nasıl hissettiğinizi düşünün. Yoksa sandalyenizde oturmayı mı tercih edersiniz.

Doğruyu söylemek gerekirse yazılımcıların dünyası 14.1 inch, bilemediniz 15.4 inch, en kötü ihtimalle 17 inch lik bir dünyadan ibaret olabiliyor. Konuşurken zar zor söyleyebileceğimiz, çekindiğimiz, korktuğumuz cümleleri dijital ortamda rahatlıkla ifade edebiliyoruz. Açıkçası klavyenin başındayken çok güçlüyüz. Ekranımızda istediğimiz gibi bir evren kurabilir galaksinin istediğimiz noktasına sıçrayabiliriz. Tabi yazının bu noktasında oluşan kişilik profiline yazılımcılar dışında pek çok insan uyabilir. Eminim ki çevrenizde gününü bilgisayar başında ve ağırlıklı olarak sosyal ağlar üzerinde geçiren pek çok genç vardır*(Sizinde bunun için günde 8 saate yakın mesai harcayan kuzeniniz var mı?)*. Onları biraz gözlemleyin ve bazı çıkarımlar elde etmeye çalışın.

Yazılımcı psikolojisi ile ilişkili analizlerimi aktarmaya başlarken önce bulunduğumuz masadan başladık ve sosyal ağlara kadar uzandık. Yazılımcının kod tarafında döktürebilirken konuşma güçlüğü çektiğini, gözlerinde sürekli bir problem olduğunu fark ettik. Tabi bilgisayarı başında sakin sakin iş yapan bir yazılımcının çoğu zaman çileden çıkabileceğini ve olur olmadık şeylere kafayı takıp son derece sinir birisi olabileceğini de unutmamak gerekir. Aslında bir psikolog olsam bu durumdaki çoğu yazılımcıya "Agresif Kişilik Bölünmesi" teşhisi koymak isterim ki böyle bir teşhis olmadığından eminim. Yandaki resimde sinir bir yazılımcıyı motiflemeye çalıştım ama bulabildiğim tek resim bu oldu 😕

Yaşananlardan : *Siz hiç programdaki kodlaması sonrasında çalışma zamanında istisna(Exception) alan yazılımcı bir arkadaşınızın masasındaki bazı eşyaları sağa sola fırlattığını gördünüz mü? Daha önceden çalıştığım şirketlerden birisinde yine dış kaynak elemanı olarak bir projeye atanmıştım. O gün farkettiğim tek gerçek, çalışma arkadaşımın ekrana son derece yakın ve sinirli bir şekilde bakmasıydı. Sonunda olan oldu ve kodun çalışması sırasında bir istisna mesajı aldı. "Hay ben senin gibi kodun bippppp!!!" diyerek devam cümlesi bittiğinde göz göze geldik ve bende son derece yanlış bir şekilde "Ne o hocam kod mu patladı?" dedim...özellikle genç yazılımcı arkadaşları bu konuda uyarmak isterim. Sakın kodunda exception alıp az önce bahsettiğim çılgın psikoloji içerisine giren bir arkadaşınıza bu şekilde yaklaşmayın. Hatta mümkünse kafanızı önünüze eğin ve kendi işinize bakın. Sonuçta olan oldu ve arkadaşım masasında duran 600 sayfalık C# kitabını alıp masama doğru fırlattı. Barmenlerin yaptığı gibi. Sonrasında da şunu söyledi "Al sen yaz o zaman..."*

Aslında bu hikaye neredeyse bir yazılımcının hayatının pamuk ipliğine bağlı olabileceğini ifade etmektedir. Yazılımcının bir türlü dizginleyemediği kodunda meydana gelen istisnanın bedeli aslında kendi hayatında kontrol edemediği bir dizi olayın başlamasına

neden olur. çevrenize bakın ve böyle arkadaşlarınız var ise dikkatlice gözlemleyin. Yüzlerinden, üzerlerindeki negatif enerjiyi çok rahat bir şekilde görebilirsiniz. Oysaki yazılımcılar kod tarafında her zaman kral değil midir? En azından kendimizi hep bu şekilde görmez miyiz? Oysaki hayatımızı normal olarak devam ettirmemiz için gerekli şartları bile sağlamaktan, kontrol etmekten çoğu zaman aciz olabiliriz.

Bu noktada yazıyı artık sonlandırma ve bir sonuca bağlamak istiyorum ancak bu çok zor. Belkide sonuçları çıkartması gereken kişi siz değerli okurlarımdır. Yine de değişmez bazı gerçekler olduğunu kabul etmeliyiz. Yazılımcının bulunduğu ortamın koşulları, kendi iç dünyasında ortaya çıkartamadığı duyguları ve başka pek çok faktörün gün içerisinde kendi davranışları üzerinde pek çok etkisi olduğu yadsınamaz bir gerçek. Şimdi birde ekibindeki yazılımcıları yönetmeye çalışan ve bu noktada onların psikolojisinin çok önemli olduğunun bilincine varan teknik liderleri, proje yöneticilerini, müdürleri düşünün. İnanın işleri bizimkinden çok ama çok daha zor. Bence hepimizin eğitim hayatında insan psikolojisi üzerine gerçek hayat pratiklerinin aktarıldığı dersler olmalı.

Yaşananlardan : *üniversiteden mezun olduğumda o dönem popüler olan MBA yüksek lisnans programına katılmıştım. İK dersine giren hocamız çok değerli birisiydi. Pek fazla kitap kullanmazdı. Zaten ağzından çıkan cümleler düzgün, akademik ve tecrübe dolu olurdu. Açıkçası o cümleleri not ettiğinizde her dönem için elinizde bir kitap zaten yazılırdı. Bir gün dersimizde şunları söyledi ve bu sözler aklımda hiç çıkmadı;*
"Arkadaşlar...Konuşun...Konuşmaktan, insanlarla iletişim kurmaktan çekinmeyin. Fikirlerinizi düşüncelerinizi elinizden geldiğince düzgün ifadeler ile agresifleşmeden karşınızdakine aktarın. Dinlemesini bilin. Ve tabi en önemlisi; Güler yüzlü olmaktan, mevzularınızı tebessümle aktarıp dinlemekten çekinmeyin. Bunu öğrenin..." Aşağı yukarı bu şekilde bir konuşmaydı. Ana fikri/fikirleri görebildiniz mi? Belki biz yazılımcıları, işletme gibi alanlarda ilerleyen kişilerden ayırabilecek noktalar yer almakta. Güler yüzlü olmak, konuşlanlık, sevecenlik, sabırlılık, agresif olmamak vs...

Bir daha bu şekilde teknik dışı bir yazı yazar mıyım bilemiyorum ama bu tespitlerin sizi bir kaç dakikalığına düşündüreceğine inanıyorum. Hepimiz insanız ve bulunduğumuz Dünya' nın yaşayan birer parçasıyız. Hayatımız uzun görünse dahi çok kısa ve aslında her birimiz evrende bir toz parçasının milyarda birinden daha küçük olan parçaların yaşam süresinden bile kısa yaşıyoruz.

Bu yazıyı başarılı bir şekilde okudunuz. Şimdi sağınızdaki veya solunuzdaki, önünüzdeki veya arkanizdaki bir arkadaşınıza bakın ve tebessüm ederek gülümseyin, bir şeyler söyleyin. En azından halini hatırını sorun yahu!

WCF WebHttp Services - Client Tarafını Geliştirmek (2010-02-05T09:45:00)

wcf,wcf 4.0,webhttp services,restful,non soap,wcf webhttp services,

 Merhaba Arkadaşlar,

Sanırım pek çoğumuz piyangodan veya diğer şans oyunlarından kendilerine tonlarca para çıksa ne yapacağını düşünmüş veya hayal etmiştir. Açıkası kendi adıma hayat etmediğimi dile getirsem yalan söylemiş olurum. Ancak ben pek çoğumuz gibi yan yana bir kaç Ferrari' yi dizmektense bir kaç yere yatırım yapmayı hayal etmişimdir hep. örneğin dünyanın sayılı bir kaç futbol kulübünün*(Barcelona, Manchester United vb...)* ve yazılım şirketinin*(Microsoft, IBM vb...)* hisselerinden satın alır ve şöyle güzel bir fon sepeti oluştururum. Neyse...Sözü niye piyangodan açtığımıza gelince...

Hatırlayacağınız üzere bir önceki yazımızda **WCF WebHttp Service'** leri ile tanımaya çalışmış ve konuyu pekiştirmek amacıyla basit bir Merhaba Dünya uygulaması geliştirmiştik. Tabi bu örneğimizde **HTTP** protokolünün yanlızca **Get** metodunu kullanmıştık. Dolayısıyla operasyonlarımızda sadece **WebGet**niteliklerinin uygulandığına şahit olduk. Ancak **HTTP** protokolüne göre **Get** dışında **Post, Put** ve **Delete** metodlarını da kullanabileceğimizi biliyoruz. Dikkat çekici bir diğer noktada örneğimizde **Get** metoduna göre talepte bulunurken basit bir tarayıcı uygulamadan faydalanmış olmamızdı. Oysaki kendi istemci uygulamamızı yazmak isteyebiliriz. Bu durumda istemci tarafından **HTTP** protokolünün **Get, Post, Put** ve **Delete** metodlarına uygun talepleri nasıl gerçekleştirebiliriz? Aslında olay servis tarafının istediği mesaj paketlerini istemci tarafında oluşturup göndermekten başka bir şey değildir. Yani **talebin(Request)** içeriğini hazırlamak ve **dönen cevabı(Response)** değerlendirmek.

İşte bu yazımızda söz konusu durumları ele alaraktan hem **Post, Put, Delete** metodlarının kullanımına bir örnek verecek hemde istemci tarafını geliştirmeye çalışacağız. Tabi öncesinde servis tarafını hazırlamamız gerekiyor. Bu örneğimizde herhangibir işe yaramasada konuyu anlamamızı kolaylaştıracak bir senaryomuz da olacak. Senaryomuza göre bir **Piyango**servisi tasarlayacağız. Bu servis, istemcilerin yeni bir piyango bileti üretebilmesine, var olan piyango biletlerini çekebilmelerine, isterlerse biletlerini silmelerine veya güncellemelerine izin veren operasyonlar içerecek. Tabiki bu operasyonlarda **HTTP** protokolünün **Get, Post, Put** ve **Delete** metodları göz önüne alınıyor olacak. Dilerseniz hiç vakit kaybetmeden **WCF REST Service Application**uygulamasını oluşturarak işe başlayalım. Uygulamamızda bilet bilgilerinin saklanması ve depolanması amacıyla basit text dosyasından yararlandığımızı belirtmek isterim. Diğer taraftan bilet bilgileri için **Ticket** isimli yardımcı bir sınıfımızda yer almaktadır.

using System;

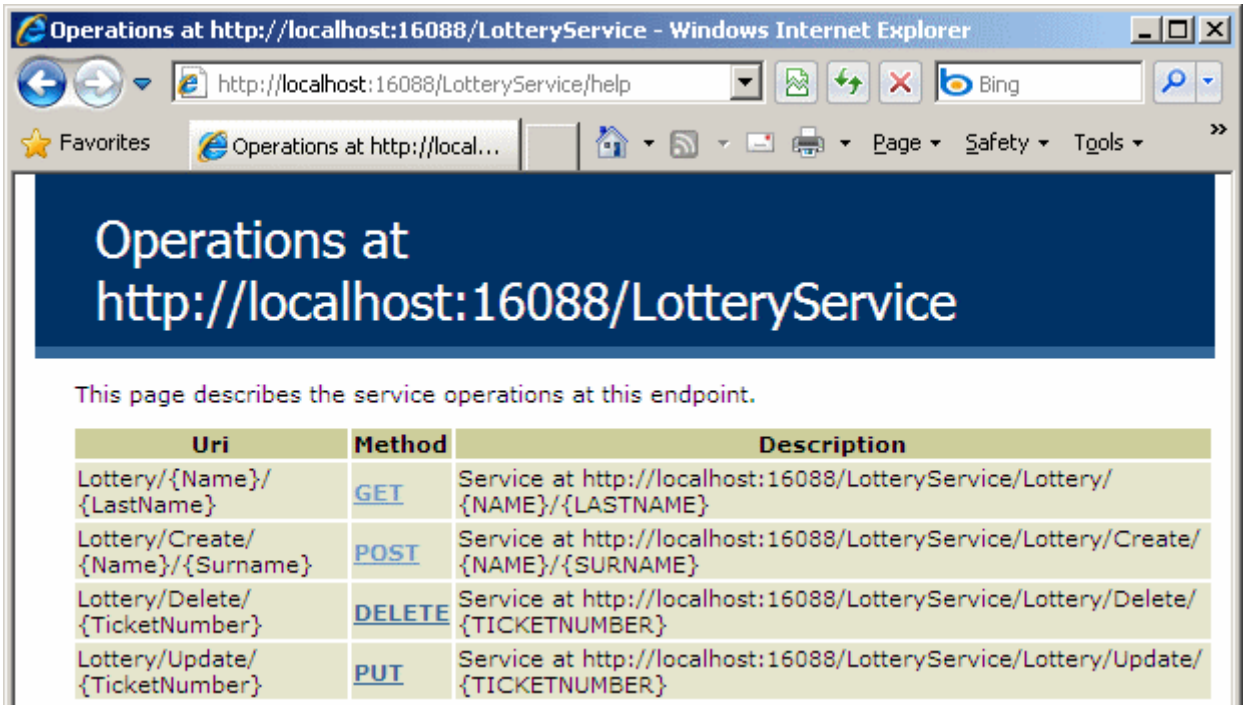
```
namespace Lesson2
{
   public class Ticket
   {
      public string Number { get; set; }
      public string Owner { get; set; }
      public DateTime TicketDate { get; set; }
      public bool NewOrUpdated { get; set; }

      public override string ToString()
      {
         return String.Format("{0}|{1}|{2}|Is New? {3}", Number, Owner,
TicketDate.ToString(),NewOrUpdated.ToString());
      }
   }
}
```

LotteryService isimli **WCF WebHttp Service** içeriği ise aşağıda görüldüğü gibidir.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;
using System.Web;

namespace Lesson2
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class LotteryService
   {
      string filePath = HttpContext.Current.Server.MapPath("~\\Tickets.txt");

      [WebGet(UriTemplate = "Lottery/{Name}/{LastName}")]
      public List<string> GetMyTickets(string Name,string LastName)
      {
         return (from line in File.ReadAllLines(filePath)
                 where line.Contains(Name + LastName)
                 select line).ToList();
      }
```

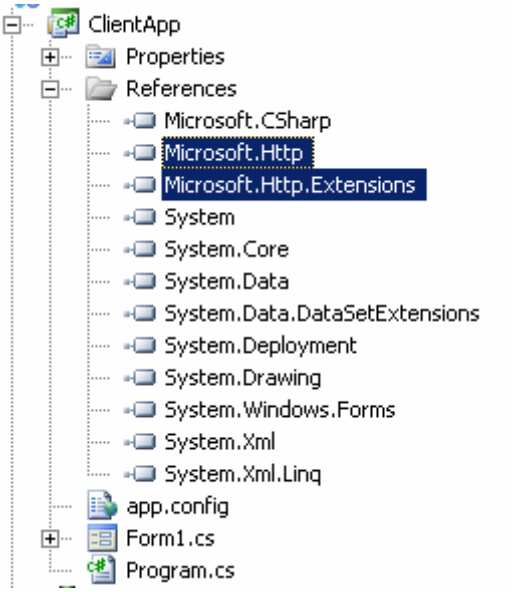
```
    [WebInvoke(UriTemplate = "Lottery/Create/{Name}/{Surname}", Method =
"POST")]
    public Ticket CreateTicket(string Name,string Surname)
    {
      Ticket createdTicket=new Ticket
      {
        Number = Guid.NewGuid().ToString(),
        Owner = String.Format("{0}{1}",Name,Surname),
        TicketDate = DateTime.Now
      };

      File.AppendAllLines(filePath,new String[]{createdTicket.ToString()});
      return createdTicket;
    }

    [WebInvoke(UriTemplate = "Lottery/Update/{TicketNumber}", Method =
"PUT")]
    public string UpdateMyTicketNumber(string TicketNumber)
    {
      string ticket = (from line in File.ReadAllLines(filePath)
              where line.Contains(TicketNumber)
              select line).First();
      string[] infos=ticket.Split('|');

      string
updatedTicket=string.Join("|",Guid.NewGuid().ToString(),infos[1],infos[2],"Is New?
",true.ToString());

      File.AppendAllLines(filePath, new string[]{updatedTicket});

      return updatedTicket;
    }

    [WebInvoke(UriTemplate = "Lottery/Delete/{TicketNumber}", Method =
"DELETE")]
    public void DeleteMyTicket(string TicketNumber)
    {
      string[] newLines = (from line in File.ReadAllLines(filePath)
              where !line.Contains(TicketNumber)
              select line).ToArray();

      File.WriteAllLines(filePath, newLines);
    }
  }
}
```

Kod parçamızda **HTTP Get,Post,Put** ve **Delete** metodlarının kullanımlarına örnek olması açısından çeşitli servis operasyonlarının yer aldığı görülmektedir. Kritik olan noktalar **WebGet** ve **WebInvoke** niteliklerinin nasıl kullanıldığıdır. Servisimizin yardım sayfasına bakıldığında, istemci tarafında oluşturulması gereken **Request** paketlerinin nasıl olacağıda kolaylıkla görülebilir. Tabi **Post** ve **Put** metodlarında bir **Request Body** kullanılmamıştır. Bir başka deyişle **Put** ve **Post** işlemleri için gerekli bilgiler servis tarafına **URL** satırından gönderilmektedir.

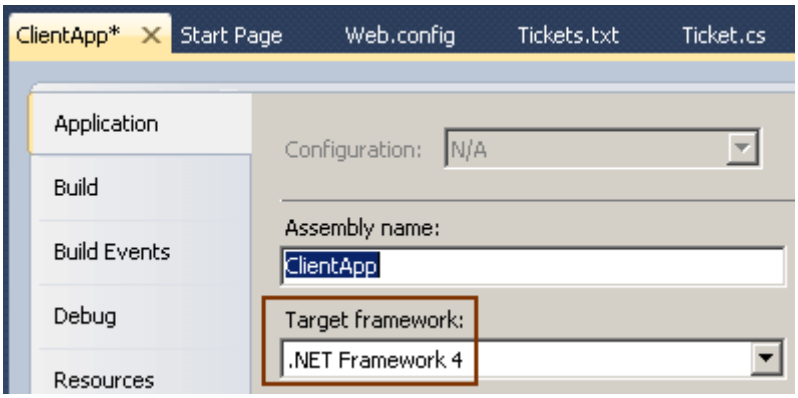


Gelelim istemci tarafına.

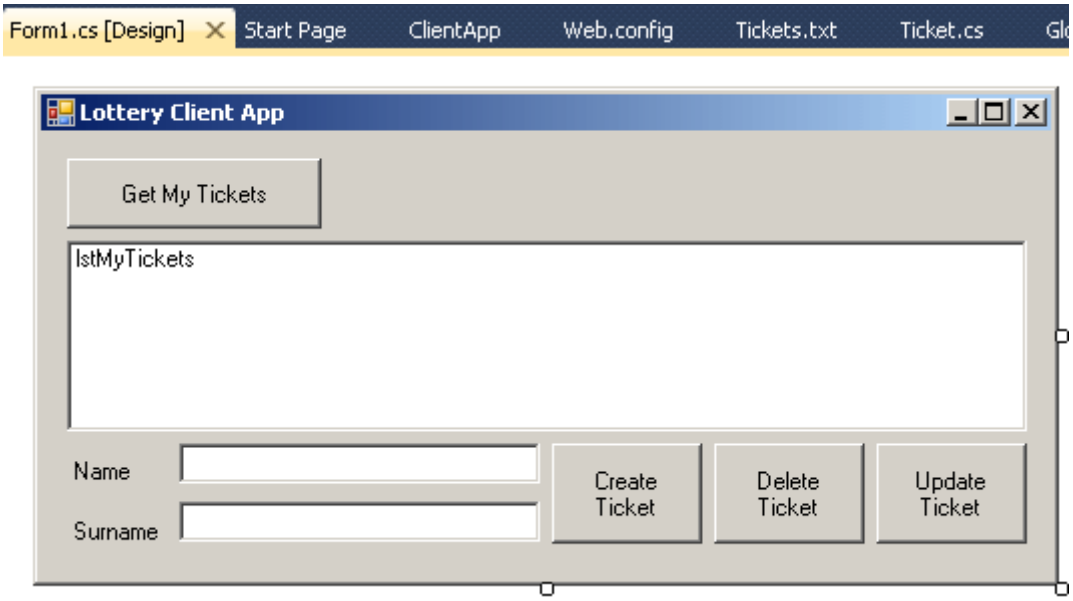
İstemciyi basit bir **WinForms** uygulaması olarak tasarlayacağız. önemli olan nokta ise, az önce tasarlanan **WCF WebHttp Service**' ini nasıl kullanabileceğimiz. Sonuçta **HTTP Get, Post, Put** ve**Delete** metodlarının istemci tarafından hazırlanması ve gönderilmesi gerekmekte. üstelik servise ait bir **WSDL** içeriği ve dolayısıyla **Proxy** üretimi de söz konusu değil. Bu noktada**WebChannelFactory, HttpWebRequest** ve **WebClient** tiplerinden yararlanabileceğimizi biliyoruz. Ne varki WCF Rest Starter Kit Preview 2 ile birlikte gelen **HttpClient** sınıfı tamda bu tip servislerin tüketilmesi için geliştirilmiş durumda. Elbette bu kit içeriğinin, **.Net Framework 4.0**' ın final sürümü ile birlikte içeriye doğrudan dahil edileceğini tahmin etmekteyiz. Şimdilik **Starter Kit** ile gelen tipi kullanacağız. Bu sebepten **Windows** uygulamamıza gerekli referansları aşağıdaki şekildende görüleceği üzere eklememiz gerekiyor.



Dikkat edilmesi gereken noktalardan biriside istemci uygulamanın hedeflediği **Framework** profilidir. Söz konusu **Starter Kit** referansları ile çalışabilmek için istemci tarafının hedef profilinin **.Net Framework 4.0 Client Profile** değil*(ki varsayılanı budur)* **.Net Framework 4.0** olması gerekmektedir.



Artık istemci için gerekli tüm ön hazırlıklar yapılmıştır. Şimdi dilerseniz servis fonksiyonelliklerini icra edebilmek amacıyla **Form** içeriğini aşağıdaki gibi düzenleyelim.

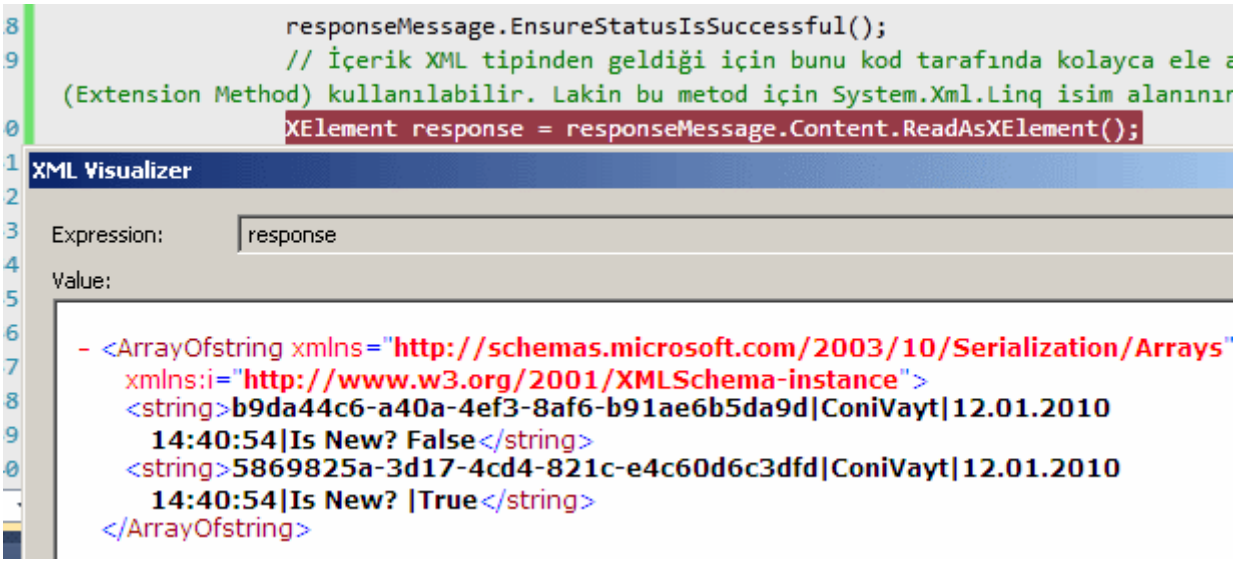


Form üzerindeki kontrolleri kullanarak bilet üretebilecek, bir bileti silip güncelleyebilecek yada var olan biletlerimizi görebileceğiz. Tabiki tüm bu fonksiyonellikler **LotteryService** isimli **WCF HttpWeb Service** üzerinden gerçekleştiriliyor olacak. İlk olarak bir kişinin sahip olduğu tüm biletleri listlemeye çalışalım. Bu noktada servis tarafındaki **GetMyTickets** operasyonu için bir çağrı yapılması gerekiyor. Söz konusu çağrı örneğin **http://localhost:16088/LotteryService/Lottery/Coni/Vayt** şeklinde olabilir. Nitekim **WebGet** niteliğinde belirtilen **URI** bilgisi bu şekildedir. Bu durumda istemci tarafında aşağıdaki kodlamayı yapabiliriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;
using System.Xml.Linq;
using Microsoft.Http;

namespace ClientApp
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void btnGetMyTickets_Click(object sender, EventArgs e)
      {
         // REST Starter Kit Preview 2' den gelen HttpClient tipi oluşturulur.
```

```
            // Parametre olarak WCF WebHttp Service' in base address bilgisi verilir.
            using (HttpClient client = new
HttpClient("http://localhost:16088/LotteryService/"))
            {
                // Get talebi için gerekli URI bilgisi oluşturulur
                string requestUri = String.Format("Lottery/{0}/{1}", txtName.Text,
txtSurname.Text);
                // Get metodu ile ilgili talep oluşturulur.
                HttpResponseMessage responseMessage=client.Get(requestUri);
                // Operasyonun başarılı olmaması halinde bir exception fırlatılması isteniyorsa
EnsureStatusIsSuccessful metodu kullanılmalıdır.
                responseMessage.EnsureStatusIsSuccessful();
                // İçerik XML tipinden geldiği için bunu kod tarafında kolayca ele almak adına
ReadAsXElement genişletme metodu(Extension Method) kullanılabilir. Lakin bu metod
için System.Xml.Linq isim alanının referans edilmesi gerekir.
                XElement response = responseMessage.Content.ReadAsXElement();
            }
        }
    }
}
```

Servisin çalıştığını varsaydığımızda yukarıdaki metodun icra edilmesi sonucu **debug** zamanında aşağıdaki sonuçlara ulaştığımızı görebiliriz. *(Tickets.txt dosyasında bazı ticket bilgileri olduğunu ve Name için Coni, Surname için Vayt bilgilerinin girildiğini varsayıyoruz)*

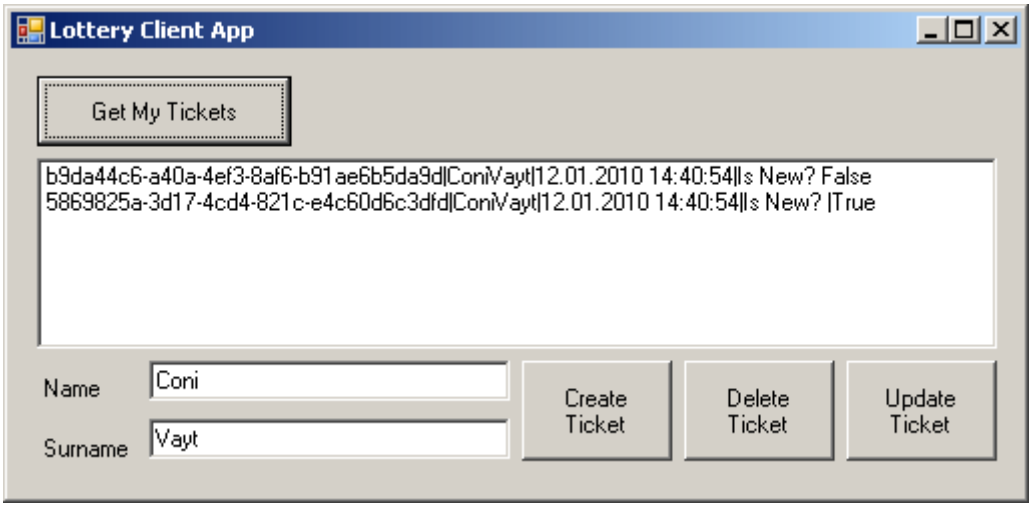


Dikkat edileceği üzere servis tarafındaki operasyonda **List<string>** tipinden olan operasyon dönüş tipi istemci tarafına, **ArrayOfstring** ve **string** isimli alt elementlerden oluşan bir **XML** içeriği olarak aktarılmıştır. Elbette **XElement** içeriğinin bu şekilde elde

edilebiliyor olması yeterli değildir. Bu içeriğin **ListBox** kontrolü içerisine serpiştirilmesini de bekliyoruz. Dolayısıyla aşağıdaki kod ilavesini de yapmamız gerekiyor.

```
XElement response = responseMessage.Content.ReadAsXElement();

lstMyTickets.Items.Clear();
var tickets = from node in response.Elements()
              select node.Value;

foreach (var ticket in tickets)
{
   lstMyTickets.Items.Add(ticket);
}
```

Dikkat edileceği üzere basit bir **XLINQ** sorgusu ile tüm elementlerin **Value** değerleri çekilmiştir. Elbette bu **XLINQ** sorgu ifadesini belirleyen kriter, servis tarafından dönen **XML** içeriğinin şemasıdır. çalışma zamanında örnek bir kullanıcı için **Get** sorgusunu gerçekleştirdiğimizde aşağıdaki ekran görüntüsündekine benzer sonuçları elde ettiğimizi görürüz.

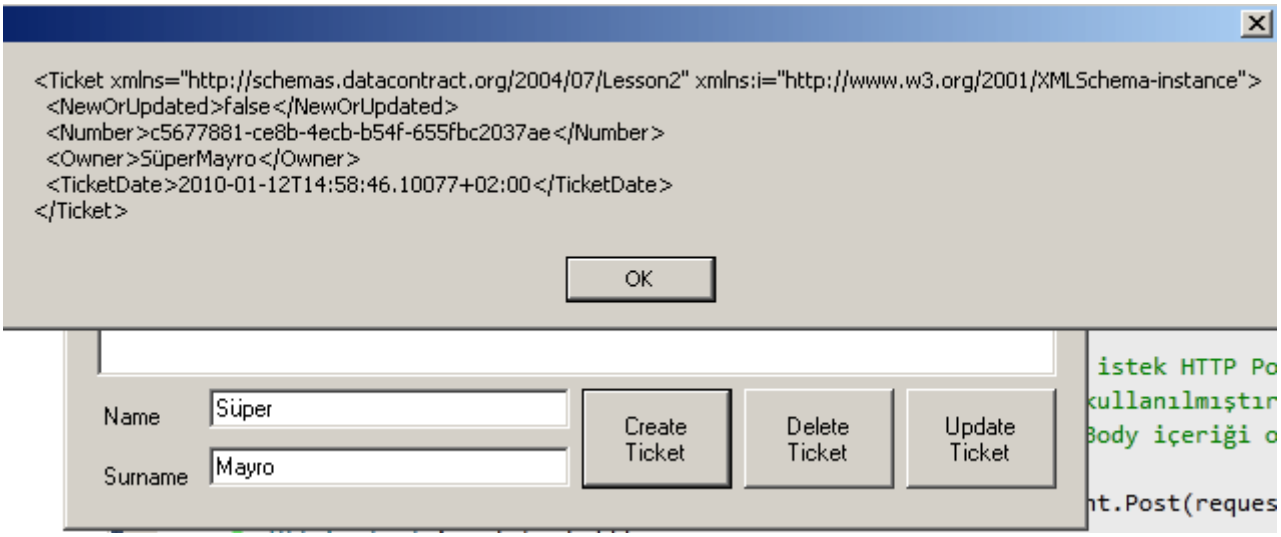


Şimdi yeni bir biletin oluşturulması için gerekli kodları yazalım.

```
private void btnCreateTicket_Click(object sender, EventArgs e)
      {
         using (HttpClient client = new
HttpClient("http://localhost:16088/LotteryService/"))
         {
            string requestUri = String.Format("Lottery/Create/{0}/{1}", txtName.Text,
txtSurname.Text);

            // Yeni bir Ticket oluşturmak için gerekli istek HTTP Post metoduna göre
yapılmaktadır. Bu sebepten HttpClient tipinin Post metodu kullanılmıştır.
```

```
            // Gönderilen talepte herhangibir Request Body içeriği olmadığından HttpContent tipinin CreateEmpty metodu kullanılmıştır.
            HttpResponseMessage responseMessage = client.Post(requestUri, HttpContent.CreateEmpty());
            responseMessage.EnsureStatusIsSuccessful();
            // Oluşturulan yeni Ticket bilgisi istemci tarafına yine bir XML içeriği olarak dönmektedir. üretilen içerik bilgi amaçlı olarak kullanıcıya gösterilir.
            XElement createdTicket = responseMessage.Content.ReadAsXElement();
            MessageBox.Show(createdTicket.ToString());
        }
    }
```

Get kullanımına benzer olmakla birlikte bu kez **HttpClient** tipinin **Post** metodundan yararlanılmaktadır. **Post** ve **Put** gibi metodlarda **Request Body**' sinin olması gerekebilir. Ancak bizim servis operasyonlarımız **Request Body** kullanmamaktadır. Bu nedenle ilgili parametreler **HttpContent.CreateEmpty()** metodu ile geçilmektedir. Bu kod parçasına göre çalışma zamanında bir bilet üretmek istediğimizde geriye aşağıdakine benzer sonuçların aktarıldığını görebiliriz.

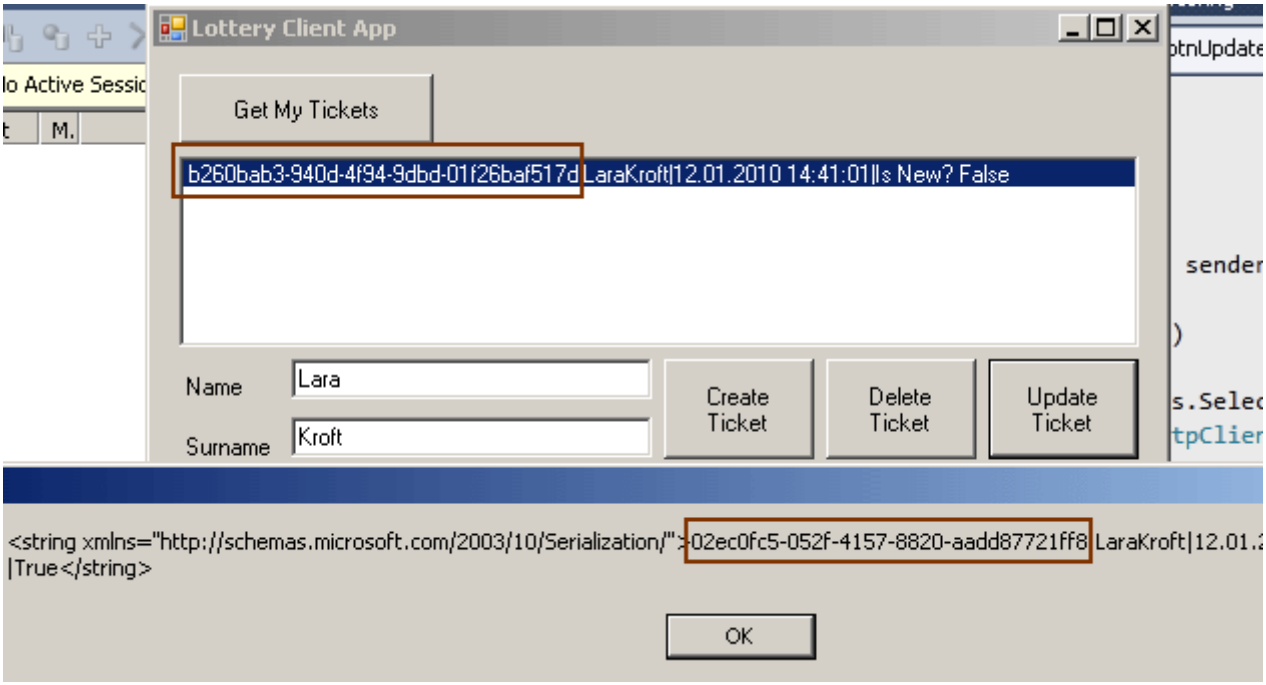


Peki ya silme ve güncelleme işlemlerinden ne haber? Bu operasyonlar için istemci tarafında aşağıdaki kodları yazmamız yeterli olacaktır.

```
private void btnDeleteTicket_Click(object sender, EventArgs e)
    {
        // öncelikle ListBox' ta seçili bir öğe olup olmadığına bakılır
        if (lstMyTickets.SelectedItem != null)
        {
            // Biletin numarası yani GUID bilgisi alınır.
            string ticketNumber = lstMyTickets.SelectedItem.ToString().Substring(0, 36);
            using (HttpClient client = new HttpClient("http://localhost:16088/LotteryService/"))
```

```
            {
                // HTTP Delete metoduna göre bir talepte bulunulur.
                string requestUri = String.Format("Lottery/Delete/{0}", ticketNumber);
                // Delete talebi için HttpClient tipinin Delete metodundan yararlanılır. Bu metodun kullanımına göre herhangibir HTTP Request Body içeriği bildirilmesi gerekli değildir.
                HttpResponseMessage responseMessage = client.Delete(requestUri);
                responseMessage.EnsureStatusIsSuccessful();

            }
        }
    }

    private void btnUpdateTicket_Click(object sender, EventArgs e)
    {
        if (lstMyTickets.SelectedItem != null)
        {
            string ticketNumber = lstMyTickets.SelectedItem.ToString().Substring(0, 36);
            using (HttpClient client = new HttpClient("http://localhost:16088/LotteryService/"))
            {
                // Update talebi hazırlanır
                string requestUri = String.Format("Lottery/Update/{0}", ticketNumber);
                // Güncelleme isteği aslında HTTP Put metoduna karşılık gelmektedir. Bunun için HttpClient tipinin Put metodundan yararlanılır.
                HttpResponseMessage responseMessage = client.Put(requestUri, HttpContent.CreateEmpty());
                responseMessage.EnsureStatusIsSuccessful();
                // Put metodunun çalıştırılması sonucu üretilen çıktı bu kez string bazlı olacak şekilde bir MessageBox aracılığıyla gösterilir.
                MessageBox.Show(responseMessage.Content.ReadAsString());
            }
        }
    }
```

örneğin var olan bir biletimizi güncellemek istediğimizi düşünelim. örnek çalışma zamanı görüntüsü aşağıdakine benzer olacaktır.



Tabi uygulamamızın pek çok yerinde **bug** ve **iş mantığı hatası** vardır. üstelik tam anlamıyla bir **istisna yönetimide(Exception Handling)** yapılmamaktadır. Ancak odaklanmamız gereken yada dikkat etmemiz gereken noktalar **WCF WebHttp Service** üzerinden **Post, Put, Delete** operasyonlarının nasıl sunulduğu ve istemci tarafında bunların nasıl ele alındığıdır. özellikle istemci tarafında kullandığımız tekniklere baktığımızda şu sonuçlara varabiliriz;

- Herhangibir proxy tipi kullanılmamıştır. Bilindiği üzere servisleri tüketmenin yollarından birisi istemci tarafında gerekli proxy tipinin üretilmesidir. **HTTP Get,Post,Put** ve **Delete** metodlarının kullanıldığı teknikte ise sadece paketlerin oluşturulup gönderilmesi ve cevapların değerlendirilmesi gerekmektedir.
- Sonuçların ilgili formata göre istemci tarafına gönderilmesi söz konusudur. Varsayılan olarak **XML** tipinden içerik döndürülmektedir. Ancak **JSON** formatında da dönüşler olabilir.
- **Get** metodundan elde edilen **XML** formatlı içerikler istemci tarafında **XElement** tipi ile ele alınabilir ve **XLINQ** kullanılarak ayrıştırılabilir.
- İstemci tarafından yapılan güncelleştirme çağrıları için **Put**, ekleme işlemlerine ait talepler için **Post**, silme işlemlerine ait talepler için **Delete** metodları kullanılır.
- **Post** ve **Put** metodları parametre olarak eğer gerekliyse bir **Body** içeriği sunmak zorunda olabilirler. Eğer sunmuyorlarsa **HttpContent.CreateEmpty()** metodu ile boş içerik gönderileceğinin belirtilmesi gerekir.

Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Lesson2.rar (196,61 kb) **[örnek Visual Studio 2010 Ultimate Beta 2 Sürümünde geliştirilmiş ancak RC sürümü üzerinde de test edilmiştir]**

WCF WebHttp Services - Tanışma (2010-02-01T14:50:00)

wcf,wcf 4.0,webhttp services,restful,non soap,wcf webhttp services,

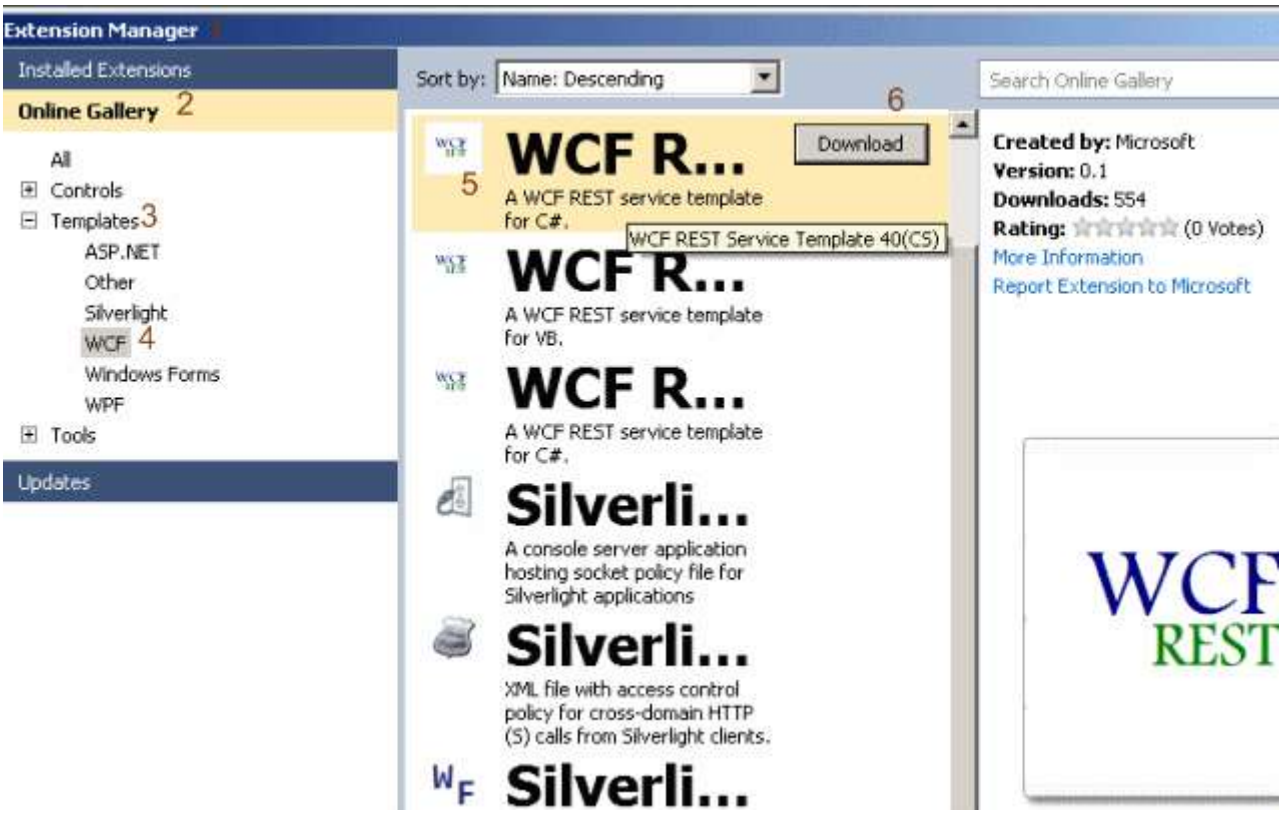
 Merhaba Arkadaşlar,

Nihayet taşlar yerli yerine oturmaya başladı. **2008** yılında düzenlenen **Microsoft PDC'** de tanıtılan sürüm ile başlayan macerada **Beta 1, Beta 2** versiyonları derken yavaş yavaş **RC, RTM** sürümlerinin çıkacağı günlere gelmekteyiz. Elbette hepizimin beklentisi bir an önce stabil bir sürüme kavuşabilmek. Bu günlerde çok doğal olarak **.Net Framework 4.0** ve **Visual Studio 2010** ürünlerinin sınırlarının daha da netleştiğini görmeye başladık. Her ne kadar henüz yayınlanmış yeni bir sürüm olmasa da, pek çok güncel ve geçerli kaynaktan okuduğumuz kadarı ile bu böyle. Taşların yerli yerine oturmaya başladığı ve herşeyin biraz daha belirginleştiği alanlardan biriside **Windows Communication Foundation 4.0.**

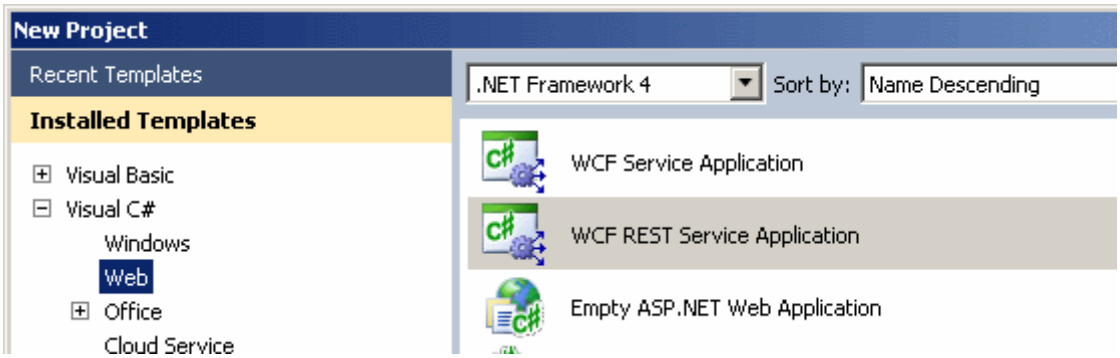
Hatırlayacağınız üzere WCF Eco System' i anlattığımız yazımızda, **WCF** alt yapısı üzerine geliştirilen ve amaca yönelik olarak farklılaştırılan servis geliştirme modellerine değinmiştik. Bunlardan biriside **WebHttp Services** idi. Bu yazımız ile birlikte **WebHttp Service'** lerini tanımaya çalışacağız. Aslında **WCF 3.5** sürümüne kazandırılan **Web programlama** teknikleri sayesinde zaten uzun bir süredir farkında olduğumuz **non-SOAP** bazlı bir modelden bahsediyoruz. Bildiğiniz üzere **WebGet** ve **WebInvoke** isimli **nitelikler(attribute)** yardımıyla servis operasyonlarının **HTTP Get,Post,Put** ve **Delete** metodlarına cevap verebilecek şekilde tasarlanması mümkün. Ancak zaman ilerledikçe **REST(REpresentational State Transfer)** modeline göre **WCF** servislerinin daha kolay geliştirilmesini sağlayan ve **WCF 4.0** içerisinde gömülecek yeni özelliklerin bir ön görünümünü bizlere sunan **WCF REST Starter Kit** ile karşılaştık.

WCF Eco System' in bir parçası olan **WebHttp Service'** ler, **.Net Framework 3.5** ile gelen **Web Programlama modeli**, **REST Starter Kit** ile tanıtılan kabiliyetler ve bunlara ek yeni özelliklerin **.Net Framework 4.0** içerisinde ele alınmasını sağlayan bir geliştirme alt yapısı olarakta düşünülebilir. Şimdi dilerseniz **WebHttp Service'** lere bir merhaba demeye çalışalım. örneğimizi **Visual Studio 2010 Ultimate Beta 2** sürümü üzerinden geliştirmeye çalışıyor olacağız. Ancak yazıyı hazırladığım tarihte araştırdığım **MSDN, The .NET Endpint** vb blog sitelerinde yer alan bilgilere göre **Visual Studio 2010'** un o anki

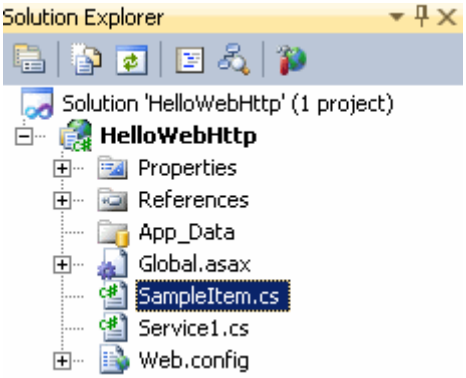
sürümü üzerinde **WebHttp Service'** leri için bir **proje şablon(project template)** bulunmamaktaydı. Bu nedenle öncelikli olarak **online template'** lerden **WebHttp Service** için olanları indirmemiz gerekiyor. Bu amaçla **Visual Studio 2010** ortamında **Tools->Extensions Manager->Online Gallery** kısmına geçiş yapıp **WCF** ile ilişkili olan şablonlardan **WCF Rest Service Template'** i indirmemiz gerekiyor. Ben **4.0** versiyonunun **C#** programlama dili destekli olanını indirdim. Son sürümde büyük ihtimalle şu anda online olarak indirdiğimiz bu şablonun ve başka diğer şablonların **Visual Studio 2010** içerisine gömülü olarak geleceğini ümit etmekteyim.



Download ve **Install** işlemlerinin ardından yolumuza devam edebiliriz. Bu amaçla, **Visual Studio 2010** ortamında yeni bir proje oluşturup, **Web** sekmesinde yer alan **WCF Rest Service Application**şablonunu seçmemiz yeterli olacaktır. Bu şablonun kurulum işlemi sonrasında çıkmaması olasıdır. Bu durumda **New Project** iletişim kutusunda yer alan **Enable the loading per-user extensions** bağlantısını kullanarak etkinleştirme işlemini yapmamız yeterli olacaktır. **Visual Studio 2010** ortamımızı tekrardan açtığımızda aşağıdaki ekran görüntüsünde olduğu gibi yeni proje şablonunun kullanabilir olduğunu göreceğiz.



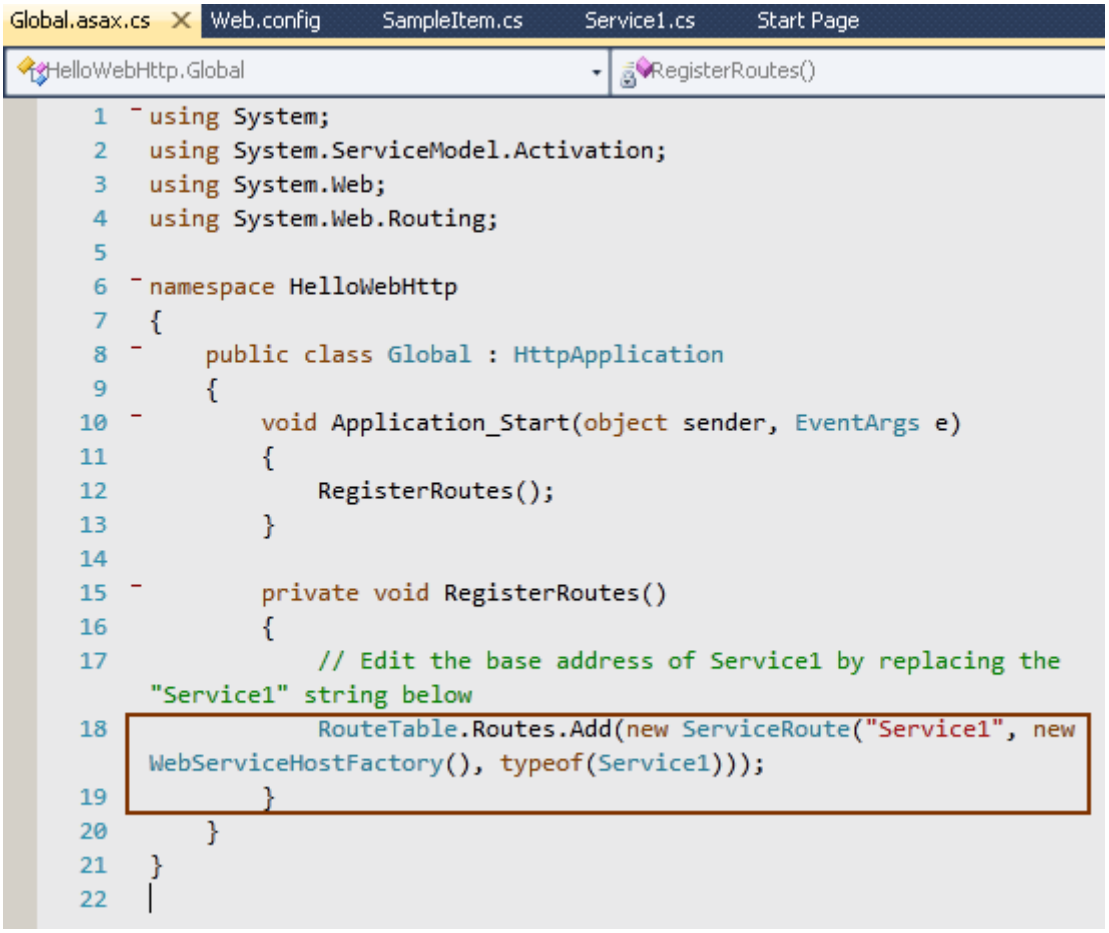
HelloWebHttp isimli servis uygulamasına ait **Solution** içeriğinin ilk etapta otmatik olarak aşağıdaki gibi oluşturulduğunu gözlemleyebiliriz.



Service1.cs isimli örnek dosya içerisinde **servis sözleşmesi(Service Contract)** yer almaktadır. Bu sözleşme içerisinde yer alan metodlara **WebGet** ve **WebInvoke niteliklerinin(Attributes)** uygulandığı görülmektedir. Bu nitelikler bildiğiniz üzere servis operasyonlarına **HTTP Post,Get,Put,Delete** çağrılarının yapılabilmesi için gereklidir. Sınıf içerisine **Get, Post, Put** ve**Delete** metodlarının her biri için örnek nitelik kullanımları serpiştirilmiştir. Ayrıca serileştirilebilir örnek bir tipte yer almaktadır*(SampleItem)*. **GetCollection** operasyonu **SampleItem** tipinden generic bir listeyi **HTTP Get** metoduna göre döndürmektedir. **Create** servis operasyonu ile yeni bir **SampleItem** nesnesinin **HTTP Post** metoduna göre örneklenmesi sağlanır. Tek bir **SampleItem** nesne örneğinin elde edilmesi için **Get** isimli servis operasyonunun aşırı yüklenmiş diğer bir versiyonu kullanılmaktadır. **Update** servis operasyonu ile **HTTP Put** metoduna göre güncelleme işlemi yapılmakta olup, **Delete** servis operasyonuda bir **SampleItem** nesnesinin **HTTP Delete** metoduna göre silinmesini sağlamaktadır. **Solution** içerisinde dikkat çekici bazı noktalar bulunmaktadır;

- örneğin **web.config** içeriği. Burada **WCF 4.0** ile birlikte gelen **basitleştirilmiş konfigurasyon(Simplified Configuration)** özelliklerine yer verilmiştir. Bu sebepten daha sade, okunaklı ve fonksiyonel bir konfigurasyon içeriği oluşmuştur.
- Dikkat çekici bir diğer noktada **global.asax** dosyasının var olmasıdır. Aslında bu bir web uygulaması olduğu için son derece normaldir. Lakin gözden kaçırılmaması gereken bir gerçek vardır; **svc** uzantılı bir servis dosyası fiziki olarak yoktur. çünkü **Asp.Net 4.0 Routing** özelliği kullanılmaktadır. çok tabi olarak yönlendirme

işlemleri için **global.asax**dosyasında yapılması gereken bazı işlemler vardır. Bu nedenle hazır olarak **global.asax** içeriği aşağıdaki gibi üretilmektedir.



- Yine servis kodlarına baktığımızda **OperationContract** niteliğinin kullanılmadığını görürüz. Oysaki **.Net 3.5** ve **WCF Rest Starter Kit** sürümlerine baktığımızda**WebGet** ve **WebInvoke** nitelikleri dışında **OperationContract** niteliğininde kullanılması gerektiğini bilmekteyiz. **OperationContract** servis sözleşmelerinden sunulan operasyonların **WCF çalışma zamanına** bildirilmesinde rol oynadığı için bu son derece doğaldır. Ne varki **WCF WebHttp** servislerinde **OperationContract** niteliği opsiyoneldir.

Şimdi servis uygulamamız üzerinde bir kaç küçük değişiklik yapalım. öncelikli olarak hayatımızı kolaylaştırmak adına **Entity Framework'** ten yararlanalım ve meşhur **Chinook**veritabanını ve işlemleri basit bir biçimde ele almak için sadece **Artist** tablosunu kullanmak istediğimizi düşünelim. Gerçi bu noktadan sonra biraz **WCF Data Service'** lere doğru kaymaya başlamış oluyoruz ancak amacımız tabiki **HTTP Get,Post,Put** ve **Delete** işlemlerini kendi kontrolümüz altında geliştirmek.

Not : *Tam bu noktada geliştirilen uygulamanın **Data Service'** ten veya **RIA Service'** ten ne farkı kaldığı sorusu akla gelebilir.* 🙂 ***WCF WebHttp*** *servislerinde asıl nokta operasyonun **non-SOAP** olacak şekilde sunulması(yani **HTTP***

Get,Post,Put,Delete*) ayrıca* ***URI, format, protocol*** *gibi bilgilerin tamamen geliştirici kontrolü altında olmasıdır.*

Şimdi servis kodlarını aşağıdaki gibi geliştirdiğimizi varsayalım.

```
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;

namespace HelloWebHttp
{
   [ServiceContract]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.PerCall)]
   public class HelloService
   {
      [WebGet(UriTemplate = "ArtistList")]
      public List<Artist> GetAllArtists()
      {
         ChinookEntities entites = new ChinookEntities();
         return entites.Artist.OrderByDescending(a => a.Name).ToList();
      }

      [WebGet(UriTemplate = "Artist/{name}")]
      public List<Artist> FindArtists(string name)
      {
         ChinookEntities entities = new ChinookEntities();

         return (from artist in entities.Artist
                 where artist.Name.Contains(name)
                 select artist).ToList();
      }

      [WebGet(UriTemplate =
"Artist/InRange?idFirst={firstId}&idSecond={secondId}")]
      public List<Artist> GetArtistsInRange(int firstId, int secondId)
      {
         ChinookEntities entities = new ChinookEntities();
         return (from a in entities.Artist
                 where a.ArtistId >= firstId && a.ArtistId <= secondId
                 select a).ToList();
      }
```

```
    }
}
```

Dikkat edileceği üzere sadece **HTTP Get** metodlarının çalışmasına yönelik 3 örnek operasyon yer almaktadır. **WebGet** niteliklerinde **UriTemplate** özelliklerine atanan değerler yardımıyla **HTTP Get** taleplerinin nasıl olması gerektiği belirlenmektedir. Servise gelen taleplerin **Routing** sürecine dahil olması için **global.asax.cs** kodlarında da aşağıdaki değişiklikleri yapmamız yeterli olacaktır.

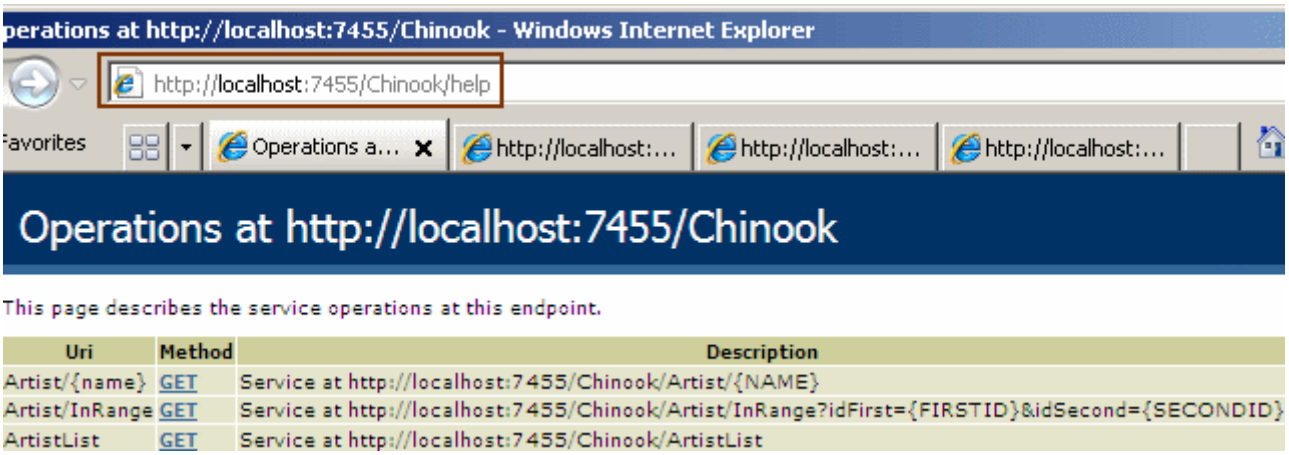
```
using System;
using System.ServiceModel.Activation;
using System.Web;
using System.Web.Routing;

namespace HelloWebHttp
{
    public class Global
        : HttpApplication
    {
        void Application_Start(object sender, EventArgs e)
        {
            RegisterRoutes();
        }

        private void RegisterRoutes()
        {
            RouteTable.Routes.Add(new ServiceRoute("Chinook", new
WebServiceHostFactory(), typeof(HelloService)));
        }
    }
}
```

Dikkat edileceği üzere **http://makineadı:portnumarası/Chinook** üzerine gelen talepler sonrasında, **WebServiceHostFactory'** nin ayağa kaldırılması ve **HelloService** sınıfının örneklenerek işleme alınmasının sağlanması gerçekleştirilmektedir. Artık testlerimize başlayabiliriz.

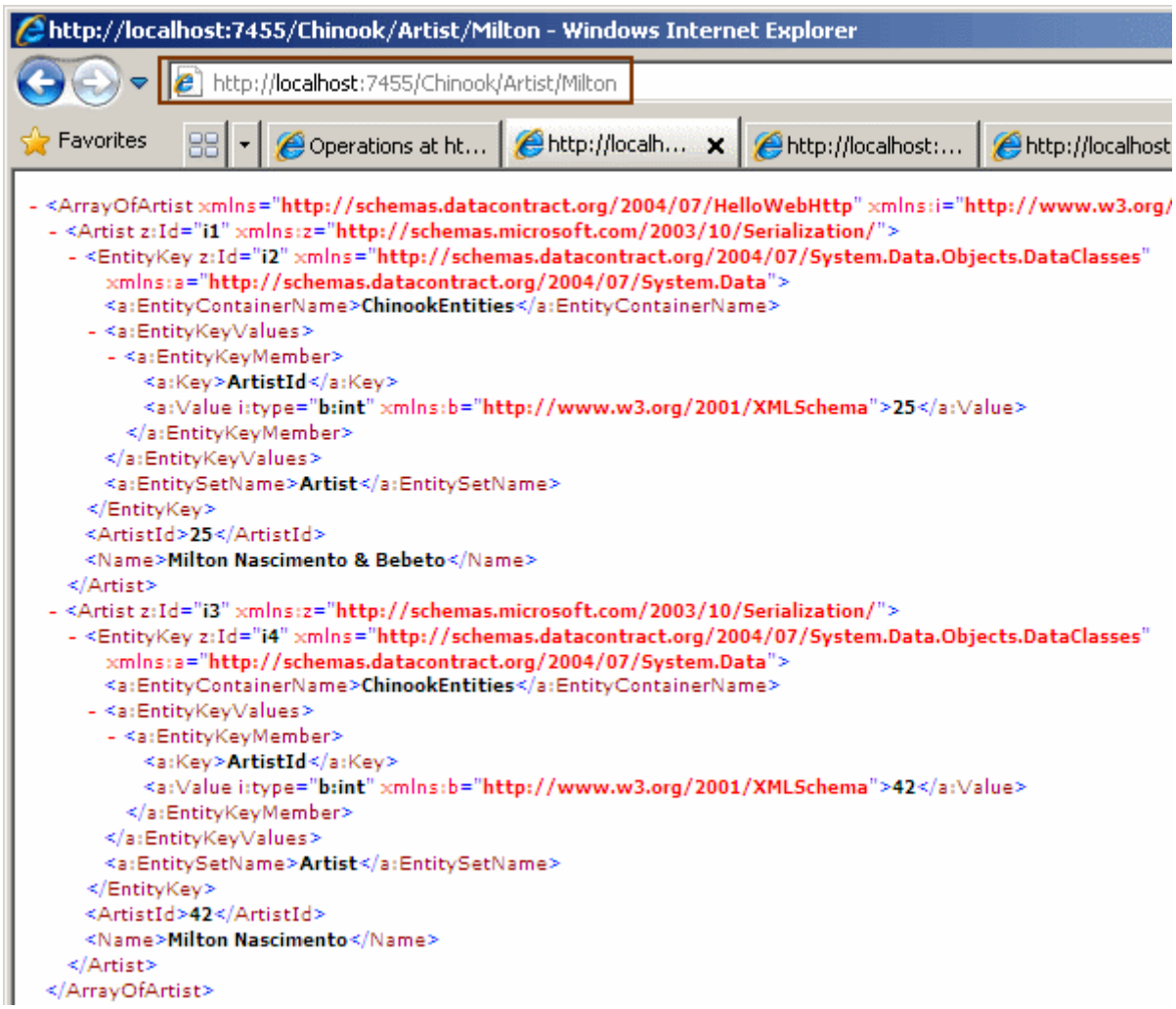
Not: *Dilerseniz Web uygulamasını **IIS** üzerinden host ederekte deneyebilirsiniz. Ancak ister **Asp.Net Development Server** ister **IIS** olsun, **URL** satırında **RegisterRoutes** metodunda yer alan**Chinook** bilgisini kullanmamız servise ulaşmamız için yeterli olacaktır.*

Tabi test derken ilk etapta servis operasyonlarını nasıl çağırabileceğimizi bilemeyebiliriz. Yada bulmak için araştırmaya üşenebiliriz. İşte bu amaçla **WCF** tarafına gelen **Auto Help**yetenekleri sayesinde çalışma zamanında yardım sayfasına gidebilir ve

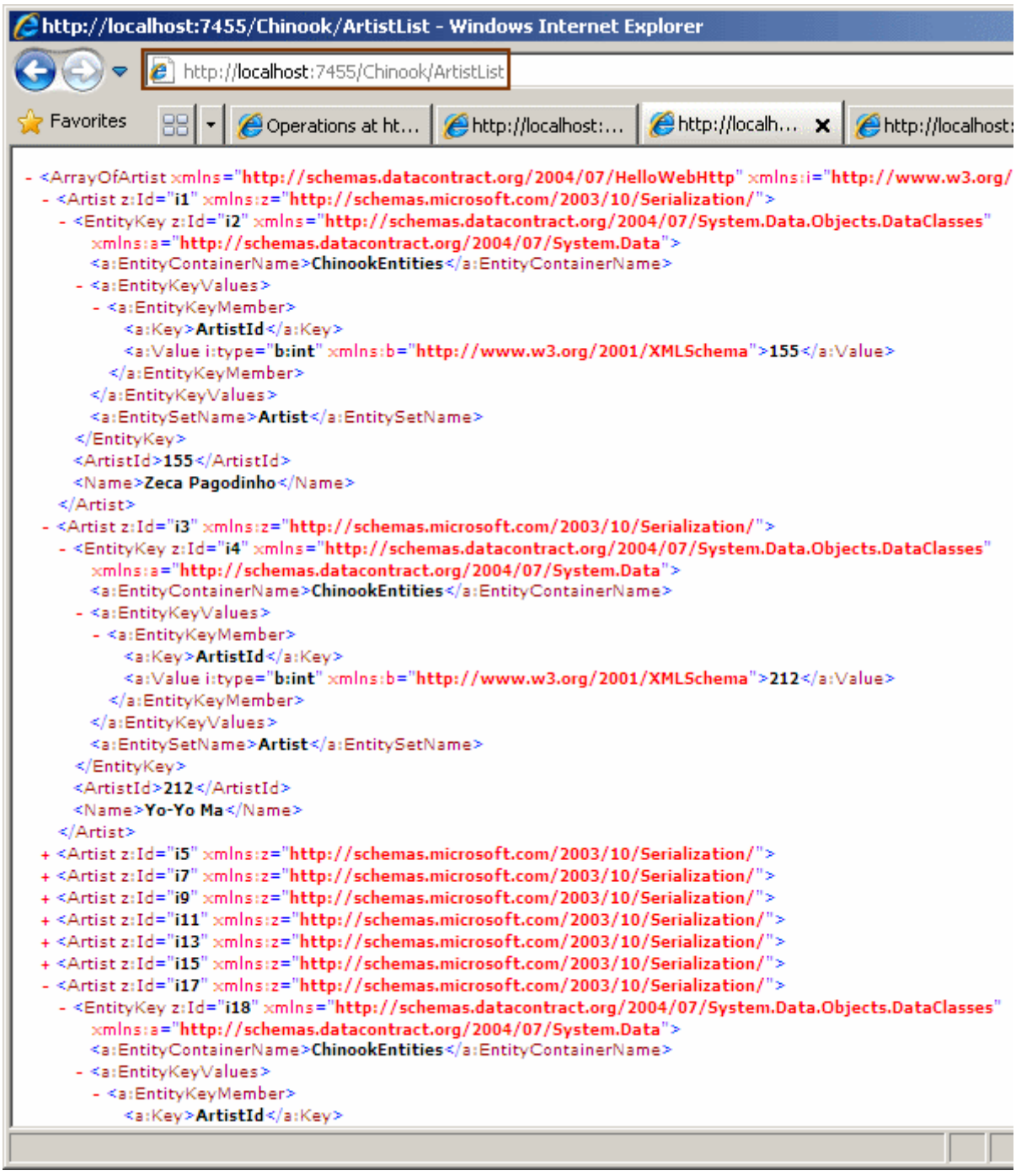
servis operasyonlarını nasıl çağırabileceğimizi, içeriklerinin ne olacağını görebiliriz.. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.



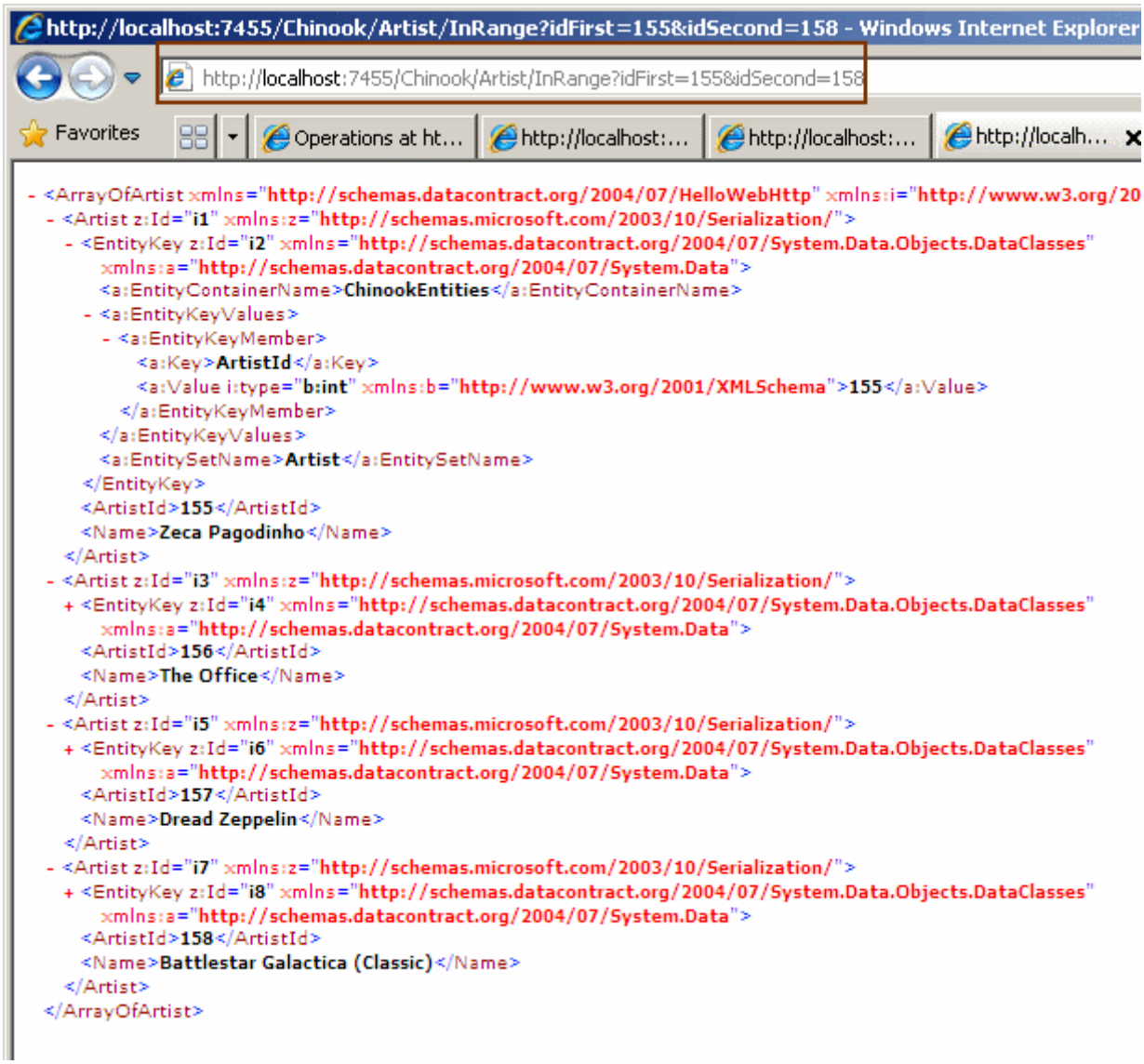
İlk olarak belirli bir kelimeyi içeren **Artist** listesini elde etmek istediğimizi düşünelim. örneğin adında **Milton** kelimesi geçenleri bulmak istiyoruz. Bu durumda **URL** satırında**Chinook/Artist/Milton** yazmamız yeterli olacaktır. Sonuçlar aşağıdaki ekran görüntüsünde olduğu gibidir.



Eğer **tüm Artist** listesini elde etmek istiyorsak bu
durumda **URL** satırından **Chinook/ArtistList** bilgisini girmemiz yeterli olacaktır. Bu durumda elde edilen sonuçlar aşağıdaki ekran görüntüsünde olduğu gibidir.



Son olarak **ArtistId** değer aralığına göre **Artist** listesini elde etmek istediğimizi düşünelim. Bu durumda **URL** satırından **Chinook/Artist/InRange?idFirst=155&idSecond=158** gibi bir **URL**satırı girmemiz yeterli olacaktır ki buna göre
örneğin **ArtistId** değerleri **155** ile **158** aralığında olanların listesini aşağıdaki ekran görüntüsünde olduğu gibi elde edebiliriz.



Oldukça basit ve etkili değil mi? **WCF WebHttp Service'** ler ile ilişkili incelemelerimize devam ediyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HelloWebHttp.rar (31,23 kb) **[örnek Visual Studio 2010 Ultimate Beta 2 Sürümünde geliştirilmiş ancak RC sürümü üzerinde de test edilmiştir]**

Correlation Nedir? Yenir mi? İçilir mi? (2010-02-01T09:25:00)

wcf,wcf 4.0,wf,wf 4.0,workflow services,

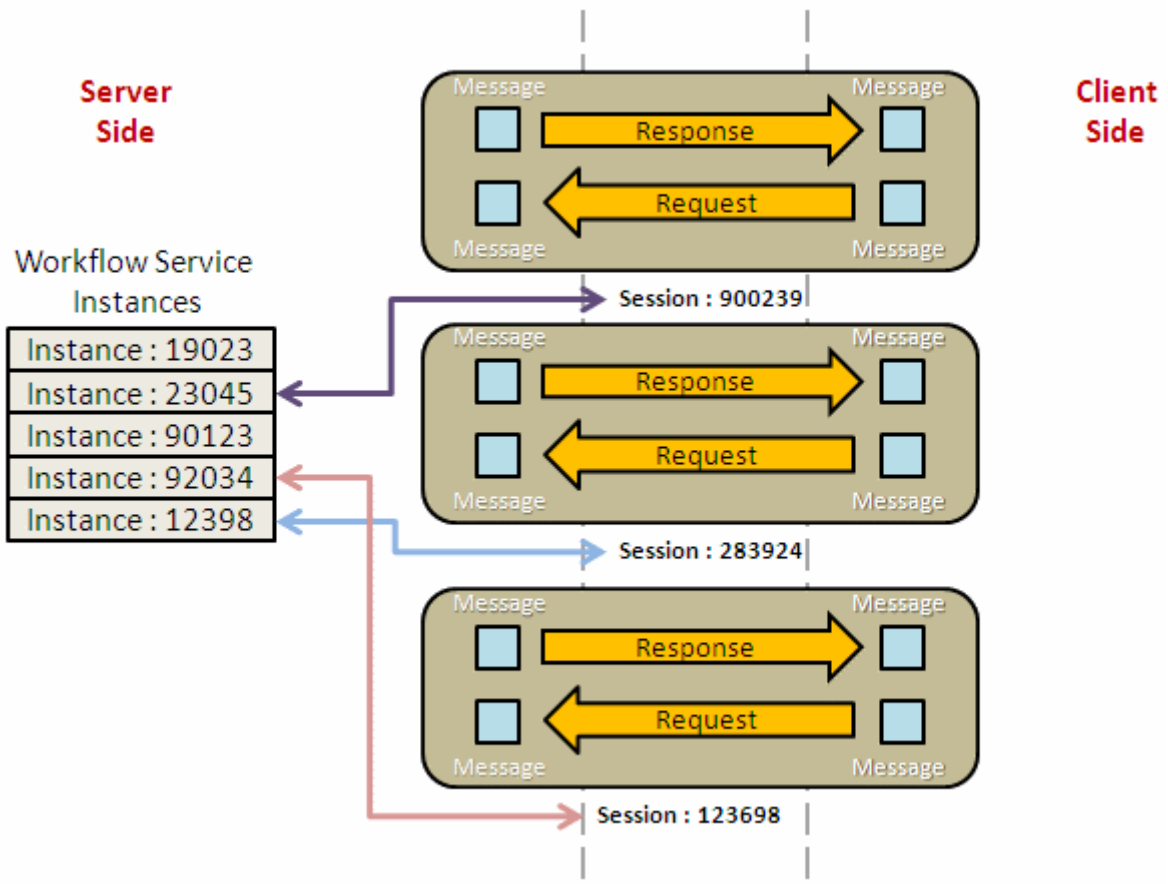
 Merhaba Arkadaşlar,

Bazen bir kavramı yada konuyu anlamakta inanılmaz zorlandığınızı hatırlayın. Ne yaparsınız? Kimisi kendisini yemeğe verir. Kimisi hayat küsermişçesine bir köşeye çekilir. Kimisi kendiyle baş başa kalır ve çığlık çığlık haykırır. Kimisi de daha akıllı davranıp bir süre tatile çıkar veya anlayamadığı kavramla ilişkili herhangibir dökümanı bir süreliğine araştırmamaya, okumamaya karar verir. Neredeyse unuturcasına bir zaman koyar araya. Sonrasında ise aynı konuyu tekrar araştırmaya karar verir. İnanın başarılı olma şansı bir önceki denemeye göre çok daha yüksek olacaktır. önemli olan noktalardan birisi, yılmadan bu iterasyona devam edebilmektir. Okudunuz, hala anlamadınız...Kısa bir ara daha...Sonra tekrar aynı konu ama mümkünse farklı kaynaklarla...

Bende bir süredir **Workflow Service'** lerde oldukça önemli olan konulardan birisi üzerinde araştırmalarımı tamamen durdurmuştum. **Correlation**. çünkü;**Matematikte "bağlılaşım/korelasyon"**, **ekonomide "bağlanım"**, **nükleer bilimlerde "bağlantı/eş ilişki"**, **denizbilimde "kaçınım"**, **tıpta "Aferent uyarıların gerekli cevabı oluşturmak üzere beyinin ilgili merkezinde birleşmesi"** olarak çevirileri yer alan bu kavramın, **Workflow Service**içerisinde ne anlama geldiğini anlamak için epey bir süre tepinmem gerekmişti. Geçtiğimiz günlerde aynı konu üzerinde yeniden durmaya ve araştırmaya ve edindiğim bilgileri sizlere paylaşmaya karar verdim. İşte elde ettiğim sonuçlar;

Correlation kavramını mesajları bir arada gruplamanın bir yolu olarak düşünülebilir ilk etapta. örneğin bir **talep(Request)** ve bu talebe karşılık gönderilen **cevap(Reply)** arasındaki ilişki **Correlation** olarak ifade edilmektedir. özellikle **WCF(Windows Communication Foundation)** tarafında **Session** bazlı haberleşmelerde mesajlar arasında bir**Correlation** oluştuğu söylenebilir.
Ancak **Correlation'** ın farklı bir yönü daha vardır. Bir **servis örneği(Instance)** ile bir **oturum(Session)** arasında da bağıntı kurulabilir. Yani bir**SessionId** değerine ait olaraktan hareket eden mesajların içeriğinde yer alan bazı veri parçalarının, **Session** ile alakalı bir servis örneği ile eşleştirilmesi mümkün olabilir. Bir başka deyişle

aynı **SessionId** değeri altındaki mesajların her zaman için aynı servis örneğine ait olduğunun anlaşılmasında, **SessionID=Service Instance ID** eşitliğinin sağlanması da**Correlation** olarak ifade edilebilir. Bu ilişki çoğunlukla **bilinçsiz(Implicit)** olarak sağlanır. Yani, geliştiricinin çoğu zaman bir aksiyonda bulunmasına gerek yoktur. Aslında bu durumu aşağıdaki şekilde olduğu gibi canlandırabiliriz.



Ancak **Workflow** örneklerinde uzun zaman süren süreçlerin ele alınması da söz konusudur***(Long Running Process)***. İstemcilerin, **Workflow Service'** ler ile olan haberleşmelerinde aradaki oturumu kapatıp ayrılmaları bu tip süreçlerde son derece yaygındır. Buna göre istemci ile servis arasındaki oturumun her an sonlanabilir olması önemli bir sorunu ortaya çıkarmaktadır; **Correlation nasıl sağlanacak**? 🤔

Bu amaçla **Workflow Service'** lerde **Correlation'** ın sağlanması için kullanılan çeşitli teknikler mevcuttur. Aslında burada da tam bir kavram kargaşası vardır. En güvenilir kaynaklardan birisi olarak ele alacağımız **MSDN**, **Correlation'** ı **Content-Based** ve **Protocol-Based** olmak üzere iki çeşide ayırmıştır. **Protocol-Based Correlation'** da kendi içerisinde **Context** ve**Request-Reply** isimli iki farklı **Correlation** tekniğini daha barındırmaktadır.

Content-Based Correlation' da **service örneği(Instance)** ile ilişkili olan veri*(Map edilmiş veri olaraktan da düşünebiliriz)* mesajın içeriğinde yer alır. örneğin mesajın **Header** veya**Body** kısımlarında bulunabilir. Dolayısıyla **Correlation'** ı sağlayan

arabirimlerin **XML** tabanlı olan bu veri içeriğini kontrol edebilmesi gerekmektedir. İşte bu noktada **XPath** gibi sorgu teknikleri devreye girmektedir. **Protocol-Based Correlation** ise iletişim mekanizmasını baz alır ve buna göre mesajlar arasında yada mesajlar ile doğru servis örneği arasında gerekli eşleştirmeyi sağlar.

Bu giriş yazımızda özellikle **Content-Based Correlation** üzerinde durabiliriz. En çok örneklenen model genellikle budur. **Workflow Service'** lerde servise gelen ve servisten giden mesajların içerdiği veri parçaları ile çalışma zamanındaki servis örnekleri arasında eşleştirme yapmak*(yani Correlation' ı sağlamak)* son derece kolaydır. Tüm mesajlaşma aktivitelerinin**CorrelationInitializers** isimli bir koleksiyonu bulunur. Bu koleksiyon içerisinde **Key-Query** çiftleri yer almaktadır. Tahmin edileceği üzere **Key** değerleri ile **Query'** lerin birbirlerinden ayrıştırılması sağlanır. En önemli nokta ise **Query'** dir. **Query** içerisinde yer alan **XPath** sorgusu ile **Content** içerisindeki veri işaret edilir. Tabi bir mesajlaşma aktivitesi, diğer bir mesajlaşma aktivitesinin başlattığı **Correlation'** u takip edebilmelidir. Bunun içinde **CorrelationWith** isimli özellikten yararlanılır. Bu bilgilere göre **Correlation'** ın bir şekilde **başlatılması(Initialize)** ve **takip edilmesi(Follow)** gerektiği anlaşılmaktadır. Başlatılan bir **Correlation'** ın takip edilmemesi halinde mesajlar ve servis örneği arasında bir eşleştirmenin yapılması söz konusu olmayacaktır. Elbette **Workflow Foundation 4.0** içerisinde bir **Correlation'** ın başlatılmasını kolaylaştıran bir aktivitede gelmektedir. **InitializeCorrelation** bileşeni.

Sanırım şu ana kadar anlattıklarımız ile kafamızda **Correlation'** ın ne olduğuna dair bir fikir oluşmuştur. Tabi konuyu kavramak tek başına yeterli değildir. Pratiğe dökmemizde yarar vardır. Ancak şu an için bu konudaki araştırmaya ara verip tatile çıkmak niyetindeyim. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WCF RIA Services - Alan Bazlı(Field Based) Rol Kontrolü [Beta 2] (2010-01-27T11:33:00)

wcf ria services,.net ria services,wcf,wcf eco system,

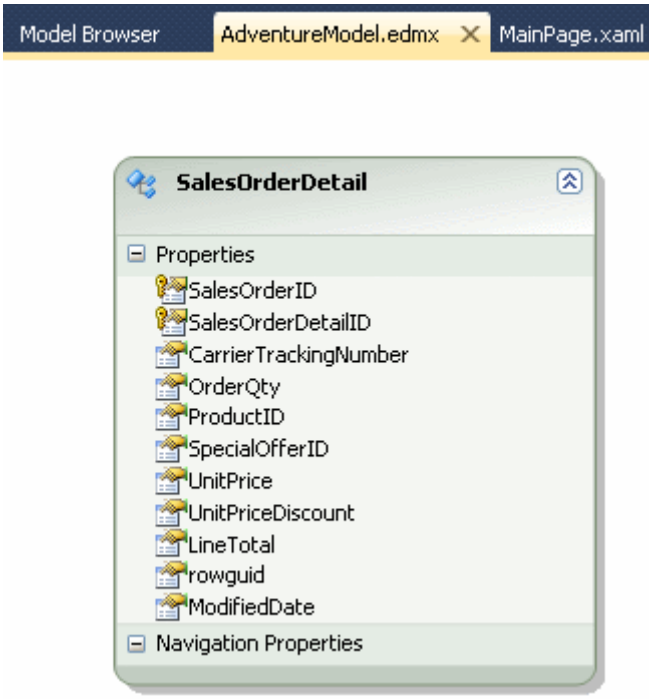
 Merhaba Arkadaşlar,

Hani bazen insanın aklına son derece zekice fikirler gelir ya... Sene **1992**. Lise öğrencisiyim. Bazı akşamlar yazlığımızdaki odamda üniversiteye hazırlanmaya çalışırdım. Güzel yaz gecelerinde, tertemiz ada ikliminde, mis gibi kokan iyotlu deniz suyunun çok yakınlarında konsantre olmak her ne kadar çok zor olsa da, buna mecburdum. Odamdaki flöresan ışığını çalışma ortamı için hiç uygun bulmazdım. Bunun yerine sarı ışığı tercih ederdim ve aynen yandaki şekilde görülene benzer bir gece lambam vardı. Aslında lambanın etrafında şık bir küre bulunmaktaydı fakat sakarlığıyla bilinen bendeniz onu bir ara kırmıştım. Tabi hal böyle olunca şöyle bir sorunla karşılaştım. Işık direkt olarak gözüme geliyor ve çok rahatsız ediyordu. çözüm olarak nemi yaptım. Dahiyane bir fikirle **Amerika'** daki bir arkadaşımın hediye ettiği **Newyork Nicks** takımının logosunu taşıyan şapkayı, lambanın üstüne güzelce yerleştirdim. özellikle ışığın gözüme direkt olarak girmesini engelleyen ama etrafı ve okuduklarımı görmemi sağlayan bir açıyı düşünüp, ölçüp biçerek, dikkatlice yerleştirdim. Kendimle gurur duyuyordum. Bu zeka ile **NASA**'ya bile gidebilirim diye düşünüyordum

Ancak gecenin ilerleyen saatlerinde değil **NASA**, sıradan bir bölümü bile kazanmamın zor olduğuna kanaat getirdim. Nitekim ampülün zaman içerisinde çevreye yaydığı aşırı ısıyı tahmin edememiştim. Ancak odanın içerisine bir yanık kokusu yayıldığında bir şeylerin ters gittiğinin farkına varabilmiştim. En nihayetinde güzelim şapkanın ortasında kocaman bir yanık izi ve erimiş kumaş parçaları ile kala kaldım. Neredeyse koca bir delik açılmıştı.

İşte geçen gün yine böyle dahiyane bir fikir gelir mi aklıma diye düşünürken, **WCF RIA Servis** operasyonlarından dönen **Entity** nesnelerinin alanlarını rol bazlı olarak ele alabilir miyiz diye sorgulamaya başladım. Peki neden böyle bir ihtiyacımız olsun? çok basit bir sebep öne sürebiliriz. Sunucu tarafındaki servisten dönen Entity örnekleri içerisindeki alanlarının bazılarının, Login olan kullanıcı tarafından görülmemesi veya kullanılamaması istenebilir. Malum **DomainService** tipi içerisinde yer alan operasyonlarda **Login** olan kullanıcının içerisinde bulunduğu rol elde

edilebilmektedir. Buna göre sorgunun üreteceği çıktı içerisine dahil edilecek alanların yetkiye göre oluşturulması sağlanabilir...mi acaba? Durumu örnek bir senaryo üzerinden incelersek çok daha anlaşılır olacaktır. öncelikli olarak **AdventureWorks** veritabanı içerisinde yer alan **SalesOrderDetail** isimli tabloyu kullanmak istediğimizi düşünelim.



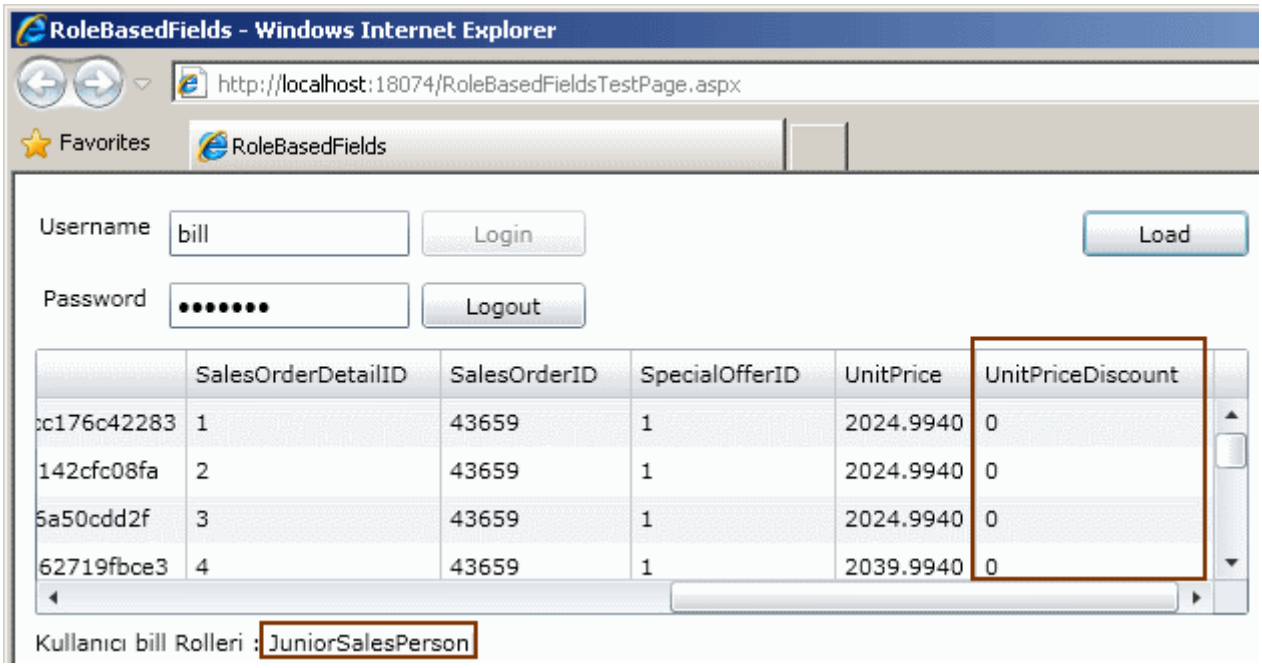
örnek senaryomuzda çok doğal olarak **Authentication** alt yapısınında tesis edilmiş olması gerekmektedir. **WCF RIA Service'** lerinde **Authentication Domain Service** kullanımından daha önceki yazılarımızda bol bol bahsettiğimizden bu detayları atlıyoruz. Ancak **ASP.NET Membership** tabanlı olarak kurulan **Authentication** alt yapısı içerisinde testler için iki rol olduğunu söyleyebiliriz. **AuthorizedSalesPerson** ve **JuniorSalesPerson**. Senaryomuza göre güya **AuthorizedSalesPerson** rolünden gelen kullanıcılar **UnitPriceDiscount** alanının değerini görebilirken,**JuniorSalesPerson** rolündeki kullanıcılar göremeyecektir. Başlangıçta **AdventureDomainService** isimli **Domain Service** sınıfını aşağıdaki gibi düzenlediğimizi düşünelim.

```
namespace RoleBasedFields.Web
{
    using System.Linq;
    using System.Web.DomainServices;
    using System.Web.DomainServices.Providers;
    using System.Web.Ria;

    [RequiresAuthentication]
    [EnableClientAccess()]
    public class AdventureDomainService
        : LinqToEntitiesDomainService<AdventureWorksEntities>
```

```
    {
        public IQueryable<SalesOrderDetail> GetSalesOrderDetails()
        {
            return (from sod in ObjectContext.SalesOrderDetails
                    select sod).Take(50);
        }
    }
}
```

Kişisel Not: 121317(Yüz yirmi bir bin üçyüz on yedi)...SalesOrderDetail *tablosunda bu kadar satır bulunmaktadır. Bu satırların tamamını istemci tarafına göndermek performans açısında tercih edilmemelidir. Zaten* ***WCF RIA Service'*** *lerin kullanımı ile ilişkili best practices tiyolarında, üretilen standart sorguların filtreler ile düzenlenmesi önerilmektedir. En azından dönen veri içeriğinin bir gözden geçirilerek gerektiğinde performans için filtrelenmesi düşünülmelidir. Bu sebepten örneğimizde* ***Take genişletme metodundan(Extension Method)*** *yararlanılarak ilk* ***50*** *satırın alınması sağlanmıştır.*

İlk etapta sunucu tarafındaki operasyonda herhangibir rol kontrolü yapılmamaktadır. Buna göre Login olan bir kullanıcı için ekran çıktısı aşağıdaki gibi olacaktır.



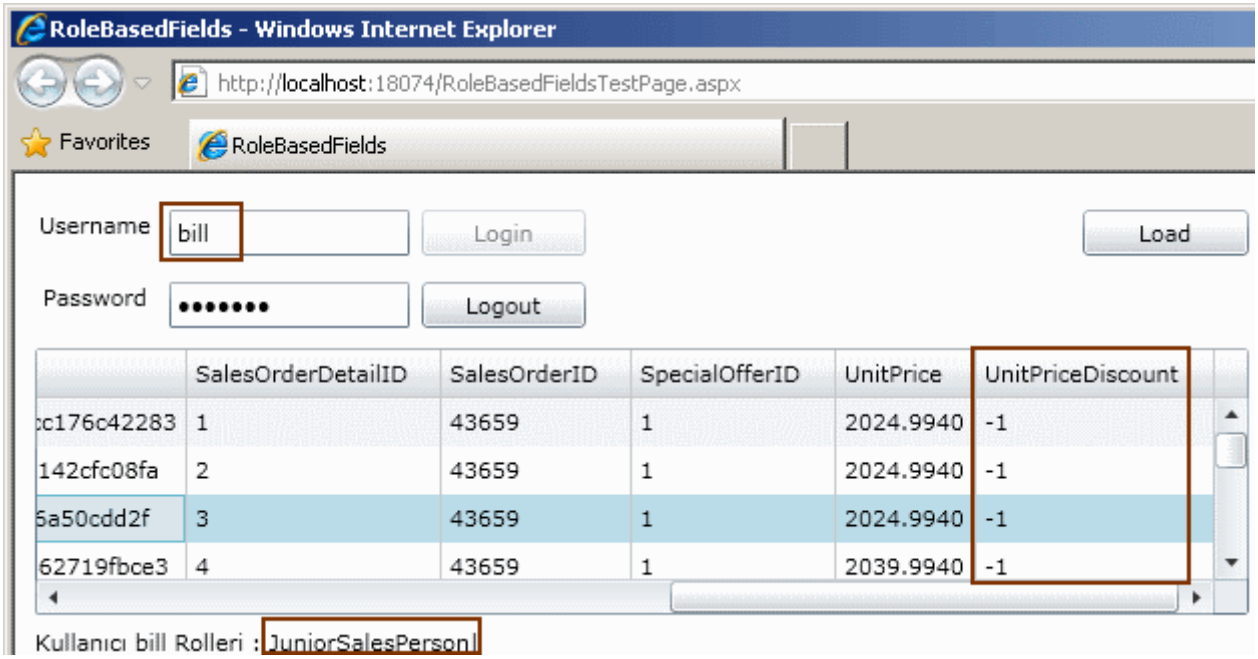
Ancak örnek senaryomuza göre **JuniorSalesPerson** rolünde olan **bill** isimli kullanıcının **UnitPriceDiscount** alanını görmemesi veya anlamlandıramaması gerekmektedir.*(Anlamlandıramamasının ne kadar zor olduğunu biraz sonra anlayacağız)* Buna göre sunucu tarafında yer alan operasyonun özelleştirilmesi gerekmektedir. Aslında yazımızın ulaşmak istediği tek nokta budur. Peki ama nasıl? 🙂

Sonuçta istemci ve sunucu tarafında eş olan **SalesOrderDetail Entity** sınıf bilgisini çalışma zamanında değiştirmemiz şu etapta pek mümkün değildir. Akla ilk gelen yöntem

result set çekilirken role göre gösterilmesi istenmeyen alana örneğin null değer atanmasını sağlamak olabilir. *(Bu konu ile ilişkili olaraktan yaptığım araştırmalarda, blog girdisini hazırladığım tarih itibariyle [Brad Abrams' ın ilgili yazısında](#) bu tip bir teknik uygulandığını gördüm)* Tabi örneğimizdeki alan **null** değer almamaktadır. Buna göre belki **-1** değer atanması sağlanabilir. Ama bu durumdada alan yine görülebilir olacaktır. Aşağıdaki kod parçasını göz önüne alalım.

```
public IQueryable<SalesOrderDetail> GetSalesOrderDetails()
    {
        var resultSet = (from sod in ObjectContext.SalesOrderDetails
                         select sod).Take(50);

        foreach (var result in resultSet)
        {
            if (ServiceContext.User.IsInRole("JuniorSalesPerson"))
                result.UnitPriceDiscount = -1;
        }

        return resultSet;
    }
```

Bu kod parçasında görüldüğü gibi **resultSet** gönderilmeden önce her bir satırı taranmakta ve **ServiceContext** üzerinden elde edilen güncel kullanıcının rolüne bakılarak**UnitPriceDiscount** alanına **-1** değer atanması sağlanmaktadır. Buna göre çalışma zamanı çıktısı bill isimli kullanıcı için aşağıdaki gibi olacaktır.



Peki istediğimiz bu muydu?

Kesinlikle değil. Bizim hayalimiz ilgili alanın istemci tarafından görülmemesini **sunucudaki operasyon üzerinden** sağlamaktı. Oysaki öğrenebildiğimiz sadece şu oldu; servis tarafındaki operasyondon dönen **resultSet** içeriğindeki veriyi istersek **Login** olan kullanıcının rolüne göre değiştirebiliriz. İşte şu anda lambaya koyduğumuz şapkanın delindiğini görmekteyiz. Benim NASA hayalleri yine yalan oldu anlayacağınız.

Peki ya çözüm?

En ideal çözüm istemci tarafında kullanıcının rolüne göre ilgili alanının gizlenmesi olarak düşünülebilir. Ancak buda optimal bir çözüm olmayacaktır. Nitekim sunucu tarafında ele alınması gereken güvenlik konulu bir iş mantığını istemeden istemci tarafına taşımak zorunda kalmış oluruz. Tabi en büyük sıkıntılardan birisi şudur. Sunucu tarafında bu rol kontrolünü başarabilsek dahi, istemci tarafında gönderilecek entity örneklerinin dinamik olarak değişebiliyor olması gerekecektir. Oldukça zor bir işlem aslında...

Anlaşılan bu konuda **WCF RIA Services** tarafında bir eksiklik var. Bende en ideal çözüm için araştırmalarıma devam ediyorum. Bakalım lambanın üzerine koyduğumuz şapkadaki deliği kapatabilecek miyiz? Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WCF RIA Services - Custom Authorization [Beta 2] (2010-01-26T10:52:00)

wcf ria services,.net ria services,wcf,wcf eco system,

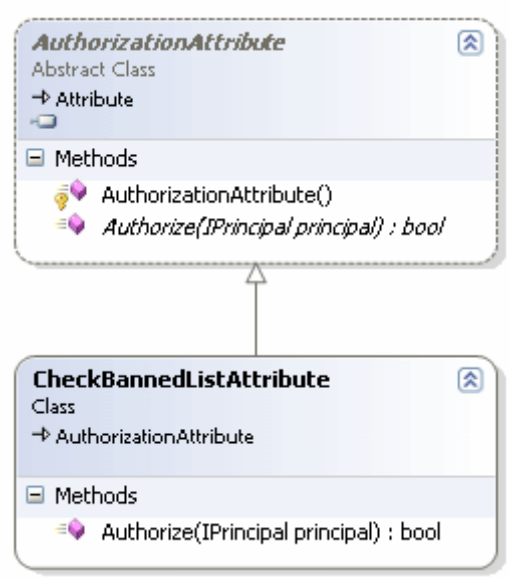
 Merhababa Arkadaşlar,

Geçtiğimiz günlerde uzun süredir yemediğim şu meşhur Dunkin & Donuts' tan bir iki kurabiye almak istedim. Ansızın gelen bu dayanılmaz istek üzerine oturduğumuz semte en yakın dükkanına gidip hem kendim hemde eşim için bir kaç tane aldım. Sonrası malum...Yanında güzel bir kahve ve harika bir tat...Tattıları afiyetle mideye indirdikten sonra evde sessiz ve sakin bir ortamın olduğunu farkettim. Bizim azman ufaklık uyumuş ve gıkı bile çıkmıyorken, yorduğu eşim divanda mışıl mışıl sızmıştı. Herkesin böyle huzurlu bir ortamı hakkettiğini düşünürken, yağmur damlalarının cama vuruşunu izliyordum. Derken ansızın bir ilham geldi ve bloğuma bir şeyler yazmamın iyi olacağı kanısına

vardım. Nede olsa gerekli glikoz yüklemesi fazlasıyla yapılmıştı. Günlüğüme yazılacaklar listeme baktığımda, sıradaki konunun **WCF RIA Service'** lerinde **özel yetkilendirme niteliklerinin(Custom Authorization Attributes)** nasıl yazılacağının anlatılması olduğunu farkettim. Neyseki hafta içi bu konu ile ilişkili olaraktan internetteki az sayıda kaynaktan bilgi edinmiştim. Tabiki en güncel ve geçerli kaynak her zamanki gibi **MSDN'** di.

Hatırlayacağınız gibi bir önceki yazımızda niteliklerden yararlanarak yetkilendirme işlemlerinin nasıl yapılabileceğini basit bir örnek üzerinden incelemeye çalışmıştık. Buna göre önemli olan nokta, **nitelik(Attribute)** yardımıyla çalışma zamanının nasıl davranacağının belirlenebilmesidir. Ancak bazen, **yetkilendirme(Authorization)** kontrolü için özel durumların ele alınması da gerekebilir. Nitekim sadece istemcinin içinde bulunduğu role göre karar vermenin dışında yapılması gereken yetki kontrolleri söz konusu olabilir. Böyle bir durumda çalışma zamanının anlayacağı kendi niteliklerimizi yazmamız gerekmektedir. çok doğal olarak bu geliştirme işleminde yazılacak olan nitelik tipinin, çalışma zamanının anlayacağı ve kullanacağı bir veya daha fazla operasyonu uygulaması şarttır. Burada tipik olarak, çalışma zamanının değerlendireceği fonksiyonellikleri barındıran bir ata nitelik sınıfından yapılacak olan **türetme(Inherit)** işleminden bahsettiğimizi ifade edebiliriz. Dilerseniz konuyu daha iyi anlamak için bu yazımızda geliştireceğimiz örnek senaryomuzdan kısaca bahsedelim.

Senaryomuza göre **Domain Service** içerisinde tanımlanmış olan herhangibir operasyonun gerçekleştirilmesi sırasında, talepte bulunan kullanıcının içerisinde bulunduğu role değil, adının özel olarak tutulan yasaklı bir listede bulunup bulunmadığına bakılması durumu ele alınmaktadır. Bu yasaklı listenin **ASP.NET Membership** tarafından hazır olarak tutulan veritabanına extend edilmiş bir tablo içerisinde tutulması mümkündür. Bunun dışında dosya tabanlı olarak **XML** veya basit **Text** formatında dahi tutulabilir. Hatta olayı biraz abartıp **Windows Registry** ayarlarında dahi tutulması ve benzer saklama alanları düşünülebilir. Hatta bu listenin bir uzak sunucuda duruyor ve ancak servis bazlı bir operasyon yardımıyla kontrollerin yapılabiliyor olması da söz konusu olabillir. Tabiki dosya tabanlı olan saklama şekli bu seçenekler arasında en az güvenli olanıdır. Nitekim dosyanın güvenliğini sağlamak, veritabanı içerisindeki bir tabloya göre çok daha zor olabilir. Ancak amacımız yetkilendirme işlemi için özel nitelik yazılması ve kullanılması olduğundan bebek adımlarıyla ilerleyeceğiz. Bu nedenle geliştireceğimiz örnekte basit olarak yasaklı listenin bir **Text** dosyada düzenli olarak tutulduğunu varsayıyor olacağız. Buna göre **System.Web.DomainServices.AuthorizationAttribute**niteliğinden türettiğimiz tipin içerisinde yer alan **Authorize** metodu içerisinde gelen kullanıcı adının, yasaklı listede olup olmadığını kontrol etmemiz yeterli olacaktır. İşte sunucu tarafında yer alan **CheckBannedListAttribute** sınıfı içeriğimiz.




```
using System.IO;
using System.Linq;
using System.Web;
using System.Web.DomainServices;

namespace ChinookCustomAuthorization.Web
{
   public class CheckBannedListAttribute
      :AuthorizationAttribute
   {
      public override bool Authorize(System.Security.Principal.IPrincipal principal)
      {
         string filePath = HttpContext.Current.Server.MapPath("~\\BannedList.txt");
         string[] bannedList=File.ReadAllLines(filePath);
         return !bannedList.Contains(principal.Identity.Name);
      }
   }
}
```

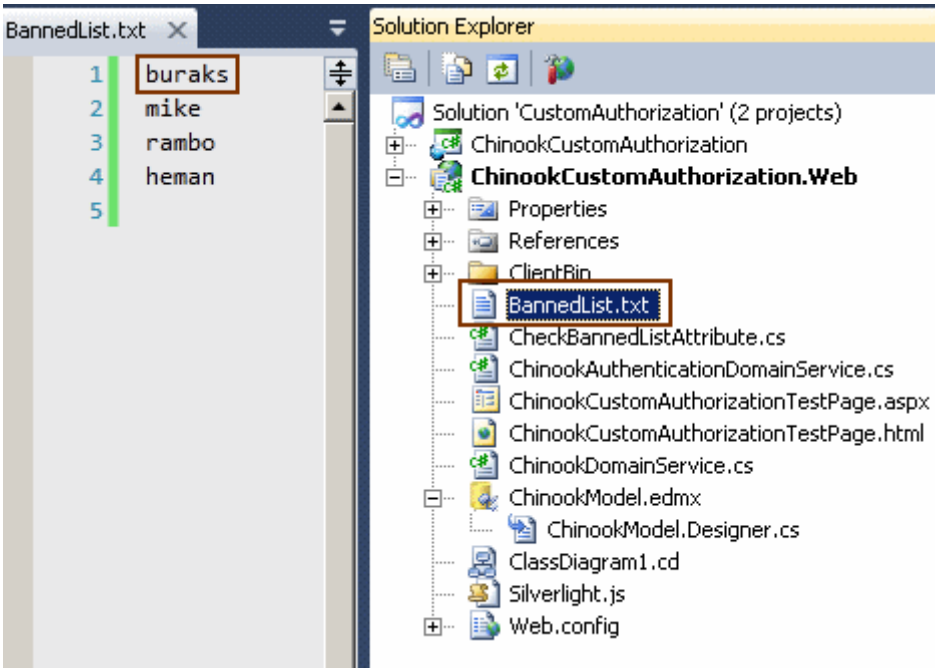
CheckBannedListAttribute sınıfı **AuthorizationAttribute** tipinden türemektedir. Buna göre sınıf diagramı görüntüsünden de fark edileceği üzere, **abstract** olan **Authorize** metodunu ezmek zorundadır. **Authorize** metodu, niteliğin uygulanacağı metodlar için gerekli yetki kontrolü operasyonunu üstlenmektedir. Dikkat edileceği üzere **Authorize** metodu parametre olarak tanıdık bir arayüzü almaktadır; **IPrincipal**. Bu arayüze gelen çalışma zamanı referansından yararlanarak sisteme giriş yapan kullanıcının adını, hangi rolde yer aldığını, doğrulanıp doğrulanmadığını öğrenebiliriz. Biz örnek senaryomuza göre **Login** olan kullanıcı adının yasaklı listenin tutulduğu **text** tabanlı dosya içerisinde olup olmadığını incelemeyi hedefliyoruz. Bu sebepten **Authorize** metodu içerisinde **BannedList** isimli ve sunucu proje içerisinde tutulan text tabanlı dosyanın tüm satırlarının yüklenmesi ve elde edilen dizi

içerisinde olup olmadığına göre **bool** tipte bir sonucun döndürülmesi söz konusu. Yazmış olduğumuz özel nitelik tipini sunucu tarafında yer alan **ChinookDomainService** sınıfı içerisinde uygulamamız ise son derece kolay.

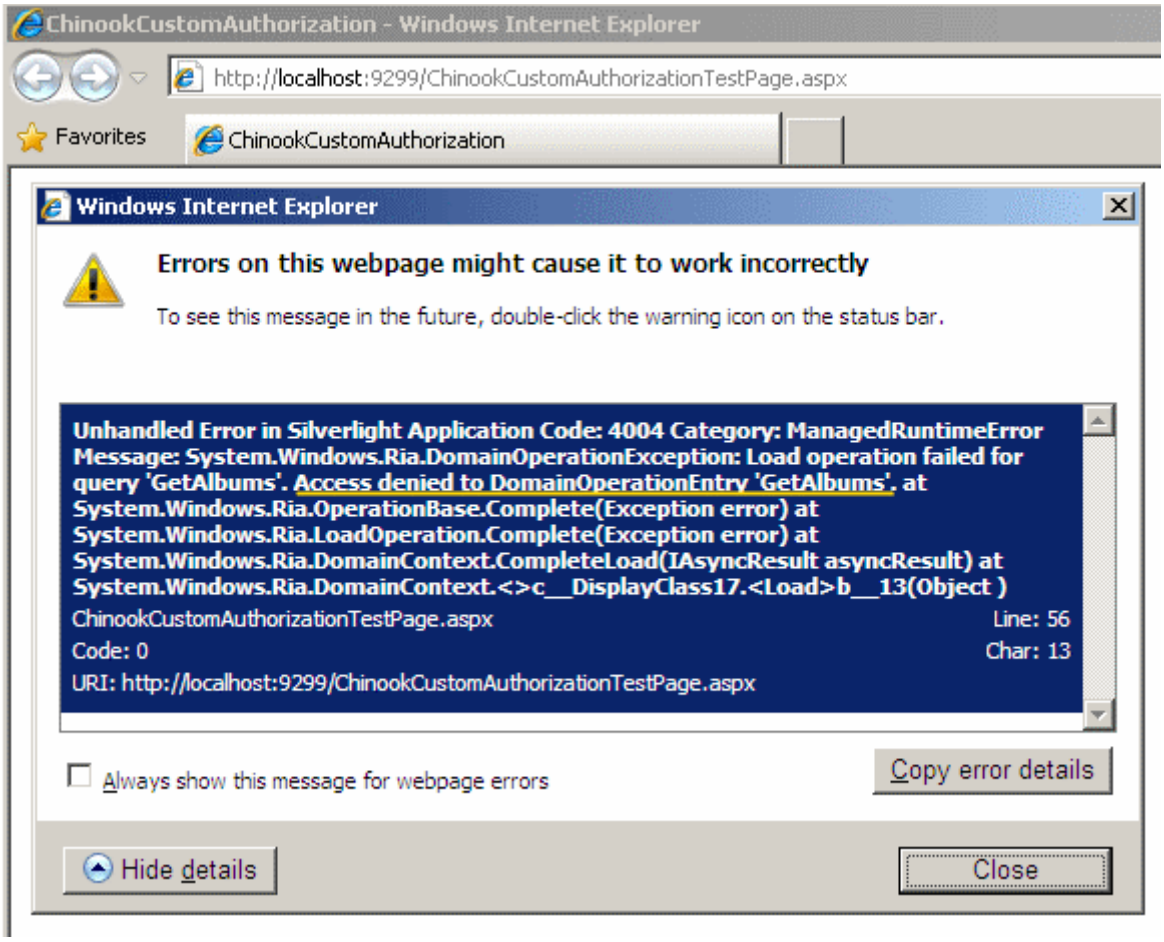
Kişisel Not :*örneğimizin kalan kısmı daha önceki yazımızda geliştirdiğimiz ile benzer. Bu nedenle detaya girerek konudan uzaklaşmamayı tercih etmekteyim.*

```
namespace ChinookCustomAuthorization.Web
{
   using System.Linq;
   using System.Web.DomainServices;
   using System.Web.DomainServices.Providers;
   using System.Web.Ria;


   [RequiresAuthentication]
   [EnableClientAccess()]
   public class ChinookDomainService
      : LinqToEntitiesDomainService<ChinookEntities>
   {
      [CheckBannedList]
      public IQueryable<Album> GetAlbums()
      {
         return this.ObjectContext.Albums;
      }
   }
}
```

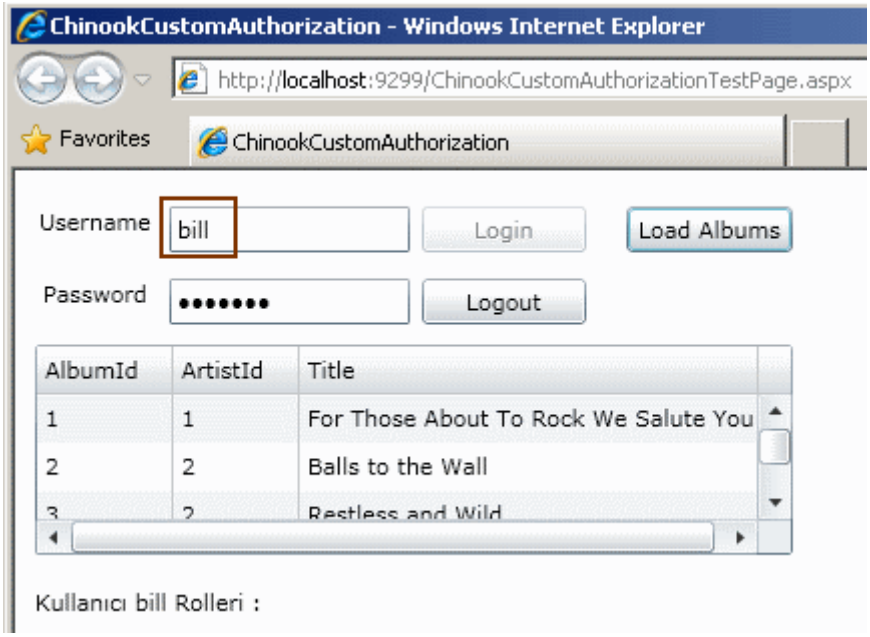
Görüldüğü gibi **CheckBannedList** niteliği, yasaklı liste yetki kontrolü yapılmak istenen operasyonun üzerinde uygulanmaktadır. Buna göre çalışma zamanında **Login** olan bir kullanıcının söz konusu **GetAlbums** operasyonunu talep etmesi halinde devreye girecektir. Söz gelimi **text** dosyamız içerisinde aşağıdaki isimlerin yer aldığını varsayalım.



Buna göre örneğin **buraks** isimli kullanıcı ile **Login** olup albüm listesinin yüklendiği operasyonu talep ettiğimizde, tarayıcı uygulama üzerinde aşağıdaki script hatasını aldığımızı görürüz.



Görüldüğü üzere **Access Denied** kelimeleri hata mesajı içerisinde yer almaktadır. çok doğal olarak yasaklı liste içerisinde yer almayan bir kullanıcı ile sistem girdiğimizde*(örneğin senaryomuza göre bill olabilir)* albüm listesinin başarılı bir şekilde elde edildiği görülecektir. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.



Sonuç olarak eklediğimiz özel **authorization** niteliği yardımıyla, **Domain Service** üzerinden çağırılacak bir operasyon için, sisteme giriş yapan kullanıcının rolüne bakmatan farklı bir yetkilendirme kontrolü gerçekleştirebildiğimizi görmüş olduk. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

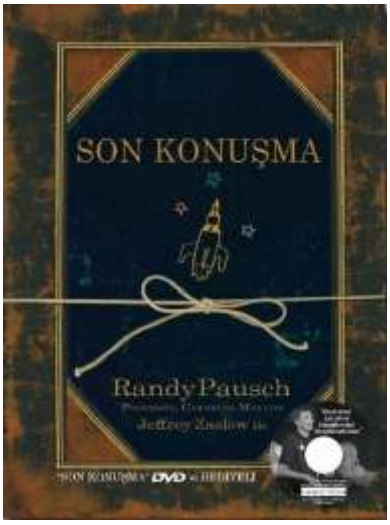
CustomAuthorization.rar (1,29 mb)

Karlı Kış Günü Reçetem (2010-01-24T20:20:00)

randy pausch,the last lecture,stock xchng,chikenfoot,feedreader,the art of unit testing,star wars,avatar,opml,

 Malum bu hafta sonu hepinizin bildiği üzere tüm yurdu karlı ve sert rüzgarlı bir hava etkisi altına aldı. Yarı yıl tatiline giren çocuklar için mutluluk verici bir olay olmakla birlikte, benim gibi böylesine karlı, buzlu, tipi şeklinde rüzgarlı bir havada dışarı çıkmayı tercih etmeyen/üşenen ve sıcak bir ortamda olduğuna, elinin altında bir sürü imkan bulunduğuna şükredenlerin olduğuna da eminim.

Peki bir yazılımcı olarak böyle bir günde neler yapılabilir? Gerekli malzemelerimizin başında elbetteki sağlam bir internet bağlantısı olmasında yarar var. Ancak tüm vaktimizi internette geçireceğimizi de söylemiyorum elbette. Buna ilaveten çay, kahve gibi bilimum içeçeklerimiz de varsa eğer süper. Deymeyin keyfimize...

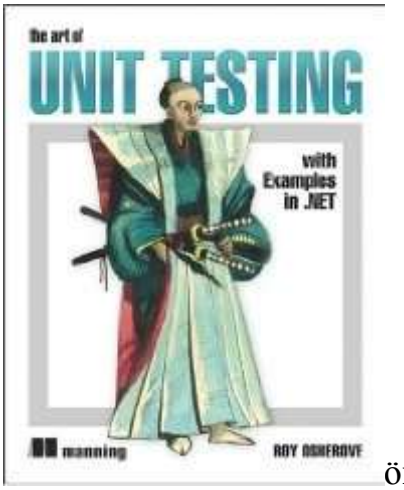


Sizde güne benim gibi son okuduğunuz kitaba devam etmek isteyerek başlayabilirsiniz. İşlerin yoğunluğundan uzun zaman önce tedarik ettiğim ama bir türlü bitiremediğim sevgili **Randy Pausch'** un **Son Konuşma** isimli kitabı örneğin...**Gerçekten çocukluk Hayallerinizi Gerçekleştirmek** ile ilgili olan bu kitabın büyük çoğunluğunu okumuştum ama kalanını tamamlamak için bu hava süper bir fırsat oluşturmuştu. Odanın camını iyice açıp içeriye dolan bembeyaz parlak ışık ve etraftaki derin sessizlik, kitap okumak için iyi bir neden sunuyordu. Henüz aranızda bu kitabı okumamış olanlar var ise mutlak suretle tedarik etmelerini şiddetle öneririm. Hayata bakış açınızın gerçekten değişeceğine emin olabilirsiniz. Kitabın kalan kısmınıda tamamladıktan sonra yüzümdeki mutluluğun bir kaç dakika gitmediğini çok iyi hatırlıyorum.

Peki ya sonra. Biraz bilgisayarın başına geçip internetten yararlanarak çevrede olan biteni takip etmek gerekebilir. Dünyadan ve yurdumuzdan ana haber başlıklarına bakılır. özellikle ve ağırlıklı olarak haftanın ilk iş gününde hava durumunun nasıl olacağı öğrenilerek işe gitmek için uygun güzergah ve vasıtalar seçilir, yetkililerin uyarılarına kulak verilir. Sonrasında belki günün anlam ve önemine dair **Windows 7** masa üstümüz için bir sürü duvar kağıdı resmine bakılabilir. Bu konuda sıkça kullandığım ve sayısız duvar kağıdını indirdiğim bir site var. örneğin duruma uygun olarak **snow, snowman** veya tezat olarak **sea, sundown** gibi kelimeleri aratıp sonuçlarına bakabilirsiniz. Tamamen serbest üyelik gerektiren sitede, kullanmak istediğiniz resimler için lisans bilgileri de yer almakta ve çoğu serbest kullanıma açık. Tabi en iyi olanları için belirli bir ücret ödemeniz gerekiyor.

Şimdi de beki biraz müzik dinlenebilir. Ses kalitesi iyi olan **MP3** çalarınız ile orjinal olarak satın aldığınız **lisanslı Müzik CD**'nizden **MP3** çalarınıza aktardığınız bir albüm pekala dinlenebilir. örneğin **NedirTv?com** adına hazırladığım pek çok **görsel derste(Screencast)** giriş kısmında çaldırdığım intro' ları içeren ve çok sevdiğim ünlü gitarist **Joe Satriani'** nin de dahil olduğu **Chikenfoot** grubunun ilk albümü. Zaten bir albümü baştan sona dinlemek size gün için gerekli enerjiyi verecektir diye düşünüyorum. Tabi müzik dinleme olayına bilgisayarınız başındayken de devam edebilirsiniz. Ancak müziği dinlerken bilgisayar ekranına değil, dışarıdaki karlı manzaraya bakmak ve derin düşüncelerde yol almak daha güzel olabilir.

Bilgisayar başındayken çok sık yaptığım ve hatta iş yerine geldiğimde her sabah düzenli olarak icre ettiğim işlerden biriside takip ettiğim **internet günlüklerine(blog)** eklenen yeni girdileri okumaktır. Burada yardımcı bir programdan faydalanabilirsiniz. örneğin ben müzik eşliğinde **FeedReader** programını kullanarak feedreader.opml (19,77 kb) dosyasındaki **OPML**içeriğinde yer alan günlükleri takip ederim. Bu tip **feed** okuma programları sayesinde takip ettiğiniz pek çok alan ile ilişkili bilgilere gazete okuyormuşçasına ulaşabilirsiniz.



önemli bir kaç günlük girdisi mutlaka dikkatinizi çekecektir. Eğer yetmiyorsa her zaman takip ettiğiniz bir yazılım kitabı baş ucunuzda olmalıdır. Böyle karlı bir hafta sonu için sizi çok fazla sıkmayacak, kalın olmayan, anlatımı düşünce yapınıza uygun bir kitabın tercih edilmesinde yarar vardır. Şahsen ben bu hafta sonu **The Art of Unit Testing with Examples in .NET** isimli kitabı tercih ettim. Bildiğiniz üzere test, yazılım geliştirme süreçlerinde çok önemli bir yere sahip. Geliştirdiğimiz ürünün 0 hata ile çıkmasından, geliştirme süreci için belirtilen sürelere uyulabilmesine kadar pek çok noktada test tekniklerinin uygulanışına rastlamaktayız. Hatta **Test Driven Development** isimli bir geliştirme süreci dahi bulunmakta. Ancak işin ana fikri her zaman için fonksiyonel birimlerin testiye başlanmasında yatıyor. Bu **320** sayfalık kitapta basit birim testi geliştirilmesinden, mock nesnelerin kullanımına kadar pek çok bilgi yer almakta. En azından bir bölümünü okuyup bilgisayar başında test etmekte yarar var.

E birazda sosyalleşmek lazım. Tabi interaktif dünyada yaşayan biz yazılımcılar için sosyalleşme genellikle **Facebook** veya**Twitter** gibi alanlar anlamına geliyor. Bu şekilde davranabileceğimiz gibi, bir kaç kat giyinip bize en yakın **AVM** sinemasına gidebilir ve**Avatar'** ı seyretmeye çalışabiliriz. Ama ben **70'** lin ortasında doğmuş ve **80'** li yıllarda çocukluğunu geçirmiş birisi olarak**Avatar'** a gidinceye kadar herkesin önce bir **Star Wars** serisini tamamlamasını öneririm. Bu nedenle dışarı çıkıp **Avatar'** a gitmektense, **CNBC-E** de bugün saat 22:00' da başlayacak olan efsaneye kadar **StarWars** sitesinde gezinmeyi ve bunu yaparken de**Soundtrack** albümünü dinlemeyi tercih ederim.

 Belki de tamamen yanlış düşüncelere bürünmüş bir yol içerisindeyimdir. Belki de yavaş yavaş Dark Side' a geçiyorumdur. O yüzden tüm bunları bir kenara bırakıp kardan adam yapmayı da tercih edebileceğinizin farkındasınızdır. Belki yandaki resimdeki kadar etkili bir kardam adam olmayacaktır yaptığınız ama mahallenin küçükleri ile geçireceğiniz ve sizi çocukluğunuza götürecek o eşsiz dakikaların değeri de paha biçilemezdir.

WCF RIA Services - Authentication Domain Service - Attribute Bazlı Yetkilendirme [Beta 2] (2010-01-22T09:45:00)

wcf ria services,.net ria services,wcf,wcf eco system,

Merhaba Arkadaşlar,

Bildiğiniz üzere bir süredir **WCF RIA Service'** lerinde **doğrulama(Authentication), yetkilendirme(Authorization), Role** ve **Profile** yönetimi konularına değinmekteyiz. **WCF RIA Service'** lerinin temel amaçlarından birisininde **RIA** tipindeki uygulamalar için **Ado.Net Entity Framework** gibi kaynaklar üzerinden **CRUD(CreateReadUpdateDelete)** operasyonlarını sağlanması olduğu düşünüldüğünde, servis fonksiyonelliklerinin yetkilendirilmeside güvenlik açısından önem arz eden konuların başında gelmektedir. Bu konu, **WCF RIA Service'** lerinde **nitelikler(Attributes)** yardımıyla ele alınabilmektedir.

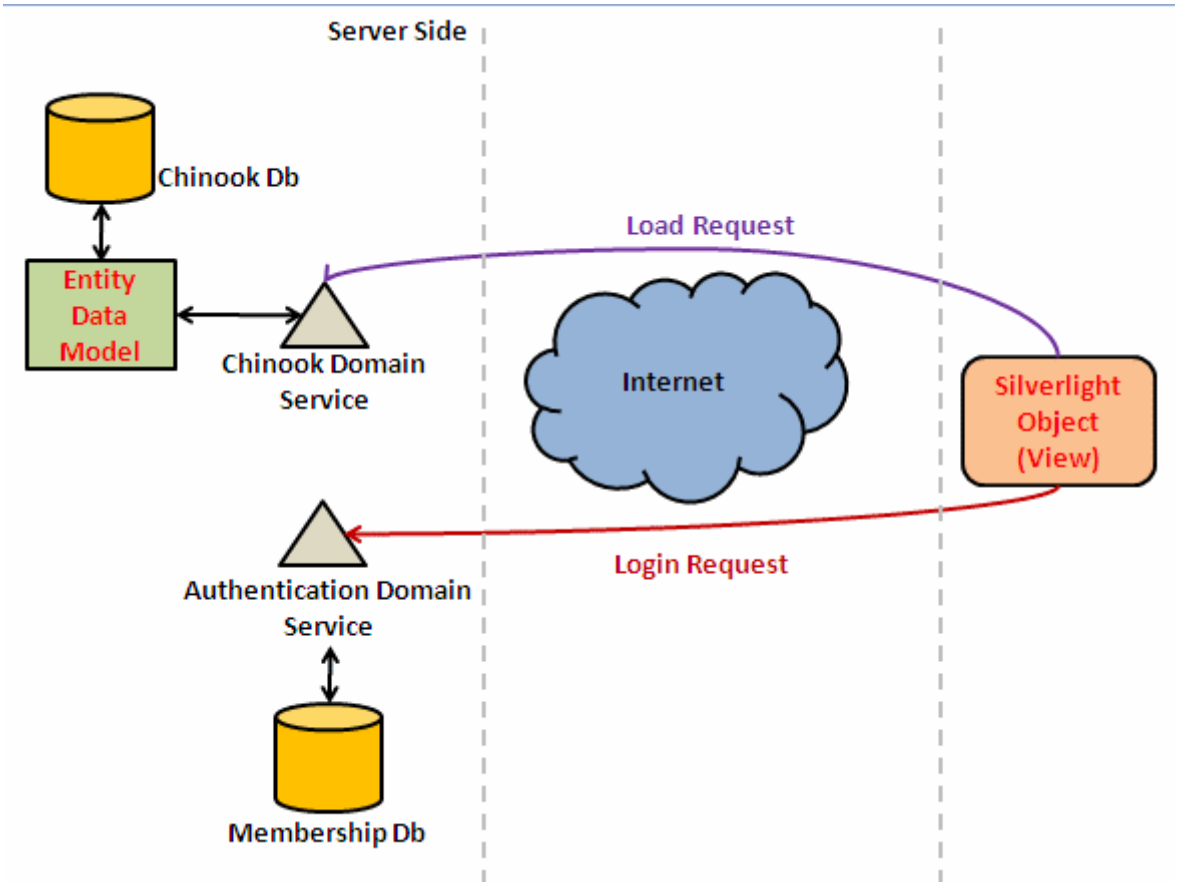
Bu noktada iki önemli niteliğin olduğunu söyleyebiliriz. Bunlardan birisi **Domain Service** sınıfının **Authentication** işlemleri çerçevesinde değerlendirilmesi gerektiğini çalışma zamanına söyleyen **RequiresAuthentication** niteliğidir. Bu

niteliği **Domain Service** tipine uygulamak yeterlidir. Diğer taraftan**, Domain Service**içerisinde tanımlanmış olan operasyonların hangi yetkiler altında çalıştırılabileceğini belirtmek için **RequiresRole** isimli nitelikten yararlanılmaktadır.**RequiresRole** niteliği **string** tipinde birden fazla parametre alabilmektedir. Bu parametre değerleri tahmin edileceği üzere rolleri ifade etmektedir. Bu teoriye göre bir **Domain Service** operasyonunun rol bazlı olaraktan yetki altında çalıştırılmasının sağlanması mümkündür.

Tabi teoride bahsettiğimiz bu kavramları pratiğe dökmemiz bizim için önemlidir. Bu nedenle yazımızın bundan sonraki kısmında basit olarak Chinook veritabanındaki kobay tablolarımızdan olan **Album** içeriğine ulaşmak için kullanılan bir operasyon üzerinde, rol bazlı yetkilendirme işlemlerini nasıl uygulayabileceğimizi incelemeye çalışacağız.

Kişisel Not : *Başlamadan önce* ***Silverlight*** *uygulamasında daha önceki yazılarda sıklıkla anlattığımız şekilde* ***Form bazlı doğrulama(Form-Based Authentication)*** *için gerekli ayarları yapmamız gerektiğini hatırlatalım. özellikle* ***role*** *yönetimini etkinleştirmemiz gerektiğini ve hem sunucu hemde istemci tarafında gerekli konfigurasyon ve kod ayarlarını uygulamamız gerektiğini unutmayalım. örneğimizde yine* ***buraks*** *ve* ***bill*** *isimli kullanıcıları değerlendiriyor olacağız. Bunlardan birisi* ***Employee*** *rolünde iken diğeri* ***Finance****rolündedir. Amaçlanan sadece* ***Finance*** *rolündekilerin, albüm listesini çekebilmesini sağlamaktır. Tabiki* ***Chinook*** *isimli veritabanını* ***Silverlight*** *uygulamasında kullanabilmek için gerekli* ***Domain Service(ChinookDomainService)*** *öğesinide eklememiz gerektiğini hatırlatmak isterim. Pek çok hatırlatmada bulundum ama önceki yazıları takip edip uygulayan arkadaşlarımız için örneği bu aşamaya getirmek son derece kolay olacaktır düşüncesindeyim* 

Gelelim örneğimize. İlk olarak durumu kuş bakışı değerlendirmeye çalışalım. Buna göre aşağıdaki şekli göz önüne alabiliriz.



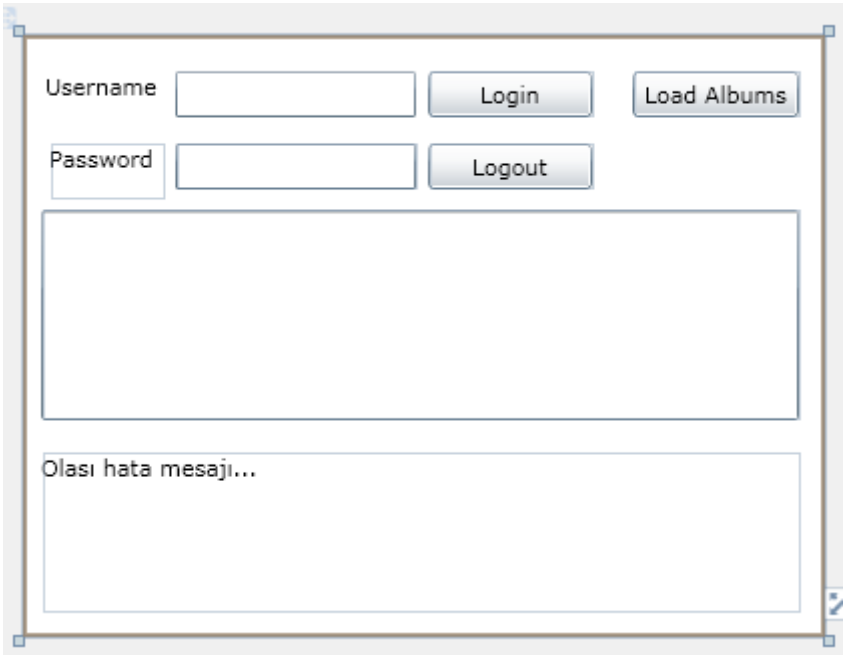
Şekildende görüleceği üzere sunucu uygulama tarafında **Chinook** veritabanına ulaşılmasını ve üzerinde **CRUD** işlemleri yapılabilmesini sağlayan*(ki bu örnekte sadece veri çekiyor olacağız)* **ChinookDomainService** isimli **Domain Service** tipi bulunmaktadır. Bu tip arada **Ado.Net Entity Framework** modelini kullanmaktadır. Diğer taraftan doğrulama işlemleri için **ASP.NET Membership** alt yapısı kullanılmakta olup istemci tarafının bu hizmeti değerlendirebilmesi için birde **Authentication Domain Service(ChinookDomainService)** öğesi yer almaktadır. İstemci tarafı veri ve güvenlik işlemleri için bu iki servisten yararlanacaktır. önemli olan noktalardan biriside hangi operasyonu nasıl yetkilendireceğimizi bilmektir. Bu noktada **ChinookDomainService** sınıfını aşağıdaki şekilde oluşturduğumuzu göz önüne alalım.

```
namespace SilverlightApplication8.Web
{
    using System.Linq;
    using System.Web.DomainServices;
    using System.Web.DomainServices.Providers;
    using System.Web.Ria;

    [RequiresAuthentication] // Servis operasyonlarında Authentication uygulanacağını belirtiyoruz.
    [EnableClientAccess()]
    public class ChinookDomainService : LinqToEntitiesDomainService<ChinookEntities>
```

```
    {
        [RequiresRole("Finance")] // Sadece Finance rolündekilerin aşağıdaki operasyonu kullanabileceğini belirtmekteyiz.
        public IQueryable<Album> GetAlbums(int artistId)
        {
            // örnek olarak ArtistId bilgisine göre albüm listesini Title bilgisine göre A...Z sırasında döndürüyoruz
            return from a in ObjectContext.Albums
                   where a.ArtistId == artistId
                   orderby a.Title
                   select a;
        }
    }
}
```

Dikkat edileceği üzere **ChinookDomainService** tipine **RequiresAuthentication** niteliği uygulanmıştır. Bununla birlikte sadece **Finance** rolündekilerin kullanımına sunulan **GetAlbums** isimli bir operasyon yer almaktadır. Yazımızın başında da belirttiğimiz üzere **RequiresRole** niteliğine parametre olarak birden fazla rol adı verilebilir. Sunucu tarafında rol bazlı yetkilendirme için yapmamız gerekenler sadece bu kadardır. Gelelim istemci tarafına. Bu amaçla **MainPage.xaml** içeriğini ve kod kısmını aşağıdaki gibi geliştirdiğimizi düşünelim.

MainPage.xaml;




```
<UserControl x:Class="SilverlightApplication8.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="300" d:DesignWidth="400" xmlns:data="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
xmlns:dataInput="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.Input">

    <Grid x:Name="LayoutRoot" Background="White">
        <Button Content="Load Albums" Height="23" HorizontalAlignment="Left"
Margin="305,18,0,0" Name="btnLoadAlbums" VerticalAlignment="Top" Width="83"
Click="btnLoadAlbums_Click" />
        <data:DataGrid AutoGenerateColumns="True" Height="105"
HorizontalAlignment="Left" Margin="10,87,0,0" Name="grdAlbums"
VerticalAlignment="Top" Width="378" />
        <dataInput:Label Height="80" HorizontalAlignment="Left" Margin="10,208,0,0"
Name="lblStatus" VerticalAlignment="Top" Width="378" Content="Olası hata mesajı..."
/>
        <dataInput:Label Height="28" HorizontalAlignment="Left" Margin="12,18,0,0"
Name="label1" VerticalAlignment="Top" Width="59" Content="Username" />
        <TextBox Height="23" HorizontalAlignment="Left" Margin="77,18,0,0"
Name="txtUsername" VerticalAlignment="Top" Width="120" />
        <dataInput:Label Height="28" HorizontalAlignment="Left" Margin="14,54,0,0"
Name="label2" VerticalAlignment="Top" Width="57" Content="Password" />
        <PasswordBox Height="23" HorizontalAlignment="Left" Margin="77,54,0,0"
Name="txtPassword" VerticalAlignment="Top" Width="120" />
        <Button Content="Login" Height="23" HorizontalAlignment="Left"
Margin="203,18,0,0" Name="btnLogin" VerticalAlignment="Top" Width="82"
Click="btnLogin_Click" />
        <Button Content="Logout" Height="23" HorizontalAlignment="Left"
Margin="203,54,0,0" Name="btnLogout" VerticalAlignment="Top" Width="82"
Click="btnLogout_Click" />
    </Grid>
</UserControl>
```

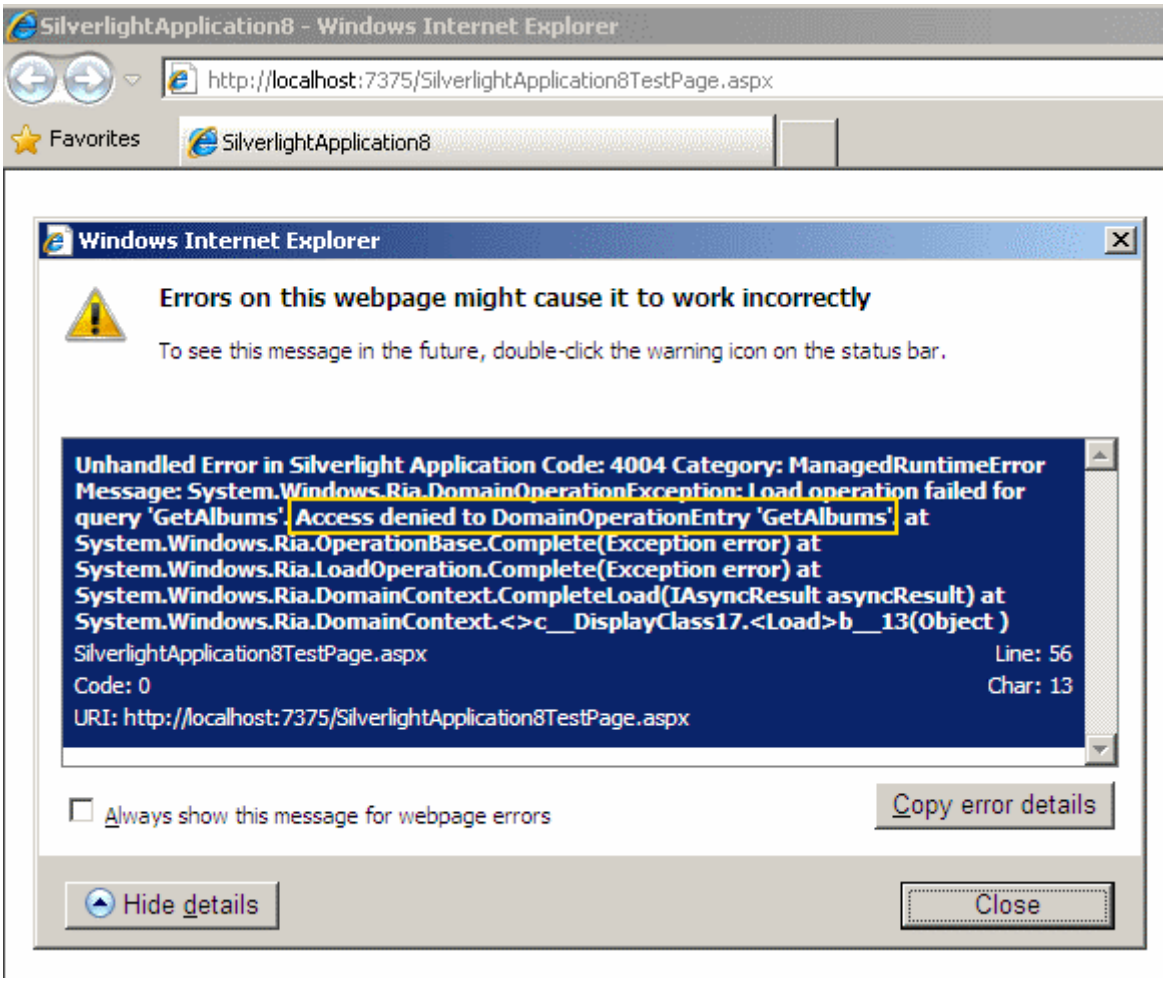
MainPage.xaml.cs;

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Ria;
using System.Windows.Ria.ApplicationServices;
using SilverlightApplication8.Web;
```

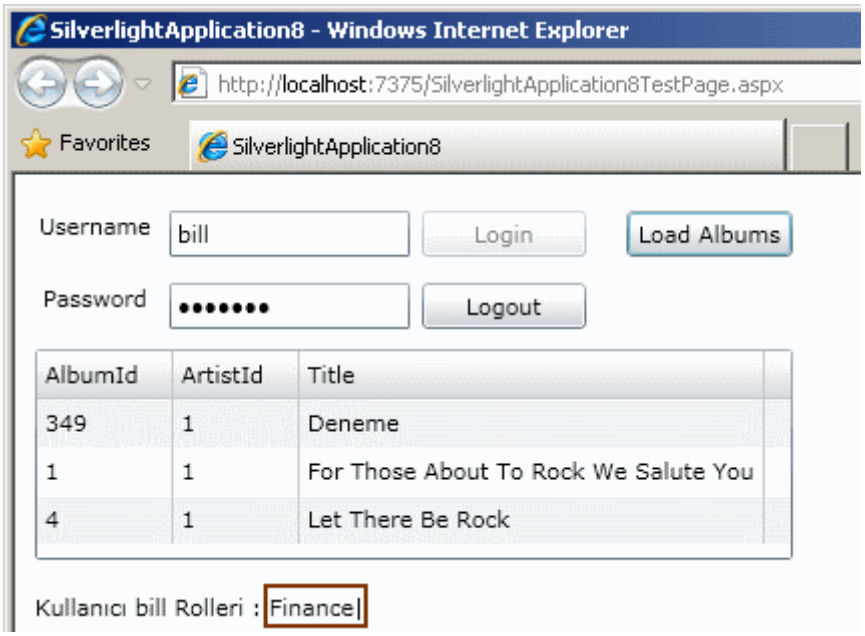
```
namespace SilverlightApplication8
{
   public partial class MainPage : UserControl
   {
      // DomainContext ve AuthenticationService kullanımı için gerekli örnekler
      ChinookDomainContext context;
      AuthenticationService authSrv;

      public MainPage()
      {
         InitializeComponent();

         btnLoadAlbums.IsEnabled = false;
         btnLogout.IsEnabled = false;

         context = new ChinookDomainContext();
         authSrv = WebContext.Current.Authentication;

         // Login işlemi olduğunda devreye girecek olay metodu yüklenir
         authSrv.LoggedIn += new
EventHandler<AuthenticationEventArgs>(authSrv_LoggedIn);
         // Logout işlemi tamamlandığında devreye girecek olay metodu yüklenir
         authSrv.LoggedOut += new
EventHandler<AuthenticationEventArgs>(authSrv_LoggedOut);
      }

      private void btnLogin_Click(object sender, RoutedEventArgs e)
      {
         // Login işlemi yapılır
         LoginOperation logOp = authSrv.Login(new
LoginParameters(txtUsername.Text, txtPassword.Password));
      }

      private void btnLogout_Click(object sender, RoutedEventArgs e)
      {
         // Logout işlemi yapılır. Hata var ise exception fırlatılması sağlanır
         authSrv.Logout(true);
      }

      void authSrv_LoggedIn(object sender, AuthenticationEventArgs e)
      {
         // Login olan kullanıcı için güncel User bilgileri alınır
         Web.User currentUser = WebContext.Current.User;
         // Kullanıcının dahil olduğu roller ve adı bilgilendirme amaçlı öğrenilir
         string roles=String.Empty;
```

```
            foreach (var role in currentUser.Roles)
            {
               roles += String.Format("{0}|", role);
            }
            lblStatus.Content =String.Format("Kullanıcı {0} Rolleri :
{1}",currentUser.Name,roles);

            btnLoadAlbums.IsEnabled = true;
            btnLogout.IsEnabled = true;
            btnLogin.IsEnabled = false;
         }

         void authSrv_LoggedOut(object sender, AuthenticationEventArgs e)
         {
            btnLogout.IsEnabled = false;
            btnLoadAlbums.IsEnabled = false;
            btnLogin.IsEnabled = true;
         }

         private void btnLoadAlbums_Click(object sender, RoutedEventArgs e)
         {
            // Login olunduktan sonra Album listesinin yüklenmesi aşamasına geçilir.
            // DomainService üzerinde yer alan GetAlbums operasyonundaki role gerekliliği
nedeni ile sadece Finance rolü için yükleme yapıldığı görülür. Aksi durumda ise bir
çalışma zamanı hatası oluşacak ve script error olarak tarayıcı üzerinde görülecektir.
            LoadOperation<Album> op =
context.Load<Album>(context.GetAlbumsQuery(1));
            grdAlbums.ItemsSource = op.Entities;
         }
      }
}
```

Aslında istemci tarafında yapılan tek şey kullanıcının **Login** ve **Logout** olmasını sağlayacak operasyonlar ile örnek olması açısından 1 numaralı şarkıcıya ait albüm listesinin çekilmesini sağlamaktır. Dikkat edileceği üzere yetkilendirme kontrolü istemci tarafında yapılamamaktadır. Bu kontrol sunucu tarafında çalışma zamanı tarafından ele alınan**Domain Service** sınıfı üzerinden ele alınmaktadır. Buna göre **Login** olan geçerli kullanıcının **Finance** rolünde olmaması halinde albüm bilgilerini getirememesi gerekmektedir ki bu gerçektende böyledir. İşte yetkisiz bir kullanıcının albümleri yüklemek istemesi halinde çalışma zamanında oluşacak durum;



Hata mesajında yer alan **Access Denied** kelimeleri olayı tüm çıplaklığıyla özetlemektedir. Diğer yandan senaryomuza göre **Finance** rolünde yer alan **bill** isimli kullanıcı ile **Login** olunup albüm listesi çekilmek istendiğinde, bilgilerin başarılı bir şekilde **DataGrid** kontrolüne çekildiği gözlemlenir. Aynen aşağıdaki şekilde görüldüğü gibi.



Tabi bu yazımızda yetkilendirme nedeniyle oluşan istisna durumu ele kontrol altına alınmamıştır. Uygulama bu istisna ile karşılaştığında sonlandırılmaktadır ve hata mesajı**script error** olarak tarayıcı uygulama üzerinden yakalanmaktadır. Ancak en basit haliyle **attribute** bazlı olaraktan yetkilendirme işlemi sunucu tarafında yer alan servisler kanalıyla gerçekleştirilebilmiştir. Bu noktada vurgulanması gereken durumlardan biriside kendi **Authorization niteliklerimizi(Attribute)** yazabileceğimizdir. Söz gelimi role göre değil ama başka bir kritere göre yetkilendirme yapmak isteyebiliriz. Bu durumu ilerleyen yazılarımızda ele almaya çalışıyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WCF RIA Services - Authentication Sample.rar (1,31 mb) *[Dosya boyutunun küçük olması için ASPNETDB.mdf içeriği çıkartılmıştır]*

WCF RIA Services - Authentication Domain Service - Profile ve Role (2010-01-21T01:10:00)

wcf ria services,.net ria services,wcf,wcf eco system,

 Merhaba Arkadaşlar,

Yandaki resmin bir renk cümbüşü oluşturup sizlere çok güzel göründüğüne eminim. Hatta bu resmin biraz sonra anlatacağımız konu ile olan ilgisini merak ediyor olabilirsiniz. Ne yazıkki yok. Sadece renk cümbüşünün benide etkilediğini ve yazının hoş görünmesi için eklediğimi itiraf edebilirim. Gelelim asıl mevzumuza.

Bir önceki yazımızda **Authentication Domain Service** konusunu incelemeye başlamış ve **RIA(Rich Internet Application)** çeşitlerinden olan**Silverlight** uygulamalarında **Form** tabanlı doğrulamanın standart **ASP.NET Membership** kaynakları üzerinden nasıl sağlanabileceğini görmüştük. **RIA** uygulamaları ile ilişkili konulardan bir diğeride rol ve profil yönetimidir. **WCF RIA Service'** lerde kullanılan **Authentication Domain Service'** lerden yararlanarak **Role** ve **Profile** yönetimide yapılabilir. çok doğal olarak **WCF RIA Service'** leri, **Asp.Net** mimarisinin rol ve profil alt yapısını kullanmaktadır.

Doğrulanan bir kullanıcının(Authenticated User), sistem içerisinde yapabileceklerini belirlerken rolüne bakılarak karar verilmesi tercih edilen yöntemlerdendir. örneğin **Administrator** rolündeki bir kullanıcı ile **Guest** rolündeki bir kullanıcının sistem içerisinde yapılabilecekleri kuvvetle muhtemel farklıdır. Burada açık bir şekilde role göre yetkinin mertebesinin belirlendiğini düşünebiliriz. Diğer yandan sistem içerisinde yer alan tüm kullanıcılar için ortak tanımlanabilecek özellikler, çalışma zamanında farklı(bazende benzer, hatta aynı) değerler alarak, her kullanıcının sistem için bir profilinin oluşmasında kullanılabilirler. çok doğal olarak bu profil özellikleri sistemden sisteme farklı şekillerde tanımlanabilir ve kullanılabilirler. Söz gelimi**RIA** uygulamasına dahil olan kullanıcıların ünvanları, doğum tarihleri, göz renkleri, cep telefonlarının gsm operatörleri, son giriş zamanları her kullanıcı için birer profil özelliği olarak değerlendirilebilir.

Bu yazımızda bir önceki örneğimizi devam ettirerek rol ve profil özelliklerinin nasıl değerlendirilebileceğini ele almaya çalışıyor olacağız. Temel amacımız rol ve profil bilgilerinin özellikle istemci tarafında nasıl kullanılabileceğini görmek olduğundan çok işe yarar bir örnek geliştirmeyeceğimizi şimdiden belirtmek isterim 😉 öncelikli olarak sunucu uygulama tarafında rol ve profil yönetimi için gerekli ayarları yapmamız gerekiyor. Bir önceki örneğimizde kullandığımız **buraks** ve **bill** isimli kullanıcıları sırasıyla **Developer** ve **Administrator** isimli rollere atadığımızı düşünerek devam

edeceğiz. Hatta testlerimizi daha iyi yapabilmek adına bill isimli kullanıcının her iki rol atlında da bulunması sağladığımızı düşünelim. Bu şekilde istemci tarafına birden fazla rol bilgisinin nasıl aktarıldığını değerlendirme fırsatımız olacaktır. Bildiğiniz üzere rol atama işlemleri için **ASP.NET Configuration** aracını kullanabiliriz. Rol atama işlemlerinin ardından örneğimizde kullanacağımız bir kaç profil özelliğini tanımlayarak devam edebiliriz. Bu amaçla **Web.config** dosyamızın içeriğinde aşağıdaki değişiklikleri yaptığımızı düşünelim.

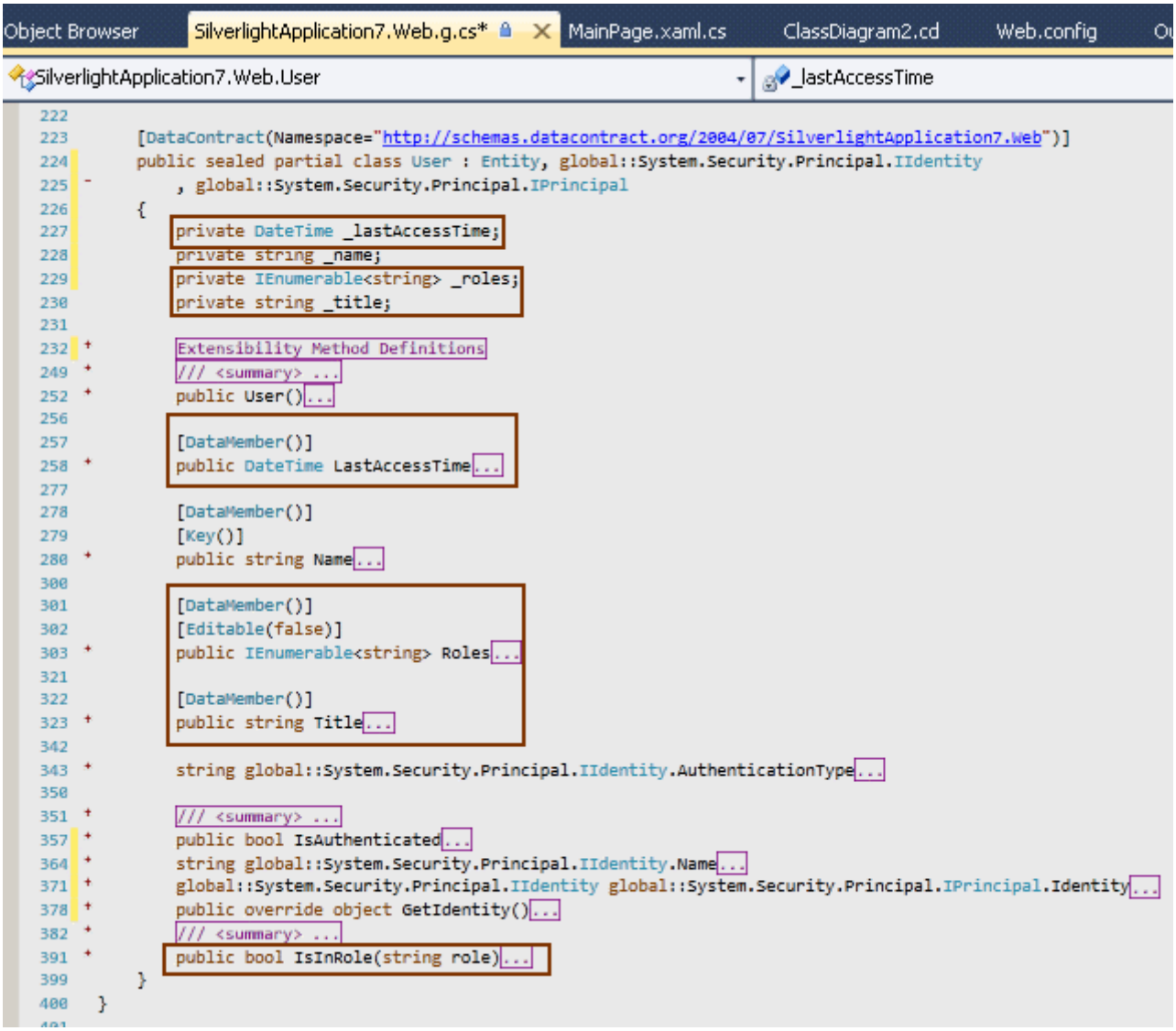
```
<configuration>
  <system.web>
    <httpModules>
      <add name="DomainServiceModule"
type="System.Web.Ria.Services.DomainServiceHttpModule, System.Web.Ria,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
    </httpModules>
    <authentication mode="Forms" />
    <roleManager enabled="true" />
    <profile enabled="true">
      <properties>
        <add name="Title"/>
        <add name="LastAccessTime" type="System.DateTime"/>
      </properties>
    </profile>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>
...
```

roleManager ve **profile** elementleri içerisinde **enabled** niteliklerine **true** değer atandığına dikkat edelim. Bu sayede **RIA** uygulamamızda **role** ve **profile** kullanımını etkinleştirmiş oluyoruz.**profile** elementi içerisinde yer alan **properties** alt elementi içerisinde ise **Title** ve **LastAccessTime** isimli özellik tanımlamalarının yapıldığını görmekteyiz. özellikle **LastAccessTime** niteliği için **DateTime** bildiriminin de yapıldığına dikkat edelim. Nitekim profil özellikleri varsayılan olarak **string** tipindendir. Bu sebepten string harici tipleri bildirmemiz gerekmektedir.

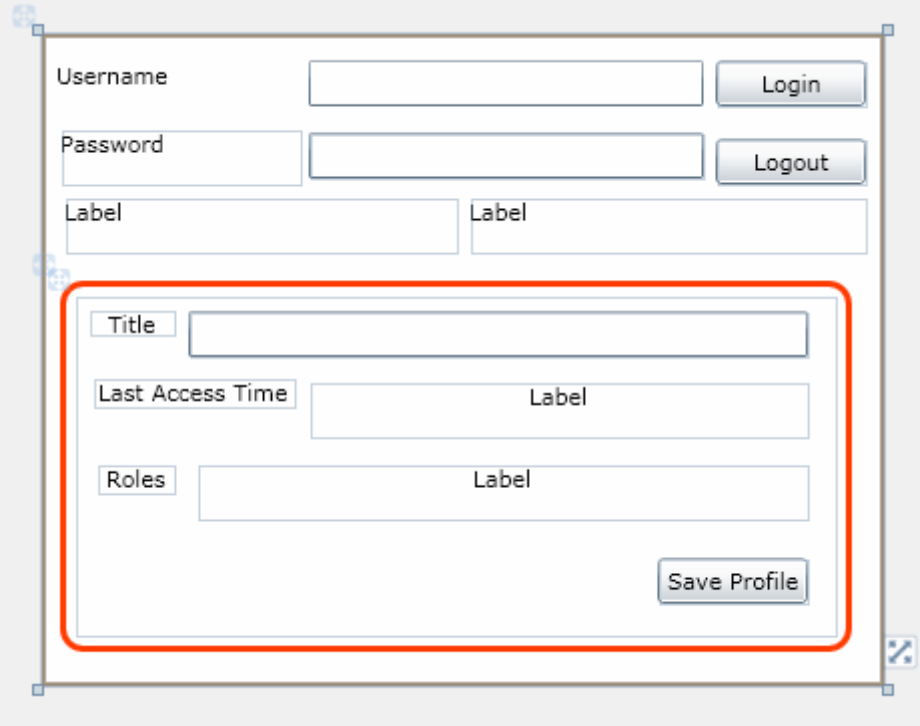
Web.config dosyasında yapılan bu bildirimler **ASP.NET** tarafındaki **role** ve **profile** alt yapıları için gereklidir. Ne varki **RIA** uygulaması tarafında da ilgili profil özelliklerinin kullanımı için gerekli bildirimlerin yapılması gerekmektedir. üstelik kullanıcının rol bilgilerinin istemci tarafında yer alan kod kısmında nasıl ele alınabileceği de şu anda soru işaretidir. Panik ve heyecan yapmadan sakin bir şekilde adım adım ilerleyelim. öncelikli olarak **OurAuthenticationService** ismiyle oluşturduğumuz **Authentication Domain Service** dosyasını açalım ve **User** isimli tipin içeriğini aşağıdaki gibi düzenleyelim.

```
using System;
using System.Web.Ria;
using System.Web.Ria.ApplicationServices;

namespace SilverlightApplication7.Web
{
   [EnableClientAccess]
   public class OurAuthenticationService
      : AuthenticationBase<User>
   {
   }

   public class User
      : UserBase
   {
      public string Title { get; set; }
      public DateTime LastAccessTime{ get; set; }
   }
}
```

Dikkat edileceği üzer **profile** elementi altında tanımlanan özellikler, **User** isimli tip içerisinde de **property** olarak bildirilmiştir. Buraya kadar yaptığımız işlemlerin ardından uygulamayı **build**ettiğimizde, istemci tarafında otomatik olarak üretilen sınıf içerisinde yer alan **User** tipininde aşağıdaki şekilde görüldüğü gibi oluşturulduğunu fark edebiliriz.



Dikkat edileceği üzere sunucu tarafında tanımladığımız **Title**, **LastAccessTime** isimli özellikler için istemci tarafındaki **User** sınıfı içerisinde de gerekli bildirimler yapılmıştır. üstelik rol işlemleri içinde **IEnumerable<string>** tipinden bir özellik(Roles) olduğu görülmektedir. Buna göre birden fazla rolün kod tarafında değerlendirilmesi mümkündür. Hatta **LINQ**ifadeleri ile **Roles** özelliği üzerinden sorgular atılabilir. Artık istemci tarafında rol ve profil işlemlerini değerlendirebilecek alt yapı hazırlıklarını tamamlamış bulunuyoruz. Şimdi**MainPage.xaml** içeriğini aşağıdaki gibi güncelleyerek yolumuza devam edebiliriz.



MainPage.xaml içeriği;

```
<UserControl x:Class="SilverlightApplication7.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="325" d:DesignWidth="420" xmlns:dataInput="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.Input"
xmlns:navigation="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Navigation">
    <Grid x:Name="LayoutRoot" Background="White" Height="325">
        <dataInput:Label Height="28" HorizontalAlignment="Left" Margin="8,14,0,0"
Name="label1" VerticalAlignment="Top" Width="120" Content="Username" />
        <dataInput:Label Height="28" HorizontalAlignment="Left" Margin="10,48,0,0"
Name="label2" VerticalAlignment="Top" Width="120" Content="Password" />
        <TextBox Height="23" HorizontalAlignment="Left" Margin="134,14,0,0"
Name="txtUsername" VerticalAlignment="Top" Width="197" />
        <PasswordBox Height="23" HorizontalAlignment="Left" Margin="134,50,0,0"
Name="txtPassword" VerticalAlignment="Top" Width="197"/>
        <Button Content="Login" Height="23" HorizontalAlignment="Left"
Margin="337,14,0,0" Name="btnLogin" VerticalAlignment="Top" Width="75"
Click="btnLogin_Click" />
        <dataInput:Label Height="28" HorizontalAlignment="Left" Margin="12,82,0,0"
Name="lblLoginStatus" VerticalAlignment="Top" Width="196" />
```

```
        <Button Content="Logout" Height="23" HorizontalAlignment="Left"
Margin="337,53,0,0" Name="btnLogout" VerticalAlignment="Top" Width="75"
Click="btnLogout_Click" />
        <dataInput:Label Height="28" HorizontalAlignment="Left" Margin="214,82,0,0"
Name="lblProcess" VerticalAlignment="Top" Width="198" />
        <Border BorderBrush="#FFFF3B00" BorderThickness="3" Height="185"
HorizontalAlignment="Left" Margin="10,124,0,0" Name="brdProfile"
VerticalAlignment="Top" Width="395" CornerRadius="10" Background="{x:Null}">
            <Canvas Height="170" Name="canvas1" Width="380">
                <dataInput:Label Canvas.Left="6" Canvas.Top="6" Height="13" Name="label3"
Width="43" Content="Title" />
                <dataInput:Label Canvas.Left="8" Canvas.Top="40" Height="15"
Name="label4" Width="101" Content="Last Access Time" />
                <TextBox Canvas.Left="56" Canvas.Top="7" Height="23" Name="txtUserTitle"
Width="309" />
                <dataInput:Label Canvas.Left="116" Canvas.Top="42" Height="28"
Name="lblUserLastAccessTime" Width="249" />
                <dataInput:Label Canvas.Left="10" Canvas.Top="83" Height="15"
Name="label5" Width="39" Content="Roles" />
                <dataInput:Label Canvas.Left="60" Canvas.Top="83" Height="28"
Name="lblRoles" Width="305" />
                <Button Canvas.Left="290" Canvas.Top="130" Content="Save Profile"
Height="23" Name="btnSaveProfile" Width="75" Click="btnSaveProfile_Click" />
            </Canvas>
        </Border>
    </Grid>
</UserControl>
```

Tasarımın biraz fakir olmasına aldırmadan ilerlemeye devam edelim. Senaryomuzu şu şekilde işletiyor olacağız; Kullanıcı Login işlemini başarılı bir şekilde gerçekleştirdiyse eğer, **WebContext.Current.User** özelliğinden elde edeceğimiz kullanıcı bilgilerini **Border** alanı içerisindeki kontrollerde göstereceğiz. Sembolik olarak **Title** ve **LastAccessTime** değerlerini ve dahil olduğu rollerin adlarını değerlendireceğiz. Sonrasında kullanıcı isterse **Save Profile** başlıklı düğmeye basaraktan yeni **Title** ve **LastAccessTime** bilgilerini kaydedebilecek. Bu basit senaryo için bir önceki yazımızda geliştirdiğimiz **MainPage.xaml.cs** içeriğini aşağıdaki gibi güncellememiz yeterli olacaktır.

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Ria.ApplicationServices;

namespace SilverlightApplication7
{
```

```
   public partial class MainPage : UserControl
   {
      // Kullanıcının doğrulanması, profil özelliklerinin yüklenmesi ve kaydedilmesi için
gerekli fonksiyonellikleri sunan tiptir.
      AuthenticationService authSrv = WebContext.Current.Authentication;

      public MainPage()
      {
         InitializeComponent();

         btnLogout.IsEnabled = false;
         brdProfile.Visibility = Visibility.Collapsed;

         authSrv.LoggedIn += new
System.EventHandler<AuthenticationEventArgs>(authSrv_LoggedIn);
         authSrv.LoggedOut += new
System.EventHandler<AuthenticationEventArgs>(authSrv_LoggedOut);
      }

      private void btnLogin_Click(object sender, RoutedEventArgs e)
      {
         lblLoginStatus.Content = String.Empty;
         lblProcess.Content = String.Empty;

         if (!authSrv.IsLoggingIn) // Eğer asenkron olarak devam eden bir Login operasyonu
yoksa
         {
            // Login işlemi için AuthenticationService tipinin Login metodu çağırılır.
            // ilk parametre ile kullanıcı adı ve şifre bilgisi gönderilir. Bu metodun aşırı
yüklenmiş versiyonları mevcuttur.
            // ikinci parametrede Login metodunun işleyişini tamamlaması sonrası devreye
giren metodun işaret edilmesi sağlanmaktadır(Action<LoginOperation> temsilci tipi ile
işaretleme yapılır). Bu metod içerisinde işlemin iptali, exception üretmesi gibi durumlarda
ele alınmaktadır.
            authSrv.Login(
               new LoginParameters(txtUsername.Text, txtPassword.Password),
               opt =>
               {
                  if (opt.IsCanceled)
                     lblProcess.Content = "Login işleminde iptal";
                  else if (opt.Error != null)
                     lblProcess.Content = opt.Error.Message;
                  else if (opt.IsComplete)
                     lblProcess.Content = "Login işlemi tamamlandı";
               }
```

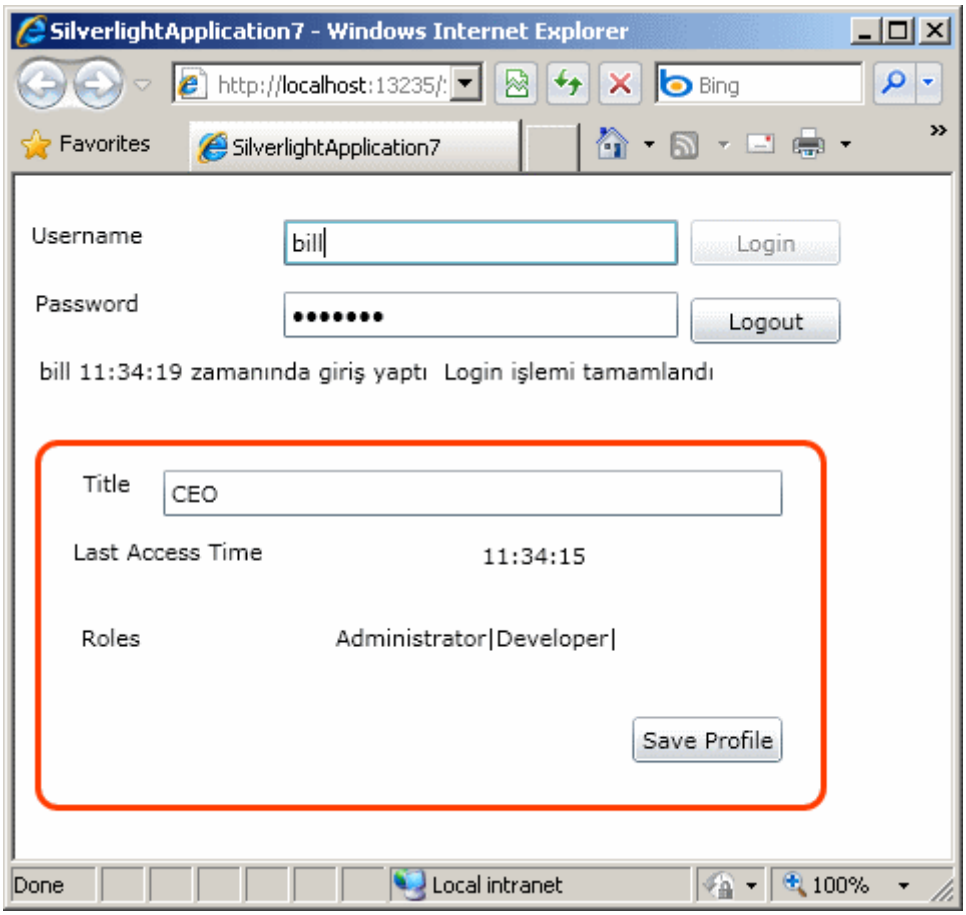
```
                , null
                );
        }
    }

    private void btnLogout_Click(object sender, RoutedEventArgs e)
    {
        lblLoginStatus.Content = String.Empty;
        lblProcess.Content = String.Empty;

        if (!authSrv.IsLoggingOut) // Eğer asenkron olarak devam eden bir Logout
operasyonu yoksa
        {
            // Logout işlemi kullanılan metod çağrısı
            // işlem tamamlandığında devreye girecek olan metod Action<LogoutOperation>
temsilcisi ile işaret edilir.
            authSrv.Logout(
              opt =>
              {
                  if (opt.IsCanceled)
                      lblProcess.Content = "Logout işleminde iptal";
                  else if (opt.Error != null)
                      lblProcess.Content = opt.Error.Message;
                  else if (opt.IsComplete)
                      lblProcess.Content = "Logout işlemi tamamlandı";
              }
            , null);
        }
    }

    // Kullanıcı başarılı bir şekilde Logout olduğunda tetiklenir
    void authSrv_LoggedOut(object sender, AuthenticationEventArgs e)
    {
        lblLoginStatus.Content = String.Format("{0} {1} zamanında çıkış yaptı",
e.User.Identity.Name, DateTime.Now.ToLongTimeString());

        btnLogin.IsEnabled = true;
        btnLogout.IsEnabled = false;
        brdProfile.Visibility = Visibility.Collapsed;
    }

    //AuthenticationService nesnesine ait Login metodunun çalıştırılması sonrasında
kullanıcı başarılı bir şekilde doğrulandıysa çalışır
    void authSrv_LoggedIn(object sender, AuthenticationEventArgs e)
    {
```

```
        lblLoginStatus.Content = String.Format("{0} {1} zamanında giriş yaptı",
e.User.Identity.Name, DateTime.Now.ToLongTimeString());

        btnLogin.IsEnabled = false;
        btnLogout.IsEnabled = true;
        brdProfile.Visibility = Visibility.Visible;
        lblRoles.Content = String.Empty;
        txtUserTitle.Text = String.Empty;

        // Login işlemi tamamlandıktan sonra WebContext.Current üzerinden giriş yapan
User bilgileri alınır
        Web.User currentUser = WebContext.Current.User;
        // Profil özelliklerinin değerleri ilgili kontrol özelliklerine atanır
        lblUserLastAccessTime.Content =
currentUser.LastAccessTime.ToLongTimeString();
        txtUserTitle.Text = currentUser.Title;

        // Kullanıcının dahil olduğu tüm roller Label kontrolü içerisinde ardışıl olarak
yazdırılır
        foreach (var role in currentUser.Roles)
        {
            lblRoles.Content += role + "|";
        }
    }

    private void btnSaveProfile_Click(object sender, RoutedEventArgs e)
    {
        // Save işleminde yine WebContext.Current üzerinden elde edilen User tipinin
özelliklerinden yararlanılır
        Web.User currentUser = WebContext.Current.User;

        // Bu kez profile özelliklerine, kontroller üzerindeki değerler atanır
        currentUser.Title = txtUserTitle.Text;
        currentUser.LastAccessTime = DateTime.Now;

        // Kaydetme operasyonu için AuthenticationService nesne örneğinin SaveUser
metodu çağırılır. Bu metodun çalıştırılması sırasında bir istisna olduğunda bunun ortama
fırlatılması için parametre olarak true değeri verilmiştir.
        SaveUserOperation operation=authSrv.SaveUser(true);
        // Kaydetme operasyonu tamamlandığında SaveUserOperation tipinin Completed
olay metodu devreye girer.
        operation.Completed +=
            (snd, arg) =>
            {
                lblProcess.Content = "Profil Save is OK";
```

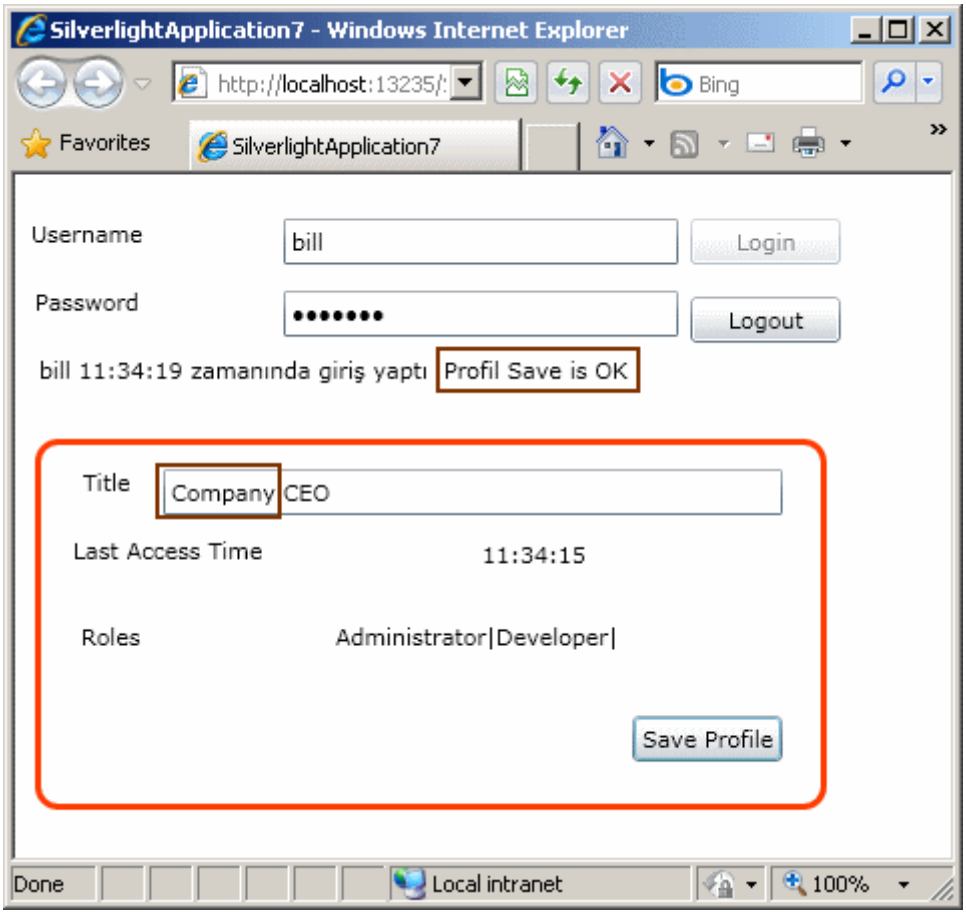
```
            };
        }
    }
}
```

(Eklediğimiz önemli kısımlar bold olarak işaretlenmiştir)

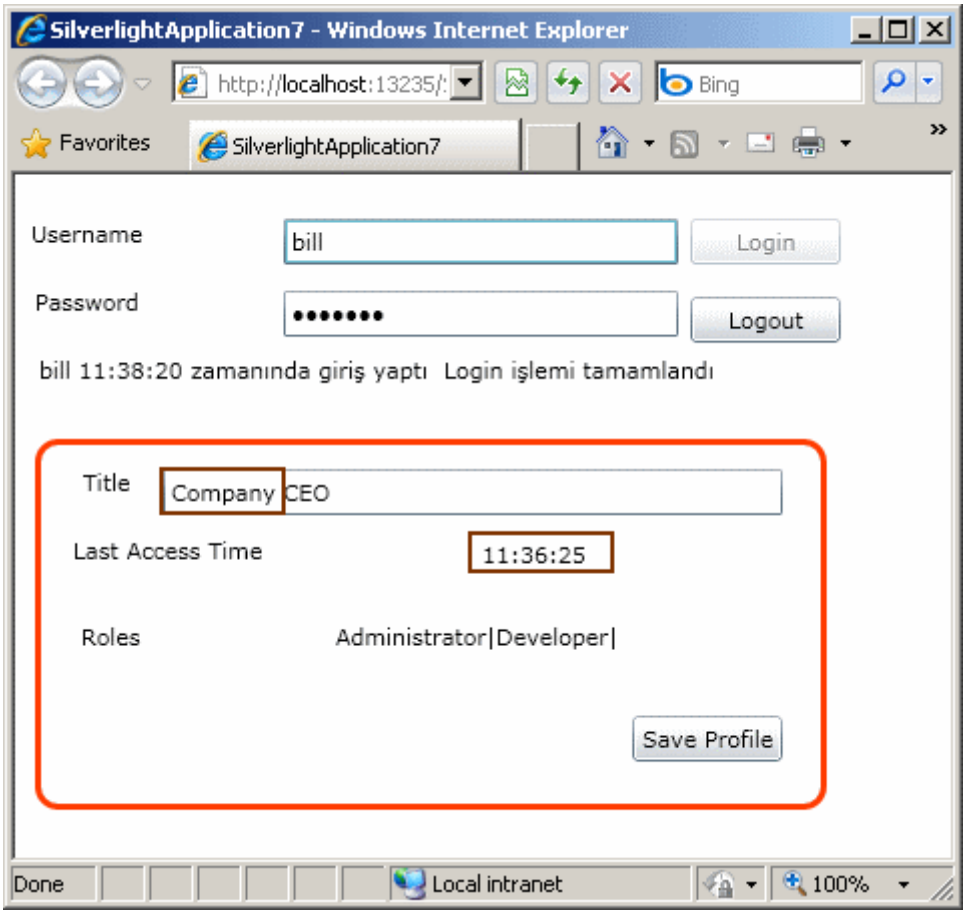
Hemen şu noktayıda vurgulayalım. İstenirse profil özelliklerinin kontrollere bağlanması sırasında **Binding** imkanlarından da yararlanılabilir. Bu durumda Login işlemini takiben otomatik olarak User bilgilerinin ilgili kontrollere bağlanması söz konusudur. Biz örneğimizde profil bilgilerinin gösterilmesi işlemlerini kod tarafında değerlendirmeye çalıştık. Ancak **Binding** işlemi ile aynı fonksiyonelliklerin uygulamaya nasıl kazandırılabileceğini incelemenizi öneririm. Dilerseniz uygulamamızın çalışma zamanını test ederek ilerleyelim. Ben **bill** isimli kullanıcı için daha önceden bir profil bilgisini test amacıyla kaydetmiştim. Bu durumda login işleminden sonra aşağıdaki görüntüye benzer sonuçlarla karşılaştım.



Şimdi profil bilgilerini değiştirip kaydettiğimizi düşünelim. Bu durumda **Save Profile** işleminin başarılı bir şekilde gerçekleştirildiğini görebiliriz.



Bu işlemin ardından tekrar **Logout** olup yeniden **Login** işlemini gerçekleştirirsek*(ki uygulamayı kapatıp yeniden başlatmakta söz konusu olabilir)* bill isimli kullanıcı için az önce kaydedilen profil bilgilerinin getirildiğini görebiliriz. Buda çalışmanın başarılı olduğunu bir ispatı olarak düşünülebilir.



Böylece geldik bir yazımızın daha sonuna. Bu yazımızda **WCF RIA Service - Authentication Domain Service** hizmetini kullanaraktan **Silverlight** uygulamalarında **Role** ve **Profile** alt yapılarının nasıl değerlendirilebileceğini incelemeye çalıştık. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

SilverlightApplication7RoleAndProfile.rar (780,46 kb) *[Dosya boyutunun küçük olması için ASPNETDB.mdf içeriği çıkartılmıştır]*

Webiner - Workflow Foundation 4.0 - Introduction [Beta 2] (2010-01-20T22:51:00)

webiner,wf 4.0,webcast,wf,

Merhaba Arkadaşlar,

.Net Framework 3.0 sürümü ile birlikte gelen köklü yeniliklere baktığımızda **Windows Presentation Foundation(WPF), Windows Communication Foundation(WCF)** ve **Workflow Foundation(WF)** alt yapı modellerinin ön plana çıktıklarını görmekteyiz. Servis bazlı çözümlere yeni bir yaklaşım getiren **WCF**, kod akışlarının görsel olarak geliştirilebilmesini sağlayan **WF** ve windows programlamaya tamamen farklı bir görsellik kazandıran **XAML** bazlı **WPF**.

.Net Framework 3.5 versiyonu yayınlandığında ise özellikle, **WCF** ve **WF** modellerinin bir birleriyle biraz daha haşır neşir olduklarını, içerisinde insan faktörü bulunan uzun süreli iş akışlarının geliştirilmesinde önemli ilerlemeler kaydedildiğini, iş akışı bazlı servislerin daha kolay yazılabildiğini gördük.

Gün geldi çok doğal olarak **.Net Framework 4.0'** ın ayak sesleri duyulmaya başlandı.*(Hatta 2008 yılının yaz aylarında yapılan Microsoft PDC' de duymaya başladık)* Bir kaç aya kadar **Release** sürümünün de çıkması beklenen **.Net Framework 4.0** içerisinde gerek **WCF** gerek **WF** tarafında önemli yenilikler olduğunu görmekteyiz.

İşte bu Webinerimizde **Workflow Foundation 4.0** modelini incelemeye ve öne çıkan yenilikleri değerlendirmeye çalıştık. önceki sürümde yer alan zorlukların anlatımı ile başlanan webinerimizde, kısaca **Workflow Foundation 4.0** ile gelen önemli yenilikler irdelenip basit bir örnek geliştirilmekte.

NedirTV?com araclılığıyla gerçekleştirilen ve **20.Ocak.2010 çarşmaba** günü saat **21:00** ile **22:00** arasında başlayacak olan webinere kayıt olmak içinhttp://msevents.microsoft.com/CUI/WebCastEventDetails.aspx?EventID=1032439662&EventCategory=4&culture=tr-TR&CountryCode=TR adresini kullanabilirsiniz.

Keyifli seyirler dilerim.

Workflow Services 4.0 - Transaction Flow [Beta 2] (2010-01-20T10:00:00)

wf 4.0,

 Merhaba Arkadaşlar,

Geçen gece ilginç bir rüya gördüm. Bir su birikintisine damlacıklar düşüyordu. önceleri yavaş yavaş ve uzun aralıklarla düşen damlalar söz konusuydu. Zaman ilerledikçe her bir damlanın suya değdiği noktada bir isim bıraktığını görmeye başladım. **int i, for, if** derken damlalar hızlanmaya başladı. Daha sık daha çok damla düşüyordu. Bazıları kocaman boyutlardaydı ve düştükleri su birikintisinde neredeyse fırtına koparıyorlardı. **.Net, C#, parallel, WPF, Ajax, ASP.Net, WCF, WF** derken damlaların artık nerelerden geldiğini takip edemez olmaya başladım. Ama damlalar iz bırakmaya devam ediyordu. **1.1, 2.0, 3.5, 4.0, Beta, RC, RTM**...derken terler içerisinde uyanmıştım

Bir kaç ay içerisinde eğer büyük bir aksilik olmassa **.Net Framework 4.0** ve **Visual Studio 2010** ürünlerinin son sürümleri yayınlanmış olacak. Şu anda gelişmeleri **Beta 2** sürümü üzerinden takip etmekteyiz. Ancak yakında **RC** ve sonrasında **RTM** sürümlerininde çıkacağını ve önemli iyileştirmeler olacağını biliyoruz. Yinede gelebilecek yenilikleri takip etmek adına araştırmalarıma devam etmekteyim. Bir süredir**Workflow Foundation 4.0** üzerine araştırma yapmıyordum. Geçtiğimiz günlerde **Transaction** yönetimi ile ilişkili olarak önemli bir açığın kapatıldığını öğrendim. Buna göre **Workflow Foundation 4.0** öncesinde, **Workflow Service'** lerde istemci tarafından başlatılan **Transaction**'ların, servis tarafına akması mümkün olmuyordu. Şimdi bir dakika...**Transaction, Flow**, **İstemciden Sunucuya**...Ihmmmm Biraz kafamız karışmış olabilir. Başlamadan önce bu konu hakkında biraz bilgi vermeye çalışalım dilerseniz.

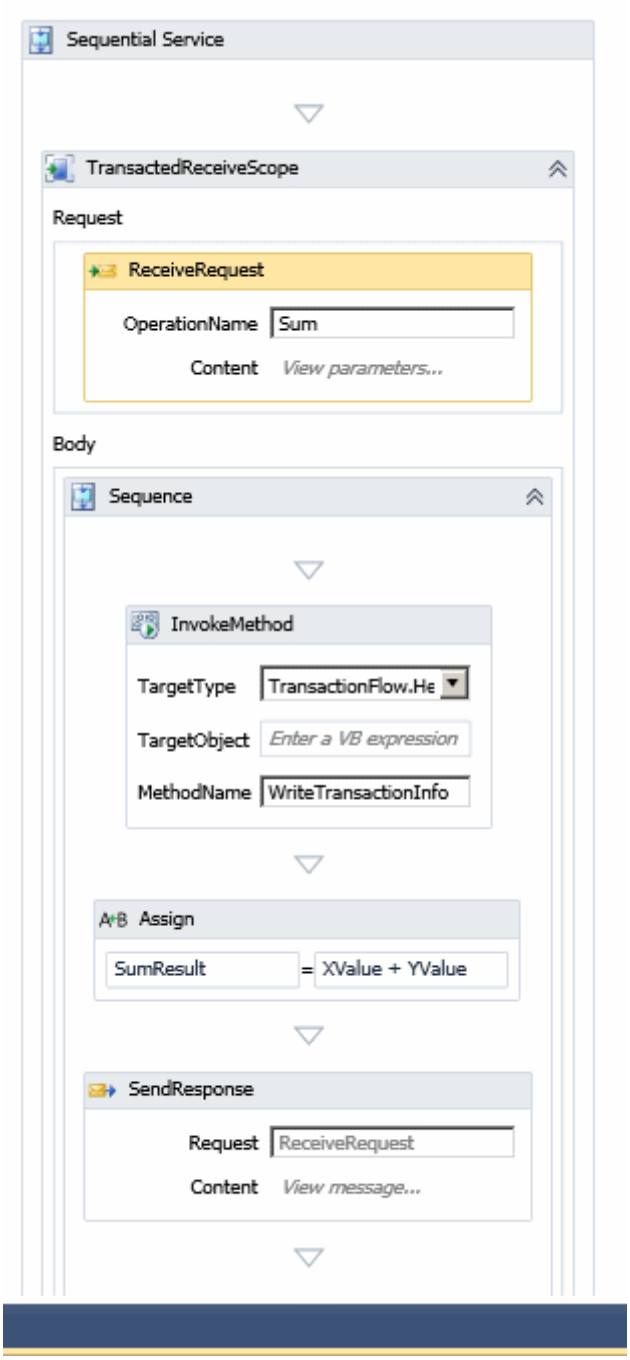
Bildiğiniz üzere bir **Transaction** içerisinde baştan sona başarılı bir şekilde tamamlanması beklenen işlemler bütünü yer alır. **Transaction** başlatıldıktan sonra içerisinde ceyran eden işlemlerin kalıcı olarak kabul görmesi, ancak tüm adımlardaki işlemlerin başarılı olmasına bağlıdır. Herhangibir adımda bir hata oluştuğunda **Transaction'** a dahil olan herkesin, **Transaction** başlamadan önceki **durumlarına(State)** dönebilmesi gerekir. üstelik bu **geri dönüşlerde(Rollback)** verilerin tutarlılığını korumak önemlidir. Hatta **Transaction**başarılı bir şekilde sonuçlandırıldığında, çıkan verilerin de anlamlı olması beklenir. Kısaca size **ACID(Atomicity,Consistensy,Isolation,Durability)** kavramını aktarmaya çalıştım. Tabi zaman ilerledikçe bir **Transaction'** ın sadece tek bir program alanı içerisinde değil,

birden fazla program alanı içerisinde ele alınmasının gerektiği durumlar oluşmuştur. Bu noktada **Dağıtık Transaction(Distributed Transaction)** kavramına geçilmektedir. Buna göre bir program alanı içerisinde başlatılan **Transaction**, başka bir program alanı içerisinde de ele alınabilir. Elbette bu tip bir senaryoda, program alanlarının farklı makineler üzerinde konuşlandırılmış olması da kuvvetle muhtemeldir. İşte bu gibi durumlarda **Transaction'** ın koordinasyonu için genellikle 3ncü parti araçların devreye girdiği görülmektedir**(Distributed Transaction Coordinator vb...)**

Gel gelelim zaman içerisinde söz konusu **Transaction'** ların servisler üzerinden akması ihtiyacı doğmuştur. Buna göre bir servis tarafından başlatılan bir **Transaction'** a, diğer bir serviste başlatılan operasyonun da dahil olması gerekebilir. Bu durumda ilgili servis operasyonlarının tamamının aynı **Transaction Scope** içerisinde ele alınması gerekmektedir. **Transaction Scope** denilince aklıma gelen ilk şey ise **Ado.Net 2.0** ile birlikte gelen **TransactionScope** tipidir. Bu tip sayesinde, blok içerisine dahil olan farklı **bağlantılar(Connection)** için aynı **Transaction Scope**'un oluşturulması ve yönetilmesi son derece kolaylaşmıştır.

Şimdi gelelim bu güne. Artık elimizin altında bir **Workflow'** un servis bazlı olarak sunulabilmesi imkanı bulunmakta. Uzun süredir. Buna göre istemcilerin, söz konusu **Workflow'** ları servis bazlı olaraktan talep edebilmesi mümkün. Hal böyle olunca istemci tarafından başlatılacak bir **Transaction'** ın, çağrıda bulunulan **Workflow Service** tarafından' da ele alınabiliyor olması istenen bir özelliktir. Dolayısıyla istemcide açılan **Transaction'** ın **Workflow Service** tarafına **akabiliyor(Flow)** olması gerekmektedir.

Workflow Foundation 4.0 Beta 2 sürümünde söz konusu işlevselliği sağlamak için **Messaging** kontrollerinde yer alan **TransactedReceiveScope** isimli aktivite bileşeninden yararlanılmaktadır. Bu bileşen içerisinde istemcilerin çağrıda bulunacağı operasyon bildirimi yer alır. Bunun içinde **Receive** aktivite bileşeninden yararlanılmaktadır. Dilerseniz olayı kavramak için basit bir örnek geliştirelim. örneğimizde aşağıdaki **XAML** içeriğine sahip bir **Workflow Service** oluşturduğumuzu göz önüne alalım.



Xaml içeriğimiz;*(Sadece Sequence içeriği belirtilmiştir)*

```
<p:Sequence DisplayName="Sequential Service"
sad:XamlDebuggerXmlReader.FileName="G:\Projects\Workflow
Foundation\Transactions\TransactionFlow\MathFlowService.xamlx"
sap:VirtualizedContainerService.HintSize="325,797">
   <p:Sequence.Variables>
     <p:Variable x:TypeArguments="CorrelationHandle" Name="handle" />
     <p:Variable x:TypeArguments="x:Int32" Name="XValue" />
     <p:Variable x:TypeArguments="x:Int32" Name="YValue" />
     <p:Variable x:TypeArguments="x:Int32" Name="SumResult" />
```

```
    </p:Sequence.Variables>
    <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">True</x:Boolean>
      </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <TransactedReceiveScope Request="{x:Reference __ReferenceID0}"
sap:VirtualizedContainerService.HintSize="303,673">
      <p:Sequence sap:VirtualizedContainerService.HintSize="277,474">
        <sap:WorkflowViewStateService.ViewState>
          <scg3:Dictionary x:TypeArguments="x:String, x:Object">
            <x:Boolean x:Key="IsExpanded">True</x:Boolean>
          </scg3:Dictionary>
        </sap:WorkflowViewStateService.ViewState>
        <p:InvokeMethod sap:VirtualizedContainerService.HintSize="255,127"
MethodName="WriteTransactionInfo" TargetType="t:Helper" />
        <p:Assign sap:VirtualizedContainerService.HintSize="255,57">
          <p:Assign.To>
            <p:OutArgument x:TypeArguments="x:Int32">[SumResult]</p:OutArgument>
          </p:Assign.To>
          <p:Assign.Value>
            <p:InArgument x:TypeArguments="x:Int32">[XValue + YValue]</p:InArgument>
          </p:Assign.Value>
        </p:Assign>
        <SendReply DisplayName="SendResponse"
sap:VirtualizedContainerService.HintSize="255,86">
          <SendReply.Request>
            <Receive x:Name="__ReferenceID0" CanCreateInstance="True"
DisplayName="ReceiveRequest" sap:VirtualizedContainerService.HintSize="277,86"
OperationName="Sum" ServiceContractName="p1:IMathFlowService">
              <Receive.CorrelatesOn>
                <MessageQuerySet />
              </Receive.CorrelatesOn>
              <Receive.CorrelationInitializers>
                <RequestReplyCorrelationInitializer CorrelationHandle="[handle]" />
              </Receive.CorrelationInitializers>
              <ReceiveParametersContent>
                <p:OutArgument x:TypeArguments="x:Int32"
x:Key="x">[XValue]</p:OutArgument>
                <p:OutArgument x:TypeArguments="x:Int32"
x:Key="y">[YValue]</p:OutArgument>
              </ReceiveParametersContent>
            </Receive>
          </SendReply.Request>
          <SendMessageContent DeclaredMessageType="x:Int32">
```

```
          <p:InArgument x:TypeArguments="x:Int32">[SumResult]</p:InArgument>
        </SendMessageContent>
      </SendReply>
    </p:Sequence>
  </TransactedReceiveScope>
 </p:Sequence>
```

Akışımız içerisinde birde yardımcı sınıf bulunmaktadır. **Helper** isimli sınıf içerisinde **InvokeMethod** aktivitesi tarafından çalıştırılan ve güncel **Transaction** ile ilişkili bilgileri dosyaya aktaran basit bir fonksiyonellik yer almaktadır.

```
using System;
using System.Text;
using System.Transactions;
using System.IO;

namespace TransactionFlow
{
   public class Helper
   {
      public static void WriteTransactionInfo()
      {
         StringBuilder builder = new StringBuilder();

         TransactionInformation currentTrx = Transaction.Current.TransactionInformation;
         builder.AppendLine(String.Format("Oluşturulma zamanı
{0}", currentTrx.CreationTime.ToString()));
         builder.AppendLine(String.Format("Local Identifier değeri
{0}", currentTrx.LocalIdentifier.ToString()));
         builder.AppendLine(String.Format("Distributed Identifier değeri
{0}", currentTrx.DistributedIdentifier.ToString()));

         File.WriteAllText("c:\\TransctionInformations.txt", builder.ToString());
      }
   }
}
```

Workflow çok basit olarak istemciden gelen iki sayının toplamını geriye döndürmek üzerine tasarlanmıştır. Ancak dikkat edilmesi gereken nokta kullandığı **Transaction** bilgileridir. Tabi şu noktada unutulmamalıdır. Hem **Workflow Service** hemde istemci uygulama **System.Transactions.dll assembly'** ını referans etmelidir. Helper sınıfı içerisinde yer alan WriteTransactionInfo metodu text tabanlı bir dosya içerisine eğer varsa güncel transaction bilgilerini yazdırmaktadır. Bunlardan ilki transaction oluşturulma zamanıdır**(CreationTime)**. Sonrasında **yerel transaction**

değeri(LocalIdentifier) ve **dağıtık transaction değerleri(DistributedIdentifier)** yazdırılır. Bu değerler **GUID** tipindendir.

Workflow Service tarafında istemciden gelecek **Transaction** akışına izin vermek için sadece **TransectedReceiveScope** bileşeninin kullanılması yeterli değildir. Konfigurasyon içerisinde de transaction akışına izin verileceğinin bildirilmesi gerekir. Bu amaçla **Workflow Service** uygulamasındaki **web.config** içeriği aşağıdaki gibi düzenlenebilir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>
  <system.serviceModel>
    <bindings>
      <wsHttpBinding>
        <binding transactionFlow="true"/>
      </wsHttpBinding>
    </bindings>
    <services>
      <service name="MathFlowService">
        <endpoint address="" binding="wsHttpBinding"
contract="IMathFlowService"/>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceMetadata httpGetEnabled="true"/>
          <serviceDebug includeExceptionDetailInFaults="false"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
  <system.webServer>
    <modules runAllManagedModulesForAllRequests="true"/>
  </system.webServer>
</configuration>
```

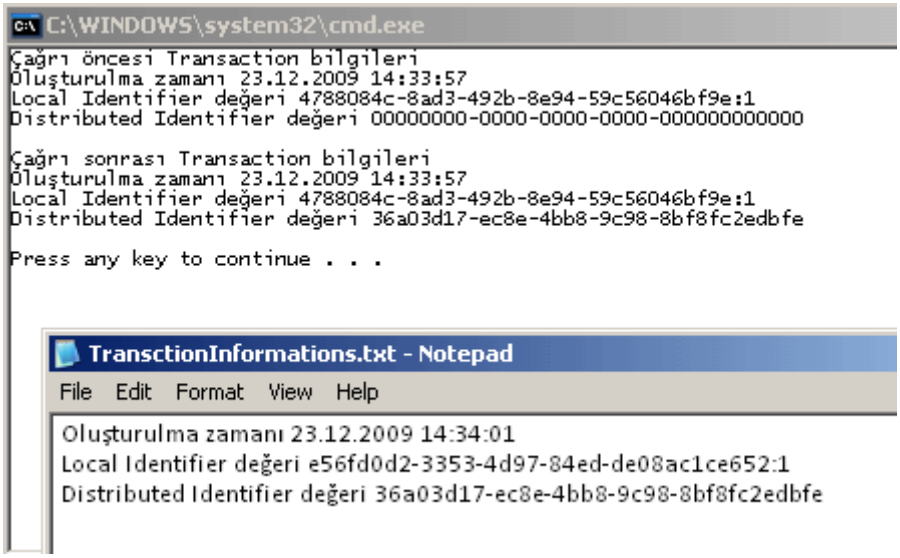
Dikkat edileceği üzere **wsHttpBinding bağlayıcı tipi(Binding Type)** için **transactionFlow** niteliğine **true** değeri atanmıştır. Bu zaten **Windows Communication Foundation(WCF)** tarafından bildiğimiz bir ilkedir. Şimdi gelelim istemci uygulama tarafına. **Console** projesi şeklinde tasarlanan istemci uygulamaya öncelikli olarak **Workflow Service** için gerekli proxy içeriği **Add Service**

Reference seçeneği ile eklenmelidir. Tabi buna uygun olarak istemci tarafında üretilen **app.config** içerisinde de transaction akışı için gerekli bildirimler otomatik olarak üretilecektir. Bunu takiben**Main** metodunda aşağıdaki örnek kodların geliştirildiğini düşünelim.

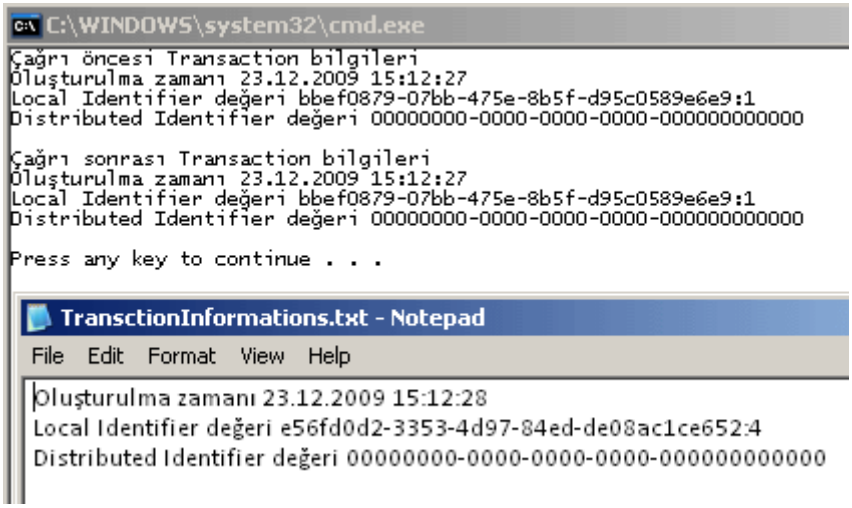
```
using System;
using System.Text;
using System.Transactions;
using ClientApp.MathFlowSpace;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         using (TransactionScope scope = new TransactionScope())
         {
            MathFlowServiceClient proxy = new MathFlowServiceClient();

            Console.WriteLine("çağrı öncesi Transaction bilgileri");
            WriteTransactionInfo();

            int? result = proxy.Sum(new Sum { x = 1, y = 4 });

            Console.WriteLine("çağrı sonrası Transaction bilgileri");
            WriteTransactionInfo();
         }
      }

      public static void WriteTransactionInfo()
      {
         StringBuilder builder = new StringBuilder();

         TransactionInformation currentTrx =
Transaction.Current.TransactionInformation;
         builder.AppendLine(String.Format("Oluşturulma zamanı
{0}", currentTrx.CreationTime.ToString()));
         builder.AppendLine(String.Format("Local Identifier değeri
{0}", currentTrx.LocalIdentifier.ToString()));
         builder.AppendLine(String.Format("Distributed Identifier değeri
{0}", currentTrx.DistributedIdentifier.ToString()));

         Console.WriteLine(builder.ToString());
      }
```

```
    }
}
```

İstemci uygulama tarafında en önemli nokta **TransactionScope** kullanımıdır. Ayrıca, **Workflow Service** operasyonunun çağırılmasından hemen önce ve sonra ortamdaki güncel**Transaction** bilgilerinin yazdırılması sağlanmıştır. Buna göre örnek bir çalışma zamanı çıktısı aşağıdaki gibi olacaktır.



Dikkat edileceği üzere istemci tarafında servis çağrısının yapılmasından sonra oluşan ve servis tarafında dosyaya
yazılan **Transaction** bilgilerindeki **DistributedIdentifier** değerleri aynıdır. Bir başka deyişle istemci ve **Workflow Service** tarafı aynı **Transaction** alanı içerisinde çalıştırılmıştır. Hemen tersi durumu ispat etmeye çalışalım. Bunun için her iki taraftaki **config** dosyalarında yer alan **transactionFlow** niteliklerine **false** değer verdiğimizi düşünelim. örneği tekrardan çalıştıralım. İşte sonuç;



Görüldüğü gibi operasyon çağrısı sonucu istemcide üretilen ve servis tarafında dosyaya yazılan **DistributedIdentifier** değerleri 0' dır. 0 olması zaten bir dağıtık transaction

oluşturulmadığı/oluşturulamadığı anlamına gelmektedir. İşte bu kadar. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Transactions.rar (54,45 kb)

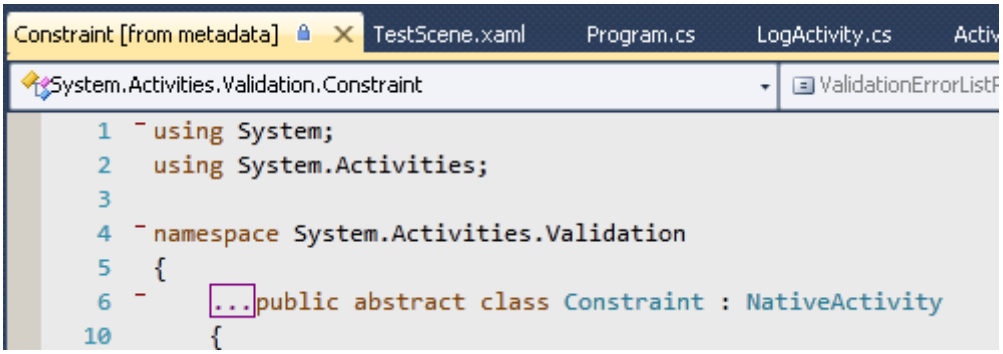
Workflow Foundation 4.0 - Declarative Validation [Beta 2] (2010-01-19T15:00:00)

wf 4.0,

Merhaba Arkadaşlar,

Sakin bir Cuma gününde bilgisayarın başında kahvemi yudumlarken ve M&M drajelerinden avuç avuç yerken araştırmalarıma devam ediyordum. Bir süredir **Workflow Foundation 4.0** ile birlikte gelen yenilikleri incelediğimden takip ettiğim bloglar ve MSDN üzerinde bu konu ile ilişkili yazıları okumaktaydım. özelliklede son iki yazımda üzerinde durmaya çalıştığım özel aktivite bileşenlerinin doğrulanması konusunu irdelemekteydim. Bu yazımızda **doğrulama(Validation)** ile ilişkili araştırmalarımı sizlerle paylaşmaya devam ediyor olacağım.

Doğrulama işlemlerinin çeşitlerine baktığımızda **Declarative Constraint** isimli bir yaklaşımın daha olduğu görülmektedir. Bu yaklaşıma göre bir aktivite ile ilişkili doğrulama mantığının **kısıt olaraktan(Constraint)** ayrı bir tip ve metod içerisinde konuşlandırılması mümkündür. *(Hatta kod dışında XAML bazlı olaraktan kısıtların konulmasıda söz konusudur)* **Workflow Foundation** alt yapısında bu tip dekleratif doğrulamalar için **Constraint** sınıfından yararlanılmaktadır. **Constraint** tipi aslında **NativeActivity** türevidir. Bir başka deyişle bir aktivitedir.



Yukarıdaki şekildende görülebileceği gibi **Constraint abstract** bir sınıftır*(dolayısıyla kendisinden türeyen tiplerin mutlaka uyması gereken kuralları bildiren, örneklenemeyen ama kendisinden türeyen tip örneklerini taşıyabilen bir sınıftır)* ve **NaticeActivity** tipinden türemektedir. Bu türetme nedeniyle aslında **Workflow** çalışma zamanının çeşitli materyallerine erişebildiğini(Scheduling, Bookmarks vb...) söyleyebiliriz. Peki pratikte kendi geliştireceğimiz aktivite bileşenleri için gerekli kısıtları nasıl koyabiliriz? **MSDN** üzerinde bu konu ile ilişkili olarak geliştirilen basit örnekte bir aktivite bileşeninin **DisplayName** özelliğinin 2 karakterden fazla olması gerekliliğinin örneklendiği görülmektedir. Bizde buna benzer bir kısıtlama geliştiriyor olacağız. Ancak örneğin farklı olması açısından, hayali yazılım şirketinin kod standartlarına göre DisplayName özelliğinin **Chinook** ön eki ile başlaması için bir kısıt getireceğiz. İşte Activity Library içerisinde tuttuğumuz örnek sınıf kodlarımız.

```
using System.Activities;
using System.Activities.Validation;

namespace CustomActivities
{
   public static class ActivityConstraints
   {
      // Constraint oluşturup geriye döndürecek basit bir static metod
      public static Constraint
ValidateActivityDisplayNameForCompanyCodeStandards()
      {
         DelegateInArgument<Activity> element = new
DelegateInArgument<Activity>();

         // Herhangibir Activity(T generic tipi olarak Activity kullanıldığından) bileşeninin
doğrulanmasında kullanılacak olan Constraint tipi örneklenir.
         // Constraint tipi aslında NativeActivity türevli bir Activity bileşenidir.
         Constraint<Activity> constraint = new Constraint<Activity>
         {
            // Body kısmı doğrulama mantığını içermektedir ve ActivityAction
tipindendir
            Body = new ActivityAction<Activity, ValidationContext>
            {
```

```
                Argument1 = element,
                Handler = new AssertValidation
                {
                    // Warning bilgisi gösterilmeyecektir. Yani Error mesajı verilecektir.
IsWarning özelliğinin varsayılan değeri false' dur.
                    IsWarning=false,
                    // e, ActivityContext tipinden bir referanstır. Dolayısıyla Constraint' in
uygulanacağı aktivite bileşeninin güncel içeriğine erişilebilmesi mümkündür.
                    // örnek doğrulamaya göre Actitiy örneğinin DisplayName özelliğinin
Chinnok kelimesi ile başlaması beklenmektedir.
                    Assertion=new InArgument<bool>(
                      e=>
                        element.Get(e).DisplayName.StartsWith("Chinook")
                      ),
                      // Eğer Chinook ismi ile başlanılmıyorsa bir hata mesajı verilir.
                    Message=new InArgument<string>("Şirketin kod standartları gereği,
özel Activity adlarının Chinook ile başlaması gerekmektedir."),
                    DisplayName="DisplayNameValidationActivity"
                }
            }
        };
        return constraint;
    }
  }
}
```

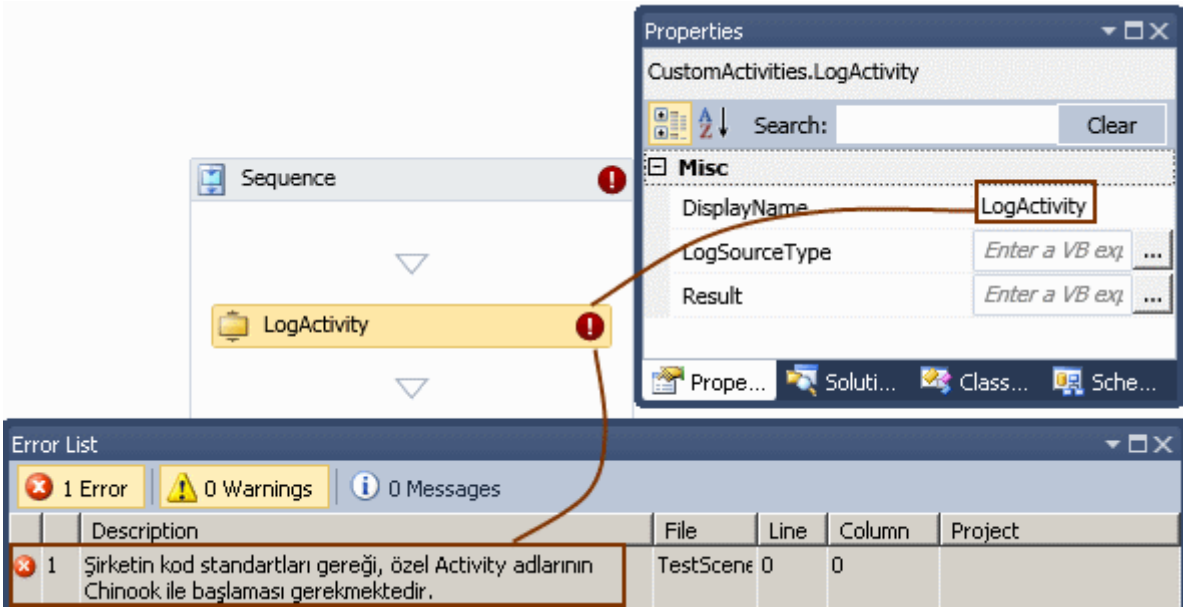
ActivityConstraints isimli **static** sınıf içerisinde yer alan **ValidateActivityDisplayNameForCompanyCodeStandards** isimli metod geriye **Constraint** tipinden bir referans döndürmektedir. **Constraint** tipinin üretimi sırasında dikkat edileceği üzere **Handler** özelliğine **AssertValidation** tipinden bir referans atanmaktadır. İşte bu sınıfın örneklenmesi sırasında kullanılan **Assertion**özelliği ilede bir **Expression** tanımlaması yapılmakta ve bu kısıtın uygulandığı **Activity** bileşeninin **DisplayName** özelliğinin **Chinook** ile başlayıp başlamadığı kontrol edilmektedir. Bu ifadeden dönecek değer göre **Message** özelliğine atanan bilginin derleme zamanında gösterilmesi sağlanmaktadir. **IsWarning** özelliği varsayılan olarak **false** değere sahiptir ve buna göre mesajın bir **Error** olarak gösterilmesi sağlanmaktadır. Ancak bu özelliğe **true** değerini atayaraktan **Warning** olarak gösterilmesi de sağlanabilir. Peki bu kısıt bir aktivite bileşenine nasıl uygulanabilir? Aslında bunun için geliştiriken aktiviteye bir bildirimde bulunulması yeterlidir. Aşağıdaki kod parçasında yer alan aktivite bileşeninde bu durum ele alınmaktadır.

```
using System.Activities;
using System.Activities.Validation;
```

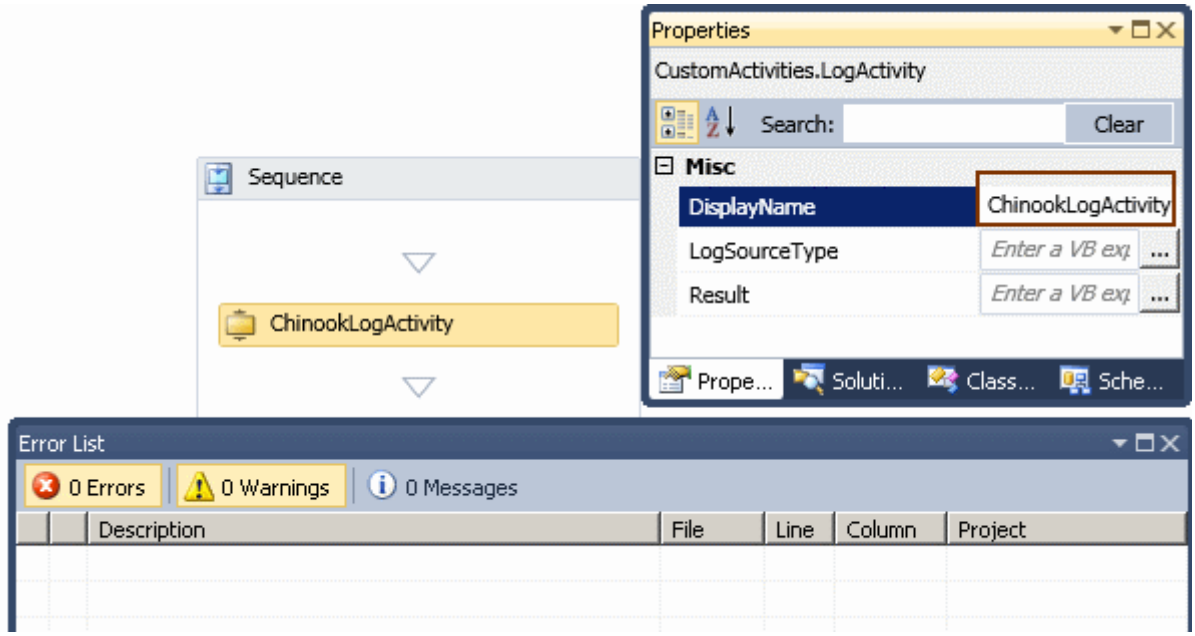
```
namespace CustomActivities
{
   public enum LogSource
   {
      File,
      Database,
      WebService,
      System
   }

   // Loglama yapan örnek bir aktivitedir.
   public sealed class LogActivity
      : CodeActivity<bool>
   {
      public InArgument<LogSource> LogSourceType { get; set; }

      public LogActivity()
      {
         // Constraint' lerin bir aktivite ile ilişkilendirilebilmesi için base referans üzerinden
ilgili koleksiyona eklenmesi gerekmektedir.
         // Constraints özelliği bir koleksiyonu referans ettiği için bir aktiviteye birden fazla
Constraint yüklenmesi mümkündür.
         base.Constraints.Add(ActivityConstraints.ValidateActivityDisplayNameForCo
mpanyCodeStandards());
      }

      protected override bool Execute(CodeActivityContext context)
      {
         //TODO@Burak: Gerçektende loglama işlemi yapılması için gerekli kodlar
yazılmalı
         switch (LogSourceType.Get(context))
         {
            case LogSource.File:
               break;
            case LogSource.Database:
               break;
            case LogSource.WebService:
               break;
            case LogSource.System:
               break;
            default:
               break;
         }
```

```
            return true;
        }
    }
}
```

Burada odaklanılması gereken tek bir yer vardır...**Yapıcı metodun(Constructor)** içerisinde yer alan kod satırı. Dikkat edileceği üzere **Constraints** özelliği üzerinden**ValidateActivityDisplayNameForCompanyCodeStandards** metodunun dönüş referansının ilgili koleksiyona eklenmesi sağlanmaktadır. Buna göre, **LogActivity** bileşeninin **Visual Studio** ortamında kullanıldığı durumlarda **DisplayName** özelliğinin ilgili kısıta göre kontrol edileceği bildirilmiş olmaktadır. Dilerseniz birde bileşeni test edelim. İşte bileşeni örnek bir Workflow üzerine ilk kez sürükleyip bıraktığımızdaki durum;



Görüldüğü üzere **DisplayName** özelliği için bir hata mesajı alınmıştır. Buna göre Chinook ön ekin kullandığımızda sorunun ortadan kalktığı görülebilir.



Constraint kullanımı dikkat edileceği üzere herhangibir aktivite bileşeni için kısıt koyabilmeyi olanaklı hale getirmektedir. **Constraint** kullanımında farklı durumlarda söz konusudur. örneğin bir aktivitenin içerisinde yer alan tüm alt aktiviteler için geçerli olacak kısıtların konulmasıda mümkün olabilir. Bu son derece doğaldır nitekim **Constraint** sınıfı örneklenirken, ilişkilendirildiği aktivite içeriğine erişebilmektedir.
Burada **DelegateInArgument<Activity>** temsilci tipinin büyük rolü vardır. öyleki Expression tanımlamasının yapıldığı ve doğrulama mantığının gerçekleştirildiği yerde, güncel aktivite referansına ulaşmak için **element.Get(e)** söz dizimi kullanılmaktadır.
Tabiki bu yazımızda **Declarative Constraint** kullanımını çok basit seviyede ele almaya çalıştık. En güncel ve detaylı bilgiyi elbetteki MSDN üzerinde bulabilirsiniz. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

DeclerativeConstraints.rar (50,29 kb)

Workflow Foundation 4.0 - Kodlama Zamanında Doğrulama(Validation) (2010-01-15T10:08:00)

wf 4.0,wf,workflow foundation,

Merhaba Arkadaşlar,

Bazen nerede duracağımızı bilmemiz gerekir ve bazende, mümkün olduğunca erken durup bazı şeyleri değiştirerek ilerlememiz...Bu teori yazılım geliştirmeninde pek çok noktasında karşımıza çıkmaktadır. Durmamız gereken noktalardan birisi, uygulamaların ürettiği ve önceden fark edebileceğimiz hatalardır*(Genellikle Exception' ları düşünebiliriz).* Ancak bazı olası hataların uygulamaların çalışması sırasında değil, çalıştırılmaya başlamadan önce bilinmesinde hem zaman hemde maliyet kazancı açısından yarar vardır. Şimdi elimizdeki materyalleri bir düşünelim. ürün geliştirmek için kullandığımız **Visual Studio** gibi gelişmiş bir araç, **.Net Framework**platformu vb...O halde bazı hataların çalışma zamanı yerine daha geliştirme aşamasındayken IDE üzerinde fark edilmesinin önemli olduğunu söyleyebiliriz. Peki geliştirme safhasında, örneğin bir Workflow aktivitesini kullanırken...Hımmmm...Sanırım nereye varmak istediğimi anladınız. Bu yazımızda, özel aktivite bileşenlerinin kodlama zamanındayken olası hataları nasıl bildirebileceğini ve böylece **doğrulamanın(Validation)** nasıl sağlanabileceğini incelemeye çalışacağız.

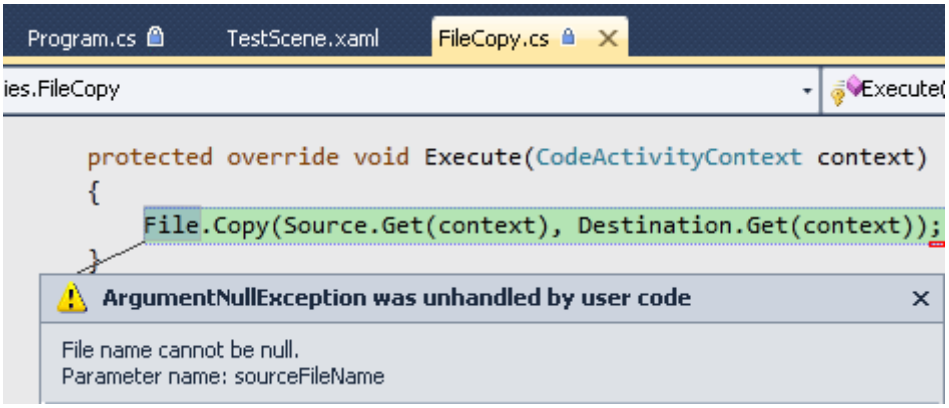
Bundan önceki yazılarımızda özel aktivite bileşenlerimizi nasıl geliştirebileceğimizi incelemeye çalışmıştık. Bu amaçla yazdığımız örneklermizde **CodeActivity** ve**AsyncCodeActivity** türevli bileşenler geliştirmiştik. özel aktivite bileşenleri geliştirilmesi sırasında dikkat edilmesi gereken önemli konulardan biriside kodlama zamanında gerçekleştirilmesini istediğimiz **doğrulama(Validation)** işlemleridir. Doğrulama için **Workflow Foundation 4.0** tarafında kullanılabilen birden fazla teknik bulunmaktadır. **Nitelik(Attribute)** bazlı kullanım dışında **Imperative** ve **Declerative** olaraktan da doğrulama işlemlerini gerçekleştirebiliriz. Aslında konuyu anlamanın en güzel yolu öncelikli olarak sorunu ortaya koymaktan geçmektedir. Bu sebepten aşağıdaki gibi bir **CodeActivity** bileşeni tasarladığımızı düşünelim.

```
using System.Activities;
using System.IO;
using System;
using System.Activities.Expressions;
```

```
namespace CustomActivities
{
   public sealed class FileCopy
      : CodeActivity
   {
      public InArgument<string> Source { get; set; }
      public InArgument<string> Destination { get; set; }

      protected override void Execute(CodeActivityContext context)
      {
         File.Copy(Source.Get(context), Destination.Get(context));
      }
   }
}
```

FileCopy isimli aktivite bileşenimizin görevi **Source** özelliğinde belirtilen dosyayı, **Destination** özelliğinde belirtilen yere kopyalamaktır. Bu işlem için **CodeActivity** tarafından gelip**ezilen(override) Execute** metodu içerisinde, **File** sınıfının **static Copy** metodundan yararlanılmaktadır. Ne varki bu aktivite bileşeninin özellikle çalışma zamanında üreteceği bazı sorunlar vardır. Aslında bunları şimdiden tahmin etmemiz son derece kolaydır.

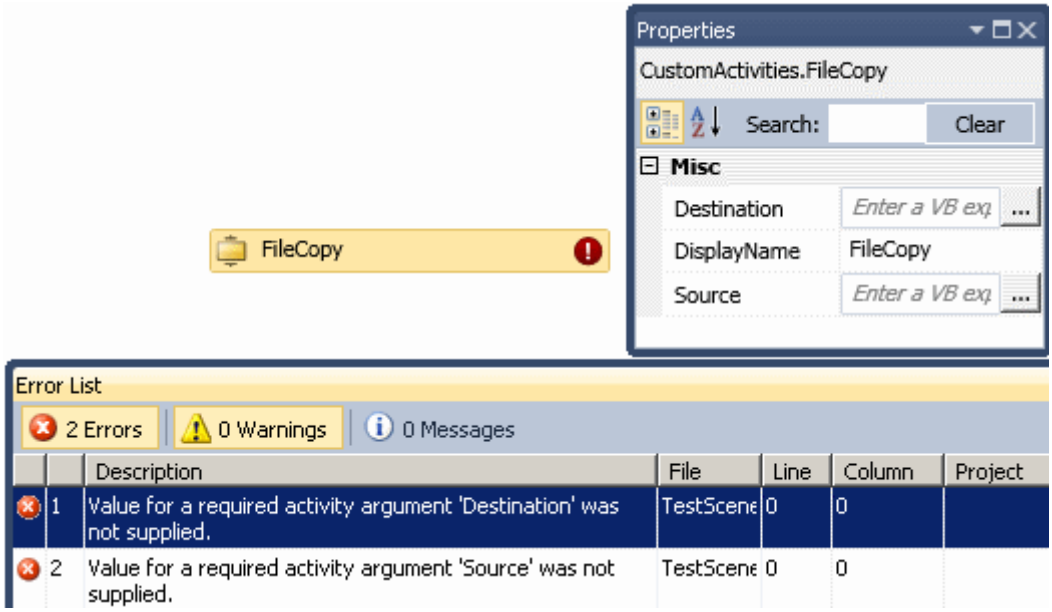
Herşeyden önce, **InArgument<string>** tipinden olan **Source** ve **Destination** özelliklerinin boş geçilmemesi yani veri ile doldurulması şarttır. Nitekim **null** verilerin **Copy** metodu içerisinde kullanılması söz konusu olamaz. Diğer yandan **Source** özelliğinde belirtilen dosyanın, sistemde gerçekten var olması gerekmektedir. Dolayısıyla **Source** özelliğinde bir değer olsa bile bunun geçerli bir dosya olup olmadığına bakılmalıdır. üçüncü olaraktan **Destination** özelliğinde belirtilen dosya adının geçerli olması ve belkide **Source** ile belirtilen dosya uzantısında olması gerekmektedir. Hatta hedef dosya zaten var ise üzerine yazılması durumu söz konusudur. Bu vakaların herhangibirinin çalışma zamanında gerçeklenmesi sonrası **istisnalar(Exceptions)** ile karşılaşılması kaçınılmazdır. Söz gelimi aktivitemizi bu haliyle örnek bir **Workflow** içerisinde icra ettirdiğimizde çalışma zamanında aşağıdaki istisna mesajını alırız.



Source özelliğinde null değer olduğundan **Copy** operasyonunun icrası sırasında **ArgumentNullException** alınmıştır. Oysaki bu hatanın çalışma zamanında değil kodlama zamanında, yani **Visual Studio** ortamında daha tasarımı gerçekleştirirken farketmemiz çok önemlidir. Bu bize zaman ve maliyet kazancı olarak geri dönecektir. İşte doğrulamanın sağlanması gereken yerlerden birisi burasıdır. Peki bunu nasıl gerçekleştirebiliriz? öncelikli olarak **Source** ve **Destination** isimli **InArgument<string>** tipinden olan özelliklerin boş bırakılmamasını sağlamalıyız. Bunun için ilgili özellikleri **RequiredArgument** niteliği ile aşağıdaki kod parçasında görüldüğü gibi işaretlememiz yeterli olacaktır.

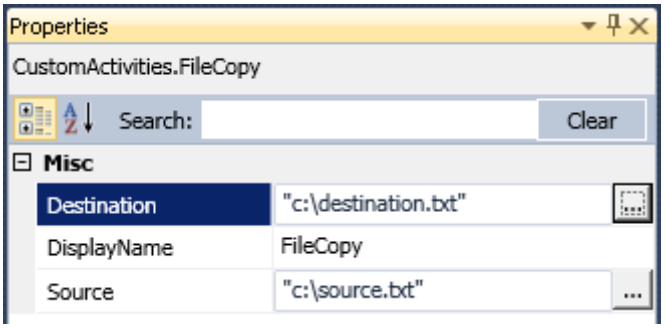
```
[RequiredArgument]
public InArgument<string> Source { get; set; }
[RequiredArgument]
public InArgument<string> Destination { get; set; }
```

Bu durumda tasarım zamanında aşağıdaki görüntü ile karşılaşırız.

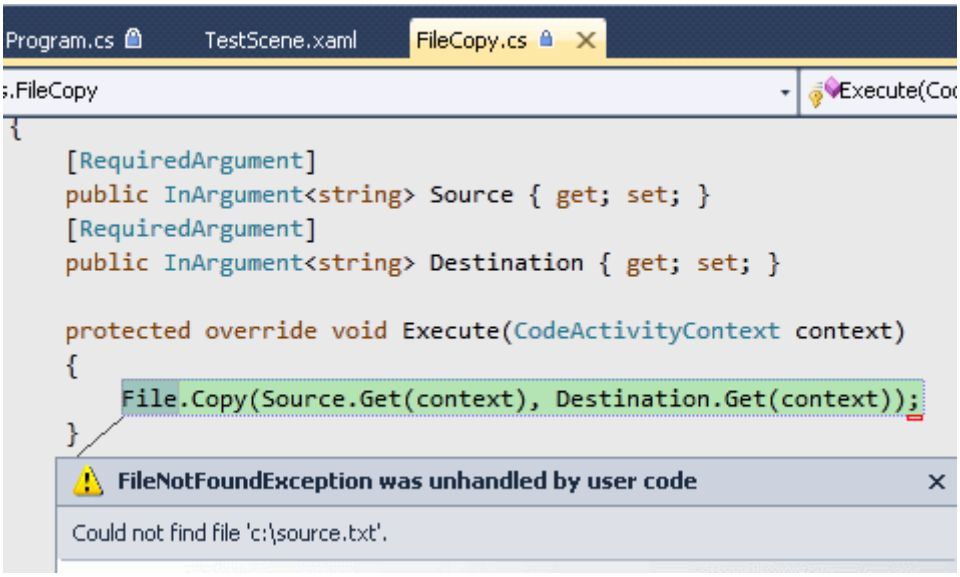


Dikkat edileceği üzere **Source** ve **Destination** değelerinin doldurulması zorunlu hale getirilmiştir. üstelik bu durum derleme işleminden sonra açık bir şekilde adeta geliştiricinin gözüne sokulmaktadır. 😃 **RequiredArgument** niteliği ile ilişkili olarak dikkat edilmesi gereken noktalardan biriside **Argument** tiplerine uygulandığında işe yarıyor olmasıdır. Yani **Soruce** ve **Destination**özelliklerinin **string** tipinden olmaları gibi bir durumda bu niteliğin bir etkisi olmayacaktır.

İlk vakayı çözümledik. Artık dosya adlarını girdiğimizi düşünebiliriz. Sıradaki sorun **Source** özelliğine girilen dosyanın sistemde olmaması halinde ortaya çıkacaktır. Buna göre yine kopyalama işlemi sırasında bir çalışma zamanı hatası alınacaktır. Durumu irdeleyebilmek için **Source** ve **Destination** özelliklerini aşağıdaki şekilde görüldüğü gibi ayarladığımızı düşünelim.



Bu durumda çalışma zamanında aşağıdaki hata mesajını alırız.

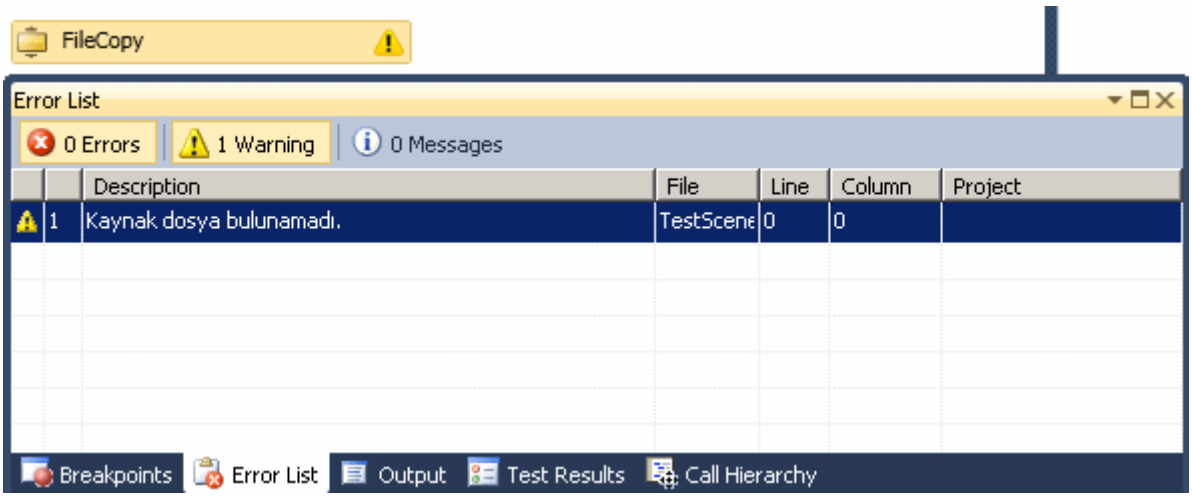


Tabi burada c:\ klasörü içerisinde **Source.txt** isimli bir dosyanın gerçektende var olmadığını düşünüyoruz. Buradaki istisnaya göre kaynak dosyanın önceden kontrol edilmesi ve derleme işleminden sonra geliştiriciye bir uyarı veya hata mesajı ile durumun bildirilmesi istenebilir. Ancak burada basit bir nitelik yardımıyla aşabileceğimizin ötesinde bir durum vardır. Nitekim doğrulama için özel bir **iş mantığı(Bussines Logic)** bulunmaktadır. İşte bu tip doğrulamaları gerçekleştirebilmek için **CodeActivity** ve **NativeActivity** türevlerinde **CacheMetadata**isimli metodun ezilmesi ve **Visual Studio** ortamına doğrulama ile ilişkili bilgilendirmenin yapılması gerekmektedir. Sözün özü **FileCopy** aktivite bileşenimizi aşağıdaki hale getirmemiz yeterli olacaktır 😉

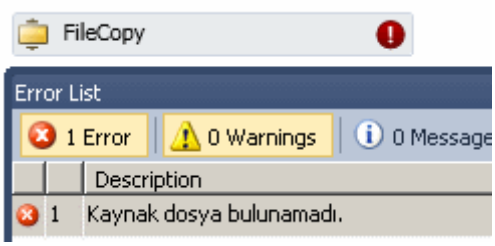
```
using System.Activities;
using System.IO;
using System;
using System.Activities.Expressions;

namespace CustomActivities
{
   public sealed class FileCopy
      : CodeActivity
   {
```

```
        [RequiredArgument]
        public InArgument<string> Source { get; set; }
        [RequiredArgument]
        public InArgument<string> Destination { get; set; }

        protected override void Execute(CodeActivityContext context)
        {
            File.Copy(Source.Get(context), Destination.Get(context));
        }

        protected override void CacheMetadata(CodeActivityMetadata metadata)
        {
            base.CacheMetadata(metadata);
            string source = ((Literal<string>)Source.Expression).Value;

            if (!File.Exists(source))
                metadata.AddValidationError(new
System.Activities.Validation.ValidationError("Kaynak dosya bulunamadı.", true));
        }
    }
}
```

Burada **Source** özelliğinin değerine bakılmakta ve belirtilen dosyanın sistemde gerçekten var olup olmadığı tespit edilmektedir. Eğer söz konusu dosya sistemde yok ise **"Kaynak dosya bulunamadı."** içeriğine sahip bir **uyarı mesajı(Warning)** üretilir. Uyarı mesajı diyoruz çünkü sonda yer alan **true** değeri bunu sağlamaktadır. Yine derleme işleminden sonra oluşacak durum aşağıdaki gibidir.



Tabi eğer **true** yerine **false** değerini kullanırsak aşağıda görülen nur topu gibi **error'** un sahibi oluruz.



Bu adımdan sonra geçerli bir kaynak dosyayı **Source** özelliğine atayarak devam edersek **Destination** özelliğine atanan değer ile ilişkili kontrolleri yapmamız gerektiği sonucuna varabiliriz. Buna göre hedef dosyanın kaynak dosya ile uzantı bakımından uyumlu olması sağlanabilir. Tabiki arka arkaya yapılan çalıştırmalar sonrasında hedef dosya zaten var ise **overwrite** ile ilişkili hataların oluşması da söz konusudur. Bu iki doğrulama işlemini siz değerli okurlarıma bir antrenman olması için bırakıyorum Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

CustomActivityValidation.rar (50,30 kb)

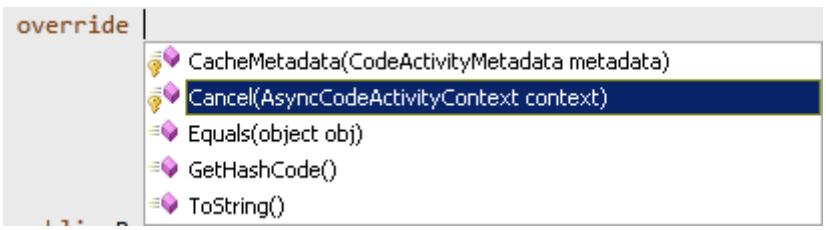
Workflow Foundation 4.0 - Custom Async Activity Geliştirmek [Beta 2] (2010-01-11T00:45:00)

wf 4.0,wf,workflow foundation,

Merhaba Arkadaşlar,

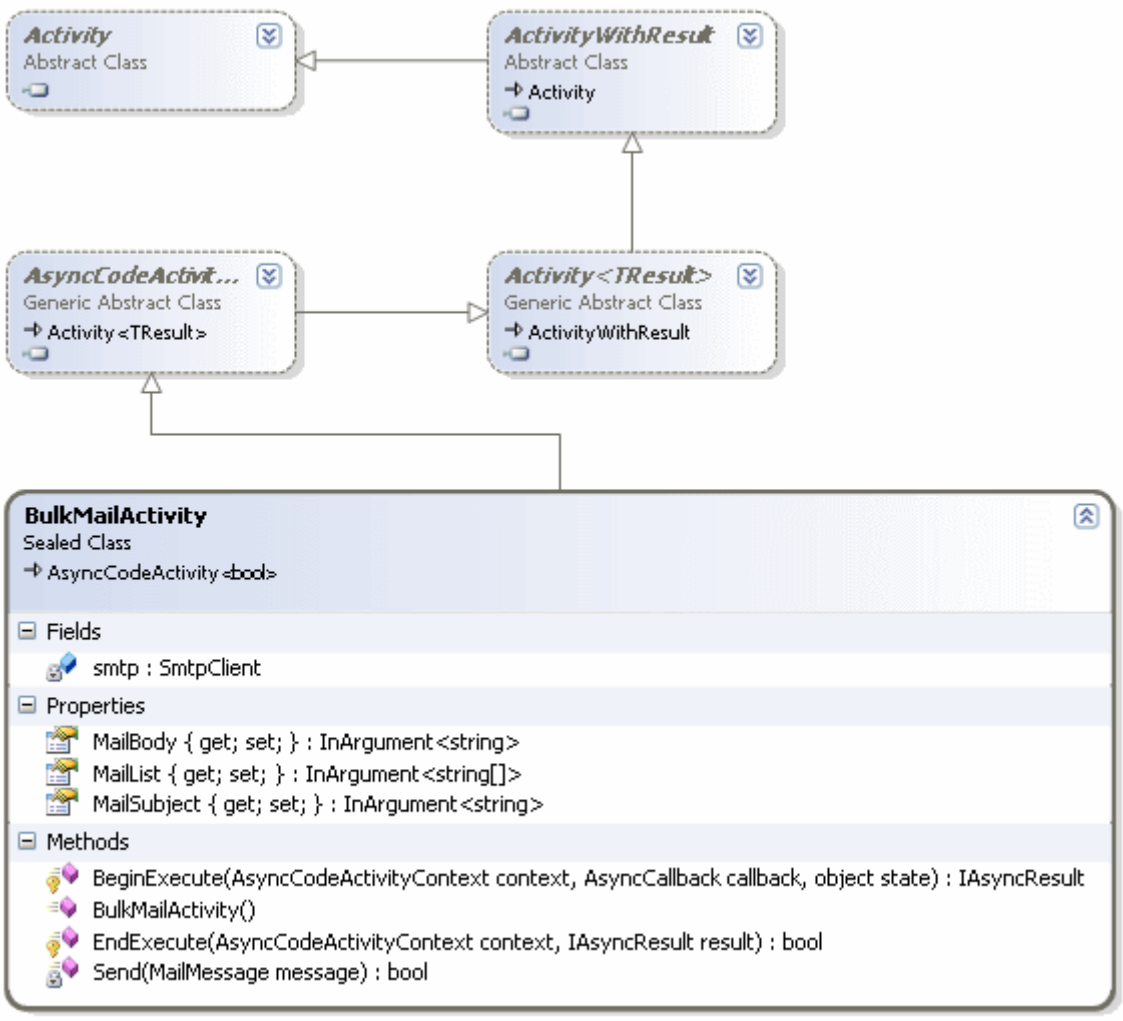
Hatırlayacağınız üzere bir önceki blog yazımızda **Workflow Foundation 4.0** üzerinde özel aktiviteleri nasıl geliştirebileceğimizi incelemeye başlamış ve bu anlamda ilk olarak**CodeActivity<T>** türevli bir bileşen üretmiştik. **Workflow Foundation 4.0** ile gelen önemli yeniliklerden biriside asenkron aktivite bileşenlerini içeriyor olmasıdır. özellikle **.Net Framework 4.0**tarafında üzerinde ağırlıklı olarak durulmaya başlanan paralel programlamanın da bir sonucu olan bu durum karşısında, geliştiricilerin asenkron olarak çalışabilen aktivite bileşenleri yazması pek tabidir.

Asenkron aktivitelerde iki temel operasyon bulunmaktadır. Bu operasyonlardan birisi asenkron olarak çalışacak işlerin başlatıcısı iken, diğerde tamamlandığında devreye girecek olanıdır. Bunlara ilaveten opsiyonel olarak asenkron işlemin iptal edilebilmesi için gerekli operasyonda eklenebilir.



Bu operasyonları içeren asenkron aktiviteleri geliştirmek
için **AsyncCodeActivity<T>** tipinden **türetme(Inherit)** yapmak yeterlidir.

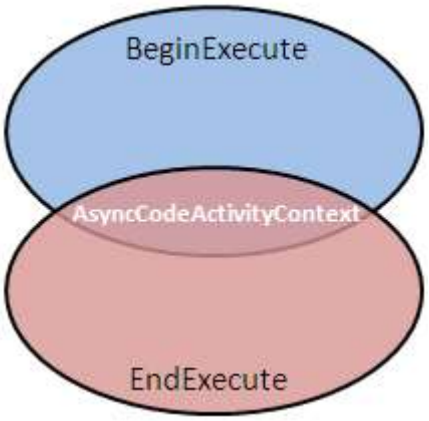
Dilerseniz hiç vakit kaybetmeden basit bir örnek üzerinden devam edelim. Bu amaçla bir önceki projemizde yer alan aktivite kütüphanemize **BulkMailActivity** isimli yeni bir **Code Activity** öğesi ekleyelim. Bu bileşen toplu olarak mail gönderme işlemlerini üstlenecektir. Gerçektende sürecin içerisinde bir noktada yer alabilen toplu mail gönderme adımının, sürecin kalan kısmınıduraksatmaması istenebilir. Böyle bir vakada asenkron olarak mail gönderme işlemini üstelenecek bir aktivite çok yararlı olacaktır. **BulkMailActivity** isimli bileşenimizin sınıf diagramı görüntüsü ve kod içeriği ise aşağıdaki gibidir.




```
using System;
using System.Activities;
using System.Collections.Generic;
using System.Net.Mail;

namespace ActivityLibrary2
{
   public sealed class BulkMailActivity
```

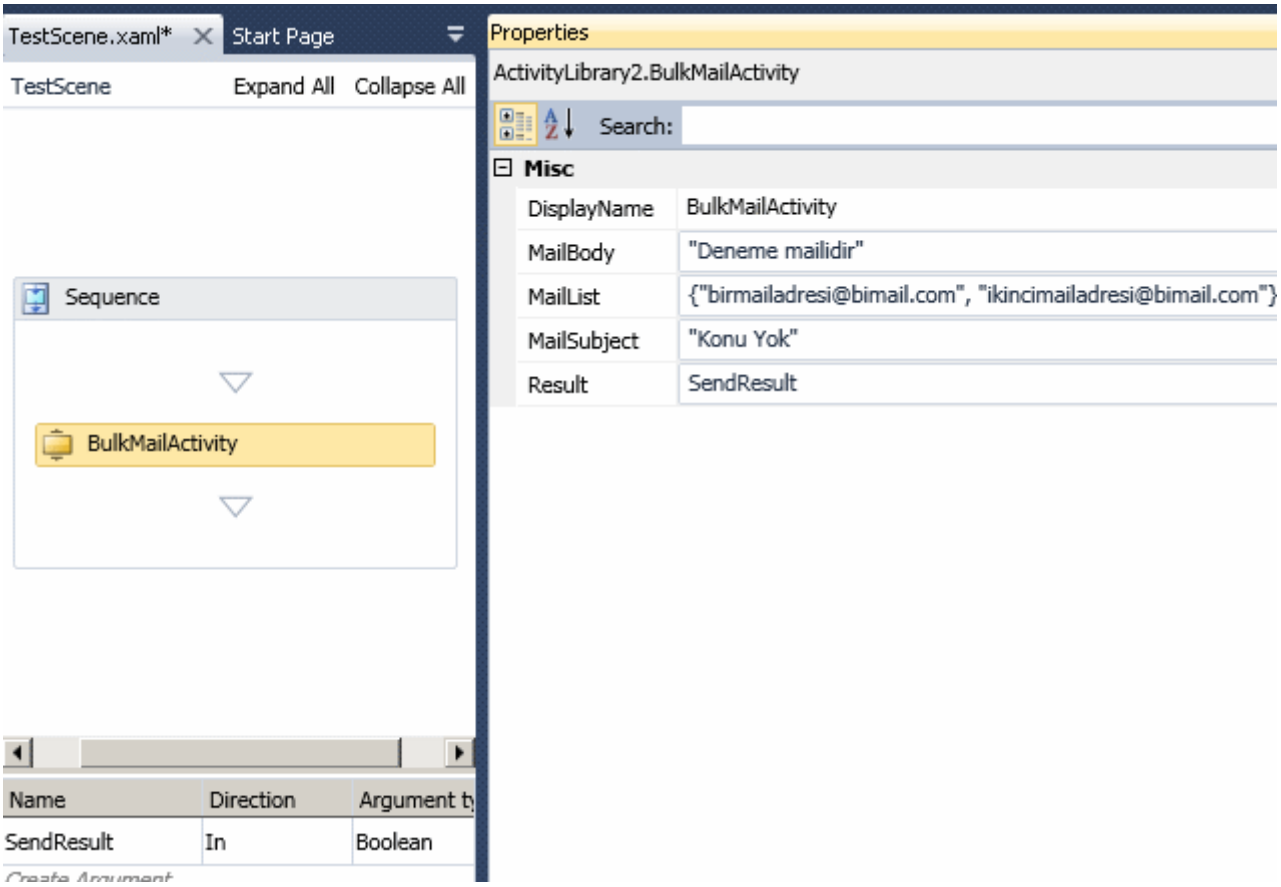
```
    : AsyncCodeActivity<bool>
  {
    public InArgument<string[]> MailList{ get; set; }
    public InArgument<string> MailBody { get; set; }
    public InArgument<string> MailSubject { get; set; }

    private SmtpClient smtp;

    public BulkMailActivity()
    {
      smtp = new SmtpClient("localhost");
    }

    protected override IAsyncResult BeginExecute(AsyncCodeActivityContext
context, AsyncCallback callback, object state)
    {
      MailMessage message = new MailMessage
      {
          Body=MailBody.Get(context),
          Subject=MailSubject.Get(context),
          From=new MailAddress("admin@wf4.com")
      };
      message.To.Add(String.Join(",", MailList.Get(context)));

      Func<MailMessage, bool> SendDelegate = new Func<MailMessage,
bool>(Send);
      context.UserState = SendDelegate;
      return SendDelegate.BeginInvoke(message, callback, state);
    }

    protected override bool EndExecute(AsyncCodeActivityContext context,
IAsyncResult result)
    {
      Func<MailMessage, bool> SendDelegate = (Func<MailMessage,
bool>)context.UserState;
      return SendDelegate.EndInvoke(result);
    }

    bool Send(MailMessage message)
    {
      smtp.Send(message);
      return true;
    }
  }
}
```

Dikkat edileceği üzere **AsyncCodeActivity<T>** türetmesinden dolayı **BeginExecute** ve **EndExecute** isimli iki metodun **ezilmesi(override)** gerekmektedir. **BeginExecute** metodu içerisinde ise**Func<T,TResult>** tipinden bir **temsilci(delegate)** kullanılarak asenkron olarak yürütülecek metodun işaret edilmesi sağlanmaktadır. çok doğal olarak asenkron kodlamada ana fikir,**temsilcilerin(delegate)** üzerinden çağırılan **BeginInvoke** ve **EndInvoke** metodlarıdır. **Func** temsilcisi sayesinde parametre ve geri dönüş tipi belli olan metodun işaret edilerek kullanılabilmesi sağlanmaktadır. Func temsilcisinin bu örnekte kullanılan versiyonuna göre ilk parametre in tipindendir ve Send metoduna gönderilebilecek olan parametreyi belirtmektedir. İkinci generic parametre ise out tipinden olup, Send metodundan döndürülecek olan tipi belirtmektedir. Dikkat edilmesi gereken noktalardan bir diğeride, **BeginExecute** ve **EndExecute** metodları arasında veri paylaşımının nasıl yapıldığıdır.



Yukarıdaki şekil sanıyorum ki bu konuda bir fikir vermektedir. Her iki metodda **AsyncCodeActivityContext** tipinden bir parametre almaktadır. **BeginExecute** metodunda bu parametrenin **UserState**özelliğine **SendDelegate** isimli **Func<MailMessage,bool>** temsilc i örneği atanmıştır. Buna göre **EndExecute** metodu içerisinde **SendDelegate** referansının yakalanması ve çok doğal olarak **EndInvoke** metodunun çağırılabilmesi mümkündür. Aktivitemiz dışarıdan mail listesini, mail gövdesini ve konu kısımlarını almaktadır ki çok daha fazla parametre alabilir.*(Size önerim MailMessage tipinin alabileceği tüm özellikleri içerecek bir mail gönderme aktivite bileşenini geliştirmeye çalışmanızdır)* Gönderme işlemi sırasında makine üzerindeki varsayılan **SMTP** tipi sunucusu kullanılır. Bu nedenle örneğin kendi bilgisayarınızda çalışması sırasında mail gönderme işleminin gerçekleşmemesi mümkündür.

Gelelim test kısmına. Bu amaçla **TestScene.xaml** içeriğini aşağıdaki gibi değiştirelim.



Görüldüğü üzere bileşenimizin ilgili özelliklerine bazı test verileri aktarılmıştır. **MailList** özelliği aslında bir **String** dizisi olduğundan değer ataması yapılırken süslü parantezli bir yazım stili kullanılmalıdır. Bunun dışında mail gönderme bileşeni sonuç olarak **bool** bir değer üretmektedir. Bu değer **SendResult** isimli aktivite bazındaki argümana atanmaktadır.

Xaml kısmı(Sadece bir kısmı verilmiştir);

```
 <x:Members>
    <x:Property Name="SendResult" Type="InArgument(x:Boolean)" />
  </x:Members>
  <mva:VisualBasic.Settings>Assembly references and imported namespaces serialized as
XML namespaces</mva:VisualBasic.Settings>
  <Sequence sad:XamlDebuggerXmlReader.FileName="C:\Documents and
Settings\bsenyurt\my documents\visual studio
10\Projects\ActivityLibrary2\ActivityLibrary2\TestScene.xaml"
sap:VirtualizedContainerService.HintSize="222,200">
    <Sequence.Variables>
      <Variable x:TypeArguments="sd1:DataTable" Name="QueryResult" />
    </Sequence.Variables>
    <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">True</x:Boolean>
```

```
    </scg3:Dictionary>
  </sap:WorkflowViewStateService.ViewState>
  <local:BulkMailActivity sap:VirtualizedContainerService.HintSize="200,22"
MailBody="Deneme mailidir" MailList="[{&amp;quot;birmailadresi@bimail.com&amp;quot;,
&amp;quot;ikincimailadresi@bimail.com&amp;quot;}]" MailSubject="Konu Yok"
Result="[SendResult]" />
 </Sequence>
```

Tabiki bu örnekte sadece **async** tipinden bir **code activity** bileşeninin nasıl geliştirilebileceği ele alınmıştır. Aslında ilave edilmesi gereken ek özelliklerde yok değildir. Söz gelimi mail gönderme işlemi sırasında oluşabilecek **istisnalar(Exception)** sonrasında nasıl davranılacaktır. Bunun için bir **Exception Handling** mekanizmasının kullanılması, bir başka deyişle**try...catch** bloklarına başvurulması gerekebilir. Diğer yandan mailleri virgül ile ayırıp toplu şekilde göndermek yerine tek tek gönderilmesi yolu da tercih edilebilir. Nitekim şu durumda, herhangibir maile yapılan gönderi sırasında oluşacak istisna sonrası kalan gönderme işlemleride icra edilmeyecektir. Sözün özü gerçek hayat senaryolarında bu tip bir bileşenin çok daha titiz yazılması şarttır. Bizim bu yazı için odaklanmamız gereken noktalar ise şunlardır;

- Asenkron aktivite bileşenleri geliştirmek için **AsyncCodeActivity<T>** tipinden türetme yapılır.
- Türetme sonrası **BeginExecute** ve **EndExecute** metodları ezilir.
- İstenirse iptal işlemlerinin ele alınabilmesi amacıyla **Cancel** metoduda ezilebilir.
- Asenkron olarak çalışacak işin doğasında **BeginInvoke** veya **EndInvoke** kabiliyetleri yoksa **Func** gibi temsilcilerden yararlanılabilir.
- **BeginExecute** ve **EndExecute** metodları arasında veri taşınması gerektiğinde, ortak parametre olan **AsyncCodeActivityContext** tipinden yararlanılır. örneğimizdeki gibi illede temsilcinin taşınmasına gerek yoktur. Söz konusu parametre ortak olarak kullanılacak bir referansın taşınması istendiğinde değerlendirilmelidir.
- çalışma zamanı, Workflow içerisinde bir asenkron bileşen ile karşılaştığında bunu yürütmeye başlar ve hemen sonraki adımdan işleyişine devam eder.

Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ActivityLibrary2Async.rar (54,25 kb)

Webiner - C# 4.0 - Yenilikler [Beta 2] (2010-01-07T02:00:00)

c# 4.0,.net framework 4.0,webiner,webcast,

Merhaba Arkadaşlar,

çok eskiden bir **Delphi** programıcısydım. Açıkçası o dönemlerde **Delphi** geliştirme ortamının hayranı olduğumu itiraf etmeliyim. **Delphi 1.0** ile başlayan profesyonel iş yaşantımda ilk geliştirmeye çalıştığım ve neredeyse para kazanmak üzere olduğum programı üniversiteden çok değerli bir sınıf arkadaşım*(Orkun Şentürk)* ile birlikte yazamamıştık. 😕 Arkadaşımın motorsiklet satın aldığı bir mağazaya indirim karşılığında basit bit stok takip programı yazacaktık. Ne varki arkadaşım bir gün motorsikletinin sinyal lambasını kırdı ve **100 Mark** değerinde olan lambayı karşılamak için programa yeni özellikler dahil ediliverdi.😃

O zamanlar **Delphi 2.0** sürümüne henüz kavuşamamıştık ve **Delphi 1.0** sürümünde veritabanı işlemleri ile ilişkili önemli sıkıntılar vardı. Sonuçta programı yazamadık. Onun yerine bir muhasebe programı satın almalarını*(MSDOS tabanlı)* önerdik. Tabi yine aynı arkadaşımla şeytanın bacağını bir sonraki projemizde kırdık ve yine Delphi 2.0 ile başarılı bir program yazıp satmayı başardık.

Ancak profesyonel yazılım geliştirme yaşantım zaman içerisinde **Anders Heijslberg(C# ın babası diyebiliriz)'** in **Microsoft** takımına geçmesiyle tamamen yön değiştirdi. **Anders'** in**Microsoft'a** geçişinden kısa bir süre sonra **.Net Framework** platformunu ve **C#** programlama dilini duyduk. Akabinde tabiki **Visual Studio.Net** geliştirme ortamı. **C#** programlama dilini daha görür görmez benimsemiş ve etkilenmiştim. Zaman ilerledikçe C# programlama dilide önemli gelişmeler gösterdi tabiki. çünkü yazılımların ihtiyaçları, gereksinimleri günden güne artıyor ve programlama dillerini kendilerine uymaları konusunda zorluyordu.

Derken **.Net Framework 2.0** köklü bir değişikliğin eklendiğini gördük. **Generic** mimari. Köklüydü çünkü **CLR(Common Language Runtime)** üzerinde değişikliklerin yapılmasını gerektiriyordu. çok doğal olaraktan dil içerisinde buna destek verecek yenilikler yapıldığına ve dile özgü başka ilavelerin getirildiğine tanık olduk. **Anonymous metodlar, yield anahtar kelimesi**, **static sınıflar** vb...Tabiki yapılan yeniliklerin belirli amaçları olduğu kesindi. Bu amaçların iyice belirgineştiği yer ise **.Net Framework 3.5**sürümüydü. Bu vesileyle **Language INtegrated Query(LINQ)** hedefli olarak gelen yenilikler ve bunun için dilde yapılan geliştirmelerle karşılaştık. **Extension metodlar,**

Anonymous tipler, lambda(=>) operatörü, Auto Property' ler, Object Initializer' lar, partial event' ler vb...

Uzun bir süredirde **C# 4.0** dili birlikte gelmesi muhtemel yenilikler söz konusu. Halen daha **Beta 2** aşamasındayız ancak yakın zamanda **RC** versiyonu ile birlikte nihai sürüme ulaşacağımızı biliyoruz. *(Her ne kadar zamanında bazı kişiler fake olarak C# 4.0 yeniliklerini bloglarında duyurup yayınlasalarda gerçekleri yansıtmıyorlardı)* özellikle **Dinamik Dil çalışma Zamanı(Dynmaic Language Runtime)** ile olan etkileşimin arttırılmasını hedef alan yeni dil özelliklerini NedirTV?com adına düzenlenen webinerimizde detayları ile inceledik. Umarım sizler için yararlı olmuştur.

Süre **58:52**

WCF Eco System (2010-01-05T10:30:00)

wcf,wcf ria services,workflow services,data services,ado.net data services,web http services,soap services,

Merhaba Arkadaşlar,

özellikle son bir iki yıllık zaman dilimi içerisinde **.Net** tarafında pek çok servis modeli ve ismiyle karşılaştık. örneğin **Astoria** kod adıyla başlayan **Ado.Net Data Services**, **Silverlight** gibi**Rich Internet Application'** ları hedef alan **.Net RIA Services** vb... (*Eğer Microsoft' un ürünleri için kullandığı kod adlarını merak ediyorsanız Wikipedia' daki ilgili listeye bakmanızı öneririm)* Hal böyle olunca ortada bir sürü kod adı ve isim oluşmaya başladı. Buda çok doğal olarak bizim gibi geliştricilerin kafasında pek çok soru işaretine neden oldu. Acaba hangi servis modelini hangi amaçlar ile kullanmalıyız? Bunların nihai sürümler yaklaştıkça isimlendirmeleri neler olacak? Ne gibi avantaj veya dez avantajları var?

Soruları arttırmak mümkün. Aslında kabul edilmesi gereken önemli bir nokta var; Tüm bu servis modelleri **.Net Framework 3.0'** dan beri var olan ve her sürümde önemli yetenekler kazanan**Windows Communication Foundation(WCF)** alt yapısı(Infrastructure diyebiliriz) üzerinde konuşlandırılmış durumda. Şu an içinde bulunduğumuz servis modellerinin çeşitliliğini ve sayısını düşündüğümüzde ise bir **Eco System'** in oluştuğunu net bir şekilde ifade edebiliriz. Aşağıdaki tabloda **WCF Eco System'** in parçaları yer almakta olup kısaca amaçları özetlenmeye çalışılmaktadır.

bility standartlarına uygun olan böylece örneğin **Java** gibi platformlar ile konuşabilen, **mesaj tabanlı güvenliğ**
nsaction akışına(Transaction Flow) izin veren, **IIS** üzerinden **HTTP** tabanlı veya **IIS** dışından host edilebile
orms veya WPF uygulaması yada basit bir Console programı olarak), pek çok **WS-*** standardını destekleyen t

li **.Net Framework 3.0** versiyonundan bu yana mevcuttur. Geliştiricilere çok daha fazla kontrol imkanı sunan
llerindeki gibi belirli bir konuya odaklanmaktan ziyade ihtiyaçlara göre düşünülen çözümlerde değerlendirilir.

rm Resource Identifier) bazlı olaraktan servis operasyonlarının **RESTful** yaklaşımına göre sunulduğu fonksiy
WCF tarafındaki bu yetenekler **.Net Framework 3.5** ile birlikte gelen **Web programlama modeli(Web Progr**
olup **.Net Framework 4.0'** da ek özellikler ile arttırılmıştır. Bu modelde verinin **Get,Post,Put** ve **Delete** gibi **H**
nulması mümkündür. Geliştiriciler verinin sunulması sırasındaki **URI** bilgisine, çıktı formatına*(örneğin JavaS*
dahalede bulunabilir.

imizi(Data Model) ve bu modelin içerdiği iş mantığın bir **RESTful** arayüzü üzerinden sunmak istediğimiz du
r. **.Net** için **Open Data Protocol** desteğini de içermektedir. Aslında ilk olarak **.Net Framework 3.5 Service Pa**
yla ortaya çıkmıştır. Bu yaklaşımda servisin sunacağı veri kaynağına ulaşırken **Ado.Net Entity Framework** g
ilir. Ancak istenildiğinde**Custom LINQ Provider'** lardanda faydalanılıp farklı veri kaynaklarının kullanılması

*ces ile **Data Services** zaman zaman bir birlerine karıştırılmaktadır. Aslında **Data Services**, veri modelinin ve b*
*e ilgilenmekte iken **RIA Service'** leri **Silverlight** uygulamalarının **end-to-end** modelinde geliştirilmesine odakl*
rağmen ilgilendikleri vakalar tamamen farklıdır.

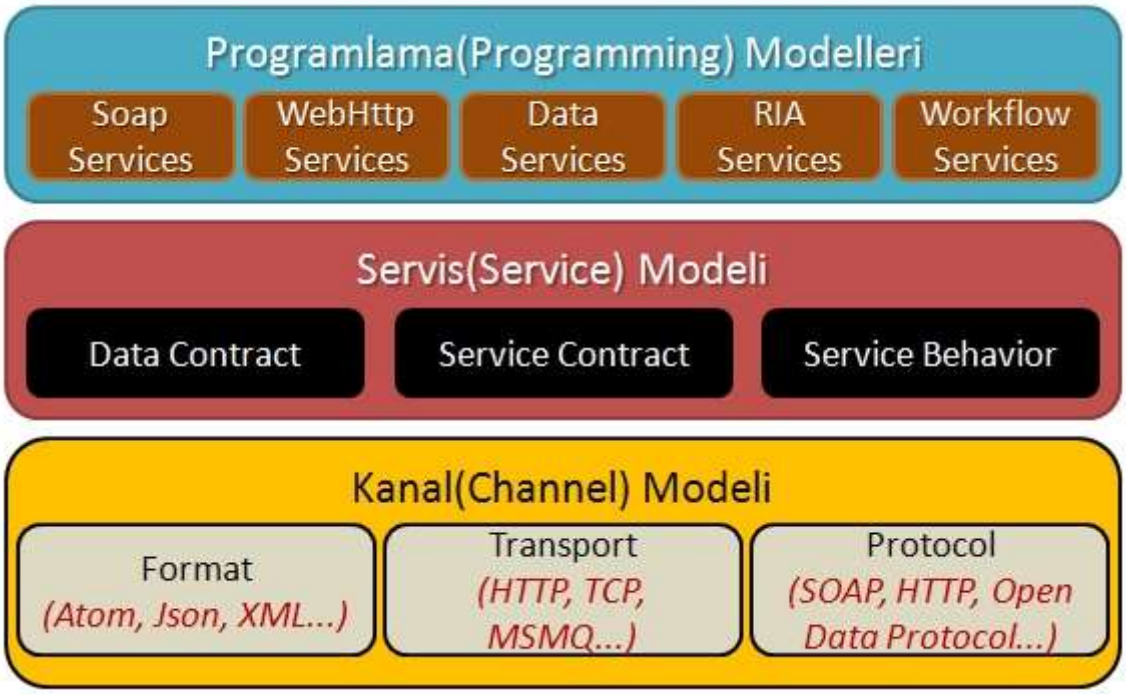
***a Service'** lerin zaman zaman **WebHttp Service'** leri ile karıştırılmasıda söz konusudur. Ancak yine her iki ser*
***ebHttp Service'** leri **URI, format** ve **protocol** bilgileri üzerinde tam yönetime izin verecek şekilde **RESTful** ser*
***Service'** lerde **RESTful**arayüzü zaten hazırdır ve sadece veri modeli ile ilişkilendirilmesi yeterlidir.*

i(Long Running) ve **kalıcılık(Persistence)** desteği olması gereken **Workflow** uygulamalarının servis bazlı ola
u modele göre **Workflow** nesnelerinin **WCF** servisi olarak sunulması, tüketilmesi ve ayrıca **Workflow** nesnel
erlendirilmesi mümkündür.

nework 3.0' da birbirlerine uzaktan bakan **WF** ve **WCF** alt yapıları, **.Net Framework 3.5** ile flört etmeye başl

gibi **Rich Internet Application(RIA)** uygulamalarında orta katmandaki iş modelinin servis bazlı olarak hem i
ni, oluşturulmasını ve kullanılmasnı kolaylaştıran **end-to-end** servis modelidir. önceki adı **.Net RIA Services** o
Silverlight 4.0 sürümünde nihai versiyonuna ulaşacağı tahmin edilmektedir.

Tabi buradaki bilgiler dışında aşağıdaki çizelgeyide göz önüne almamızda yarar vardır. Bu çizelgede WCF Eco System' in mimari modeli gösterilmektedir.



Aslında **WCF** alt yapısı üzerinde duran servis programlama modelleri, geliştiricilerin vakaya göre WCF alt yapı detaylarından uzaklaşmasını sağlamaktadır. Bu, özellikle **Data Services** ve**RIA Services** modellerinde ön plana çıkmaktadır. Yine de istenildiğinde bu modelleri esnetebileceğimizi bilmeliyiz. Hatta bana kalırsa **WCF** alt yapısı üzerine oturan kendi programlama modellerimizi de geliştirebiliriz. Ancak buna gerek olup olmadığını tartışmalıyız.

Umarım WCF Eco System hakkında biraz fikir sahibi olabilmişizdir. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

[NedirTv?com Ocak 2010 Webinerleri (2010-01-04T11:15:00)]

webiner,webcast,

Merhaba Arkadaşlar,

Editör olduğum NedirTv?com sitesi webinerleri **2010** yılına yeni girdiğimiz şu günlerde **Ocak** ayı boyunca da devam ediyor olacak. Bu ayın konuları arasında **C# 4.0, İnternet Kazanç Sistemler, Workflow Foundation 4.0, Microsoft SQL Server 2008 İş Zekası, Asp.Net 4.0 Veriye Erişimde Yenilikler,WCF RIA Services, Parallel Programming** yer almakta. Hafta için günlerine denk gelen ve 21:00 başlangıç saatli olan

Webinerleri sektörün önde gelen isimleri sunmakta. Webiner programına ve katılmak için gerekli **Live Meeting** linklerini aşağıda bulabilirsiniz. Katılımlarınızı bekliyoruz.

Not : Etkinliği facebook'taki ajandanıza kaydetmek ve etkinlik duyurularını takip etmek için bu linki kullanabilirsiniz.

Konu: C# 4.0 Yenilikleri
Zaman: 6 Ocak çarşamba – 21:00
Link: https://www.livemeeting.com/cc/mvp/join?id=C6M9S4&role=attend
Konuşmacı: Burak Selim ŞENYURT

Konu: İnternet Kazanç Sistemleri(Affiliate Marketing)
Zaman: 10 Ocak Pazar - 21:00
Link: https://www.livemeeting.com/cc/mvp/join?id=DQW34N&role=attend
Konuşmacı: Cihan BAÄžDATLI

Konu: WCF RIA Services – Authentication, Authorization, Profile
Zaman: 13 Ocak çarşamba – 21:00
Link: http://msevents.microsoft.com/CUI/EventDetail.aspx?EventID=1032439658&Culture=TR-TR
Konuşmacı: Burak Selim ŞENYURT

Konu: Microsoft SQL Server 2008 ile İş Zekası-1
Zaman: 16 Ocak Cumartesi – 21:00
Link: http://msevents.microsoft.com/CUI/EventDetail.aspx?EventID=1032439660&Culture=TR-TR
Konuşmacı: Osman çOKAKOÄžLU

Konu: Workflow Foundation 4.0(Beta 2)
Zaman: 20 Ocak çarşamba – 21:00 – 22:00
Link: http://msevents.microsoft.com/CUI/EventDetail.aspx?EventID=1032439662&Culture=TR-TR
Konuşmacı: Burak Selim ŞENYURT

Konu: ASP.NET 4.0(Beta 2) - Veriye Erişimde Yenilikler
Zaman: 22 Ocak Cuma - 21:00-22:00
Link: http://msevents.microsoft.com/CUI/EventDetail.aspx?EventID=1032439664&Culture=TR-TR
Konuşmacı: Uğur UMUTLUOÄžLU

Konu: .Net 4.0 ile Paralel Programlama
Zaman: 27 Ocak çarşamba - 21:00-22:00
Link: http://msevents.microsoft.com/CUI/EventDetail.aspx?EventID=1032439666&Culture=TR-TR
Konuşmacı: Burak Selim ŞENYURT

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Workflow Foundation 4.0 - Custom Activity Geliştirmek [Beta 2] (2010-01-01T16:00:00)

wf 4.0,

Merhaba Arkadaşlar,

Programlamaya profesyonel olarak adım attığım yıllarda henüz **.Net** mimarisi geliştirilmeden önce **Delphi** programlama dili ile ürünler yazmaya çalışırdım. Aslında **.Net** çıktığından beri uğraşmadığım **Delphi** programlama dilini düşündüğümde aklıma ilk gelen **hızlı geliştirme(Rapid Development)** için sunduğu zengin **Component** sekmeleridir. Sayısız bileşen sayesinde müşteri ihtiyaçlarına çok hızlı cevap verebilecek ürünler geliştirebildiğimi de gayet net hatırlayabiliyorum. Tabi **.Net** dünyasına geçiş yapıp Visual Studio.Net(2005,2008,2010) ortamı ile karşılaştırıldığında çok fazla bileşen olduğu göze çarpmaktaydı. Ancak **.Net** tarafında da kodlama yeteneklerinin daha ön plana çıktığı bir gerçekti.

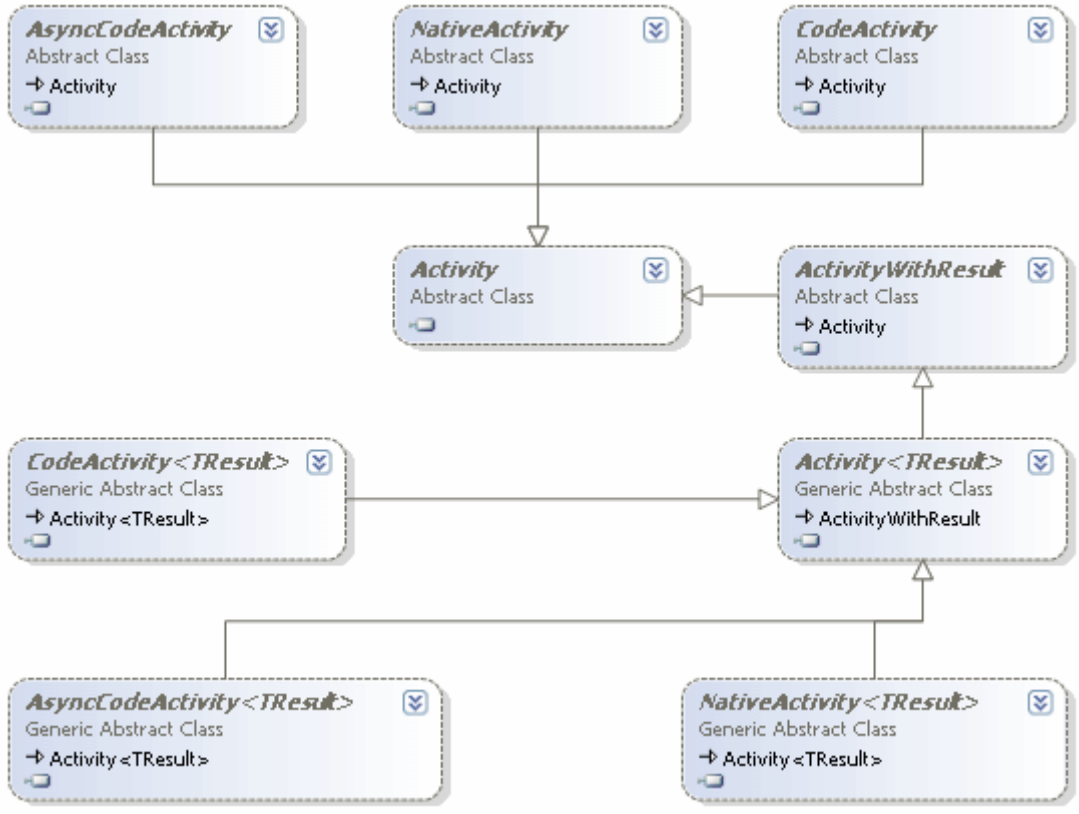
Ancak ister **Delphi** olsun ister **.Net** ister **Java** platformu, görsel program geliştirme ortamlarında var olan bileşenlerin yeterli gelmediği durumlar ile karşılaşılabilir. Bu durumda, geliştiriciler kendi bileşenlerini yazmaya çalışırlar(Yada ihtiyaçlarını karşılayan bileşenleri satın alma yoluna giderler). Bunun pek çok nedeni olabilir. Bu nedenler basit **Web** tabanlı bir **TextBox** kontrolünün güvenlik açığı, çok sık kullanılan **IPV4**girişleri için bir kontrolün bulunmayışı veya birden fazla kontrolün birleşiminin pek çok yerde kullanılması gerekliliği vb olabilir...

özel bileşenlerin geliştirilmesi genellikle bellidir. Bunlardan birisi bileşenin sıfırdan yazılmasıdır. Bu zaman zaman zahmetli bir yoldur. Söz gelimi **Windows** veya **Web** tarafında ekrana basılacak olan çıktının üretilmesi için gerekli kodlamaların yapılması gerekir. Bu **Windows** tarafında **Paint** olayının ezilerek **GDI/DirectX** gibi tekniklerin konuşturulması veya **Web** tarafında**HTML** ve **Javascript** destekli kodlamaların yapılmasını gerektirebilir.

İkinci bir yol var olan bileşenlerin tasarım ortamında bir arada değerlendirilerek composite üretimlerin gerçekleştirilmesidir. Yine **Windows** ve **Web** tarafındaki **User Control**' leri bu anlamda düşünebiliriz. Diğer bir teknik ise, var olan bir bileşenden türetme yoluna gidilerek geliştirmenin yapılmasıdır ki bu çok sıkça tercih edilmektedir. Peki **Workflow Foundation** alt yapısında kendi bileşenlerimizi nasıl yazabiliriz? Aslında **Workflow Foundation** tarafında kullanılan **Activity**' lerin,**Windows** veya **Web** tarafındaki görsel olan/olmayan kontrollerden pekte farkı yoktur. Aktivite' lerde **Workflow Foundation** tarafında geliştirici tarafından değerlendirilen özel bileşenlerdir.

İşte bu yazımızda **Workflow Foundation 4.0** alt yapısında, kendi aktivite bileşenlerimizi nasıl geliştirebileceğimizi incelemeye çalışıyor olacağız. An itibariyle **Workflow Foundation 4.0 Beta 2** sürümü üzerinden ilerleyeceğiz. Aslında **Custom Activity** geliştirmek için pek çok neden sıralanabilir. Genel olarak var olan **built-in** aktivite bileşenlerinin yeterli gelmediği veya özel business logic' lerin içeren aktivitelerin pek çok akışta kullanılması gerektiği durumlarda özel aktivite bileşenlerinden yararlanılabilir. özel aktiviteleri geliştirmeye başlamadan önce **.Net Framework 4.0 Beta 2** bünyesindeki aktivite hiyerarşisini iyi bir şekilde incelemek gerekmektedir.

Genel Aktivite hiyerarşisi;



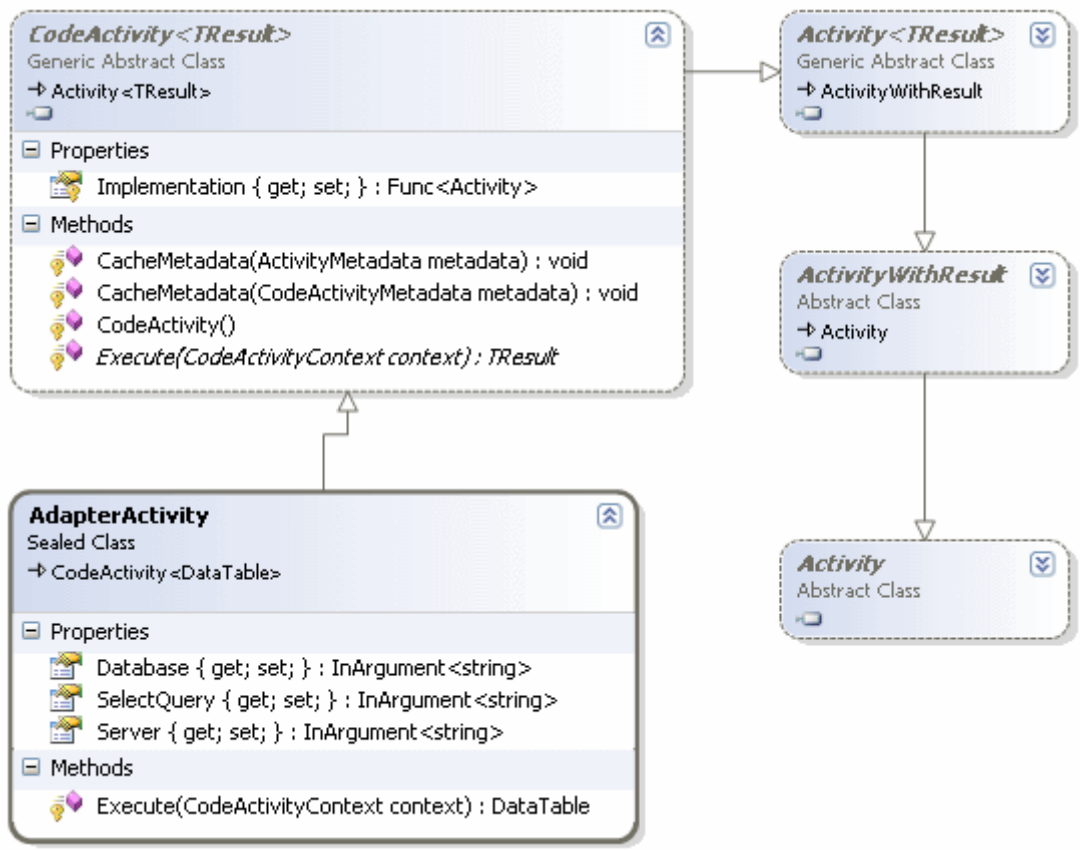
Burada önemli olan nokta, her aktivitenin bir **Activity** tipi olması gerekmediğidir. Evet nesne yönelimli teoriye göre böyledir ancak mantıksal çerçevede böyle değildir. özellikle generic ve normal tip yönelimlerine bakıldığında amaca göre uygun olan aktivite bileşeninden türetilme yoluna gidilmesinin tercih edillmesi gerektiği ortadadır. Bu nedenle

doğrudan **Activity** bileşeni yerine**CodeActivity, AsyncCodeActivity, NativeActivity** veya bunların generic olan versiyonlarından türetme işlemleri yapılmaktadır. Aslında doğrudan **Activity/Activity<T>** tipinden türetme yolu da tercih edilebilir. Aslında biraz kafamızın karıştığı ortadadır. Hangi türetmeyi seçeğimize karar vermek için şu noktalara dikkat etmemiz yeterli olacaktır;

- Diğer aktivitelerin bir arada kullanılmasından ve doğrudan **XAML** tanımlamalarından oluşturalacak özel aktivite bileşenlerinin*(bunları Asp.Net veya Windows uygulamalarındaki User Control' lere benzetebiliriz)* **Activity/Activity<T>** tipinden türetilmesi,
- özel aktivitenin çalışma zamanında sonlandırılması sırasında bazı kodlamalara ihtiyaç duyuluyorsa **CodeActivity**,
- Aktivite bileşeninin içerdiği işlemleri asenkron olarak yürütebilmesi isteniyorsa **AsyncCodeActivity**,
- Eğer özel aktivite bileşeni çalışma zamanı verilerine erişecekse*(örneğin diğer aktivitelere ulaşılıp kataloglanması, takvimlendirilmesi-scheduling vb...)* **NativeActivity**,

türevli olması tercih edilmektedir.

Görüldüğü üzere aslında özel aktivite bileşeni geliştirmek için birden fazla yol vardır. Dilerseniz konuyu basit bir örnek ile süsleyerek ilerlemeye çalışalım. Bizim için en basit olanı seçeceğiz. Bu amaçla **CodeActivity<T>** türevli bir özel bileşen geliştirmeye çalışacağız. Söz konusu bileşeni **Workflow Activity Library** tipinden olan proje içerisine yeni bir **Code Activity** öğesi ekleyerek örnekleyebiliriz. **AdapterActivity** ismiyle geliştireceğimiz bileşenimizin temel amacı Select sorgusu ile bir veritabanı tablosunun içeriğini DataTable tipinden bir referans olarak geriye döndürmektir. **AdapterActivity** isimli örnek bileşenimizin kod içeriği aşağıdaki gibidir.




```
using System;
using System.Activities;
using System.Data;
using System.Data.SqlClient;

namespace ActivityLibrary2
{
   public sealed class AdapterActivity
      : CodeActivity<DataTable>
   {
      // Aktivite içerisinde kullanılan bazı özellikler
      public InArgument<string> Server { get; set; }
      public InArgument<string> Database { get; set; }
      public InArgument<string> SelectQuery { get; set; }

      // Execute metodu ezilir. AdapterActivity tipi, CodeActivity<DataTable> sınıfından türediğinden Execute metodunun DataTable türünden bir değer döndürmesi sağlanmalıdır
      // Execute metodunun dönüş değeri, Designer tarafında Result isimli özellik ile yakalanabilir
      protected override DataTable Execute(CodeActivityContext context)
      {
         // InArgument<T> tipinden olan değişken değerlerinin çalışma zamanında alınması için Get metodlarından yararlanılır. Metod parametre olarak o anki Aktivite içeriğinin
```

referansını kullanır.

```
        string conStr = String.Format("data source={0};database={1};integrated
security=SSPI", Server.Get(context), Database.Get(context));
        DataTable table=null;
        using (SqlConnection conn = new SqlConnection(conStr))
        {
           SqlDataAdapter adapter = new SqlDataAdapter(SelectQuery.Get(context), conn);
           table = new DataTable();
           adapter.Fill(table);
        }
        return table;
      }
   }
}
```

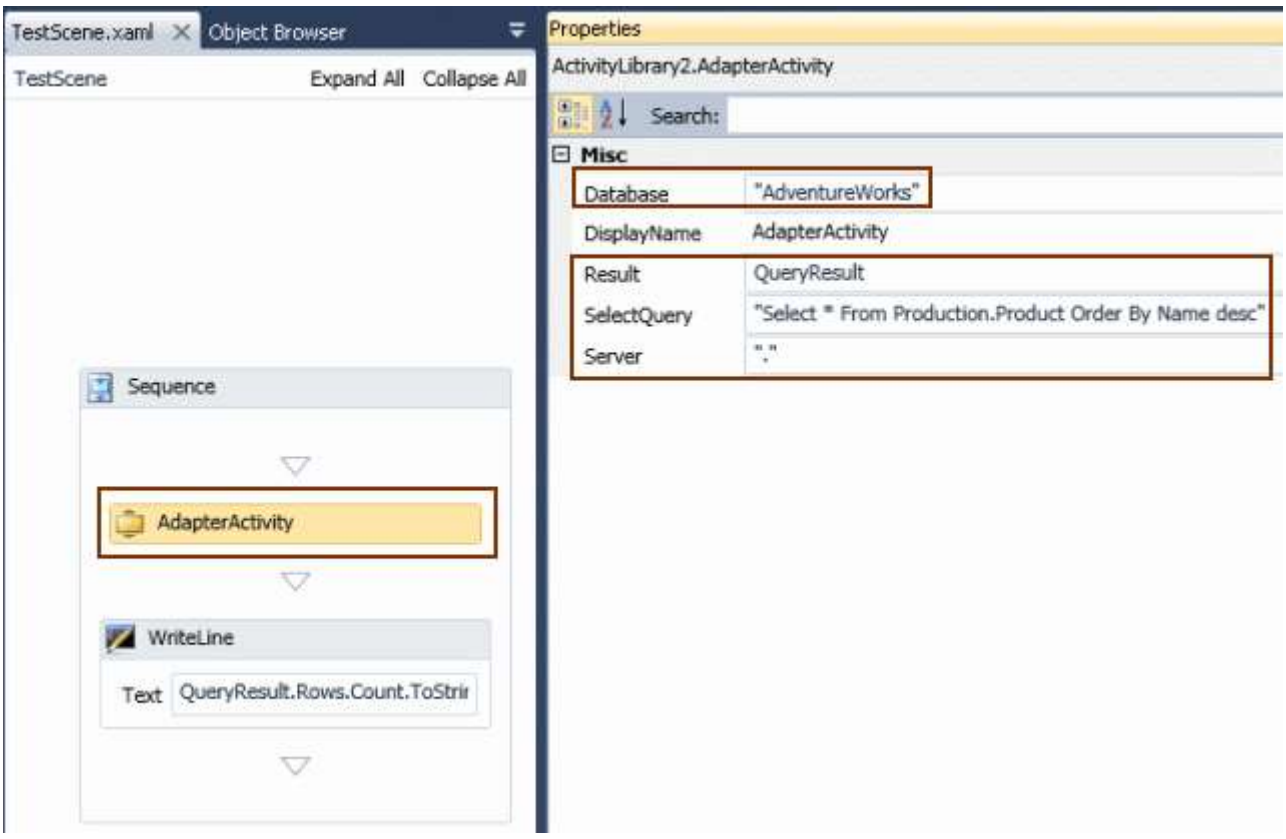
CodeActivity<T> türevli olan **AdapterActivity** bileşeni, bir **Select** sorgusunu çalıştırıp sonucu **DataTable** içerisine aktarmak üzere tasarlanmıştır. çok tabi olarak örneği basit tuttuğumuzu ifade etmek isterim. **UserId,Password** gibi **ConnectionString** için önemli olan alanlar yer almamaktadır. Bunun yerine varsayılan olarak Windows doğrulama modunda bağlanılabildiği düşünülmektedir. üstelik yazılan SQL sorgusunun geçerli olup olmadığına dair bir kontrolde bulunmamaktadır ki olmasında yarar vardır. Nitekim hatalı veya **SQL Injection** saldırılarına açık tipte bir SQL sorgusunun yazılabiliyor olması arzu edilmeyecektir. Aslında tüm bu iki handikap sadece tablo adının alınıp Select sorgusunun kod içerisinde dinamik olarak oluşturulmasıyla da çözümlenebilir. Ancak odaklanmamız gereken daha önemli bir mevzu vardır. **Override** edilimiş olan **Execute** metodu. Bu metod söz konusu kod aktivitesi çalıştığında işletilecek olan kodları içermektedir. Dikkat edileceği üzere o an üzerinde çalıştığı aktivitenin çalışma zamanı içeriğini kullanabilmesi için **CodeActivityContext** tipinden bir parametre ile süslenmiştir. Bu parametre kod içerisinde **Server, Database** ve **SelectQuery** gibi özelliklerin çalışma zamanı değerlerini almak için çağırılan **Get** metodlarında parametre olaraktan da kullanılmaktadır. Dikkat çekici bir diğer nokta **Execute** metodunun dönüş tipidir. Varsayılan olarak **void** olan **Execute** metodu, **AdapterActivity'** nin**CodeActivity<DataTable>** türevl i olması nedeniyle geriye **DataTable** döndürmektedir. Nitekim **generic** olarak yapılan türetmeye göre **T** tipi, aynı zamanda **Execute** metodunun geri dönüşü tipidir.

Bu basit bileşeni test etmek için aşağıdaki **XAML** ve tasarım zamanı içeriğine sahip **TestScene** isimli bir aktivite oluşturduğumuzu düşünebiliriz. *(Uzun olan Namespace tanımlamaları göz ardı edilerek sadece Sequence içeriği verilmiştir)*

```
<Sequence sad:XamlDebuggerXmlReader.FileName="C:\Documents and
Settings\bsenyurt\my documents\visual studio
10\Projects\ActivityLibrary2\ActivityLibrary2\TestScene.xaml"
sap:VirtualizedContainerService.HintSize="232,245">
  <Sequence.Variables>
    <Variable x:TypeArguments="sd1:DataTable" Name="QueryResult" />
```

```
    </Sequence.Variables>
    <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">True</x:Boolean>
      </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <local:AdapterActivity Database="AdventureWorks"
sap:VirtualizedContainerService.HintSize="210,22" Result="[QueryResult]"
SelectQuery="Select * From Production.Product Order By Name desc" Server="."
/>
    <WriteLine sap:VirtualizedContainerService.HintSize="210,59" Text="[QueryResult.
Rows.Count.ToString()]" />
  </Sequence>
```

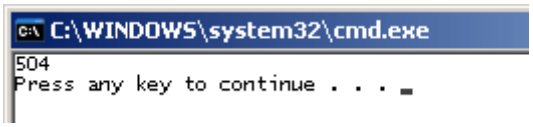
Tasarım Zamanı(Design Time);



Tasarım zamanında dikkat edileceği üzere kod tarafında eklediğimiz **Database, Server** ve **SelectQuery** gibi özelliklere ulaşılabilmektedir. **TestScene** aktivitesi **QueryResult** isimli bir **Variable**değerine sahiptir. Bu değişkenin içeriği ise **AdapterActivity** isimli bileşenin **Result** özelliğidir. Bir başka deyişle **CodeActivity<T>** türevli özel aktivitenin **Execute** metoduna ait dönüş değerinin, **Result** isimli özelliğe aktarıldığını söyleyebiliriz.
Gelelim **TestScene.xaml** aktivitesinin çalıştırılacağı kodlara;

```
using System.Activities;

namespace ActivityLibrary2
{
   public class Program
   {
      static void Main(string[] args)
      {
         var result=WorkflowInvoker.Invoke(new TestScene());
      }
   }
}
```

Görüldüğü gibi tek yaptığımız **WorkflowInvoker** tipinin **Invoke** metodu ile yeni bir **TestScene** aktivite örneğini çalıştırmaktır. Test için **AdventureWorks** veritabanında yer alan **Products** tablosunu kullandığımızda, standart olarak **504** olan satır sayısı değerinin elde edilmesi gerekmektedir. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.



Custom Activity geliştirmek ile ilişkili diğer teknikleri de ilerleyen yazılarımızda ele almaya çalışıyor olacağız. Söz gelimi aktivitilerin çoğu için tasarım zamanı desteği bulunmaktadır. Kendi geliştireceğimiz aktivite bileşenleri için bu tip destekleri sunabilir miyiz? Sunabilirsek nasıl?

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ActivityLibrary2.rar (47,00 kb)

Webiner - .Net 4.0 ile Paralel Programlamaya Giriş [Beta 2] (2009-12-26T01:45:00)

task parallel library,plinq,parallel programming,webcast,webiner,

Merhaba Arkadaşlar,

Bundan yıllar önce üniversitede öğrenciyken ilk kişisel bilgisayarımı almak üzere rahmetli babam ile birlikte **Perpa'** ya gittiğimi hatırlıyorum. O sıralar piyasada var olan bilgisayar dergilerini hatırlıyorum da...**486 DX-33** tabanlı işlemcilere sahip sistemler anlatılıyor ve öneriliyordu. Ancak bütçe belli olunca **Perpa'** daki mağazada ancak **486 SX-25** işlemcili bir modeli alabileceğimizi farketmiştik. Aradaki tek farkın, **SX-25'** in **DX-33**'e göre matematiksel hesaplamalarda daha yavaş olması olduğunu hatırlıyorum. Tabi bu eksiklik, matematiksel hesaplamalarda, dosya giriş çıkış işlemlerinde, sunucu bazlı süreçlerde, yapay zeka algoritmalarında, grafik işleme uygulamalarında dez avantaj oluşturacak bir fark yaratıyordu. özellikle hız ve performans açısından. Ancak o zamanlar daha çok oyunlara olan etkisini düşünüyordum.

Gel zaman git zaman **Intel** firmasının kurucusu **Gordon Moore'** un meşhur yasasını okudum. 18 ayda bir ikiye katlanan ve üretim maaliyetleri aynı kalan yada azalan transistor sayılarından bahsediyor ve hızların nasıl artacağını ifade ediyordu. *(Tabi bu kanun günümüzde geçerliliğine neredeyse yitirdi.)* Sonrasında ise işin ardı arkası kesilmedi. **200 MMX, Pentium, Celeron** işlemciler derken bir baktık diz üstü bilgisayarlarımızda bile **4** çekirdek var. *(örneğin şu an bu yazıyı yazdığım bilgisayarda çift çekirdek işlemci bulunmakta)* Artık **32** işlemcili bir bilgisayarın maaliyeti bundan onlarca yıl öncesi kadar yüksek değil. Hatta bu kadar çok işlemciye sahip sistemleri kurmak artık hayal değil. Hal böyle olunca, uygulamalarda **Multi-Thread** yaklaşımlarda bir hayli önem kazandı. özellikle son yıllarda **Microsoft** bu anlamda yazılımcıların daha kolay kod geliştirebilmesine olanak tanıyan yenilikler duyurdu. İşte bunlardan biriside paralel programlama alt yapısı. Bu Webinerimizde **.Net Framework 4.0** içerisine gömülü olarak gelen paralel programlama alt yapısını inceldik. Kaçıran arkadaşlarımız **NedirTV?com** adresinden söz konusu webineri indirip izleyebilirler. İyi seyirler dilerim.

Süre : 58:55

.Net 4.0 Öncesi ThreadPool Kullanımı (2009-12-23T12:15:00)

c#,.net framework,

 Merhaba Arkadaşlar,

İlk okulda eminimki pek çok arkadaşımız havuz problemlerinden müzdarip olmuştur. Genellikle bu havuzlarda ikiden fazla musluk olması neredeyse garantidir ve genellikle bu musluklardan bazıları havuzu belirli sürelerde doldururken, bazılarıda belirli sürelerde boşaltır. Hatta zamanla bu muslukların ne kadar hızla su doldurduğu veya boşalttığıda işin içerisine girer ve aslında sadece yüzmek için kullanabileceğimiz güzelim havuz koca bir problem haline dönüşür. Ben açıkçası bu problemleri çözmekte hep yetersiz kalmışımdır. Hatta çoğunlukla atmasyon cevaplar ürettiğimi itiraf edebilirim. Nitekim havuz deyince aklıma genellikle yandaki resimde görülen manzara gelir. Peki havuz problemlerinden kurtulabildik mi? Ihhh çünkü artık **Multi-thread** uygulamalar ile uğraşmaktayız ve elimizde yönetebileceğimiz n sayıda **Thread** olabiliyor. Bu **Thread'** ler bir havuz içerisinde toplanabilir mi peki? Evet toplanır...İşte bu günkü konumuz. **ThreadPool** kullanımı.

ThreadPool; arka planda belli bir işi yapmak üzere planlanmış görevlerin **Thread'** lere bölünmesi ve bu **Thread'** lerin bir koleksiyon şeklinde tutularak asenkron işleyişlerinin yönetilmesi amacıyla kullanılan **sarmalayıcı(Wrapper)** bir tip olarak düşünülebilir. Genellikle sunucu tabanlı uygulamalarda değerlendirildiği gözlemlenmektedir. örneğin **Windows Service'** leri içerisinde **ThreadPool** kullanımı mantıklıdır. Bunun dışında **dosya giriş çıkış(IO), yapay zeka, veritabanı, Karmaşık Matematik problemlerin çözüm algoritmaları** gibi **Multi-Threading** gerektiren işlemlerde değerlendirilebilir.

Burada önemli olan noktalardan birisi çalışma modelidir. Aslında yapılmak istenen işe ait havuza gelen her talep, havuz içerisinde bir **Thread'** e atanır ve asenkron olarak yürütülür. Burada ana uygulama **Thread'** ine bir bağımlılık söz konusu değildir*(ki ana uygulamanın ThreadPool tarafından değerlendirilen işlemlerin tamamlanmasını beklemesi yönünde uyarılması gerekebilir).* Hatta alt taleplerin bekletilmeside söz konusu değildir. Bununla birlikte önemli olan noktalardan biriside, işi biten bir görevin sahibi olan **Thread'** in tekrardan kullanılıncaya dek havuzda yer alan kuyruğa atılmasıdır. Burada kuyrukta yer alan **Thread'** in, ana uygulama tarafından tekrardan kullanılması halinde, **Thread** oluşturma maliyetlerinin önüne geçilmesi mümkün hale gelmektedir ki bu bir avantajdır. Tabiki havuzunda belirli bir kapasitesi vardır*(Varsayılan olarak 25 Thread)*.

Bu kapasitenin dolu olması halinde ek olarak gelen görevler kuyrukta kalır ve ancak işleyen**Thread'** ler çalıştırılmaya müsait olduklarında icra edilebilir.

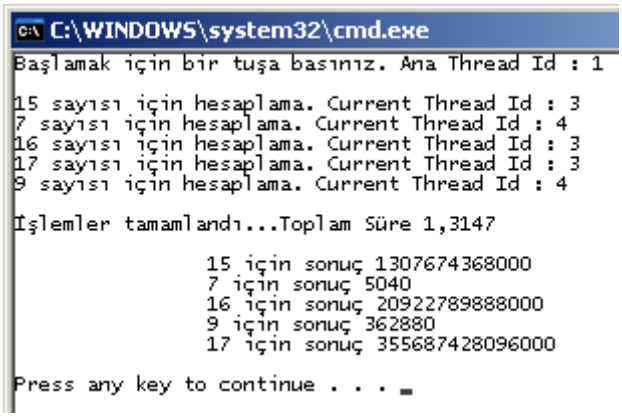
Buraya kadar anlattıklarımızı değerlendirecek olursak, **Thread** yönetiminin daha kolay bir şekilde ele alınabildiğini görebiliriz. Hatta **ThreadPool'** un temel olarak iki fonksiyonu olduğunu da düşünebiliriz. Bunlardan birincisi havuzda yer alan **Thread'** lerin üstlendiği işlerin tamamlanma durumlarını takip etmek, koordinasyonu sağlamak ve ikinci olarakta **Thread** koleksiyonunu bir kuyruk düzeninde yönetmektir. Tabi bu durumda **ThreadPool** tipinin yönettiği koleksiyona **thread ekleme(enqueue)** ve **çıkarma(dequeue)** işlemleri ile ilişkili bir algoritma içeridiğini ifade edebiliriz. Hatta kaç **Thread'** e ihtiyaç duyulacağını hesaplamak gibi önemli yeteneklere sahip olduğunu da söylemeliyiz. Açıkçası bu iki fonksiyonelliğin içerdiği algoritmaları geliştirmekle uğraşmak yerine işi **ThreadPool'** a bırakmak daha optimal bir çözüm olarak görülmelidir. Yine de istendiğinde kendi **ThreadPool** tiplerimizi de geliştirebiliriz. Tabi bilindiği üzere **.Net Framework 4.0** ile birlikte **ThreadPool** üzerinde de bazı geliştirmeler yapılmıştır. Bu geliştirmelere göre, havuzun çalışma mantığı değişmiştir. Ancak şu anda bu konuya girmeyeceğiz. önce var olan modeli bir öğrenelim. *(.Net 4.0 tarafındaki ThreadPool kabiliyetlerini biraz daha cesaret toplayıp ileride incelemeyi ve sizlere aktarmayı planlıyorum)* Dilerseniz bu kadar laf kalabalığından sonra basit bir örnek ile devam edelim. **Visual Studio 2008** ortamında ve **.Net Framework 3.5** tabanlı olaraktan bir **Console** uygulaması geliştireceğiz. İşte örnek kodlarımız.

```
using System;
using System.Diagnostics;
using System.Threading;

namespace WhatIsThreadPool
{
   class Program
   {
      static ManualResetEvent[] mrEvents;
      static int[] testNumbers; // Faktöryel değerleri hesap edilecek sayıların tutulacağı dizi.
      static long[] results; // Faktöryel sonuçlarının tutulacağı dizi
      static int testCount = 5; // Denemesayısı

      static void Main(string[] args)
      {
         Console.WriteLine("Başlamak için bir tuşa basınız. Ana Thread Id :
{0}",Thread.CurrentThread.ManagedThreadId.ToString());
         Console.ReadLine();

         // Ana Thread' i pool içinde çalışan Thread' lerin bittiği konusunda bilgilendirecek
ManualResetEvent nesne dizisi oluşturulur
         mrEvents = new ManualResetEvent[testCount];
         results = new long[testCount];
```

```
            testNumbers = new int[testCount];
            Random rnd = new Random();

            Stopwatch watcher = new Stopwatch();
            watcher.Start();

            for (int i = 0; i < testCount; i++)
            {
               // Başlangıçta ManualResetEvent nesnesi false değer ile üretilir.
               mrEvents[i] = new ManualResetEvent(false);
               testNumbers[i] = rnd.Next(1, 20); // örnek bir test sayısı üretimi
               ThreadPool.QueueUserWorkItem(new WaitCallback(ThreadWork), i);
            }

            // Tüm Thread' lerin işi bitinceye kadar ana uygulamayı duraksat
            WaitHandle.WaitAll(mrEvents);

            watcher.Stop();
            Console.WriteLine("\nİşlemler tamamlandı...Toplam Süre {0}
\n",watcher.Elapsed.TotalMilliseconds.ToString());

            for (int i=0;i<results.Length;i++)
            {
               Console.WriteLine("\t\t{0} için sonuç {1} ",testNumbers[i].ToString(),
results[i].ToString());
            }
            Console.ReadLine();
        }

        // ThreadPool içerisindeki Thread' lerin işaret ettiği metod. WaitCallback temsilcisinin
bildirimine uygun olaraktan object tipinden parametre almakta ve değer döndürmemektedir
        static void ThreadWork(object obj)
        {
            int currentNumber = (int)obj;

            Console.WriteLine("{0} sayısı için hesaplama. Current Thread Id :
{1}",testNumbers[currentNumber].ToString(),Thread.CurrentThread.ManagedThreadI
d.ToString());

            // Faktöryel hesaplamasını gerçekleştiren metod çağrısı
            results[currentNumber]=Factorial(testNumbers[currentNumber]);
            // Ana Thread' in bilgilendirilmesi sağlanır.
            mrEvents[currentNumber].Set();
        }
```

```
        // Faktöryel hesabını yapan recursive metod
        static long Factorial(int number)
        {
            long result;
            if (number == 0
                || number == 1)
                result = 1;
            else
                result = Factorial(number - 1) * number;

             return result;
        }
    }
}
```

Uygulamamızda 1 ile 20 arasındaki rastgele 5 sayının **Faktöryel** değerlerinin hesaplanmasında **ThreadPool'** dan yararlanılmıştır. örneği çalıştırdığımızda her seferinde farklı sonuçlar elde etmemiz söz konusudur. İşte benim yakaladığım sonuçlardan birisi.



Dikkat edileceği üzere **ThreadPool** tarafında iki **Thread** üretilmiş ve 5 sayısal değer için gerçekleştirilen faktöryel hesaplamaları bu **Thread'** ler tarafından ele alınmıştır.

Kişisel Not : *Tabi işin kolayına kaçtığımızı ifade etmek isterim. özellikle **21** sayısı dahil sonraki faktöryel hesaplarında eksi değerlere geçtiğimizden sayı aralığımız çok sınırlı. Bir diğer yanıltıcı nokta ise toplam hesaplama süresi. Buradaki faktöryel işlemleri aslında pek yorucu işlemler değildir. Bu nedenle aynı örneği **Thread**mekanizması olmadan çalıştırdığınızda hesaplama süresinin çok çok daha kısa sürdüğünü görebilirsiniz. Elbette bizim odaklandığımız nokta bu değil. Aslında **ThreadWork** metodunun çalıştırdığı **Factorial** fonksiyonunun gerçekten uzun süren yoğun işlemler gerçekleştirdiği düşünülebilir. Bu durumda **ThreadPool**mekanizması bize zaman yönünden avantaj getirecektir. Hatta işlemlerin uzunluğuna göre Thread sayısını arttırmasıda mümkün olabilir. Buna göre çıkartmamız gerken bir derste gerçekten ihtiyaç olunduğunda **ThreadPool** modelinin kullanılmasının uygun olduğudur.*

Tabi dikkat edilmesi gereken bir kaç nokta olduğunu söyleyebiliriz. öncelikli olarak Workflow Foundation mimarisinde de sık sık karşımıza çıkan **ManualResetEvent** tipinden bahsedelim. Bu tipten yararlanarak bir **Thread'** den başka bir **Thread'** e işin bittiğine dair sinyal gönderilmesi mümkündür. Bu noktada **ThreadPool** içerisinde çalışmakta olan **Thread'** lerin işlemlerini bitirmesini takiben, **ThreadPool'** un sahibi olan ana **Thread'** in*(ki burada Main metodunun yer aldığı Program tipine ait Thread' den bahsediyoruz)* işlemlerin tamamlandığı yönünde uyarılması gerekmektedir. Bu sebepten **ManualResetEvent** nesne örnekleri, **Set** metodu yardımıyla diğer **Thread'** i işlemlerin bittiği yönünde uyarmaktadır.

Ana uygulama için önem arz eden noktalardan biriside, **ThreadPool** içerisinde çalıştırılmakta olan **Thread'** lerin tamamının görevleri sonuçlanıncaya kadar beklemesi gerekebileceğidir. Bu sebepten **ManualResetEvent** tipinden olan dizinin tüm elemanlarının **Set** metodunun çalıştırılıp ana uygulama **Thread'** ini uyarması gerekmektedir. örnek kod parçasından da görüleceği üzere söz konusu duraksatma işlemi için **WaitHandle** tipine ait **static WaitAll** metodunun çağırılması yeterlidir.

Thread' ler ile ilişkili görevlerin **ThreadPool** tarafından yönetilen kuyruğa atılmaları için **QueueUserWorkItem** metodundan yararlanılmaktadır. Bu metodun ilk parametresi dikkat edileceği üzere **WaitCallback temsilcisi(delegate)** tipindendir. Bu temsilci **object** tipinden parametre alan ve **geriye değer döndürmeyen(void)** metodları işaret edebilir. Buna göre **Thread'** lerin eşleştiği görevleri üstlenen fonksiyonların söz konusu metod modeline uygun olması gerekmektedir. İlgili metod içerisinde dikkat edileceği üzere **ManualResetEvent** nesne örneği üzerinden **Set** metodu çağrısı gerçekleştirilmektedir. Tabi **C# 3.0** tarafında gelen **lambda operatörü(=>)** sayesinde aynı kodun aşağıdaki şekilde yazılmasıda mümkündür.

```
for (int i = 0; i < testCount; i++)
        {
            // Başlangıçta ManualResetEvent nesnesi false değer ile üretilir.
            mrEvents[i] = new ManualResetEvent(false);
            testNumbers[i] = rnd.Next(1, 20); // örnek bir test sayısı üretimi
            // ThreadPool.QueueUserWorkItem(new WaitCallback(ThreadWork), i);

            ThreadPool.QueueUserWorkItem((obj) =>
            {
                int currentNumber = (int)obj;

                Console.WriteLine("{0} sayısı için hesaplama. Current Thread Id : {1}",
testNumbers[currentNumber].ToString(),
Thread.CurrentThread.ManagedThreadId.ToString());

                // Faktöryel hesaplamasını gerçekleştiren metod çağrısı
                results[currentNumber] = Factorial(testNumbers[currentNumber]);
                // Ana Thread' in bilgilendirilmesi sağlanır.
```

```
            mrEvents[currentNumber].Set();
        }
        , i);
    }
```

Böylece geldik bir yazımızın daha sonuna. Umarım **ThreadPool** konusunda biraz fikir sahibi olabilmişizdir. Tekraradan görüşünceye dek hepinize mutlu günler dilerim.

WhatIsThreadPool.rar (23,73 kb)

C# 4.0 - Invariance, Covariance, Contravariance ??? (2009-12-22T16:15:00)

c# 4.0,.net framework 4.0,

 Merhaba Arkadaşlar,

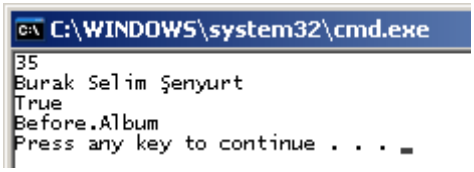
Bundan yıllar önce*(aslında 2005 yılında...çok eski bir tarih gibi görünmese de yazılım dünyası için çok çok uzun zaman önce anlamına gelmekte.)* daha genç bir makale yazarıyken **C# 2.0 delegate** tiplerinde co-variance, contra-variance durumlarını incelemeye çalışmıştım. Kişisel görüşüme göre, anlaşılmasından ziyade iyi bir şekilde analiz edilerek anlatılması çok zor olan bir konu **Co-Variance, Contra-Variance**. üstelik bu kavramların çıkış noktasında yer alan **Variant, Invariant tip** kavramları düşünüldüğünde konuyu anlamak için epey bir çaba sarf etmemiz gerekebiliyor. Hatta anlayamadığımız durumlarda neredeyse bulunduğumuz duruma isyan eder bir hale gelebiliyoruz. Bu tip zor konularda benim öğrenmek üzerine uyguladığım strateji aslında pek çoğumuzun da uyguladığı bir yöntem. önce sorunu örnekler ile anlamaya çalışmak, getirilen çözümü görmek ve en son olarak tanımlamaları yapmak. Bu önce kavram tanımlaması, sonra örnek uygulamanın yapılmasından ziyade daha etkili bir öğrenme şeklidir diye düşünüyorum. öyleyse vakit kaybetmeden analizimize başlayalım.

.Net' in başından beri...

Aslında **.Net'** in ilk duyurulduğu ve **C#, Vb.Net** gibi nesne yönelimli **yönetimli dillerin(Managed Languages)** dünyaya geldiği anlardan bu yana kalıtımsal ilişkide olan

tipler arasında bazı referans geçişlerinin yapıldığını bilmekteyiz. Burada tiplerin polimorfik özellikte olabilmelerinin de payı büyük. Bu sebepten .Net Framework' ün tüm sürümlerinde aşağıdaki gibi bir kod parçası olası.

```
using System;

namespace Before
{
   class Program
   {
      static void Main(string[] args)
      {
         WriteString(35);
         WriteString("Burak Selim Şenyurt");
         WriteString(true);
         WriteString(new Album { AlbumID = 1, Title = "Chikenfoot" });
      }

      static void WriteString(object obj)
      {
         Console.WriteLine(obj.ToString());
      }
   }
   class Album
   {
      public int AlbumID { get; set; }
      public string Title { get; set; }
   }
}
```



Bu kod parçasında yer alan **WriteString** metodu tüm **.Net** tiplerinin atası olan **object** tipinden parametre almaktadır. Bu ata tip-alt tip ilişkisi nedeniyle metoda herhangibir **.Net** tipinin atanması söz konusudur. üstelik herkes bir **Object** tipi olduğundan **ToString** metodunu uygulamakta veya uygulamasa bile **object** tipinin varsayılan **ToString** metodu çalıştırılabilmektedir. Nitekim **Album** tipi içerisinde **ToString** metodu ezilmemiş olmasına rağmen **Object** sınıfındaki varsayılan **ToString** metodunun çalıştırılması sağlanmıştır. *(Bildiğiniz üzere ToString metodu object tipi içerisinde **virtual** olarak tanımlanmıştır ve alt tiplerde ezildiği takdirde objcet nesne referansı üzerinden ezilen versiyonunun çalıştırılması söz konusudur.)*

Yolumuza devam edelim ve bu sefer aşağıdaki kod parçasını göz önüne alalım.

```
static void Main(string[] args)
{
  object albm = CreateAlbum(2, "Is There a Love in space[Joe Satriani]");
}
static Album CreateAlbum(int albumId,string title)
{
  return new Album{AlbumID=albumId,Title=title};
}
```

Bu kod parçasında yer alan **CreateAlbum** metodu **Album** tipinden bir değer döndürmektedir. **Main** metodu içerisinde ise **CreateAlbum** çağrısı sonucunun **object** tipine atanması söz konusudur. Her iki kod parçasınında çalışma zamanında veya derleme zamanında bir hata üretmesini beklemeyiz. Şimdi koltuklarınıza yaslanın ve takip eden paragrafı okuyun...

İlk kod parçasının çalışması doğaldır nitekim .Net' in tüm sürümlerinde parametreler Covariant tiptedir. Diğer yandan ikinci kod parçasının da çalışması doğaldır çünkü dönüş tipleride Contravariant' tır.

Bir şey anladınız mı? Açıkçası ben halen daha durumu tam olarak netleştiremediğimizi düşünüyorum. öyleyse...

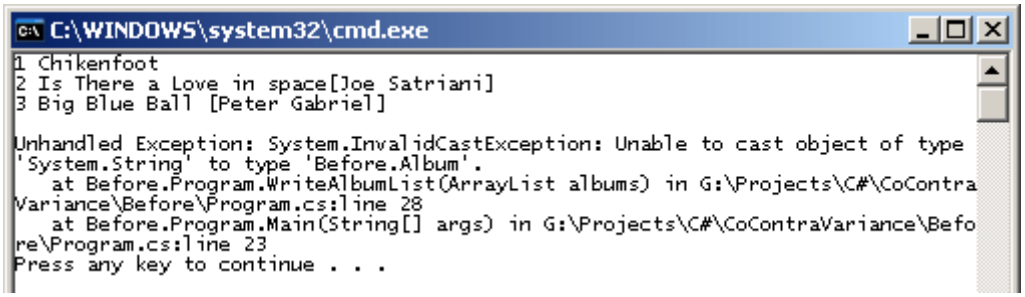
Generic mimariden önceki koleksiyonlarda durum...

İşin içerisine **object** tipinden değerler ile çalışan generic mimari öncesi koleksiyonlar girdiğinde, Covariance veya Contravariance olmanın artık güvenli olup olmadığından söz edilmeye başlanmaktadır. Güvenlik tip bazındadır. Şimdi bu durumu anlamaya çalışarak devam edelim.

```
ArrayList albumList = new ArrayList();
albumList.Add(new Album { AlbumID = 1, Title = "Chikenfoot" });
albumList.Add(new Album { AlbumID = 2, Title = "Is There a Love in space[Joe Satriani]" });
albumList.Add(new Album { AlbumID = 3, Title = "Big Blue Ball [Peter Gabriel]" });
```

ArrayList gibi koleksiyonların **object** tipi ile çalışmalarının sebebi herhangibir tip için koleksiyon bazlı özelliklerin kullanılabilmesini sağlamaktır *(Tabi bu, tip güvensiz-unsafe bir durumu oluşturmuş ve sonrasında generic mimari getirilmiştir).* Diğer yandan **object** ile çalışmaları nedeniyle, herhangibir **.Net** tipini bünyesinde barındırabilirler. Buna göre ilk örneğimizi ve bu son kod parçasını göz önüne alırsak koleksiyonların Covariance özelliğini sağladığını*(yani Covariant tipte olduklarını)* düşünebiliriz. Ancak buda tam olarak doğru değildir. Şimdi aşağıdaki kod parçasını göz önüne alalım.

```
using System;
using System.Collections;

namespace Before
{
   class Program
   {
      static void Main(string[] args)
      {
         ArrayList albumList = new ArrayList();
         albumList.Add(new Album { AlbumID = 1, Title = "Chikenfoot" });
         albumList.Add(new Album { AlbumID = 2, Title = "Is There a Love in space[Joe
Satriani]" });
         albumList.Add(new Album { AlbumID = 3, Title = "Big Blue Ball [Peter Gabriel]"
});
         albumList.Add("Reality Killed The Video Star [Robbie Willams]");

         WriteAlbumList(albumList);
      }

      static void WriteAlbumList(ArrayList albums)
      {
         foreach (Album albm in albums)
         {
            Console.WriteLine(albm.ToString());
         }
      }
   }
   class Album
   {
      public int AlbumID { get; set; }
      public string Title { get; set; }

      public override string ToString()
      {
         return String.Format("{0} {1}", AlbumID.ToString(), Title);
      }
   }
}
```



çok doğal olarak **foreach** döngüsü içerisinde **albums** koleksiyonu üzerinde dolaşılırken sadece **Album** tipleri ele alınmak istenmiştir. Ancak birisi kazayla **albumList** isimli koleksiyona **string**tipte bir değişken göndermiş ve yukarıdaki çalışma zamanı hatasının alınmasına neden olmuştur. Aman tanrımmmm!!! Şimdi koleksiyonlar için bahsedilen **Covariant**tipte oldukları gerçeği **Un-Safe Covariant** olarak düzeltilmelidir. Nitekim tip güvenliğinin garanti altına alınması mümkün olmamıştır. Hatta pek çok kaynak object tipi ile çalışan koleksiyonların aslında tamamen **Invariant** olduklarını ifade etmektedir. Kafamız gittikçe karışıyor değilmi. öyleyse...

Peki ya diziler(Arrays)...

Aşağıdaki ekran görüntüsünde yer alan kod parçasını göz önüne alalım.

WriteAll metodu **object** tipinden bir dizi ile çalışmaktadır. Buna göre **string** tipinden bir dizinin bu metoda parametre olarak aktarılması mümkündür. Burada **Covariance** olma durumu söz konusudur. Elbette güvensiz olan versiyonu. çünkü **WriteAll** metodu içerisinde bir tip güvenliği yoktur. Diğer yandan ilginç olan durum **int** tipinden bir diziyi göndermek istediğimizde ortaya çıkmaktadır. Böyle bir durumda derleme zamanı hatası alınacaktır. **Int** tipi değer tipi olduğundan **object** gibi bir referans türüne atanması mümkün değildir. İşte bu noktada **object**tipinden dizinin **Invariance** özellik gösterdiğini söyleyebiliriz.

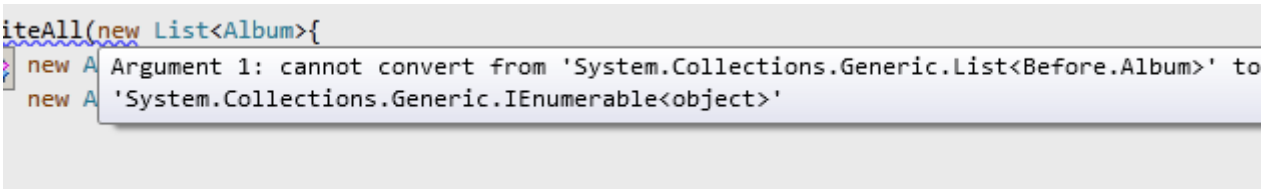
Buna göre; Diziler bazı durumlarda **Invariant** bazı durumlarda ise **Covariant** tip olarak görülebilirler. Ancak **Covariant** olsalar dahi tip güvensiz olacaklardır(Unsafe). Diğer yandan **değer türlü(Value Type)** diziler her zaman için **Invariantce** özellik gösterir.

Generic koleksiyonlar...

Gelelim **tip güvenli(Type Safe)** olan generic koleksiyonlara. **Generic** mimari **.Net** içerisinde bir devrim yaratmış ve major versiyonlamaya gidilmesine neden olmuştur. Generic mimari sayesinde örneğin koleksiyonların sadece söylenen tiple çalışabileceği daha kodlama zamanındayken belirtilebilmekte ve böylece çalışma zamanında tip güvenliğinin aşılması engellenmektedir. Peki ya aşağıdaki kod parçasını göz önüne aldığımızda;

```
using System;
using System.Collections;
using System.Collections.Generic;

namespace Before
{
   class Program
   {
      static void Main(string[] args)
      {
         WriteAll(new List<Album>{
            new Album { AlbumID = 1, Title = "Chikenfoot" },
            new Album { AlbumID = 2, Title = "Is There a Love in space[Joe Satriani]"
}
         }
         );

      }
      static void WriteAll(IEnumerable<object> parameters)
      {
         foreach (object parameter in parameters)
         {
            Console.WriteLine(parameter.ToString());
         }
      }
   }
   class Album
   {
      public int AlbumID { get; set; }
      public string Title { get; set; }

      public override string ToString()
      {
         return String.Format("{0} {1}", AlbumID.ToString(), Title);
      }
   }
}
```

Burada **WriteAll** metodu **IEnumerable<object>** tipinden bir parametre almaktadır. Buna göre tüm **T** generic tiplerinin **object** tipinden türeyeceği düşünüldüğünde herhangibir sorun olmayacağı sonucuna varılabilir *(En azından kafamızdaki compiler bu kodu sorunsuz olarak derleyecektir. İlk etapta...*) Yani **Covariance** olma durumu söz konusudur diyebiliriz...Mi acaba? İşte derleme zamanının bize vereceği cevap...

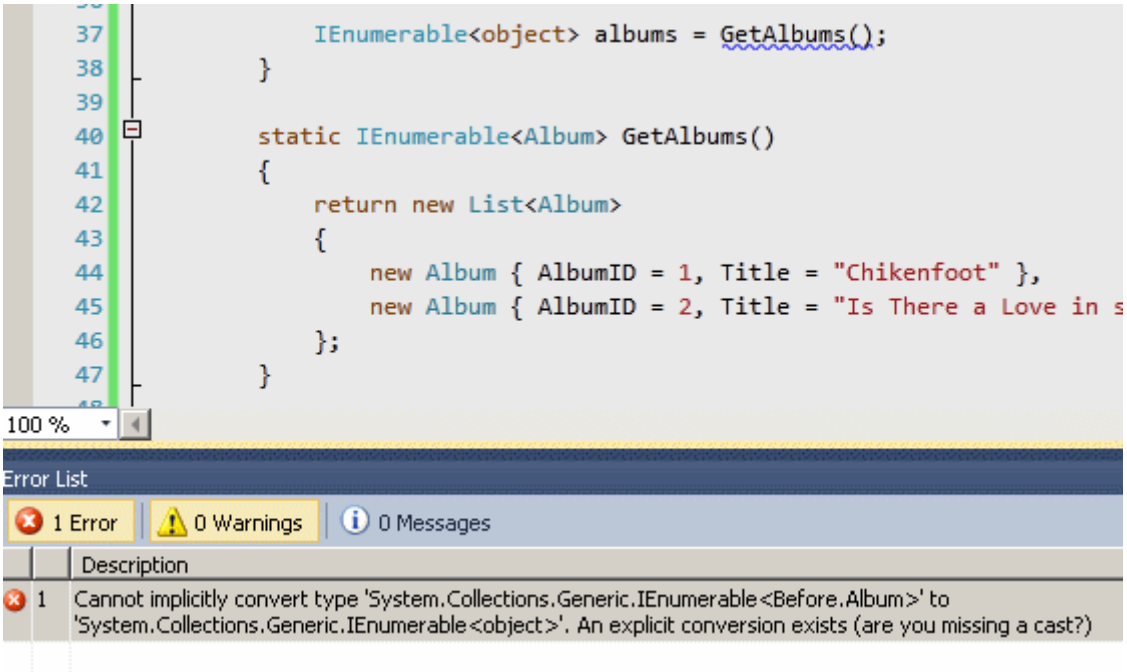


Görüldüğü üzere **Album** tipi ile çalışan koleksiyonun **object** tipine dönüştürülemeyeceğini belirten bir hata mesajı ile karşı karşıyayız. Buna göre **generic**koleksiyonların aslında **Invariant** tipte olduklarını ifade edebiliyoruz. Bu nedenle generic koleksiyonlar ne **Covariance** nede **Contravariance** özellik göstermektedir.
Aslında **generic** koleksiyonlarda özel bir durum olarak **T** tipleri için türetmenin söz konusu olmadığını söyleyebiliriz. Bu sebepten **IEnumerable<Album>** ün **IEnumberable<object>** tarafından taşınması söz konusuolmamaktadır. Birde aşağıdaki kod parçasını göz önüne alalım;

```
using System;
using System.Collections;
using System.Collections.Generic;

namespace Before
{
   class Program
   {
      static void Main(string[] args)
      {
         IEnumerable<object> albums = GetAlbums();
      }

      static IEnumerable<Album> GetAlbums()
      {
         return new List<Album>
         {
            new Album { AlbumID = 1, Title = "Chikenfoot" },
            new Album { AlbumID = 2, Title = "Is There a Love in space[Joe Satriani]" }
         };
      }
   }
   class Album
   {
```

```
        public int AlbumID { get; set; }
        public string Title { get; set; }

        public override string ToString()
        {
            return String.Format("{0} {1}", AlbumID.ToString(), Title);
        }
    }

}
```

GetAlbums metodu **IEnumerable<Album>** tipinden bir referans döndürmektedir. **Album, Object**' in alt tipidir. Buna göre **GetAlbums** metodunun sonucunun **IEnumerable<object>** tipinden bir arayüze atanabiliyor olması düşünülebilir. Oysaki derleme zamanında aşağıdaki hata mesajı alınacaktır.



Görüldüğü üzere bir dönüştürme hatası alınmaktadır.

Şimdi buraya kadar anlattıklarımızı gözden geçirecek olursak belki şu cümleleri sarf edebiliriz.

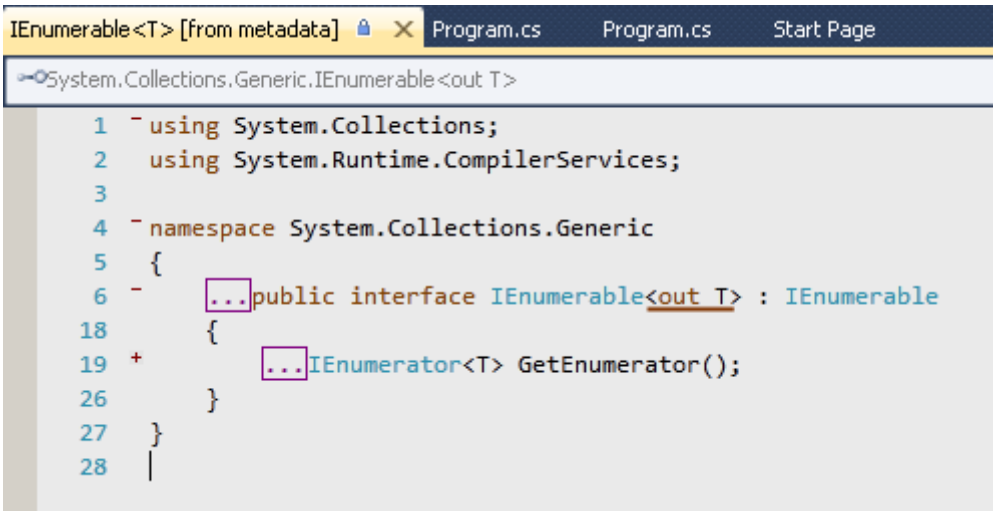
- **Invariant** tiplerin kullanıldığı yerlerde, belirtilen tipin birebir aynısının ele alınması gerekmektedir*(.Net 4.0 öncesi generic koleksiyonlar)*
- **Covaraint** tiplerin kullanılabildiği yerlerde, alt tipten olan değişkenlerin parametre olarak aktarılması mümkündür.
- **Covariant**' lık için güvensiz olma(Unsafe) durumlarıda söz konusudur.*(Object tipli koleksiyonlar ve diziler)*

- **Contravariant'** lığa göre ise dönüş tipinin metoddan dönen tipin üst tipiden bir nesne örneğine atanması mümkündür.
- **Covariant** tipler için geçerli olan güvensiz tip sendromu doğal olarak **Contravariant** tipler içinde söz konusudur.

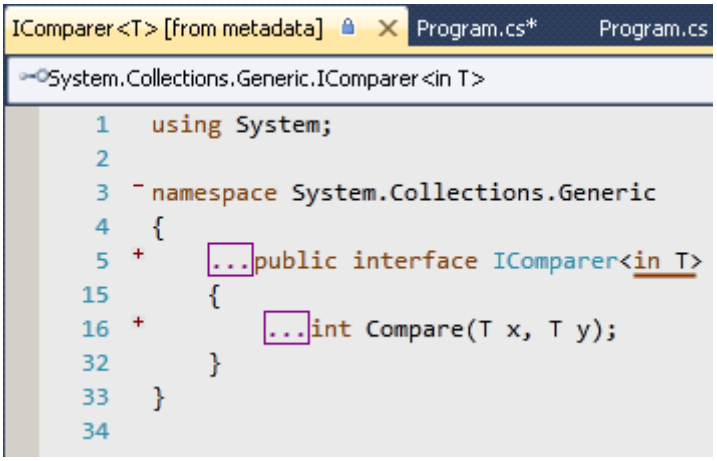
Peki .Net 4.0 sonrasında...

.Net 4.0 versiyonunda **Generic** koleksiyonların katı olan tip kavramı bozularak **Covariant** ve **Contravariant** olmalarına izin verildiğini söyleyebiliriz. Ancak burada önemli bir fark olduğundan bahsetmemiz gerekiyor. Generic koleksiyonların Covariant ve Contravariant olarak çalışabilmeleri sağlanırken bunun güvenli olaraktan(Type Safe) gerçekleştirilebilmesi sağlanmış. Bu noktada örneğin object tipinden olan koleksiyonların covariance ve contravariance davranışlarından ayrıldığını ifade edebiliriz. Peki bu yeni kabiliyetler nasıl aktarılmış?

İşte **IEnumerable<T>** arayüzünün **.Net 4.0'** da tanımlanış şekli.



out T tanımlaması mutlaka dikkatiniz çekmiştir. Bu durum **output safe** olarak adlandırılmaktadır. Şimdi başka bir tipi göz önüne alalım.



Bu sefer **in T** kullanımı dikkati çekmektedir. Bu durum ise **Input Safe** olarak adlandırılmaktadır. Her iki durumda alt tarafta öyle ele alınmaktadırki generic koleksiyonlarda Covariant ve Contravariant' lığın tip güvenli olarak ele alınması mümkün olmaktadır. Sonuç olarak kafamız karışsada biraz **.Net 4.0** ile birlikte generic koleksiyonların tip güvenli olaraktan**Covariant** ve **Contravariant** özellik gösterebilmelerinin mümkün hale geldiğini söyleyebiliriz. Bununla birlikte **out T** ve **in T** kullanımlarının şu an için sadece generic koleksiyonlar ve**temsilcilerde(delegate)** söz konusu olduğunu belirtelim. Dolayısıyla generic temsilcilerinde tip güvenli Covariant veya Contravariant olarak ele alınmaları mümkündür. Buna göre **out T** ve **in T**için şu ifadeleri kullanabiliriz.

- **out T Covariant** tip kullanımını sağlamaktadır. Buna göre örneğin **IEnumerable<object>** tipte parametre alan bir metoda, **IEnumerable<Product>** gibi bir referansın atanabilmesi mümkündür.
- **in T Contravariant** tip kullanımı sağlamaktadır. Buna göre örneğin **IEnumberable<string>** dönen bir metodun sonucunun **IEnumerable<Object>** tipine atanması mümkündür.

Dolayısıyla aşağıdaki örnek kod parçası sorunsuz olarak derlenecek ve çalışacaktır.

```
using System.Collections.Generic;

namespace NowInNet4
{
   class Program
   {
      static void Main(string[] args)
      {
         List<Product> products = new List<Product>
         {
            new Product{ProductId=1,Name="Americano Coffee",ListPrice=10},
            new Product{ProductId=2,Name="English Royal Tea",ListPrice=12}
         };

         Process(products);

         IEnumerable<object> allProducts = GetProducts();
      }

      static IEnumerable<Product> GetProducts()
      {
         return new List<Product>
         {
            new Product{ProductId=1,Name="Americano Coffee",ListPrice=10},
            new Product{ProductId=2,Name="English Royal Tea",ListPrice=12}
```

```
        };
    }

    static void Process(IEnumerable<object> parameters)
    {
        // Bir takım işlemler
    }
}
class Product
{
    public int ProductId { get; set; }
    public string Name { get; set; }
    public decimal ListPrice { get; set; }
}
```

Ben en azından biraz olsun anlamış durumdayım. Umarım sizlerede en iyi şekilde aktarabilmişimdir. Tekrardan görüşünceye dekhepinize mutlu günler dilerim.

CoContraVariance.rar (38,87 kb)

Webiner - WCF RIA Services (2009-12-20T00:10:00)

wcf,wcf ria services,silverlight,wcf eco system,

Merhaba Arkadaşlar,

Bildiğiniz üzere geçtiğimiz hafta içerisinde kişisel bilgisayarımdaki teknik bir aksaklık nedeniyle **NedirTv?com** sitesi aracılığıyla yayınladığımız **WCF RIA Services** webinerimiz yarım kalmıştı. **18.12.2009 Cuma** gecesi 21:00 - 22:00 saatleri arasında ise herhangibir sorun olmadan Webinerimizi tamamlamayı başardık.
Ayrıca Webiner kaydı **NedirTv?com** sitesi üzerinden yayınlandı.

Bu Webinerimizde kısaca **WCF RIA Service** kavramını tanımaya çalıştık. Bildiğiniz üzere **.Net RIA Services** ismi **WCF RIA Services** olarak değiştirildi. Bu **WCF Eco System** yaklaşımının bir sonucu. **WCF RIA Service**' leri, **zengin içerikli internet uygulamalarının(Rich Internet Applications), çok katmanlı mimariyi(N-Tier)** uygulamaları halinde sunucu üzerindeki kaynaklara ulaşmalarında büyük önem taşımaktadır. özellikle **sunum katmanı(Presentation Layer)** ile **orta katman(Middler Tier)** arasındaki **uygulama mantığının(Application Logic)** paylaşılmasında önemli bir rol üstlenmektedirler. Bu anlamda sunucu kaynaklarının kullanımına ilişkin olarak, uygulama mantığının etkili ve kolay bir şekilde ele alınması mümkün hale gelmektedir. üstelik geliştiricilerin servis alt yapısı, iletişim şekli gibi bir takım detayları düşünmesine gerek yoktur. Nitekim, **Visual Studio** ortamına entegre gelen **Domain Service** şablonu yardımıyla sadece uygulama mantığı üzerine odaklanılması yeterlidir. Elbette veri kaynağına olan erişiminde kullanılabilen **Entity Data Model**' in yetenekleride yadsınamaz.

Tabi konu **RIA** uygulamaları olunca akla ilk gelen **Silverlight** nesneleridir. Bizde Webinerimizde **Silverlight 4.0** odaklı olarak geliştirilen bir uygulamada **WCF RIA Service** kullanımını incelemeye çalıştık. Keyifli seyirler dilerim...

Süre -> **54:52**

C# 4.0 ile Code Contracts (2009-12-18T14:40:00)

c# 4.0,.net framework 4.0,

Merhaba Arkadaşlar,

Microsoft gibi dev yazılım firmalarının araştırma geliştirme ekipleri ve labarotuvar çalışmaları her zaman ilgimi çekmiştir. Herhalde pek çok yazılımcının hayalleri arasında bu tip firmalarda çalışmak ve yeni fikirleri ortaya atarak diğer yazılımcılara sunmak yer almaktadır. **Microsoft**' un **DevLabs**isimli portalında bu tip fikirlerin labarotuvar çalışmalarının yer aldığını görebilirsiniz. örneğin son zamanların popüler konularından

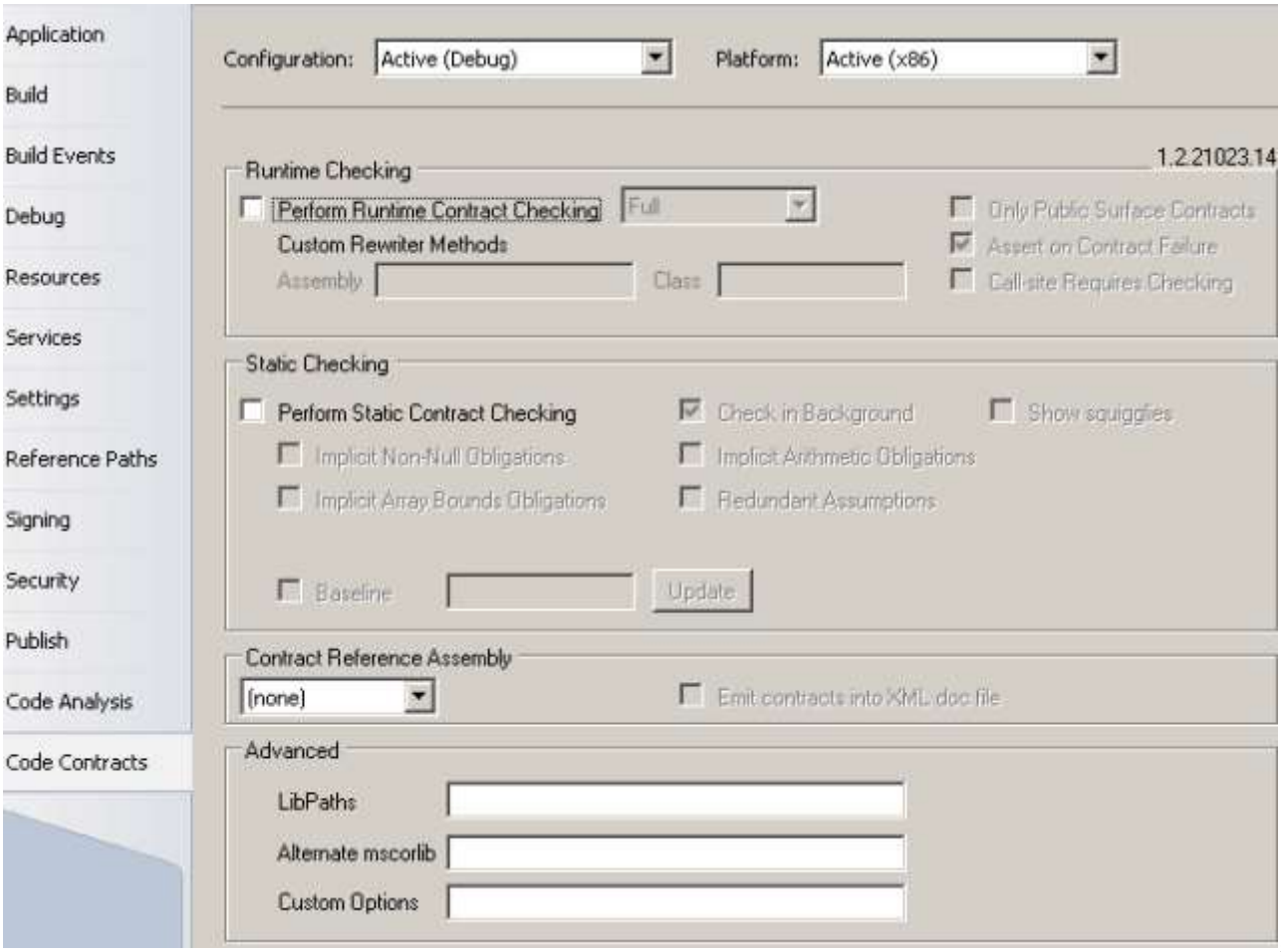
birisi olan**observable** koleksiyonları kullanarak **asenkron(Asynchronous)** ve **olay güdümlü(Event Based)** programlamayı kolaylaştıran **Reactive Extensions,** yeni başlayanlara yazılım anlatan **Small Basic** yada **White Box** testleri için geliştirilen **Pex**... Tabi daha pek çok labarotuvar çalışması yer almaktadır. Bunlardan biriside **Code Contracts'** tır.

Uzun süredir ilgimi çeken ama fırsat bulamadığım konulardan birisidir **Code Contracts**. özellikle test süreçlerinde önem arz eden ve kodun çalışma zamanında veya kodlama zamanında varsayımsal bazı koşulları sağlayıp sağlamadığını tespit etmemizi sağlayan bir yenilik olarak düşünülebilir. Bu noktada kodun **ön koşullu(Pre-Conditions)** veya **son koşullu(Post-Conditions)** olarak test edilmesinin mümkün olduğunu söyleyebiliriz. Bu iki yeteneğe ek olarak bir nesnenin **durumunun(State)** beklendiği gibi olmasının kontrolüde **Object Invariant** özelliği sayesinde gerçekleştirilebilmektedir.

Normal şartlarda **Microsoft.Diagnostics.Contracts isim alanı(Namespace)** altında yer alan tipler ile kod sözleşmelerinin kullanılması mümkündür. Bu isim alanı **.Net Framework 4.0 Base Class Library** içerisinde doğrudan gelmektedir. **.Net Framework 4.0** öncesi sürümlerde kullanmak istediğimizdeyse**Microsoft.Contracts.dll assembly'** ının projeye referans edilmesi*(örneğin XP işletim sisteminde **C:\Program Files\Microsoft\Contracts\PublicAssemblies\v3.5** adresinde yer almaktadır)* gerekmektedir. Kolay kullanım için yazıyı hazırladığım tarih itibariyle **DevLabs**' ten gerekli **aracın(Tool)** indirilerek kurulmasında yarar vardır. **Code Contracts, Microsoft** [**DevLabs**](http://msdn.microsoft.com/en-us/devlabs/default.aspx)içerisine dahil edilmiş önemli projelerden birisidir. **Code Contracts'** ın bloğun yazıldığı tarih itibariyle iki farklı versiyonu bulunmaktadır. **Standart** versiyon **çalışma zamanı kontrollerini(Runtime Checking)** yapabilmekteyken, **Team System** için üretilen versiyon **static kontroller(Static Checking)** yapabilmektedir. İlgili sürümler yine **Visual Studio 2010 Beta 2**sürümü ile de çalışmaktadır.

***Not :** Eğer elinizde benim gibi **Visual Studio 2010 Ultimate Beta 2** sürümü var ise, **Team System Edition** sürümünü kurabilirsiniz.*

Kurulum işlemi sonrasında **Visual Studio Ultimate 2010 Beta 2** üzerinde oluşturulan projelerin **özelliklerine(Properties)** aşağıdaki ekran görüntüsünde yer alan bir ara birimin dahil edildiğini görürüz.



Burada yer alan detaylı özellikleri zaman içerisinde öğreniyor olacağız. İlk etapta **Static Checking** kısmının sadece **Team System** destekli **Visual Studio**' lar üzerinde*(Visual Studio 2008 içinde geçerlidir)* etkinleştirilebildiğini belirtelim. Aslında bu kabiliyetlerin özellikle çalışma zamanında yapılan **if...try...catch** gibi kontrollerden ne gibi farkı olduğunu kavramamız oldukça önemlidir. örneğin dökümantasyon avantajları bunlardan birisidir. Söz gelimi metod parametrelerinin **gereklilikleri(Requirements)**, olası **istisnalar(Exceptions)** ve gerekli **izinler(Permissions)** için otomatik dökümantasyon üretimi sağlanabilmektedir. Diğer yandan **Unit Test** tarafına getirdiği avantajlardan da bahsedilebilir. Daha anlamılı **Unit Test'** lerin üretilebilmesi, özellikle her**sözleşmenin(Contracts)** bir analist gibi davranması ve çalıştırılan **Test** için **pass/fail** işaretlemesini sağlayabilmesi vb...Tabiki konuyu daha iyi kavrayabilmek adına bol bol örnek geliştirmekte yarar vardır. İşe çok basit bir **Hello World** uygulaması ile başlamakta yarar olacağı kanısındayım. Bu nedenle aşağıda yer alan **Console** uygulaması kodlarını yazdığımızı varsayalım.

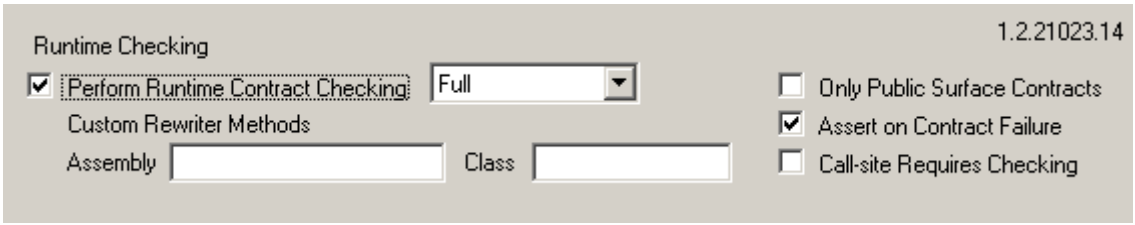
```
using System.Diagnostics.Contracts;

namespace CodeContracts
{
   class Program
   {
```

```
        static void Main(string[] args)
        {
            ChinookContext context = new ChinookContext();
            context.CreateAlbum(null, 1);
            Album result=context.CreateAlbum("The Best", 1);
            Contract.Ensures(result.Name.Length > 10, "Album nesnesinin
oluşturulmasında albüm adının 10 karakterden fazla olması beklenir");
        }
    }

    class ChinookContext
    {
        public Album CreateAlbum(string albumName, int albumId)
        {
            Contract.Requires(!string.IsNullOrEmpty(albumName),"Album nesnesinin
oluşturulması için Album adının null veya boş olmaması gerekir");
            Album albm=new Album {
                AlbumId=albumId
                ,Name=albumName };
            return albm;
        }
    }

    class Album
    {
        public int AlbumId { get; set; }
        public string Name { get; set; }
    }
}
```

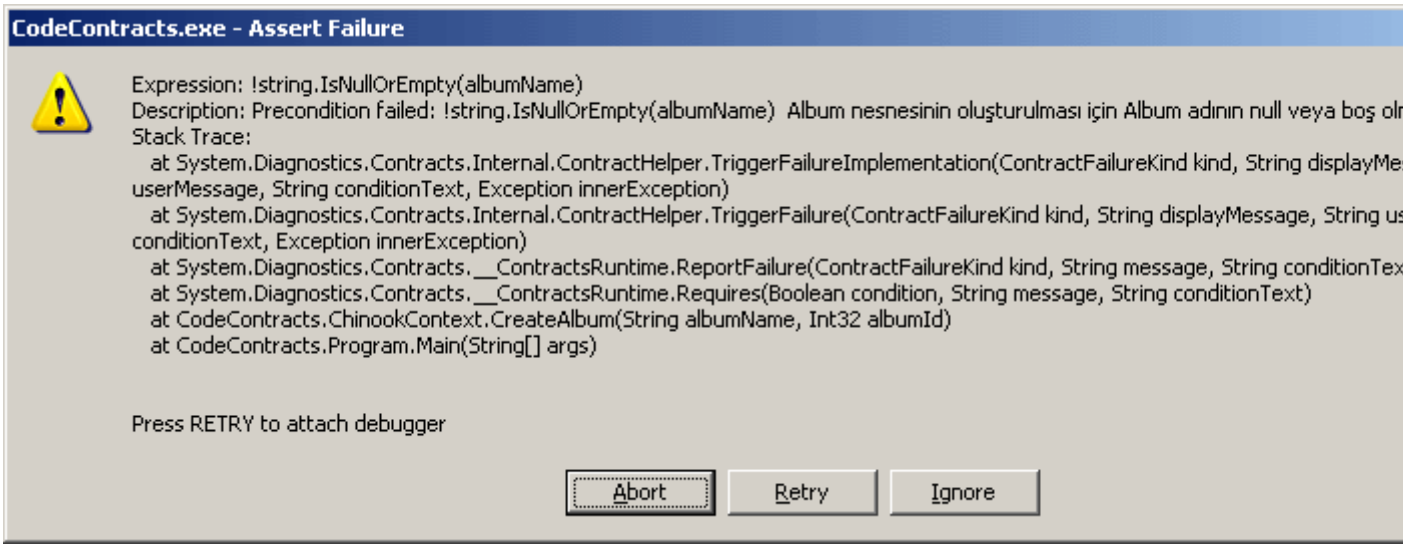
Bu örnek kod parçasında **Album** isimli sınıfa ait iki nesne örneği üretiminin gerçekleştirildiğini görmekteyiz. **CreateAlbum** isimli metod içerisinde **Contract.Requires** isimli bir **static** fonksiyon çağrısı olduğu hemen dikkatinizi çekmiş olmalıdır. Bu metod ile bir **ön koşul(Pre-Condition)** belirtilmektedir. Bu koşula göre **CreateAlbum** metoduna gelen **albumName** parametresinin değerinin **null** veya **boş <u>olmaması</u>** beklenmektedir. Diğer yandan **Main** metodunun son satırında da **Contract.Ensures** isimli bir metod çağrısı yer almaktadır. Bu çağrı ilede bir **son koşul(Post-Condition)** tanımlaması yapılmaktadır. Bu koşula göre **CreateAlbum** metodu ile üretilen ikinci **Album** nesne örneğinin **Name** özelliğinin karakter sayısının **10'** un üzerinde olması istenmektedir.

örneği bu haliyle çalıştırdığımızda hiç bir sorun olmadığını görürüz. Hımmm...Enteresan bir durum. Oysaki **Requires** veya **Ensures** metod çağrılarından en az birisine takılmamız gerekirdi. Aslında sorun henüz kod sözleşmelerinin **çalışma zamanı(runtime)** veya **static** olarak izlenmesi gerektiğini belirtmemiş olmamız. Bunun

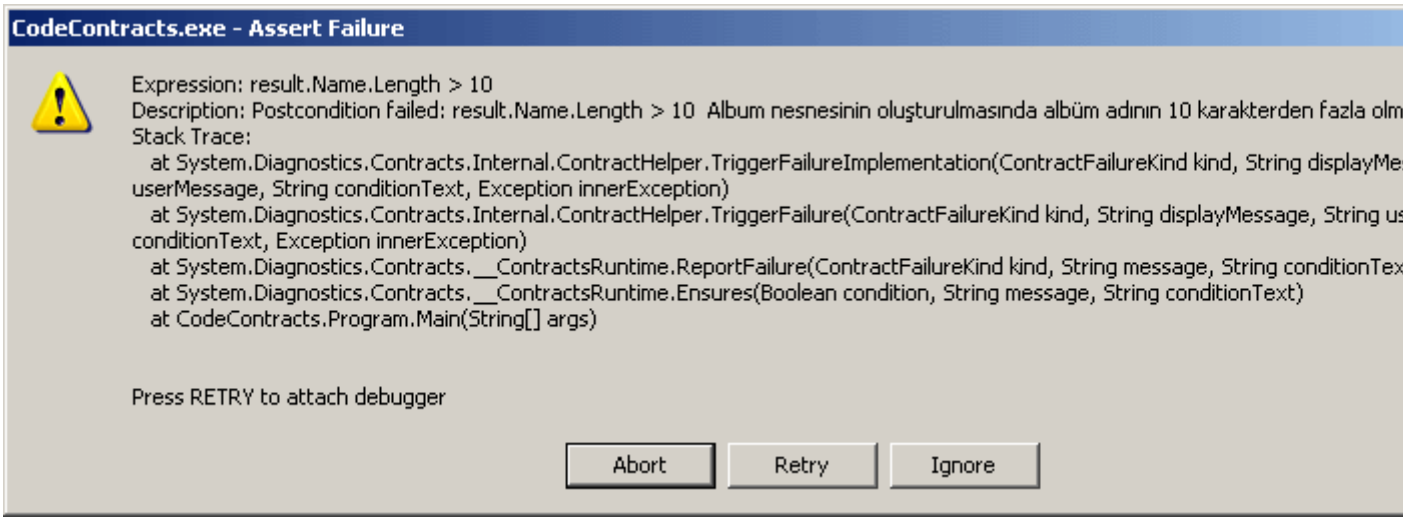
için **Tool** ile birlikte projeye eklenen **Contracts** sekmesindeki **Runtime Checking** kutusunu işaretlememiz yeterlidir.



Şimdi örneğimizi çalıştırdÄ±ğÄ±mÄ±zda ilk olarak aşağÄ±daki mesaj kutusu ile karşÄ±laştÄ±ğÄ±mÄ±zÄ± görürüz.



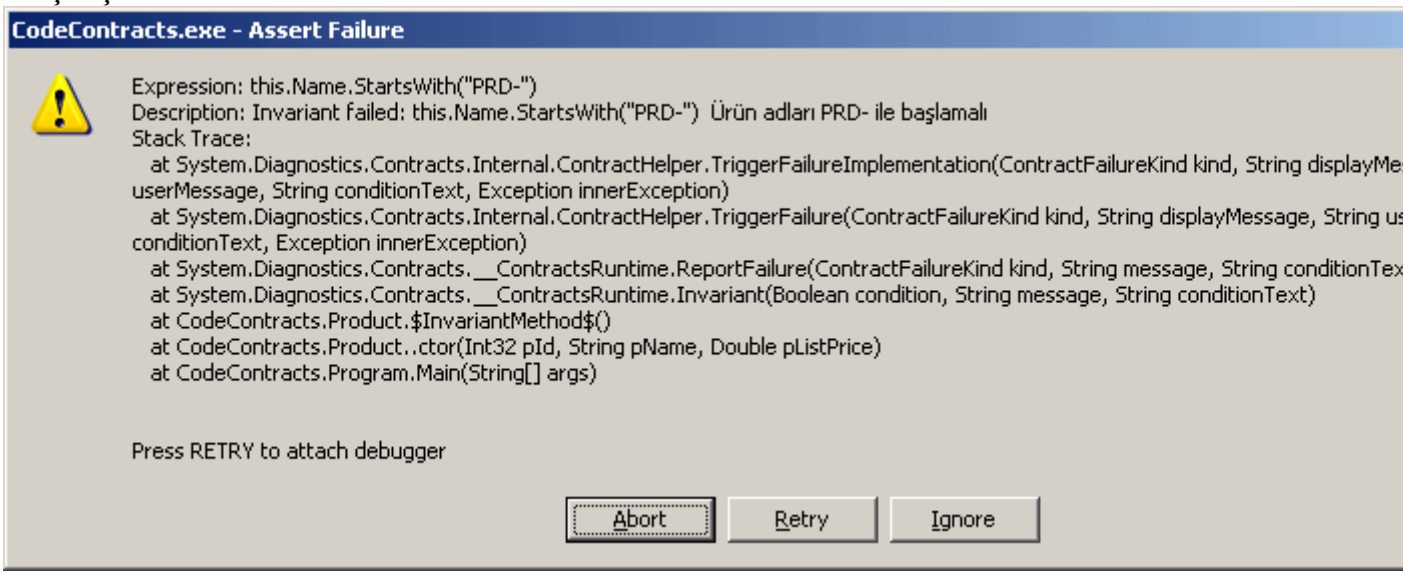
Dikkat edileceği üzere **Album** adÄ±nÄ±n **null** geçilmesi nedeniyle bir uyarÄ± mesajÄ± üretilmiştir. Bir başka deyişle ön koşulun sağlanamadÄ±ğÄ± açÄ±k bir şekilde görülmektedir. Bu noktadan sonra hatayÄ± görmezden gelip ilerleme şansÄ±mÄ±z vardÄ±r. **Ignore** düğmesine basarak devam ettiğimiz takdirde bu kez **Post-Condition'** a takÄ±ldÄ±ğÄ±mÄ±zÄ± görebiliriz.



üretilen uyarÄ± mesajÄ±na göre ikinci **Album** nesnesinin adÄ±nÄ±n karaketer uzunluğunun istenildiği gibi olmadÄ±ğÄ± gözlemlenmektedir. Şimdi dilerseniz **Object Invariant** konusuna dair basit bir örnek geliştirmeye çalÄ±şarak devam edelim. **Invariant** özelliğini bir nesnenin istemci tarafÄ±ndan ele alÄ±ndÄ±ğÄ± yerdeki **durumunun(State)** iyi olmasÄ± ile alakalÄ± bir yetenek şeklinde düşünebiliriz. Bir nesnenin durumunu içerdiği alanlar ifade ettiğinden, bu alanlarÄ±n beklenen şekilde olmasÄ± **iyi bir nesne(good object)** olduğu anlamÄ±na da gelecektir. İşte örnek kod parçasÄ±.

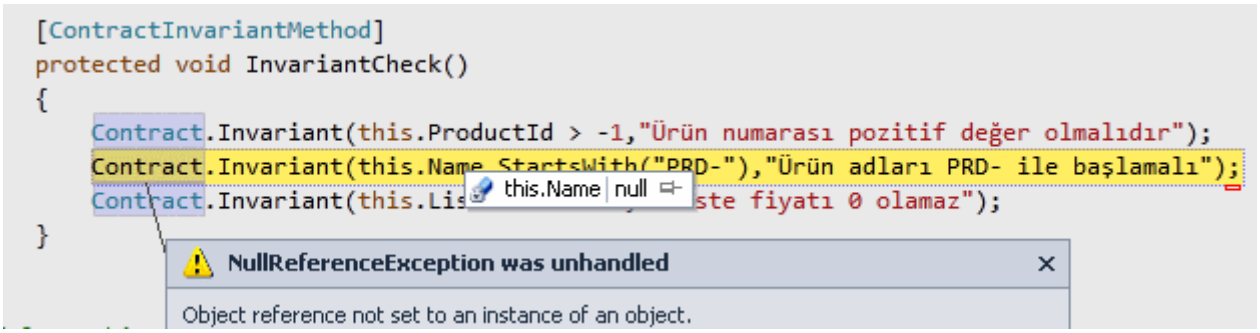
```
using System.Diagnostics.Contracts;

namespace CodeContracts
{
   class Program
   {
      static void Main(string[] args)
      {
         Product bardak = new Product(1000,"Bardak",3.45);
      }
   }

   class Product
   {
      private int ProductId;
      private string Name;
      private double ListPrice;

      public Product(int pId,string pName,double pListPrice)
      {
         ProductId = pId;
         Name = pName;
         ListPrice = pListPrice;
      }

      [ContractInvariantMethod]
      protected void InvariantCheck()
      {
         Contract.Invariant(this.ProductId > -1,"ürün numarası pozitif değer
olmalıdır");
         Contract.Invariant(this.Name.StartsWith("PRD-"),"ürün adları PRD- ile
başlamalı");
         Contract.Invariant(this.ListPrice != 0, "Liste fiyatı 0 olamaz");
      }
   }
}
```

Product isimli sınıfta **ContractInvariantMethod niteliği(attribute)** ile imzalanmış bir metod bulunmaktadır. Geriye değer döndürmeyen ve **protected** olarak işaretlenmiş bu metod içerisinde dikkat edileceği üzere **Contract** sınıfının **static Invariant** metoduna yapılan bazı çağrılar bulunmaktadır. Bu çağrılar içerisinde, üretilen **Product** nesnesinin durumunu simgeleyen alanların değerleri kontrol edilmektedir. örnek çalıştırıldığında aşağıdaki mesaj ile
karşılaşılacaktır.



Bu son derece doğaldır nitekim ürün adı **Invariant** çağrısında olduğu gibi **PRD-** ön eki ile başlamamaktadır. Dolayısıyla nesne örneği istenilen duruma sahip değildir.

***Kişisel Not: Invariant** kontrolünde dikkat edilmesi gereken iki durum vardır. **Auto Property**' ler kullanıldığında çalışma zamanında bir istisna mesajı alınacaktır. Bu istisna**Contract.Invariant** çağrılarının yapıldığı yerde gerçekleşecektir. Bunun sebebi **Invariant** çağrılarının **null** değer içeren özellikleri kontrol etmeye çalışmasıdır. Söz konusu durum varsayılan yapıcı metodlar kullanıldığında da nüksetmektedir. Söz gelimi yukarıdaki kod parçasında **Product** tipi için varsayılan yapıcı metod yazılıp buna göre bir **Product** nesnesi örneklendiğinde aşağıdaki **çalışma zamanı istisna(Runtime Exception)** mesajı ile karşılaşılır.*



***Name** alanı **string** tipten olduğu için nesnenin ilk örneklenmesi sırasında **varsayılan yapıcı metod(Default Constructor)** tarafından **null** değer ile beslenir. Bunun sonucu olarakta**Invariant** metodu **null** değer üzerinde kontrol yapmaya çalışmaktadır.*

Elbette **Code Contracts** konusu burada anlatıldığı kadar yalın ve sade değildir. Aksine dökümantasyonuna bakıldığında çok fazla kuralı olduğu görülmektedir. özellikle**Static Checking** özelliği derleme işlemi sırasında bazı kod sözleşmelerinin kontrolü sağlamaktadır. Konuyu araştırdıkça ve öğrendikçe sizlere daha fazlasını aktarmaya çalışıyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

CodeContracts.rar (25,40 kb)

FORParallelism (2009-12-16T13:55:00)

task parallel library,

Merhaba Arkadaşlar,

Günümüz yazılım teknolojilerinin belkide en popüler olan konularından biriside **paralel programlamadır(Parallel Programming)**. özellikle kullanıcı bilgisayarlarının artık birden fazla çekirdeğe sahip işlemcilerle donatılmış olduğu düşünüldüğünde geliştirme ortamlarının da(.Net Framework 4.0' da olduğu üzere) paralel programlamaya daha fazla destek vermeye başladığını görmekteyiz. Aslında zaten var olan araçlar ile paralel programlama tekniklerini uygulayabilmekteyiz. Ne varki kodlanmasının karmaşık olması bir yana, birden fazla tekniğin kullanılabiliyor olması, hangisinin daha performanslı olduğunun anlaşılması için test aşamalarının da önemini ortaya çıkarmakta. **Microsoft**cephesi bir süredir, paralel programlama kütüphanesi ile söz konusu tekniklere ait tasarımları aza indirgeyip kolay geliştirilebilir ve performanslı sonuçlar üreten tiplerin tasarlanması ve geliştirilmesini gereçekleştirmekte. **.Net Framework 4.0** içerisinde doğrudan gelen **Task Parallel Library**kütüphanesi bu anlamda önemli kabiliyetler içermekte.

Peki elimizde bu kütüphane olmasaydı? O zaman n sayıda tekrar edecek olan bir işlemi paralel hale getirmek için nasıl bir kodlama yapmamız gerekirdi?

Söz gelimi başlangıç ve bitiş değerleri parametrik olan bir döngünün içerisinden çağırılan bir fonksiyonun, birden fazla **Thread'** e bölünerek çalıştırılmasını istediğimizi düşünelim. Aslında teorik olarak makinede kaç işlemci yada kaç çekirdek var ise o sayıda Thread

açılması tercih edilir. Buna göre tekrar edecek olan işlemler belirli aralıklara bölünerek bu aralıkların açılan Thread' ler tarafından ele alınması sağlanır. Ne demek istediğimi aşağıdaki örnek kod parçası ile aktarmaya çalışayım.

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;

namespace Parallelism
{
   class Program
   {
      static void Main(string[] args)
      {
         ParallelFor(48, 98,
            (i) =>
            {
               double r = 0;
               for (int j = 0; j < i * 99999; j++)
                  r = Math.Sqrt((i * Math.PI) / Math.E);
               Console.Write("{0} ", i.ToString());
            }
         );

         Console.ReadLine();
      }

      static void ParallelFor(int lowerBound, int upperBound, Action<int> body)
      {
         int processorCount = Environment.ProcessorCount; // İşlemci/çekirdek sayısı bulundu
         int range = (upperBound - lowerBound) / processorCount; // Yaklaşık iterasyon sayısı hesaplanır.
         Console.WriteLine("İşlemci/çekirdek Sayısı : {0} , Iterasyon Boyutu {1}\n", processorCount.ToString(), range.ToString());

         #region Birinci Yöntem (List<Thread> Kullanımı)

         List<Thread> threads = new List<Thread>(processorCount); // İşlemci/çekirdek sayısı kadar Thread taşıyacak koleksiyon tanımlanır.

         // İşlemci/çekirdek sayısı kadar çalışacak bir döngü
         for (int processor = 0; processor < processorCount; processor++)
         {
```

```
            // Thread tarafından ele alınacak değer aralığı hesaplanır
            int startPoint = (processor * range) + lowerBound;
            int endPoint = (processor == processor - 1) ? upperBound : startPoint + range;
            Console.WriteLine("Start : {0} End : {1}", startPoint.ToString(),
endPoint.ToString());
            // Her bir çekirdek için bir Thread oluşturulur ve içerisinde iterasyon aralığı
uzunluğunda bir döngü oluşturularak parametre olarak gelen fonksiyon çalıştırılır
            threads.Add(new Thread(() =>
               {
                  for (int i = startPoint; i < endPoint; i++)
                  {
                     body(i);
                  }
               }
            )
            );
         }

         // Thread' ler çalıştırılır
         foreach (Thread t in threads)
         {
            t.Start();
         }
         // Thread' lerin tamamlanması beklenir
         foreach (Thread t in threads)
         {
            t.Join();
         }
         #endregion
      }
   }
}
```

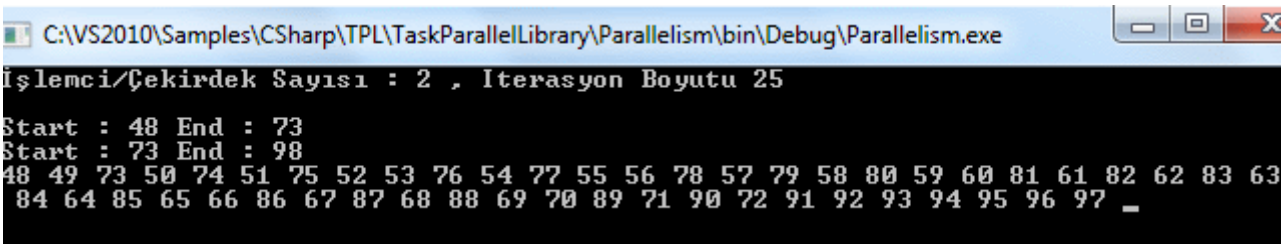
örneğimizde tamamen anlamsız olan bir döngü çalıştırıldığını görmektesiniz. İçerdiği bazı hesaplamalar sayesinde zaman alan bir işlemler bütünü söz konusu. Burada söz konusu olan operasyonun birden fazla Thread' e bölünerek çalıştırılması içinde **ParallelFor** isimli yardımcı metoddan yararlanılmaktadır. Bu modelde **ParallelFor** isimli metod döngünün başlangıç ve bitiş değerlerini almakta, ayrıca çalıştıracağı fonksiyonu işaret eden **Action<T>** tipinden bir parametre almaktadır. Metoda göre **Thread'** lerin değerlendireceği aralıklar hesaplanır. **Thread'** ler basit bir **List<T>** koleksiyonunda tutulmakta olup çalıştırılmaları ve tamamlanmalarının beklenmeleri için iki **foreach** döngüsünden yararlanılır. İlk döngü oluşturulan **Thread'** leri başlatırken diğeride oluşturulan **Thread'** leri **Main Thread'** e katıp uygulamanın sonlanması için söz konusu **Thread'** lerin işlerinin bitirilmesinin beklenmesini garanti etmektedir. Dikkat edilmesi gereken nokta **işlemci/çekirdek sayısı** kadar **Thread** oluşturulması ve oluşturulan

her bir **Thread'** in yaklaşık olarak hesap edilen iterasyon alanı kadar değeri hesaba katarak Action<T> ile gelen operasyonu çalıştırmasıdır.

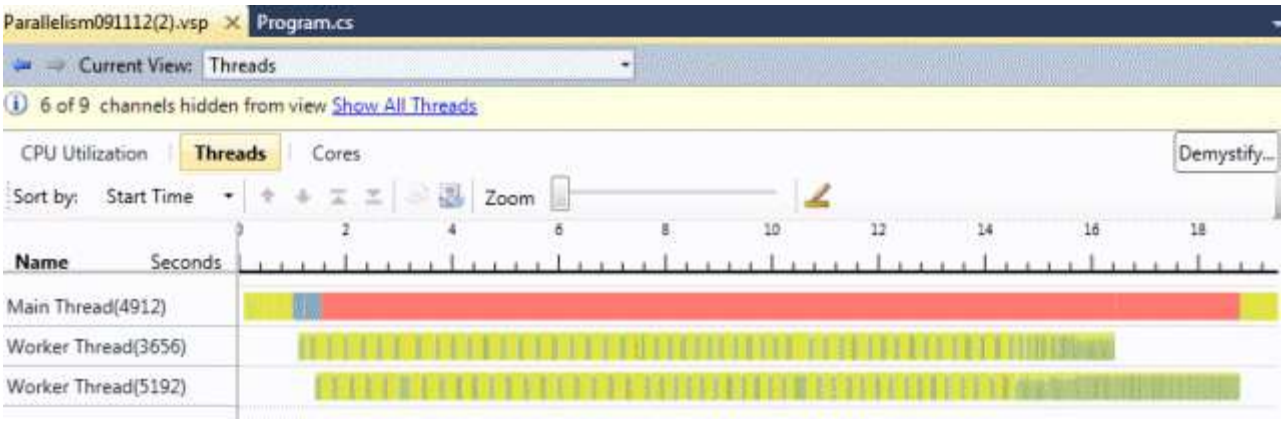
Peki çalışma zamanındaki durum nedir?

Bu konuda çok şanslıyız nitekim Visual Studio 2010 ile birlikte son derece etkili performans analiz araçları gelmekte. Geliştirmeyi yapmakta olduğumuz **Visual Studio 2010 Ultimate Beta 2**sürümünde yer alan **Concurrency Profiler** raporuna bakıldığında, yukarıdaki örnek için aşağıdaki sonuçların elde edildiği görülür.

İlk çalışmanın sonucu oluşan ekran görüntüsü;



İlk çalışma sonucu elde edilen Concurrency Profiler çıktısı;



Sarı alanlar bir **Thread'** in çalışmakta olduğunu ama diğer bir Thread tarafından o süre boyunca etkisizleştirildiğini göstermektedir. Yeşil renkli alanlar **Thread'** in işini yaptığı zaman aralıklarıdır. Sarı bölgelerin fazla olması performansı olumsuz yönde etkileyen bir faktördür. Nitekim **aşırı talebin(Oversubscription)** oluştuğunu göstermektedir. Peki iyileştirmenin bir yolu olabilir mi? Aslında **ThreadPool** tipinden yararlanarak **Therad** yönetiminin sisteme bırakılması sağlanabilir. İşte buna göre yazılan yeni modelimiz;

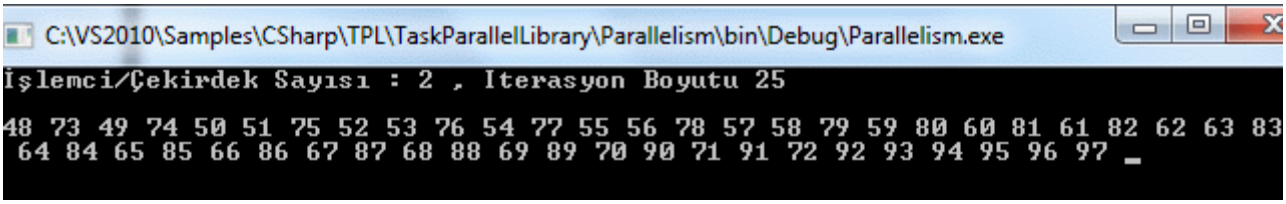
```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;
```

```
namespace Parallelism
{
   class Program
   {
      static void Main(string[] args)
      {
         ParallelFor(48, 98,
            (i) =>
            {
               double r = 0;
               for (int j = 0; j < i * 99999; j++)
                  r = Math.Sqrt((i * Math.PI) / Math.E);
               Console.Write("{0} ", i.ToString());
            }
         );

         Console.ReadLine();
      }

      static void ParallelFor(int lowerBound, int upperBound, Action<int> body)
      {
         int processorCount = Environment.ProcessorCount; // İşlemci/çekirdek sayısı
bulundu
         int range = (upperBound - lowerBound) / processorCount; // Yaklaşık iterasyon
sayısı hesaplanır.
         Console.WriteLine("İşlemci/çekirdek Sayısı : {0} , Iterasyon Boyutu {1}\n",
processorCount.ToString(), range.ToString());

         #region İkinci Yöntem (ThreadPool)

         int remainingProcessor = processorCount;
         // ManualResetEvent bir olay meydana geldiğinde beklemekte olan bir veya daha
çok Thread' e bilgilendirmede bulunur.
         ManualResetEvent manuelResetEvent = new ManualResetEvent(false);
         for (int processor = 0; processor < processorCount; processor++)
         {
            int startPoint = (processor * range) + lowerBound;
            int endPoint = (processor == processorCount - 1) ? upperBound : startPoint +
range;

            // ThreadPool, Thread' ler için bir havuz sağlar ve asenkron işleyişin yönetimini
sağlar
            // QueueUserWorkItem yürütülmek üzere bir metodu kuyruğa atar.ThreadPool
içerisindeki ilgili thread kullanılabilir olduğunda da metod icra edilir.
            ThreadPool.QueueUserWorkItem((o) =>
```

```
            {
                for (int i = startPoint; i < endPoint; i++)
                {
                    body(i);
                }
                // Birden fazla Thread tarafından kullanılan remainingProcessor değeri azaltılır ve 0 olup olmadığı kontrol edilir. Eğer 0 ise bekleyen tüm Thread' ler için sinyal verilir
                if (Interlocked.Decrement(ref remainingProcessor) == 0)
                    manuelResetEvent.Set();
            }
        );

    }
    // Diğer Thread' ler tamamlanıncaya kadar(ki Set dolayısıyla sinyal geldiğinde anlaşılır) ana thread' i bekletecektir.
    manuelResetEvent.WaitOne();
    manuelResetEvent.Close(); // Güncel WaitHandle ile alakalı tüm kaynaklar serbest bırakılır(Release).

    #endregion
  }
 }
}
```

Bu sefer **ThreadPool** tipinin **QueueUserWorkItem** static metodu kullanılmıştır. **ThreadPool** bir önceki modele göre **Thread** yönetimini daha iyi yapmaktadır. öyleyse yeni modele göre oluşan çalışma zamanı **Thread** analizine bir bakalım.

İkinci modele göre çalışma sonucu;



İkinci modele göre Concurrency Profiler çıktısı;

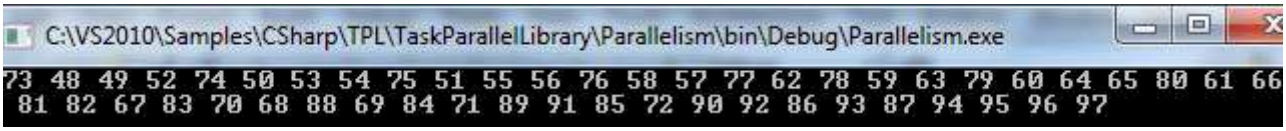
Bu rapora göre Thread' lerin toplam çalışma sürelerinin bir önceki modele göre azaldığı görülmektedir. Sarı alanların süresi daha az görünsede sayıları yinede çok azalmış değildir. Dolayısıyla **aşırı talep(Oversubscription)** durumu devam ediyor görünmektedir. Ancak ana Thread' in sadece Thread' lerin çalışması tamamlanıncaya kadar bloklandığı gözlemlenmektedir. Peki yeni bir yöntem tercih edilebilir mi? Evet edilebilir. Aslında **ThreadPool** kullanımının iyileştirilmesi yoluna gidilebilir ki biz daha fazla ilerlemeyeceğiz...

Gördüğünüz üzere çoğu geliştirici açısından ileri seviyede kalan bir kodlama gerekmektedir. özellikle geliştiricinin Thread konusuna son derece iyi hakim olması şarttır. Her ne kadar söz konusu karmaşık teknikler birer tasarım kalıbı olarak şekillenmiş olsalarda geliştiricinin kafa ayarını da fazla çizdirmemek gerekir. Buda yazımızın neden kafayı çizmiş bir bilgisayarcı resmi ile başladığının ispatıdır😀 İşte **Task Parallel Library** ile birlikte gelen tipler bu anlamda işleri kolaylaştırmaktadır. Ama tabiki **Concurrency Profiler** ile üretilen rapor sonuçlarını değerlendirmek gerekir.*(Bu tip karmaşık teknikleri tercih ederken kişisel görüşüme göre programcının performans mı? kolay ve hızlı kodlama mı? sorusuna verdiği cevap büyük önem kazanmaktadır)* İşte aynı süreç için **Parallel.For** kullanımı ve rapor sonuçları;

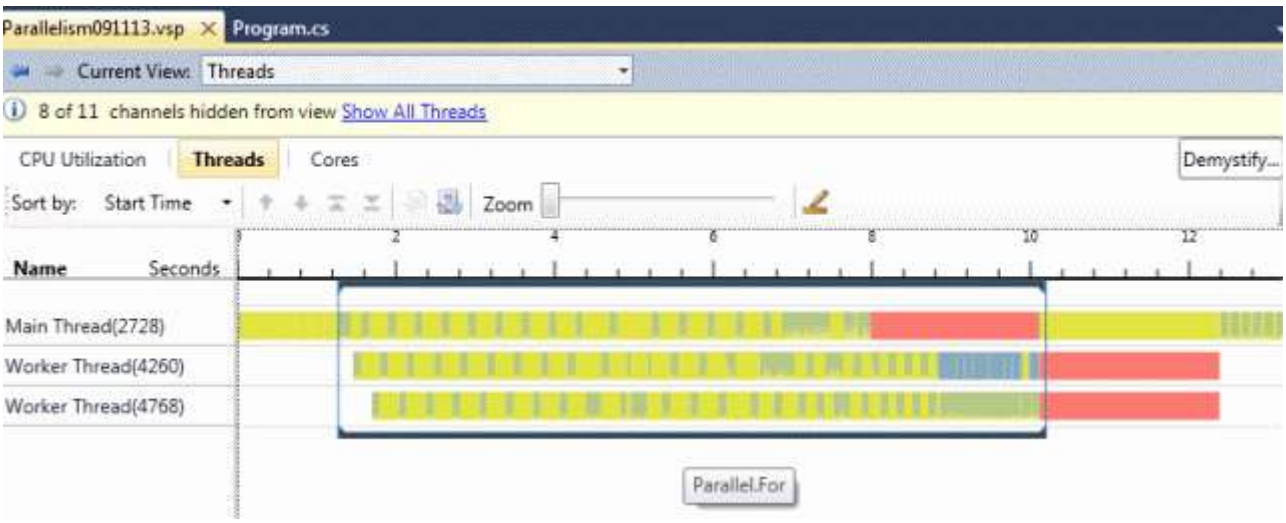
```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Threading.Tasks;

namespace Parallelism
{
   class Program
   {
      static void Main(string[] args)
      {
         Parallel.For(48, 98, (i) =>
            {
               double r = 0;
               for (int j = 0; j < i * 99999; j++)
                  r = Math.Sqrt((i * Math.PI) / Math.E);
```

```
                Console.Write("{0} ", i.ToString());
            }
        );

        Console.ReadLine();
    }
  }
}
```

Parallel.For kullanımının sonucu;



Parallel.For kullanımına göre Concurrency Profiler çıktısı;



Bu rapora göre Thread işlemlerinin tamamlanma süresinin çok daha azaldığı görülmektedir. Ayrıca Sarı alanların sayısında belirgin ölçüde azalma gözlemlenmektedir. İlginç olan noktalardan biriside **Main Thread'** de bloklanmanın(Kırmızı alanlar) diğer modellere göre çok daha az sayıda olmasıdır. Bir başka deyişle aşırı talep(Oversubscription) durumu biraz daha azalmıştır.

***Raporlar ile ilişkili not**: Döngülerin çalışma zamanında açtıkları **Thread'** lerin çalışma şeklini raporlamak amacıyla **Visual Studio 2010** ile birlikte gelen **Debug**menüsündeki **Start Peformance Analysis** öğesi kullanılmaktadır. Bu öğe ile açılan sihirbazda bizim örneğimiz için **Concurrency** seçeneği işaretlenmelidir. Ayrıca bu seçeneğin alt seçimi olan **Visualize the behavior of multithreaded application** kutucuğunun işaretlenmiş olması da gerekmektedir. Ancak bu son seçenek **Windows 7, Server 2008** işletim sistemleri üzerinde kullanılabilmektedir. Raporların oluşturulması programın çalıştırılması ile birlikte başlamaktadır. Bu nedenle söz konusu analiz raporlarının üretilmesi zaman alabilir. Raporlarda n sayıda **Thread'** in*

*görülmesi mümkündür. örneklerimizdeki analizlerimizi kolayca incelemek için sadece ilgili **Thread**' lerin çalışma zamanı durumları göz önüne alınmıştır. Diğerleri ise gizlenmiştir. üretilen analiz raporundaki **Threads** kısmında yer alan renklerin belirli anlamları vardır. Sarı renkler **Preemption**olarak adlandırılmakta olup genellikle aşırı talep(Oversubscription) ile ilişkili süreleri belirtmektedir. Yeşil alanlar Thread' in iş yaptığını gösteren zaman aralıkları iken kırmızı alanlar bloklama yapılan zaman aralıklarını ifade etmektedir.*

Tabiki bu test sonuçları, uygulamanın çalıştığı sistemin donanımsal özelliklerine göre değişiklik gösterecektir. Ancak sonuç olarak **Parallel.For** döngüsünün paralel işlemleri daha efektif olarak yürüttüğünü düşünebiliriz. Bunlara ek olarak aslında **Parallel.For** döngüsünün sağladığı başka avantajlarda vardır. Bunlar aşağıda listelenmiştir.

- **Etkili Yük Dengelemesi (Load Balancing) :** Parallel.For Thread' ler arasındaki yük dağılımını organize eder.
- **Dinamik Thread Sayısı :** Parallel.For akılldır ve zaman aşımları durumunda döngü içerisindeki Thread sayılarını dinamik olarak ayarlayabilir.
- **Yüksek Değer Aralıkları : Parallel.For** metodu **Int32** dışında **Int64** tipini de kullanılabilir.
- **Konfigurasyon Seçenekleri :** örneğin Parallel.For içerisinde açılacak olan Thread sayısı için limit belirlenebilir.
- **İstisna Yönetimi (Exception Handling) :** Döngü içerisinde bir istisna oluştuğunda, dahil olan tüm Thread' ler mümkün olduğunca kısa sürede işlemlerini durdururlar. Aslında varsayılan olarak yeni iterasyonların başlaması durdurulur. Bu nedenle exception sonrası Thread' lerin yürüttüğü bazı iterasyon adımları devam edebilir ancak yenilerinin başlatılması exception nedeniyle engellenir.
- **İç içe paralellik (Nested Parallelism) :**İç içe çalıştırılan Parallel.For döngüleri birbirlerinin Thread kaynaklarını koordineli olarak paylaşarak çalışırlar.
- vb...

Şimdi bu avantajları kendi yazdığımız **ParallelFor** metodu içinde gerçellemeye çalıştığımızı düşünelim. Hatta deneyin Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

TaskParallelLibrary.rar (89,38 kb)

C# 4.0 - COM Interop İyileştirmelerinden Dynamic Import ve Ommiting Ref [Beta 2] (2009-12-15T09:30:00)

c# 4.0,.net framework 4.0,


Merhaba Arkadaşlar,

Hani bazen insanın canı şöyle çıtır çıtır kuruyemiş çeker ya...Hatta çoğunlukla bir film seyrederken, maç izlerken, arkadaşları ile sohhet ederken, internette surf yaparken iyi gider ya...Hatta birisinin blog yazısını okurken kuruyemişleri yerken daha bir heyecanlı, istekli olur ya... İşte bende bu düşünceyle yola çıkıp siz değerli okurlarım kuruyemiş yerken kısa zamanda bir şeyler öğrenebilin, keyifli bir on dakika geçirin diye bu yazıyı hazırladım. Bakalım bu yazımızda bizleri hangi macera bekliyor.

Bildiğiniz üzere **C# 4.0** ile birlikte yine köklü dil değişiklikleri hayatımıza girmiş bulunmakta. özellikle dinamik diller ile olan etkileşimin arttırılması ve **COM** dünyası ile olan haberleşmede getirilen yenilikler son derece önemli. Bu gelen yenilikler arasında dynamic anahtar kelimesi, opsiyonal ve isimlendirilmiş parametrelerde(Optional & Named Parameters) en çok göze çarpanlar arasında yer almakta. Ancak çok fazla irdelenmeyen fakat özellikle **COMInterop** dünyasını ilgilendiren minik ve önemli iyileştirmelerde bulunmakta. Bu neden bu kısa yazımızda söz konusu minik iyileştirmelerden ikisini çok basit olarak incelemeye çalışıyor olacağız.

Kişisel Not : *Bu konuda en doğru bilgilere ve güncel örneklere yazının hazırlandığı tarih itibariyle MSDN sitesinden ulaşabileceğinizi de hatırlatmak isterim.*

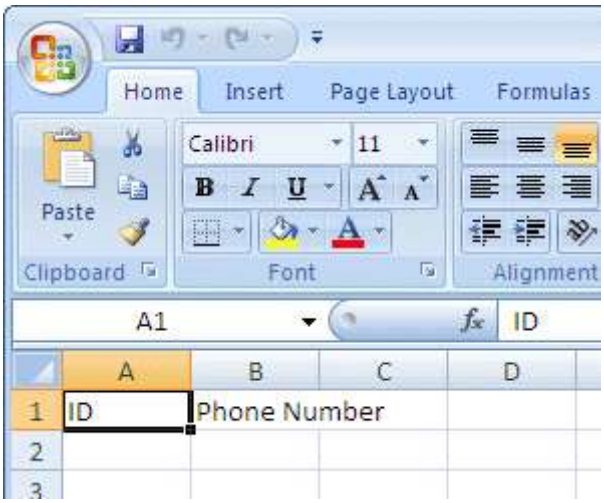
Dynamic Import;

Dynamic anahtar kelimesi yardımıyla **static** olmayan ve **COM**, dinamik diller gibi ortamlardan gelen nesnelerin ele alınabilmesi mümkün hale gelmektedir.
Peki **COM Interop** ile olan etkileşimde **Dynamic**tiplerin çaktırmadan geldiğini biliyor muydunuz? Dilerseniz konuyu irdelemek için aşağıdaki kod parçasını göz önüne alalım.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Excel = Microsoft.Office.Interop.Excel;
```

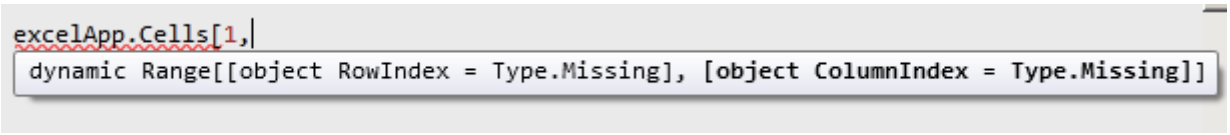
```
namespace COMInteropFeatures
{
   class Program
   {
      static void Main(string[] args)
      {
         var excelApp = new Excel.Application();
         excelApp.Workbooks.Add();
         excelApp.Visible = true;

         #region DynamicImport özelliği olmadan önce

         ((Excel.Range)excelApp.Cells[1, 1]).Value2 = "ID";

         Excel.Range range12 = (Excel.Range)excelApp.Cells[1, 2];
         range12.Value2 = "Phone Number";

         #endregion
      }
   }
}
```

Bu örnekte Excel API' sine ulaşılaraktan bir Workbook oluşturulması ve aşağıdaki görüntünün üretilmesi sağlanmaktadır.



üzerinde önemle durmamız gereken nokta ise 1,1 ile 1,2 koordinatlarındaki hücrelere veriyi nasıl yazdığımızdır. Dikkat edilecek olursa 1,1 hücresine yazı yazmak için **Range** tipine bir dönüştürme işlemi yapılmıştır. Bu dönüşüm işlemi sonrasında **Value2** özelliğine ulaşılabilmiştir. Devam eden kod satırında ise önce **Range** tipine dönüştürme ve atama işlemi yapılmış, sonrasında Value2 özelliğine gidilmiştir. Bu açıdan bakıldığında bir dönüştürme işlemi yapılmasının kaçınılmaz olduğu görülmektedir. Elbette bu eskiden

böyleydi. Artık **dynamic** tipinin **COM Interop** nesneleri içerisine serpiştirildiğini görmekteyiz. Aşağıdaki görüntü bu durumu son derece iyi açıklamaktadır.

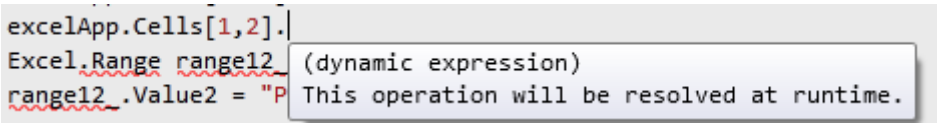


Görüldüğü gibi **Cells** üzerinden ulaşılan tip dynamic olarak ele alınmaktadır. Buna göre yukarıdaki kod parçası **dynamic import** kabiliyeti sayesinde dönüştürme işlemlerine gerek duyulmadan aşağıdaki gibi yazılabilir.

excelApp.Cells[1, 1].Value = "ID";

Excel.Range range12_ = excelApp.Cells[1, 2]; // Cast yapılmadan doğrudan atama işlemi
range12_.Value2 = "Phone Number";

Sonuç aynı olacaktır. Dikkat edilmesi gereken nokta herhangibir **cast** işlemi yapılmasına gerek olmayışıdır. Diğer yandan **dynamic** olan tiplerin çalışma zamanında çözümlenmesi söz konusu olduğundan aşağıdaki durumda bir handikap olarak görülebilir.



Cells[1,2]. sonrasında kullanabileceğimiz metodları gören, bilen, hatırlayan var mı? 🤔

Omitting Ref;

Yine **COM Interop** nesneleri ile olan münasibetlerimizde yaşadığımız sorunlardan biriside **ref** tipinden parametrelerin aktarılması için mutlaka geçici de olsa değişken tanımlamaları yapılması gerekliliğidir. Durumu daha net anlayabilmek için aşağıdaki kod parçasını göz önüne alalım.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Word = Microsoft.Office.Interop.Word;

namespace COMInteropFeatures
{
   class Program
   {
      static void Main(string[] args)
```

```
        {
            #region omitting ref

            #region öncesi

            Word.Application wordApp = new Word.Application();
            wordApp.Visible = true;
            object filePath = Environment.CurrentDirectory+"\\Belge.docx";
            object missing = Type.Missing;

            wordApp.Documents.Open(ref filePath, ref missing, ref missing, ref
missing, ref missing, ref missing,  ref missing, ref missing, ref missing, ref missing, ref
missing,  ref missing, ref missing, ref missing, ref missing, ref missing);

            #endregion

            #endregion
        }
    }
}
```

Bu seferki örneğimizde çok basit olarak **Word Interop** nesnesini kullanarak Belge.docx isimli dosyanın açılması sağlanmaktadır. Ancak **Open** metodunun yazılışı mutlaka dikkatinizi çekmiştir. Peki bir sürü **ref missing** yazmamış dışında bir sıkıntı görebiliyor musnuz? Aslında Named ve Optional Parametre özellikleri ile bu kod stilinden zaten kurtulduk. Ne varki buradaki sıkıntı bu değil. Sıkıntı, **ref** tipinden olan parametreler için **missing** isimli **object** tipinden bir değişken tanımlamak zorunda olmamız. Bu birden fazla çeşitte ref parametresi alan bir**COM Interop** çağrısı için birden fazla geçici değişken tanımlamak zorunda kalabiliriz anlamına da gelmekte. İşte **C# 4.0** ile gelen **Omitting Ref***(ref' leri göz ardı etmek olarak düşünebiliriz)* kabiliyeti sayesinde artık ref olarak kullanılması gereken parametrelere **değer türü(Value Type)** şeklinde argüman geçirebilmekteyiz. Peki ref kullanımından kaçılıyor mu? Elbetteki hayır. Arka planda derleyici bizim için gerekli geçici değişkenleri zaten oluşturuyor ve metod yine referans tipinden gelen parametreler ile çalışıyor. Kısacası yukarıdaki kodu aşağıdaki şekilde yazmamız mümkün.

```
wordApp.Documents.Open(filePath, Type.Missing, Type.Missing);
```

Dikkat edileceği üzere doğrudan değer ataması yapılmış, herhangibir değişken kullanımına gidilmemiştir.

İşte sizlere bir kaç dakika içerisinde çerez niyetine okuyup öğrenebileceğiniz bir yazı. Umarım faydalı olmuştur. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

.Net 4.0 - Lazy Initialization [Beta 2] (2009-12-14T10:30:00)

c# 4.0,.net framework 4.0,

 Merhaba Arkadaşlar,

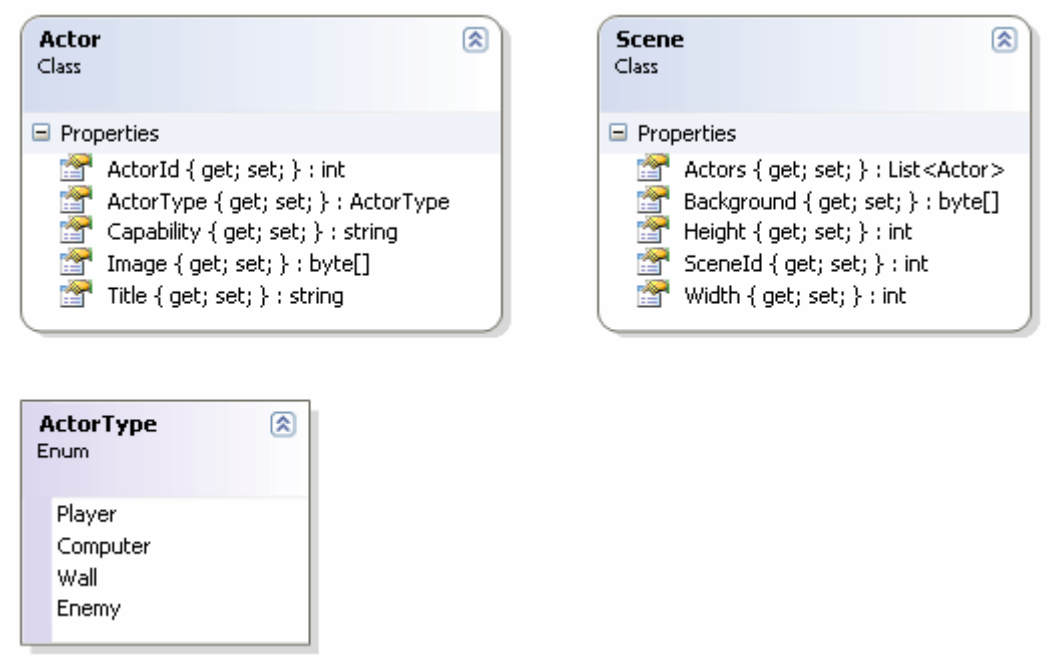
Eminim hepimiz arada sırada tembellik yapıyor ve ilk bulduğumuz rahat köşeye kıvrılıp hiç bir şeyi düşünmeden rahatça uyuyabiliyoruz. Eğer bulunduğumuz yer uyumaya çok müsait değilse yandaki kedi gibi ortama ayak uydurup yinede uyuyoruz Siz hiç gözleri açık uyuyabilen insanlar gördünüz mü? Bunun adı düpe düze tembellik olabiliyor bazen. İhtiyaç dışında uyumak ve hiç bir şeyle uğraşmadan öylece kala kalmak tembelliğin doruk noktaya ulaştığı anlar olarak düşünülebilir. Ama gelin görün ki, programatik ortamda da nesnelerin zaman zaman tembellik etmesi gerekmektedir. Bugünkü yazımızda bu konuyu değerlendiriyor olacağız. Peki neymiş şu Lazy Initialization bir bakalım.

Lazy Initialization yetenekleri sayesinde programların gereksiz bellek tüketimlerinin önüne geçilebilir ve ayrıca performans kazanımı sağlanabilir. Aslında uygulamalarımızda Lazy Initialization kullanmamız için gerek ve yeter iki sebep bulunmaktadır. İlk olarak üretilme maliyetleri yüksek olan nesnelerin tanımlandıkları anda oluşturulmaları yerine, kullanılmaya başladıkları yerde oluşturulmaları gerektiği durumlarda tercih edilebilir. Bu noktada aklımıza **LINQ** içerisinde zaten var olan **Deferred Execution** özelliği gelmektedir ki **Lazy Initialization** yeteneklerine sahip **.Net CLR** tipleride tam olarak bunu sağlamak üzere tasarlanmıştır. İkinci bir sebep olarak, maliyeti yüksek olan bazı hesaplamaların tamamlanmasından sonra ilgili nesnelerin oluşturulması istenebilir. Söz gelimi uygulamamın çalıştırılması ile üretilen n sayıda nesneden sadece gerekli olanların üretilmesi istendiği durumlarda **Lazy Initialization** yeteneklerinden yararlanılabilir.

çok doğal olarak nesnelerin ihtiyaç duyuldukları anda oluşturulmasının sağlanması için kendi tekniklerimizi de geliştirebiliriz. Bu noktada **Reflection'** ın büyük önemi olduğunu vurgulamak gerekir. Nitekim nesnelerin oluşturulmasI işlemini üstlenecek bir **Wrapper'** ın mutlaka tasarlanmış olması ve hatta içerisinde **Instance** üretimi için gerekli kodlamaların yapılması gerekmektedir. **Activator** sınıfının çalışma zamanında nesne üretimi ile ilişkili çeşitli **static** metodları bu amaçla kullanılabilir. Ancak **.Net Framework 4.0** ile birlikte **Lazy Initialization** için kullanılabilecek **Lazy<T>** sarmalayıcı sınıfı gelmektedir. Bu tip sayesinden nesnelerin gerek duyulduğu zamanlarda oluşturulmasının sağlanması son

derece kolaylaştırılmaktadır. üstelik **Lazy<T> Thread Safe** bir tip olarak kullanılabilmektedir.

Dilerseniz konuyu biraz daha net kavramak adına basit bir örnek geliştirelim. örnek senaryomuza göre program içerisinde ele alınan bir oyun sahnesi ve bu sahne içerisinde yer alan grafiksel şekillerin tutulduğu çeşitli nesnelerin üretim işleminde Lazy Initialization tekniklerinin nasıl kullanıldığını incelemeye çalışacağız. Bu senaryoda **Lazy Initialization** kullanmamız için öne sürdüğümüz sebep ise şu olacak; programın ilerleyen adımlarında pek çok farklı sahne ve bu sahne içerisinde yer alan grafik nesnelerin yeri geldiğinde oluşturulması ve böylece programın başlangıcında n sayıda sahne için yapılacak olan oluşturulma maliyetinin azaltılması. Bu amaçla ilk olarak **Visual Studio 2010 Ultimate Beta 2** üzerinde aşağıdaki tiplere sahip bir **Console** uygulaması geliştirdiğimizi düşünelim.




```
using System.Collections.Generic;

namespace BeLazy
{
   enum ActorType
   {
      Player,
      Computer,
      Wall,
      Enemy
   }
   class Actor
   {
      public int ActorId { get; set; }
```

```
        public string Title { get; set; }
        public string Capability { get; set; }
        public ActorType ActorType { get; set; }
        public byte[] Image { get; set; }

        public override string ToString()
        {
            return string.Format("Actor Id {0} Title {1} Capability {2} Actor Type {3}",
ActorId.ToString(), Title, Capability,ActorType.ToString());
        }
    }
    class Scene
    {
        public int SceneId { get; set; }
        public int Width { get; set; }
        public int Height { get; set; }
        public byte[] Background { get; set; }
        public List<Actor> Actors { get; set; }

        public override string ToString()
        {
            return string.Format("SceneId {0} Widht {1} Height {2}", SceneId.ToString(),
Width.ToString(), Height.ToString());
        }
    }
}
```

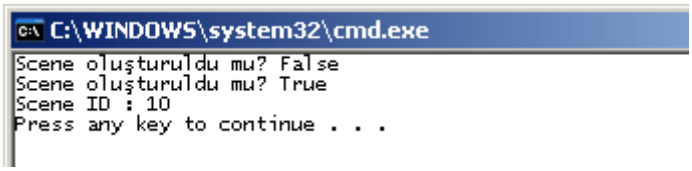
Tamamen hayali olarak tasarlanmış olan bu tiplerde ana fikir **Scene** tipi içerisinde bir **Actor** listesinin olmasıdır. Her iki tip içerisinde **byte[]** tipinden özellikler yer almaktadır. **Scene** nesnelerinin sayısının çok fazla olduğu bir durumda, tümünün bir seferde oluşturulması bellek üzerindeki tüketimi arttırabileceği gibi program performansınıda olumsuz yönde etkileyebilir. Bu nedenle**Lazy Initialization** tekniklerinden faydalanarak sadece gereksinim duyulduğu yerde oluşturulma işlemi yolu tercih edilebilir.

Peki oluşturulma işlemi hangi aşamada gerçekleşmektedir? Bu noktada **Lazy<T>** tipinin **Value** özelliği devreye girer. **Value** özelliğinin kullanıldığı ilk yerde **T** tipinin oluşturulma işlemi başlamaktadır. Şimdi bu durumu analiz etmek amacıyla Program kodunun içeriğini aşağıdaki gibi yazdığımızı düşünelim.

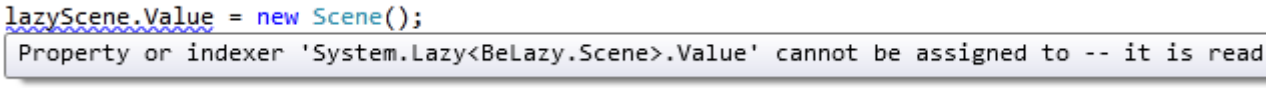
```
using System;

namespace BeLazy
{
    class Program
```

```
    {
        static void Main(string[] args)
        {
            Lazy<Scene> lazyScene = new Lazy<Scene>();

            Console.WriteLine("Scene oluşturuldu mu? {0} ",lazyScene.IsValueCreated);
            lazyScene.Value.SceneId = 10;
            Console.WriteLine("Scene oluşturuldu mu? {0} ", lazyScene.IsValueCreated);
            Console.WriteLine("Scene ID : {0}",lazyScene.Value.SceneId.ToString());
        }
    }
}
```

İlk olarak **Lazy<Scene>** tipinden bir nesne örneği oluşturulmaktadır. Sonrasında **IsValueCreated** özelliği yardımıyla **Scene** nesnesinin oluşturulup oluşturulmadığına bakılır. Bir sonraki satırda dikkat edileceği üzere **Value** özelliği üzerinden gidilerek **SceneId** için değer ataması yapılmıştır. Uygulamanın çalışma zamanı çıktısı aşağıdaki gibidir.



Volaaaa!!! 😎 Görüldüğü üzere **Value** özelliği çağırılana kadar **IsValueCreated** özelliği **false** değer döndürmektedir. Bir başka deyişle söz konusu nesne henüz oluşturulmamıştır. Ancak **Value** özelliğinin kullanılması ile birlikte bir **Scene** nesnesinin örneklendiği görülmektedir. Burada dikkat edilmesi gereken noktalardan biriside **Value** özelliğinin**Readonly** olmasıdır. Bir başka deyişle **Value** özelliği ile elde edilen bir nesne referansına yeni bir atama gerçekleştirilemez.



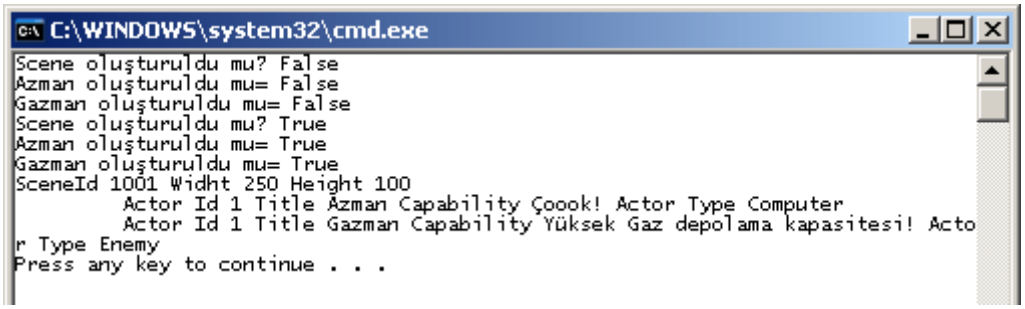
Ama tabiki **Value** üzerinden gidilen tipin özellikleri değiştirilebilir. örneğimizde dikkat çekici noktalardan bir diğeri de **Scene** nesnesinin aslında karmaşık bir tip olarak içerisinde başka tipleride barındırıyor olmasıdır. Buna göre **Value** özelliğinden yararlanılarak diğer tiplerin değerlerinin de atanması sağlanabilir...Ya da...Ya da aşağıdaki kod parçasında olduğu gibi tembel bir üretim gerçekleştirilebilir.

```
using System;
using System.Collections.Generic;
```

```
namespace BeLazy
{
   class Program
   {
      static void Main(string[] args)
      {
         Lazy<Actor> azman = new Lazy<Actor>(() =>
            new Actor {
                ActorId=1,
                 ActorType= ActorType.Computer,
                  Capability="çoook!",
                   Image=new byte[1000],
                    Title="Azman"
            }
            );

         Lazy<Actor> gazman = new Lazy<Actor>(() =>
            new Actor
            {
               ActorId = 1,
               ActorType = ActorType.Enemy,
               Capability = "Yüksek Gaz depolama kapasitesi!",
               Image = new byte[5000],
               Title = "Gazman"
            }
            );

         Lazy<Scene> scene = new Lazy<Scene>(() =>
            new Scene
            {
               SceneId = 1001,
               Background = new byte[10000],
               Height = 100,
               Width = 250,
               Actors = new List<Actor> { azman.Value, gazman.Value }
            }
            );

         Console.WriteLine("Scene oluşturuldu mu? {0} ",scene.IsValueCreated);
         Console.WriteLine("Azman oluşturuldu mu= {0}",azman.IsValueCreated);
         Console.WriteLine("Gazman oluşturuldu mu= {0}", gazman.IsValueCreated);

         Scene currentScene=scene.Value;

         Console.WriteLine("Scene oluşturuldu mu? {0} ", scene.IsValueCreated);
```

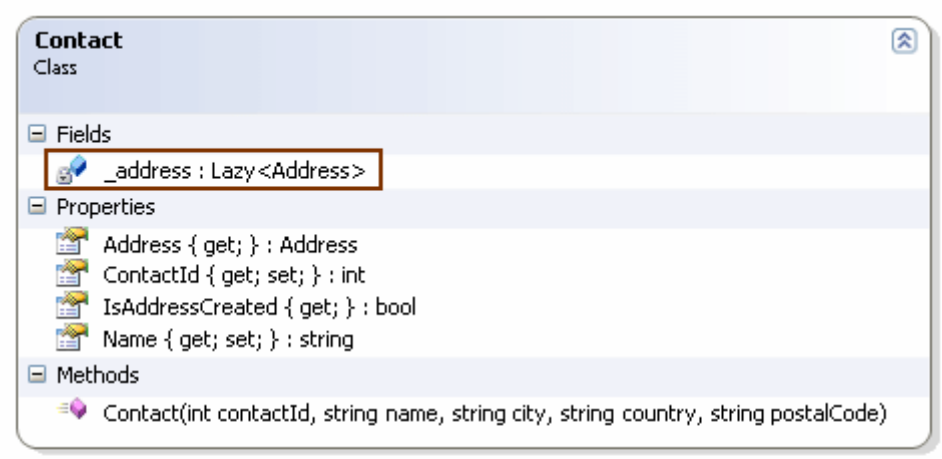
```
            Console.WriteLine("Azman oluşturuldu mu= {0}", azman.IsValueCreated);
            Console.WriteLine("Gazman oluşturuldu mu= {0}", gazman.IsValueCreated);

            Console.WriteLine(currentScene.ToString());

            foreach (Actor actor in currentScene.Actors)
            {
               Console.WriteLine("\t {0}",actor.ToString());
            }
        }
    }
}
```

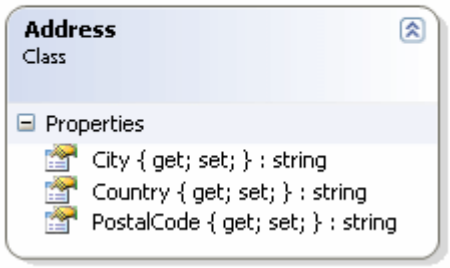
Buna göre çalışma zamanı içeriği aşağıdaki gibi olacaktır.



Görüldüğü üzere **Scene** ve **Actors** özelliğinin işaret ettiği koleksiyon içerisindeki tüm **Actor** tipleri **Lazy** olarak üretilmektedir. üretim noktası ise **currentScene** değişkenin **Value**özelliği ile atamanın yapıldığı satırdır. Bu satırdan sonra **Scene** nesnesi ve içeriğindeki **Actor** nesnelerinin oluşturulması gerçekleştirilmektedir. Tabi istenirse **Actor** oluşturma işlemleride sonraki bir adıma bırakılabilir.

İlginç olan bir noktada, **Lazy<T>** tipinden özelliklerin(Type Property) tanımlanabilmesidir. Dilerseniz aşağıdaki örnek kod parçasını göz önüne alalım.



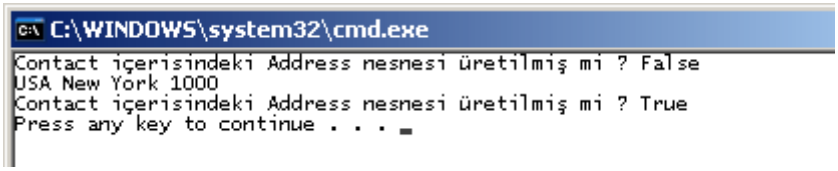



```
using System;
using System.Collections.Generic;

namespace BeLazy
{
   class Program
   {
      static void Main(string[] args)
      {
         Contact cntc = new Contact(1, "Burak Selim Şenyurt", "New York", "USA",
"1000");
         Console.WriteLine("Contact içerisindeki Address nesnesi üretilmiş mi ?
{0}", cntc.IsAddressCreated);
         Console.WriteLine("{0} {1}
{2}",cntc.Address.Country,cntc.Address.City,cntc.Address.PostalCode);
         Console.WriteLine("Contact içerisindeki Address nesnesi üretilmiş mi ?
{0}", cntc.IsAddressCreated);
      }
   }

   class Contact
   {
      private Lazy<Address> _address;
```

```
        public Contact(int contactId,string name,string city,string country,string postalCode)
        {
            _address = new Lazy<Address>(
                () => new Address {
                    City=city,
                    Country=country,
                    PostalCode=postalCode
                }
                );

            ContactId = contactId;
            Name = name;
        }

        public Address Address
        {
            get
            {
                return _address.Value;
            }
        }
        public bool IsAddressCreated
        {
            get
            {
                return _address.IsValueCreated;
            }
        }
        public int ContactId { get; set; }
        public string Name { get; set; }

    }

    class Address
    {
        public string City { get; set; }
        public string Country { get; set; }
        public string PostalCode { get; set; }
    }
}
```

örnekte yer alan **Contact** sınıfı içerisinde **Lazy<Address>** tipinden **private** bir alan tanımlandığı görülmektedir. Bu alanın oluşturulması **Contact** sınıfının yapıcı metodu içerisinde gerçekleştirilmektedir. **Address** isimli özellik ilede, **Lazy<Address>** tipinden olan **_address** alanının **Value** özelliğinin değeri geriye döndürülmektedir.

Zaten **Value** özelliğine referans atanması gerçekleştirilemediğinden **Address** isimli özellik sadece **get** bloğuna sahip olacak şekilde tanımlanmıştır**(Read Only)**. Bu kodlamaya göre **Contact** sınıfına ait bir nesne örneklendikten sonra içerisinde yer alan **Address** nesnesi hemen oluşturulmaz. Ancak **Contact** sınıfının nesne örneği üzerinden **Address** özelliğine gidilir ve **Value** özelliğine ulaşılırsa örnekleme işlemi gerçekleşecektir. Bu durum çalışma zamanında daha net bir şekilde görülebilir.



Dikkat edileceği üzere **Contact** nesnesi örneklense bile **Address** nesnesi henüz oluşturulmamıştır. Ancak **Address** nesnesinin **Value** özelliğine ulaşıldığı ilk yerde bu işlem gerçekleşmektedir.

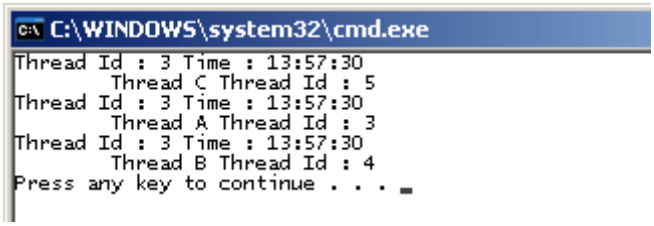
Lazy<T> tipinin kullanımında dikkatli olunması gereken noktalardan biriside **Multi-Thread** operasyonlardır. özellikle birden fazla **Thread'** in aynı **Lazy<T>** nesne örneğini kullanması halinde ilgili nesnenin hangi **Thread** içerisinde oluşturulacağının bir önemi kalmamaktadır. Ancak oluşturulma işlemleri sırasında meydana gelecek **istisnalarda(Exceptions)** diğer **Thread'** lerinde otomatik olarak etkilenmesi söz konusudur. Buna ek olarak **Lazy<T>** tipleri oluşturulurken **Thread Safe** olup olmayacakları çok basit bir şekilde belirlenebilir. Bu durmu daha net bir şekilde anlayabilmek adına dilerseniz aşağıdaki kod parçasını göz önüne alalım.

```
using System;
using System.Collections.Generic;
using System.Threading;

namespace BeLazy
{
   class Program
   {
      static void Main(string[] args)
      {
         Lazy<Information> information = new Lazy<Information>();

         Thread threadA=new Thread(
            ()=>
               {
                  Console.WriteLine(information.Value.ToString());
                  Console.WriteLine("\tThread A Thread Id :
{0}",Thread.CurrentThread.ManagedThreadId.ToString());
               }
               );
```

```
            Thread threadB = new Thread(
                () =>
                {
                    Console.WriteLine(information.Value.ToString());
                    Console.WriteLine("\tThread B Thread Id :
{0}", Thread.CurrentThread.ManagedThreadId.ToString());
                }
                );

            Thread threadC = new Thread(
                () =>
                {
                    Console.WriteLine(information.Value.ToString());
                    Console.WriteLine("\tThread C Thread Id :
{0}", Thread.CurrentThread.ManagedThreadId.ToString());
                }
                );

            threadA.Start();
            threadB.Start();
            threadC.Start();

            threadA.Join();
            threadB.Join();
            threadC.Join();
        }
    }

    class Information
    {
        private int threadId;
        private string initializeTime;

        public Information()
        {
            threadId = Thread.CurrentThread.ManagedThreadId;
            initializeTime = DateTime.Now.ToLongTimeString();
            Thread.Sleep(2000);
        }

        public override string ToString()
        {
            return String.Format("Thread Id : {0} Time : {1}", threadId.ToString(),
initializeTime);
        }
```

```
    }
}
```

örneğimizde 3 farklı **Thread'** in çalıştırıldığı görülmektedir. Bu **Thread'** lerin tamamı kendi içlerinde **Lazy<Information>** tipinden olan **information** değişkenini kullanmaktadır. Buna göre **Thread'** lerden hangisi ilk olarak **Value** özelliğine erişirse, **Information** nesnesi örneklemiş olacaktır. Buna ek olarak çalışma zamanındaki sonuçlara bakıldığında tüm**Thread'** lerin ilk üretilen **Information** nesne örneğine ulaştığı bu nedenle **Information** yapıcı metodunda oluşturulan **ManagedThreadId** değerlerinin ve zamanların aynı olduğu görülür. İşte sonuçlar.



Bu senaryoya göre nesnenin hangi **Thread** içerisinde oluşturulduğunun hiç bir önemi yoktur. Şimdi akla şöyle bir soru gelebilir. **Lazy<Information>** örneği oluşturulurken **Thread-Safe** davranılacağı nerede belirtilmiştir?

Aslında **Lazy<T>** tipinin **isThreadSafe** parametresinin değeri varsayılan olarak **true'** dur ve istenirse yapıcı metod içerisinde değiştirilebilir. özelliğe **false** değer atanması halinde genellikle ilgili nesnenin tek bir **thread** içerisinde kullanıldığı varsayılır ve bu azda olsa performans kazanımına neden olur. Ancak birden fazla **thread'** in aynı **Lazy<T>** nesnesini kullanacağı hallerde koordinasyon daha büyük önem kazanmaktadır ve bu nedenle **isThreadSafe** özelliğinin değerinin **true** olarak bırakılması önerilir.

Kişisel Not : *özellikle birden fazla **Thread'** in aynı nesnenin kendi içlerinde farklı kopyalarını alarak kullanmaları istendiği durumda **.Net 4.0** ile gelen **ThreadLocal** tipinden yararlanılabilir ki **.Net 3.5'** te bunun için **ThreadStatic** isimli bir nitelikten yararlanılmaktadır. İşte size güzel bir araştırma konusu* 😃

Buraya kadar anlattıklarımıza göre dikkat edilmesi gereken bazı noktalar olduğuda aşikardır;

- Lazy<T> içerisinde kullanılacak tipin mutlaka **varsayılan yapıcı(Default Constructor)** metodu olmalıdır. Nitekim oluşturma işleminde aslında **Activator** sınıfının**CreateInstance** metodu devreye girmektedir.
- **Value** özelliği **ReadOnly'** dir. Bu nedenle kullanıldıktan sonra T tipinden bir referansın atanması gerçekleştirilemez.
- **Multi-Thread** senaryolarda **isThreadSafe** özelliğini **true** olarak bırakmak doğru bir yaklaşımdır.
- **Single-Thread** senaryolarda **isThreadSafe** özelliği performans kazanımı adına istenirse **false** bırakılabilir.

- **Value** özelliğine erişildiğinde oluşabilecek bir istisna halinde kodun ilerleyen kısımlarında **Value** özelliğine giden her çağrı için aynı **Exception** örneği söz konusu olacaktır. Yani Value özelliğinin ilk çağırıldığı yerde oluşacak bir istisna devam eden Value çağrılarına da akacaktır. Bu **Multi-Thread** senaryolarda da **isThreadSafe** özelliğinin değeri **true** bırakıldığı sürece de geçerlidir.
- çekinilmesi gereken nokta şudur; **isThreadSafe** özelliğine **false** değer verilmesi halinde **thread'** ler arasında senkronizasyon ortadan kalkacağından, bir **Thread'** in **exception** gördüğü noktada diğer bir thread kullanılabilir bir **Value** özelliği ile karşılaşabilir. Aman dikkat.

Umarım faydalı bir yazı olmuştur. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

BeLazy.rar (28,31 kb)

Microsoft Distributed Cache(Velocity Project) - Hello World (2009-12-11T09:25:00)

velocity,microsoft distributed cache,windows server appfabric,

Merhaba Arkadaşlar,

Bazen bir otomobilin çok hızlı gitmesi, yüksek süratlere çıktığında direksiyonunun titrememesi, yolda sağa sola savrulmaması, keskin virajlarda rahatlıkla tutunabilmesi, vitesler arasındaki geçişlerde torkunun mümkün olduğunca korunabilmesi, frenajlarda kontrolü sağlaması vb... istenir. Yarış otomobillerinde bu ve benzeri ihtiyaçlar için çok sık duyduğumuz tunning olarak isimlendirilen geliştirmeler yapılır. Bazen yarışın kategorisine göre neredeyse teknolojinin sınırlarını zorlayacak yeniliklere şahit olunur. Ama özetle aracın performansını arttıracak her yol, kuralları çerçevesinde mübahtır. Performansı arttırmak önemli bir kriterdir ve her zaman kolay bir şekilde sağlanamamaktadır. Bazen motor yağına konulan katkı maddeleri ile basit şekilde, bazen yarış pistindeki ıslak zemine göre takılan lastikler zor şekilde vb... Yine enteresan bir giriş olduğunun farkındayım ama bu günkü konumuz yazılımda performansı arttırmak adına kullanılan önemli tekniklerden birisi ile ilgilidir.**Caching**. **Web** uygulamalarında sıklıkla karşılaştığımız, **Enterprise**

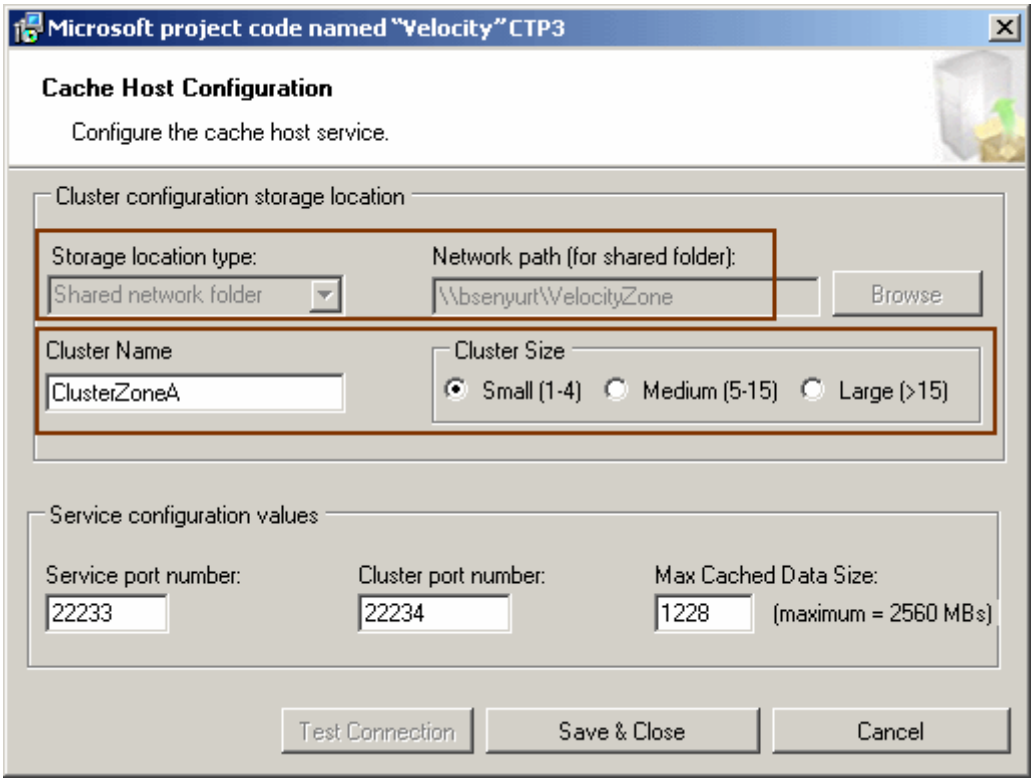
Library Caching Block sayesinde Web uygulaması sınırlarını aşarak diğer uygulamalarda daha kolay kullanılabilir hale gelmiş bu kavramın servis bazlı hale getirilebildiğini duysanız acaba ne düşünürdünüz  İşte **Microsoft'** un uzun bir süre önce duyurduğu ve şu anda **CTP 3** sürümü bulunan kod adı **Velocity** projesi...

Microsoft' un Velocity kod adlı projesi aslında **Distributed Caching Service** olarakta bilinmektedir. Projenin en büyük amacı, her çeşit veri içeriğinin ön belleklenerek saklanabilmesini sağlamaktır. Bu açıdan düşünüldüğünde **Enterprise Library** içerisindeki **Caching** bloğu veya **Asp.Net** içerisinde kullanılan **Cache** tekniklerinden bir farkı yokmuş gibi görünebilir. Hatta tüm bu modellerin ortak amacının, uygulamaların fiziksel veri bölgelerine tekrar tekrar gitmelerini engelleyip performansı arttırmak olduğu da aşikardır. Aslında aradaki tek fark Velocity projesinde bir servis anlayışının olması değildir. Farkı görmek için projenin detaylarına belikde daha yakından ve derinden bakmak gerekmektedir. öncelikli olarak projenin ne gibi yetenekleri sunduğuna maddeler halined bir bakalım;

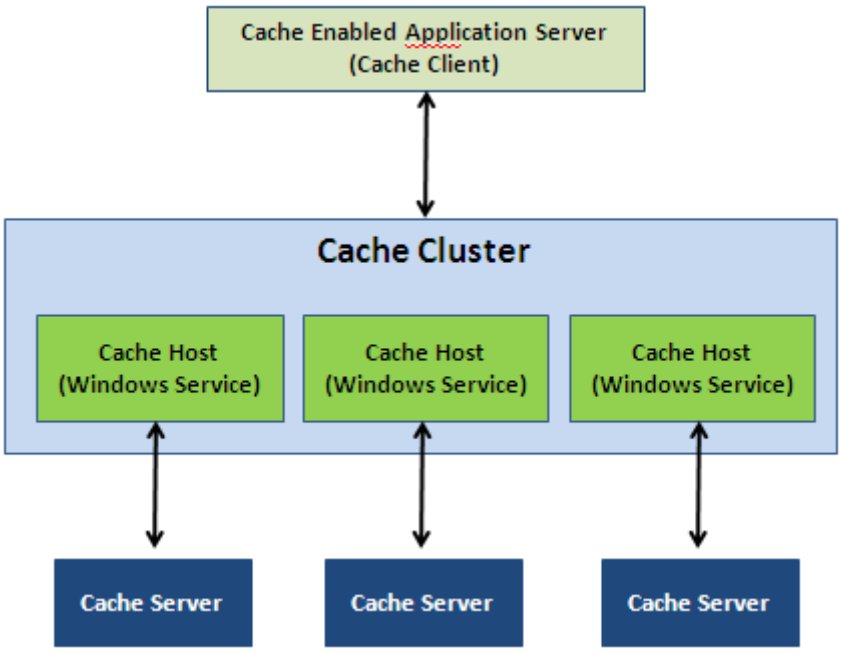
- Veritabanı veya Hard Disc üzerindeki herhangibir veri içeriğinin Cache' lenebilmesi
- Yüksek ölçeklenebilirlik(High Scalability)
- Cache yönetiminin Windows Service tarafından sağlanması(Varsayılan olarak Network Service hesabı ile yüklenen bir Windows Service' inden bahsediyoruz)
- Clustering ile Cache üzerinde yük dağılımının dengelenmesinin(Load Balancing) otomatik yönetimi
- Cache içerisindeki veriye anahtar alanlar, farklı tanımlayıcılar yada name tag' leri ile erişilebilmesi
- Optimistic ve Pesimistic Concurrency modellerini desteklemesi
- Asp.Net Session nesnelerinin Distributed Cache üzerinden veritabanına yazılmadan tutulabilmesinin sağlanması
- TCP/IP bazlı servis alt yapısının kullanılmas(Dikkat; Firewall istisnaları oluşabilir)

vb...

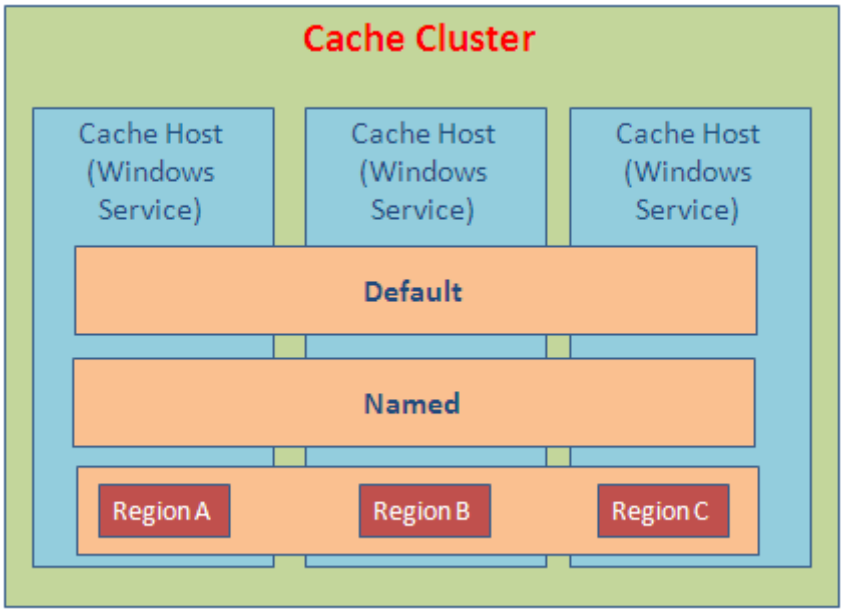
Şimdi bu konu ile ilişkili bir örnek geliştirmeye çalışacağız. Ama öncesinde Cache yönetimini üstlenecek makineye **Velocity** projesini kurmamız gerekiyor. Ne varki bu kurulum her ne kadar Next-Next şeklinde görünse de dikkat edilmesi gereken bazı noktalar var. Kurulum ile ilişkili en önemli noktalardan birisi **depolama yeri(Storage Location)** ve **tipi(Storage Type)**. Ben kendi kurulumumda**ağ paylaşımı(Network Shared)** yapılmış ve gerekli **izinler(Permissions)** tamamlanmış olan **VelocityZone** isimli bir klasörü kullanmayı planlıyorum. Bu belirleme işlemi sırasında sizlere sorun çıkartabilecek noktalar **\\makineAdı\KlasorAdı** isimli bir lokasyonun olmayışı*(ki bunu kendimiz belirliyoruz)* ve bu alanla ilişkili yeterli **izinlerin(Permission)** bulunmayışı olacaktır. Bu sorunlar aslında **Test Connection** düğmesine tıklandığında çıkan hata mesajları yardımıyla görülebilir. Eğer herhangibir sorun yoksa **Cluster** oluşturulması adımına geçilmektedir. örnekte ben **ClusterZoneA** isimli ve **Small** tipinden bir **Cluster** oluşturdum.



Small seçimi aslında arka planda **1** ile **4** arası **Cache Server'** ın kullanılabileceğini ifade etmektedir. Tahmin edileceği üzere örneği yerel makine üzerinden çalıştırdığımızdan burada tek bir **Cache Server** mevcuttur. Tam bu noktada aslında mimari modelden biraz bahsetmekte yarar olacağı kanısındayım. Aslında istemci için mantıksal olarak tek bir **Cache** birimi ile çalışmaktadır. Fakat birden fazla **Cache Server** olabilir ve bunlar arka planda farklı makinelerdeki **Cache Service'** leri*(ki Windows Service' leridir)* refere edebilir. Buda zaten **Load Balancing** kurulumu açısından önemlidir. Aynen aşağıdaki şekilde olduğu gibi.

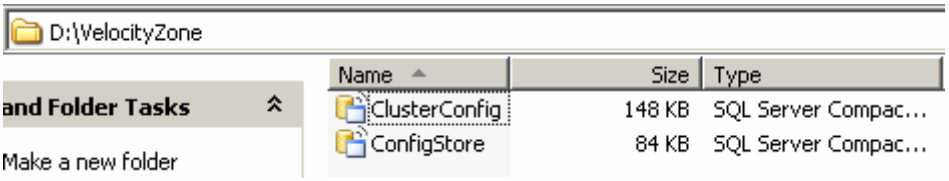


Birde mantıksal modele bakalım mı? Haydi bakalım.



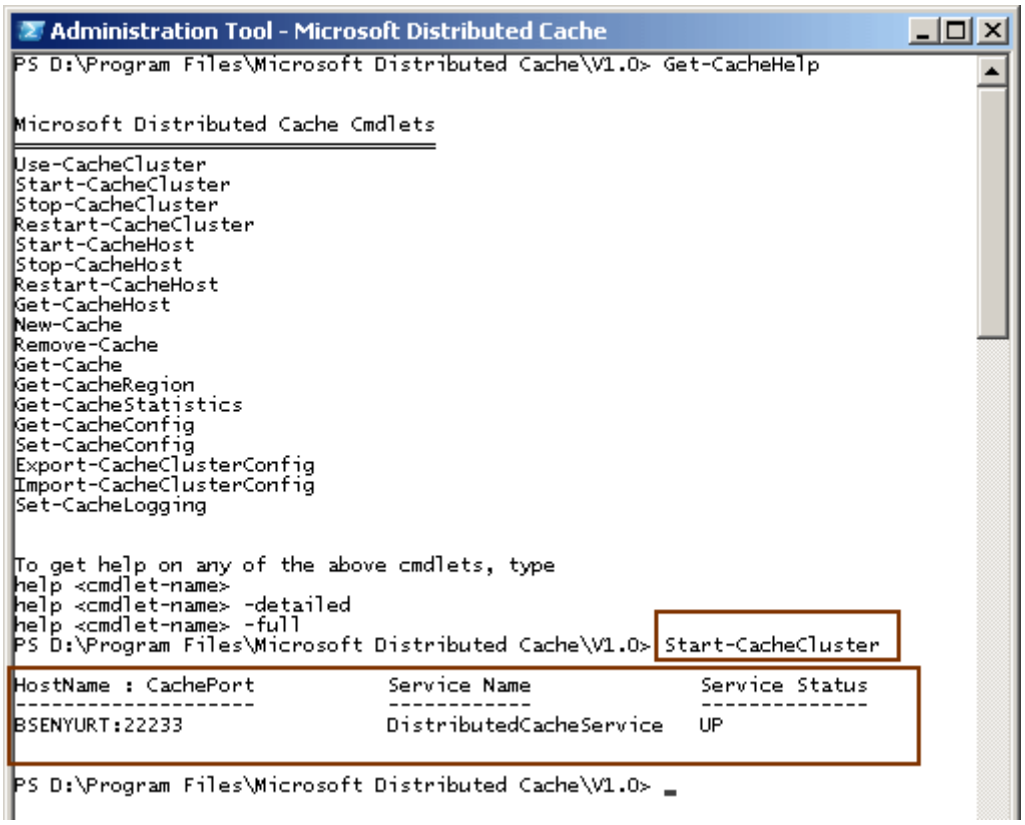
İlk kurulumnda varsayılan Cache mantığı kullanılır. Ancak istenirse farklı isimlendirmelere sahip birden fazla Cache alanı kullanılabilir. özellikle uygulamalar kaç adet olursa olsun her biri için ayrı ayrı Cache isimlendirilmeleri kullanılabileceğini düşünebiliriz. Buna göre her farklı Cache birbirlerinden tamamen bağımsız olarak değerlendirilebilir. örneğin **Policy'** leri ayrı ayrı belirlenebilir. Diğer yandan **Regions** bazlı mantıkta çalışma zamanında ayarlamalar yapılması gerekmektedir. Bir başka deyişle **Region'** ların oluşturulması çalışma zamanında gerçekleştirilir. **Region** mantığına göre **Cache** içeriklerine **Key'** den farklı olarak **Tag'** ler yardımıyla*(ki birden fazla Tag değerinin bir Cache nesnesi ile ilişkilendirebiliriz)* ulaşılabilir. Burada **Region** bazlı bir ayrıştırma yapıldığından arama gibi işlemlerde **Tag'** lerden yararlanılabilir. **Region** mantığında **Cache** içerikleri tek bir host içerisinde tutulur. Yani diğer modellerdeki gibi servisler tarafından dağıtık olarak kullanılmazlar. Bu önemli bir farktır. Nitekim arama fonksiyonelliğinin avantajı ortaya çıkarken, Named mantığında olduğu gibi dağıtık kullanım söz konusu olmadığından **ölçeklenebilirlik(Scalability)** ortadan kalkmaktadır.

Kurulum işleminin sonrasında klasör içeriğinin aşağıdaki gibi şekillendiğini görebiliriz.

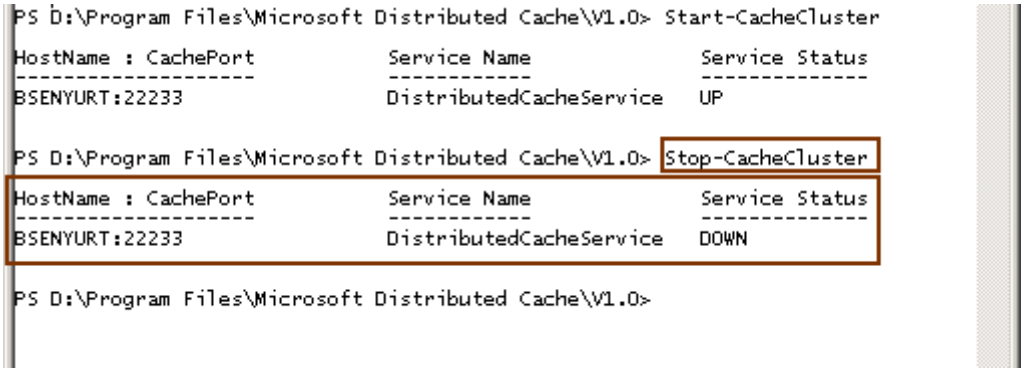


Kurulum işlemi bu şekilde tamamlandıktan sonra programlardan**(Microsoft Distributed Cache | Administration Tool - Microsoft Distributed Cache)** ilgili **Administration Tool** aracınının kullanılarak **Cluster'** ın başlatılması gerekmektedir. Bu

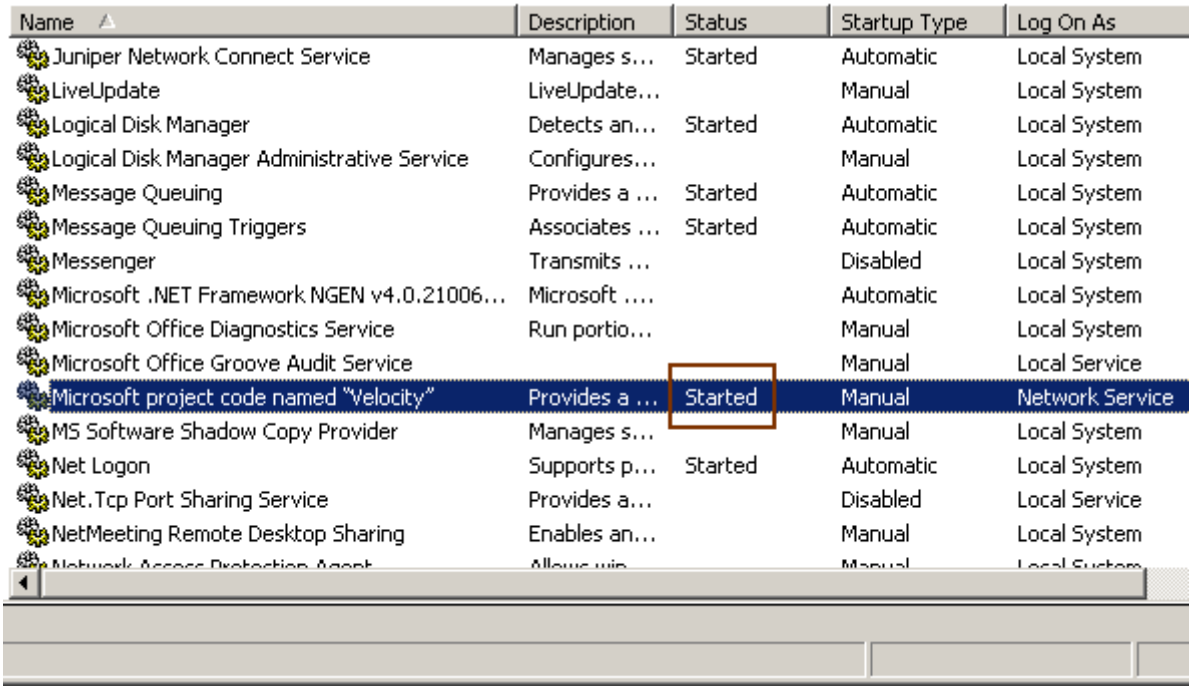
amaçla **Administration Tool** komut satırından **Start-CacheCluster** yazmamız yeterli olacaktır. Böylece ilgili **Distributed Cache Service**örneğinin vermiş olduğumuz kriterlere göre ilgili **Port** üzerinden çalıştırılması sağlanmış olur.



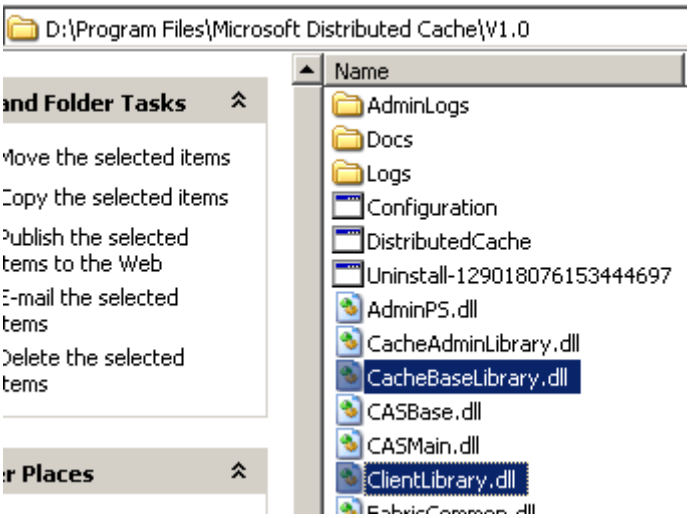
Yukarıdaki ekran görüntüsünde **Get-CacheHelp** kullanımı sonucu yönetsel işlemler için hangi komutların kullanılması gerektiğide görülmektedir. örneğin başlattığımız**Cache Cluster** hizmetini durdurmak için **Stop-CacheCluster** komutunu kullanmamız yeterlidir.



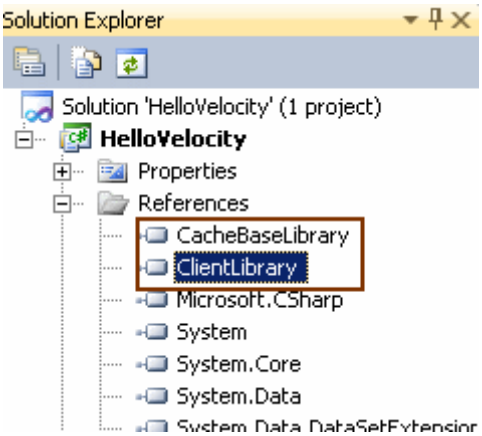
Servis başarılı bir şekilde çalıştırıldığında bunu **Services** aracından da görebiliriz.



Artık kurulumu yaptığımıza ve servisi çalıştırdığımızda göre örnek bir uygulama geliştirerek **Velocity** üzerinden **Cache'** lemenin nasıl yapılabileceğine bakabiliriz. Tabiki söz konusu uygulamanın **Velocity** servisini kullanabilmesi için gerekli bazı **Assembly'** ları referans etmesi gerektiği de ortadadır. Bu referanslar aslında kurulum sonrası **Program Files\Microsoft Distributed Cache\V1.0** klasörüne atılacaktır.



Şimdi **Visual Studio 2010 Ultimate Beta 2** ortamında örnek bir **Windows Forms** uygulaması açalım ve ilgili **Assembly'** ları projemize referans edelim.



Uygulamamızda Velocity projesini kullanabilmemiz için çalışma zamanına bazı konfigurasyon bilgilerinin bildiriminin yapılması gerekmektedir. Bu nedenle **App.config** dosyasının içeriğini aşağıdaki gibi değiştirerek devam edebiliriz.

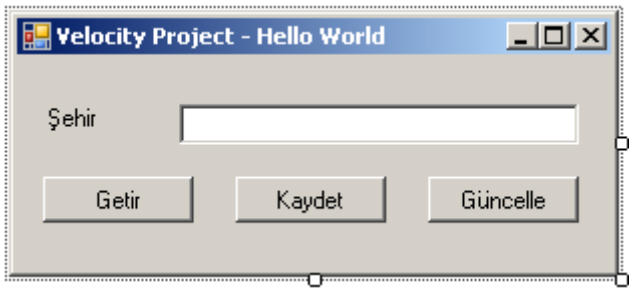
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section name="dataCacheClient"
             type="Microsoft.Data.Caching.DataCacheClientSection, CacheBaseLibrary"
             allowLocation="true"
             allowDefinition="Everywhere"/>
  </configSections>
  <dataCacheClient deployment="simple">
    <localCache isEnabled="true" sync="TTLBased" ttlValue="300" />
    <hosts>
      <host name="bsenyurt" cachePort="22233"
cacheHostName="DistributedCacheService"/>
    </hosts>
  </dataCacheClient>
</configuration>
```

Tahmin edileceği üzere uygulamanın **Distributed Cache** sistemini nasıl kullanacağına dair tüm konfigurasyon bilgileri buradaki ilgili elementler içerisinde bildirilmektedir. Söz gelimi host kısmında az önce başlatılan servise ait bilgiler yer almaktadır. **Port** bilgisi, host makinenin adı gibi. Yine **localCache** elementi içerisinde **Cache** üzerinde verinin tutulma süresinin belirlenmesi için **ttlValue**niteliğine saniye cinsinden bir değer atanmaktadır. Aynı element içerisinde yer alan **sync** niteliğine atanan **TTLBased** değeri ile zaman aşımlı bir **Cache'** leme kullanılacağı bildirilmektedir.

Kişisel Not : *Buradaki ayarları manuel olarak yapmamız bir dezavantaj olarak düşünülebilir. Malum* ***Enterprise Libary*** *içerisinde dahi bu tip ayarlar görsel olarak kolayca yapılabilmektedir. üstelik* ***App.config*** *dosyası içerisinde* ***intelli-sense*** *özelliğide olmadığından şimdilik* ***Copy-Paste*** *kuralları ile ilerlemek zorunda kalmış durumdayız. Bu*

*durumun Relase sürümde özellikle **Visual Studio 2010** ile olan entegrasyonunda düzeleceğini düşünmek istiyorum.*

Neyse biz kaldığımız yerden devam edelim. Windows Form' umuzu aşağıdaki gibi tasarladığımızı düşünelim.



Aslında **Hello World** uygulamamızın amacı çok basit. **Cache**' e veri ekleyebilmek, var olanı çekebilmek ve güncelleyebilmek. **String** tipinden bir veri içeriğini saklıyor olacağız. Ancak tabiki kullanıcı tanımlı tiplerin de saklanabileceğini bir kere daha hatırlatalım. Uygulama kodlarmızı aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.Windows.Forms;
using Microsoft.Data.Caching;

namespace HelloVelocity
{
   public partial class Form1 : Form
   {
      // DataCache nesnesi tanımlanır
      DataCache dCache = null;

      public Form1()
      {
         InitializeComponent();

         DataCacheFactory factory = new DataCacheFactory();
         //Fabrika nesne örneğinin GetDefaultCache metodundan yararlanılarak varsayılan
DataCache nesne referansı elde edilir.
         dCache = factory.GetDefaultCache();
      }

      // Cache' den veri çekme işlemi
      private void btnGetFromCache_Click(object sender, EventArgs e)
      {
         // Get metodu parametre olarak Cache' de duran nesnenin Key özelliğini alır.
```

örneğimizde Key özelliğinin değeri City' dir

```
            string cacheContent = dCache.Get("City") as string;

            if (!String.IsNullOrEmpty(cacheContent))
            {
               // Eğer cacheContext içeriği bol veya null değilse TextBox kontrolüne aktarılır
               txtCities.Text = cacheContent;
            }
         }

         // Cache' e veri ekleme işlemi için kullanılır
         private void btnSaveToCache_Click(object sender, EventArgs e)
         {
            try
            {
               // Add metodunun ilk parametresi Cache' de tutulacak veriyi işaret eden Key değeridir. İkinci parametre ile Cache' de duracak veri içeriği belirtilir.
               dCache.Add("City", txtCities.Text);
            }
            catch (Exception excp)
            {
               MessageBox.Show(excp.Message);
            }
         }

         // Cache' de zaten var olan bir veri içeriğinin güncellemek için kullanılır.
         private void btnUpdate_Click(object sender, EventArgs e)
         {
            try
            {
               // İlk parametre Cache' de duran nesnenin Key değeridir. İkinci parametre ise yeni halidir.
               dCache.Put("City", txtCities.Text);
            }
            catch (Exception excp)
            {
               MessageBox.Show(excp.Message);
            }
         }
      }
   }
```

Aslında **DataCache** nesnesi **Cache** ile ilişkili tüm yönetsel işlemleri üstlenmektedir. **Add, Put** ve **Get** metodları sırasıyla **Cache'** e veri ekleme, Cache' deki veriyi güncelleme ve Cache' den veri çekme işlemleri için kullanılmaktadır. Tüm bu metodlar

mutlaka **Key** ve **Value** değerlerine ihtiyaç duymaktadır. Yani Cache' de duran veriyi işaret eden bir anahtar ile içerik bilgisi. örnekte**Cache** üzerinde **String** bir içerik tutulmaktadır. Ancak karmaşık tiplerin tutulmasıda mümkündür. Söz gelimi geliştirici tarafından tanımlanmış bir nesne içeriğide **Cache** içerisinde tutulabilir. Nitekim özellikle **Get** metodunun dönüş tipine bakıldığında **object** olduğu görülmektedir. Buda zaten herşeyi açıklamaktadır.

Peki örneğimizi nasıl test edeceğiz? İşte örnek senaryo adımları;

1. TextBox içerisine bir şehir adı(örneğin İstanbul) girip Kaydet tuşuna basınız.
2. Programı kapatınız ve tekrardan çalıştırıp Getir tuşuna basınız.
3. Eğer Getir tuşuna bastıktan sonra 1nci adımda girdiğiniz Şehir adını TextBox içerisinde görüyorsanız sonraki adımdan devam ediniz.
4. TextBox içeriğinde değişiklik yapınız. örneğin farklı bir şehir adı giriniz ve bu kez Güncelle tuşuna basınız.
5. Programı yine kapatıp açınız ve Getir tuşuna basarak 4ncü adımda yaptığınız güncellemenin getirildiğinden emin olunuz.
6. Şehir adını değiştirip veya aynı bırakıp tekrardan Ekle tuşuna basınız.
7. 6ncı adımdan sonra **"ErrorCode<ERRCA0010>:Cache::Add: An attempt is being made to create a object with a Key that already exists in the cache.Cache will only accept unique key value for objects."** içerikli bir hata mesajı aldığınızdan emin olunuz*(Nitekim tekrar kaydetmek istediğiniz Key değeri zaten mevcuttur)*
8. Uygulamayı kapatınız ve bir kaç dakika sonra*(tam olarak 300 saniye=5 dakika)* tekrar başlatıp Getir tuşuna basınız.
9. 8nci adımdan sonra Cache içeriğinin TextBox' a gelmediğinden emin olunuz.

Böylece geldik bir Hello World uygulamamızın daha sonuna. Bu konuda yeni bilgiler edindikçe sizlere paylaşmaya devam ediyor olacağım. Nitekim mimarisine baktığımızda son derece derin bir konu olduğunu farketmiş olmalısınız. özellikle şu sıralar bir kaç Cluster Server' a kurulum yapıp Load Balancing ile ilgili testleri nasıl yapabileceğimi düşünmekteyim. Tabi bunun için öncesinden Environment' in tesis edilmesi gerekiyor. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Detaylı bilgi için tıklayın.
CTP3 sürümünü Download etmek için tıklayın.

HelloVelocity.rar (285,93 kb)

WCF RIA Services - Authentication Domain Service (2009-12-10T13:40:00)

wcf ria services,.net ria services,wcf,wcf eco system,

Merhaba Arkadaşlar,

Bazen insanın yapmak zorunda olduğu bazı şeyler gözünde büyür. örneğin havalimanlarında kontrollerden geçerek uçağa ulaşmak o daracı rahatsız koltuklara sığmak için çabalamak. önce ilk kapıda bir güvenlik kontrollünden geçilir, ardından Check-In işlemi için kimlikle birlikte bir kontrolden daha geçilir*(hatta bagajımız var ise tartılır ve gerekiyorsa ekstra para ödenir),* ardından uçağa bineceğimiz kapılara gitmek için bir kontrolden daha geçilir, ardından uçağa binerken bilet ve kimlik ile kontrolden bir kere daha geçilir. Keşke insanoğlu daha barışçıl olsaymış dedirten aramalar ile karşılaşılır zaman içerisinde. Güvenlik kontrollerinin temel amaçlarından birisi de, gerçekten sizin kimliğinizde söylenen kişi olduğunuzu görmek ve hatta elinizdeki biletiniz ile söz konusu uçağa binebilecek yetkiye sahip olduğunu öğrenmektir. Tabi arada sıra **Administrator** seviyesinde pek çok insan güvenlik kontrollerine takılmadan protokol kapısından giriş yaparak yollarına devam edebilirler. üstelik yürüdükleri hat üzerinde kırmızı halılarda bulunabilir. 😉 özellikle bu son durumda, giriş yapan kişilerin rollerininde büyük önemi vardır.

Sözün özü bir noktaya güvenlik kontrolünden geçerek girmemiz gerektiğinde doğrulanma, yetki kontrolü, rol gibi faktörlerle karşılaşırız. Zaten yazılım dünyasında da**üyelik tabanlı(Membership Based)** olarak çalışan sistemlerde, kullanıcıların bazı şeyleri yapabilmesi için önce **doğrulanmaları(Authenticate)** gerekir. Buna ilaveten, doğrulanan kullanıcıların yetkilerine bakaraktan bir takım işlemleri yapıp yapmamalarına **izin verilmesi(Authorization),** hatta bu sırada rollerinin de değerlendirilmesi söz konusudur. Dahası doğrulanan ve yetkisi dahilinde bir yere ulaşan bireyin kendine has profilini tanımlayan özellikleride bulunabilir.

Web tabanlı uygulamalarda sıkça karşılaştığımız **doğrulama(Authentication)** işlemlerinin genellikle **Form-Based** veya **Windows-Based** olarak yapılabildiğini görürüz.*(Hatta Microsoft Passport hizmetininde kullanılması mümkündür)* **Asp.Net 2.0** ile birlikte getirilen **Membership** alt yapısı sayesinde **SQL** üzerinde tutulabilen hazır üyelik sistemlerinden kolaylıkla yararlanabiliriz. Hatta ilk Asp.Net sürümünden bu yana, **Active Directory** sayesinde intranet tabanlı web uygulamalarında **Windows Domain** kullanıcılarını ve rollerini değerlendirebiliriz. Temel amaç aslında son derece açıktır; **Gerçekten tanınan(Authenticated User)** kullanıcıların belirli işlemleri

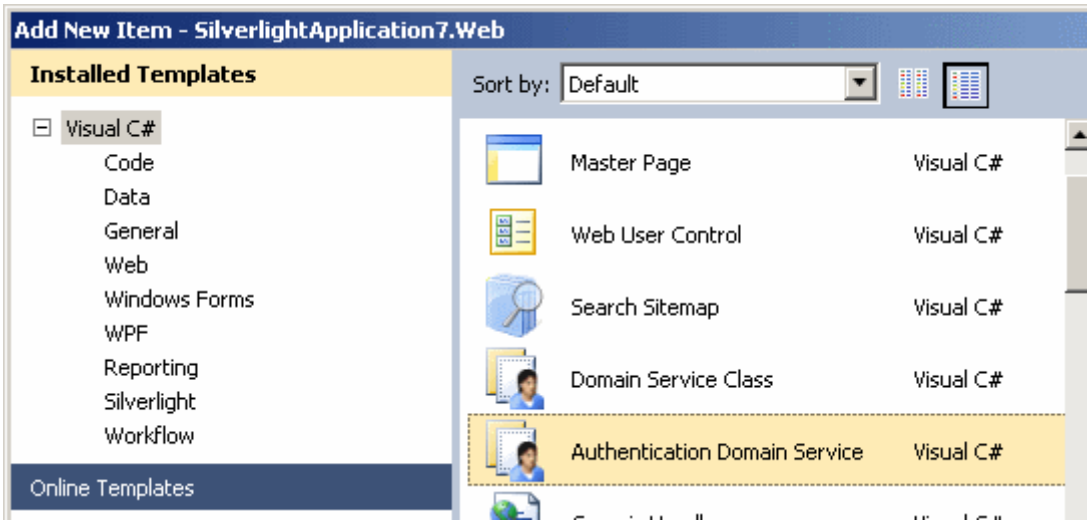
yapabilmesi ve neler yapabileceklerine karar verilirken **yetkilerine(Authorization)** yada rollerine bakılabilmesi.

Sözü fazla uzatmadan **WCF RIA Services** tarafındaki duruma bakalım. Sonuç itibariyle **RIA(Rich Internet Application)** için sıklıkla değerlendirdiğimiz **Silverlight** uygulamalarınında,**Asp.Net Membership** veya **Windows Domain** yapılarından yararlanarak doğrulama, yetkilendirme ve rollendirme işlemlerini yapılabilmesi mümkündür. Bu amaçla **WCF RIA Services'** ler ile birlikte gelen **Authentication Domain Service** tipinden yararlanılmaktadır. Bu yazımızdaki amacımız da, çok basit anlamda bir **RIA** uygulamasında doğrulama işlemlerinin nasıl ele alınabileceğini incelemektir.

İşe ilk olarak **Silverlight 4.0** ve **WCR RIA Services** destekli bir **Sliverlight** uygulaması oluşturarak başlamalıyız. Bilindiği üzere **Silverlight** çözümlerinde **Web** tabanlı bir sunucu ve**Silverlight** kontrollerini barındıran bir uygulama söz konusudur. **Silverlight** tarafında geliştirilen kullanıcı kontrollerinin host ediliği **Web** uygulaması **Asp.Net** tabanlı olduğundan pekala bir**Membership** sistemine sahip olabilir. Bu nedenle sunucu uygulama üzerinde **Membership API'** sini kullanarak işe başlamakta ve **Form-Based Authentication'** ı kullanacağımızı belirtmekte yarar vardır. Bildiğiniz üzere **Membership** işlemleri için, **Web** projesinin **Asp.Net Configuration** aracı kullanılabilir. Ben örneğimizde **SQL Express Edition** üzerinde konuşlandırılan**Membership** veritabanını kullanmayı tercih ettim ve şifreleri **123456.** olan iki kullanıcı oluşturdum**(buraks ve bill). Form-Based Authentication** kullandığımız için **web.config** dosyasında yer alan **authentication** elementinin içeriğinin aşağıdaki gibi oluşturulması gerekmektedir*(Tabi Asp.Net Configuration aracını kullanarak From Internet seçeneğini işaretlediğimizde bu ayar otomatik olarak web.config içerisine işlenecektir)*

```
...
<system.web>
      <httpModules>
         <add name="DomainServiceModule"
type="System.Web.Ria.Services.DomainServiceHttpModule, System.Web.Ria,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
      </httpModules>
      <!-- Form tabanlı doğrulama(Form-Based Authentication) kullanılacağı belirtilir.-->
      <authentication mode="Forms" />
      <compilation debug="true" targetFramework="4.0" />
   </system.web>
...
```

Bu işlemin ardından yine sunucu uygulamaya **Authentication Domain Service** tipinden bir sınıf eklenmesi gerekmektedir.



örnekte **OurAuthenticationService** isimiyle eklediğimiz dosyanın içeriği otomatik olarak oluşturulacak ve aşağıdaki kod parçasında görüldüğü gibi olacaktır.

```
using System.Web.Ria;
using System.Web.Ria.ApplicationServices;

namespace SilverlightApplication7.Web
{
   [EnableClientAccess]
   public class OurAuthenticationService : AuthenticationBase<User>
   {
      // To enable Forms/Windows Authentication for the Web Application,
      // edit the appropriate section of web.config file.
   }

   public class User : UserBase
   {
      // NOTE: Profile properties can be added here
      // To enable profiles, edit the appropriate section of web.config file.

      // public string MyProfileProperty { get; set; }
   }
}
```

Burada yer alan **AuthenticationBase<User>** türevli sınıf doğrulama işlemleri sırasında istemci tarafından kullanılacak tiptir. Diğer yandan **User** isimli tip üyelere has profil özelliklerinin uygulanabilmesi için kullanılmaktadır. Söz gelimi, **Membership API** üzerinde tanımlanmış ve yetkisi olan bir kullanıcının **Silverlight** uygulamasını kullanmaya başlaması ile birlikte**Title, Birthdate** gibi geliştirici tanımlı bir takım bilgilerinin profil özelliği olarak kullanılabilmesi mümkündür. Tabi **User** tipi içerisinde profil özelliklerinin tanımlanması yeterli değildir.**Web.config** dosyasında da aynı profil özelliklerinin bildirilmesi ve profil yönetiminin etkinleştirilmesi gerekmektedir. *(Bu*

yazımızda geliştireceğimiz örnekte doğrulama işlemlerini çok basit bir seviyede anlamak istediğimizden profil özellikleri ile uğraşılmayacaktır). Yapılan ekleme işleminden sonra projenin **Build** edilmesi halinde istemci tarafında **WebContext**, **User** ve**OurAuthenticationContext** isimli tiplerin oluşturulduğu gözlemlenir. **User** sınıfına ait nesne örneği ile tahmin edileceği üzere sunucu tarafında tanımlanan **User** tipine erişilebilmesi mümkündür. Diğer taraftan **WebContext** tipi, istemci tarafından **Authentication Domain Service'** e erişilebilmesini ve **User** bilgisinin alınabilmesini sağlamaktadır. Buraya kadar ki kısımda sunucu tarafı için gerekli aşağıdaki işlemleri yaptığımızı söyleyebiliriz;

- Form Tabanlı doğrulama için Membership veritabanının oluşturulması ve buna bağlı olarak web.config dosyasında gerekli ayarların yapılması,
- Gerekiyorsa role seçeneğinin etkinleştirilmesi,
- Authentication Domain Service tipinin sunucu projeye eklenmesi,
- Gerekiyorsa User tipi için profil özelliklerinin kod ve web.config bazında oluşturulması

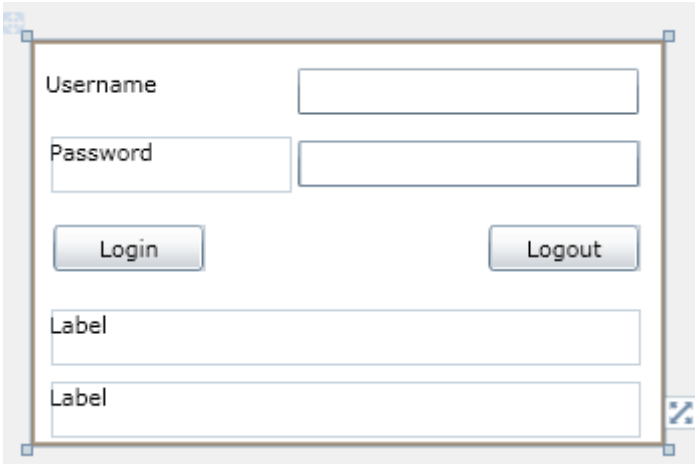
Ancak istemci tarafında, sunucu üzerindeki **Authentication Domain Service** hizmetinin kullanılabilmesi için bir takım işlemler yapılmalıdır. öncelikli olarak **App.Xaml** içeriğinin aşağıdaki hale getirilerek **Web.Context** ve **Form** tabanlı doğrulamanın kullanılacağının belirtilmesi gerekir*(Buradaki kodlamaların aslında ilerleyen sürümlerde otomatik hale getirileceğini ümit etmekteyim)*

```
<Application xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:app="clr-namespace:SilverlightApplication7"
             xmlns:appSrv="clr-
namespace:System.Windows.Ria.ApplicationServices;assembly=System.Windows.Ria"
             x:Class="SilverlightApplication7.App"
             >
    <Application.Resources>

    </Application.Resources>
    <!-- Aşağıdaki kısım Form-Based authentication kullanımı için eklenmelidir-->
    <Application.ApplicationLifetimeObjects>
        <app:WebContext>
            <app:WebContext.Authentication>
                <appSrv:FormsAuthentication />
            </app:WebContext.Authentication>
        </app:WebContext>
    </Application.ApplicationLifetimeObjects>

</Application>
```

App.Xaml içerisinde yer alan tanımlamalar da yeterli değildir. **WebContext'** in **XAML** içerisinde kullanılabilir olması için **App.Xaml.cs** kod tarafında aşağıdaki ilavenin yapılması gerekmektedir.

```
private void Application_Startup(object sender, StartupEventArgs e)
        {
            // XAML içerisinde WebContext nesnesinin kullanılabilmesi için Resources koleksiyonuna ilgili WebContext nesne örneğinin eklenmesi gerekir.
            this.Resources.Add("WebContext", WebContext.Current);
            this.RootVisual = new MainPage();
        }
```

Artık **Silverlight** kontrolünün içeriğinin tasarlanmasına başlanabilir. örneğimizde **Login** ve **Logout** işlemlerinin yapılması irdelenecektir. Buna göre aşağıdaki tasarım görüntüsü ve **XAML** içeriğine sahip olacak şekilde **MainPage.xaml** dosyasını düzenleyerek ilerlediğimizi düşünelim.



XAML İçeriği;

```
<UserControl x:Class="SilverlightApplication7.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="202" d:DesignWidth="316" xmlns:dataInput="clr-namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.Input">

    <Grid x:Name="LayoutRoot" Background="White">
        <dataInput:Label Height="28" HorizontalAlignment="Left" Margin="8,14,0,0" Name="label1" VerticalAlignment="Top" Width="120" Content="Username" />
        <dataInput:Label Height="28" HorizontalAlignment="Left" Margin="10,48,0,0" Name="label2" VerticalAlignment="Top" Width="120" Content="Password" />
```

```
        <TextBox Height="23" HorizontalAlignment="Left" Margin="134,14,0,0"
Name="txtUsername" VerticalAlignment="Top" Width="170" />
        <PasswordBox Height="23" HorizontalAlignment="Left" Margin="134,50,0,0"
Name="txtPassword" VerticalAlignment="Top" Width="170"/>
        <Button Content="Login" Height="23" HorizontalAlignment="Left"
Margin="12,92,0,0" Name="btnLogin" VerticalAlignment="Top" Width="75"
Click="btnLogin_Click" />
        <dataInput:Label Height="28" HorizontalAlignment="Left" Margin="10,134,0,0"
Name="lblLoginStatus" VerticalAlignment="Top" Width="294" />
        <Button Content="Logout" Height="23" HorizontalAlignment="Left"
Margin="229,92,0,0" Name="btnLogout" VerticalAlignment="Top" Width="75"
Click="btnLogout_Click" />
        <dataInput:Label Height="28" HorizontalAlignment="Left" Margin="10,170,0,0"
Name="lblProcess" VerticalAlignment="Top" Width="294" />
    </Grid>
</UserControl>
```

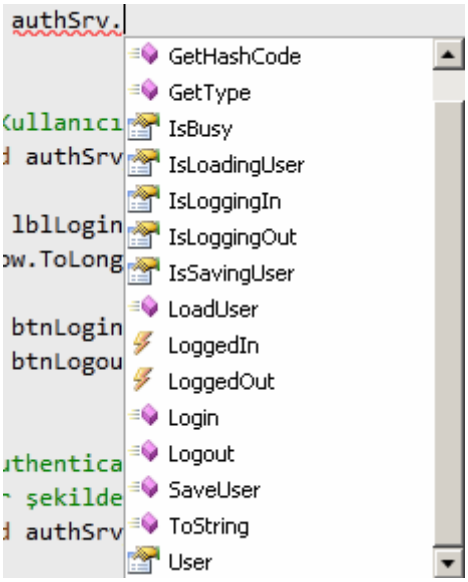
Gelelim kod kısmına;

```
using System;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Ria.ApplicationServices;

namespace SilverlightApplication7
{
    public partial class MainPage : UserControl
    {
        // Kullanıcının doğrulanması, profil özelliklerinin yüklenmesi ve kaydedilmesi için
gerekli fonksiyonellikleri sunan tiptir.
        AuthenticationService authSrv = WebContext.Current.Authentication;

        public MainPage()
        {
            InitializeComponent();

            btnLogout.IsEnabled = false;

            authSrv.LoggedIn += new
System.EventHandler<AuthenticationEventArgs>(authSrv_LoggedIn);
            authSrv.LoggedOut += new
System.EventHandler<AuthenticationEventArgs>(authSrv_LoggedOut);
        }
```

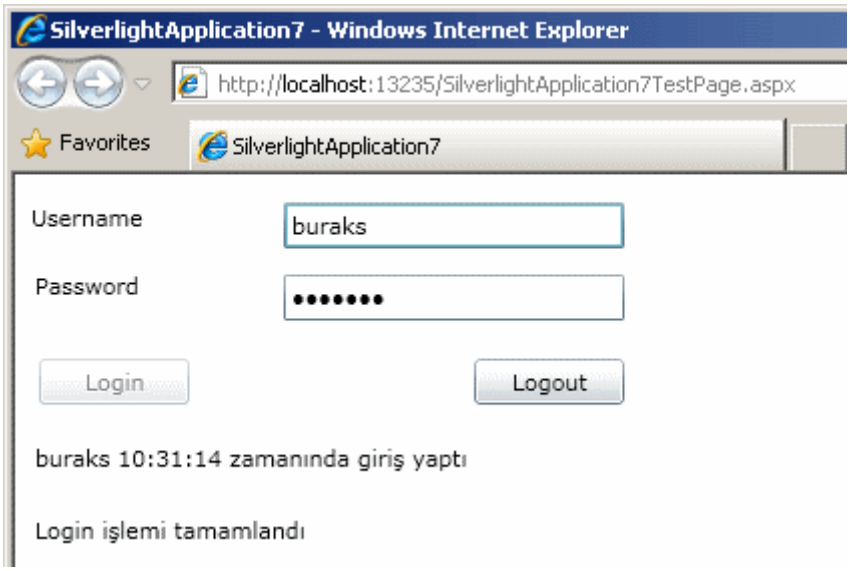
```
private void btnLogin_Click(object sender, RoutedEventArgs e)
{
   lblLoginStatus.Content = String.Empty;
   lblProcess.Content = String.Empty;

   if (!authSrv.IsLoggingIn) // Eğer asenkron olarak devam eden bir Login operasyonu yoksa
   {
      // Login işlemi için AuthenticationService tipinin Login metodu çağırılır.
      // ilk parametre ile kullanıcı adı ve şifre bilgisi gönderilir. Bu metodun aşırı yüklenmiş versiyonları mevcuttur.
      // ikinci parametrede Login metodunun işleyişini tamamlaması sonrası devreye giren metodun işaret edilmesi sağlanmaktadır(Action<LoginOperation> temsilci tipi ile işaretleme yapılır). Bu metod içerisinde işlemin iptali, exception üretmesi gibi durumlarda ele alınmaktadır.
      authSrv.Login(
         new LoginParameters(txtUsername.Text, txtPassword.Password),
         opt =>
         {
            if (opt.IsCanceled)
               lblProcess.Content = "Login işleminde iptal";
            else if (opt.Error != null)
               lblProcess.Content = opt.Error.Message;
            else if (opt.IsComplete)
               lblProcess.Content = "Login işlemi tamamlandı";
         }
         , null
         );
   }
}

private void btnLogout_Click(object sender, RoutedEventArgs e)
{
   lblLoginStatus.Content = String.Empty;
   lblProcess.Content = String.Empty;

   if (!authSrv.IsLoggingOut) // Eğer asenkron olarak devam eden bir Logout operasyonu yoksa
   {
      // Logout işlemi kullanılan metod çağrısı
      // işlem tamamlandığında devreye girecek olan metod Action<LogoutOperation> temsilcisi ile işaret edilir.
      authSrv.Logout(
        opt =>
        {
```

```
                if (opt.IsCanceled)
                    lblProcess.Content = "Logout işleminde iptal";
                else if (opt.Error != null)
                    lblProcess.Content = opt.Error.Message;
                else if (opt.IsComplete)
                    lblProcess.Content = "Logout işlemi tamamlandı";
                }
            , null);
        }

    }

    // Kullanıcı başarılı bir şekilde Logout olduğunda tetiklenir
    void authSrv_LoggedOut(object sender, AuthenticationEventArgs e)
    {
        lblLoginStatus.Content = String.Format("{0} {1} zamanında çıkış yaptı",
e.User.Identity.Name, DateTime.Now.ToLongTimeString());

        btnLogin.IsEnabled = true;
        btnLogout.IsEnabled = false;
    }

    //AuthenticationService nesnesine ait Login metodunun çalıştırılması sonrasında
kullanıcı başarılı bir şekilde doğrulandıysa çalışır
    void authSrv_LoggedIn(object sender, AuthenticationEventArgs e)
    {
        lblLoginStatus.Content = String.Format("{0} {1} zamanında giriş yaptı",
e.User.Identity.Name, DateTime.Now.ToLongTimeString());

        btnLogin.IsEnabled = false;
        btnLogout.IsEnabled = true;
    }
  }
}
```

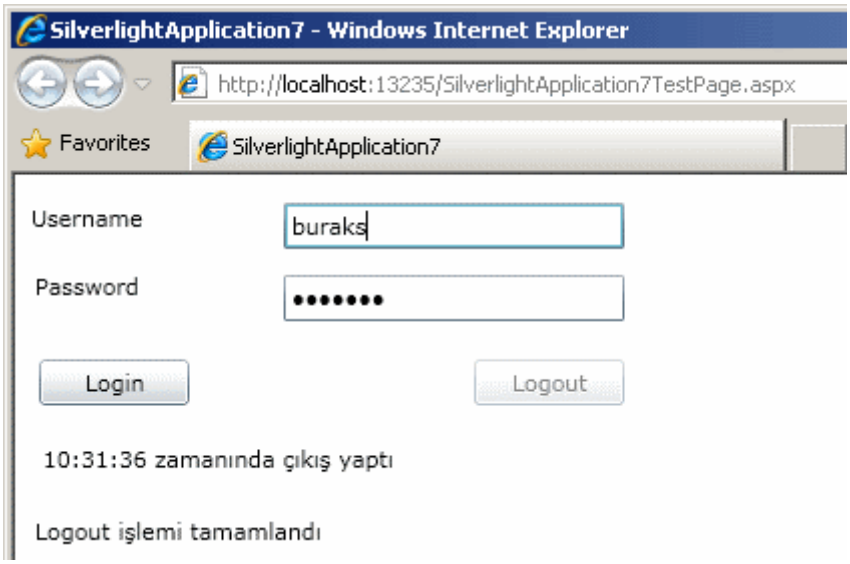
Aslında bütün iş yükünü **AuthenticationService** nesne örneği almaktadır. **Login** ve **Logout** operasyonları için kullanılan bu nesne örneği üzerinden profil ve rol yönetimi için gerekli pek çok operasyonda gerçekleştirilebilir. örneğin doğrulanan kullanıcı bilgilerine ulaşılabilir(User), kullanıcı bilgilerinin(özellikle profil bilgileri) değiştirilmesi, kayıt edilmesi veya yüklenmesi sağlanabilir. Aşağıdaki şekilde kullanılabilecek özellik ve üye metodlar görülmektedir.



Uygulama açıldıktan sonra geçerli bir kullanıcı adı ve şifre ile **Login** işlemini yapmak istediğimizde aşağıdaki sonuçları elde ettiğimizi görebiliriz.



Dikkat edileceği üzere kullanıcı başarılı bir şekilde tanınmıştır. Login olan kullanıcının Logout olması için gerekli işlemi yaptığımızda ise aşağıdaki sonuç ile karşılaşırız.



çok doğal olarak geçersiz bir kullanıcı adı veya şifre girişinde Login işleminin gerçekleştirilemediği görülecektir. Elbette bu yazımızdaki senaryomuzda sembolik olarak Login ve Logout operasyonlarının işlevsellikleri ön plana çıkartılmıştır. Dolayısıyla Login olunduktan sonra asıl işlemlerin yapılacağı sayfaya yönlenilmesi ve hatta rol kontrolüne göre ilerlenmesi işlemleri göz ardı edilmiştir. Oysaki gerçek hayat projelerinden bu işlemlerin ele alınması şarttır. Bu tip konuları belki bir görsel ders üzerinden incelemeye çalışabiliriz. Buraya kadar yaptıklarımıza baktığımızda, istemci tarafında aşağıdaki temel işlemleri yaptığımızı özetleyebiliriz;

- App.Xaml ve App.Xaml.cs içerisinden Form tabanlı doğrulama servisinin kullanılması için gerekli ayarlar,
- Kod tarafında, AuthenticationService nesne örneğinin WebContext.Current ile elde edilmesi ve gerekli kodlamalar,

Görüldüğü üzere **WCF RIA Service**' ler ile birlikte gelen **Authentication Domain Service** tipini kullanarak, **Silverlight** formatlı **RIA** uygulamalarının, **Asp.Net Web** uygulamalarında sıklıkla ele alınan **doğrulama(Authentication), yetkilendirme(Authorization), rol(Role)** ve **profil(Profile)** yönetimi kabiliyetlerine sahip olması sağlanabilmektedir. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

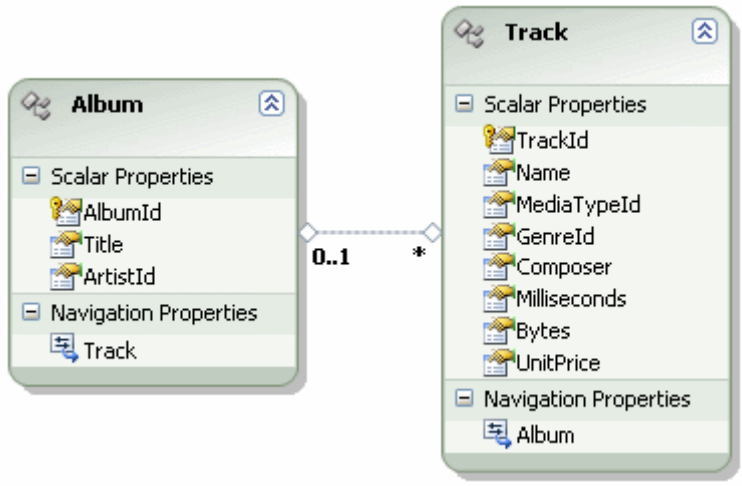
SilverlightApplication7.rar (626,55 kb) *[Dosya Boyutunun küçük olması amacıyla, Memberhip için kullanılan ASPNET_DB veritabanı silinmiştir]*

Ado.Net Entity Framework 4.0 - Lazy Loading [Beta 2] (2009-12-07T11:27:00)

ado.net entity framework,

Merhaba Arkadaşlar,

Ado.Net Entity Framework' ün en çok eleştirilen yönlerinden birisi, **ORM(Object Relational Mapping)** ihtiyacını karşılayacak bir alt yapı olarak tasarlanmasına rağmen **ORM'** in karakteristik özelliklerinden birisi olan **Lazy Loading'** in*(Bir nesnenin ihtiyaç duyulduğu noktada veri kaynağından yüklenmesini-Load hedefleyen bir tasarım kalıbı olarak düşünülebilir)* tam manada desteklemiyor olmasıdır. Zaman içerisinde **Deferred Loading** veya **Explicity Lazy Loading** gibi isimler ile **Ado.Net Entity Framework** içerisine bir takım özellikler eklenerek bu eksiklik ortadan kaldırılmaya çalışılsada gerçek manada **4.0** versiyonunun **Beta 2** sürümünde, beklenildiği gibi bir iyileştirme olduğunu görmekteyiz. Bu yazımızdaki temel amacımız ise, **Ado.Net Entity Framework'** ün bir önceki sürümünde durumun ne olduğunu anlamaya çalışıp, **4.0 Beta 2** sürümünde **Lazy Loading** adına hangi özelliğin getirildiğini(belkide getirilmediğini) ve nasıl kullanıldığını görebilmektir. örneklerimizde **Codeplex** üzerinden açık kaynak olarak yayınlanan **Chinook** veritabanını kullanıyor olacağız. Haydi parmakları sıvayalım örneklerimizden ilkini **Visual Studio 2008** diğerini ise **Visual Studio 2010 Ultimate Beta 2** ortamında geliştiriyor olacağız. Ancak her iki örnekte aşağıdaki şekilde görülen tabloları kullanıyor olacak.



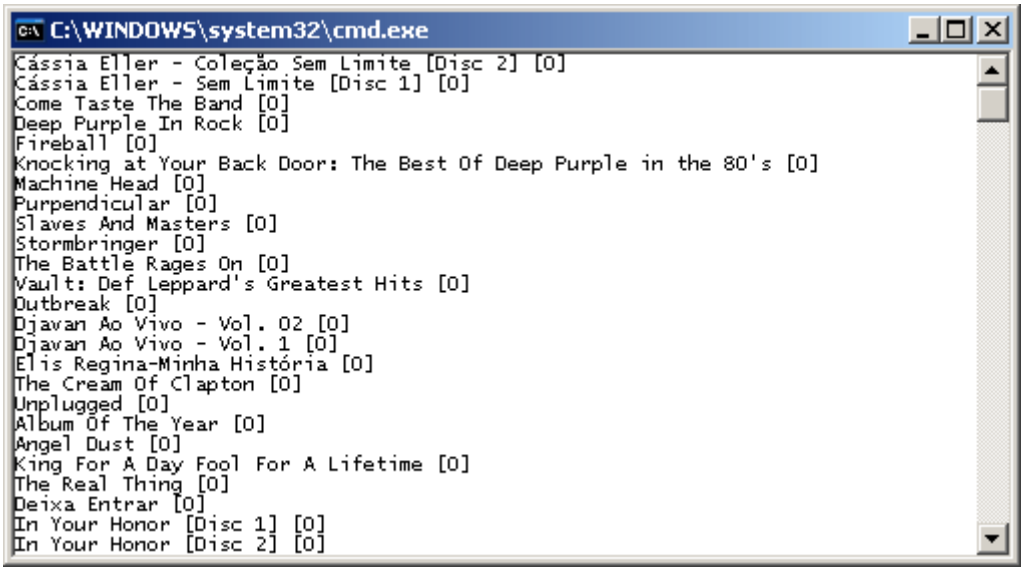
Visual Studio 2008 ve Ado.Net Entity Framework V1.0

İlk olarak **Console** uygulamamızda aşağıdaki gibi bir kod parçası geliştirdiğimizi düşüelim.

```
using System;
using System.Linq;

namespace Before
{
   class Program
   {
      static void Main(string[] args)
      {
         using (ChinookEntities context = new ChinookEntities())
         {
```

```
            var albums = from a in context.Album
                         select a;

            foreach (Album albm in albums)
               Console.WriteLine("{0} [{1}]",albm.Title,albm.Track.Count.ToString());

        }
    }
  }
}
```

Bu kod parçasında Album listesi elde edildikten sonra Title bilgileri ile birlikte, o anki albüme ait Track sayıları ekrana yazdırılmaktadır. Yani albümler ve bu albümler altında kaç şarkı olduğu bilgisine ulaşmak istediğimizi düşünebilir. Buna göre çalışma zamanı görüntüsü aşağıdaki gibidir.



Hatta **SQL Server Profiler** aracına bakıldığında söz konusu kod parçası için aşağıdaki sorgunun çalıştırıldığı görülür.

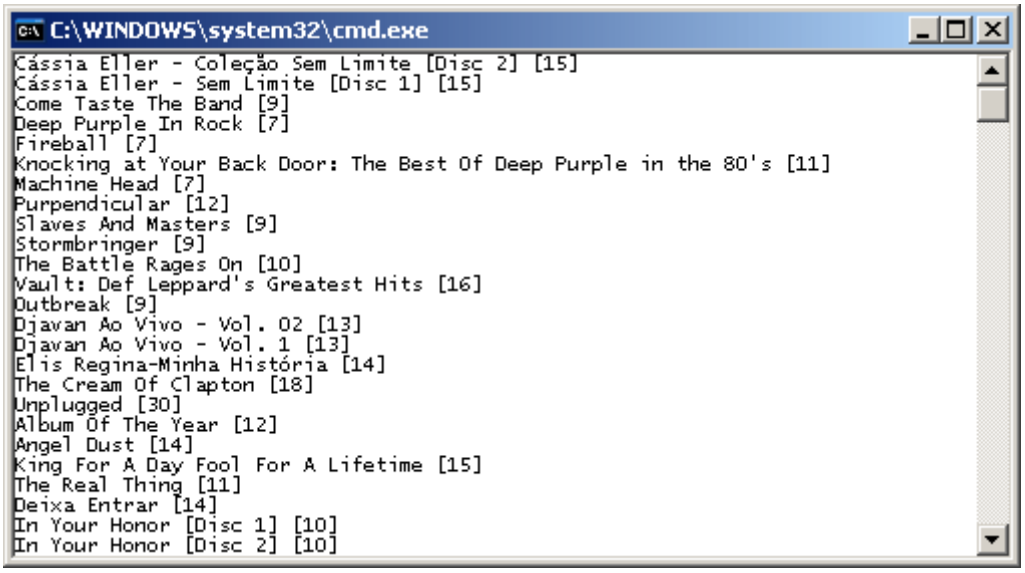
```
SELECT
[Extent1].[AlbumId] AS [AlbumId],
[Extent1].[Title] AS [Title],
[Extent1].[ArtistId] AS [ArtistId]
FROM [dbo].[Album] AS [Extent1]
```

Bir sorun var mı? 😕 Aslında var. Dikkat edileceği üzere güncel **Album Entity** nesnesi ile ilişkili olan **Track** nesneslerinin toplam sayıları(Count) her zaman 0 olarak elde edilmiştir. Bir başka deyişle X Entity nesnesi ile ilişkili olan Y Entity nesnesine dair herhangibir sorgunun işletilmesi söz konusu olmamıştır. Oysaki **Lazy Loading** kalıbına göre çalışma zamanındaki **Album**nesne örneklerine ait **Track** özellikleri üzerinden o anki **Track** nesne örneğinin herhangibir üyesine erişilmek istendiğinde*(örnekteki Count gibi)*, SQL

tarafındada gerekli sorguların çalıştırılacağı düşünülür. Peki ilk versiyonda yaşanan bu durum üzerine ne yapılmaktaydı? Temel olarak iki basit yöntem ile bu durumu çözüme kavuşturabiliriz. Bunlardan birisi **Include**metodunun **LINQ** ifadesinde aşağıdaki gibi kullanılmasıdır.

```
var albums = from a in context.Album.Include("Track")
                    select a;
```

Bu durumda çalışma zamanı görüntüsü aşağıdaki gibi olacaktır.



Görüldüğü üzere albümler içerisinde yer alan parçaların toplam sayıları elde edilebilmiştir. Bunun için SQL tarafında da aşağıdaki sorgunun çalıştırıldığı gözlemlenmektedir.

```
SELECT
[Project1].[AlbumId] AS [AlbumId],
[Project1].[Title] AS [Title],
[Project1].[ArtistId] AS [ArtistId],
[Project1].[C1] AS [C1],
[Project1].[C3] AS [C2],
[Project1].[C2] AS [C3],
[Project1].[TrackId] AS [TrackId],
[Project1].[Name] AS [Name],
[Project1].[MediaTypeId] AS [MediaTypeId],
[Project1].[GenreId] AS [GenreId],
[Project1].[Composer] AS [Composer],
[Project1].[Milliseconds] AS [Milliseconds],
[Project1].[Bytes] AS [Bytes],
[Project1].[UnitPrice] AS [UnitPrice],
[Project1].[AlbumId1] AS [AlbumId1]
FROM ( SELECT
```

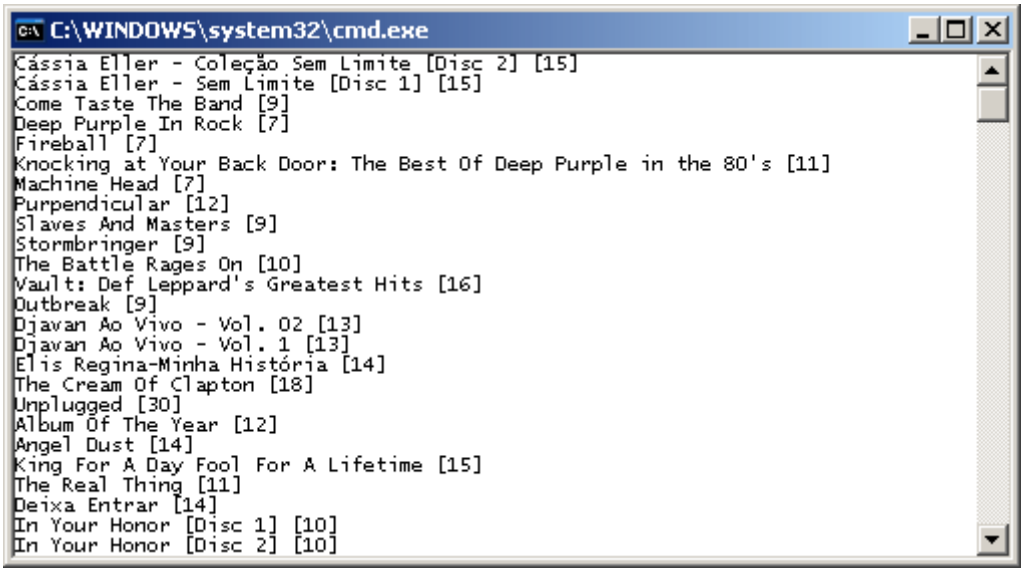
```
 [Extent1].[AlbumId] AS [AlbumId],
 [Extent1].[Title] AS [Title],
 [Extent1].[ArtistId] AS [ArtistId],
 1 AS [C1],
 [Extent2].[TrackId] AS [TrackId],
 [Extent2].[Name] AS [Name],
 [Extent2].[AlbumId] AS [AlbumId1],
 [Extent2].[MediaTypeId] AS [MediaTypeId],
 [Extent2].[GenreId] AS [GenreId],
 [Extent2].[Composer] AS [Composer],
 [Extent2].[Milliseconds] AS [Milliseconds],
 [Extent2].[Bytes] AS [Bytes],
 [Extent2].[UnitPrice] AS [UnitPrice],
 CASE WHEN ([Extent2].[TrackId] IS NULL) THEN CAST(NULL AS int) ELSE 1 END
AS [C2],
 CASE WHEN ([Extent2].[TrackId] IS NULL) THEN CAST(NULL AS int) ELSE 1 END
AS [C3]
 FROM  [dbo].[Album] AS [Extent1]
 LEFT OUTER JOIN [dbo].[Track] AS [Extent2] ON [Extent1].[AlbumId] =
[Extent2].[AlbumId]
)  AS [Project1]
ORDER BY [Project1].[AlbumId] ASC, [Project1].[C3] ASC
```

Dikkat edileceği üzere **Album** ve **Track** tablolarının **Left Outer Join** ile birleştirilmesi söz konusudur. Dahası, sonucun elde edilmesi için tek bir SQL ifadesinin çalıştırılması yeterli olmuştur. Ancak dikkat çeken noktalardan biriside sadece **Count** değeri ile ilgilenmemize rağmen **Track** tablosundaki tüm alanların ifadeye dahil edilmesidir. Zaten **Count** hesabı için SQL tarafında çalıştırılmış bir ifade de bulunmamaktadır. Bunun yerine kod tarafına çekilen **Track** listesinin toplam değerinin hesaplanması söz konusudur.

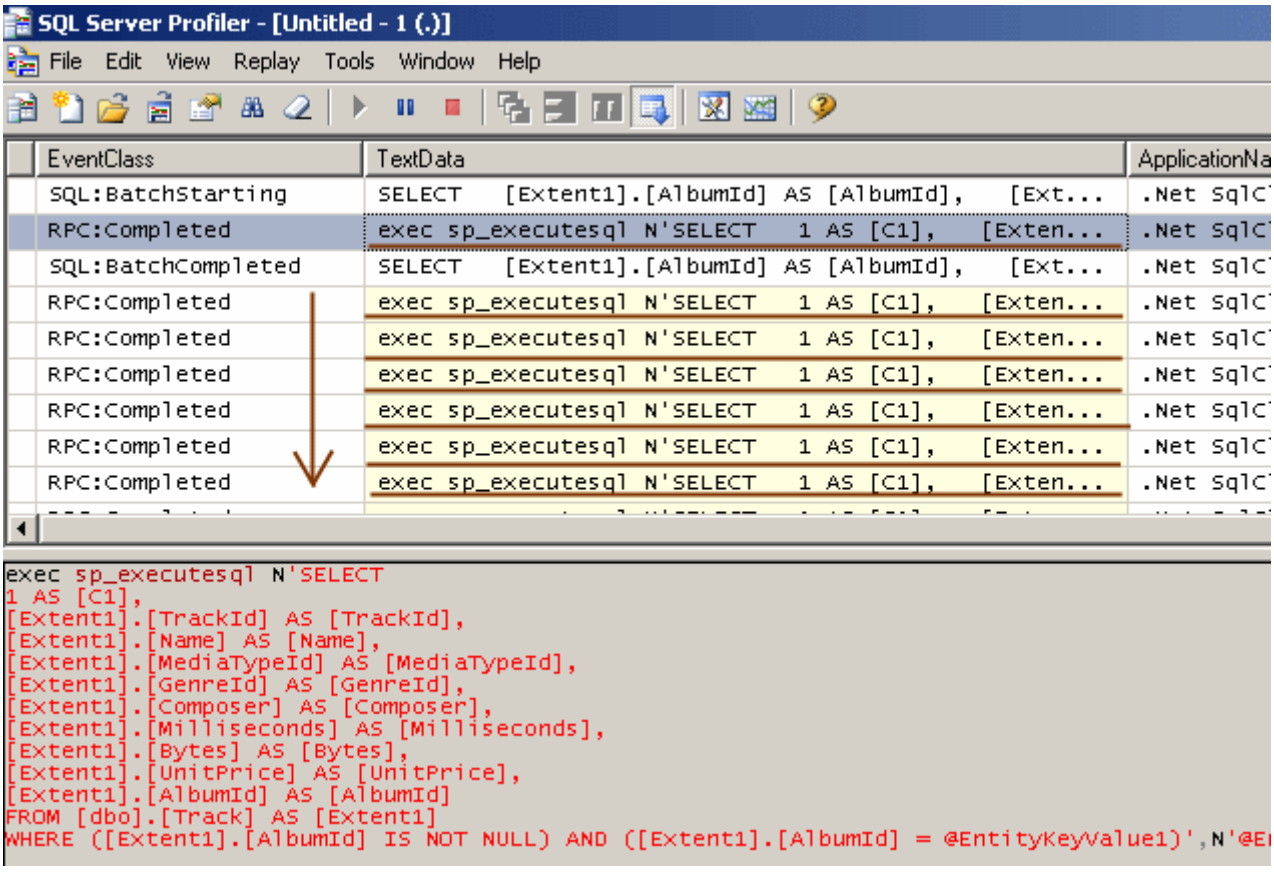
Diğer bir teknik ise **Load** metodunun kullanılmasıdır. Bu amaçla foreach döngüsünde aşağıdaki düzenlemeyi yaptığımızı düşünelim.

```
foreach (Album albm in albums)
{
  albm.Track.Load();
  Console.WriteLine("{0} [{1}]", albm.Title, albm.Track.Count.ToString());
}
```

Bu durumda çalışma zamanı görüntüsü **Include** kullanımındaki ile aynı olacaktır.



SQL Server Profiler aracına bakıldığında ise aşağıdakine benzer bir ekran görüntüsü ile karşılaşılacaktır.



Dikkat edileceği üzere **foreach** döngüsü içerisinde **Load** metodunun çağırıldığı her an arka planda bir **SQL** sorgusunun çalıştırıldığı, yine sadece **Count** ile ilgilenmemize rağmen tüm **Track** alanlarının işe katıldığı gözlemlenebilir. Hatta **Count** ile ilişkili olarak **SQL** tarafında çalıştırılan bir ifade bulunmamaktadır. Bu değer, kod tarafındaki**Track** listesinin ilgili **Album** için elde edilmesinden sonra hesaplanmaktadır. Aslında tam bu noktada, ele alınan senaryo

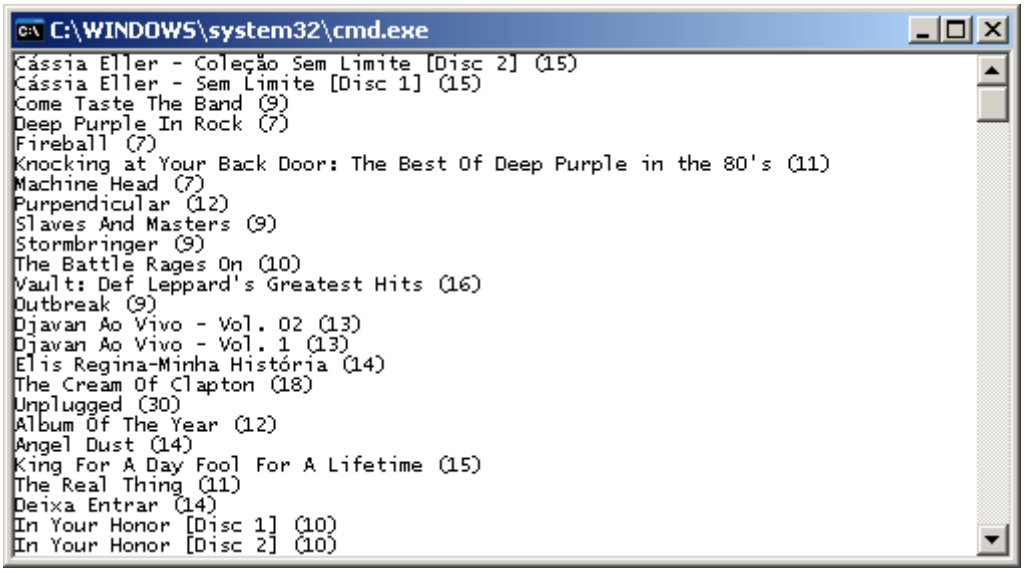
için **Include** metodunun kullanımının, **Load** metoduna göre daha efektif olduğunu düşünebiliriz. Nitekim istemci ve SQL sunucusu arasında sadece tek bir sorgusunun çalıştırılması söz konusudur. Diğer yandan **foreach** döngüsü içerisinde senaryoya göre bazı koşullar sağlandığı takdirde **Count** özelliğinin kullanılması istendiği durumlarda, **Load** metodunun tercih edilmesi yoluna gidilebilir. Her neyse...Bakalım **Ado.Net Entity Framework 4.0 Beta 2** içerisinde gerçek anlamda **Lazy Loading** için ne yapılmıştır.

Visual Studio 2010 Beta 2 ve Ado.Net Entity Framework 4.0 Beta 2

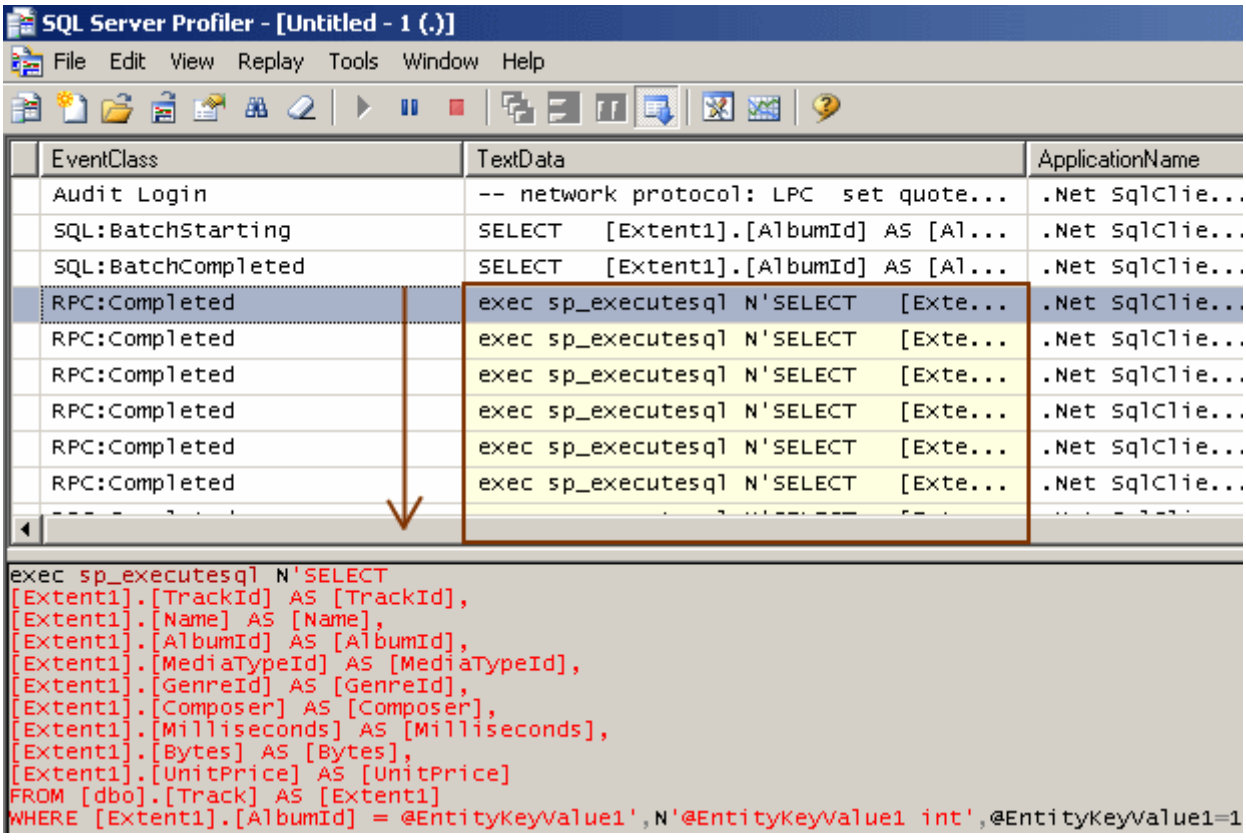
Yeni sürümde **ObjectContext** türevli tip üzerinden uygulanabilen **LazyLoadingEnabled** isimli bir **özellik(Property)** bulunmaktadır. Aslında varsayılan olarak **Lazy Laoding** özelliğinin açık olduğunu söyleyebiliriz. Aynı örneği **Visual Studio 2010 Beta 2** üzerinde denediğimizi düşünelim.*(Entity adlarının oluşturulması sırasında çoğullaştırma özelliği etkinleştirilmiştir. Bu nedenle Albums ve Tracks isimlendirmeleri söz konusudur)*

```
using System;
using System.Linq;

namespace ReallyLazy
{
   class Program
   {
      static void Main(string[] args)
      {
         using (ChinookEntities context = new ChinookEntities())
         {
            var albumsWithTracks = from a in context.Albums
                                   select a;

            foreach (Album albm in albumsWithTracks)
            {
               Console.WriteLine("{0} ({1})",albm.Title,albm.Tracks.Count.ToString());
            }
         }
      }
   }
}
```

çalışma zamanında aşağıdaki sonuçları elde ederiz.



Dikkat edileceği üzere kod içerisinde herhangibir şey belirtmememize rağmen **Album** nesneleri ile ilişkili olan **Track** nesnelerinin **Count** özelliklerinin değerleri elde edilebilmiştir. Peki arka planda çalışan SQL sorgusu(sorguları)? İşte **SQL Server Profiler** aracından elde edilen sonuçlar;



Görüldüğü gibi **foreach** döngüsü içerisinde **Track.Count** özelliğine gidildiği her noktada bir SQL sorgusu çalıştırılmış ve o anki albüme bağlı olan parça bilgileri çekilmiştir. Dikkat edileceği üzere **Count** için SQL tarafında yapılmış özel bir sorgulama yoktur. Bu değer kod tarafında elde edilmektedir. Aslında bu çalışma şeklinin Load metodunun

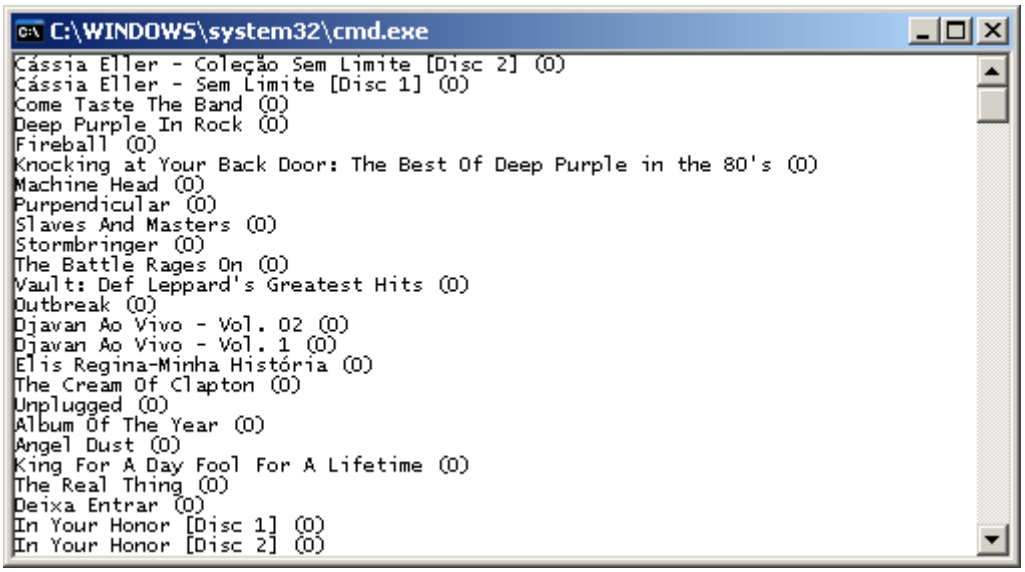
kullanımındaki ile benzer olduğunu söyleyebiliriz. Benzer diyorum çünkü Load kullanımında çalıştırılan SQL sorgusu ile buradaki arasında fark vardır. *(Bakalım iki şekil arasındaki 9 farkı bulabilecek misiniz?* 🙂 *)*

Kişisel Not : *Aslında bazı* ***blog*** *yazılarında* ***LazyLoadingEnabled*** *özelliğinin* ***true*** *olması halinde bu şekilde çalıştığı gösterilmiştir. İşte sürüm farklılıklarının bir sonucu daha. Bu nedenle Beta 2 üzerinde yazdığımız bu konunun gelecek sürümlerde değişikliğe uğraması söz konusu olabilir.*

Şimdi kodumuzda aşağıdaki değişikliği yaptığımızı düşünelim.

```
using (ChinookEntities context = new ChinookEntities())
{
  context.ContextOptions.LazyLoadingEnabled = false;
```

Burada **LazyLoadingEnabled** özelliğine **false** değer atanması sonucu aşağıdaki sonuçlar ile karşılaşılacaktır.



Beklediğimiz gibi **Lazy Loading** özelliğini kapattık. Peki ya SQL tarafı? İşte **SQL Server Profiler'** dan yakalanan SQL sorgusu.

```
SELECT
[Extent1].[AlbumId] AS [AlbumId],
[Extent1].[Title] AS [Title],
[Extent1].[ArtistId] AS [ArtistId]
FROM [dbo].[Album] AS [Extent1]
```

Sanırım yazımızın başladığı noktadaki sonuçlara döndük ne dersiniz? 🙂 özetle **Ado.Net Entity Framework 4.0 Beta 2** sürümünde **Lazy Loading** kabiliyetinin varsayılan olarak açık geldiğini ve istendiğinde **Context** nesnesinin **LazyLoadingEnabled** özelliğine atanacak **false** değeri ile kapatılabileceğini görmüş olduk. Bakalım sürümün ileriki

versiyonlarında ne gibi yenilikler olacak. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Before.rar (41,07 kb)

ReallyLazy.rar (98,20 kb)

Workflow Foundation 4.0 - Persistence [Beta 2] (2009-12-01T08:29:00)

wf,wf 4.0,

 Merhaba Arkadaşlar,

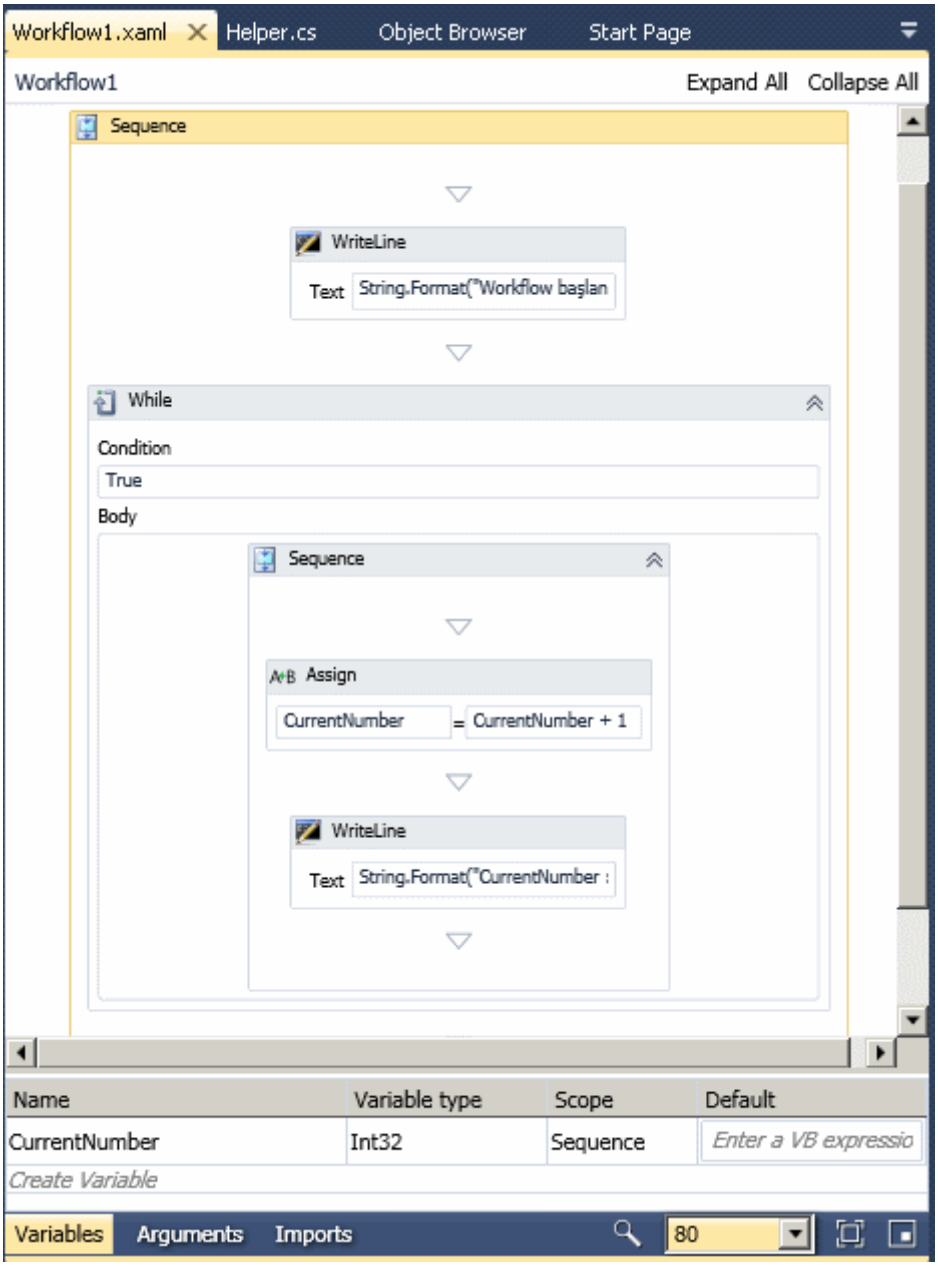
Bundan bir kaç yıl önce, eşimle birlike İtalya' daki Amca' sını ziyarete gitmiştik. Amcamız, Milano şehrinde uzun yıllar restoran işiyle uğraşmış oldukça yetenekli bir aşçı ve işletmeciydi. Bir gün İtalya' da bizi davet ettiği bir restoranda yemek yerken güzel bir tavsiyede bulunduğunu hatırlıyorum; *"Yemekte bitiremediğiniz salatalar mı var? Yemekten sonra onları tavada ısıtın ve buzdolabına koyun. Ertesi gün yine soslayarak yiyebilirsiniz. Tabiki ağır soslu salatalar değilde hafif olanlardan bahsediyorum"*. Bir anlamda salatayı herhangibir anda o anki haliyle saklayıp*(ki ben buna Persist etmek demek istiyorum)* sonra t zamanında yeniden yemek. Her ne kadar bu senaryoda salatayı yıllarca saklamak zor olsada*(ki bu durumdada bazı sonuçlarına katlanmak gerekebilir)* yazılım dünyasında uzun süreli bir iş akışının yıllarca saklanabilmesi mümkündür. İşte bu günkü konumuz; **Workflow Foundation 4.0 ile Persistence** işlemleri nasıl gerçekleştirilebilir...

Workflow Foundation modeli ile geliştirilen **uzun süreli işlemlerde(Long Running Process)** en önemli konulardan biriside, **Workflow** örneğinin herhangibir **t** anında **kalıcı olarak saklanabilmesi(Persist)** ve istenildiğinde saklandığı yerdeki içeriği ile birlikte tekrardan ayağa kaldırılabilmesidir. **Persist** edilecek verilerin nerede saklanacağına ilişkin olarak çalışma zamanının varsayılan tutumu belleği kullanmaktır. Ancak çok sayıda **Workflow** örneğinin **Long Running Process** olarak değerlendirildiği gerçek hayat vakalarında hem yönetim(Administration) hemde kalıcılığın daha kuvvetli olması adına **SQL** veritabanı ortamının değerlendirilmesi çok daha doğru bir davranıştır.

İlk versiyonlarından bu yana, **Workflow** tarafında **persistence** alanı olarak **SQL** veritabanının kullanılması bazı **SQL Script'** lerinin çalıştırılıp gerekli veritabanının oluşturulması ile başlamaktadır. Bu veritabanının **Workflow** çalışma zamanı tarafından kullanılacağının belirtilmesi sırasında bağlantı bilgisi verilmesi yeterlidir. Buda önemli bir noktadır ki veritabanını, **Workflow** uygulamalarını host edip çalıştıran sunucunun dışında bir yerde tutma şansına sahip olabiliriz. Yazımızın bundan sonraki kısımlarında, **Workflow Foundation 4.0 Beta 2** üzerinden **Persistence** sistemini nasıl kullanabileceğimize çok basit bir örnek üzerinden bakmaya çalışacağız. İşe ilk olarak**WFPersistenceStore***(ki siz istediğiniz bir veritabanı adını verebilirsiniz)* isimli bir veritabanını oluşturarak başlayabiliriz. Bu işlemin ardından varsayılan olarak**C:\WINDOWS\Microsoft.NET\Framework\v4.0.21006\SQL\en** adresinde duran **SqlWorkflowInstanceStoreSchema, SqlWorkflowInstanceStoreLogic sql** script' lerini sırasıyla oluşturulan veritabanı üzerinde çalıştırmamız gerekmektedir. Söz konusu işlem sonrasında **Workflow** örneklerinin tutulması ile ilişkili olaraktan gerekli veritabanı nesnelerinin üretildiği görülebilir*(Tables, Stored Procedures, Views vb...).*

Workflow tarafında **Persistence** işlemleri için birden fazla yol kullanılabilir. Bunların içerisinde belkide en basiti **Persist** isimli aktivite bileşeninin Workflow içerisinde bir noktada değerlendirilmesidir. Bu yol dışında istenirse **WorkflowApplication** nesne örneğinin **PersistableIdle** özelliğide değerlendirilebilir. Ancak unutulmaması gereken noktalardan birisi de **Workflow** örneklerinin saklanma durumlarının aslında uzun süreli süreçler için geçerli oluşudur. Bu nedenle ilgili özellik ve konfigurasyon işlemleri esas itibariyle **WorkflowApplication** tipi üzerinden yapılmaktadır.

Dilerseniz daha fazla vakit kaybetmeden örneğimizi geliştirmeye başlayalım. İlk olarak basit bir **Workflow Console Application** projesi oluşturarak yolumuza devam edebiliriz. **Workflow** uygulamamızda**SQL** tabanlı bir **Persistence** sistemi kullanacağımızdan gerekli **assembly'** larında projeye referans edilmesi gerekmektedir. Bu amaçla uygulamamıza **System.Activities.DurableInstancing.dll'** ini eklememiz gerekmektedir. Ayrıca **System.Runtime.dll** assembly' ının **4.0** versiyonun da eklenmesi gerekmektedir. Söz konusu referans eklemeleri sonrasında tasarım zamanındaki görüntüsü aşağıdaki gibi olan çok basit bir Workflow geliştirerek devam edebiliriz.



Workflow örneğine ait **XAML** içeriğinin sadece Sequence bölümünden oluşan parçası aşağıdaki gibidir.

```
 <Sequence sad:XamlDebuggerXmlReader.FileName="c:\documents and
settings\bsenyurt\my documents\visual studio
10\Projects\WFPersistence\WFPersistence\Workflow1.xaml"
sap:VirtualizedContainerService.HintSize="486,614">
   <Sequence.Variables>
     <Variable x:TypeArguments="x:Int32" Name="CurrentNumber" />
   </Sequence.Variables>
   <sap:WorkflowViewStateService.ViewState>
     <scg3:Dictionary x:TypeArguments="x:String, x:Object">
       <x:Boolean x:Key="IsExpanded">True</x:Boolean>
```

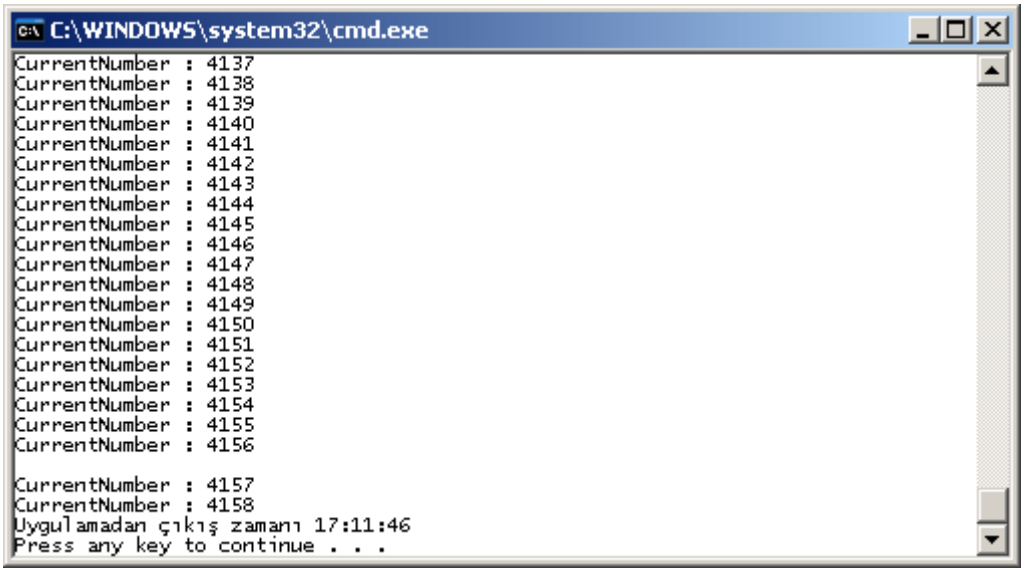
```
      </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <WriteLine sap:VirtualizedContainerService.HintSize="464,59"
Text="[String.Format(&amp;quot;Workflow başlangıcı {0}&amp;quot;,
DateTime.Now.ToLongTimeString())]" />
    <While sap:VirtualizedContainerService.HintSize="464,391" Condition="True">
      <Sequence sap:VirtualizedContainerService.HintSize="438,280">
        <sap:WorkflowViewStateService.ViewState>
          <scg3:Dictionary x:TypeArguments="x:String, x:Object">
            <x:Boolean x:Key="IsExpanded">True</x:Boolean>
          </scg3:Dictionary>
        </sap:WorkflowViewStateService.ViewState>
        <Assign sap:VirtualizedContainerService.HintSize="242,57">
          <Assign.To>
            <OutArgument x:TypeArguments="x:Int32">[CurrentNumber]</OutArgument>
          </Assign.To>
          <Assign.Value>
            <InArgument x:TypeArguments="x:Int32">[CurrentNumber + 1]</InArgument>
          </Assign.Value>
        </Assign>
        <WriteLine sap:VirtualizedContainerService.HintSize="242,59"
Text="[String.Format(&amp;quot;CurrentNumber : {0} &amp;quot;, CurrentNumber.ToString())]"
/>
      </Sequence>
    </While>
  </Sequence>
```

Tasarım ekranından görebileceğiniz üzere **Workflow** örneğimiz bir **WriteLine** ile başlamakta ve sonrasında sonsuz bir döngüye girmektedir. Bu sonsuz döngü için **While** bileşeninden yararlanılmaktadır. Sonsuz döngü içerisinde yer alan **Assign** bileşeni sayesinde, **CurrentNumber** isimli **Sequence** seviyesindeki **Variable'** ın değeri sürekli olarak arttırılmaktadır. Buna göre yazacağımız program kodunun önemi vardır. **Workflow** örneğini Host eden **Console** uygulamasını öyle yazmalıyızki, uygulama kapandığında **Workflow** örneğinin o anki hali**Persistence** tablolarına yazılabilsin. Bu nedenle **Program.cs** içeriğini aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.Activities;
using System.Activities.DurableInstancing;

namespace WFPersistence
{
```

```
class Program
{
    static void Main(string[] args)
    {
        string conStr = "data source=.;database=WFPersistenceStore;integrated
security=SSPI";

        // Persistence işleminin yapılacağı veritabanını işaret edecek ve yönetecek nesne
örneklenir
        SqlWorkflowInstanceStore instanceStore = new
SqlWorkflowInstanceStore(conStr);

        // Workflow nesnesi örneklenir
        Workflow1 wf1 = new Workflow1();

        // WorkflowApplication nesnesi örnekleni ve wf1' i çalıştıracağı belirtilir
        WorkflowApplication wfApp = new WorkflowApplication(wf1);
        // WorkflowApplication nesne örneğinin persistence store olarak instanceStore
isimli SqlWorkflowInstanceStore nesne örneğini kullanacağı ve dolayısıyla saklama
işlemlerinin WFPersistenceStore veritabanında gerçekleştirileceği belirtilir
        wfApp.InstanceStore = instanceStore;

        // WorkflowApplication başlatılır
        wfApp.Run();

        Console.WriteLine("Uygulamadan çıkmak için bir tuşa basın");
        Console.ReadLine();

        // Unload metodunun çalıştırılması nedeniyle Workflow örneğinin o anda
bulunduğu içerik itibariyle Persist edilmeside sağlanmaktadır.
        wfApp.Unload();

        Console.WriteLine("Uygulamadan çıkış zamanı
{0}",DateTime.Now.ToLongTimeString());
    }
}
}
```

örneğimizi çalıştırmadan önce **SQL Server Profiler** aracınında açık olduğundan emin olalım. Böylece arka planda gerçekleşen insert işlemlerini görme şansımız olacaktır. Ben örneğimizi bu haliyle test ederken aşağıdaki sonuçlar ile karşılaştım. *(Bilinçli olarak belirli süre geçtikten sonra uygulamayı kapattım)*



Dikkat edileceği üzer **While** döngüsünde **CurrentNumber** değeri **4158** iken çıkılmıştır. Tabiki sizin testlerinizde bu değer çok daha farklı olabilir. Uygulama **Unload** metodu çağrısını gerçekleştirdikten sonra **SQL Server Profiler** aracından yakalan sorgu aşağıdaki gibidir*(SQL İçeriği çok uzun olduğundan son kısmı kırpılmıştır)*

exec sp_executesql N'begin transaction
declare @result int
exec @result = **[System.Activities.DurableInstancing].[SaveInstance]** @instanceId, @surrogateLockOwnerId, @handleInstanceVersion, @handleIsBoundToLock, @primitiveDataProperties, @complexDataProperties, @writeOnlyPrimitiveDataProperties, @writeOnlyComplexDataProperties, @metadataProperties, @metadataIsConsistent, @encodingOption, @pendingTimer, @suspensionStateChange, @suspensionReason, @keysToAssociate, @keysToComplete, @keysToFree, @concatenatedKeyProperties, @unlockInstance, @isReadyToRun, @isCompleted,
@lastMachineRunOn, @executionStatus, @blockingBookmarks, @operationTimeout ;
if (@result = 0)
begin
commit transaction
end
else
rollback transaction
',N'**@instanceId** uniqueidentifier,@surrogateLockOwnerId bigint,@handleInstanceVersion bigint, @handleIsBoundToLock bit, @pendingTimer datetime,@unlockInstance bit,@suspensionStateChange tinyint,@suspensionReason nvarchar(450),@isCompleted bit,@isReadyToRun bit, @operationTimeout int,@keysToAssociate xml,@keysToComplete xml, @keysToFree xml,@concatenatedKeyProperties varbinary(8000),@primitiveDataProperties varbinary(8000), @complexDataProperties varbinary(1664),

@writeOnlyPrimitiveDataProperties varbinary(470) ,@writeOnlyComplexDataProperties varbinary(860),@metadataProperties varbinary(8000), @metadataIsConsistent bit,@encodingOption tinyint, @lastMachineRunOn varchar(900),@executionStatus varchar(900),@blockingBookmarks varchar(900)',

@instanceId='074A5D57-C268-43E6-B516-DFE49894D7A7',

@surrogateLockOwnerId=26,@handleInstanceVersion=-1, @handleIsBoundToLock=0,@pendingTimer=NULL,@unlockInstance=1, @suspensionStateChange=0,@suspensionReason=NULL,@isCompleted=0,@isReadyToRun=1, @operationTimeout=29922, @keysToAssociate=NULL,@keysToComplete=NULL ,@keysToFree=NULL, @concatenatedKeyProperties=NULL,@primitiveDataProperties=NULL, @complexDataProperties=...

Görüldüğü üzere **Transaction** içerisine alınmış olan ve **SaveInstance** isimli **Stored Procedure** için yapılan bir çağrısı söz konusudur. Yapılan çağrıda şu an için dikkat edilmesi gereken nokta parametre olarak gelen **instanceId** değeridir. Bu işlemin ardından **Instances** isimli **View** içeriğine bakılırsa söz konusu **InstanceId** değerini içeren bir satırın üretildiği görülmektedir.

10	F4FBB667-94A4-430D-B1D3-BCFEB540AEAA	0
11	11C71F79-0315-4DFC-9929-5430C59F07AC	0
12	75BC92C3-C241-4BFA-8191-6C3F54FFDD7C	0
13	074A5D57-C268-43E6-B516-DFE49894D7A7	0

Peki ya şimdi ne olacak?

T zaman sonra*(Gün, Ay, Yıl bile olabilir. Yeterki SQL tarafındaki bilgilere zarar gelmesin),* bu **Workflow** örneğinin kaldığı yerden ayağa kaldırılarak devam etmesi istenebilir. İşte bu durumda **Persist** edilmiş olan örneğin, kaydedildiği haliyle tekrardan yüklenmesi gerekecektir. Geliştirdiğimiz örnek senaryoya göre, **CurrentNumber** değerinin kaldığı yerden başlayarak devam etmesi gerekmektedir. **Persist** edilmiş olan bir **Workflow** örneğinin tekrardan ayağa kaldırılması için **GUID** tipinden olan **InstanceId** değerinin **Load** metoduna parametre olarak geçirilmesi yeterlidir. İşte Persist edilmiş olan Workflow örneğini ayağa kaldırmak için kullanacağımız kodlarımız.

```
  // WorkflowApplication nesne örneğinin persistence store olarak instanceStore isimli
wInstanceStore nesne örneğini kullanacağı ve dolayısıyla saklama işlemlerinin
nceStore veritabanında gerçekleştirileceği belirtilir
  wfApp.InstanceStore = instanceStore;

  // Persist edilmiş Workflow örneğinin canlandırılması için parametre olarak o Workfl
urulduğunda üretilen InstanceID değeri kullanılır.
  wfApp.Load(new Guid("074A5D57-C268-43E6-B516-DFE49894D7A7"));

  // WorkflowApplication başlatılır
  wfApp.Run();

  Console.WriteLine("Uygulamadan çıkmak için bir tuşa basın");
  Console.ReadLine();

  // Unload metodunun çalıştırılması nedeniyle Workflow örneğinin o anda bulunduğu
bariyle Persist edilmeside sağlanmaktadır.
  wfApp.Unload();

  Console.WriteLine("Uygulamadan çıkış zamanı {0}",DateTime.Now.ToLongTimeString());
```

Tam **Load** metodunun olduğu yerde **breakpoint** koyarak ilerlemenizi öneririm. **Persist** edilen **Workflow** örneğinin canlandırılması esnasında **SQL** tarafında **LoadInstance** isimli bir Stored Procedure' ün çalıştırılması sağlanmaktadır. İşte çalıştırılan SQL sorgusu.

exec **[System.Activities.DurableInstancing].[LoadInstance]** @surrogateLockOwnerId=27,@operationType=3, @keyToLoadBy='00000000-0000-0000-0000-000000000000',

@instanceId='074A5D57-C268-43E6-B516-DFE49894D7A7',

@handleInstanceVersion=-1,@handleIsBoundToLock=0,@keysToAssociate=NULL, @encodingOption=1,@concatenatedKeyProperties=NULL, @operationTimeout=29812

Hatta bu noktada iken **IsLocked** alanın söz konusu **Workflow** örneği için 1 olarak set edildiği gözlemlenebilir. Bir başka deyişle Workflow örneği bu işlem sırasında diğer bir talebe kapatılmış durumdadır.

Results | Messages

	InstanceId	IsLocked	PendingTimer	Creation
1	2C90BBE6-876C-4879-9229-BA0E72AF33CA	0	NULL	2009-1
2	10CDA1A5-9E4F-4658-8C16-FE095591EF05	0	NULL	2009-1
3	C36434A5-DBCE-4D08-883A-61E63711610E	0	NULL	2009-1
4	362ED276-3338-41D9-BD15-CD434FDDB2C0	0	NULL	2009-1
5	DAC6D010-625F-482D-A1A2-64DBA4D6AEB8	0	NULL	2009-1
6	CA5DEBAD-2904-4B52-8B21-FADAF0AC786A	0	NULL	2009-1
7	DCE5F0FA-DB78-4533-B545-8E5461887A35	0	NULL	2009-1
8	5EEA8B42-D4A5-4A8F-9634-3BB80C306374	0	NULL	2009-1
9	92AB9B71-BAA8-4527-8BFC-9EDB1F9F26E4	0	NULL	2009-1
10	F4FBB667-94A4-430D-B1D3-BCFEB540AEAA	0	NULL	2009-1
11	11C71F79-0315-4DFC-9929-5430C59F07AC	0	NULL	2009-1
12	75BC92C3-C241-4BFA-8191-6C3F54FFDD7C	0	NULL	2009-1
13	074A5D57-C268-43E6-B516-DFE49894D7A7	1	NULL	2009-1

Peki ya uygulamanın durumu? Uygulama yeniden çalıştırıldığında **CurrentNumber** değerinin kaldığı yerden devam ettiği gözlemlenecektir. Bunu görme zevkini siz değerli okurlarıma bırakıyorum **Persistence** işlemi bu örneğimizde **WorkflowApplication** nesnesi tarafından ele alınmıştır. Ancak **Workflow Service** isimli önemli bir gerçekde vardır. Bir **Workflow Service'** in uzun süreli olarak saklanması da, gerçek hayat senaryolarında yaşanan vakalardan birisidir. Bu durumu bir sonraki yazımızda ele almaya çalışacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Kişisel Not : *Kalıcı olarak saklama işlemini yazımızın başında belirttiğimiz üzere,* ***Persist*** *aktivitesi ilede kolayca gerçekleştirebiliriz. Bu konu ile ilişkili bir görsel dersi ilerleyen tarihlerde yayınlamaya çalışıyor olacağım.*

WFPersistence.rar (37,63 kb)

INETA Next Tur Programı Belli Oldu (2009-11-30T01:00:00)

ineta,seminer,

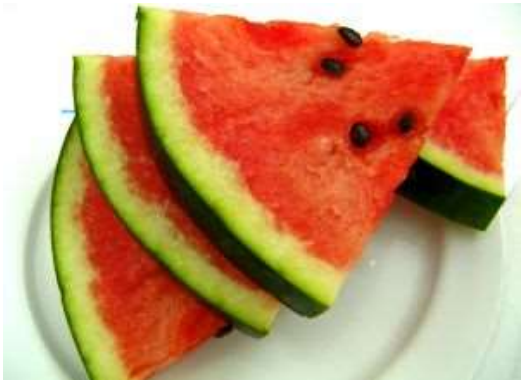
Merhaba Arkadaşlar,

Microsoft' un geliştirmeye yönelik sunduğu ürünlerin yeni sürümlerinde artık 4.0' lı rakamlar bolca yer almakta. C# 4.0, Asp.Net 4.0, WCF 4.0, WF 4.0, Silverlight 4.0, WPF

4.0, Ajax 4.0 vb...Tabi bazı ürünlerde bir sonraki versiyonlarına atlamış durumda. örneğin Visual Studio 2010, Visual Basic 10 gibi.

Peki Microsoft' un bu gelecek sürümlerde, geliştiriciler için ne gibi yenilikleri getirmekte olduğunu merak ediyor musunuz?

O zaman buyrun INETA Next turuna. önce Diyarbakır' da başlayacak olan program sonrasında sırasıyla İstanbul, Kayseri ve Denizli ile devam edecek. Programın tüm detayını aşağıda bulabilirsiniz.



İstanbul(Microsoft Ofisi)
12 Aralık;
10.00-11.00 Silverlight 4 - Daron Yöndem (yazgelistir.com)
11.00-12.00 ASP.NET MVC 2 - Oğuz Yağmur (csharpnedir.com)
14.00-15.00 IIS Media Services - Muammer Benzeş (birliktegelistir.com)
15.15-16.15 Visual Basic 10 Yenilikleri - Daron Yöndem (yazgelistir.com)
13 Aralık
11.00-12.30 ASP.NET 4.0 Yenilikleri - Uğur Umutluoğlu (nedirtv.com)
14.30-15.45 Workflow Foundation 4.0 - Burak Selim Şenyurt (csharpnedir.com)
16.00-17.00 C# 4.0 Yenilikleri - Erkan Balaban (ceviz.net)
17.15-18.00 IE 8 Toolbar Geliştirme - Barış Kanlıca (yazgelistir.com)

Diyarba
5 Aralık
11.00-12
14.00-15
16.00-17
6 Aralık
11.00-12
14.00-15
16.00-17

Kayseri
19 Aralı
11.00-12
14.00-15
15.45-16
20 Aralı
11.00-12
14.00-15
15.45-16

Denizli(Pamukkale üniversitesi - Kongre Kültür Merkezi Ana Salon)
26 Aralık
11.00-12.30 Silverlight 4.0 - Daron Yöndem (yazgelistir.com)
14.00-15.30 C# 4.0 Yenilikleri - Oğuz Yağmur (csharpnedir.com)
15.45-16.45 Asp.NET 4.0 Yenilikleri - Uğur Umutluoğlu (nedirtv.com)
27 Aralık
11.00-.12.30 Visual Basic 10 Yenilikleri - Daron Yöndem (yazgelistir.com)
14.00-15.30 Asp.NET MVC 2 - Oğuz Yağmur (csharpnedir.com)
15.45-16.45 ASP.NET AJAX 4.0 - Uğur Umutluoğlu (nedirtv.com)

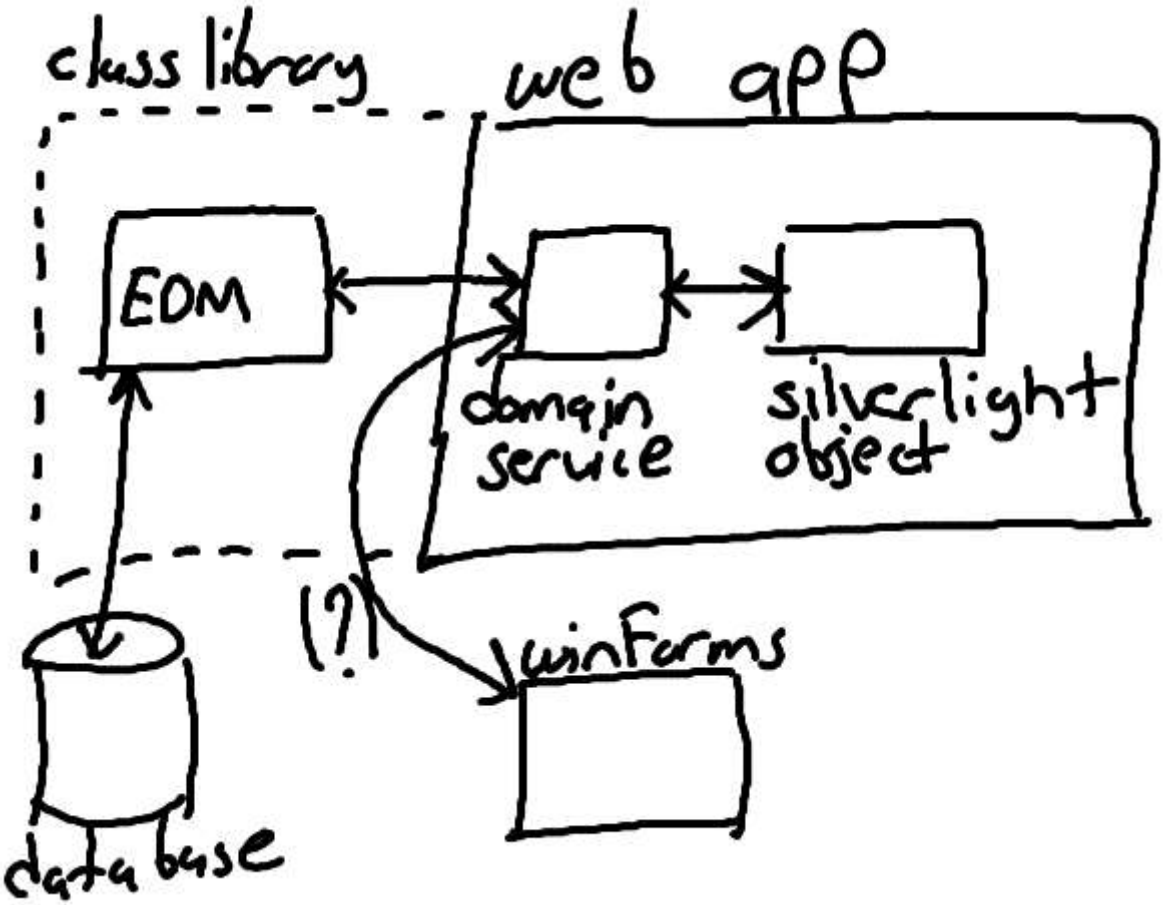
Kayıt için www.inetatr.org adresini ziyaret etmeyi unutmayın!

WCF RIA Services - Gerçekten WCF (2009-11-26T08:19:00)

wcf ria services,.net ria services,wcf,wcf eco system,

Merhaba Arkadaşlar,

Uzun ve yorucu bir geceydi...Dün gece **WCF RIA Service'** leri ile ilişkili görsel bir dersin hazırlıklarını yaparken sevgili **Mehmet Cengiz** arkadaşımın hediyesi olan tablet üzerinde aşağıdaki şekli çizdiğimi farkettim. Bu şekilde **WCF RIA Service'** i kullanan basit bir **Silverlight** uygulamasının anlaşılır hali yer almaktadır. **Web** uygulamamız, içerisinde **Silverlight** nesnesi barındıran test sayfası ve **Domain Service** sınıfı, dışarıda veya aynı alanda duran **Ado.Net Entity Data Model** içeriği ve onun kullandığı veritabanı. Soru işareti içeren ok ve Winforms kutucuğunu ise sonradan eklemeye karar verdim ve araştırmam da işte bu şekilde başladı

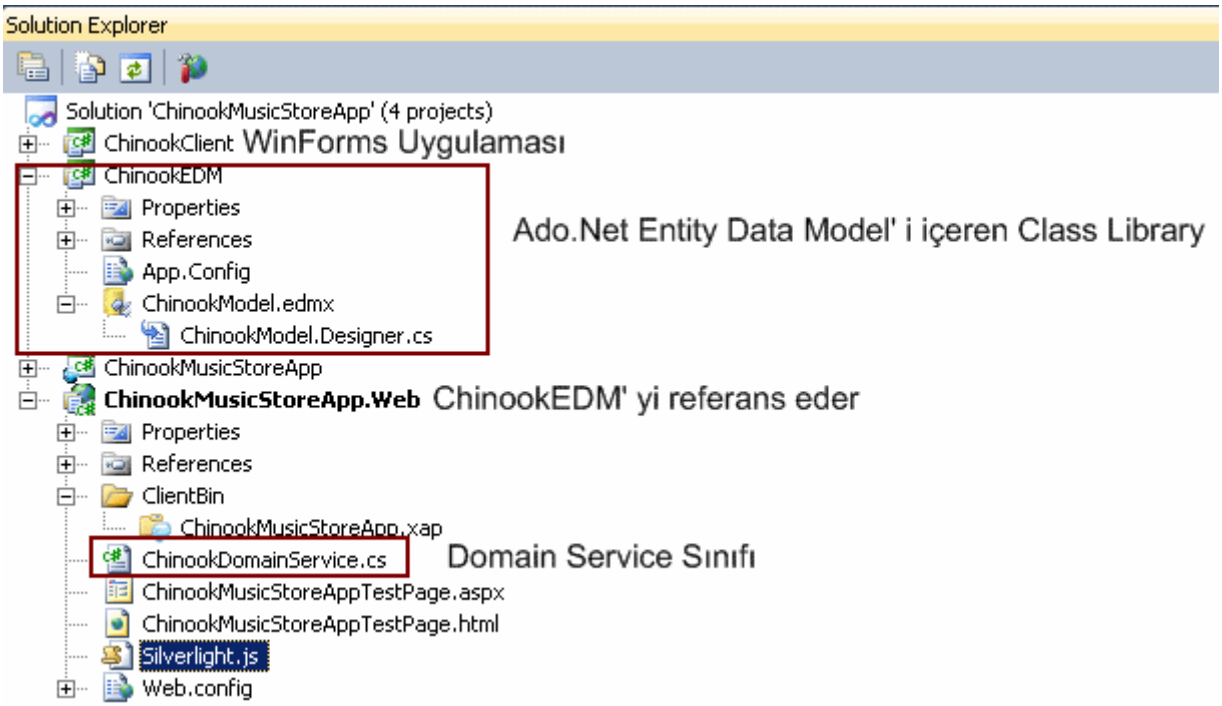


Gerçektende **Silverlight** uygulamlarını **RIA Service'** lerini kullanacak şekilde oluşturduğumuzda, genellikle **Silverlight** nesnesini host eden bir **Web** uygulaması olması gerekmektedir. Bu web uygulamasının içerisinde yer alan ve **Sliverlight** tarafına veri hizmeti sunan **RIA Service'** leri, harici bir **Class Library** içerisinde bulunabilen bir **Entity Data Model'** i(yada aynı web domain içerisindekini) kullanabilir. Bu şekli çizmeye çalışırken aklıma takılan ve soru işareti bırakan husus ise şudur; **Web uygulaması dışarısında yer alan bir diğer uygulamanın, Web alanı içerisinde yer alan bir RIA Service' ini kullanması mümkün olabilir mi?**

Sonuç itibariyle **WCF** alt yapısı üzerine oturan **RIA Service'** leri web tabanlı olarak host edilmekte ve hizmet sunarken WCF çalışma zamanı motoru tarafından yürütülmektedirler. Bu nedenle belkide dışarıya **WSDL** içeriği sunabilir ve **Silverlight** gibi **Rich Internet Application'** lar dışındaki uygulamalar tarafından da kullanılabilirler. İşte bu sorunun cevabını ararken Brad Abrams' ın blog yazısı ile karşılaştım. **Brad Abrams** blog yazısında **Silverlight Business Application** tipinden geliştirdiği bir proje üzerinden ilgili konuyu irdeleyerek **RIA Service'** lerin aslında birer **WCF Service** olduğunu ispat etmektedir. Konuyu bende bu haliyle incelemek istediğimden aynı örneğin benzerini bu kez **Silverlight Application** tipinden olan bir projede yapmayı denedim. Ancak çok küçük bir özelliği kullanmamam nedeni ile hata ile karşılaştım. Bu hatanın sebebini ve çözümünü bulmam biraz zamanımı aldı. Dolayısıyla**Brad Abrams'** ın kullandığı proje şablonu yerine

normal bir **Silverlight Application** üzerinden benzer bir senaryoyu uygulamaya karar verdim. İşte bu tecrübenin hikayesi;

İlk olarak aşağıdaki şekilde görülen tipte bir Solution içeriği oluşturarak işe başlamaya karar verdim.



Silverlight Application şablonunda bir **Solution** açılmasına rağmen aslında **ChinookMusicSotreApp** isimli proje hiç kullanılmayacaktır. **ChinookMusicStoreApp.Web** isimli web projesi,**ChinookEDM** isimli sınıf kütüphanesini referans etmektedir. Bu şekilde **Albums** ve **Artist** isimli **Chinook** tablolarına ait Entity karşılıklarını içermekte olan **Ado.Net Entity Data Model**öğesini kullanabilmektedir. Diğer yandan Web tarafına eklenen **Domain Service** sınıfı içeriğinde **Album** tipi üzerinden **Insert, Update, Delete** işlemleri yapılmasına izin verecek şekilde geliştirmeler yapılmıştır. Bu adımları tamamladıktan sonra **ChinookDomainService** sınıfının içeriği biraz değiştirerek aşağıdaki hale getirdim. Aslında tek yaptığım GetAlbumsByFirstLetter isimli metodu eklemek oldu.

```
namespace ChinookMusicStoreApp.Web
{
    using System.Data;
    using System.Linq;
    using System.Web.DomainServices.Providers;
    using System.Web.Ria;
    using ChinookEDM;
```

```
[EnableClientAccess()]
public class ChinookDomainService
    : LinqToEntitiesDomainService<ChinookEntities>
{
    // firstLetter parametresi ile başlayan Album' lerin getirilmesini sağlar
    public IQueryable<Album> GetAlbumsByFirstLetter(string firstLetter)
    {
        return from album in ObjectContext.Albums
               where album.Title.StartsWith(firstLetter)
               orderby album.Title
               select album;
    }


    // Tüm Artist' lerin getirilmesini sağlar
    public IQueryable<Artist> GetArtists()
    {
        return this.ObjectContext.Artists;
    }

    // Yeni bir Album eklenmesi için kullanılır
    public void InsertAlbum(Album album)
    {
        if ((album.EntityState != EntityState.Added))
        {
            if ((album.EntityState != EntityState.Detached))
            {
                this.ObjectContext.ObjectStateManager.ChangeObjectState(album,
EntityState.Added);
            }
            else
            {
                this.ObjectContext.AddToAlbums(album);
            }
        }
    }

    // Bir Album' ü güncelleştirmek için kullanılır
    public void UpdateAlbum(Album currentAlbum)
    {
        if ((currentAlbum.EntityState == EntityState.Detached))
        {
            this.ObjectContext.AttachAsModified(currentAlbum,
this.ChangeSet.GetOriginal(currentAlbum));
```

```
        }
    }

    // Bir Albumu silmek için kullanılır
    public void DeleteAlbum(Album album)
    {
        if ((album.EntityState == EntityState.Detached))
        {
            this.ObjectContext.Attach(album);
        }
        this.ObjectContext.DeleteObject(album);
    }
  }
}
```

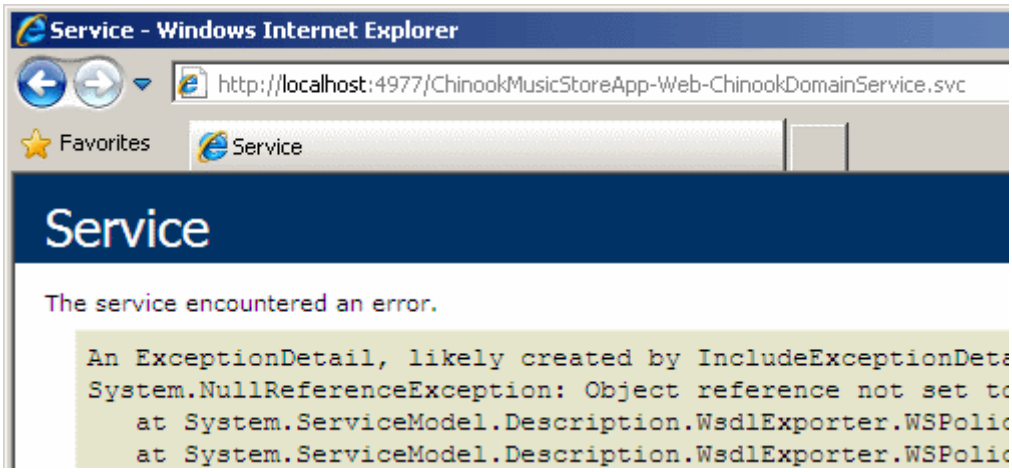
Domain Service sınıfı bu şekilde hazırlandıktan sonra artık asıl işimize odaklanabiliriz. İlk hedefimiz **ChinookDomainService** isimli sınıfın aslında çalışma zamanı için bir**WCF Service** olduğunu göstermektir. Buna göre Web üzerinden **svc** uzantılı olarak erişilebiliyor olması gerekmektedir. Peki WCF çalışma zamanı için herhangibir yerde bir tanımlama yer almakta mıdır? Aslında **web.config** dosyası içeriğine bakıldığında **system.ServiceModel** elementinin aşağıdaki gibi eklenmiş olduğu görülecektir.

Web.config | ChinookDomainService.cs | ChinookModel.edmx | Form1.cs [Design] | Start Page

```
<?xml version="1.0"?>
<configuration>

    <system.web>
        <httpModules>
            <add name="DomainServiceModule"
type="System.Web.Ria.Services.DomainServiceHttpModule, System.Web.Ria,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
        </httpModules>
        <compilation debug="true" targetFramework="4.0" />
    </system.web>
    <system.webServer>
        <modules runAllManagedModulesForAllRequests="true">
            <add name="DomainServiceModule" preCondition="managedHandler"
                type="System.Web.Ria.Services.DomainServiceHttpModule,
System.Web.Ria, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=31BF3856AD364E35" />
        </modules>
        <validation validateIntegratedModeConfiguration="false" />
    </system.webServer>

    <system.serviceModel>
        <serviceHostingEnvironment aspNetCompatibilityEnabled="true" />
    </system.serviceModel>
</configuration>

```

önemli Not : *örneğimizde* ***EDM'*** *yi Web uygulamamıza referans ettiğimiz bir* ***Class Library*** *içerisinde tuttuğumuz için, bu kütüphanenin* ***App.config*** *dosyası içerisine yazılan****connectionString*** *bilgisinin,* ***Web*** *uygulamasının* ***web.config*** *dosyasına eklenmesi gerekmektedir. Aksi takdirde çalışma zamanında hata alınacaktır.*

Hımmm...O halde **Asp.net development server'** ın çalıştırılmasını sağlayıp herhangibir tarayıcıdan örneğimize göre **http://localhost:4977/ChinookMusicStoreApp-Web-ChinookDomainService.svc** adresini talep ettiğimde bir Service ekranı ile karşılaşmam gerekmektedir. Oysaki bu denemenin ardından ben aşağıdaki ekran görüntüsü ile karşılaştım. 😟

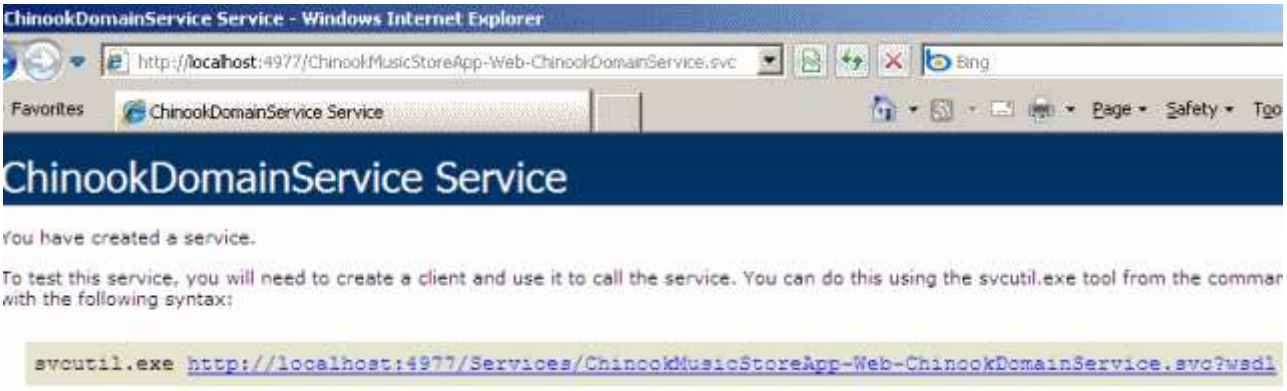


Oysaki **Brad Abrams** örneğinde svc uzantısından sonra servis içeriği görüntülenmiş hatta **WSDL** çıktısı bile elde edilebilmiştir. Acaba sorun nerededir?

Yaptığım araştırmalar sonucunda bu servise erişmek için bir **Authentication** ayarlamasının yapılması gerektiğini öğrendim. Bu çok doğaldı çünkü **RIA Service'** in kullanılmak istendiği **Web Domain'** i dışarısına açılması gibi bir durum söz konusuydu. Dolayısıyla **web.config** dosyası içerisinde **system.Web** elementi altında **authentication** için gerekli tanımlamaların yapılması gerekmekteydi. Bu örnekte herhangibir doğrulama kullanmadığımdan*(Forms, Passport vb...)* **mode** değerini **None** olarak bırakmayı tercih ettim.

```
<?xml version="1.0"?>
<configuration>

  <system.web>
    <httpModules>
      <add name="DomainServiceModule"
type="System.Web.Ria.Services.DomainServiceHttpModule, System.Web.Ria,
Version=4.0.0.0, Culture=neutral, PublicKeyToken=31BF3856AD364E35" />
    </httpModules>
    <compilation debug="true" targetFramework="4.0" />
   <authentication mode="None"/>
  </system.web>
.
.
.
```

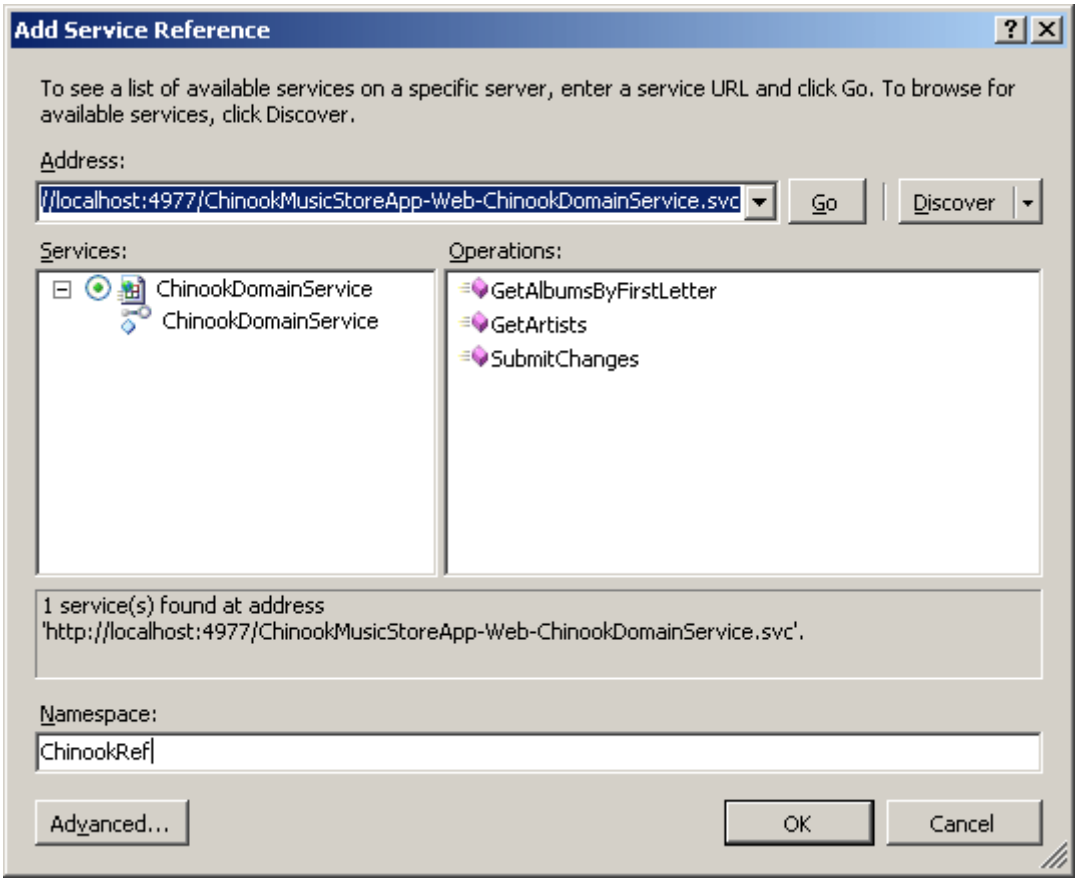
Bu işlemin ardından **http://localhost:4977/ChinookMusicStoreApp-Web-ChinookDomainService.svc** adresine tekrardan talepte bulunduğumda aşağıdaki görüntü ile karşılaştım.



Hatta **WSDL** talebi sonrası söz konusu servisin **Description** içeriğinin de geldiğini gördüm.

Demekki **Domain Service** sınıfı yada geliştirdiğimiz **RIA Service** gerçektende bir WCF Service' miş.

Ancak ispatı tamamlamak için söz konusu servisi örnek bir istemcide kullanabiliyor olmam da gerekmekteydi. Bu nedenle **Solution** içerisinde yer alan **Win Forms** uygulamasına **Add Service Reference** ile**http://localhost:4977/ChinookMusicStoreApp-Web-ChinookDomainService.svc** adresinden gerekli **Proxy** üretiminin gerçekleştirilmesi yeterli olacaktı. Ben bu şekilde yoluma devam ettim ve aşağıdaki ekran görüntüsünde olduğu ilgili servis içeriğini izin verilen operasyonları ile*(SubmitChanges metodunun da olduğuna dikkat edelim)* üretilebildiğini gördüm.

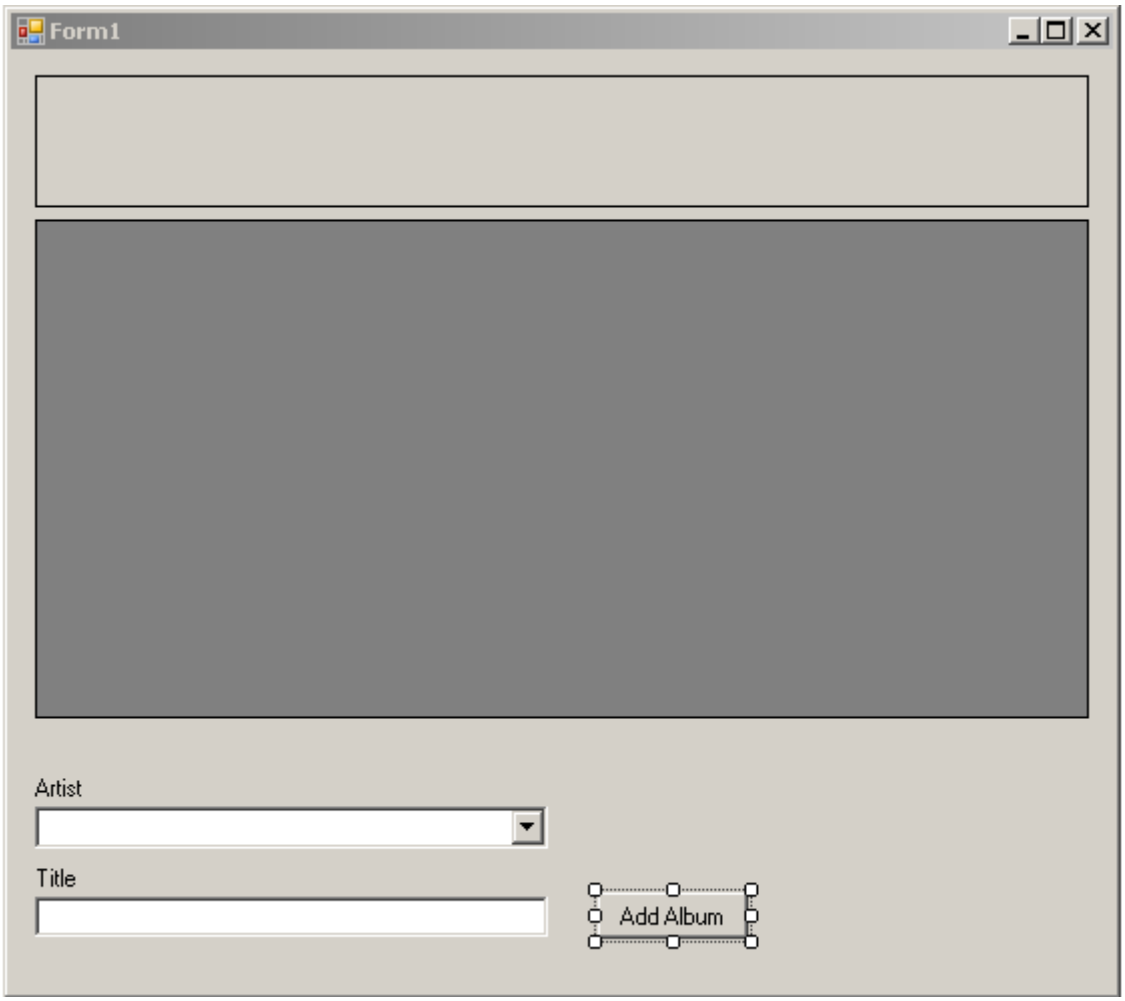


İlgili ekleme işlemi sonrasında **app.config** dosyası içeriğinin aşağıdaki gibi üretildiğini farkettim.

app.config | Start Page | Object Browser | Web.config | ChinookDomainService.cs

```
 1  <?xml version="1.0" encoding="utf-8" ?>
 2  <configuration>
 3      <system.serviceModel>
 4          <bindings>
 5              <basicHttpBinding>...</basicHttpBinding>
21              <customBinding>...</customBinding>
36          </bindings>
37          <client>
38              <endpoint address="http://localhost:4977/Services/
    ChinookMusicStoreApp-Web-ChinookDomainService.svc/soap"
39                  binding="basicHttpBinding"
    bindingConfiguration="BasicHttpBinding_ChinookDomainService"
40                  contract="ChinookRef.ChinookDomainService"
    name="BasicHttpBinding_ChinookDomainService" />
41              <endpoint address="http://localhost:4977/Services/
    ChinookMusicStoreApp-Web-ChinookDomainService.svc/binary"
42                  binding="customBinding"
    bindingConfiguration="BinaryHttpBinding_ChinookDomainService"
43                  contract="ChinookRef.ChinookDomainService"
    name="BinaryHttpBinding_ChinookDomainService" />
44          </client>
45      </system.serviceModel>
46  </configuration>
```

Dikkat edileceği üzere iki adet **EndPoint** tanımlaması yapıldığı görülmektedir. Ben yazının bundan sonraki kısmında aşağıdaki ekran görüntüsüne sahip bir **Form**geliştirerek ilerlemeyi tercih etmekteyim.



Aslında bir önceki yazımızda yaptığımız gibi A' dan Z'ye Button bileşenlerimiz bulunmaktadır ve herhangibirine basıldığında bu harf ile başlayan albümler listelenmektedir. Buna ek olarak yeni bir Album ekleme özelliğide bulunmaktadır. Yeni bir Album nesnesi örneklendiğinde **Title** ve **ArtistId** değerlerinin mutlaka girilmesi gerekmektedir. **ArtistId** içinse kullanıcının var olan**Artist**' lerden herhangibirini seçmesi sağlanmaktadır. İşte Form uygulamamıza ait kodlarımız.

```
using System;
using System.Windows.Forms;
using ChinookClient.ChinookRef;


namespace ChinookClient
{
   public partial class Form1
      : Form
```

```
{
    ChinookDomainServiceClient proxy;

    public Form1()
    {
        InitializeComponent();

        proxy = new
ChinookDomainServiceClient("BasicHttpBinding_ChinookDomainService");
    }

    private void Form1_Load(object sender, EventArgs e)
    {
        for (int i = 65; i < 91; i++)
        {
            Button btn = new Button();
            btn.Width = 24;
            btn.Height = 24;
            btn.Text = ((char)i).ToString();
            btn.Click += (snd, ea) =>
                {
                    grdAlbums.DataSource=proxy.GetAlbumsByFirstLetter(btn.Text).Roo
tResults;
                };
            pnlButtons.Controls.Add(btn);
        }

        cmbArtists.DataSource=proxy.GetArtists().RootResults;
    }

    private void btnAddNewAlbum_Click(object sender, EventArgs e)
    {
        if (!String.IsNullOrEmpty(txtTitle.Text))
        {
            Album albm = new Album
            {
                Title = txtTitle.Text,
                 ArtistId=((Artist)cmbArtists.SelectedItem).ArtistId
            };

            ChangeSetEntry[] entry = new ChangeSetEntry[]{
                new ChangeSetEntry {
                 Entity=albm,
                  Operation= DomainOperation.Insert
```

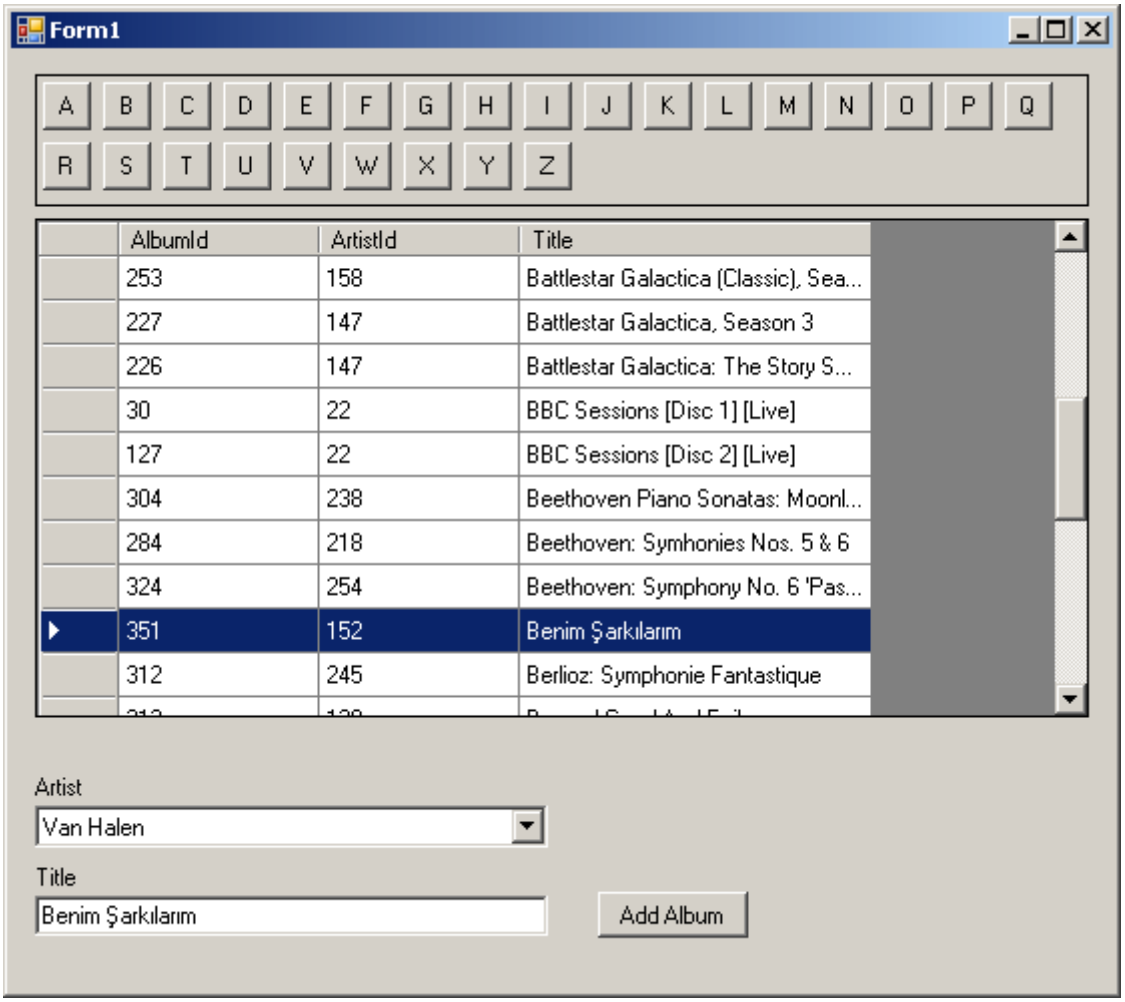
```
            }
            };

            proxy.SubmitChanges(entry);
        }
    }
  }
}
```

Album' lerin alfabetik olarak elde edilmesi veya Artist listesinin getirilmesi bir yana, en önemli noktalardan biriside yeni bir Album nesnesinin nasıl eklendiğidir. Dikkat edileceği üzere bu işlem için**ChangeSetEntry** tipinden bir dizi kullanılmıştır. Aslında **Insert, Update** ve **Delete** işlemlerinde bu tipten yararlanılmakta ve toplu olarak değişikliklerin bildirilmesi sağlanabilmektedir. örnekte bir **Album** nesnesinin eklenmesi istendiğinden **ChangeSetEntry** oluşturulurken **Operation** özelliğine **DomainOperation.Insert** sabit değeri atanmıştır. Buna göre **albm** isimli **Album** nesne örneği için bir **Insert** işlemi yapılacağı vurgulanmaktadır. **ChangeSetEntry** dizisi içerisinde yer alan **insert, update** ve **delete** işlemlerinin servis tarafına gönderilmesi içinse **SubmitChanges** metodunun kullanılması yeterlidir.

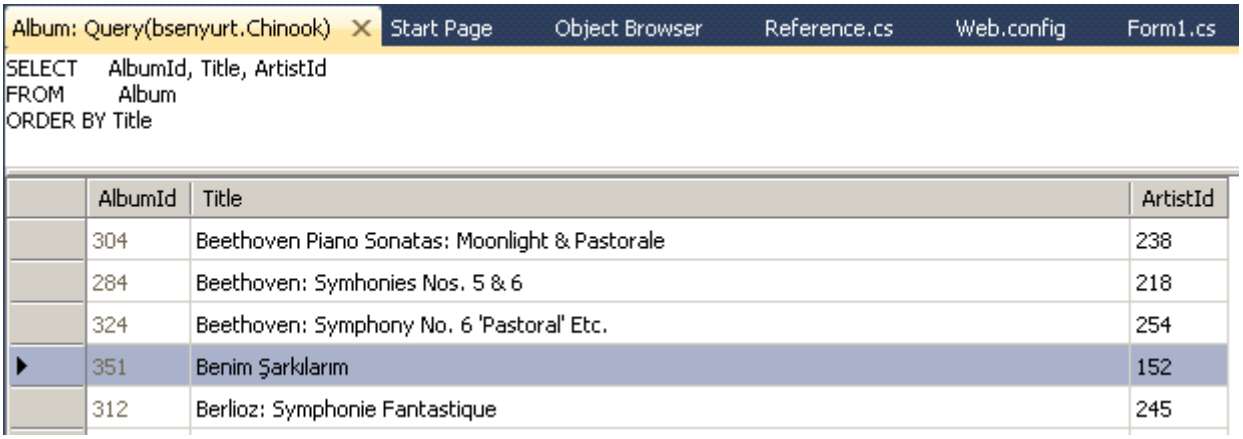
önemli Not : *Artist bilgilerinin ComboBox içerisinde görülebilmesi için istemci tarafına atılan Artist sıfınında* ***ToString*** *metodu override edilmiştir.*

Bilindiği üzere ***Windows Forms*** *uygulamalarında* ***List*** *kontrollerinin içeriklerine* ***Object*** *türünden herhangibir referans atanabilmektedir. örneğimizde de* ***Artist*** *nesne örneklerinin atanması söz konusudur. Ancak* ***ComboBox*** *içerisindeki öğelerde hangi bilgilerin görüneceği* ***ToString*** *metodunun ezilmesi ile sağlanabilir. Tabi bu durumda bir sorunda ortaya çıkmaktadır.****Service'*** *te yapılacak değişiklikler sonrasında istemci tarafındaki referans güncellenirse* ***Entity*** *sınıfları tekrardan oluşturulacağından* ***ezilen(Override) ToString*** *metodunun uçacağı görülecektir. Bu duruma dikkat etmek gerekir.*

İşte çalışma zamanına ait örnek bir görüntü.

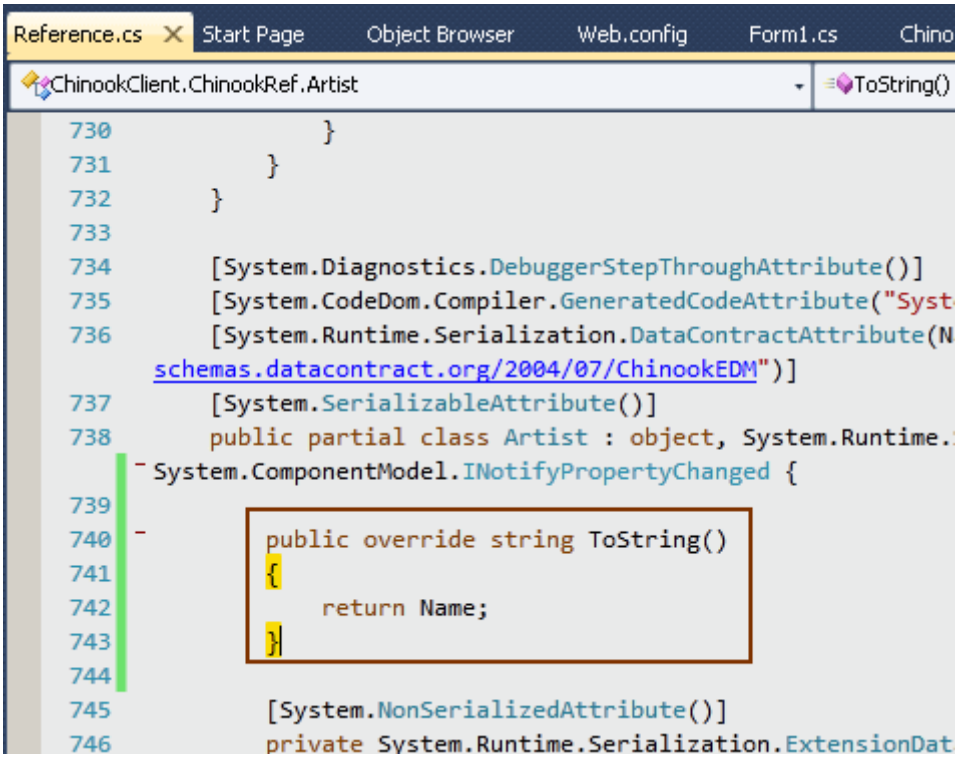


Test olarak **Van Halen** isimli grub için **Benim Şarkılarım** isimli yeni bir albüm eklenmeye çalışılmıştır. Bu işlemin ardınan SQL tarafına bakıldığında **Album** isimli tabloya da ilgili satırın eklendiği gözlemlenebilir.



İşte bu kadar. **WCF RIA Service'** ler ile ilişkili araştırmalarıma devam etmekteyim. Yeni bilgiler edindikçe sizlerle paylaşıyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ChinookMusicStoreApp.rar (590,82 kb)

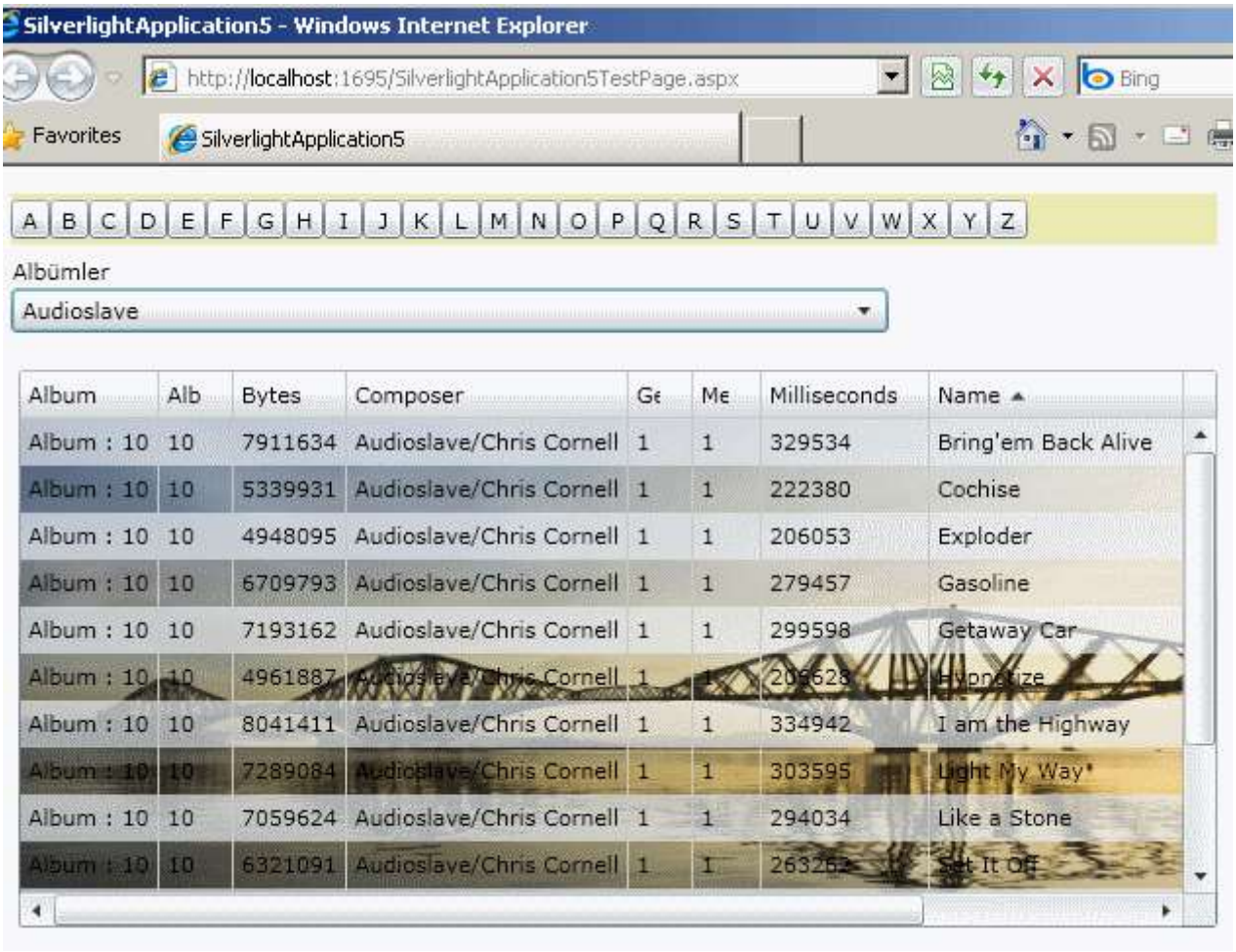


WCF RIA Services - Kendi Sorgularımızı Kullanmak (2009-11-25T13:30:00)

wcf ria services,.net ria services,wcf,wcf eco system,

Merhaba Arkadaşlar,

Bir önceki yazımızda **WCF RIA Service**' lerine kısa bir giriş yapmış ve ilk Hello World uygulamamızı geliştirmiştik. Bu yazımızda yine **Chinook** veritabanında yer alan albümlerin alfabetik olarak elde edilebildiği ve bunlara bağlı parçalarında gösterilebildiği bir Silverlight uygulaması yazmaya çalışacağız. Bu örnekteki temel amacımız ise, kendi sorgulama metodlarımızı ilgili **DomainService** sınıfı içerisinde nasıl geliştirebileceğimizi görmek ve **Silverlight** uygulamasında göze daha hoş gelecek*(Her ne kadar buna kendimde inanamasam da)* bir arayüzü tasarlayabilmek olacak. İlk etapta hedefimizin aşağıdaki ekran görüntüsünde yer alan uygulama arayüzü ve fonksiyonelliğine ulaşmak olduğunu ifade etmek isterim.



Dikkat edileceği üzere **A...Z'** ye kadar sıralanmış bir **Button** kümesi görülmektedir. Bu düğmelerden herhangibirisine basıldığında, o harf ile başlayan albümlerin isimleri **ComboBox**bileşenine doldurulmaktadır. Kullanıcı eğer **ComboBox** bileşeninden bir albümü seçerse, bu albüm içerisinde yer alan şarkı listeside alt tarafta yer alan ve arka planında harikulade 😎 bir manzaraya sahip olan **DataGrid** kontrolü içerisine doldurulmaktadır. Bu örnekte baş harfine göre albüm' lerin getirilebilmesi ve seçilen albüme ait olan şarkıların çekilmesi için **WCF RIA Service** içerisinde gerekli sorgu metodlarının yazılmış olması gerekmektedir. Bildiğiniz gibi **Ado.Net Entity Data Model** nesnesi içeriğinden seçilen **Table, View** yada **Stored Procedure'** lere göre **Domain Service** sınıfının içeriğinde hazır metodlar oluşmaktadır. Ancak bu metodlar her zaman için yeterli olmayabilir. özellikle çok büyük boyutta veri kümelerinin döndürülmesi yerine performans açısından filtrelenmiş içeriklerin tedarik edilmesi tercih edilmelidir. Bu açıdan bakıldığında Domain Service sınıfı içerisine kendi operasyonlarımı eklemek veya var olanları uygun bir şekilde güncelleştirmek kaçınılmazıdır. Bu bilgilerden yola çıkarsak, geliştireceğimiz örnekte ilk hedefimiz üretilen **Domain Service** sınıfının metodlarını kendi istediğimiz şekilde geliştirmek olacaktır.

***Kişisel Not: Entity Data Model** ve **DomainService'** in nasıl hazırlanması gerektiğini bir önceki yazımızda incelediğimizden burada tekrar edilmeyecektir. Ancak Entity Data*

Model içerisinde Album ve Track tablolarının karşılıklarının kullanıldığını belirtmek isterim.

İlk olarak, **ChinookDomainService** adı ile oluşturacağımız **Domain Service** sınıfının içeriğindeki tüm operasyonları silip aşağıdaki hale getirdiğimizi düşünelim.

```
namespace SilverlightApplication5.Web
{
   using System.Linq;
   using System.Web.DomainServices.Providers;
   using System.Web.Ria;

   [EnableClientAccess()]
   public class ChinookDomainService : LinqToEntitiesDomainService<ChinookEntities>
   {
      public IQueryable<Album> GetAlbumsByFirstLetter(string firstLetter)
      {
         return from albm in ObjectContext.Album
                where albm.Title.StartsWith(firstLetter)
                orderby albm.Title
                select albm;
      }

      public IQueryable<Track> GetTracks(int albumId)
      {
         return from track in ObjectContext.Track
                where track.AlbumId == albumId
                orderby track.Name
                select track;
      }
   }
}
```

GetAlbumsByFirstLetter ve **GetTracks** isimli metodlar **IQueryable<T>** tipinden referanslar döndürmektedir. **GetAlbumsByFirstLetter** metodu parametre olarak **string** bir bilgi almakta ve **Title** bilgisi bu içerik ile başlayanların listesini geriye döndürmektedir*(örneğin baş harfi A olan albümlerin elde edilmesi)*. Diğer yandan **GetTracks** metodu, **albumId** isimli parametre sayesinde, bir albüme bağlı olan şarkıların listesini **Name** alanına göre alfabetik sırada döndürmektedir. Böylece çok basit olsalardan kendi operasyonlarımızı tanımlamış bulunmaktayız. Solution' u bu haliyle derlediğimizde **Silverlight** uygulaması içerisinde yer alan ChinookDomainContext sınıfında uygun metod çağrılarının oluşturulduğunu açık bir şekilde görebiliriz.

```
        /// <summary>
        /// Returns an EntityQuery for query operation 'GetAlbumsByFirstLetter'.
        /// </summary>
        public EntityQuery<Album> GetAlbumsByFirstLetterQuery(string firstLetter)
        {
            Dictionary<string, object> parameters = new Dictionary<string, object>();
            parameters.Add("firstLetter", firstLetter);
            this.ValidateMethod("GetAlbumsByFirstLetterQuery", parameters);
            return base.CreateQuery<Album>("GetAlbumsByFirstLetter", parameters, false,
 true);
        }

        /// <summary>
        /// Returns an EntityQuery for query operation 'GetTracks'.
        /// </summary>
        public EntityQuery<Track> GetTracksQuery(int albumId)
        {
            Dictionary<string, object> parameters = new Dictionary<string, object>();
            parameters.Add("albumId", albumId);
            this.ValidateMethod("GetTracksQuery", parameters);
            return base.CreateQuery<Track>("GetTracks", parameters, false, true);
        }
```

DomainService sınıfını bu şekilde düzenledikten sonra sıra **Silverlight** uygulamasını geliştirmeye geldi. Bu amaçla MainPage.**XAML** içeriğini aşağıdaki gibi geliştirdiğimizi düşünelim.

```
<UserControl x:Class="SilverlightApplication5.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
    xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
    mc:Ignorable="d"
    d:DesignHeight="401" d:DesignWidth="640" xmlns:dataInput="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.Input"
xmlns:data="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data">

    <Grid x:Name="LayoutRoot" Background="White">
        <ComboBox Height="23" HorizontalAlignment="Left" Margin="11,62,0,0"
Name="cmbAlbums" VerticalAlignment="Top" Width="449"
SelectionChanged="cmbAlbums_SelectionChanged">
            <ComboBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel>
                        <TextBlock Text="{Binding Title}"/>
                    </StackPanel>
                </DataTemplate>
```

```
            </ComboBox.ItemTemplate>
        </ComboBox>
        <dataInput:Label Height="28" HorizontalAlignment="Left" Margin="12,46,0,0"
Name="label1" VerticalAlignment="Top" Width="120" Content="Albümler" />
        <StackPanel ScrollViewer.HorizontalScrollBarVisibility="Auto" Height="26"
HorizontalAlignment="Left" Margin="11,14,0,0" Name="pnlButtons"
VerticalAlignment="Top" Width="617" Background="#FFEBEBB2"
Orientation="Horizontal" UseLayoutRounding="True"></StackPanel>
        <data:DataGrid AutoGenerateColumns="True" Height="285"
HorizontalAlignment="Left" Margin="15,104,0,0" Name="grdTracks"
VerticalAlignment="Top" Width="613" Visibility="Visible">
            <data:DataGrid.Background>
                <ImageBrush
ImageSource="/SilverlightApplication5;component/Images/1243016_70835687.jpg"
/>
            </data:DataGrid.Background>
        </data:DataGrid>
    </Grid>
</UserControl>
```

Belkide dikkat çeken ilk nokta **ComboBox** kontrolü içerisinde bir **DataTemplate** kullanılmasıdır. Nitekim **DataTemplate** kullanarak **Title** alanını göstermek istediğimizi belirtmediğimiz durumda, **AlbumId** alanının getirildiğini görürüz. Bu bilgi son kullanıcı açısından çok anlamlı değildir. Ama tabiki **DataTemplate** içerisinde yer alan **StackPanel** elementinde istenilen kontroller kullanılarak daha fazla **Album** bilgisinin ComboBox' ın her bir öğesinde gösterilmeside sağlanabilir.

Gelelim MainPage için kod tarafına;

```
using System.Windows.Controls;
using System.Windows.Ria;
using SilverlightApplication5.Web;

namespace SilverlightApplication5
{
    public partial class MainPage
        : UserControl
    {
        ChinookDomainContext context= new ChinookDomainContext();

        void LoadButtons()
        {
            for (int i = 65; i < 91; i++)
            {
                Button btn = new Button();
```

```
            btn.Width = 20;
            btn.Height = 20;
            btn.Name = "Button_" + i.ToString();
            btn.Content = ((char)i).ToString();
            pnlButtons.Children.Add(btn);

            btn.Click += (o,e) =>
               {
                  grdTracks.Visibility = System.Windows.Visibility.Collapsed;
                  LoadOperation<Album> albumLoadOpt =
context.Load<Album>(context.GetAlbumsByFirstLetterQuery(btn.Content.ToString(
)));
                  cmbAlbums.ItemsSource = albumLoadOpt.Entities;
               };
         }
      }

      public MainPage()
      {
         InitializeComponent();

         LoadButtons();
         grdTracks.Visibility = System.Windows.Visibility.Collapsed;
      }

      private void cmbAlbums_SelectionChanged(object sender,
SelectionChangedEventArgs e)
      {
         if (e.AddedItems.Count > 0)
         {
            int albumId = ((Album)e.AddedItems[0]).AlbumId;
            LoadOperation<Track> tracks = context.Load<Track>(
               context.GetTracksQuery(albumId),
               (load) =>
               {
                  grdTracks.Visibility = System.Windows.Visibility.Visible;
               },
               null
               );
            grdTracks.ItemsSource = tracks.Entities;
         }
      }
   }
}
```

MainPage yapıcı metodu içerisinde **Button** bileşenlerinin oluşturulması işlemi gerçekleştirilmektedir. Bu işlem sırasında her **Button** bileşeni için **Click** olayının yüklenmesi sağlanmaktadır. Dikkat edileceği üzere **Click** olay metodu içerisinde, o anda basılan düğmenin **Content** bilgisinden yararlanılarak bir **LoadOperation** oluşturulur. Ayrıca, **ChinookDomainContext** nesne örneği üzerinden yapılan **GetAlbumsByFirstLetterQuery** metodu ile gerekli sorgunun elde edilmesi sağlanır. Bundan sonra ise **ComboBox** kontrolünün**ItemsSource** özelliğine gerekli veri bağlama işlemi yapılır. **ComboBox** kontrolünde bir öğenin seçilmesi halinde devreye giren **SelectionChanged** metodunda ise bu kez **GetTracks** servis metodunun çalıştırılması için gerekli işlemler yapılmaktadır. Yanlız bu seferki kullanımda **ChinookDomainContext** referansına ait **Load** metodunun ikinci parametresine dikkat edilmelidir. Bu parametre söz konusu **Load** işlemi tamamlandıktan sonra devreye girecek bir metodu işaret edecek **Action<T>** tipinden bir **temsilcidir(delegate).** Burada **Load** işlemi tamamlandığında görünür olmayan **DataGrid** kontrolünün görünür hale getirilmesi için sembolik bir işlem yapıldığını belirtebiliriz. Ancak ana fikir, **Load** operasyonunun tamamlanması ile kontrolü ele alabileceğimiz bir metodun işaret edilebiliyor olmasıdır. Bu kodlamanın ardından **DataGrid** kontrolünün **ItemsSource** özelliğine gerekli veri bağlama işleminin yapılması yeterlidir. Uygulamayı bu noktadan sonra teste çıkartabiliriz. Sonuç olarak yazımızın başında belirttiğimiz ekran görüntüsüne benzer sonuçları elde ebiliyor olmamız gerekmektedir.

Peki neler öğrendik?

- **WCF RIA Service'** lerinde sihirbaz yardımıyla **Entity Data Model'** den otomatik olarak üretilen **Domain Service** sınıf metodları yerine kendi sorgulama metodlarımızı kullanabileceğimizi, var olanları istersek güncelleştirebileceğimizi,
- İstemci tarafında, **Domain Service** sınıfı içerisindeki operasyonlara yapılacak olan çağrılarda **Callback** metodlarının değerlendirilerek yükleme tamamlandıktan sonrasını anlayıp bazı işlemler yaptırabileceğimizi,
- **Silverlight** uygulaması içerisindeki kontrollerde **DataTemplate** kullanarak, servis tarafından çekilen **Entiy** içeriklerinin sadece istediğimiz alanlarının kullanılabileceğini,
- **Silverlight** tarafında dinamik olarak kontrollerin nasıl üretilip ilgili elementlere eklenebileceğini,

öğrendik.

hatta **Background** özelliğine **Picture** ekleyebileceğimizi ve bu sayede daha hoş bir görüntü sunabileceğimizi farkettik demek istesemde, bu önemsenecek bir mevzu değildir.

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

SilverlightApplication5.rar (5,28 mb)

WCF RIA Services - Bir Merhaba Diyelim (2009-11-24T10:56:00)

wcf ria services,.net ria services,wcf,wcf eco system,

Merhaba Arkadaşlar,

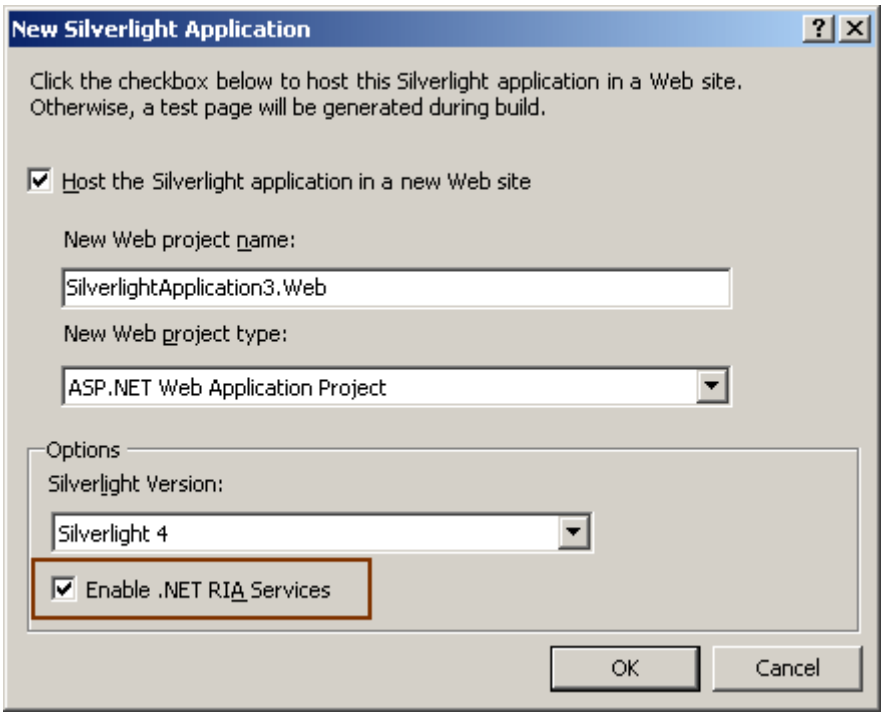
Yağmurlu günler ve kış geldikçe bir blog yazarının ilham gelmesini bekleyerek zaman geçirmesine hiç mi hiç gerek yoktur? *(Tabi yandaki resimde görülen köprüye o açıdan uzun uzun baktığınızda bloğunuza yazacak çok güzel fikirler edinebilirsiniz)* Aslında bu gün itibariyle blog yazısına konsantre olmak için gerekli şartlar zaten mevcuttur. Bir adet bilgisayar, internet bağlantısı, gerekli referans kitaplar*(eğer konu ile ilişkili bulunabilirse)*, kapalı bir hava, güzel bir müzik ve onu kulaktan beyin hücrelerine kaliteli bir şekilde aktaracak kulaklıklar ile yağmurlu bir gün. Tabi insan bazen bloğuna yazarken bir solukta işe yarayacak eserlerde çıkartmak isteyebilir. İşte ben bu yüzden **Hello World** uygulamalarını çok severim. Zaten **Microsoft** pek çok eğitim materyalinde detaya girmeden önce, basit bir **Hello World** uygulaması ile **"aslında biz bu konsept ile neyi yapabiliyoruz"** sorusuna cevap vererek başlamayı tercih etmiştir,etmektedir. örneğin çok çok eski **Xml Web Services** eğitiminde böyle bir giriş bulunmaktadır. önce yazılmış olan bir **Xml Web Service'** inin kullanıdırıması öğretilir. Dikkat edin eğitimin amacı Xml Web Service' lerini geliştirmek. öyleyse bu gün menümüzde ne var bir bakalım.

Takip eden arkadaşlarımız bir süre önce Microsoft PDC 2009 konferansının gerçekeştiğini ve pek çok yeniliğin tanıtıldığını bilirler. Benim açımdan önemli olan gelişmelerden biriside pek çok teknolojinin adının değişmesi olmuştur. özellikle **WCF** tabanlı olarak geliştirilen pek çok yardımcı servis modelinin adı, benimde istediğim ve beklediğim gibi tekilleştirildi. Buna göre **Ado.Net Data Services'** ler **WCF Data Services** ve **Rich Internet Application'** lar için n-tier sorununu servis bazlı olarak kolayca aşmamızı sağlayan **.Net RIA Services'** da **WCF RIA Services** olarak isim değiştirmiştir. Aslında bu bilgilerden yararlanıldığında bir **WCF** eko sisteminin oluşturulduğunu ve içerisinde **Workflow Services, Data Services, RIA Services, Web HTTP Services** ve **Core Services** gibi kavramların yer aldığını ifade edebiliriz. Şimdilik bu eko sistemin içerisine çok fazla girmeyeceğiz. Bu yazımızdaki amacımız**Beta'** sı yayımlanan **WCF RIA Services'** e **Bir Merhaba** diyebilmek.

Bildiğiniz üzere **RIA Service'** leri ile **Silverlight** gibi **Rich Internet Application** istemcilerinin **n-tier** modeline göre geliştirilebilmesi oldukça

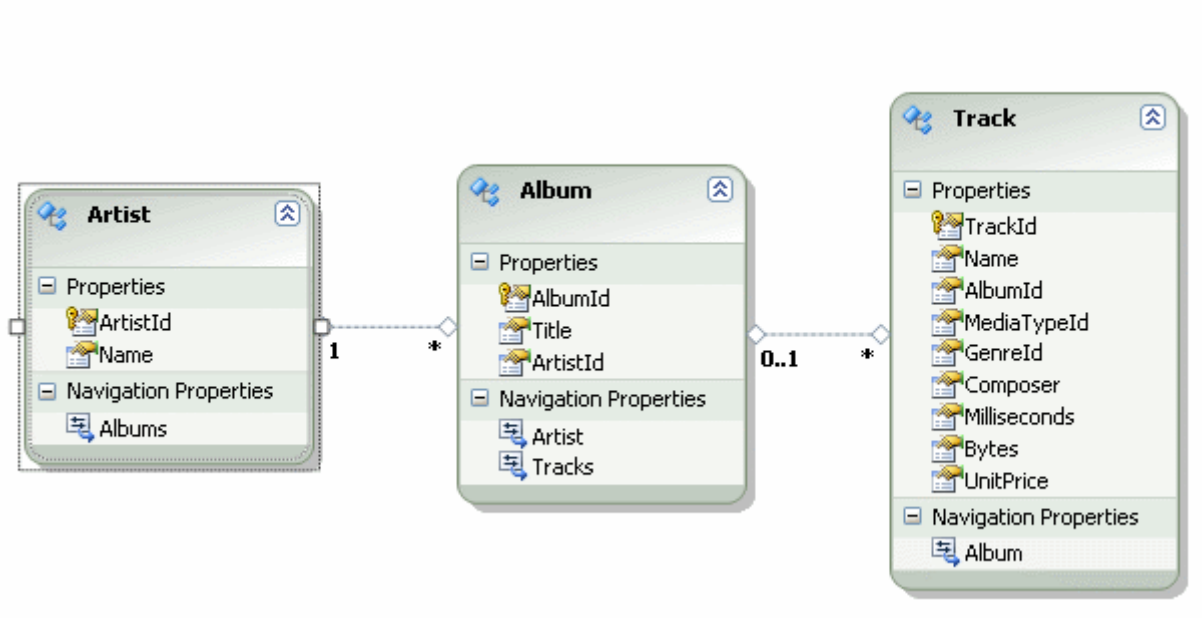
kolaylaştırılmaktadır. özellikle **Silverlight 4.0** ile **Visual Studio 2010** tarafına getirilen yeni özellikler*(örneğin Windows uygulamalarındaki gibi Data Sources kısmının kullanılabilmesi ve bu sayede sürükle-bırak desteği)* ve **WCF RIA Services** bir araya geldiğinde oldukça güzel sonuçlar ortaya çıktığını söyleyebiliriz. Aslında söz konusu kolaylıkları görsel derslerimizde incelemeye çalışçağımız şimdiden belirtmek isterim. Ancak hemen öncesinde çok basit bir **Hello World** uygulaması yaparak, **Silverlight** uygulaması içerisinden **WCF RIA Service'** lerin nasıl kullanılabileceğini görelim. örneğimizi mümkün olduğunda basit bir şekilde gerçekleştireceğiz. **WCF RIA Service'** imiz arka planda **Entity Data Model'** i kullanıyor olacak. Bu amaçla kaynak bir veritabanını SQL üzerinde ele alacağız. Ancak bu sefer AdventureWorks değil 🙂 Yihaaaa!!! Bu kez örnek olarak Chinook isimli **Codeplex** üzerinden yayınlanan ve pek çok MVP, Microsoft çalışanı ve profesyonlin konu anlatımlarında kullandığı açık kaynak veritabanını ele alacağız. Haydi bakalım parmakları sıvayalım. 😊

İlk olarak örneğimizi **Visual Studio 2010 Ultimate Beta 2** üzerinde geliştirdiğimizi ifade etmek isterim. Diğer yandan Silverlight 4.0 Beta ve WCF RIA Services kurulumlarının da yapılmış olması gerektiğini hatırlatalım. Eğer bu kurulumlar tamamlandıysa işe basit bir **Silverlight Application** projesi oluşturarak başlayabiliriz. Oluşturma işlemi sırasında, **Enable .NET RIA Services** seçeneğini etkinleştirmemiz gerekmektedir ki WCF RIA Service' leri kullanabilelim*(Tabi isim değişse de IDE üzerinde henüz değişmemiş olduğu gözden kaçmamalıdır. Belkide isim değişmez. Immm...Bilemiyorum. Değişirse iyi olur tabi)*

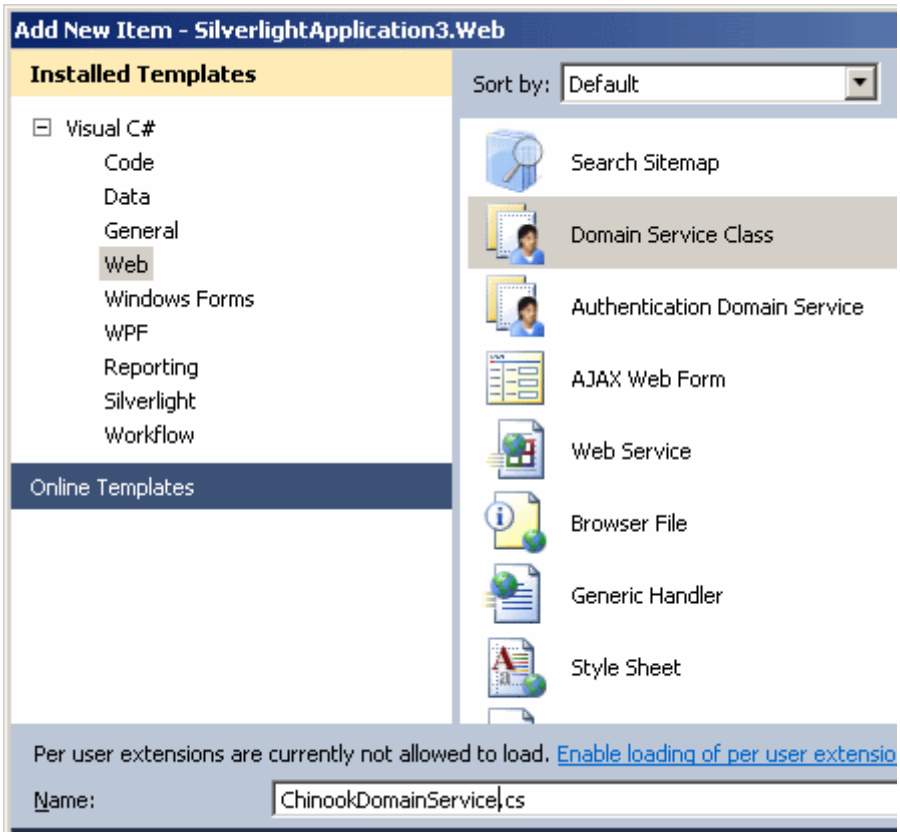


Projemizin bu şekilde oluşturulması sonrasında **Silverlight** uygulamasının host edileceği ayrı bir **Web** uygulamasının da oluşturulduğunu görebiliriz. Söz konusu **Web** uygulaması hem**Entity Data Model'** i hemde **DomainService** sınıfını içerecektir. Veri modeli

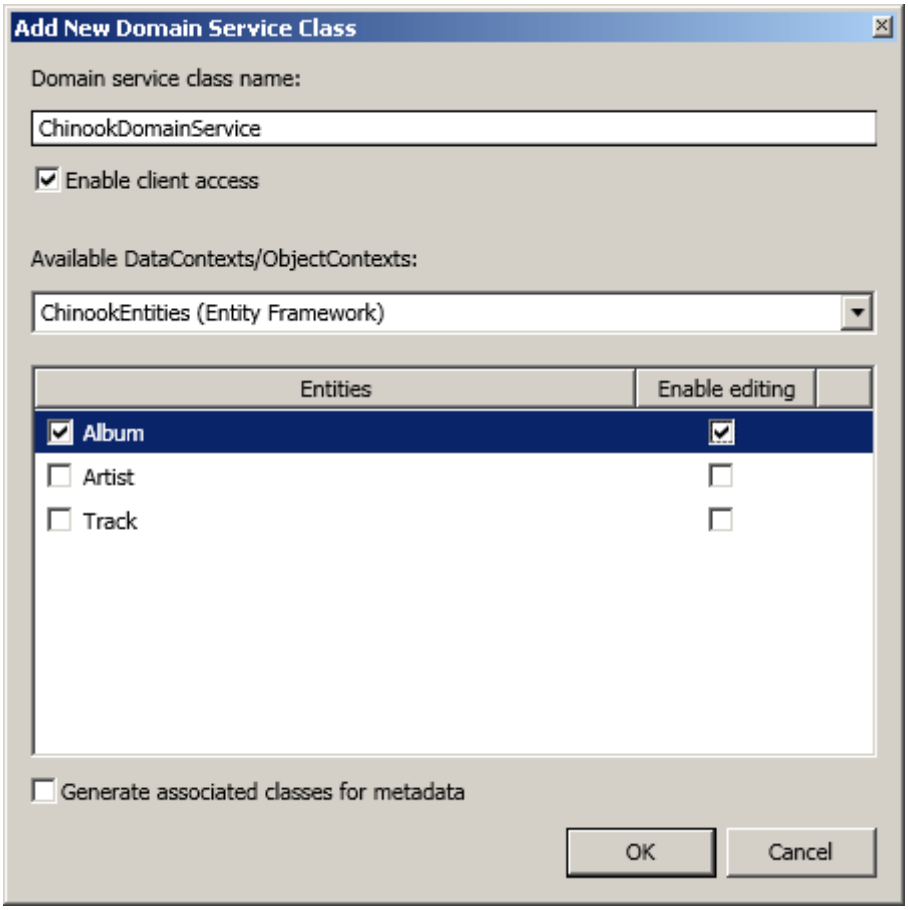
için **ChinookModel** isimli yeni bir **Ado.Net Entity Data Model** öğesini **Web** projesine ekleyerek devam edelim. Başlangıç için aşağıdaki şekilde görülen **Entity** tiplerinin oluşturulmasını sağlayabiliriz.



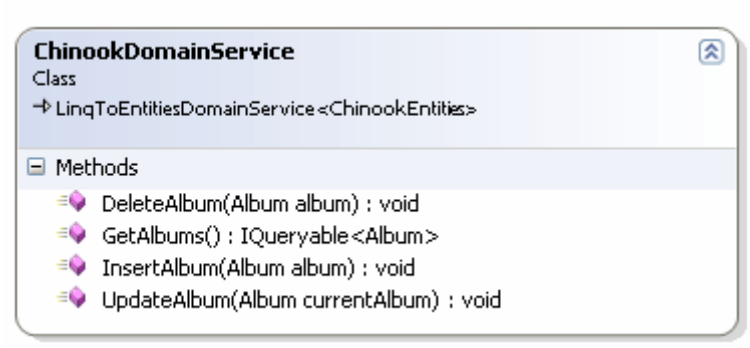
Web uygulaması tarafında artık bir veri modelimiz bulunmaktadır. **Silverlight** uygulamasına veri modelinden hizmet sunabilmek için***(CRUD-CreateReadUpdateDelete** işlemleri)*, yine **Web** projesine bir**DomainService** sınıfı ekleyerek devam etmemiz gerekmektedir. Bunun için **Web** sekmesinden **Domain Service Class** öğesini seçmemiz yeterlidir.



Karşımıza çıkacak olan **Wizard** adımlarında, kullanacağımız veri modelini ve ilgili **Entity** tiplerini seçerek ilerliyor olacağız. Burada hemen bir ipucunu vermek isterim; **Entity Data Model**oluşturulduktan sonra projeyi derlemezsek, **Available Data Contexts/ObjectContexts ComboBox'** ında herhangibir içeriğin çıkmadığı görülecektir. Derleme işleminden sonra ise aşağıdaki şekilde görüldüğü üzere sadece **Album Entity** tipini seçerek ilerleyelim.

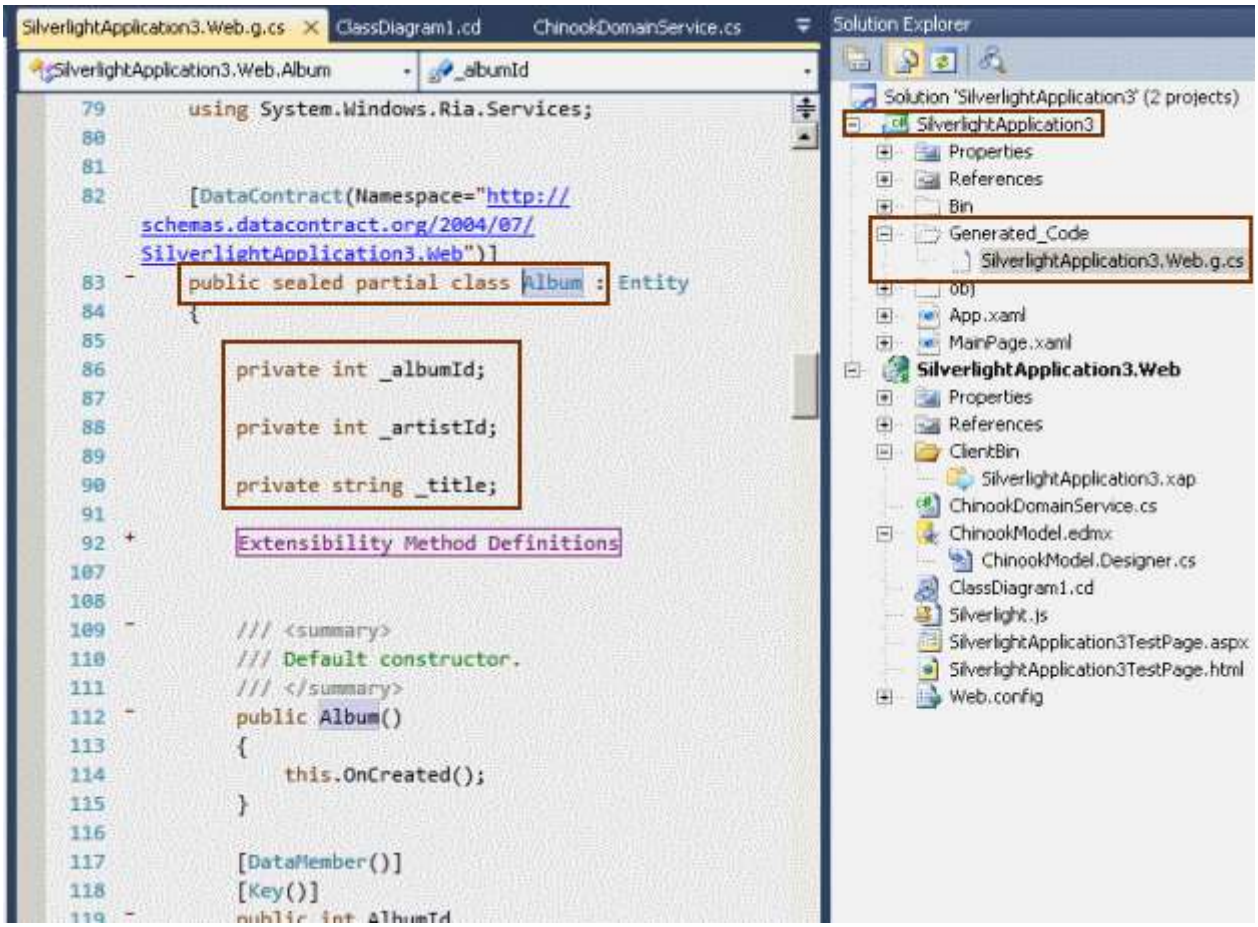


Album Entity' si üzerinden örneğimizde veri ekleme, çıkartma ve silme işlemlerini bu yazımızda yapmıyor olacağız ancak arka planda oluşturulan servis sınıfı içerisinde ne gibi kodlamalar yapıldığını görmekte yarar var. Bu adımı geçtikten sonra **Web** projesi tarafında üretilen **ChinookDomainService** sınıfını kısaca incelemenizi tavsiye ederim. İlk etapta sınıf diagramına baktığımızda aşağıdaki içeriğe sahip olduğunu görebiliriz.



Dikkat edileceğiz üzere **LinqToEntitiesDomainService<ChinookEntities>** tipinden türeyen sınıf içerisinde **CRUD** işlemleri için üretilmiş dört metod bulunmaktadır. **GetAlbums** isimli metod**IQueryable<Album>** tipinden bir referans döndürmektedir. Buda istemci tarafından **LINQ** ifadeleri ile sorgulanabilir bir içeriğin yakalanabileceğini göstermektedir. Ayrıca **Delete, Insert** ve **Update**işlemlerinde parametre olarak **Album** tipinden nesne örneklerine ihtiyaç duyulmaktadır ki bunların

tamamı **Silverlight** uygulaması tarafından da kullanılabilir.
Nitekim **Silverlight** uygulamasındaki gizli dosya içeriklerine bakıldığında gerekli **Entity** tipinin bu tarafa da taşındığı görülebilir.



Web tarafında yer alan **DomainService** sınıfı içerisinde şimdilik **GetAlbums** metodunu kullanıyor olacağız. Nitekim ilk amacımız **Album** listesini **Silverlight** tarafındaki bir **DataGrid**kontrolüne çekmek olacak.

```
[EnableClientAccess()]
   public class ChinookDomainService
      : LinqToEntitiesDomainService<ChinookEntities>
   {
      public IQueryable<Album> GetAlbums()
      {
         return this.ObjectContext.Albums;
      }
// KOD DEVAM EDİYOR...
```

Burada önemli olan noktalardan birisi sınıfın başında istemci erişimine izin verildiğini belirten **EnableClientAccess** isimli **niteliğin(Attribute)** kullanılmış olmasıdır. Diğer yandan**GetAlbums** metodu aslında **ObjectContext** içerisinde **Albums** özelliği üzerinden album içeriğini döndürmektedir. Dolayısıyla **LINQ** sorgusu ile **Where** gibi operatorler

kullanılabilir ve metod istendiğinde parametrik olarak çalıştırılabilir*(Bu gibi pek çok farklı konuyu görsel derslerimde ele almayı planladığım için burada fazla detaya girmeyeceğiz).* Artık **Silverlight**uygulamamızı tasarlamaya başlayabiliriz. Bu amaçla **MainPage.xaml** içeriğini aşağıdaki gibi geliştirdiğimizi düşünelim.

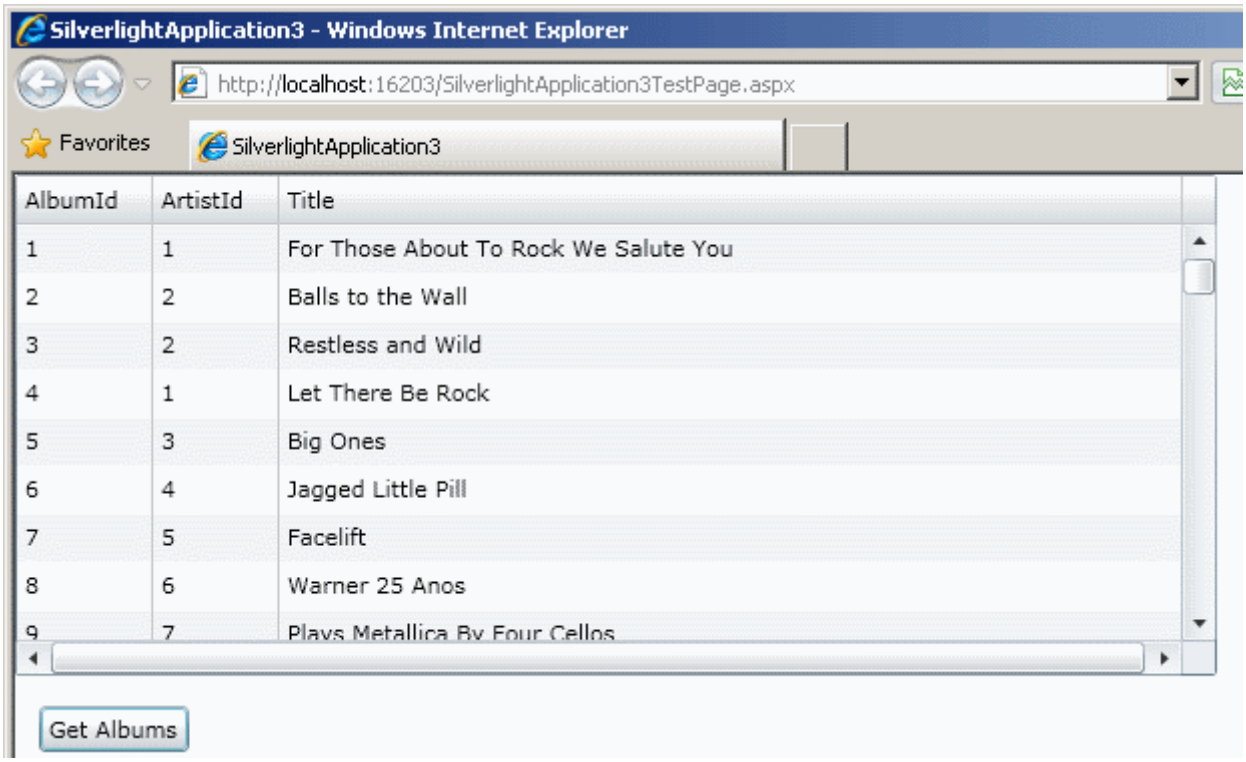
```
<UserControl x:Class="SilverlightApplication3.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
   xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
   mc:Ignorable="d"
   d:DesignHeight="300" d:DesignWidth="600" xmlns:data="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data">

   <Grid x:Name="LayoutRoot" Background="White">
      <data:DataGrid AutoGenerateColumns="True"
                  Height="250"
                  HorizontalAlignment="Left"
                  Name="grdAlbums"
                  VerticalAlignment="Top"
                  Width="600" />
      <Button Content="Get Albums"
            Height="23"
            HorizontalAlignment="Left"
            Margin="12,265,0,0"
            Name="btnGetAlbums"
            VerticalAlignment="Top"
            Width="75"
            Click="btnGetAlbums_Click" />
   </Grid>
</UserControl>
```

Burada küçük bir ipucu daha vermek isterim;
başlangıçta **DataGrid** bileşeninin **AutoGenerateColumns** özelliği **False** değerine sahiptir. Bu nedenle kodlama doğru bir şekilde yapılsa dahi içeriğin **DataGrid** kontrolüne aktarılmadı görülecektir. Dolayısıyla ilgili niteliğin değerini bu örnek için **True** yapmayı unutmamalıyız. **btnGetAlbums** isimli düğmeye basıldığında tüm Album listesinin bir DataGrid içerisine doldurulmasını sağlamak için, kod tarafını aşağıdaki gibi geliştirmemiz yeterlidir.

```
using System.Windows;
using System.Windows.Controls;
using System.Windows.Ria;
using SilverlightApplication3.Web;
```

```
namespace SilverlightApplication3
{
   public partial class MainPage
      : UserControl
   {
      ChinookDomainContext context;

      public MainPage()
      {
         InitializeComponent();
         // DomainContext nesnesi örneklenir
         context = new ChinookDomainContext();
      }

      private void btnGetAlbums_Click(object sender, RoutedEventArgs e)
      {
         // Asenkron Load operasyonunu gerçekleştirecek tip tanımlanır
         // Tip parametre olarak Album' leri çekmek için gerekli sorguyu üreten
GetAlbumsQuery metodunu kullanmaktadır.
         LoadOperation<Album> loader =
context.Load<Album>(context.GetAlbumsQuery());
         grdAlbums.ItemsSource = loader.Entities; // Load operasyonu tarafından elde
edilen Entite yüklenir
      }
   }
}
```

Ve çalışma zamanındaki sonuç;



Tabi daha ne kolaylıklar bar Visual Studio tarafında bir bilseniz. İştahınızı kabartmış olabilirim ama devamı için görsel videolarımı bekleminizi öneririm. Umarım vakit bulup çekeceğim. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

SilverlightApplication3.rar (1,28 mb)

Microsoft.Net Services - Service Bus için REST Tabanlı Hello World (2009-11-20T08:45:00)

windows azure,service bus,wcf,rest,

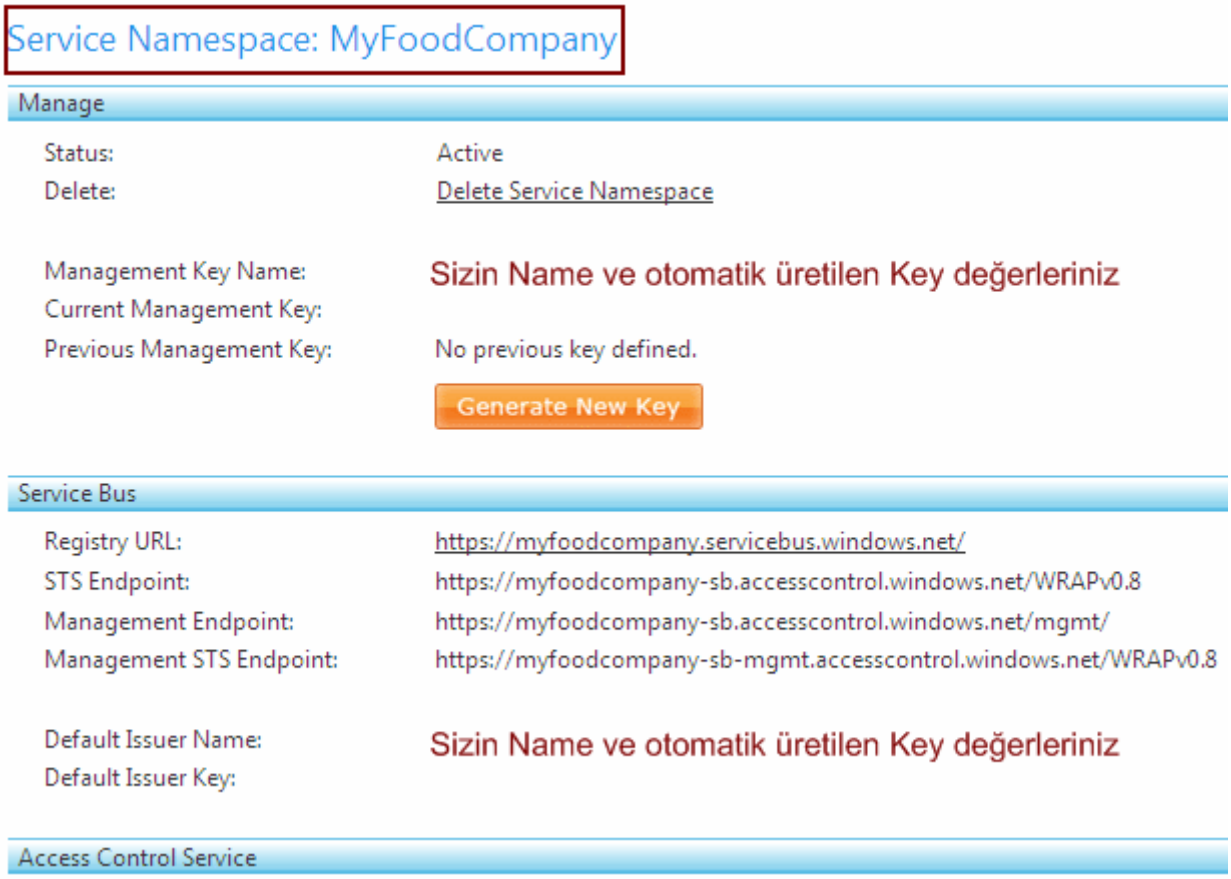
Merhaba Arkadaşlar,

Bir önceki yazımızda **Microsoft.Net Services** alt yapsının önemli parçalarından birisi olan **Service Bus** hizmetini incelemeye çalışmış ve basit bir **Hello World** uygulama koleksiyonu geliştirmiştik. Bu yazımızda ise **REST** bazlı geliştirilen bir **WCF** servisine herhangibir tarayıcı yardımıyla **HTTP Get** metoduna göre, **Service Bus** üzerinden nasıl ulaşabileceğimizi incelemeye çalışacağız.

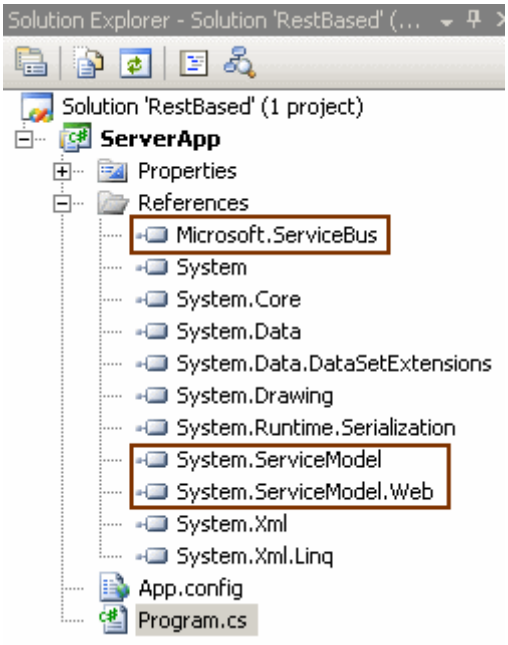
REST bazlı modelde bilindiği üzere Web tabanlı olarak yayınlanan servislere **HTTP** protokolünün **Get,Post,Put,Delete** gibi metodları yardımıyla erişilebilmektedir. Bu **URL** bazlı erişim sayesinde herhangibir tarayıcı uygulamanın söz konusu servis operasyonlarını kullanabilmesi mümkündür. üstelik istemciler arada bir **proxy** nesnesine ihtiyaç duymadan doğrudan**HTTP** taleplerini gönderebilir. Bunlara ek olarak birde servis operasyonlarının **Syndication(RSS,Atom gibi)** tabanlı içerik

yayınlayabilme kabiliyetleri eklendiğinde Web programlama modeline uygun bir hizmet üretiminin gerçekleştirilebildiği gözlemlenecektir.

WCF mimarisi, **.Net Framework 3.5** sürümünden bu yana Web programlama modelini desteklemektedir. Bu nedenle **REST(Representational State Transfer)** bazlı **WCF** servisleri kolayca geliştirilebilir. Bizde bu günkü örneğimizde sunucu tarafında, **RSS 2.0** formatında içerik yayınlaması yapan ve **REST** tabanlı olarak çalışabilen bir **WCF** servisini geliştirecek ve buna olan istemci erişimleri için **Service Bus'** tan yararlanacağız. Tahmin edeceğiniz üzere **Microsoft.Net Services** üzerinde örneğimiz için bir **Service Namespace'** i oluşturulması gerekmektedir. Ben bu günkü örneğimiz için günlük öneri yemek listesini sunacak bir hizmete uygun aşağıdaki isimlendirmeyi kullanmayı tercih ettim.



Bundan sonraki ilk adımımız servis uygulamasını geliştirmek olacaktır. Söz konusu uygulama **WCF Web programlama modelini(Web Programming Model)** kullanacağından ve **Service Bus**üzerinden iletişimi sağlayacağından aşağıdaki şekilde görülen **Microsoft.ServiceBus, System.ServiceModel.Web** ve **System.ServiceModel assembly**' larını referans etmelidir.



Gelelim sunucu uygulama tarafındaki kodlarımıza;

```
using System;
using System.Collections.Generic;
using System.ServiceModel;
using System.ServiceModel.Syndication;
using System.ServiceModel.Web;
using Microsoft.ServiceBus;

namespace ServerApp
{
   // Servis sözleşmesi
   [ServiceContract(Namespace =
"http://www.buraksenyurt.com/MyFoodCompany")]
   public interface IDailyFoodListContract
   {
      [OperationContract]
      [WebGet] // Http Get taleplerine cevap verecek bir operasyon olduğunu belirtiyoruz
      Rss20FeedFormatter GetDailyFoodList(string Day);
   }

   // Servisi uygulayan tip
   [ServiceBehavior(Name = "DailyFoodListService", Namespace =
"http://www.buraksenyurt.com/MyFoodCompany")]
   public class DailyFoodListService
      :IDailyFoodListContract
   {
      #region IDailyFoodListContract Members
```

```
        // GetDailyFoodList metodu RSS 2.0 Formatında basit bir Syndication içeriği
döndürmektedir.
        public Rss20FeedFormatter GetDailyFoodList(string Day)
        {
            // öncelikli olarak Feed oluşturulur
            SyndicationFeed foodFeed = new SyndicationFeed();
            foodFeed.Id=String.Format("Day_{0}",Day);
            foodFeed.Title = new TextSyndicationContent("Günlük Yemek Listesi");

            // Feed içerisindeki Item listesi hazırlanır
            List<SyndicationItem> foodItems = new List<SyndicationItem>();
            foodItems.Add(new SyndicationItem( "Aperatifler","Patlıcanlı Musakka",
new Uri("http://myfoodcompany/Food/Musakka"), "10001", DateTime.Now));
            foodItems.Add(new SyndicationItem("çorbalar", "Mercimek çorbası", new
Uri("http://myfoodcompany/Food/MercimekCorba"), "12034", DateTime.Now));
            foodItems.Add(new SyndicationItem("Ana Yemekler", "Mozeralla Peynirli
Makarna", new Uri("http://myfoodcompany/Food/MakarnaMozeralla"), "10025",
DateTime.Now));

            // Item listesi feed' e eklenir.
            foodFeed.Items=foodItems;

            // RSS 2.0 formatındaki Feed içeriği üretimi için SyndicationFeed nesne örneği
Rss20FeedFormatter sınıfının yapıcı metoduna parametre olarak geçirilir.
            return new Rss20FeedFormatter(foodFeed);
        }

        #endregion
    }

    //// Kanal arayüzü
    //public interface IDailyFoodListChannel
    //    : IDailyFoodListContract, IClientChannel
    //{
    //}

    class Program
    {
        static void Main(string[] args)
        {
            Uri address = ServiceBusEnvironment.CreateServiceUri("https",
"MyFoodCompany", "Foods");

            // REST bazlı bir WCF Servisi host işlemi gerçekleştirileceğinden WebServiceHost
nesne örneğinden yararlanılır
```

```
        WebServiceHost host = new WebServiceHost(typeof(DailyFoodListService),
address);
        host.Open(); // Servis açılır

        Console.WriteLine("Servis açıldı. Servis durumu {0}\nService Adresi {1}",
host.State,address.ToString());
        Console.WriteLine("Operasyon Adı : GetDailyFoodList\n");
        Console.WriteLine("çıkmak için bir tuşa basınız");
        Console.ReadLine();

        host.Close(); // Servis kapatılır
      }
   }
}
```

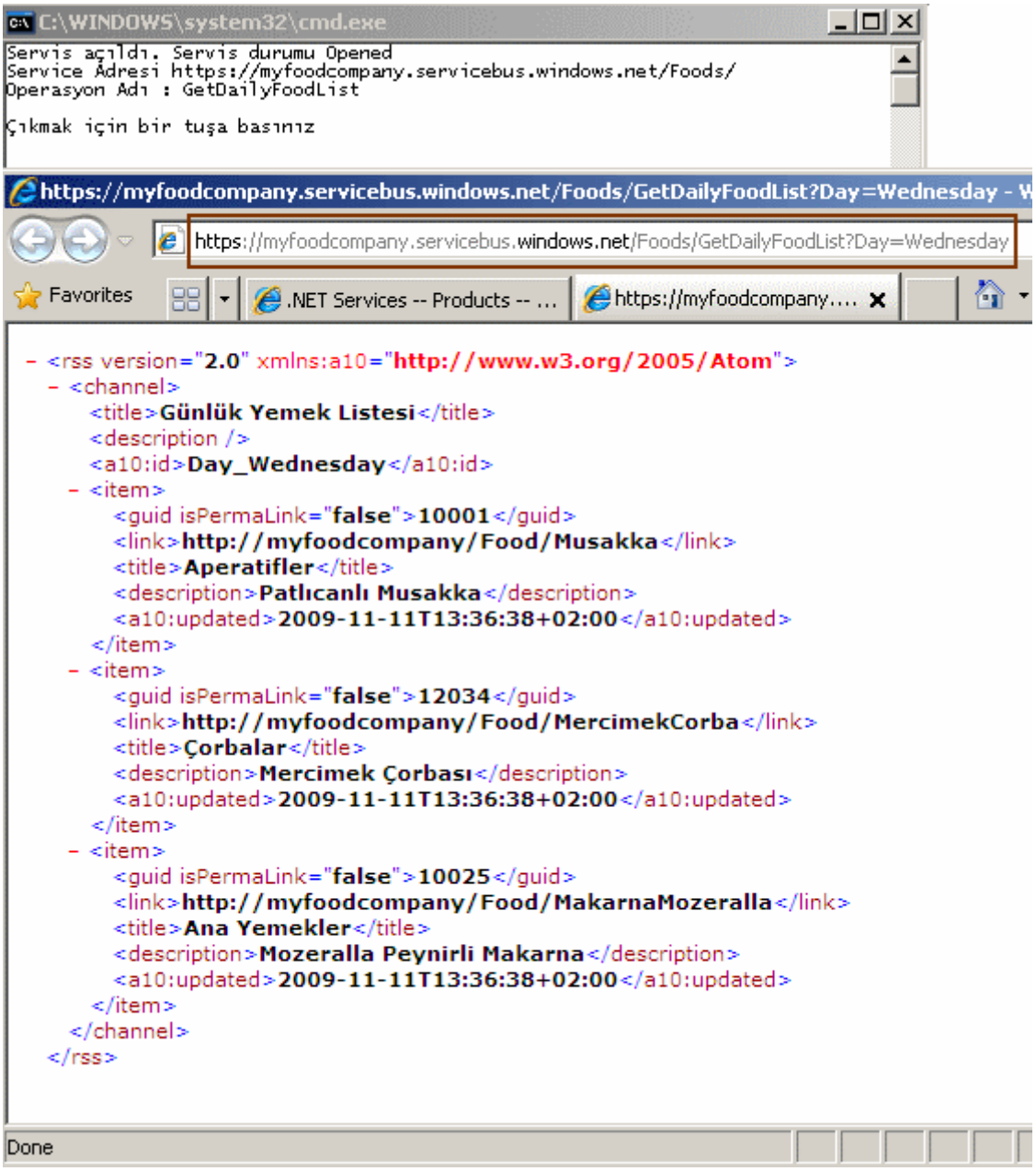
WCF Servis sözleşmemiz içerisinde yer alan **GetDailyFoodList** operasyonu, **string** parametre alıp **RSS 2.0** formatında içerik üreten basit bir fonksiyonellik sunmaktadır. Bu operasyonun **HTTP Get**taleplerine cevap verebilmesi istendiğinden **WebGet** niteliği ile imzalanması gerekmektedir. Diğer yandan WCF çalışma zamanının **REST** tabanlı talepleri değerlendirmesi istendiğinden**ServiceHost** yerine **WebServiceHost** tipinin kullanılması gerektiğide aşikardır. Bunun dışında servis operasyonumuz tamamen deneysel olarak günlük öneri yemek listesini üretmektedir. Pekala dünya üzerindeki restoran zincirlerine veya herhangibir evin mutfağındaki dokunmatik ekranlara bu tip bir servis yardımıyla hizmet götürüyor olsaydık daha fazla parametre alacak ve veri kaynağı olarak **Azure** üzerinde yer alacak bir **Database Service'** ini kullanacak bir operasyon da geliştirebilirdik. Kimbilir belki ilerleyen zamanlarda bu tip komple bir örneğide geliştirme fırsatımız olur. 😉

Servis uygulamamız tarafında çalışma zamanı konfigurasyon bilgileri içinde **App.config** dosyasının aşağıdaki gibi düzenlenmesi yeterlidir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <webHttpRelayBinding>
        <binding name="DailyFoodListServiceBinding">
          <security relayClientAuthenticationType="None" />
        </binding>
      </webHttpRelayBinding>
    </bindings>
    <services>
      <service behaviorConfiguration="DailyFoodServiceBehavior"
name="ServerApp.DailyFoodListService">
        <endpoint address="" behaviorConfiguration="credentialBehavior"
```

```
binding="webHttpRelayBinding"
bindingConfiguration="DailyFoodListServiceBinding" name="RelayEndpoint"
contract="ServerApp.IDailyFoodListContract" />
    </service>
  </services>
  <behaviors>
    <endpointBehaviors>
      <behavior name="credentialBehavior">
        <transportClientEndpointBehavior credentialType="SharedSecret">
          <clientCredentials>
            <sharedSecret issuerName="Sizin Issuer Name değeriniz"
issuerSecret="Sizin için üretilen Key değeri"/>
          </clientCredentials>
        </transportClientEndpointBehavior>
      </behavior>
    </endpointBehaviors>
    <serviceBehaviors>
      <behavior name="DailyFoodServiceBehavior">
        <serviceDebug httpHelpPageEnabled="false" httpsHelpPageEnabled="false"
includeExceptionDetailInFaults="True" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
 </system.serviceModel>
</configuration>
```

Konfigurasyon içeriğinde dikkat çekici noktaların başında kullanılan bağlayıcı tip gelmektedir(**webHttpRelayBinding**). Bu gereklidir nitekim **Service Bus** ile iletişim kurulacaktır. Diğer yandan **clientCredentials** elementi içerisinde, **Service Bus** projemizdeki ilgili **Namespace** için üretilen **Issuer Name** ve **Issuer Key** değerleri yer almalıdır. Bundan sonrası son derece kolaydır. Servis uygulaması çalıştırıldıktan sonra tarayıcı üzerinden **GetDailyFoodList** operasyonuna yapılan çağrı aşağıdaki örnek ekran görüntüsünde olduğu gibi karşılanacaktır.



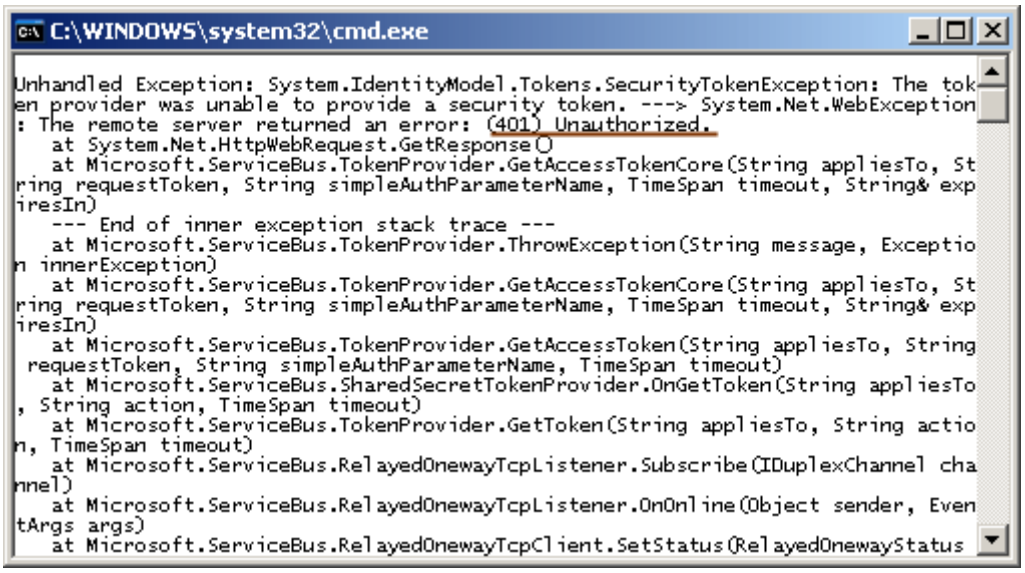
Görüldüğü üzere **Wednesday** için bir talepte bulunulmuş ve bir içerik elde edilmiştir. Bu örnekte en çok dikkat edilmesi gereken nokta ise operasyon talebinin yapıldığı adres bilgisidir.

https://myfoodcompany.servicebus.windows.net/Foods/**GetDailyFoodList?Day=Wednesday** Volaaa!!! 😃 Parametreyi nasıl gönderdiğimize, https protokolüne göre bir URL bilgisi yazıldığına dikkat edilmelidir.

Tabiki işin sağlamasını yapmak amacıyla servis uygulamasını kapatıp aynı talebi göndermeyi deneyebilirisiniz. Bu durumda aşağıdaki ekran görüntüsü ile karşılaşılacaktır.

Yanlız burada dikkat edilmesi gereken noktalardan birisi de, geriye bir istisna(Exception) mesajının gönderilmeyişidir. Bu çok doğaldır nitekim servis uygulaması çalışmıyor olsa bile Service Bus üzerinde gelen talebin değerlendirildiği bir hizmet bulunmaktadır.

örneği geliştirirken meraktan yaptığım bir testte yanlış **Issuer Key** değeri göndermek oldu. Böyle bir vakada gönderilen **HTTP Get** talebi sonrası aşağıdaki ekran görüntüsü ile karşılaşılacaktır.



Görüldüğü üzere **401 Unauthorized** hatasını aldık. Bir başka deyişle **Credential** bilgilerimiz doğrulanmadı.

Bu hızlı örnek ile **Service Bus** hizmetinden **REST** bazlı olarakta nasıl yararlanabileceğimizi görmüş olduk. Bir önceki yazımızda kullandığımız örnekte geliştirdiğimiz istemci uygulamada hatırlayacağınız üzere, sözleşme tanımlamış, kanal oluşturmuş ve TCP bazlı olan iletişim üzerinden kanal bazlı metod çağrısı ile isteğimizi **Service Bus**tarafına göndermiştik. **REST** modelinde ise bir istemci geliştirmediğimizi sadece bir **URL** kullandığımızı hatırlatmak isterim. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

RestBased.rar (25,30 kb)

Screencast - Asp.Net 4.0 - Session State Compression [Beta 2] (2009-11-19T09:45:00)

asp.net 4.0 beta 2,asp.net,

Merhaba Arkadaşlar,

Web uygulamalarının **Stateless** olmaları nedeniyle, istemcilerin sayfalar arasında gezerken takip etmek istedikleri verileri kolayca taşıyabilmeleri için ele alınan yöntemlerden birisi de **Session** kullanımıdır. Sunucu tarafında **Session** içerikleri 3 farklı şekilde yönetilebilmektedir. Varsayılan olarak **In-Proc** mod geçerlidir. Yani **Asp.Net Worker Process** tarafından web uygulamasına ait bellek alanında saklanmaktadır. Ancak **Web Farm** kullanıldığı vakalarda **Session** bilgilerinin arka tarafta yer alan başka sunucularda yönetilmesi de sağlanabilir. Bu noktada Sql veritabanı veya State Server isimli **Windows Service** seçenekleri geçerlidir. özellikle **In-Proc** dışında kullanılan modlarda serileşen içeriğin çok büyük boyutlarda olmasına karşın **Asp.Net 4.0** ile birlikte sıkıştırılabilme opsiyonunun getirildiği görülmektedir. Bunun için Web.config dosyasında yapılacak küçük bir ekleme yeterlidir. Devamı görsel dersimizde

Süre : 14:53

Yeni görsel derslerde görüşmek dileğiyle hepinize mutlu günler dilerim.

Asp.Net 4.0 - Heryerde Cache [Beta 2] (2009-11-18T13:20:00)

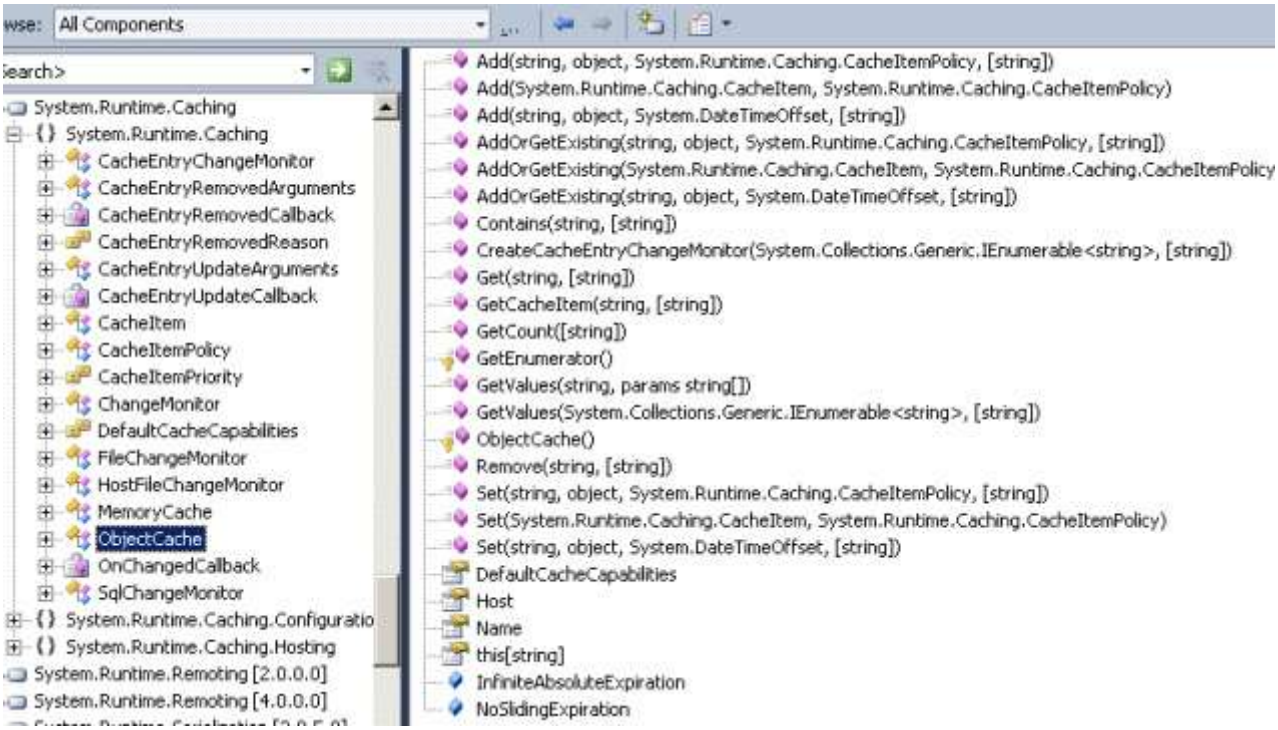
asp.net 4.0 beta 2,asp.net,


Merhaba Arkadaşlar,

Kronometrelerinizi hazır edin! Bu yazımızda **Absolute** ve **Sliding Expiration** modelinde ön bellekleme işlemlerini **Windows** tabanlı bir uygulama üzerinde gerçekleştiriyor olacağız. Durun bir dakika...**Windows** mu? Evet evet yanlış duymadınız **Windows**. Aslında başlıktaki konu ile tamamen tezat bir durum. Gerçekten de öyle mi acaba? Gelin şu meseleyi açıklığa kavuşturalım. 🙂

ön bellekleme işlemleri ile performans arttırımı çoğunlukla web uygulamalarında akla gelen bir konudur. Ancak gerçek hayat uygulamaları sadece Web tabanlı değildir. **Windows Forms, WPF** gibi masaüstü uygulamalarından tutunda, katmanları ifade eden **Class Library'** lere kadar çok çeşitli ürünler yer almaktadır. Dolayısıyla performans kazanımı, iş yükünün hafifletilmesi için ön bellekleme işleminin sadece Web uygulamalarına bağımlı kalması düşünülemez. Peki Web ortamı dışında **ön bellekleme(Caching)** teknikleri için hangi imkanlar bulunmaktadır?

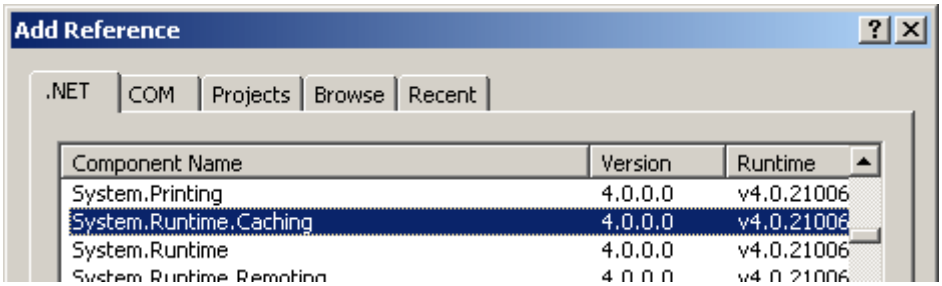
Aslında listenin başında **System.Web.Caching.dll** assembly' ının Web dışındaki uygulamalara referans edilerek kullanılmasının yer aldığını söyleyebiliriz. Ne varki bir Windows uygulamasına **Web** alanına ait bir **Assembly'** ın referans edilmeside son derece gariptir. 😕 Daha etkili bir yöntem olarak **Enterprise Library** içerisinde yer alan **Caching** bloğunun kullanılması tercih edilebilir. Bunların haricinde henüz incelediğim ve yakında sizinle ilk bilgilerimi paylaşacağım **Velocity** projesi de yer almaktadır ki **dağıtık ön bellekleme(Distributed Caching)** ve dolayısıyla **Cache** üzerinde **Load Balancing** vb imkanlar sunmakta olan bir projedir. Ancak **Asp.Net 4.0'** ın duyurulması ile birlikte özellikle Overivew dökümününda belirtilen yeni bir koz daha bulunmaktadır. **System.Runtime.Caching.dll assembly'** ı. Biz bu gün geliştireceğimiz örnekte **ObjectCache** tipinden yararlanarak ön bellekleme işlemlerini yapmaya çalışacağız.



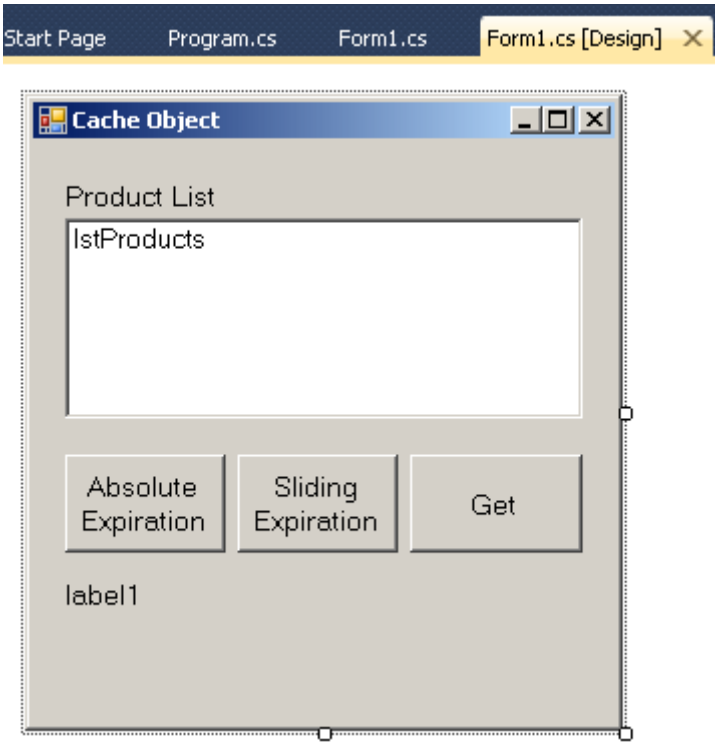
Bu assembly içerisinde yer alan tiplerden yararlanarak özel **Cache** sağlayıcılar, tipler geliştirebileceğimiz gibi **In-Memory** ön bellekleme yeteneklerini web dışındaki uygulamalarda da kullanabiliriz. üstelik **.Net Framework** içerisine dahil edilmiş olması, **Enterprise Library** gibi üçüncü parti kurulumlara ihtiyaç duymayışı, kullanımının Web tarafındaki **Cache** nesnesi ile neredeyse aynı olmasıda avantaj olarak görülebilir. Bu yazımızda söz konusu ön bellekleme tekniklerine sadece iki açıdan bakıyor olacağız. **Sliding** ve **Absolute Expiration** tipli ön bellekleme işlemleri.

Bilindiği üzere ön bellekte tutulacak olan verinin ne kadar süre ile duracağının belirtilmesi **Absolute Expiration** tekniğini ilgilendiren bir meseledir. Diğer yandan belirtilen süre içerisinde ön bellekte tutulan **Cache** nesnesine talep gelmesi sonrası, o andan itibaren tekrardan belirtilen süre kadar yaşam ömrünün uzatılması da **Sliding Expiration** tekniğini ilgilendiren bir konudur. Bunlara ek olarak ön belleklemede popüler olarak kullanılan **bağımlılık(Dependency)** gibi tekniklerde bulunmaktadır. Buna görede **Cache** nesnesinin içeriğinin tazelenmesinin bir sebebe bağımlı hale getirilmesi sağlanabilir. Bu sebep basit bir dosyadaki değişiklik olabilir örneğin.

Dilerseniz hiç vakit kaybetmeden **Visual Studio 2010 Ultimate Beta 2** üzerinde basit bir **Windows** uygulması geliştirerek yolumuza devam edelim. örneğimizde **Product** tipinden nesne örnekleri tutan generic bir **List<T>** koleksiyonunun içeriğinin **Absolute** ve **Sliding** olarak ön belleklenmesi işlemlerini göz önüne alıyor olacağız. Tabi ilk etapta **System.Runtime.Caching** assembly' ının **Win Forms** uygulamasına referans edilmesi gerekmektedir.



WinForms uygulamamızın basit ekran görüntüsü ise aşağıdaki gibidir.



Gelelim kodlarımıza;

```
using System;
using System.Collections.Generic;
using System.Runtime.Caching;
using System.Windows.Forms;

namespace UsingObjectCache
{
   public partial class Form1
      : Form
   {
      // Cache nesnelerine erişmek için kullanılan tip tanımlanır
      ObjectCache cachedObject;
```

```
public Form1()
{
   InitializeComponent();

   // Varsayılan MemoryCache referansı elde edilir
   cachedObject = MemoryCache.Default;
}

// Cache üzerinden Products Key değerine sahip içeriği getirmek için kullanılan olay metodudur
private void btnGet_Click(object sender, EventArgs e)
{
   // Eğer ProductList Key değerine sahip ön belleklenmiş bir içerik var ise
   if (cachedObject["Products"] != null)
   {
      // Indeksleyici yardımıyla ilgili Cache nesnesi çekilir
      lstProducts.DataSource = cachedObject["Products"] as List<Product>;
      lblInformation.Text = "ürün listesi içeriği ön bellekten getirildi";
   }
   else
   {
      lblInformation.Text = "Cache içerisinde söz konusu liste yok\n Bilgiler tekrardan üretilecek";
      lstProducts.DataSource = CreateProductList();
   }
}

// Cache' lenen veriyi belirli bir süreliğine tutmak için gerekli ayarlamaları yapan olay metodudur.
private void btnAbsoluteExpiration_Click(object sender, EventArgs e)
{
   // Zaten Products isimli bir Cache içeriği yoksa, bu Key değerine sahip olan bir Cache içeriği oluşturulur ve o andan itibaren 30 saniye süreyle ön bellekleneceği belirtilir.
   if (cachedObject["Products"] == null)
      if (cachedObject.Add("Products", CreateProductList(), new DateTimeOffset(DateTime.Now.AddSeconds(30))))
         lblInformation.Text = "Absolute Expiration(30 Seconds)";
}

// Kayan süreli Cache' leme için gerekli işlemleri yapan olay metodudur.
// Buna göre 30 saniyelik süre dolmadan gelen talepler, Cache' te tutulmas süresini o andan itibaren tekrar 30 saniye ileri götürürler
private void btnSlidingExpiration_Click(object sender, EventArgs e)
{
   // Absolute Expiration, Sliding Expiration, Dependency gibi Cache çeşitlerini
```

istersek ilke(Policy) olarak tanımlayabiliriz.

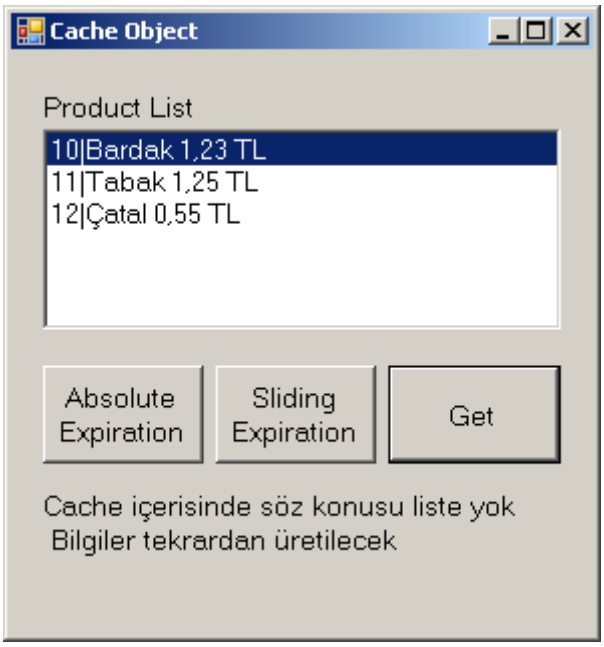
```
        CacheItemPolicy policy = new CacheItemPolicy();
        policy.SlidingExpiration = TimeSpan.FromSeconds(30);

        if (cachedObject["Products"] == null)
           if (cachedObject.Add("Products", CreateProductList(), policy))
              lblInformation.Text = "Sliding Expiration(30 Seconds)";
     }

     // Maliyetli olduğu varsayılan ve içeriği ön bellekte tutulabilecek olan sembolik bir
fonksiyonellik
     private List<Product> CreateProductList()
     {
        return new List<Product>
           {
              new Product{ ProductId=10, Name="Bardak", ListPrice=1.23M},
              new Product{ ProductId=11, Name="Tabak", ListPrice=1.25M},
              new Product{ ProductId=12, Name="çatal", ListPrice=0.55M}
           };
     }
  }

  // Yardımcı tip
  class Product
  {
     public int ProductId { get; set; }
     public string Name { get; set; }
     public decimal ListPrice { get; set; }

     public override string ToString()
     {
        return String.Format("{0}|{1} {2}", ProductId.ToString(), Name,
ListPrice.ToString("C2"));
     }
  }
}
```

Cache yönetimi için **MemoryCache.Default** özelliğinin ürettiği **ObjectCache** nesne referansı kullanılmaktadır. **ObjectCache** tipinin indeksleyicisinden yararlanılarak ön bellekte tutulan bir **Key**değerine ve doğal olarak işaret ettiği **Object** tipinden içeriğe ulaşmak mümkündür. Diğer yandan **Add** ve **Set** gibi metodlar yardımıyla ön belleğe veri atılmasıda sağlanabilir. Burada dikkat çekici noktalardan biriside ön bellekleme tipi için **ilkelerden(Policy)** yararlanılabilmesidir. örneğimizde **Sliding Expiration** için **CacheItemPolicy** nesne örneği tanımlanmıştır. Bu nesne örneği aslında ön bellekleme ilkesini ifade etmektedir. önbelleğe atılacak nesne içeriğinin bu ilkeye göre

tutulacağını belirtmek içinse **Add** metoduna parametre olarak verilmesi yeterli olmuştur. Aslında **Add** metodunun aşırı yüklenmiş versiyonlarından birisi de **Absolute Expiration** tekniğine göre parametre almaktadır ki buda örneğimizde kullanılmıştır.

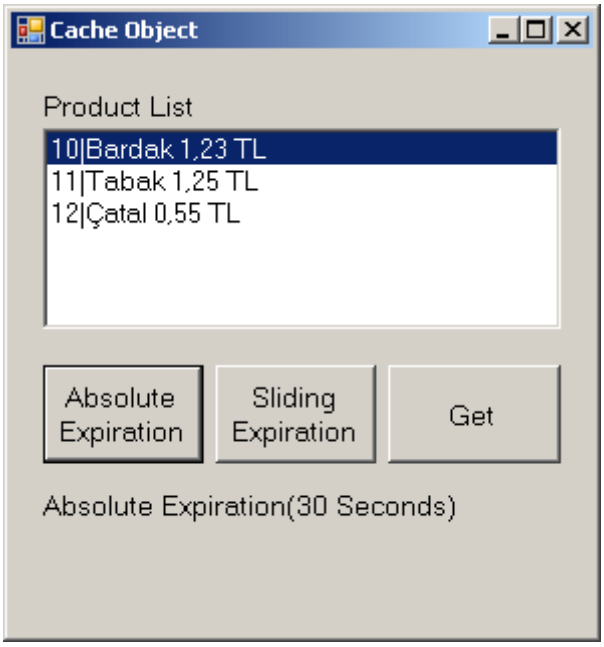
Gelelim testlerimize. Hatırlarsanız kronometrelerinizi hazır tutmanızı söylemiştim. Bu tabiki işin şakası Nitekim kodun içerisinde de test amaçlı olarak bir kronometre kullanılabilir. çalışma zamanındaki testlerimizi aşağıdaki adımlarda olduğu gibi geliştirelim.

1 - Uygulama başlatıldıktan sonra Get başlıklı düğmeye basılır ve aşağıdaki ekran görüntüsü ile karşılaşılır.

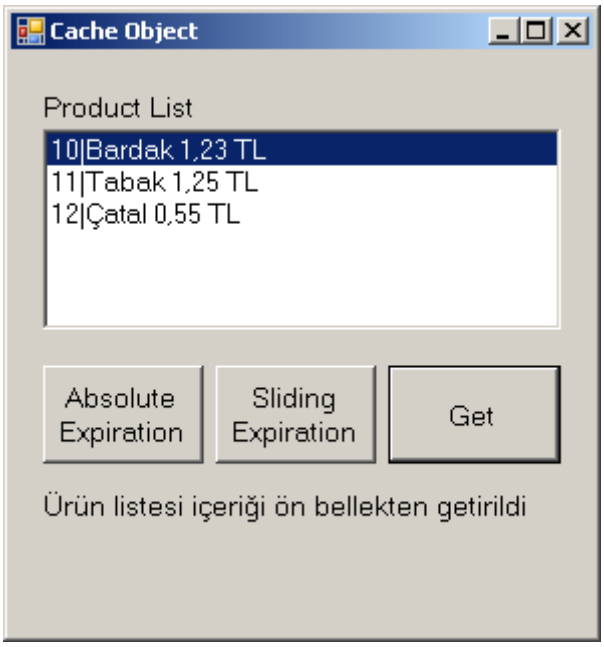


Get içerisinde yapılan çağrıda **ObjectCache** içerisinde **Products** isimli bir **Key** bulunmadığından ürün listesinin maliyeti yüksek olduğu düşünülen bir metod ile üretilmesi gerçekleştirilir.

2 - 1nci testten sonra Absolute Expiration veya Sliding Expiration düğmelerinden birisi kullanılır. Absolute Expiration başlıklı düğme tıklandığında aşağıdaki ekran görüntüsü ile karşılaşılmalıdır.

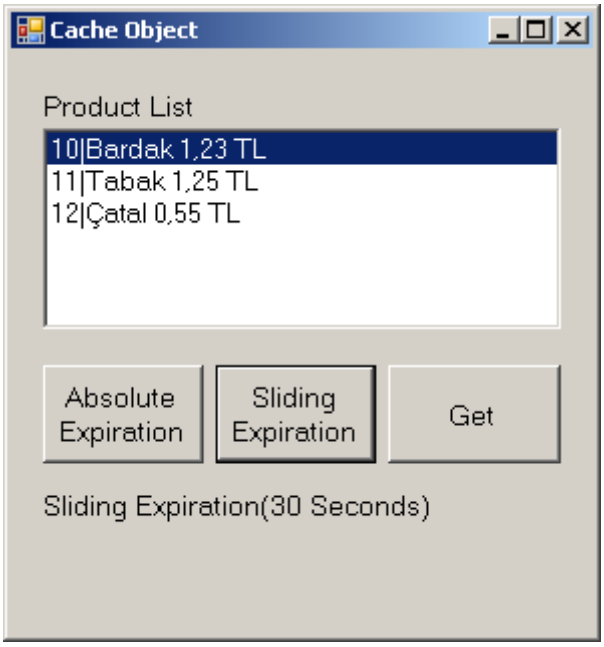


Buna göre Product listesi ön bellekte **30 saniye** süreyle saklanacaktır. Süre dolmadan Get düğmesine basılırsa aşağıdaki ekran görüntüsü ile karşılaşılmalıdır.



Şimdi **30 saniyelik** süre sona erdikten sonra tekrar Get düğmesine basılırsa artık ön bellekte bir veri tutulmadığından ürün listesi üretim işleminin tekrar yapıldığı görülmelidir.

3 - 2nci testin bitmesinden sonra seçiminize göre diğer ön bellekleme tekniğini ele alabilirsiniz. Benim sırama göre şimdi **Sliding Expiration** testinin yapılması gerekmektedir. Program açık iken**Sliding Expiration** başlıklı düğmeye asarsanız aşağıdaki ekran görüntüsü ile karşılaşırsınız.



Bu adımdan sonra **30 saniyelik** süre dolmadan tekrar Get düğmesine basarsanız içeriğin ön bellekten getirildiğini görebilirsiniz. Ancak önemli olan nokta şudur; **ön bellekte durma süresi, 30 saniyelik süre içerisinde Get düğmesine bastığınız andan itibaren 30 saniye sonrasına uzamasıdır**.

4 - Uygulamayı herhangibir **Cache** tekniği uygulandıktan sonra ilgili süreler aşılmadan kapatıp tekrar açınız ve yine Get düğmesine basınız. Bu durumda içeriğin ön bellekten değil tekrardan üretim ile geldiğini görmelisiniz ki bu son derece doğaldır. çünkü uygulama sonlandırılmış ve kendisi için ayrılan bellek içeriği bir sonraki uygulama örneği için geçersiz hale gelmiştir. Hımmm...!!! Aslında bu çokda istediğimiz bir vaka olmayabilir.

Tüm bunlar bir yana en iyi test yöntemlerinden biriside uygulmayı **Debug** ederek incelemeniz olacaktır ki bunu yapmanızı şiddetle tavsiye ederim. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

UsingObjectCache.rar (42,82 kb)

Asp.Net 4.0 - Özelleştirilmiş Cache Sağlayıcısı(Custom Cache Provider) [Beta 2] (2009-11-17T15:10:00)

asp.net 4.0,asp.net,

Merhaba Arkdaşlar,

çok çok uzun zamandır **Asp.Net** üzerine eğilmiyordum. Hem tasarım yönünden kabiliyetsiz olmam*(iki rengi bir araya getir deseler kesin uyumsuz renkler çıkartırım)* hemde servis yönelimli mimari dünyasına dalmış olmamın bunda büyük bir rol oynadığını itiraf etmek isterim. Yine de **Asp.Net 4.0** ile birlikte gelen yenilikleri okuyunca biraz olsun araştırmak ve edindiğim tecrübeleri sizlere aktarmak istedim. Dikkatimi çeken ilk özellik**ön bellekleme(Caching)** sisteminin genişletilebilmesi ile alakalıydı. Bilindiği üzere web uygulamalarında performansı arttırmanın en etikili yollarından biriside sunucu tarafındaki yükü azaltarak mümkün olabilmektedir. Bu manada istemci tarafına, sunucu üzerinde ön belleğe atılmış hazır veri çıktılarını göndermek etkili bir yaklaşımdır. **Asp.Net** tarafında ön bellekleme için farklı teknikler kullanılabilir. **Son süre bildirimli(Expire Date), uzatmalı(Sliding), bağımlı(Dependency, SqlCacheDependency, FileDependency gibi)** vb...üstelik ön belleklenecek veri içeriği sayfa bazında, Web User Control bazında vb olabilir. Yine de eksik olan bir şeyler vardır. Herşeyden önemlisi ön bellekleme gerçekten bellek üzerinde yapılmaktadır. 🙂 Basit bir blog sitesi için ön belleklenecek veri içeriği çok büyük problem teşkil etmeyebilir. Ne varki çok sayıda kullanıcıya hizmet veren portallerde ön belleklenen nesnelerin sayısının, içeriğinin artması, beraberinde belleğinde ölçeklendirilmesi ihtiyacını doğurmaktadır. Buna göre daha çok bellek almak gibi bir maliyet altına girmek gerekebilir. Bu duruma karşın **Asp.Net 4.0** ile birlikte ön bellekleme işlemini daha kolay bir şekilde özelleştirebilme şansına sahibiz. Bir başka deyişle kendi **Cache Provider** tiplerimizi geliştirerek ön bellekleme yerini ve modelini değiştirebilir kendi algoritmalarımızı işin içerisine katabiliriz. Tabi **Beta 2** sürümüne göre bu konuyu incelediğimden ve yazıyı hazırladığım tarih itibariyle internet üzerinde çok fazla kaynak bulamadığımdan halen daha pek çok noktada soru işareti vuku bulmuş durumdadır. Bu nedenle zaman içerisinde bu konuda çok daha detaylı bilgiye ulaşabileceğimizi düşünmekteyim.

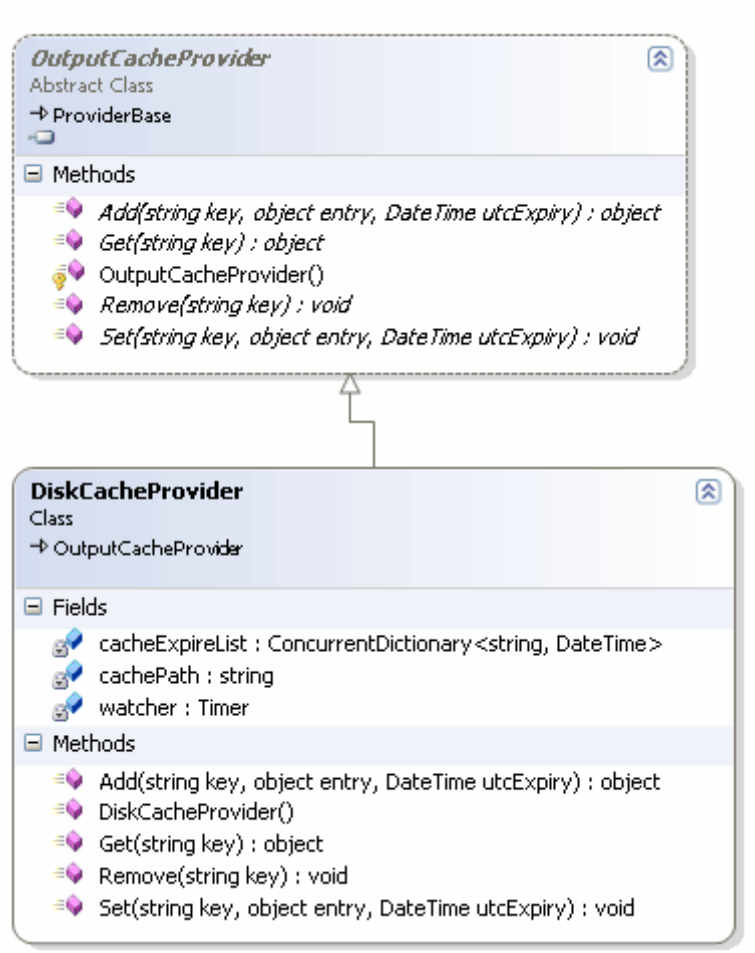
Dilerseniz hiç vakit kaybetmeden örnek bir senaryo üzerinden hareket edelim. Amacımız web uygulamamızda kullanılan bir **Web User Control'** ün içeriğini zamana bağlı olarak ön belleklerken, dosya tabanlı bir sistemden yararlanmak. Buna göre ön belleğe atılacak olan nesne içeriğinin fiziki olarak bir dosya içerisine **serileştirmeyi(Serialization)** planlıyoruz. çok doğal olarak web uygulaması içerisinde n sayıda sayfa ve n sayıda Web User Control olabilir. Buda fiziki olarak her **Web User Control** için birden fazla serileştirilmiş veri içeriği tutan dosya anlamına gelmektedir. Diğer yandan sistemimizde **Expire** eden dosyaların silinmesi işlemlerini de göz önüne almalıyız. Yani bu dosyaların içerikleri gelen

yeni bir talep sonrasında yeniden üretilmeli, talep gelmediği ve Duration dolduğunda ise silinmelidir şeklide bir yol tercih ediyoruz.

Peki **Asp.Net 4.0** tarafında bu tip özelleştirilmiş bir **Cache** sağlayıcısı için ne getirilmiştir?

Aslında tahmin etmek oldukça kolaydır. Var olan bir sisteme yeni bir eklenti ilave etmek istiyorsak tercih edilecek yollardan birisi, sistemin bize söylediği kurallara uymaktır. Şimdilik bu kuralı söyleyen**OutputCacheProvider** isimli **abstract** bir sınıftır. Söz konusu sınıf **Add, Get, Set** ve **Remove** isimli abstract metodlar içermektedir. Buna göre geliştireceğimiz **Cache Provider** sınıfının bu fonksiyonları mutlaka ve mutlaka ezmesi gerekmektedir. Bir başka deyişle **Cache** sistemi için gerekli temel **CRUD** operasyonlarının tarafımızdan uygulanması gerekmektedir. Peki yeterli midir? Elbette değildir. Bir şekilde web uygulaması tarafına, kendi özel **Cache** sağlayıcımızı kullanabileceğimizi ifade etmemiz gerekecektir ki bunun içinde tahmin edeceğiniz üzere **web.config**dosyasından yararlanılacaktır. Geliştireceğimiz örnekte özel **Cache** sağlayıcısını çok kısıtlı olarak kullanabileceğiz. İlk hedefimiz senaryomuzda belirttiğimiz üzere **Web User Control** içeriklerini ön belleklemektir. Buna göre standart olarak kullandığımız **OutputCache** direktifinde bir şekilde özel Cache sağlayıcımızı da işaret edebilmeliyiz ki buda oldukça kolaydır. öyleyse hiç vakit kaybetmeden örneğimizi geliştirmeye başlayalım. İlk olarak **DiskCacheProvider** isimli **OutputCacheProvider** tipinden türeyen aşağıdaki sınıfı geliştirmeliyiz.

DiskCacheProvider sınıfına ait diagram;



Custom Provider kodu;

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;
using System.Timers;
using System.Web.Caching;
using System.Collections.Concurrent;

namespace CustomCaching
{
   public class DiskCacheProvider
      : OutputCacheProvider
   {
      // Expire olan Cache nesnelerini tutan dosyaların silinmesi için bir Timer nesnesi kullanılır
      Timer watcher;
      // Cache nesnelerinin serileştirildiği dosyaların tutulduğu klasör
      string cachePath = "C:\\Cache\\";
```

```
// Dosya adı ve Expire zamanlarını tutan koleksiyon nesnesi. Eş zamanlı çıkartma işlemine destek verebilmek için .Net 4.0 ile gelen Concurrent koleksiyonlardan birisi kullanılmaktadır.
ConcurrentDictionary<string, DateTime> cacheExpireList;

// Yapıcı metod içerisinde gerekli nesne başlatma işlemleri yapılır
public DiskCacheProvider()
{
   cacheExpireList = new ConcurrentDictionary<string, DateTime>();

   // Timer nesne örneği 3 saniyede bir Elapsed olayını tetikleyecektir
   watcher = new Timer(3000);
   // Elapsed olayı içerisinde Expire olan output cache dosyalarının bulunması sağlanır
   watcher.Elapsed += (o, e) =>
   {
      // Koleksiyonda duran Expire zamanı ile güncel zaman karşılaştırılarak bir sonuca gidilmeye çalışılır
      var discardedList = from cacheItem in cacheExpireList
                          where cacheItem.Value < DateTime.Now
                          select cacheItem;
      // Expire olmaya aday olan Cache nesnelerine ait dosyalar için Remove metodu çağırılır.
      // Eğer normal bir Dictionary<T,K> koleksiyonu kullanılırsa çalışma zamanında InvalidOperationException alınabilir. Nitekim discardedList ile gezilirken cacheExpireList' in değişmiş olma ihtimali bulunabilir. Bu nedenle ConcurrentDictionary<T,K> kullanılması tercih edilmiştir
      foreach (var discarded in discardedList)
      {
         Remove(discarded.Key);
         // Koleksiyondan çıkartılır
         DateTime discardedDate;
         cacheExpireList.TryRemove(discarded.Key, out discardedDate);
      }
   };
   // Timer nesne örneği başlatılır
   watcher.Start();
}

// utcExpiry parametresi, Cache için Expire süresini belirtir
public override object Add(string key, object entry, DateTime utcExpiry)
{
   FileStream fs = new FileStream(String.Format("{0}{1}.binary", cachePath, key), FileMode.Create, FileAccess.Write);
   BinaryFormatter formatter = new BinaryFormatter();
   formatter.Serialize(fs, entry);
```

```
            fs.Close();
            cacheExpireList.TryAdd(key, utcExpiry.ToLocalTime());
            return entry;
        }

        // Cache' den nesnesi elde etmek için kullanılan metoddur
        // OutputCacheProvider abstract sınıfından gelen ve ezilmesi mecburi olan bir
fonksiyondur.
        public override object Get(string key)
        {
            string path = String.Format("{0}{1}.binary", cachePath, key);
            // örnekteki sistem Cache nesne içeriklerini dosya tabanlı olarak tutmaktadır. Bu
noktada dosyanın sistemde var olup olmadığına bakılarak Cache' lenen bir içerik olup
olmadığı sonucuna varılabilir
            if (File.Exists(path))
            {
                FileStream fs = new FileStream(path, FileMode.Open, FileAccess.Read);
                BinaryFormatter formatter = new BinaryFormatter();
                // Ters serileştirme işlemi ile dosya içeriğinden Cache' lenen nesnenin
canlandırılması sağlanır
                object result = formatter.Deserialize(fs);
                fs.Close();
                return result;
            }
            else
                return null;
        }

        // Cache nesnesinin kaldırılması için kullanılan metoddur. OutputCacheProvider
abstract sınıfından gelen ve ezilmesi mecburi olan bir fonksiyondur.
        // OutputCacheProvider abstract sınıfından gelen ve ezilmesi mecburi olan bir
fonksiyondur.
        public override void Remove(string key)
        {
            string path = String.Format("{0}{1}.binary", cachePath, key);
            if (File.Exists(path)) // Eğer dosya var ise Cache' lenen bir nesne var olduğu
sonucuna ulaşabiliriz
            {
                // Dosya silinir
                File.Delete(path);
            }
        }

        // Set metodunda Cache in saklanması işlemleri gerçekleştirilir. Genellikle overwrite
mantığına göre çalışır. Yani Cache' lenen nesne varsada üzerine yazılması yoluna gidilir
```

```
        // OutputCacheProvider abstract sınıfından gelen ve ezilmesi mecburi olan bir
fonksiyondur.
        public override void Set(string key, object entry, DateTime utcExpiry)
        {
            // Cache' lenen içerik varsa overwrite işlemi yapılır.
            string path = String.Format("{0}{1}.binary", cachePath, key);

            FileStream fs = new FileStream(path, FileMode.Create, FileAccess.Write);
            BinaryFormatter formatter = new BinaryFormatter();
            // Cache nesnesi serileştirilerek dosyaya yazdırılır
            formatter.Serialize(fs, entry);
            fs.Close();
            // Cache' lenen nesne serileştirildiği dosyada tutulduğundan expire sürelerini takip
edebilmek için ilgili koleksiyonda bilgilendirme yapılır.
            cacheExpireList.TryAdd(key, utcExpiry.ToLocalTime());
        }
    }
}
```

Şimdi çok kısaca neler yaptığımıza bir bakalım. Cache sağlayıcımız Cache' lenecek olan nesneleri varsayılan olarak **C:\Cache** isimli bir klasör altında dosyalamaktadır*(Aslında bu bilgide ilgili provider' a örneğin yapıcı metod yardımıyla web.config dosyası üzerinden geçirilebilir)*. Söz konusu nesne içerikleri ilgili dosyalara **binary** formatta serileştirilmektedir. Bu amaçla **BinaryFormatter** tipinden yararlanılmaktadır. özellikle Cache' e nesne atma işlemi sırasında devreye giren **Set** metodu içerisinde, dosya adı için **key** parametresininin kullanıldığına dikkat edilmelidir. Hatta asıl dikkat edilmesi gereken nokta her cache için değişik bir key değerinin üretildiğidir*(Ancak ispatını yapamadım henüz bunu belirteyim)*. Diğer yandan **Expire**süreleride ilgili metodlara(örneğin Set) parametre olarak gelmektedir. Bu süre bilgilerinden yararlanılarak ön bellekten düşürme işlemleri yapılmalıdır. örneği geliştirirken beklentilerimi boşa çıkaran noktalardan biriside aslında Expire süreleri ile ilişkilidir. Şöyleki; örneği geliştirirken **Remove** metodunun belirtilen **Duration** süresi sonlandığında otomatik olarak devreye gireceğini tahmin etmiştim. Ancak bu şekilde olmadı 😕 Bu nedenle Cache' lenen dosyaların zamanı geldiğinde silinmesi için bir mekanizmanın geliştirilmesi gerekiyordu. Bu amaçla **Timer**nesnesinden yararlanarak **Elapsed** olayı içerisinde gerekli silme işlemlerini gerçekleştirmeyi tercih ettim. çok doğal olarak hangi dosyaların silinmesi gerektiğini anlayabilmek içinde basit bir koleksiyon tabanlı yapıyı tercih ettim. Tabiki Thread Safe bir yapı söz konusu değil. Hatta eş zamanlı gerçekleşebiliecek çıkarma işlemlerine karşın kolaya kaçıp**ConcurrentCollection<T,K>** kullandığımı ifade edebilirim. Aslına bakarsanız şu an için tek derdim gerçekten özelleştirilmiş bu Cache sağlayıcısının çalışıp çalışmadığını görebilmek. 😎

Şimdi bu **Cache Provider** tipini web uygulamamızda nasıl kullanacağımızı belirtmemiz gerekiyor. Bu amaçla **Web.config** dosyası içerisinde aşağıdaki eklemeleri yapmamız yeterli olacaktır.

web.config içeriği;

```
<?xml version="1.0"?>
<configuration>

   <system.web>
      <compilation debug="true" targetFramework="4.0" />
     <caching>
      <outputCache defaultProvider="AspNetInternalProvider">
       <providers>
         <add name="DiskBasedCacheProvider"
type="CustomCaching.DiskCacheProvider,CustomCaching"/>
       </providers>
      </outputCache>
     </caching>
   </system.web>
   <system.webServer>
     <modules runAllManagedModulesForAllRequests="true"/>
   </system.webServer>

</configuration>
```

Dikkat edileceği üzere **providers** elementi içerisinde **DiskBasedCacheProvider** isimli bir bildirim yer almakta ve geliştirdiğimiz **DiskCacheProvider** tipini işaret etmektedir. Bununla birlikte **outputCache**elementinin **defaultProvider** niteliğinde yer alan **AspNetInternalProvider** değeri, standart **Asp.Net** ön bellekleme sisteminin kullanılacağını ifade etmektedir. örneğimize aşağıdaki basit Web User Control' ü ekleyerek devam edebiliriz.

Web User Control içeriği;

```
<%@ Control Language="C#" AutoEventWireup="true"
CodeBehind="CurrentDateTime.ascx.cs" Inherits="CustomCaching.CurrentDateTime" %>

<%@ OutputCache Duration="20" VaryByParam="none"
ProviderName="DiskBasedCacheProvider" %>
<style type="text/css">
   .style1
   {
      color: #FFCC00;
   }
</style>
<div style="background-color:Gray">
   <strong><span class="style1">Güncel Zaman :
</span>
```

```
<asp:Label ID="Label1" runat="server" Text="Label" CssClass="style1"></asp:Label>
   </strong>
</div>
```

OutputCache direktifi içerisindeki **ProviderName** niteliği mutlaka dikkatinizi çekmiştir. Burada **DiskBasedCacheProvider** isimli bir değer kullanılmaktadır. Buda bilindiği üzere **web.config** dosyasında yer alan özel **Cache** sağlayıcı tipini işaret etmektedir. Duration değeri **20** olduğu için söz konusu **Web User Control** içeriğinin **20 saniye** süreyle tutulması söz konusudur.

Web User Control kodu;

```
using System;

namespace CustomCaching
{
   public partial class CurrentDateTime : System.Web.UI.UserControl
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Label1.Text = DateTime.Now.ToLongTimeString();
      }
   }
}
```

Ve hemen arkasından basit bir Web sayfası...

Default.aspx içeriği;

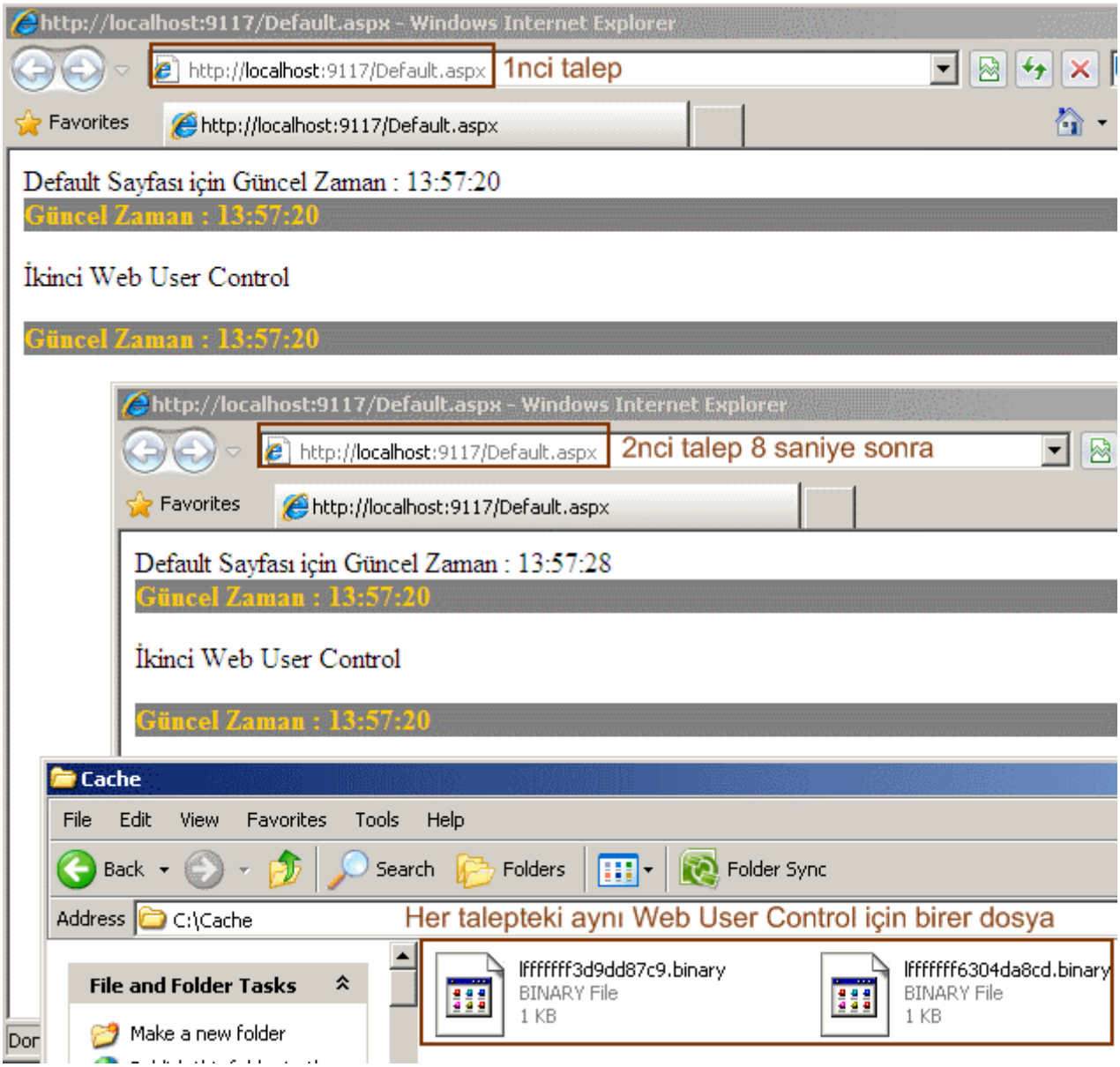
```
<%@ Page Language="C#" AutoEventWireup="true" CodeBehind="Default.aspx.cs"
Inherits="CustomCaching.Default" %>

<%@ Register src="CurrentDateTime.ascx" tagname="CurrentDateTime" tagprefix="uc1"
%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
   <title></title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
```

```
      Default Sayfası için Güncel Zaman :
      <asp:Label ID="Label1" runat="server" Text="Label"></asp:Label>

   </div>
   <uc1:CurrentDateTime ID="CurrentDateTime1" runat="server" />
   <p>
      İkinci Web User Control</p>
   <uc1:CurrentDateTime ID="CurrentDateTime2" runat="server" />
   </form>
</body>
</html>
```

Default.aspx kodu

```
using System;

namespace CustomCaching
{
   public partial class Default : System.Web.UI.Page
   {
      protected void Page_Load(object sender, EventArgs e)
      {
         Label1.Text = DateTime.Now.ToLongTimeString();
      }
   }
}
```

Tahmin edileceği üzere o anki zaman bilgisini hem aspx sayfası hemde **Web User Control** içerisinden göstererek bir karşılaştırma yapmaya çalışacağız. öyleki; eğer Cache sağlayıcımız devreye girerse, Web User Control içeriği 20 saniyeliğine fiziki olarak tutulan bir dosyadan karşılanacak ve bu sebeple sayfanın zaman bilgisi değişirken kendisinin ki sabit olarak kalacak. Tabiki belirtilen **Duration** süresi kadar. Artık testlere başlayabiliriz. Size önerim indirdiğiniz örneği mutlaka **Debug** ederek incelemenizdir. özellikle Set ve Get metodları ile Elpased olayında durmanızı tavsiye ederim. Ben ilk olarak aynı web sayfasına iki farklı talep gönderip aşağıdaki çıktıları elde ettim.



Görüldüğü üzere iki farklı talep gönderilmiş ve özellikle ikince talepte Web User Control içerisindeki zamanın değişmediği görülmüştür. çünkü bu içerik ilk talep ile birlikte **20** saniyeliğini fiziki olarak bir dosyaya serileştirilmiş ve bu zaman dilim içerisinde sürekli olarak ilgili dosyadan **ters serileşerek(DeSerialization)** getirilmiş bir **HTML** çıktısıdır. 20 saniyelik süre sona erdikten sonra sayfalarda herhangibir yeni talep oluşturmassak, ilgili fiziki dosyalarında Cache klasöründen silindiği gözlemlenir. Geliştirdiğimiz örnekte sadece **Set, Get** ve **Remove**metodları işlevsel durumdadır. Yani söz konusu vakaya göre **Add** metodu herhangibir sebeple çalışmamıştır.

Hemen bir noktayı aydınlığa kavuşturalım. Geliştirdiğimiz örnekte Web User Control' ün üretimi özel Cache sağlayıcısı içerisindeki serileştirme ve ters serileştirme gibi işlemlerin çıkarttığı maliyetten daha az olabilir. Yani aslında bu örneğe göre bir Cache sistemi kullanılmasına gerek duyulmaz. Bizim amacımız sadece özel bir Cache sağlayıcısının nasıl yazılabileceğini **Asp.Net 4.0 Beta 2** cephesinden incelemektir. Tabiki genişletilebilir Cache sisteminin daha esnek imkanlar sunacağıda belirtilmektedir. Aslında bu konu ile

ilişkili özet bir bilgiyi **asp.net** sitesinden indireceğiniz dökümanda bulabilirsiniz. Böylece geldik bir yazımızın daha sonuna. Tabiki buradaki eksikleri ve gereksinimleri en iyi değerlendirecek kişi sevgili arkadaşımUğur Umutluoğlu' dur(Asp.Net MVP). Tekrardan görüşünceye dek hepinze mutlu günler dilerim.

CustomCaching.rar (26,21 kb)

INETA Next Bomb Aralıkta Patlıyor (2009-11-17T08:26:00)

seminer,

Ado.Net Entity Framework 4.0 - Stored Procedures ve Complex Types (2009-11-16T10:55:00)

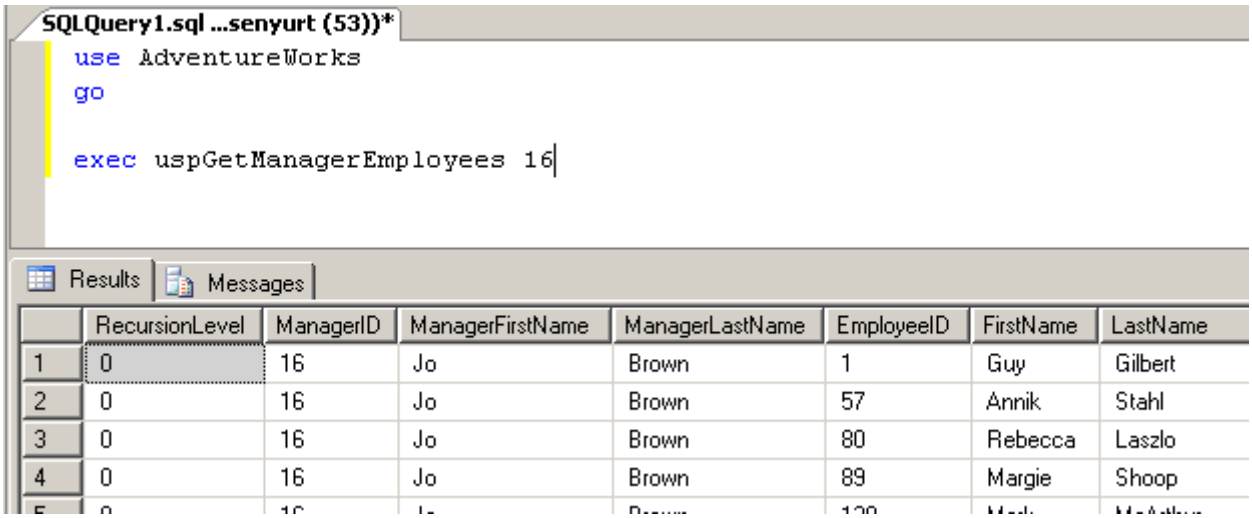
ado.net entity framework,

Merhaba Arkadaşlar,

Ado.Net Entity Framework 4.0 ile birlikte gelecek/gelmekte olan yeniliklerden birisi de, **Stored Procedure'** lerin dönüş tipi ile alakalıdır. Henüz tam olarak bitirilememiş olan bu özellik şu anki haliyle bir **Stored Procedure'** den geriye **karmaşık bir tipinin(Complex Type)** döndürülebilmesine izin vermektedir. Bunun için Designer tarafında destek sunulmaktadır. Aslında önceki **Ado.Net Entity Framework** sürümünde bir **Stored Procedure'** ün **Entity** modeli içerisine eklenmesi sonrasında dönüş

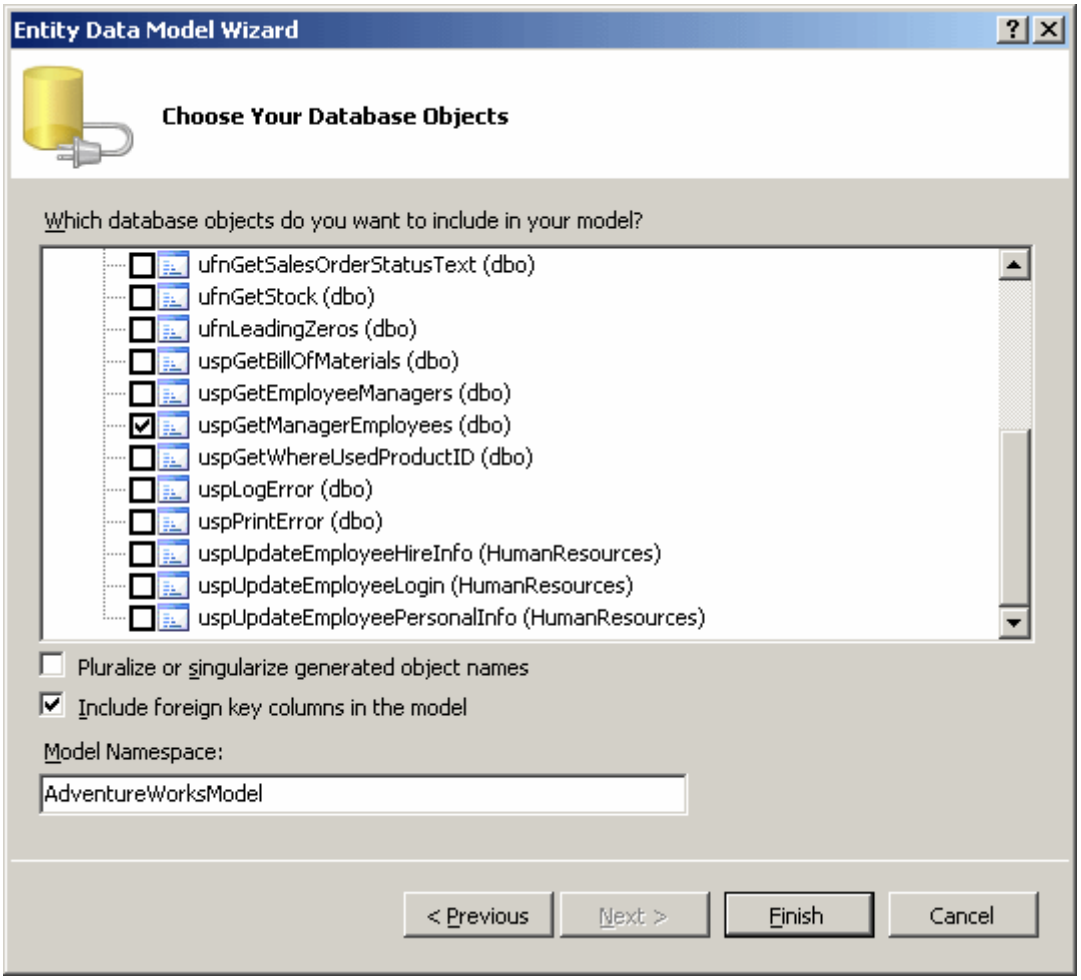
kümesinin **Scalars** veya **Entities** olarak kullanılması sağlanabilmekteydi. Ancak bir**Stored Procedure** çıktısının Complex Type bazlı olaraktan kod tarafında ele alınamayışı da de önemli bir eksiklikti. Bakalım **4.0** versiyonunda bu eksikliği gidermek adına neler yapılmış. Yazımızın ilerleyen kısımlarında bu özeliği anlamaya çalışıyor olacağız.

İlk olarak kendimize kobay bir **Stored Procedure** seçelim Bu amaçla **Adventure Works** veritabanında yer alan **uspGetManagerEmployees Stored Procedure'** ünü kullanabiliriz. Bu procedure parametre olarak **ManagerID** isimli int tipinden bir dğer almakta ve aşağıdaki örnek çıktıyı üretmektedir.



	RecursionLevel	ManagerID	ManagerFirstName	ManagerLastName	EmployeeID	FirstName	LastName
1	0	16	Jo	Brown	1	Guy	Gilbert
2	0	16	Jo	Brown	57	Annik	Stahl
3	0	16	Jo	Brown	80	Rebecca	Laszlo
4	0	16	Jo	Brown	89	Margie	Shoop

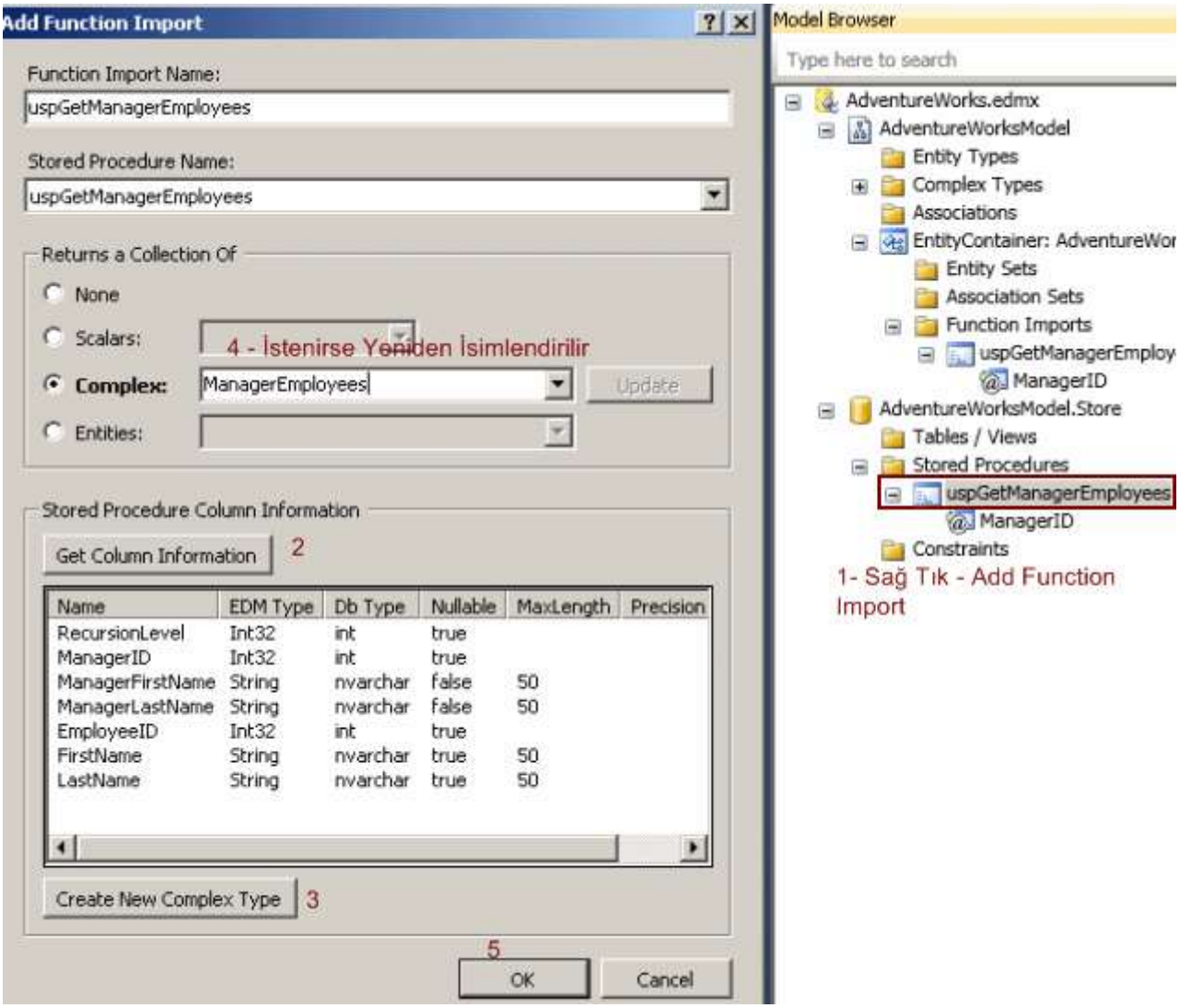
Pekala **Stored Procedure'** ümüzün çıktısı, **RecursionLevel,ManagerID,ManagerFirstName, ManagerLastName, EmployeeID, FirstName** ve **LastName** alanlarının karşılıklarını **özellik(Property)** olarak içeren sınıfa ait nesne örneklerinden oluşan bir nesne kümesi ile ifade edilebilir. Bunu gerçekleştirmek için **Visual Studio 2010 Ultimate Beta 2** sürümünde oluşturacağımız **Console** uygulamamıza **Adventure Works** veritabanı için yeni bir **Ado.Net Entity Data Model** öğesi ekleyip sadece **uspGetManagerEmployees** SP' sini seçtiğimizi düşünelim.



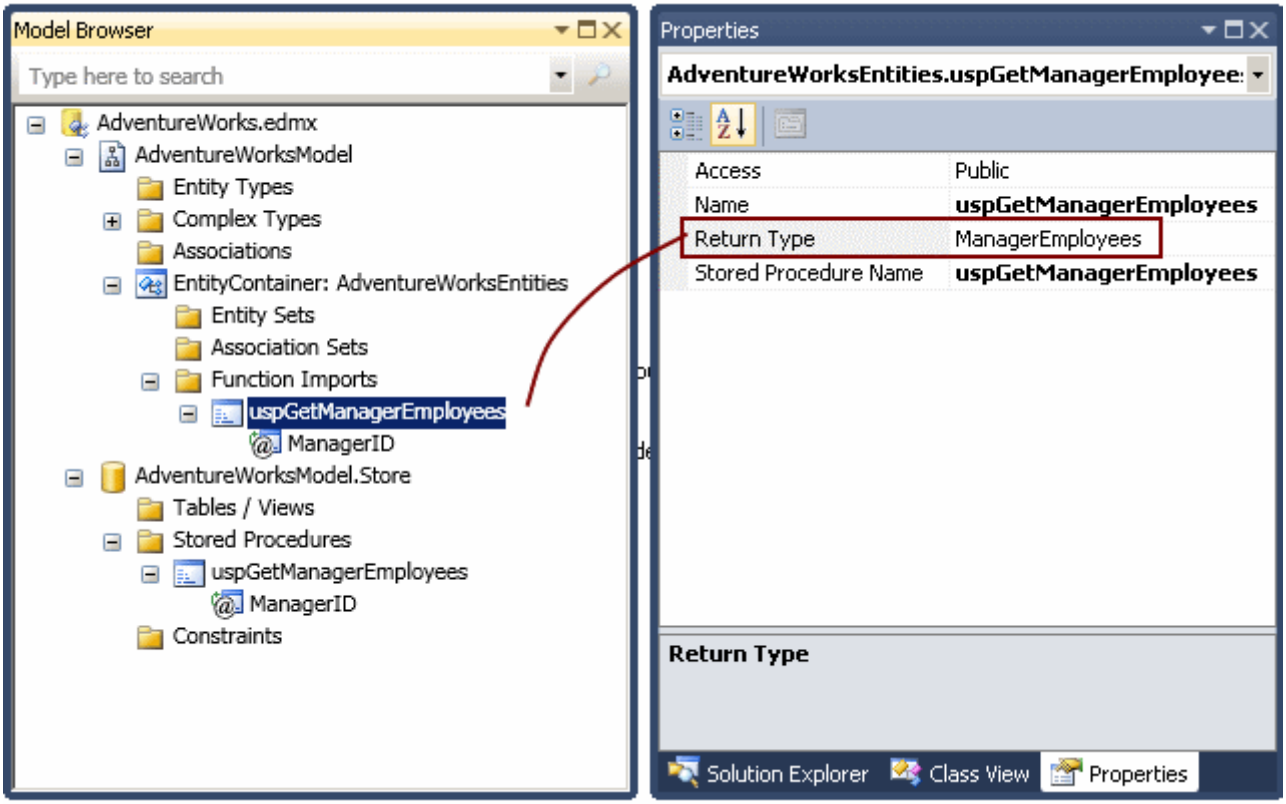
Şimdi adım adım ilerleyerek **Complex Type** üretimini gerçekleştireceğiz.

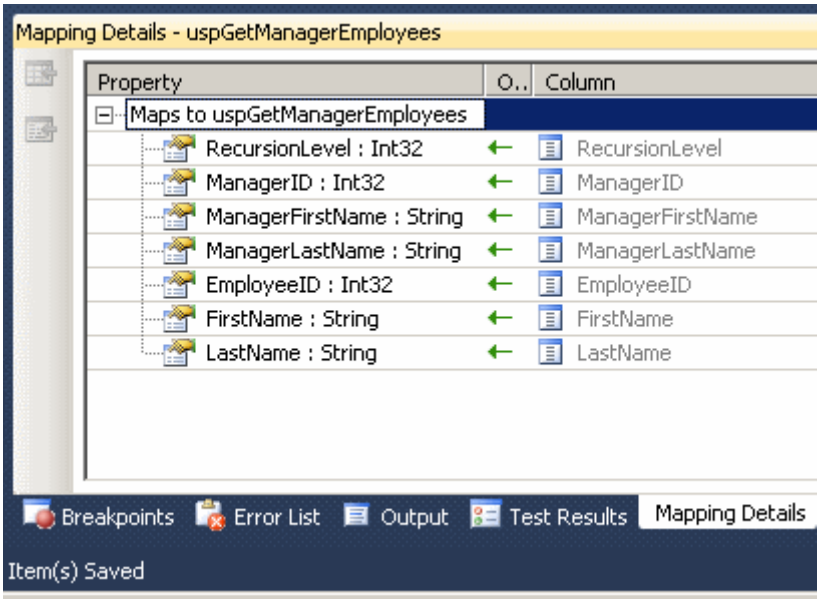
1. İlk olarak Model **Browser** ve **AdventureWorksModel.Store** kısmından ilgili **Stored Procedure** adına sağ tıklanıp **Add Function Import** seçimi yapılmalıdır.
2. Seçimin ardından gelen ekranda **Get Column Information** düğmesine basıldığı takdirde ilgili Stored Procedure' den dönen sonuç kümesi için uygun olan kolon bilgilerinin getirildiği görülür. Burada hem **EDM Type** hemde **SQL Type** bilgileri yer almaktadır.
3. Sonraki adımda ise **Create New Complex Type** düğmesine tıklanarak **Stored Procedure'** ün sonuç kümesi için gerekli sınıfın üretilmesi sağlanır.
4. İstenirse üretilen tip adı Complex kutucuğundan değiştirilir*(Biz örneğimizde tip adını ManagerEmployees olarak yeniledik)*
5. Ok tuşuna basılarak işlem tamamlanır.

Aşağıdaki şekilde bahsetmiş olduğumuz adımlar görsel olarak özetlenmektedir.

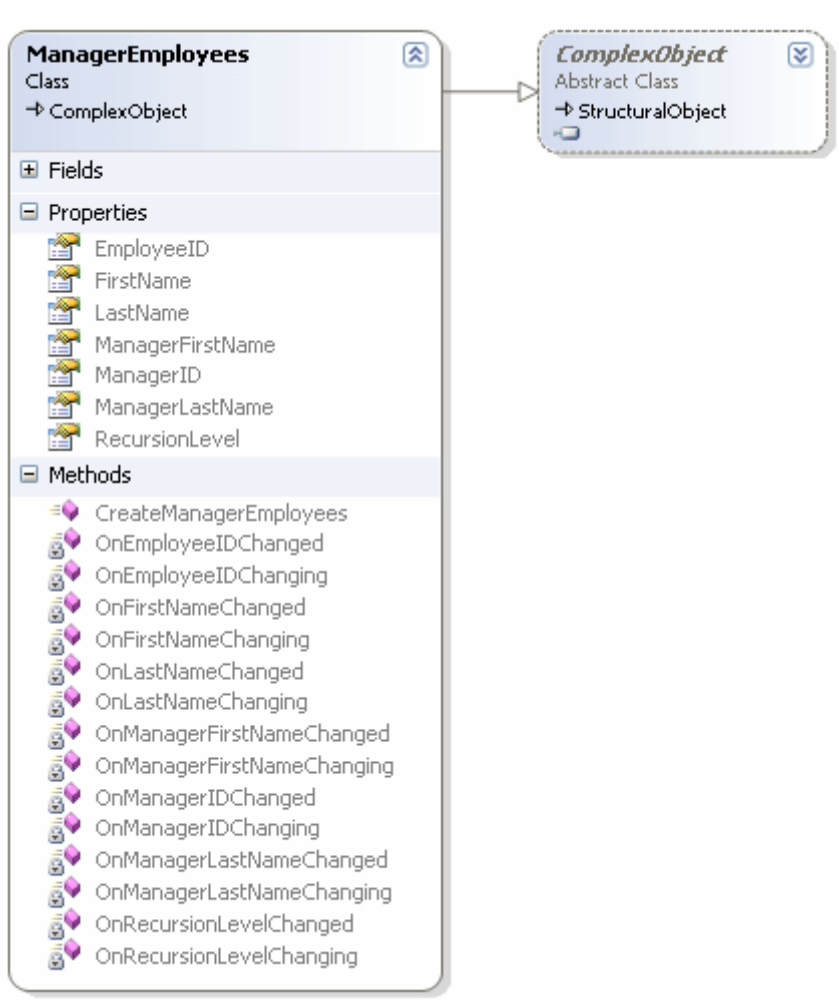


Şimdi arka planda neler olduğuna bir bakalım. İlk olarak **Entity Data Model** tarafında söz konusu **Stored Procedure** için bir **Return Type** ve **Mapping Details** kısmında gerekli kolon-özellik eşleştirmelerinin yapıldığı görülür.

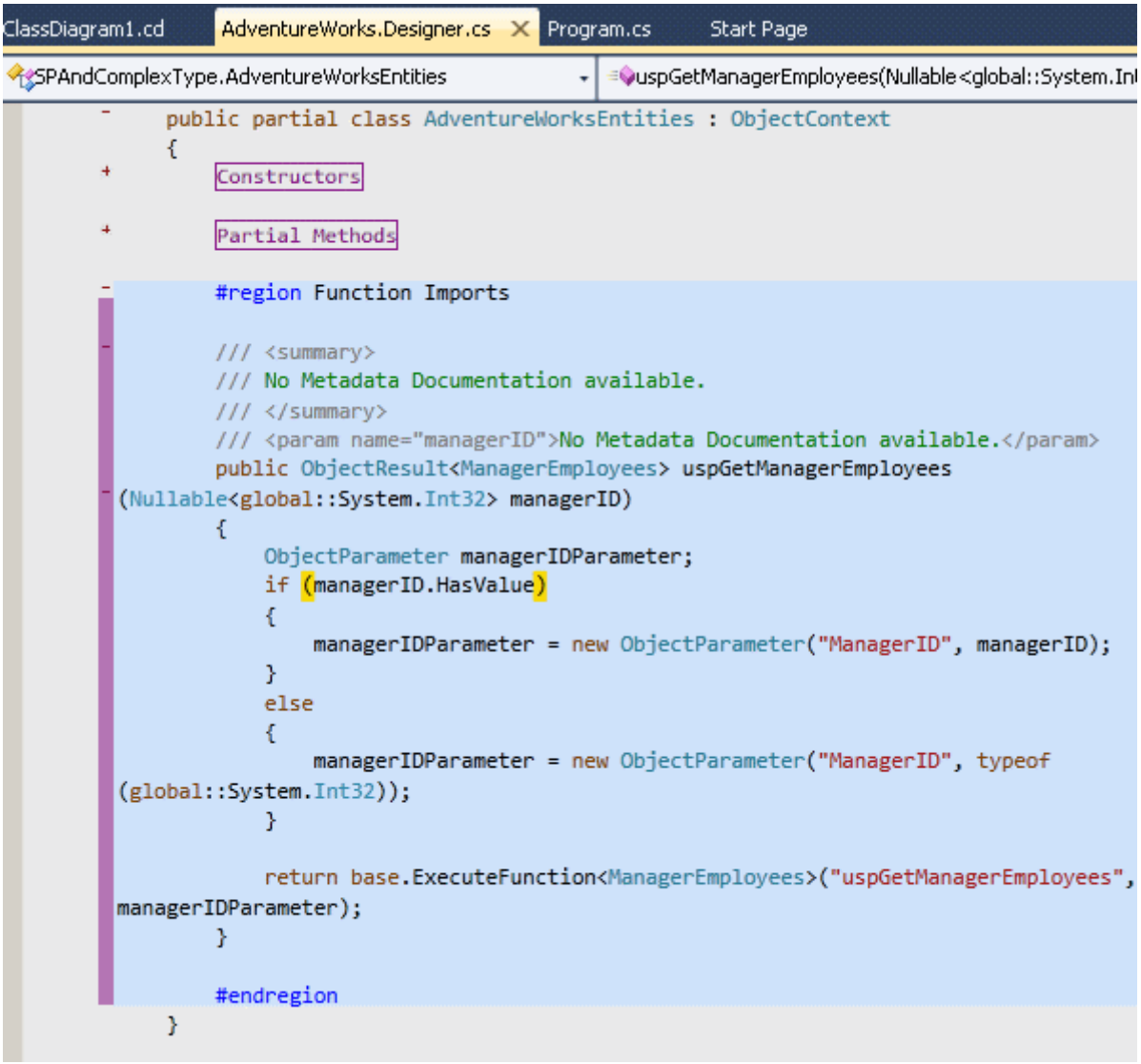




Buna ek olarak **Return Type** için kod tarafında **ManagerEmployees** isimli **ComplexObject** türevli bir sınıfın üretildiği gözlemlenir. **Class Diagram** görüntüsünde bu durum açık bir şekilde izlenebilmektedir.



çok doğal olarak söz konusu **Stored Procedure'** ün kod içerisinde kullanılabilmesi için gerekli özelliğin de **AdventureWorksEntities Context** tipi içerisine eklendiği görülebilir.



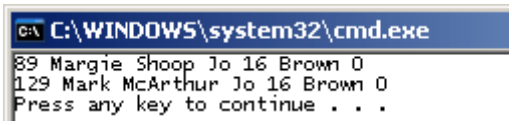
Dikkat edileceği üzere **uspGetManagerEmployees** isimli **özelliğin(Property)** dönüşü **ObjectResult<ManagerEmployees>** tipindendir. Bu tip **IEnumerable<T> ve IEnumerablearayüzlerini(Interface)** implemente ettiğinden **LINQ** sorgularında kaynak veri olarak kullanılabilir.

Dilerseniz birde söz konusu **Stored Procedure'** e ait çıktıyı örnek kod parçasında değerlendirmeye çalışalım. Bunun için aşağıdaki gibi basit bir kodlama yapmamız yeterli olacaktır.

```
using System;
using System.Data.Objects;
using System.Linq;

namespace SPAndComplexType
{
   class Program
```

```
   {
      static void Main(string[] args)
      {
         using (AdventureWorksEntities context = new AdventureWorksEntities())
         {
            ObjectResult<ManagerEmployees>
resultSet=context.uspGetManagerEmployees(16);

            var subResult = from me in resultSet
                        where me.FirstName[0] == 'M'
                        select me;

            foreach (var managerEmployee in subResult)
            {
               Console.WriteLine("{0} {1} {2} {3} {4} {5} {6}",
                  managerEmployee.EmployeeID.ToString(),
                  managerEmployee.FirstName,
                  managerEmployee.LastName,
                  managerEmployee.ManagerFirstName,
                  managerEmployee.ManagerID,
                  managerEmployee.ManagerLastName,
                  managerEmployee.RecursionLevel
                  );
            }
         }
      }
   }
}
```

örnek kod parçasında **uspGetManagerEmployess** özelliği çağırılmış ve **ManagerID** değeri **16** olan **Employee** listesinin getirilmesi sağlanmıştır. Bu işlemin arkasından da elde edilen sonuç kümesi üzerinden basit bir **LINQ** sorgusu çalıştırılmış ve **FirstName** alanının ilk harfi **M** olanların listelenmesi sağlanmıştır. Hemen bir hatırlatma yapalım; **SQL**tarafındaki uspGetManagerEmployees Stored Procedure' ünün çalıştırılması **context** nesne örneği üzerinden **uspGetManagerEmployess** özelliğinin çağırılması ile birlikte gerçekleşmektedir. Son olarak uygulamayı çalıştırdığımızda aşağıdaki çıktıyı elde ettiğimizi görürüz.



Böylece geldik bir yazımızın daha sonuna. Ado.Net Entity Framework 4.0 ile ilişkili yenilikleri öğrendikçe sizlerede aktarmaya çalışıyor olacağım. Bu konuda ilk elden bilgi

almak isteyen arkadaşlarımızın **Ado.Net Team Blog'** unu mutlaka takip etmesini öneririm. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

SPAndComplexType.rar (36,40 kb)

Ado.Net Data Services 1.5 CTP2 - Data Binding Bölüm 2 (2009-11-16T08:09:00)

ado.net data services,

Merhaba Arkadaşlar,

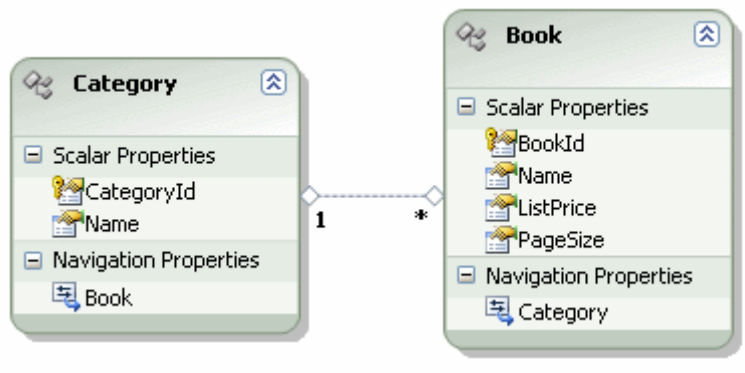
Orta uzunlukta bir yazı için hazır mısınız? Analizi dikkat gerektiren bir **Ado.Net Data Service** örneği geliştiriyor olacağız. Genellikle bu tarz yazılara ait kodları geliştirirken sıkılmamak ve zihnimi açık tutmak için ya kahve içerim yada mutluluktan havalara uçarmış gibi yazabilmek ve kan şekerimi üst seviyede tutabilmek için bazen değişik şekerlemelerden yerim. Aynen yandaki resimde olduğu gibi.😎

Hatırlayacağınız gibi bir önceki yazımızda, **Ado.Net Data Service** için istemci taraflı veri bağlama işlemlerinde **DataServiceCollection<T>**kolekisyonunu değerlendirmeye çalışmış ve istemci tarafında bu konuyu ele almak için basit bir **WPF** uygulaması geliştirmiştik. Bir önceki örneğimiz aslında tek yönlü veri bağlama işlemine örnek olmasında rağmen, iki yönlü modeli de desteklemektedir. Bu yazımızda geliştireceğimiz örneğimizdeki hedefimiz ise, istemci tarafındaki koleksiyonlar üzerinden yapmış olduğumuz güncelleştirme, ekleme ve silme işlemlerini servis tarafına yansıtmaktır.

Aslında süreç son derece basittir. Veri bağlı kontroller üzerinde yapılan güncelleştirme hareketleri, **DataServiceCollection** koleksiyonu üzerinde de gerçeklenecektir. Sonrasında ise**DataServiceContext** türevli nesne örneği üzerinden **SaveChanges** metodunun çağırılması yeterli olacaktır. Bu sayede koleksiyon üzerinden yapılan tüm güncelleştirme, ekleme ve silme işlemlerinin, servis tarafına bir talep olarak gönderilmesi ve sunucu üzerinde de kullanılan veri kaynağına göre*(Entity Framework veya Custom LINQ Provider)* uygun bir veri işleminin yapılması sağlanır. Biz örneğimizde kendi geliştireceğimiz bir veritabanı içeriğini ve **Ado.Net Entity Framework** modelini

kullanacağımızdan, sunucu üzerinde **SQL** sorgularının çalıştırıldığını göreceğiz. Dilerseniz vakit kaybetmeden örneğimizi geliştirmeye başlayalım ve işleyişini analiz edelim.

örneğimizde kendi geliştirdiğimiz **BookShop** isimli bir veritabanını kullanıyor olacağız. Hayali olarak bir kitapçının veri ambarı olarak tasarladığımızı farz edebiliriz. Söz konusu veritabanını oluşturmak için BookShopDbScripts.sql (13,83 kb) dosyasından yararlanabilirsiniz. Bu dosya içerisinde veritabanı, tabloların oluşturulması ve örnek veri girişleri için gerekli **SQL Script'** leri bulunmaktadır. Bu noktadan yola çıkarak geliştireceğimiz **Ado.Net Data Service** örneğinde kullanacağımız **Entity DataModel** diagramını aşağıdaki gibi tasarlayabiliriz. örneğimizdeki amacımız kitap güncellemek, kitap eklemek ve silmek gibi işlemler olacaktır.

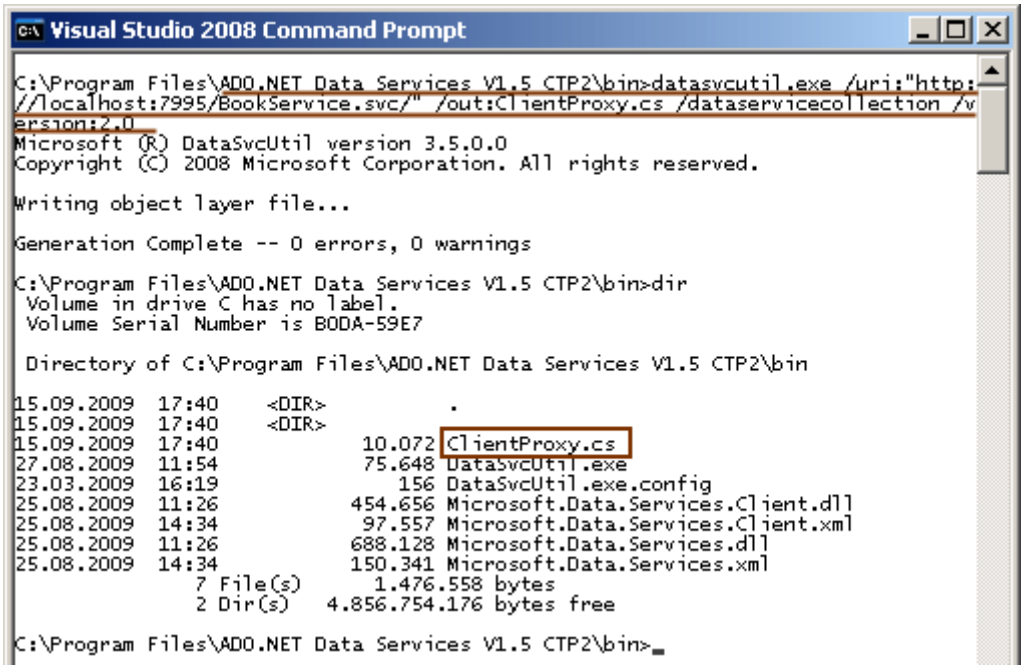


Gelelim **Ado.Net Data Service** örneğimize. **Version 1.5 CTP2** versiyonuna göre eklediğimiz servisin kod içeriğini aşağıdaki gibi tasarlayabiliriz.

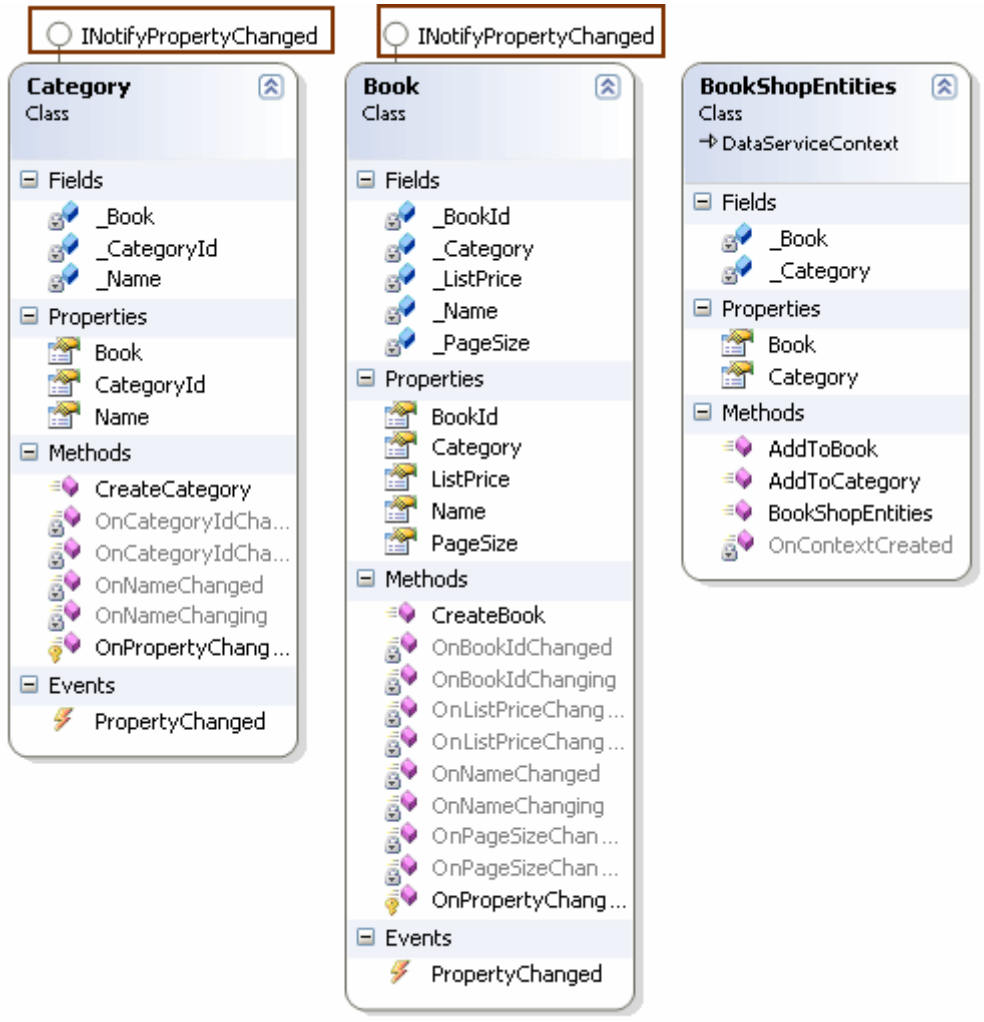
```
using System.Data.Services;

namespace BookShop
{
   public class BookService
      : DataService<BookShopEntities>
   {
      public static void InitializeService(DataServiceConfiguration config)
      {
         config.SetEntitySetAccessRule("*", EntitySetRights.All);
         config.DataServiceBehavior.MaxProtocolVersion =
System.Data.Services.Common.DataServiceProtocolVersion.V2;
    }
   }
}
```

İstemci tarafını yine bir **WPF** uygulaması olarak tasarlayıp, iki yönlü veri bağlama kabiliyetlerini etkin bir şekilde kullanmayı planlıyoruz. Yazıyı hazırladığım sıralarda, **Visual Studio 2008** üzerinde garip ve gizemli bir sorunla karşılaştım. öyleki, istemci uygulamada **Add Service Reference** ile **proxy** üretimi yapılmasına rağmen,

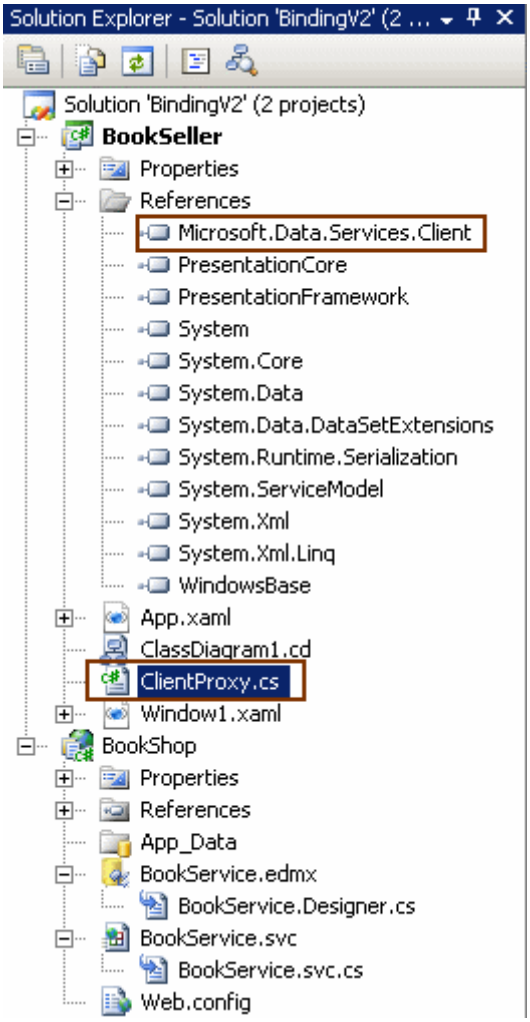
indirilen **Entity** tiplerinin**INotifyPropertyChanged** arayüzünü uygulamadıklarını gördüm. Ama tabiki çaresiz değildim. **DataSvcUtil** aracının **version 1.5 CTP 2** sürümünü aşağıda görüldüğü gibi kullanarak, istemci için gerekli olan
ve **INotifyPropertyChanged** arayüzünü implemente etmiş tipleri üretebiliriz.



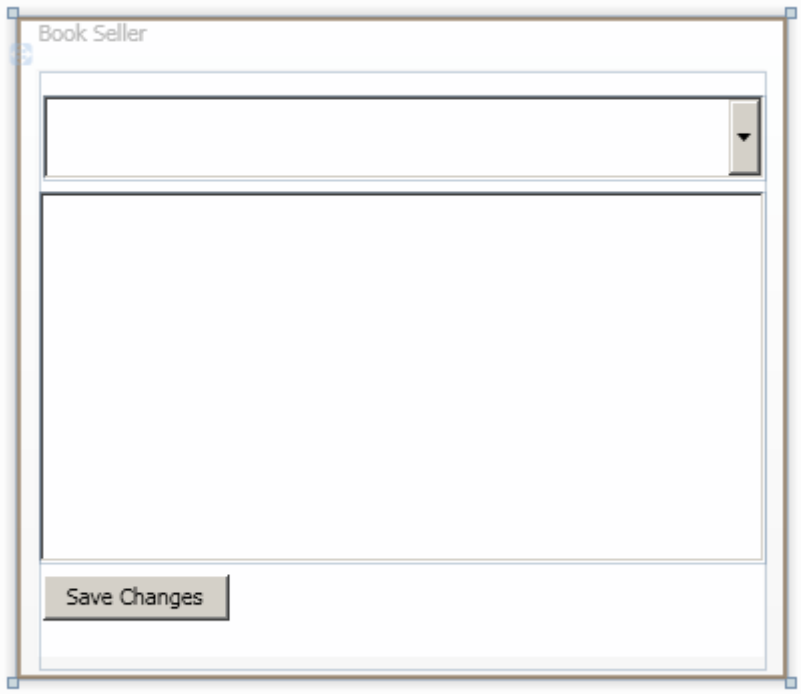
Bu noktadan sonra üretilen sınıfı istemci tarafında kullanmamız yeterli olacaktır. İşte istemci tarafı için üretilen **Entity** tipleri.



Yanlız komut satırından yapılan **proxy** sınıfını istemcide kullanabilmek için, **Ado.Net Data Services Version 1.5 CTP2** ile gelen **Microsoft.Data.Services.Client assembly**' ının projeye referans edilmesi gerekmektedir. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.



Bu ön hazırlıklardan sonra istemci tarafındaki **Window1** içeriğini başlangıçta aşağıdaki gibi tasarladığımızı düşünelim.



Burada üst tarafta yer alan **ComboBox** içerisinde kitap kategorileri, alt tarafta yer alan **ListBox** kontrolünde ise seçilen kategoriye bağlı kitaplar görüntülenecektir. Yanlız **XAML** tarafına baktığınızda **WPF** açısından etkili bazı tekniklerin kullanıldığını görebilirsiniz.

XAML İçeriği;

```
<Window x:Class="BookSeller.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Book Seller" Height="330" Width="384">
   <Grid Name="grdBook" Height="298">
      <ComboBox Height="42"
Margin="2,12,0,0" Name="cmbCategories" VerticalAlignment="Top"  IsSynchronized
WithCurrentItem="true" ItemsSource="{Binding}">
         <ComboBox.ItemTemplate>
            <DataTemplate>
               <StackPanel Orientation="Horizontal">
                  <TextBlock Name="txtCategoryId" Text="{Binding
Path=CategoryId}" FontSize="22" Foreground="RosyBrown"/>
                  <TextBlock Name="txtCategoryName" Text="{Binding
Path=Name}"/>
               </StackPanel>
            </DataTemplate>
         </ComboBox.ItemTemplate>
      </ComboBox>
      <ListBox IsSynchronizedWithCurrentItem="True" Name="lstProducts"
```

```
Margin="0,60,0,53" ItemsSource="{Binding Book}">
        <ListBox.ItemTemplate>
          <DataTemplate>
            <StackPanel Orientation="Vertical">
              <TextBox Name="txtProductName" Text="{Binding Name}"
Width="250"/>
              <TextBox Name="txtListPrice" Text="{Binding ListPrice}"
Width="100" Foreground="Gold" Background="Black"
HorizontalAlignment="Right"/>
            </StackPanel>
          </DataTemplate>
        </ListBox.ItemTemplate>
      </ListBox>
      <Button Height="23" HorizontalAlignment="Left" Margin="2,0,0,24"
Name="btnSaveChanges" VerticalAlignment="Bottom" Width="93"
Click="btnSaveChanges_Click">Save Changes</Button>
   </Grid>
</Window>
```

Her şeyden önce, **ComboBox** içeriğinin her bir öğesinin birer **StackPanel** olduğunu ve **CategoryId** ile **Name** alanlarının içeriklerinin **TextBlock**' ların **Text** özelliklerine bağlandıklarını görebiliriz. Benzer durum **ListBox** kontrolü içinde geçerlidir. Ne varki **ListBox** kontrolünün içerisinde yer alan ve **Text** özellikleri **Book Entity**' sinin **Name** ile **ListPrice** alanlarına bağlanmış olan **TextBox** kontrolleri, aslında kullanıcı tarafından düzenlenebilirde. Volaaaa!!! öyleyse akla şu gelebilir.

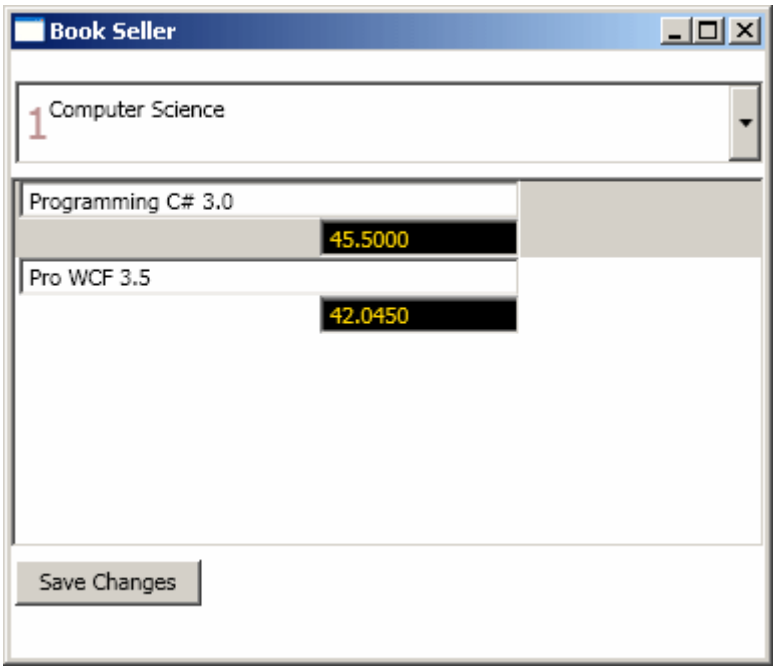
"Eğer bu kontrollerin Text özellikleri, kod tarafındaki Entity örneklerine bağlanmışlarsa ve çalışma zamanında içerikleri değiştirilirse bu düzenlemeler Entity içeriklerine de yansır mı? Peki diyelim ki yansıdı. SaveChanges metodunu çağırdığımızda bu değişikliler servis tarafına da yansır mı?"

Aslında bu sorularının tamamının cevabı **Evet**' tir. Ama tabiki bizim bu durumu analiz etmemiz ve gözümüzle görmemiz şart. Buna göre kod içeriğimizi aşağıdaki gibi geliştirmemiz yeterlidir.

```
using System;
using System.Data.Services.Client;
using System.Windows;
using BookShopModel;

namespace BookSeller
{
   public partial class Window1
        : Window
   {
```

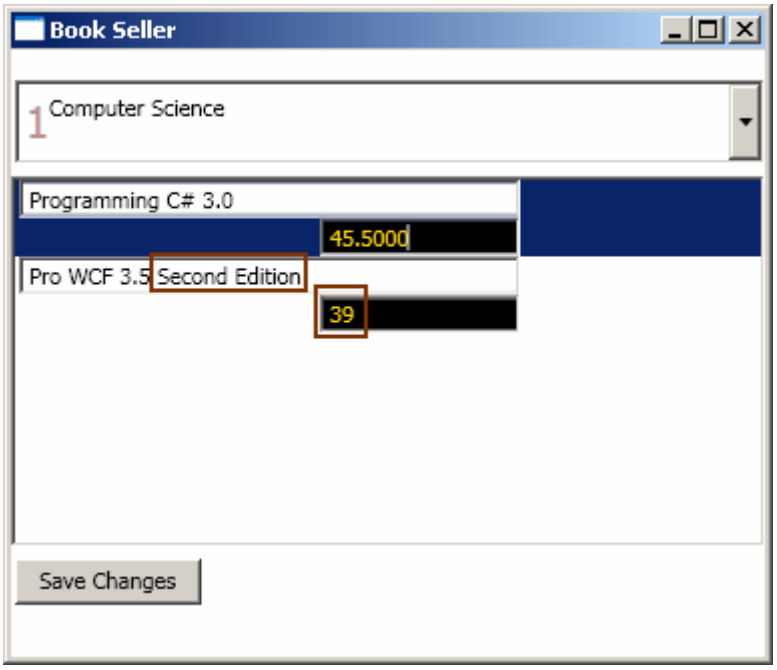
```
        BookShopEntities bs = null;
        DataServiceCollection<Category> _bookShopCollection = null;

        public Window1()
        {
            InitializeComponent();

            bs = new BookShopEntities(new
Uri("http://localhost:7995/BookService.svc/"));
            _bookShopCollection = DataServiceCollection.CreateTracked<Category>(bs,
bs.Category.Expand("Book"));
            grdBook.DataContext = _bookShopCollection;
        }

        private void btnSaveChanges_Click(object sender, RoutedEventArgs e)
        {
            bs.SaveChanges();
        }
    }
}
```

Programı ilk çalıştırdığımızda aşağıdaki bilgilerin geldiğini görebiliriz. Her ne kadar tasarım konusunda zayıf bir örnek olsada, **ListBox** içerisinde her bir satırda düzenlenebilir, veriye bağlanmış kontrollerin yer alması dahi benim için önemli bir adımdır. 🙂

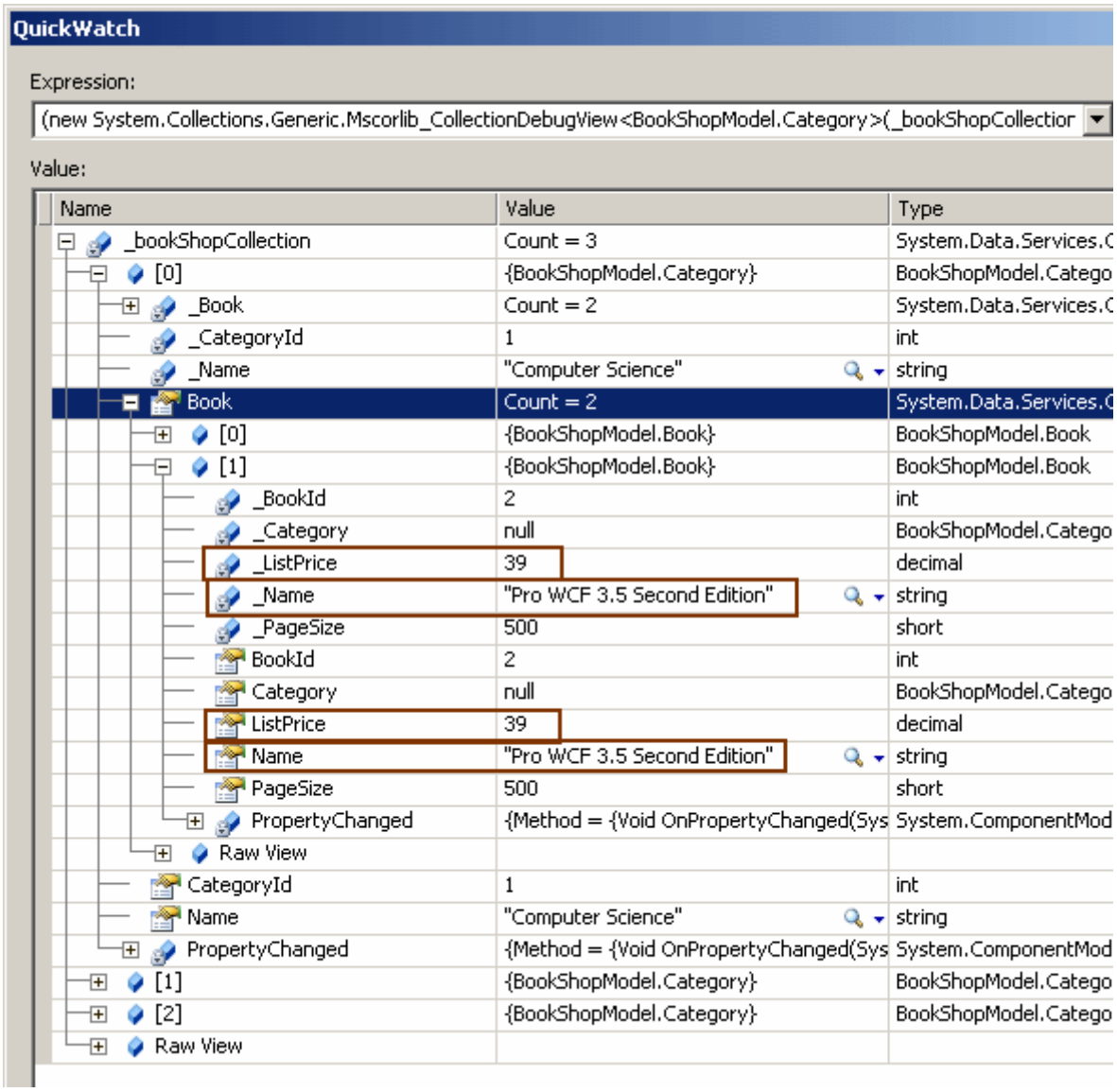


Şimdi **ListBox** içerisindeki verilerde değişiklik yapıldığını varsayalım. örnek olarak **Pro WCF 3.5** isimli kitabın adına **Second Edition** kelimelerini ilave ettiğimizi

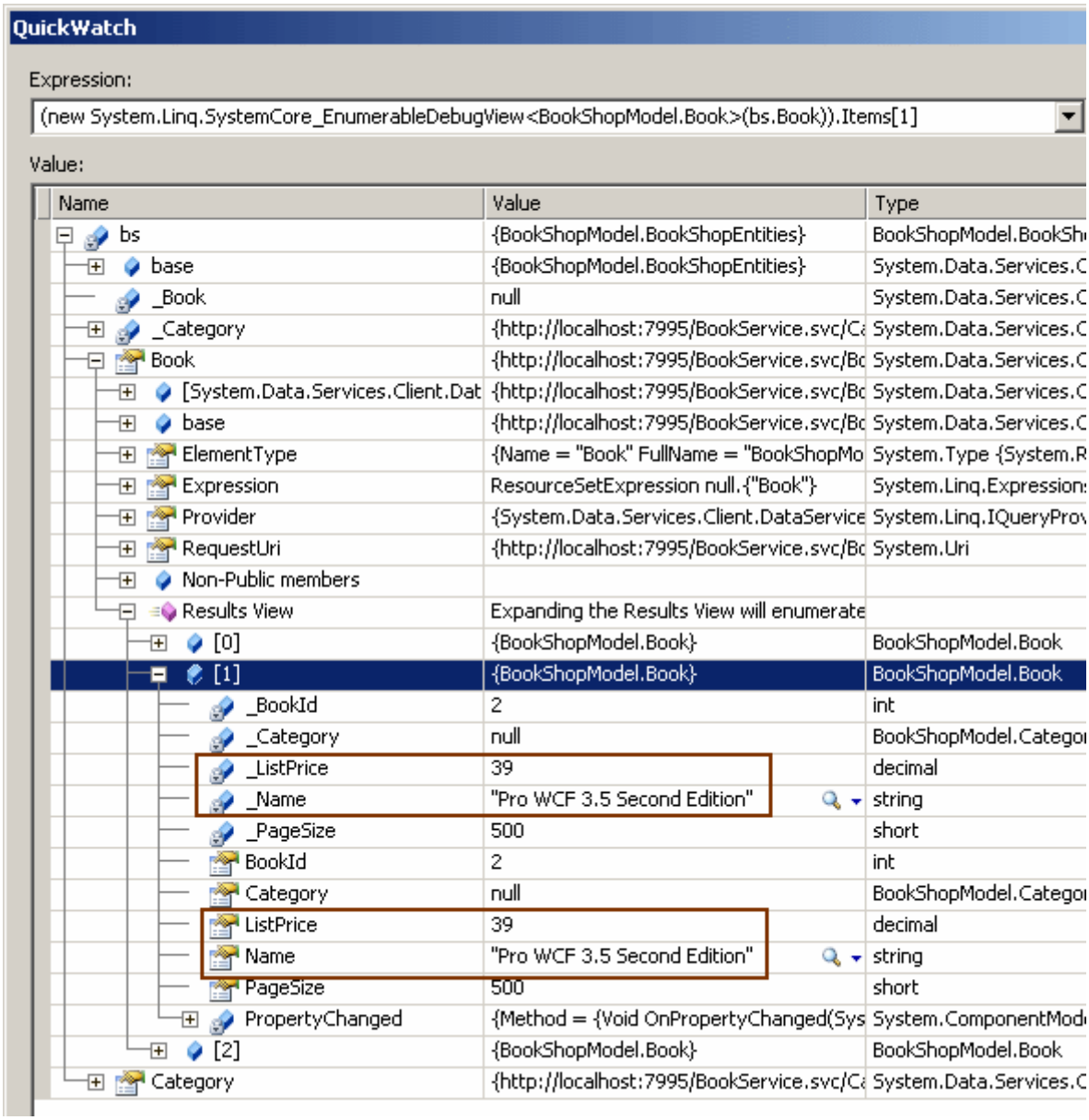
ve **ListPrice** değerini**39** birim olarak değiştirdiğimizi düşünebiliriz. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.



Şimdi **Save Changes** başlıklı düğmeye basalım ve **BookShopEntities** nesne örneği üzerinden **SaveChanges** metodunun çağırıldığı yerde koyduğumuz **break point** üzerinde duralım. Yapamamız
gereken, **BookShopEntities** ve **DataServiceCollection** içeriklerini **Watch** ile incelemek olacaktır. İlk olarak **_bookShopCollection** içeriğine bir bakalım.



Hımmmmm... Görüldüğü üzere **Category** üzerinden gittiğimiz **Book** özelliğine bağlı koleksiyonda az önce yapılan **Name** ve **ListPrice** değişikliklerin gerçekleştirildiği gözlemlenmektedir. Benzer şekilde **BookShopEntities** nesne örneğinin içeriğine baktığımızda, ilgili **Book** örneği için aynı değişikliklerin yansıtıldığını görebiliriz.

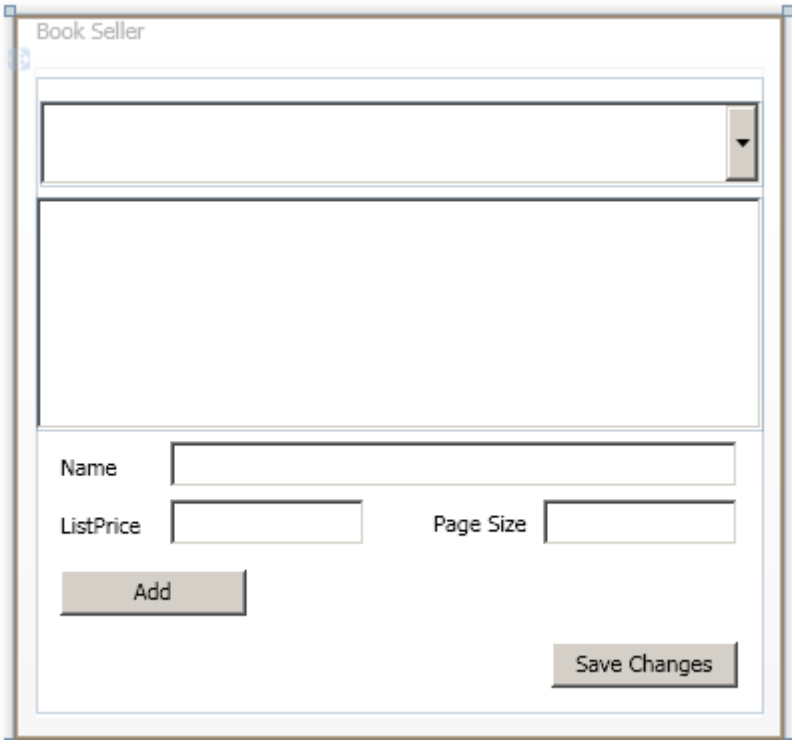


Dolayısıyla **iki yönlü veri bağlama(Two Way DataBinding)** nedeniyle kontroller üzerine yansıyan verilerde yapılan değişiklikler, istemci tarafındaki ilgili koleksiyonlara da yansıtılmaktadır. Buna göre **SaveChanges** metoduna yapılan çağrı geçildiğinde, servis tarafına gerekli güncelleştirme talebinin gittiği ve aşağıdaki **SQL** sorgusunun çalıştırıldığı gözlemlenir.

```
exec sp_executesql N'update [dbo].[Book]
set [Name] = @0, [ListPrice] = @1, [PageSize] = @2
where ([BookId] = @3)
',N'@0 nvarchar(26),@1 decimal(19,4),@2 smallint,@3 int',@0=N'Pro WCF 3.5 Second Edition',@1=39.0000,@2=500,@3=2
```

Eğer birden fazla **Book** örneğinin özelliği değiştirilirse, **SaveChanges** metodu çağrısı sonrasında her bir güncelleştirme için ayrı bir **SQL Update** sorgusunun çalıştırıldığı da

gözlemlenecektir. Şimdi örnek bir veri ekleme işlemi yapalım. Bu amaçla **Window1** içeriğini aşağıdaki gibi değiştirdiğimizi düşünebiliriz.



Yapılan bu değişikliklerden sonra ise **XAML** içeriğinin aşağıdaki gibi oluştuğu gözlemlenebilir.

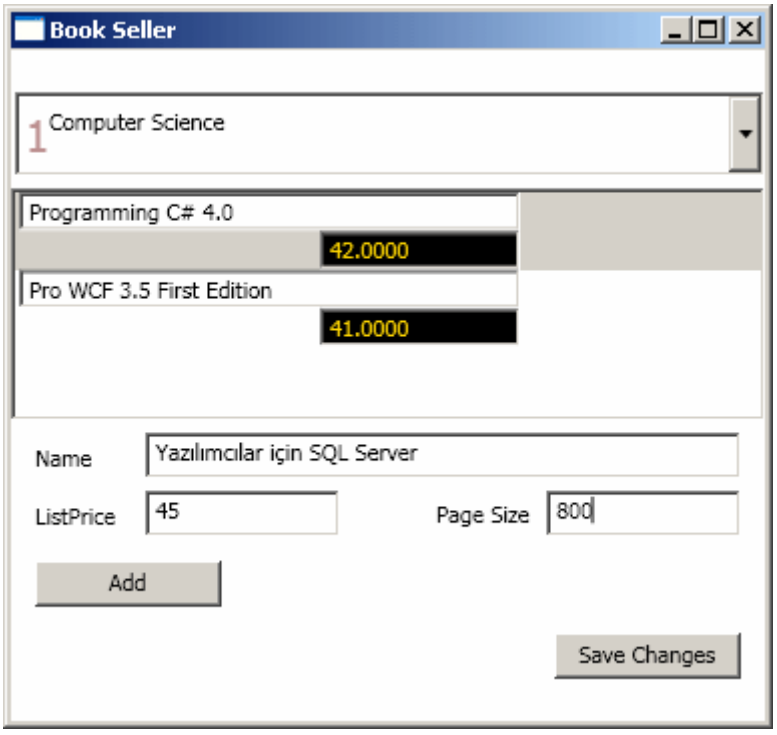
```
<Window x:Class="BookSeller.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Book Seller" Height="362" Width="384">
   <Grid Name="grdBook" Height="317">
      <ComboBox Height="42" Margin="2,12,0,0" Name="cmbCategories"
VerticalAlignment="Top"  IsSynchronizedWithCurrentItem="true"
ItemsSource="{Binding}">
         <ComboBox.ItemTemplate>
            <DataTemplate>
               <StackPanel Orientation="Horizontal">
                  <TextBlock Name="txtCategoryId" Text="{Binding Path=CategoryId}"
FontSize="22" Foreground="RosyBrown"/>
                  <TextBlock Name="txtCategoryName" Text="{Binding Path=Name}"/>
               </StackPanel>
            </DataTemplate>
         </ComboBox.ItemTemplate>
      </ComboBox>
      <ListBox IsSynchronizedWithCurrentItem="True" Name="lstProducts"
Margin="0,60,0,141" ItemsSource="{Binding Book}">
```

```
        <ListBox.ItemTemplate>
          <DataTemplate>
            <StackPanel Orientation="Vertical">
              <TextBox Name="txtProductName" Text="{Binding Name}"
Width="250"/>
                <TextBox Name="txtListPrice" Text="{Binding ListPrice}" Width="100"
Foreground="Gold" Background="Black" HorizontalAlignment="Right"/>
            </StackPanel>
          </DataTemplate>
        </ListBox.ItemTemplate>
      </ListBox>
      <Button Height="23" HorizontalAlignment="Right" Margin="0,0,12,12"
Name="btnSaveChanges" VerticalAlignment="Bottom" Width="93"
Click="btnSaveChanges_Click">Save Changes</Button>
      <Label Height="28" HorizontalAlignment="Left" Margin="7,0,0,106"
Name="label1" VerticalAlignment="Bottom" Width="54">Name</Label>
      <Label Height="28" HorizontalAlignment="Left" Margin="7,0,0,77" Name="label2"
VerticalAlignment="Bottom" Width="54">ListPrice</Label>
      <Label Height="28" HorizontalAlignment="Right" Margin="0,0,104,78"
Name="label3" VerticalAlignment="Bottom" Width="65">Page Size</Label>
      <TextBox Height="23" Margin="67,0,12,112" Name="txtName"
VerticalAlignment="Bottom" />
      <TextBox Height="23" Margin="67,0,0,83" Name="txtListPrice"
VerticalAlignment="Bottom" HorizontalAlignment="Left" Width="97" />
      <TextBox Height="23" Margin="0,0,12,83" Name="txtPageSize"
VerticalAlignment="Bottom" HorizontalAlignment="Right" Width="97" />
      <Button Height="23" HorizontalAlignment="Left" Margin="12,0,0,48"
Name="btnAddBook" VerticalAlignment="Bottom" Width="93"
Click="btnAddBook_Click">Add</Button>
    </Grid>
</Window>
```

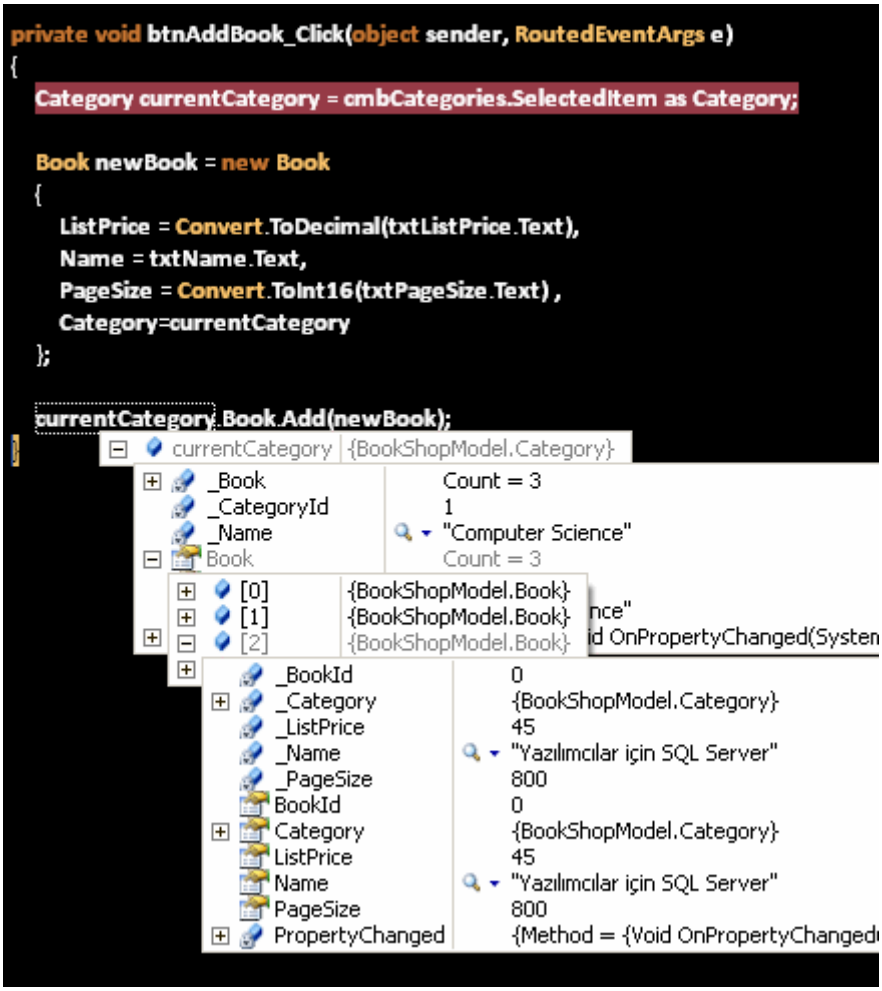
Kullanıcı bu forma göre yeni bir kitap ekleyebilmelidir. Bir kitabın bir kategori altında olması gerektiğinden, oluşturulacak kitabın hangi kategoriye ekleneceğinin belirlenmesi sırasında ComboBox' ta seçili olan **Category** nesne örneğinden yararlanabiliriz. Kitabın tabiki öncelikle nesnel olarak oluşturulması gerekmektedir. Sonrasında ise **ComboBox'** ta seçili olan **Category**öğesinin **Book** özelliği yardımıyla ilgili koleksiyona eklenmelidir. Bu ekleme işleminin ardından yapılacak olan **SaveChanges** çağrısı, ekleme işlemi için **Ado.Net Data Service** tarafına uygun talebin gönderilmesini sağlayacaktır. Bunun doğal sonucu olarakta sunucu tarafında uygun olan **SQL Insert** sorgusu çalıştırılacaktır. Bakalım gerçekten böyle mi? 🙂 Bu amaçla **Add** başlıklı **Button** kontrolümüzün **Click** olay metodunu aşağıdaki gibi kodladığımızı düşünelim.

```
private void btnAddBook_Click(object sender, RoutedEventArgs e)
    {
      Category currentCategory = cmbCategories.SelectedItem as Category;

      Book newBook = new Book
      {
        ListPrice = Convert.ToDecimal(txtListPrice.Text),
        Name = txtName.Text,
        PageSize = Convert.ToInt16(txtPageSize.Text) ,
        Category=currentCategory
      };

      currentCategory.Book.Add(newBook);
    }
```

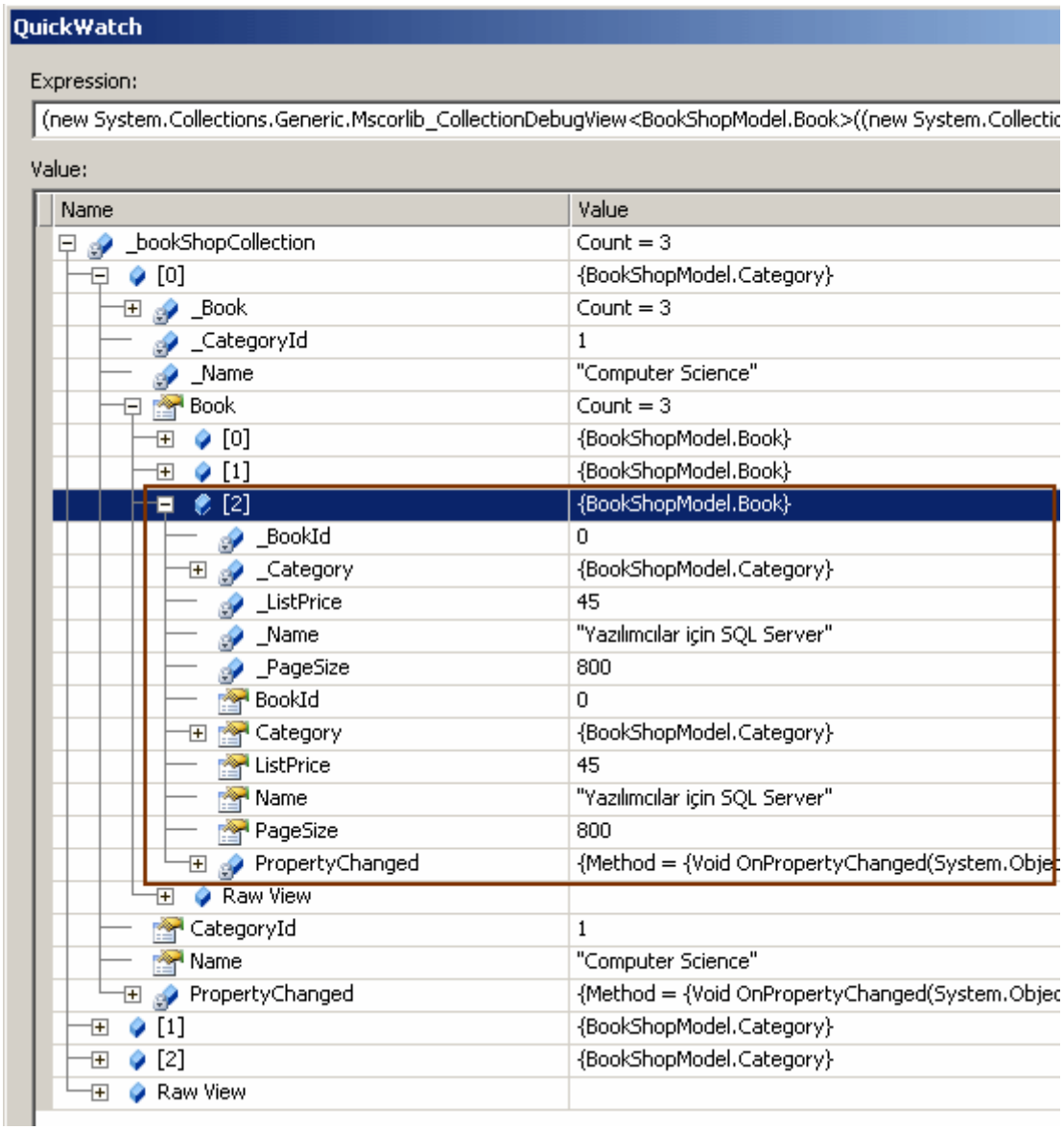
İlk olarak kitabın ekleneceği kategori bulunmaktadır.
Burada **SelectedItem** özelliğinin **Category** tipine dönüştürüldüğüne dikkat edilmelidir. Sonrasında yeni bir **Book** nesnesi örneklenir ve ilgili özellikleri kontrollerden alınır.*(Burada herhangibir hatalı giriş kontrolü yapmadığımızı belirtelim. Aslında yapmamız gerekiyor ancak şu an için odaklanmamız gereken kısım bu değil. Yinede siz örneği denerken mutlaka olası hataların önüne geçmenizi sağlayacak eklemeleri yapmayı unutmayın)* Dikkat edilmesi gereken noktalardan biriside, **Book** nesnesi örneklenirken **Category** özelliğine **currentCategory** değişkeninin referansını atamamızdır. Bundan sonraki kısım ise son derece basittir. örneklenen **Book** nesne örneği, o an seçili olan **Category** nesnesinin **Book** özelliğinin temsil ettiği koleksiyona atanır. Birde çalışma zamanına bakalım. Aşağıdaki örnekte bir veri girişi yapılmak istendiğini görüyoruz.



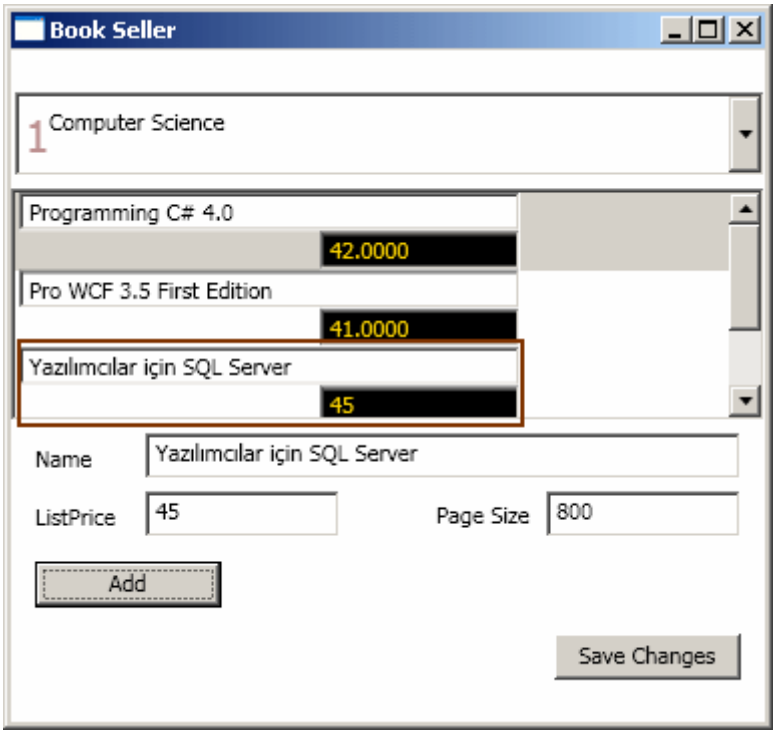
Şimdi **Add** düğmesine basarsak ve kodu **debug** edersek **currentCategory** değişkeninin içeriğinin aşağıdaki gibi güncellendiğini görürüz.



Görüldüğü gibi yeni **Book** nesne örneği ilgili kategori altına eklenmiştir.
öyleyse birde **DataServiceCollection** içeriğimize bakalım.



Yeni eklenen **Book** nesne örneğinin **DataServiceCollection** nesne örneği içerisinde de yer aldığı gözlemlenmektedir. üstelik çalışma zamanının yeni görüntüsüde aşağıdaki gibidir.



Bağlılık buna denir desek yeridir Nitekim yapmış olduğumuz nesne eklemesinden **ListBox** kontrolüde otomatik olarak etkilenmiş ve içeriğini yenilemiştir. Artık **Save Changes** başlıklı düğmeye basarak değişiklikleri servis tarafına gönderebiliriz. Bu işlem yapıldığı takdirde **SQL** tarafında aşağıdaki sorgunun çalıştırıldığı gözlemlenir.

```
exec sp_executesql N'insert [dbo].[Book]([Name], [ListPrice], [CategoryId], [PageSize])
values (@0, @1, @2, @3)
select [BookId]
from [dbo].[Book]
where @@ROWCOUNT > 0 and [BookId] = scope_identity()',N'@0 nvarchar(28),@1
decimal(19,4),@2 int,@3 smallint',@0=N'Yazılımcılar için SQL
Server',@1=45.0000,@2=1,@3=800
```

İşte bu kadar...Sırada silme işlemi var ama uzun olan bir yazının verdiği yorgunluğu yaşayan ben bu kutsal görevi siz değerli okurlarıma bırakıyorum Silme operasyonunu uygularken**debug** işlemlerini yapmayı ve çalışma zamanını analiz edip **SQL** tarafında neler olup bittiğini incelemeyi unutmayın. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

BindingV2.rar (93,31 kb)

Task Parallel Library(TPL) - Detached Tasks [Beta 2] (2009-11-12T15:00:00)

task parallel library,

Merhaba Arkadaşlar,

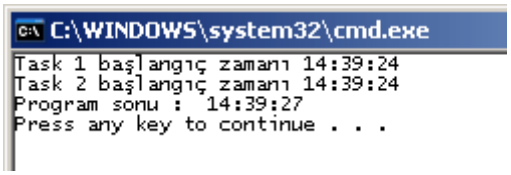
Bir önceki yazımızda **Task Parallel Library** tarafında **.Net Framework 4.0 Beta 2** tabanlı olarak **iptal işlemleri(Task Cancellation)** için yapılan değişikliklere değinmeye çalışmıştık. **TPL** tarafında yapılan değişikliklerden birisi de iç içe çalışan **Task'** ler arasındaki **Parent - Child** ilişkiye yönelik olarak yapılmıştır. Aslında basit bir davranış değişikliği olduğunu söyleyebiliriz. Konuyu daha net kavramak amacıyla aşağıdaki örnek kod parçasını göz önüne alalım.

```
using System;
using System.Threading;
using System.Threading.Tasks;

namespace DetachedTasks
{
   class Program
   {
      static void Main(string[] args)
      {
         Task task1 = Task.Factory.StartNew(() =>
         {
            Console.WriteLine("Task 1 başlangıç zamanı
{0}",DateTime.Now.ToLongTimeString());
            Task task2 = Task.Factory.StartNew(() =>
               {
                  Console.WriteLine("Task 2 başlangıç zamanı {0}",
DateTime.Now.ToLongTimeString());
                  Thread.Sleep(6000);
               }
            );
            Thread.Sleep(3000);
         }
         );

         task1.Wait();
         Console.WriteLine("Program sonu :  {0}",DateTime.Now.ToLongTimeString());
      }
   }
}
```

öncelikli olarak kodumuzda neler yaptığımıza bir bakalım. örnekte iki ayrı **Task** nesnesinin kullanıldığı görülmektedir. **task2** isimli nesne örneği, **task1** içerisinde üretilmekte ve kullanılmaktadır. Buna göre **task1'** in, **task2'** nin **parent'** ı olması gerektiğini düşünebiliriz. Aslında önemli olan nokta **task1** üzerinden yapılan **Wait** çağrısı sonucu programın nasıl davranacağıdır. Bu noktada **task1.Wait()** çağrısının **Beta 1** sürümünde farklı değerlendirildiğini belirtmemiz gerekiyor. Şöyle ki;

Beta 1 sürümüne göre **task2** otomatik olarak **task1'** e bağımlı hale getirilmekteydi**(Attached Task)**. Yani **task1** üzerinden yapılan **Wait** çağrısı sonucu **task1'** in tamamlanması beklenirken, içeride çalışmakta olan **task2'** ninde çalışmasını tamamlanması gerekmekteydi. Bu varsayılan davranış biçimiydi. Ancak bazı durumlarda **task1** beklenirken, içerisinde yer alan **task2'** nin beklemesi istenmeyebilir. Bir başka deyişle **Wait** çağrısı sonrasına geçilmesi için **task1'** içerisinde alt **Task'** ler tarafından yapılan işlemler dışında kalanların bitirilmesinin yeterli olması istenebilir. Beta 1' de bu durumu gerçeklemek için **TaskCreationOptions.DetachedFromParent enum** sabiti değerinde yararlanılmaktaydı(ki Beta 2' de kaldırıldı). Ancak Beta 2 sürümünde durum değiştirildi.

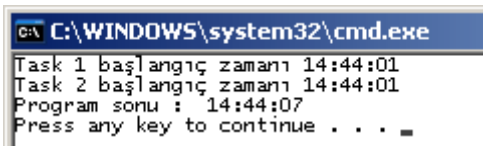
Beta 2 sürümüne göre varsayılan olarak alt **Task** nesne örnekleri içerisinde yer aldıkları üst **Task'** lere bağlı değildir*(Varsayılan olarak **Detached**)*. Açıkçası, varsayılan olarak **Task'** lerin **Detached** olarak tesis edildiklerini ifade edebiliriz. Buna göre yukarıdaki kodun çalışma zamanı çıktısı Beta 2 sürümü için aşağıda görüldüğü gibi olacaktır.



Dikkat edileceği üzere **task1** ile aynı zaman dilimi içerisinde **task2** başlatılmış ancak **task2** nin tamamlanması beklenmeden **task1'** deki işlemler bittiği için program sonuna gelinmiştir. Nitekim **task2** varsayılan olarak **task1'** e bağlanmadığından**(Detached) task1.Wait** çağrısı gerçekten sadece **task1'** in işleyişinin tamamlanması ile ilgilenmiştir. Peki **task2'** nin **task1'** e bağlanması için ne yapılmalıdır? Bu amaçla **task2** nesne örneğinin üretildiği yerde **TaskCreationOptions.AttachedToParent enum** sabiti değerinin kullanılması gerekmektedir. Dolayısıyla kodu aşağıdaki gibi güncellememiz yeterli olacaktır.

```
Task task1 = Task.Factory.StartNew(() =>
        {
           Console.WriteLine("Task 1 başlangıç zamanı
{0}",DateTime.Now.ToLongTimeString());
           Task task2 = Task.Factory.StartNew(() =>
              {
                 Console.WriteLine("Task 2 başlangıç zamanı {0}",
DateTime.Now.ToLongTimeString());
                 Thread.Sleep(6000);
              },TaskCreationOptions.AttachedToParent
           );
           Thread.Sleep(3000);
        }
        );
```

Kodun bu şekilde çalıştırılması sonucu aşağıdaki ekran çıktısı ile karşılaşılacaktır.



AttachedToParent değeri nedeni ile **task2**, **task1'** e bağlı hale getirilmiştir. Yani **task1**, **task2'** nin **parent Task'** i olarak belirlenmiştir. Bu durumda **task1.Wait** çağrısının yapıldığı noktada **Attach** edilmiş tüm **Task** referansları değerlendirileceğinden **task1'** inde tamamlanması beklenilmiştir. Bu durum her iki çalışmadaki **Program Sonu** sürelerinden anlaşılabilmektedir. Nitekim ilk çalışmada **task2** içerisindeki **6 saniyelik Sleep** çağrısı hesaba katılmazken, ikinci örnekte katılmıştır. **Task Parallel Library** ile ilişkili olarak **Beta 2'** de gelen diğer değişiklikleri ele aldığımız başka bir yazımızda görüşmek dileğiyle, hepinize mutlu günler dilerim.

Task Parallel Library(TPL) - İptal İşlemi [Beta 2] (2009-11-12T11:00:00)

task parallel library,

Merhaba Arkadaşlar,

Uzun süredir **.Net Framework 4.0'** ın bir parçası olarak gelen paralel programlama alt yapısı ile uğraşmıyordum. En son **Beta 1** sürümündeyken **Task Parallel Library** ve **PLINQ** ile ilişkili konulara bakma fırsatım olmuştu. Zaman ilerledi ve **.Net Framework 4.0 Beta 2** sürümü yayınlandı. Bu sürümde **Beta 1'** e göre bazı farklılıklar bulunmakta. Yani farklılıkları yeniden öğrenme aşamasına gelmiş durumdayız. Bunu **WF** tarafında, **WCF** tarafında gördüğümüz gibi halen gelişmekte olan Paralel programlama alt yapısında da görmekteyiz. İşte bu günkü yazımızda herhangibir **Task'** in iptal sürecinin **Beta 2** sürümünde nasıl değerlendirildiğini incelemeye çalışıyoruz olacağız. **Beta 1**sürümünde bir **Task'** in iptal edilmesi için aşağıdaki kod tasarımı kullanılmaktaydı.

```
Task parallelTask= Task.Factory.StartNew(() =>
{
 for (; ; )
 {
  if (Task.Current.IsCancellationRequested)
  {
   Task.Current.AcknowledgeCancellation();
   return;
  }
  //TODO: Gerekli işlemler
 }
});
```

ve kodun herhangibir yerinde **Task'** in iptal edilmesi için ilgili **Task** nesne örneği üzerinden **Cancel** metodu çağırılmaktaydı.

parallelTask.**Cancel();**

Bu modelde bir **Task'** in iptal istemi geldiğinde, çalışmakta olan güncel **Task'** in üzerinden **IsCancellationRequested** özelliğinin değerine bakılması gerekmekteydi. Eğer bu özelliğin değeri **true** ise bu durumda ilgili **Task'** in iptal işlemi ile ilişkili olaraktan bilgilendirilmesi amacıyla **AcknowledgeCancellation** metodu çağırılmakta ve örneğin **return** gibi bir çağrı ile paralel yüreyen operasyondan çıkılmaktaydı. Ancak **Beta 1** sürümündeki bu yaklaşımın bazı handikapları da bulunmaktaydı. Söz gelimi iptal isteğinin kontrolü sırasında yürüyen süreci kesmek için return gibi bir çıkış kullanılması, Task tipi üzerinden static Current özelliğne gidilmek zorunda kalınması, iptal ile ilişkili tüm fonksiyonellik ve özelliklerin Task tipi üzerinde yer alması vb...Bu sebeplerden dolayı **Beta 2** sürümünde bir **Task'** in iptal edilme işlemi yeniden değerlendirelerek daha tutarlı bir hale getirildi. Buna göre yeni modeli aşağıdaki örnekte görüldüğü üzere özetleyebiliriz.

```
using System;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace TaskParallelLibrary
{
   public partial class Form1 : Form
   {
      CancellationTokenSource source;
      CancellationToken token;
      Task parallelTask;

      public Form1()
      {
         InitializeComponent();

         source = new CancellationTokenSource();
         token = source.Token;
      }

      private void btnStart_Click(object sender, EventArgs e)
      {
         parallelTask = Task.Factory.StartNew(() =>
         {
            for (int i = 1000; i < 1000000; i++)
            {
```

```
                // Eğer Cancel isteği gelirse OperationCancelled istisnası fırlatılır.
                token.ThrowIfCancellationRequested();
                // Burada bir takım işlemler yapılmakta olduğunu düşünebiliriz
            }
        }
        , token
        ); // StarNew metodunda kullanılan ikinci parametre ile CancellationToken
referansının aktarıldığında dikkat edelim.
    }

    private void btnCancel_Click(object sender, EventArgs e)
    {
        source.Cancel(); // İptal işlemi için Task referansı yerine CancellationTokenSource
referansı kullanılmaktadır.
    }
  }
}
```

Beta 2 ile gelen iptal modelinde **CancellationTokenSource**, **CancellationToken** isimli yeni tiplerin kullanıldığı görülmektedir. Dikkat çekici noktalardan birisi, **Task'** in başlatılması işlemi sırasında ikinci parametre olarak bir **CancellationToken** referansının gönderilmesidir. Bu referanstan yararlanılarak **for** döngüsü içerisinde **ThrowIfCancellationRequested** metodu çağırılmaktadır. İşte bir yenilik daha. Bu metod, token referansı ile ilişkilendirilmiş olan **Task'** e bir iptal isteği gelip gelmediğini kontrol etmektedir. Eğer böyle bir istek gelmişse**OperationCancelledException** tipinden olan istisnayı fırlatmakta ve çalışmakta olan **Task'** in iptal edilmesine neden olmaktadır. Bir önceki modelde görüldüğü gibi bir **if** kontrolü yapılmasına ve **return** gibi bir çıkış yolu kullanılmasına gerek kalmamaktadır. Bir diğer önemli noktada, iptal işlemi için **Task** referansı yerine **CancellationTokenSource** nesne örneğinin kullanılıyor olmasıdır.

İptal işlemi ile ilişkili üyelerin tamamı **Task** tipinden çıkartılmıştır**(Cancel,AcknowledgeCancellation,IsCancellationRequested,CurrentTask vb...) Cancel** çağrısı için**CancellationTokenSource** , **ThrowIfCancellationRequested** çağrısı ile iptal kontrolü ve operasyonun kesilmesi için **CancellationToken** referanslarının kullanıldığına dikkat edelim. Buna göre bir iptal işlemi, aynı **CancellationToken** referansını kullanan birden fazla **Task'** ede uygulanabilir. Zaten bu amaçla, **StartNew** gibi bazı metodların yeni aşırı yüklenmiş versiyonlarına **CancellationToken** referansının taşınabiliyor olması sağlanmıştır.

Bakalım **Beta 2** sürümüne göre paralel programlama alt yapısında başka ne gibi yenilikler bulunmaktadır. Bunları ilerleyen yazılarımızda incelemeye devam ediyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Microsoft.Net Services - Service Bus için Hello World (2009-11-11T09:45:00)

windows azure,service bus,wcf,

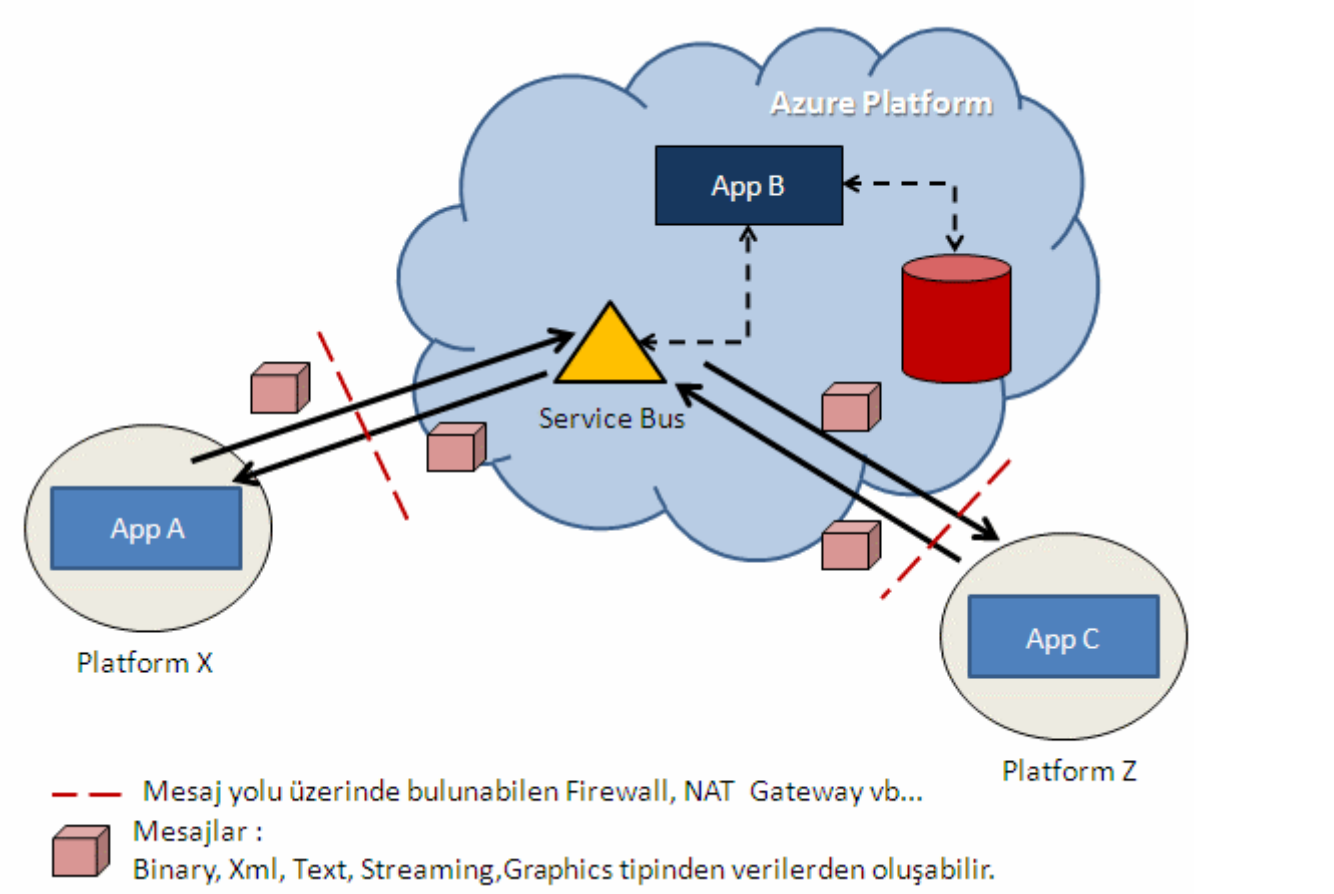
Merhaba Arkadaşlar,

Bu gün itibairyle **İstanbul'** da sağnak yağışlı bir hava hakim. Oysaki bir kaç güne kadar güneşli ve ılıman bir hava vardı. Hal böyle olunca **Cloud Computing** ile ilgili bir şeyler karıştırmanın tam vaktidir diye düşündüm. Daha önceki Windows Azure Service Platformu Hakkında İlk İzlenimler başlıklı yazımızda Amazon' dan **Cloud Computing with Windows Azure Platform** isimli bir kitabı sipariş ettiğimi ve önümüzdeki dönemlerde bu konu ile ilişkili yazılarımı sizlerle paylaşacağımı belirtmiştim. Ne var ki aradan geçen uzun süreye ve**Amazon'** dan kitabı elden teslim eden kurye ile sipariş etmeme rağmen kitap elime bir türlü ulaşmadı 😕 ve Amerika' da bir çıkış noktasında takıldı kaldı. Oysaki kütüphanede yer alan sayısız kitap hep sorunsuz ulaşmıştı. Bir kargo firması bir Amazon ile yazışma derken arada gidip gelen sayısız elektronik postanın arkasından en azından paramı geri almaya başardım ama ne varki kitabıma ulaşamadım. Peki yıldım mı? Yılmadım. Bu kez siparişimi Amazon.co.uk sitesinden verdim*(Sterlin farklarını hesaba katmanızı öneririm)*. Kitap siparişin bir sonraki gününde **UPS** tarafından bana ulaştırıldı. En azından içim rahatladı ve kitabıma kavuştum.😊 Gel gelelim şu an okuduğunuz yazıyı yazmak için bu kitabı bir günde bitirmedim elbetteki. Aslına bakarsanız geçtiğimiz günlerde **.Net Services SDK'** sının Kasım sürümü ve çok doğal olarak dökümantasyonu yayınlandı. İşte bu yazımızda dökümantasyondan edindiğim ilk izlenimler ışığında geliştireceğimiz basit bir Hello World örneği yazmaya çalışıyor olacağız.

Windows Azure platformunun önemli parçalarından birisi olan **Microsoft .Net Services**, internet tabanlı uygulama servisleri olarak düşünülebilir. Bu anlmada internet tabanlı uygulamaların Cloud üzerinde yer alan uygulamalar veya **kaynaklar(Resources)** ile iletişimini servis bazlı olarak koordine edebilen bir servis alt yapısı olarak düşünülebilir. Şu aşamada **Microsoft .Net Service'**lerin iki uygulama biçimi vardır. **Service Bus** ve **Access Control Service**. özellikle **Firewall** arkasında kalan istemcilerin **Cloud** üzerinde yer alan bir uygulama ile haberleşmesi sırasında gerekli olan karmaşık konfigurasyon işlemleri, güvenlik gibi konuları **Service Bus** üzerine alarak kolay bir şekilde çözümlemeye çalışır. öyleki Firewall arkasında duran uygulamaların Cloud üzerindeki bir uygulama ile olan

haberleşmesinde genellikle port açılması veya VPN kullanılması yolu tercih edilir. Port açılması bir güvenlik riski doğurmakla birlikte, VPN kullanımında da sistem seviyesinde karmaşık konfigurasyon ayarları yapılması gerekmektedir. Oysaki Service Bus bu karmaşıklığı ele alarak istemcilerin Cloud üzerindeki uygulamalar ile konuşabilmesini kolaylaştırmaktadır. üstelin **Authentication** gibi işlevlere de sahiptir.

Service Bus çok basit anlamda birbirlerine **zayıf bağlı olan(Loosely Coupled)** uygulamaların güvenli bir şekilde iletişim kurabilmesini sağlamak, platforma bakılmaksızın gerekli karmaşık konfigurasyon ayarlarını sağlamak amacıyla kullanılmaktadır. İlk etapta **Windows Azure** platformu üzerinde yer alan **Azure** ve **SQL Service** uygulamaları ile olan iletişiminde kullanılabileceği düşünülebilir. Bu açıdan bakıldığında Azure platformu üzerindeki uygulamalar ve veritabanları ile haberleşilirken kullanılan bir servis alt yapısı olarakta görülebilir. Aşağıdaki şekil **Service Bus** alt yapısını bu açıdan değerlendirmektedir.



Aslında bu şekil bize şunları ifade etmektedir; **App A**, üzerinde bulunduğu **Platform**' daki **Firewall** gibi sorunlara takılmadan **Azure Platform**' u üzerinde konuşlandırılmış bir Cloud servisine, uygulamasına veya kaynağına(Resource) erişmek için **Service Bus** alt yapısını kullanmaktadır. Şekilde yer alan **App B**, Cloud üzerinde yer alan bir uygulama olarak düşünülmektedir. Buna göre **App A** ile **App B**' nin haberleşmesinde **Service Bus** gerekli bağlantıyı, koordinasyonu sağlamakta ve güvenli bir iletişimi tesis etmektedir.

Diğer taraftan **App A**yine **Service Bus** aracılığıyla başka bir platform üzerinde yer alan **App C** isimli uygulama ilede güvenli bir iletişim sağlayabilmektedir.

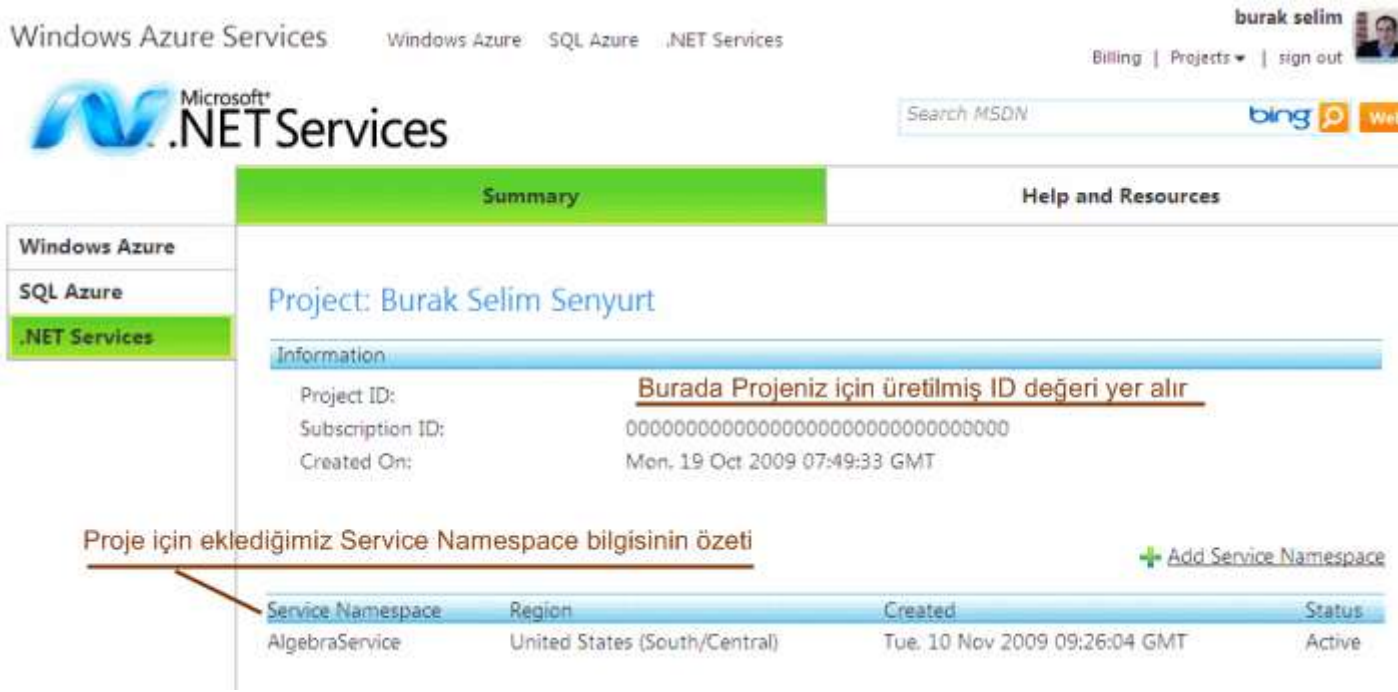
Peki **Service Bus** alt yapısının en belirgin özellikleri nelerdir?

- **Firewall, NAT Gateway** gibi unsurların arkasında duran uygulama ve servislerin birbirleriyle haberleşebilmelerini gerekli konfigurasyon ayarlamalarını üstüne aldığı için kolaylaştırır. Geliştiricinin veya tarafların hangi sistem gereksinimlerine ihtiyaçları olduğunu, bulundukları farklı platformlar arasındaki haberleşme sorunlarını düşünmelerine gerek yoktur.
- Standart **REST** modelini de desteklediğinden kolay bir şekilde kullanılabilir ama istenirse **WCF** bazlı yaklaşım değerlendirilerek profesyon programcılar tarafından değerlendirilmeside sağlanabilir.
- **.Net** platformundan olmayan uygulamalara **REST** ve **HTTP** tabanlı erişilebilmesini sağlar.
- Cloud üzerindeki servislere **Anonymous** kullanıcıların erişebilmesi, izin verildiği takdirde mümkündür.
- Servislerin internet üzerinden erişilebilen sabit **URL** adresleri ile **keşfedilmesi(Discovery)** lokasyona bakılmaksızın mümkündür.
- Servislere yapılan şüpheli saldırıların bloklanmasına yardımcı olur.

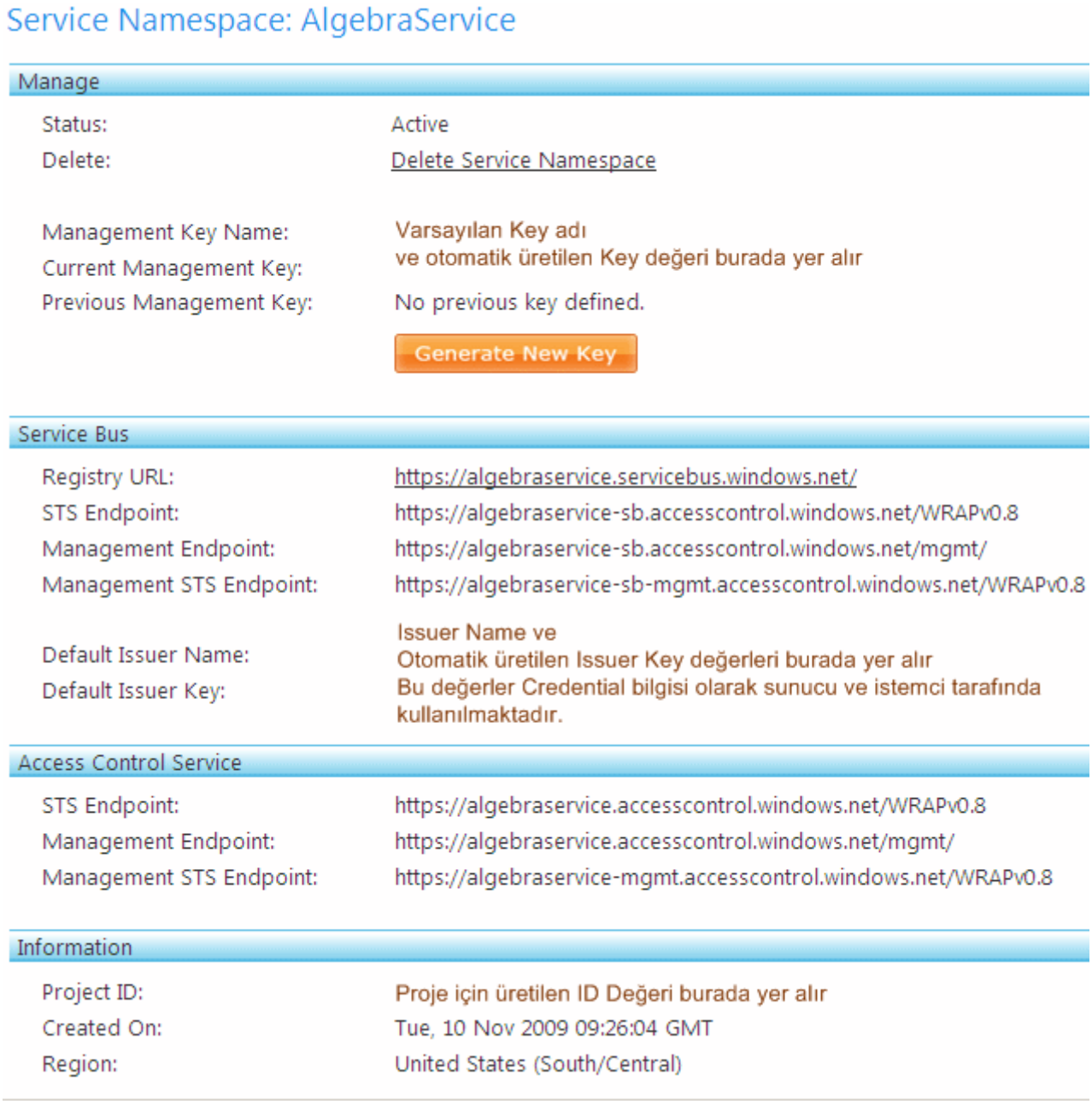
vb...

Bu kısa **How To** niteliğindeki yazımızda aynı makine üzerinde yer alan bir istemci ile yine aynı makine üzerinde yer alan ve servisi host eden başka bir uygulama arasında **Service Bus** aracılığıyla **Credential** bazlı bir iletişimin nasıl kurulabileceği incelenmektedir. Gerçek anlamda **Cloud** üzerindeki bir uygulama, servis veya kaynak ile haberleşilmese de durumu simule edebileceğimiz bir örnek olacaktır.

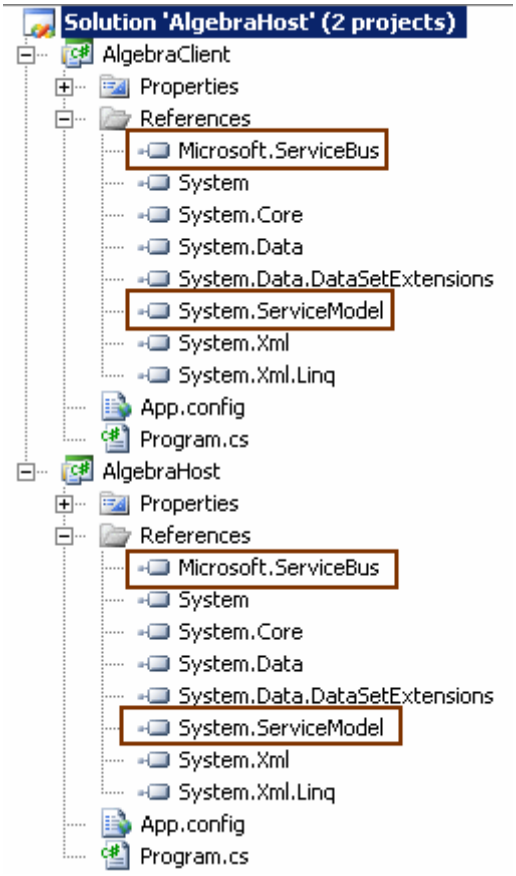
Tabiki işin daha çok detayı vardır ancak adet olduğu üzere konuyu kavramının en iyi yolunun çok basit bir örnek geliştirmekle mümkün olabileceği kanısındayım. İşlemlere başlamadan önce Microsoft .Net Services SDK' sının Kasım 2009 sürümünü indirmemiz ve sistemimize yüklememiz gerekmektedir. Bu **SDK** içerisinde **.Net Services** olanaklarından yararlabilmemiz için gerekli tipler ve üyeleri yer almaktadır. **SDK'** nın yüklenmesi yeterli değildir. Ayrıca **Windows Azure** platformu üzerinden bir servis hesabının açılması ve burada kullanacağımız **Service Bus** için bir **Namespace** bildirimi yapılması gerekmektedir. Söz konusu hizmetlerden şu an için ücretsiz yararlanılabilmekte olup **2010** içerisinde ücrete tabi olacağına dair bilgiler de yer almaktadır. *(Ancak yazıyı hazırladığım bu günlerde **Microsoft.Net Service'** leri ücretsiz olarak kullanılabilmekteydi.)* Kayıt işlemleri için https://netservices.azure.com/ adresinden yararlanılmaktadır. örneğin benim **Azure** üzerinde yer alan projemde bu yazımızdaki örnek için oluşturduğum **AlgebraService Namespace** ve bilgileri aşağıdaki şekilde görüldüğü gibidir.



Dilerseniz **Namespace** detaylarına da bakabilir gerekirse silerek hizmetten kaldırabilirsiniz. **Namespace** oluşturulduktan sonra istemci ve servis tarafının kullanacağı **ehliyet(Credential)** bilgilerinin de otomatik olarak oluşturulduğu görülecektir. **Service Bus** örneğimiz için bunlar **Default Issuer Name** ve **Default Issuer Key** değerleridir. Bu bilgiler **Service Namespace**' ine ait **Summary**bölgesinde görülebilmektedir.



örneğimizde servis ve istemci uygulamalar aynı makine üzerinde geliştirilecek olsalarda, aralarındaki **Credential** tabanlı iletişim için yukarıda bilgileri yer alan Service Bus hizmeti kullanılacaktır.**Service** ve **Host** uygulamalarımız **Visual Studio 2008** ortamında geliştirilmekte olan basit **Console** uygulamalarıdır ve her ikiside, **System.ServiceModel** ile **.Net Services SDK'** sının yüklenmesi sonrasında gelen **Microsoft.ServiceBus assembly'** larını referans etmektedir.



Servis tarafındaki kodlarımızı aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Description;
using Microsoft.ServiceBus;

namespace AlgebraHost
{
   // Servis sözleşmesi
   [ServiceContract(Namespace = "http://algebraservice/ServiceBus/")]
   public interface IAlgebraContract
   {
      [OperationContract]
      double Sum(double x, double y);
   }

   public interface IAlgebraChannel
      : IAlgebraContract, IClientChannel
   {
   }
```

```
// Sözleşmeyi uygulayan tip
[ServiceBehavior(Name="Algebra")]
public class AlgebraService
    : IAlgebraContract
{
    #region IAlgebraContract Members

    public double Sum(double x, double y)
    {
        return x + y;
    }

    #endregion
}


class Program
{
    static void Main(string[] args)
    {
        // Servis adresi elde edilir
        // İlk parametre schema, ikinci parametre Service Bus üzerinde oluşturulan Solution adı ve üçüncü parametrede servis yoludur
        Uri serviceUri = ServiceBusEnvironment.CreateServiceUri("sb", "AlgebraService", "AlgebraService");

        // İletişim için gerekli olan issuer adı ve şifresi bilgileri TransportClientEndpointBehavior nesne örneğinde toplanır
        TransportClientEndpointBehavior credential = new TransportClientEndpointBehavior();
        credential.CredentialType = TransportClientCredentialType.SharedSecret; // Credential tipi
        credential.Credentials.SharedSecret.IssuerName = "SİZİN ISSUER NAME DEÄžERİNİZİ"; // Issuer adı
        credential.Credentials.SharedSecret.IssuerSecret = "SİZİN İçİN üRETİLEN KEY DEÄžERİ// Issuer için otomatik üretilmiş olan key değeri

        // ServiceHost nesne örneklenir
        // İlk parametre servis tipidir
        // İkinci parametre ise servis adresidir
        ServiceHost host = new ServiceHost(typeof(AlgebraService), serviceUri);
        IEndpointBehavior serviceRegistrySettings = new ServiceRegistrySettings(DiscoveryType.Public);
```

```
        // Config dosyasında tanımlanan tüm Endpoint' lere gerekli Credential davranışı
eklenir.
        foreach (ServiceEndpoint endpoint in host.Description.Endpoints)
        {
           endpoint.Behaviors.Add(serviceRegistrySettings);
           endpoint.Behaviors.Add(credential);
        }

        // Servis açılır
        host.Open();

        Console.WriteLine("Servisin orjina adresi \n {0}:\n Service Durumu {1}
",serviceUri,host.State);
        Console.WriteLine("Kapatmak için bir tuşa basınız");
        Console.ReadLine();

        // Servis kapatılır
        host.Close();
      }
   }
}
```

Aslında standart bir **WCF** Servisi yazılmış ve **host** edilmiştir. Ancak **Service Bus** üzerinde tanımlı **Service Namespace'** in kullanılması içinde bir takım ek işlemler yapılmıştır. Söz gelimi bu iletişim için gerekli **Credential** bilgileri **TransportClientCredentialType** tipi yardımıyla servisin üzerindeki tüm **endPoint'** lere birer çalışma zamanı **davranışı(Behavior)**olarak bildirilmektedir. Buna göre servis, istemci ile olan tüm iletişiminde **Azure** üzerindeki projemiz için üretilen **Name** ve **Key** değerleri kullanılacaktır. Diğer yandan servisin adresi belirlenirken,**Service Namespace** bilgisinin kullanıldığı dikkatten kaçmamalıdır. Nitekim **Azure** üzerindeki ilgili servisin adresinin bilinmesi gerekmektedir. Bu amaçla **sb** isimli **Service Bus** schema adından da**CreateServiceUri** metodu içerisinde yararlanılmaktadır. Elbette servis tarafında **WCF çalışma zamanı(WCF Runtim)** için gerekli konfigurasyon ayarlarının da bildirilmesi gerekmektedir. Bu amaçla servis uygulamasına ait **App.config** dosyası aşağıdaki gibi geliştirilebilir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="AlgebraHost.AlgebraService">
        <endpoint
contract="AlgebraHost.IAlgebraContract" binding="netTcpRelayBinding"/>
      </service>
    </services>
```

```
  </system.serviceModel>
</configuration>
```

Dikkat çekici nokta **TCP** bazlı iletişim için kullanılan **bağlayıcı tiptir(Binding Type).**

Gelelim yine Console olarak tasarladığımız istemci uygulama tarafı kodlarına.

```
using System;
using System.ServiceModel;
using Microsoft.ServiceBus;

namespace AlgebraClient
{
   // Sözleşme tipi(Contract Type)
   [ServiceContract(Namespace = "http://algebraservice/ServiceBus/")]
   public interface IAlgebraContract
   {
      [OperationContract]
      double Sum(double x, double y);
   }

   public interface IAlgebraChannel
      : IAlgebraContract, IClientChannel
   {
   }

   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Başlamak için bir tuşa basınız");
         Console.ReadLine();

         // Bağlantı protokolünün (Http, Tcp) otomatik olarak belirleneceği bildirilir
         ServiceBusEnvironment.SystemConnectivity.Mode =
ConnectivityMode.AutoDetect;

         string sbNamespace = "AlgebraService";
         string sbIssuer = "SİZİN ISSUER NAME DEÄžERİNİZ";
         string sbIssuerKey = "SİZİN ISSUER KEY DEÄžERİNİZ";

         // Servis adresi üretilir
         Uri serviceUri = ServiceBusEnvironment.CreateServiceUri("sb",
sbNamespace, "AlgebraService");
```

```
        // İletişim için gerekli olan issuer adı ve şifresi TransportClientEndpointBehavior
tipinden nesne örneği ile bildirilir.
        TransportClientEndpointBehavior credential = new
TransportClientEndpointBehavior();
        credential.CredentialType = TransportClientCredentialType.SharedSecret;
        credential.Credentials.SharedSecret.IssuerName = sbIssuer;
        credential.Credentials.SharedSecret.IssuerSecret = sbIssuerKey;

        // İletişimin kanalını üretecek olan fabrika nesnesi oluşturulur. İlk parametre ile
config dosyasındaki Client Endpoint yeri belirtilir, ikinci parametre ilede Service Bus
üzerindeki adres bilgisi belirtilir
        ChannelFactory<IAlgebraChannel> channelFactory = new
ChannelFactory<IAlgebraChannel>("AlgebraEndpoint", new
EndpointAddress(serviceUri));
        // İletişimin hangi kullanıcı adı ve şifre ile gerçekleştirileceği ilgili
TransportClientEndpointBehavior nesne örneğinin davranış olarak bildirilmesiyle
gerçeklenir
        channelFactory.Endpoint.Behaviors.Add(credential);

        // İletişim kanalı oluşturulur ve açılır
        IAlgebraChannel clientChannel = channelFactory.CreateChannel();
        clientChannel.Open();

        // Sum operasyonuna çağrı yapılır
        Console.WriteLine("Sum Result {0} + {1} = {2} ",
2,4,clientChannel.Sum(2,4).ToString());

        // İletişim kanalı ve kanal üretme fabrikası kapatılır
        clientChannel.Close();
        channelFactory.Close();
      }
   }
}
```

İstemci tarafında da dikkat edileceği üzere Servis sözleşmesnin bir kopyası yer almaktadır. Nitekim çalışma zamanındaki **proxy** üretimi için servis sözleşmesinin sunduğu içeriğin bilinmesi gerekmektedir. Burada devreye ilgili servis sözleşmesini implemente eden kanal arayüz referansıda girmektedir. İstemci tarafı için yapılan adımlar aslında sırasıyla aşağıdaki gibidir;

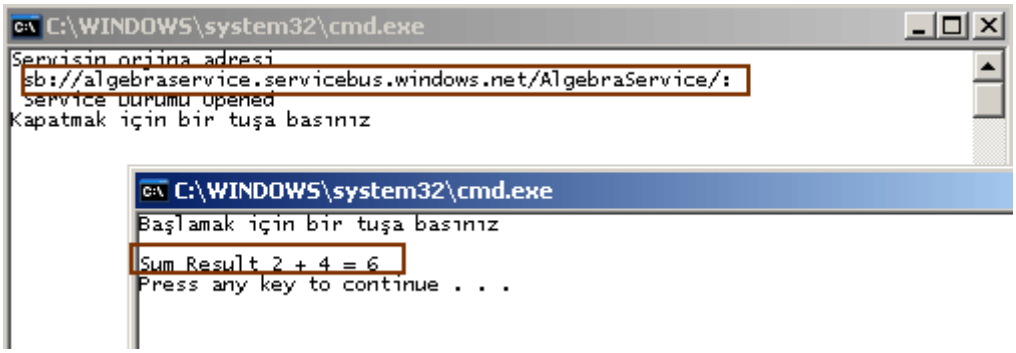
1. Bağlantı modu belirlenir***(ConnectivityMode)***
2. Host servisin adresi belirlenir*(Uri bilgisi belirlenirken Azure Projesi üzerinde oluşturduğumuz **Service Namespace** adı kullanılır)*
3. Gerekli **Crendetial** tanımlamaları yapılır ve bağlantı için uygulanması sağlanır***(TransportClientEndpointBehavior)***

4. Kanal oluşturulur ve **Credential'** ı değerlendirmesi **davranış(Behavior)** eklenmesi yardımıyla belirtilir
5. Kanal bağlantısı açılır.
6. Gerekli servis operasyonları icra edilir.*(Kobay olarak sıkça kulladığımız **Sum** operasyonu)*
7. Bağlantılar kapatılır.

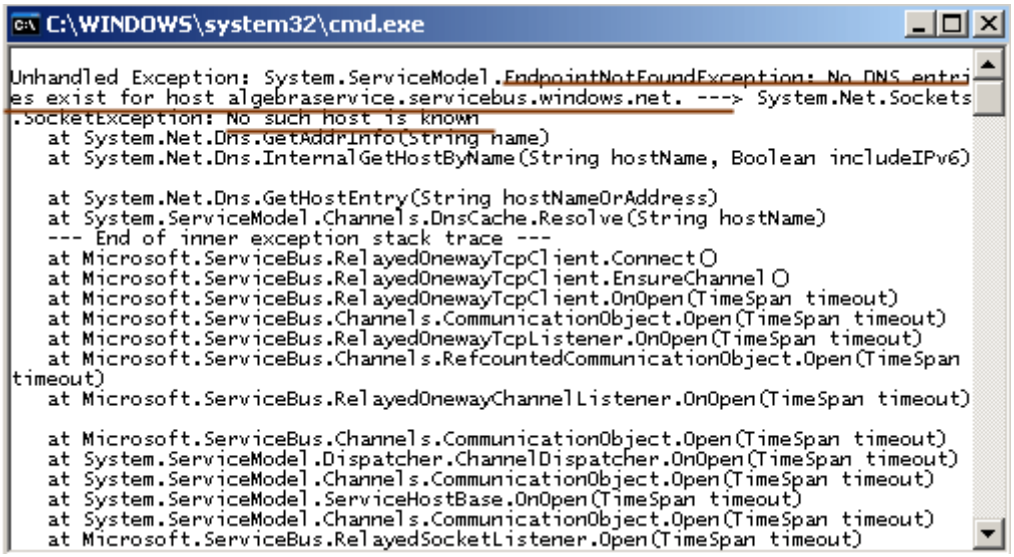
çok doğal olarak istemci uygulamanın **WCF çalışma zamanı** içinde bir takım konfigırasyon ayarlarının yapılması gerekmektedir. İşte istemci tarafı config dosyası içeriği;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <endpoint name="AlgebraEndpoint"
contract="AlgebraClient.IAlgebraContract" binding="netTcpRelayBinding"/>
      <!-- Service Bus ile iletişim kurmak için TCP protokolü kullanılacaktır.
netTcpRelayBinding bunu belirtmektedir-->
    </client>
  </system.serviceModel>
</configuration>
```

önce sunucu uygulamamız sonrasında ise istemci uygulamamız çalıştırıldığında aşağıdakine benzer bir ekran görüntüsü ile karşılaşılması muhtemeldir.



Eğer bu şekilde bir sonuç aldıysak istemci uygulamamızın host edilen servis ile **Service Bus** üzerinden haberleştiğini düşünebiliriz. Aslında emin olmak için deneme amacıyla oluşturduğumuz **Service Namespace'** ini kaldırmamız yeterli olacaktır Lakin bu durumda aşağıdaki sonuç ile karşılaşırız**(EndpointNotFoundException)**



Böylece **Azure Service Platformu** üzerindeki ilk atılımımızı gerçekleştirmiş olduk arkadaşlar. Umarım birşeyler aktarabilmişimdir. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

AlgebraHost.rar (47,27 kb)

Screencast - Visual Studio 2010 IDE Geliştirmeleri (7 Eğlenceli Özellik) (2009-11-10T09:30:00)

visual studio 2010,screencast,

Merhaba Arkadaşlar,

Yazılımcıların gerçek hayattaki en önemli yardımcılarından ve vazgeçilmezlerinden biriside geliştirme ortamlarıdır(IDE). Microsoft, Visual Studio ürün ailesini her versiyonda geliştirerek yazılımcıların .Net Framework alt yapısını en etkili şekilde kullanabilmesini sağlamaya çalışmaktadır. Bu görsel dersimizde Visual Studio 2010 Beta 2 sürümü ile geliştiricinin hayatlarını kolaylaştıran yedi basit özellik incelenmektedir.

- Highlight Reference
- Navigate To
- View Call Hierarcy
- Asenkron Reference Dialog Penceresi
- Intellisense Geliştirmeleri
- Zoom
- Text Selection

Süre : 13:01

Yeni görsel derslerde görüşmek dileğiyle hepinize mutlu günler dilerim.

Ado.Net Data Services 1.5 CTP2 - Data Binding Bölüm 1 (2009-11-09T03:30:00)

ado.net data services,

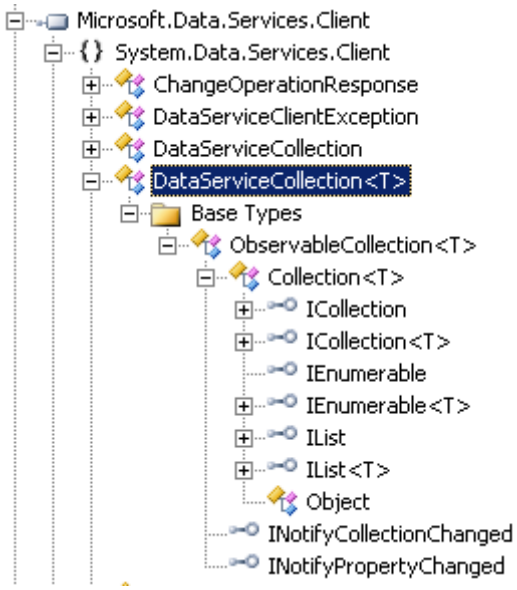
Merhaba Arkadaşlar,

Ado.Net Data Services ile geliştirilen servislerin tüketilmesi sırasında önem arz eden konulardan biriside, istemci tarafındaki **veri bağlama(DataBinding)** işlemleridir. öyleki, servisin tüketicisi olan istemcilerin

- Veriye bağlanılması,
- Bağlanılan verilerin ilgili kontrollerde gösterilmesi,
- Kontroller üzerinden yapılan değişikliklerin aslında bağlanılan Entity içeriklerinde de gerçekleştirilmesi,
- İstemci tarafındaki veri içeriğindeki değişimlerin servis tarafına da gönderilmesi(SaveChanges çağrısı sonrası)

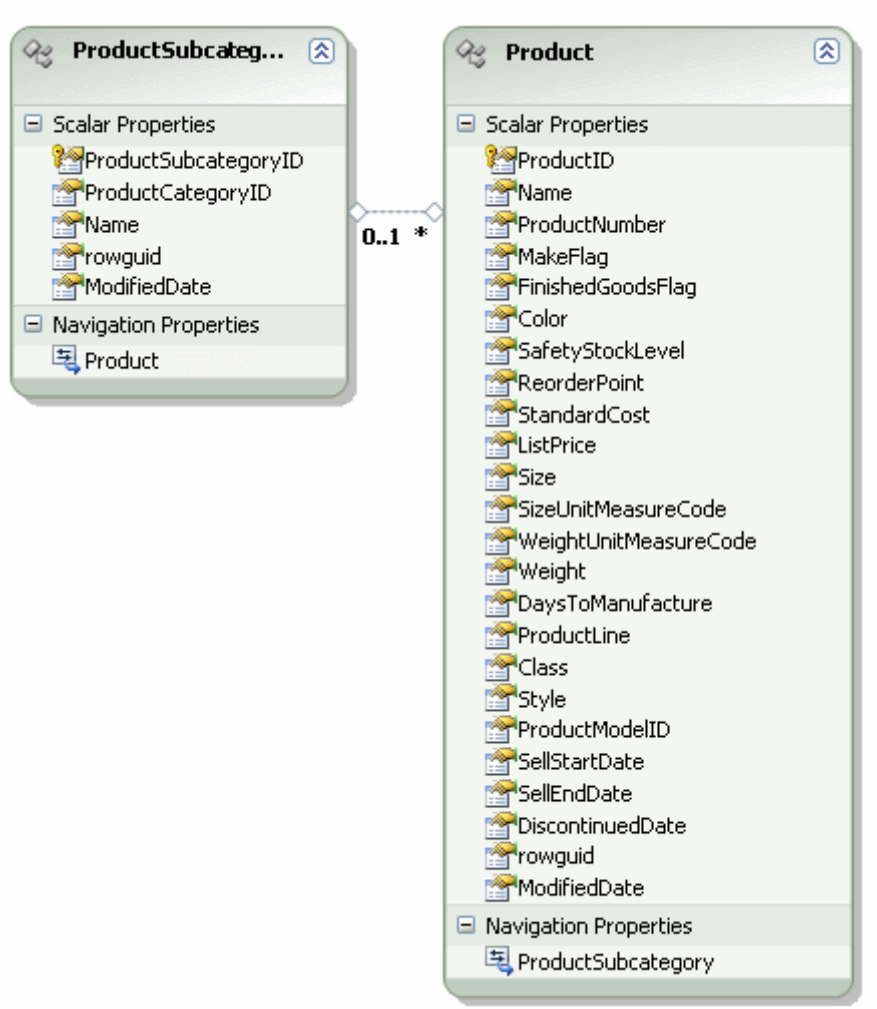
gibi fonksiyonellikleri desteklemesi gerekir. Ancak istmeci tarafında **WPF(Windows Presentation Foundation)** veya **Silverlight** gibi zengin içerik sağlayan uygulamalar söz konusu olduğunda, bu modellerin getirdiği veri bağlama kolaylıklarından da yararlanılmalıdır.

Ado.Net Data Services v1.5 ile birlikte istemci tarafına getirilen **DataServiceCollection** isimli koleksiyonun veri bağlama işlemlerinde kullanılabilmekte olup, **CTP2** versiyonunda dahada iyileştirilmiş olarak karşımıza çıkmaktadır. Buna göre istemci tarafı için üretilen kütüphanede**(Client Library)** kolaylaştırıcı değişiklikler yapıldığı söylenebilir. **DataServiceCollection<T>** koleksiyonu **ObservabelCollection<T>** tipinden türemekte olup, **INotifyPropertyChanged** ve **INotifyCollectionChanged arayüzlerini(Interface)** uygulamaktadır. Aşağıdaki **Object Browser** çıktısında bu tipin içeriği açık bir şekilde görülmektedir.



Bu nedenle **WPF** ve **Silverlight** tarafında ele alınanveri bağlama(DataBinding) ihtiyaçlarını hem **one-way** hemde **two-way** olarak karşılayabilecek bir koleksiyon olarak düşünülmelidir. Buna göre **WPF** ve **Silverlight** tarafındaki veri bağlı kontrollerin, **DataServiceCollection** sınıfından örneklenen koleksiyonları kullanabilmeleri mümkündür. Tahmin edileceği üzere iki yönlü bağlama sayesinde istemci **Context'** i ile **Entity'** ler arasındaki iletişimde, değişikliklerin karşılıklı olarak yansıtılabilmesi otomatikleştirilmektedir. Bir **DataServiceCollection** basit olarak, **Ado.Net Data Service** tarafına yapılacak bir çağrı ile kolayca doldurulabilir. özellikle **CTP2'** de istemci tarafında **DataServiceCollection** örneklerinin daha güçlü bir şekilde ele alınması için gerekli iyileştirmelerin yapıldığı görülmektedir.

Dilerseniz **Ado.Net Data Service** hizmetlerinin kullanıldığı senaryolarda, veri bağlama işlemlerinin nasıl yapılacağını örnekler üzerinden incelemeye başlayalım. İlk olarak **tek yönlü (One Way)** sonrasında ise **iki yönlü(Two Way)** veri bağlama işlemlerini ele alıyor olacağız. İşe ilk olarak basit bir **Asp.Net Web Application** projesi oluşturarak başlayabiliriz. Projemizde **Ado.Net Entity Framework** tabanlı bir veri kaynağı kullanıyor olacağız. İstemci tarafında **one-to-many** ilişki içerisinde değerlendirilebilecek tipleri ele almak istediğimizden kobay olarak **AdventureWorks** veritabanındaki **ProductSubcategory** ve **Product** tablolarını kullanıyor olacağız. **Ado.Net Entitiy Diagramımız** aşağıda görüldüğü gibidir.



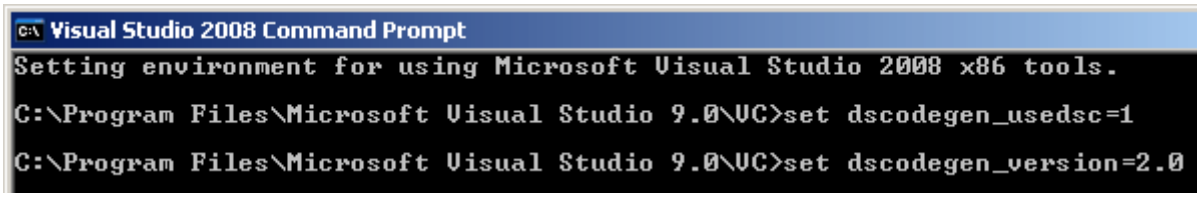
Ado.Net Data Service öğesini projeye ekledikten sonra, **ProductionService.svc**' ye ait kod içeriğini aşağıdaki gibi düzenleyebiliriz.

```
using System.Data.Services;

namespace AdventureServices
{
   public class ProductionService
      : DataService<AdventureWorksEntities>
   {
      public static void InitializeService(DataServiceConfiguration config)
      {
         config.SetEntitySetAccessRule("*", EntitySetRights.All);
         config.SetServiceOperationAccessRule("*", ServiceOperationRights.All);

         config.DataServiceBehavior.MaxProtocolVersion =
System.Data.Services.Common.DataServiceProtocolVersion.V2;
    }
```

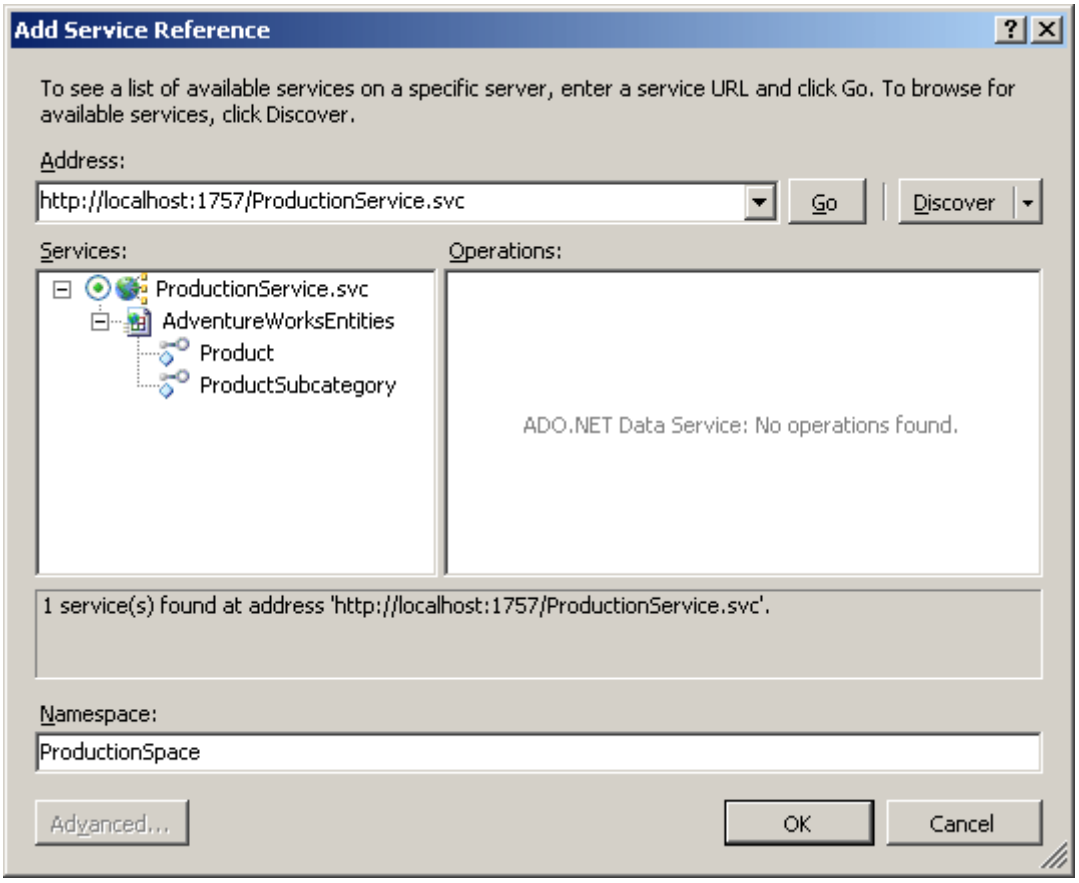
```
    }
}
```

önemli olan noktalarda birisi versiyon olarak **CTP2'** nin kullanılacağının **DataServiceProtocolVersion.V2** ile belirtilmesidir.

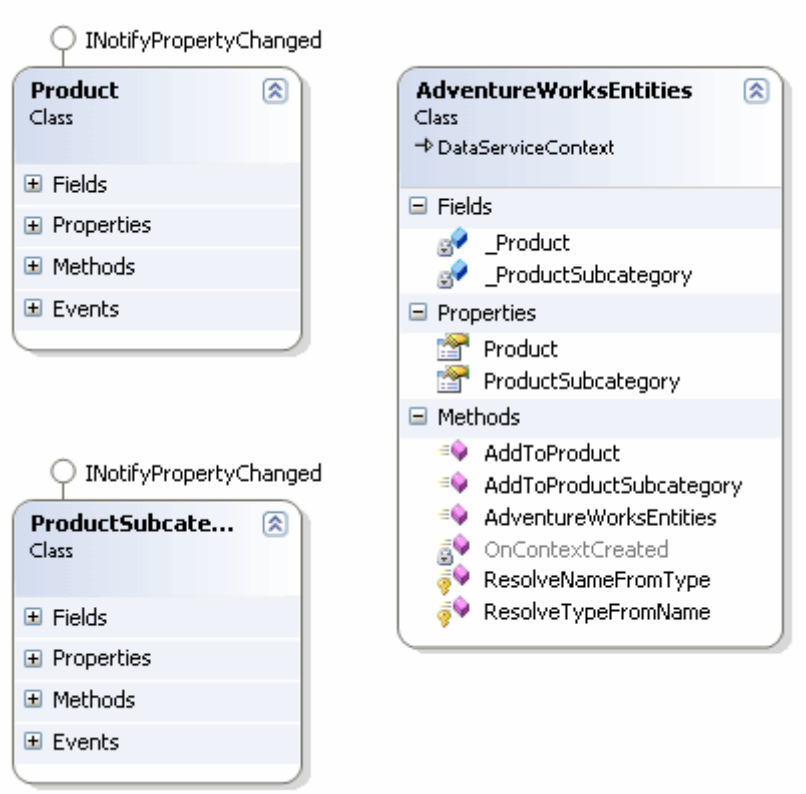
Artık istemci tarafını tasarlamaya başlayabiliriz. Ancak öncesinde istemci tarafının **Bağlayıcı Arayüzlerini(Binding Interfaces)** uygulayabilmesi için komut satırından***(Visual Studio 2008 Command Prompt)*** kod üretici ile ilgili bazı işlemlerin yapılması gerekmektedir. Visual Studio tarafından ele alanın ilgili **kod üretim(Code Generation)** kütüphanesine bu uygulama işini bildirmek için aşağıdaki komutlarım çalıştırılması yeterli olacaktır.



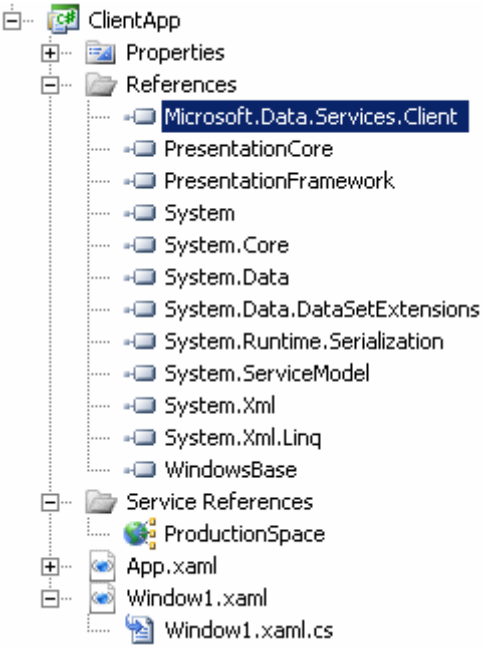
Bu kez gerçekten istemci tarafını yazmaya başlayabiliriz. Bu amaçla basit bir **WPF** uygulaması geliştireceğiz. *(Size tavsiyem aynı örneği Silverlight üzerinde geliştirmeye çalışmanız olacaktır.)* Uygulamamızı oluşturduktan sonra ilk yapacağımız işlem, **Ado.Net Data Service'** imize ait **URL** adresinden gerekli servis referansının, **Add Service Reference** yardımıyla projeye eklenmesi olacaktır.



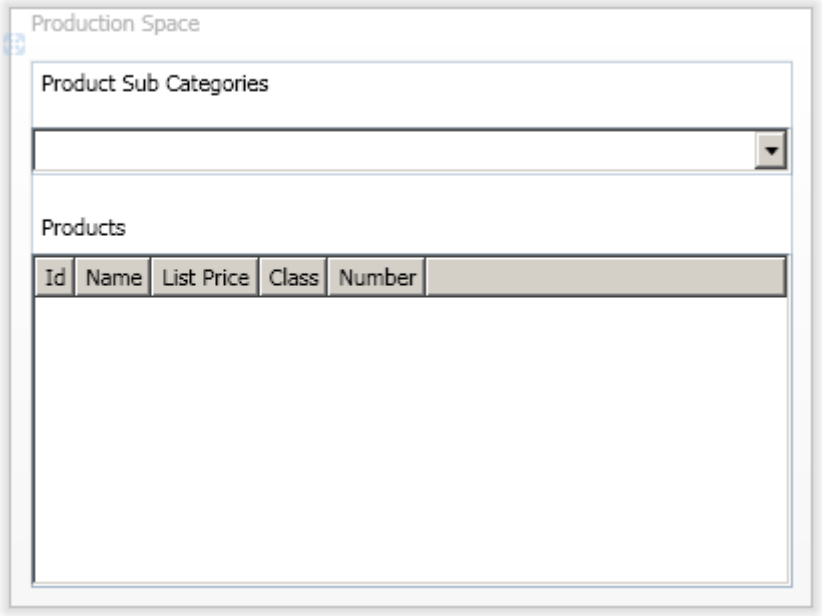
Görüldüğü üzere, servis üzerinden sunduğumuz **Product** ve **ProductSubcategory Entity** içerikleri buraya yansıtılmaktadır. Referansın eklenmesinden sonra istemci tarafında aşağıdaki sınıf diagramında görülen tiplerin oluşturulduğu fark edilebilir.



Dikkat edileceği üzere **Product** ve **ProductSubcategory** tiplerine **INotifyPropertyChanged** arayüzü uygulanmıştır. Buna göre söz konusu tiplere ait özelliklerde olacak değişimler, örneklerin bağlandığı ortamlara otomatik olarak bildirilecektir. Elbette tam tersi durumda geçerlidir. Yine servis referansının eklenmesi sonrası, istemci tarafına **Microsoft.Data.Services.Client assembly'** ının da bildirildiği görülebilir ki bu assembly **DataServiceCollection<T>** gibi önemli tipleri içermektedir.



WPF uygulamamızın **Window1** içeriğini görsel olarak aşağıdaki gibi tasarladığımızı düşünelim.



Görsel içeriğimizde yer alan **ComboBox** bileşenini **ProductSubcategory** içeriği ile dolduracağız. Diğer yandan alt tarafta
görülen **GridView** kontrolünde, **ComboBox** bileşeninden seçilen alt kategoriye bağlı ürünlerin*(dolayısıyla Product nesne örneklerinin)* bazı özellikleri gösterilecektir. çok doğal olarak **ComboBox** ve **GridView** bileşenlerinin, **Data Service** tarafından gelen içeriğe bağlanmaları gerekmektedir. üstelik **ProductSubcategory** ve **Product** entity' leri arasında bire çok ilişki söz konusu olduğundan, **ComboBox** kontrolünde bir öğeden diğerine geçildiğinde, buna bağlı ürünlerinde **GridView** kontrolünde gösterilmesi sağlanmalıdır. Bu durumlar göz önüne alındığında söz konusu **XAML** içeriğini aşağıdaki gibi tasarlamamız yeterli olacaktır.

```
<Window x:Class="ClientApp.Window1"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Title="Production Space" Height="300" Width="401">
   <Grid x:Name="grdProduction">
      <ComboBox ItemsSource="{Binding}" Height="23" Margin="0,33,0,0"
Name="cmbSubCategories"
VerticalAlignment="Top" IsSynchronizedWithCurrentItem="True">
         <ComboBox.ItemTemplate>
            <DataTemplate>
               <TextBlock Text="{Binding Path=Name}"/>
            </DataTemplate>
         </ComboBox.ItemTemplate>
      </ComboBox>
```

```
    <Label Height="28" HorizontalAlignment="Left" Margin="0,-1,0,0" Name="label1"
VerticalAlignment="Top" Width="136">Product Sub Categories</Label>
    <Label Height="28" HorizontalAlignment="Left" Margin="0,71,0,0" Name="label2"
VerticalAlignment="Top" Width="136">Products</Label>
    <ListView ItemsSource="{Binding Product}" Margin="0,96,0,0">
      <ListView.View>
        <GridView>
          <GridViewColumn Header="Id" DisplayMemberBinding="{Binding
Path=ProductID}"/>
          <GridViewColumn Header="Name" DisplayMemberBinding="{Binding
Path=Name}"/>
          <GridViewColumn Header="List Price" DisplayMemberBinding="{Binding
Path=ListPrice}"/>
          <GridViewColumn Header="Class" DisplayMemberBinding="{Binding
Path=Class}"/>
          <GridViewColumn Header="Number" DisplayMemberBinding="{Binding
Path=ProductNumber}"/>
        </GridView>
      </ListView.View>
    </ListView>
  </Grid>
</Window>
```

ComboBox kontrolünün **ItemsSource** özelliğinin **Binding** olarak bırakıldığına dikkat edelim. Buna göre **grdProduction** isimli Grid kontrolünün **DataContext** kaynağı ne ise,**ComboBox** kontrolü bu kaynağa otomatik olarak bağlanacak ve her bir öğesinde, **Name** alanının değerini***({Binding Path=Name} den dolayı)*** gösterecektir. Dolayısıyla kod tarafında**Grid** kontrolünün **DataContext** özelliğine atanan veri kümesi önem arz etmektedir. Diğer yandan **ComboBox** kontrolünün **IsSynchronizedWithCurrentItem** niteliğine **true** değeri atandığına da dikkat edilmelidir. Bu sayede **ComboBox** içerisinde olan değişiklikler, diğer veri bağlı kontrollerede iletilecektir. Yani **GridView** kontrolünün alt kategoriye bağlı ürünler ile doldurulması işleminin gerçekleştirilebilmesi için söz konusu niteliğin **true** değerine sahip olması gerekmektedir. **ListView** bileşeninin **ItemsSource** özelliğine **{Binding Product}** değeri atanmıştır. Buna göre, **ListView** içerisindeki veri bağlı kontrollerin **Product** kaynağına bağlanabileceği belirtilmiş olur ki bu sayede **GridView** kontrolünün **GridViewColumn** elementlerinin**DisplayMemberBinding** niteliklerine **Product** tipine ait özelliklerin adları atanmıştır. Peki tüm bu veri bağlı kontrollerin baz alacağı veri kaynağı nerede atanacaktır?

Bu amaçla kod tarafında aşağıdaki işlemleri yapmamız yeterli olacaktır.

Window1 Code içeriği;

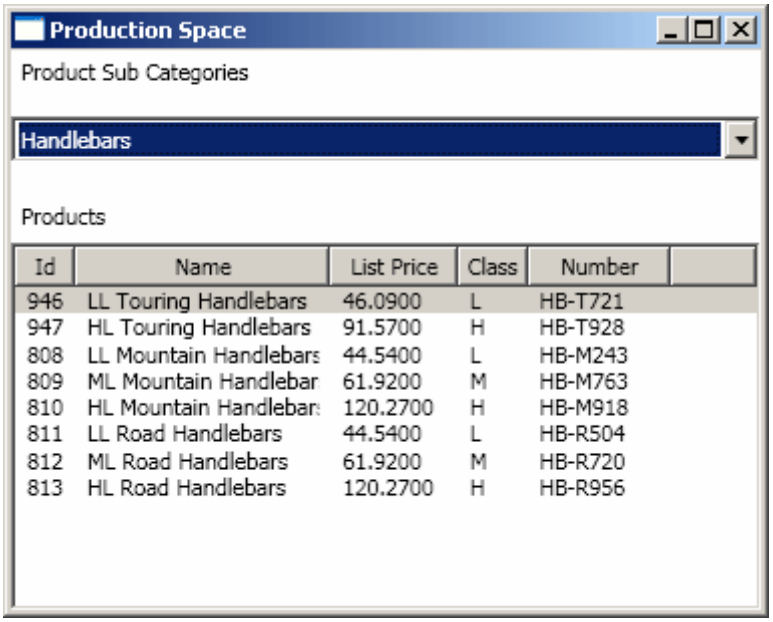
```
using System;
using System.Data.Services.Client;
using System.Windows;
using ClientApp.ProductionSpace;

namespace ClientApp
{
   public partial class Window1
      : Window
   {
      AdventureWorksEntities adw = new AdventureWorksEntities(new
Uri("http://localhost:1757/ProductionService.svc/"));

      public Window1()
      {
         InitializeComponent();

         grdProduction.DataContext = DataServiceCollection
            .CreateTracked<ProductSubcategory>(adw,
adw.ProductSubcategory.Expand("Product"));
      }
   }
}
```

DataServiceContext türevli nesne örneği oluşturulduktan sonra **Window1 yapıcı metodu(Constructor)** içerisinde **DataServiceCollection** üzerinden **CreateTracked** isimli bir çağrı yapıldığı görülmektedir. Bu çağrıda **ProductSubcategory** tipinden bir nesne kümesinin listesi alınmaktadır. Ayrıca metodun ikinci parametresine dikkat edilecek olursa, her bir **ProductSubcategory**için **Product** genişletmesinin yapıldığı, yani alt kategoriye bağlı olan ürünlerinde talep edildiği görülmektedir. Uygulamanın çalıştırılması sonrasında **CreateTracked**metodunun çağırıldığı yerde **Breakpoint** ile ilerlenir ve **SQL Server Profiler** aracından arka planda çalıştırılan sorgu incelenirse aşağıdaki ifadenin yürütüldüğü görülebilir.

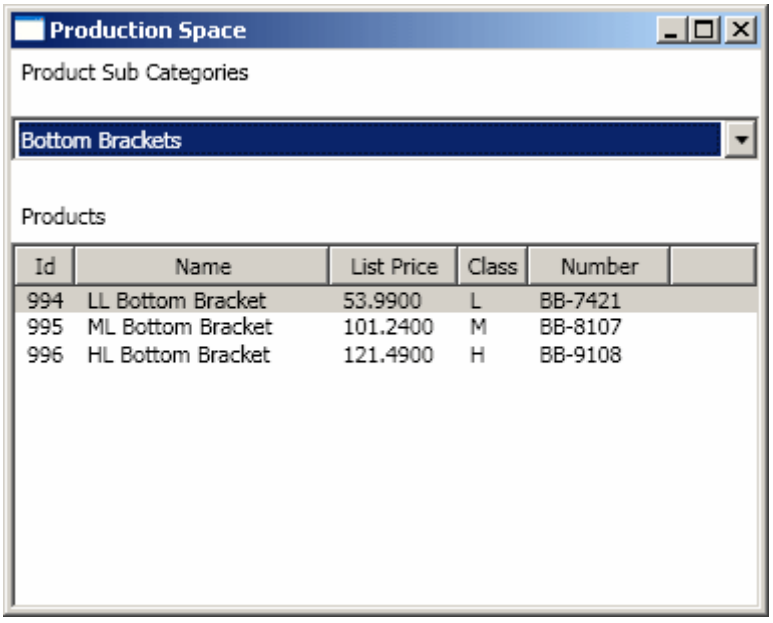
```
SELECT
[Project1].[ProductSubcategoryID] AS [ProductSubcategoryID],
[Project1].[ProductCategoryID] AS [ProductCategoryID], [Project1].[Name] AS [Name],
[Project1].[rowguid] AS [rowguid], [Project1].[ModifiedDate] AS [ModifiedDate],
[Project1].[C1] AS [C1], [Project1].[C2] AS [C2], [Project1].[ProductID] AS [ProductID],
[Project1].[Name1] AS [Name1], [Project1].[ProductNumber] AS [ProductNumber],
[Project1].[MakeFlag] AS [MakeFlag], [Project1].[FinishedGoodsFlag] AS
[FinishedGoodsFlag], [Project1].[Color] AS [Color], [Project1].[SafetyStockLevel] AS
[SafetyStockLevel], [Project1].[ReorderPoint] AS [ReorderPoint],
[Project1].[StandardCost] AS [StandardCost], [Project1].[ListPrice] AS [ListPrice],
[Project1].[Size] AS [Size], [Project1].[SizeUnitMeasureCode] AS
```

```
[SizeUnitMeasureCode], [Project1].[WeightUnitMeasureCode] AS
[WeightUnitMeasureCode], [Project1].[Weight] AS [Weight],
[Project1].[DaysToManufacture] AS [DaysToManufacture], [Project1].[ProductLine] AS
[ProductLine], [Project1].[Class] AS [Class], [Project1].[Style] AS [Style],
[Project1].[ProductModelID] AS [ProductModelID], [Project1].[SellStartDate] AS
[SellStartDate], [Project1].[SellEndDate] AS [SellEndDate], [Project1].[DiscontinuedDate]
AS [DiscontinuedDate], [Project1].[rowguid1] AS [rowguid1], [Project1].[ModifiedDate1]
AS [ModifiedDate1]
FROM ( SELECT
 [Extent1].[ProductSubcategoryID] AS
[ProductSubcategoryID],  [Extent1].[ProductCategoryID] AS
[ProductCategoryID],  [Extent1].[Name] AS [Name],  [Extent1].[rowguid] AS
[rowguid],  [Extent1].[ModifiedDate] AS [ModifiedDate],  1 AS
[C1],  [Extent2].[ProductID] AS [ProductID],  [Extent2].[Name] AS
[Name1],  [Extent2].[ProductNumber] AS [ProductNumber],  [Extent2].[MakeFlag] AS
[MakeFlag],  [Extent2].[FinishedGoodsFlag] AS [FinishedGoodsFlag],  [Extent2].[Color]
AS [Color],  [Extent2].[SafetyStockLevel] AS
[SafetyStockLevel],  [Extent2].[ReorderPoint] AS
[ReorderPoint],  [Extent2].[StandardCost] AS [StandardCost],  [Extent2].[ListPrice] AS
[ListPrice],  [Extent2].[Size] AS [Size],  [Extent2].[SizeUnitMeasureCode] AS
[SizeUnitMeasureCode],  [Extent2].[WeightUnitMeasureCode] AS
[WeightUnitMeasureCode],  [Extent2].[Weight] AS
[Weight],  [Extent2].[DaysToManufacture] AS
[DaysToManufacture],  [Extent2].[ProductLine] AS [ProductLine],  [Extent2].[Class] AS
[Class],  [Extent2].[Style] AS [Style],  [Extent2].[ProductModelID] AS
[ProductModelID],  [Extent2].[SellStartDate] AS [SellStartDate],  [Extent2].[SellEndDate]
AS [SellEndDate],  [Extent2].[DiscontinuedDate] AS
[DiscontinuedDate],  [Extent2].[rowguid] AS [rowguid1],  [Extent2].[ModifiedDate] AS
[ModifiedDate1],
 CASE WHEN ([Extent2].[ProductID] IS NULL) THEN CAST(NULL AS int) ELSE 1
END AS [C2]
 FROM  [Production].[ProductSubcategory] AS [Extent1]
 LEFT OUTER JOIN [Production].[Product] AS [Extent2] ON
[Extent1].[ProductSubcategoryID] = [Extent2].[ProductSubcategoryID]
)  AS [Project1]
ORDER BY [Project1].[ProductSubcategoryID] ASC, [Project1].[C2] ASC
```

çok fazla alan var değil mi? 🤒 Her neyse...Gelelim çalışma zamanındaki duruma. örneğin **Handlebase** alt kategorisini seçtiğimizde aşağıdaki şekilde görülen durum oluşmaktadır.



Başka bir alt kategori seçtiğimizde ise*(örneğin Bottom Brackets)* buna bağlı ürünlerin **GridView** kontrolüne doldurulduğu görülecektir.



Tabiki burada sadece entity içeriklerinin doldurulması ve veri kontrollerine **tek yönlü*(One-Way)*** bağlanması söz konusudur. Ancak tahmin edileceği üzere birde kontroller üzerinden verilerde yapılan değişiklikler sonrası bunların **Entity** içeriklerine yansıtılması ve sonrasında **SaveChanges** metodu ile tüm değişikliklerin servis tarafına gönderilmesi söz konusu olabilir ki buda iki yönlü(Two Way) bağlamanın tesis edilmesi ile kolayca gerçekleştirilebilir. Nitekim **two-way** binding metoduna göre, kolekisyonda olacak değişimler, **SaveChanges** metoduna yapılan çağrı sonucu servis tarafına ve dolayısıyla sunucu üzerindeki veri kaynağına da iletilecekteir. Bu konuyu bir sonraki yazımızda ele alıyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

DataBinding.rar (116,66 kb)

Screencast - Visual Studio 2010 ile Test Driven Development (2009-11-08T01:00:00)

test driven development,visual studio,screencast,

Merhaba Arkadaşlar,

özellikle **çevik(Agile)** süreçlerde kullanılan önemli tekniklerden biriside **Test Güdümlü Geliştirme(Test Driven Development)** dir. Bu tekniğin ana fikri ise **Red-Green-Refactor** kelimelerinden oluşmaktadır. Herşey bir test geliştirilerek başlar. Testin öncelikle **başarılı olmaması(Fail)** hedeflenir. Bu **Red** olarak isimlendirilen adımdır. Sonrasında **testten geçebilmek(Pass)** için gerekli adımlar miminum kod eforu sarfedilerek gerçekleştirilir ki buda **Green** isimli adım olarak adlandırılmaktadır. **Green** adımı ile test başarılı bir şekilde geçildikten sonra **kodun yeniden düzenlenmesi(Refactor)** işlemleri yapılır. Tüm bu işlemlerin sonucunda testlerden başarı ile geçmiş ve amaca yönelik tüm kodlamaları içeren bir ürün oluşmaktadır. İşin ilginç olan noktalarından birisi de, geliştirmeye konu olan **tiplerin(Types)** ve **üyelerinin(Members->Fields,Properties,Methods...)** testi yazarken ortaya çıkmasıdır. Peki en basit haliyle **Test Driven Development** nasıl yapılabilir. Merak ediyorsanız tıklayın

Süre : 22dk 27sn

Yeni görsel derslerde görüşmek dileğiyle hepinize mutlu günler dilerim.

Screencast - Ado.Net Data Services 1.5 - Paging (2009-11-06T09:00:00)

ado.net data services,screencast,

Merhaba Arkadaşlar,

Ado.Net Data Services 1.5 CTP2 ile birlikte gelen yeniliklerden biriside sunucu tarafındaki verilerin **sayfalanarak(Paging)** gönderilebilmesidir. Asp.Net Web uygulamalarında sıklıkla kullandığımız sayfalama tekniğinin bir benzeri olarak düşünüldüğünde, istemci ve sunucu tarafında belirgin performans kazanımlarına neden olan bir özelliktir. Nitekim büyük çaplı verilerin bir bütün halinde ve hemen her istemci

talebi sonrasında ilgili veri kaynağından*(Entity Framework ve Custom LINQ Provider üzerinden)* çekilmesi hem sunucu tarafında fazladan iş yüküne neden olmakta hemde istemci tarafına çok büyük boyutta veri akmasına neden olmaktadır. Kullanımı son derece kolay olan bu özelliği incelediğimiz görsel dersimizde **SQL Profiler** aracından da yararlanarak arka planda çalıştırılan sorguları analiz etme şansına da sahip olacağız. İyi seyirler

Yeni görsel derslerde görüşmek dileğiyle hepinize mutlu günler dilerim.

Screencast - Visual Studio 2010 Debug Genişletmeleri - 1 (2009-11-03T09:00:00)

visual studio 2010,screencast,

Merhaba Arkadaşlar,

Visual Studio geliştirme ortamının yazılımcılar için çok önemli bir araç olduğu ortadadır. İlk versiyonundan**(Visual Studio.NET 2002 - Rainer)** itibaren her yeni versiyonda, geliştiricileri heyecanlandıran ve hayatlarını inanılmaz ölçüde kolaylaştıran özellikler barındırarak gelmektedir. Şöyle bir geçmiş zamana bakıyorumda ilk **Visual Studio.Net** bizleri oldukça heyecanlandırmıştı. Sonrasında **Whidbey** kod adlı **Visual Studio 2005'** in büyüsüne kapıldık ve derken **Visual Studio 2008(Orcas)** karşımıza geldi. özellikle **Service Pack**yüklemeleri sonrasında **IDE'** ye kazandırılan pek çok yenilik ile geliştiricilerin hemen her ihtiyacını karşılayan bir ortam ile karşı karşıya kaldık zaman içerisinde. Ama gerçekten öyle miydi acaba?

Artık **.Net Framework** üzerinde yapılan geliştirmelerde **IDE'** yi zorlayıp dahada iyileştirilmesine neden oluyor. örneğin **WPF** yetenekleri ile donatılmış bir tasarım ortamımız var. örneğin paralel programlama ile ilişkili debug yeteneklerimiz ve uygulamaların çeşitli tipte performans ölçümlerini yapmamızı sağlayan IDE içi araçlarımız var.

Elbette dış faktörleride unutmamak gerekiz. Yazılım mimarilerinin tasarımları sırasında baş rol oynayan pek çok kavram**(UML diagramları** gibi) ve **TFS'** in ihtiyaçları sonucu, gerekli tüm grafiksel çizimlere aynı IDE içerisinden erişebilmek ve etkin bir şekilde kullanabilmek artık zaruri bir gereksinim haline gelmiştir. Daha önceden elimizde yanlız **Class Diagram** ve **System Diagram** gibi basit ve ilkel grafik araçları var iken

artık **UML** diagramlarının bile IDE içerisine gömülüyor olduğunu gördük. Hatta Layer ve Graph diagramlarımız bile var artık.

Ancak biz bu yeni görsel dersimizde basit bebek adımları ile işe başlıyor olacağız ve ilk etapta **Visual Studio 2010 Beta 2(Hawaii)** üzerinde Debug genişletmelerine değinmeye çalışacağız.

önce **Breakpoint'** ler ile ilişkili genişletmeleri inceleyeceğiz. Söz gelimi tüm **Breakpoint'** lerin **Export** ve **Import** edilmesi işlemlerini göreceğiz veya Breakpoint' ler içerisinde arama seçeneklerine bakacağız. Sonrasında **Data Tips** özelliğine değineceğiz. Devam eden kısımda ise çalışmakta olan **Thread'** leri nasıl izleyebileceğimizi ve çalışma zamanının metinsel içeriğini nasıl **Dump** edebileceğimize şahit olacağız. üstelik tüm **Export, Import** ve **Dump** gibi işlemlerde **XML** içerikli kaynaklardan yararlanıldığını anlayacağız. E öyleyse ne duruyorsunuz, haydi tıklayın... 😀

Yeni görsel derslerde görüşmek dileğiyle hepinize mutlu günler dilerim.

WF 4.0 : WorkflowInvoker ile Single Thread, WorkflowApplication ile Multi-Thread [Beta 2] (2009-11-02T11:35:00)

wf,wf 4.0,

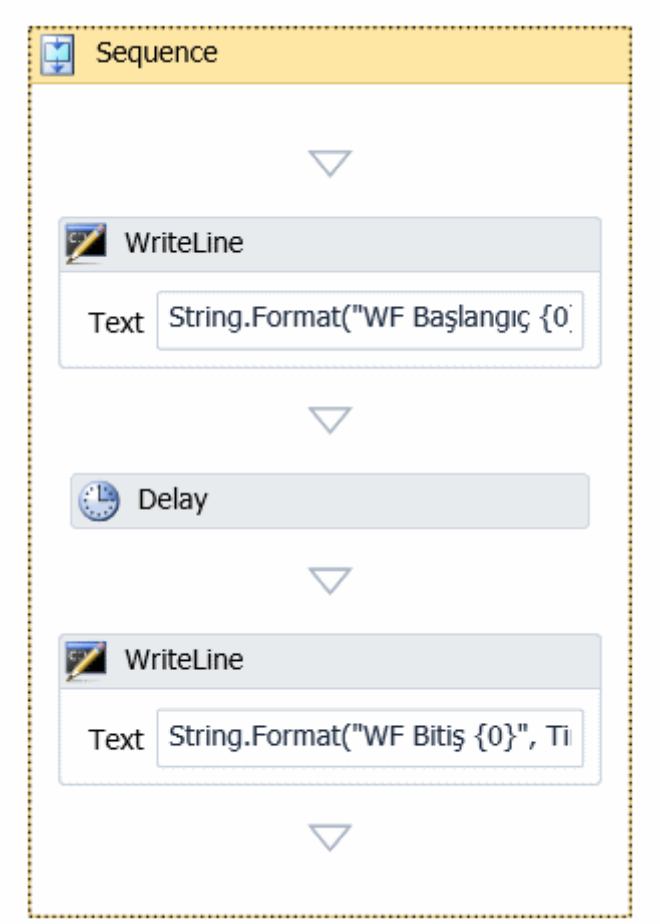
Merhaba Arkadaşlar,

Hiç müzik dinlerken bir yandan da kod yazmayı denediniz mi? üstelik çevre ile olan etkileşiminiz devam ederken Söz gelimi hareketli bir parçayı tempo tutarak dinleyip ondan tamamen bağımsız bir şekilde geliştirmeye devam ederken yan masadaki arkadaşınızdan gelen "Dün akşamki maçı seyrettin mi?...Ronaldo ne gol attı öyle..." sorusuna da rakip takımın orta sahasını kattığınız bir yorumda bulunup diğer taraftanda kahve içtiğinizi düşünebilirsiniz. Tabiki insan beyninin büyülü dünyası ve eş zamanlı olarak çalışma yetenekleri zaman zaman geliştirdiğimiz uygulamalara da yansımaktadır. Böyle bir giriş yaptığımıza göre **multi-thread** bir takım işlemleri anlatacağımı düşünmüş olmalısınız. İşte bu gün konumuz **Workflow Foundation 4.0** üzerindeki **Single-Thread** ve **Multi-Thread** çalıştırma modelleri.

WF 4.0 öncesinde bir **Workflow** örneğini çalıştırmak için **WorkflowRuntime** sınıfından yararlanılmaktadır. Aşağıdaki kod parçasında **Visual Studio 2008** üzerinde geliştirilen basit bir **WF** örneğinin çalıştırılması için otomatik olarak üretilen kod görülmektedir.

```
using(WorkflowRuntime workflowRuntime = new WorkflowRuntime())
        {
            AutoResetEvent waitHandle = new AutoResetEvent(false);

            workflowRuntime.WorkflowCompleted += delegate(object sender,
WorkflowCompletedEventArgs e) {waitHandle.Set();};

            workflowRuntime.WorkflowTerminated += delegate(object sender,
WorkflowTerminatedEventArgs e)
            {
                Console.WriteLine(e.Exception.Message);
                waitHandle.Set();
            };

            WorkflowInstance instance =
workflowRuntime.CreateWorkflow(typeof(WorkflowConsoleApplication1.Workflow
1));
            instance.Start();

            waitHandle.WaitOne();
        }
```

Ancak **Workflow Foundation 4.0** içerisinde bir **Workflow** örneğini çalıştırmak için iki farklı yol sunulmaktadır. İlk yol daha önceki yazı ve görsel derslerimizde de sıklıkla bahsettiğimiz **WorkflowInvoker** sınıfına ait **static Invoke** metodunun kullanılmasıdır. Bu tekniğin en önemli özelliği **Workflow** örneğinin çalıştığı uygulamaya ait **Thread**içerisinde senkron olaran yürütülmesini sağlamasıdır. Dilerseniz ne demek istediğimize basit bir örnek yardımıyla bakmaya çalışalım. **Visual Studio 2010 Ultimate Beta 2**sürümü üzerinden oluşturduğumuz **Workflow Console Application** içerisinde aşağıdaki **Workflow1** içeriği göz önüne alınmaktadır.



Akışın xaml içeriği ise aşağıdaki gibidir.

<Activity mc:Ignorable="sap" x:Class="WorkflowConsoleApplication5.Workflow1" mva:VisualBasic.Settings="Assembly references and imported namespaces serialized as XML namespaces" xmlns="http://schemas.microsoft.com/netfx/2009/xaml/activities" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" xmlns:mv="clr-namespace:Microsoft.VisualBasic;assembly=Microsoft.VisualBasic, Version=10.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" xmlns:mv1="clr-namespace:Microsoft.VisualBasic;assembly=System" xmlns:mva="clr-namespace:Microsoft.VisualBasic.Activities;assembly=System.Activities" xmlns:s="clr-namespace:System;assembly=mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" xmlns:s1="clr-namespace:System;assembly=mscorlib" xmlns:s2="clr-namespace:System;assembly=System" xmlns:s3="clr-namespace:System;assembly=System.Xml" xmlns:s4="clr-namespace:System;assembly=System.Core" xmlns:sad="clr-namespace:System.Activities.Debugger;assembly=System.Activities" xmlns:sap="http://schemas.microsoft.com/netfx/2009/xaml/activities/presentation" xmlns:scg="clr-namespace:System.Collections.Generic;assembly=System" xmlns:scg1="clr-namespace:System.Collections.Generic;assembly=System.ServiceModel" xmlns:scg2="clr-namespace:System.Collections.Generic;assembly=System.Core"

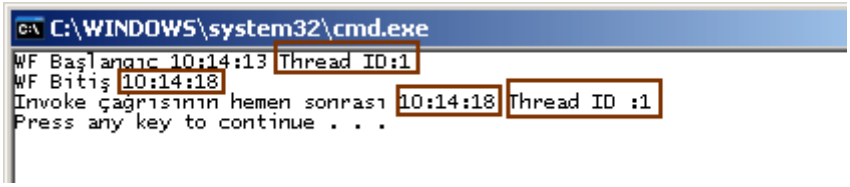
```
xmlns:scg3="clr-namespace:System.Collections.Generic;assembly=mscorlib"
xmlns:sd="clr-namespace:System.Data;assembly=System.Data" xmlns:sd1="clr-
namespace:System.Data;assembly=System.Data.DataSetExtensions" xmlns:sl="clr-
namespace:System.Linq;assembly=System.Core" xmlns:st="clr-
namespace:System.Threading;assembly=mscorlib, Version=4.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" xmlns:st1="clr-
namespace:System.Text;assembly=mscorlib"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <Sequence sad:XamlDebuggerXmlReader.FileName="c:\documents and
settings\bsenyurt\my documents\visual studio
10\Projects\WorkflowConsoleApplication5\WorkflowConsoleApplication5\Workflow1.xa
ml" sap:VirtualizedContainerService.HintSize="232,344">
    <sap:WorkflowViewStateService.ViewState>
      <scg3:Dictionary x:TypeArguments="x:String, x:Object">
        <x:Boolean x:Key="IsExpanded">True</x:Boolean>
      </scg3:Dictionary>
    </sap:WorkflowViewStateService.ViewState>
    <WriteLine sap:VirtualizedContainerService.HintSize="210,59" Text="[String.Forma
t(&amp;quot;WF Başlangıç {0} Thread ID:{1}&amp;quot;, TimeString,
Thread.CurrentThread.ManagedThreadId.ToString())]" />
    <Delay Duration="00:00:05" sap:VirtualizedContainerService.HintSize="210,22" />
    <WriteLine sap:VirtualizedContainerService.HintSize="210,59" Text="[String.Forma
t(&amp;quot;WF Bitiş {0}&amp;quot;, TimeString)]" />
  </Sequence>
</Activity>
```

Workflow1 içerisinde dikkat çekici noktalardan birisi **Delay** aktivitesi ile sanal bir geciktirmenin uygulanmış olmasıdır. Bunun yanında ilk **WriteLine** bileşeni içerisinde güncel **ManagedThreadId**değerinin yazdırılması sağlanmaktadır. Şimdi bu **Workflow** örneğini yürütmek için aşağıdaki kod parçasını geliştirdiğimizi düşünelim.

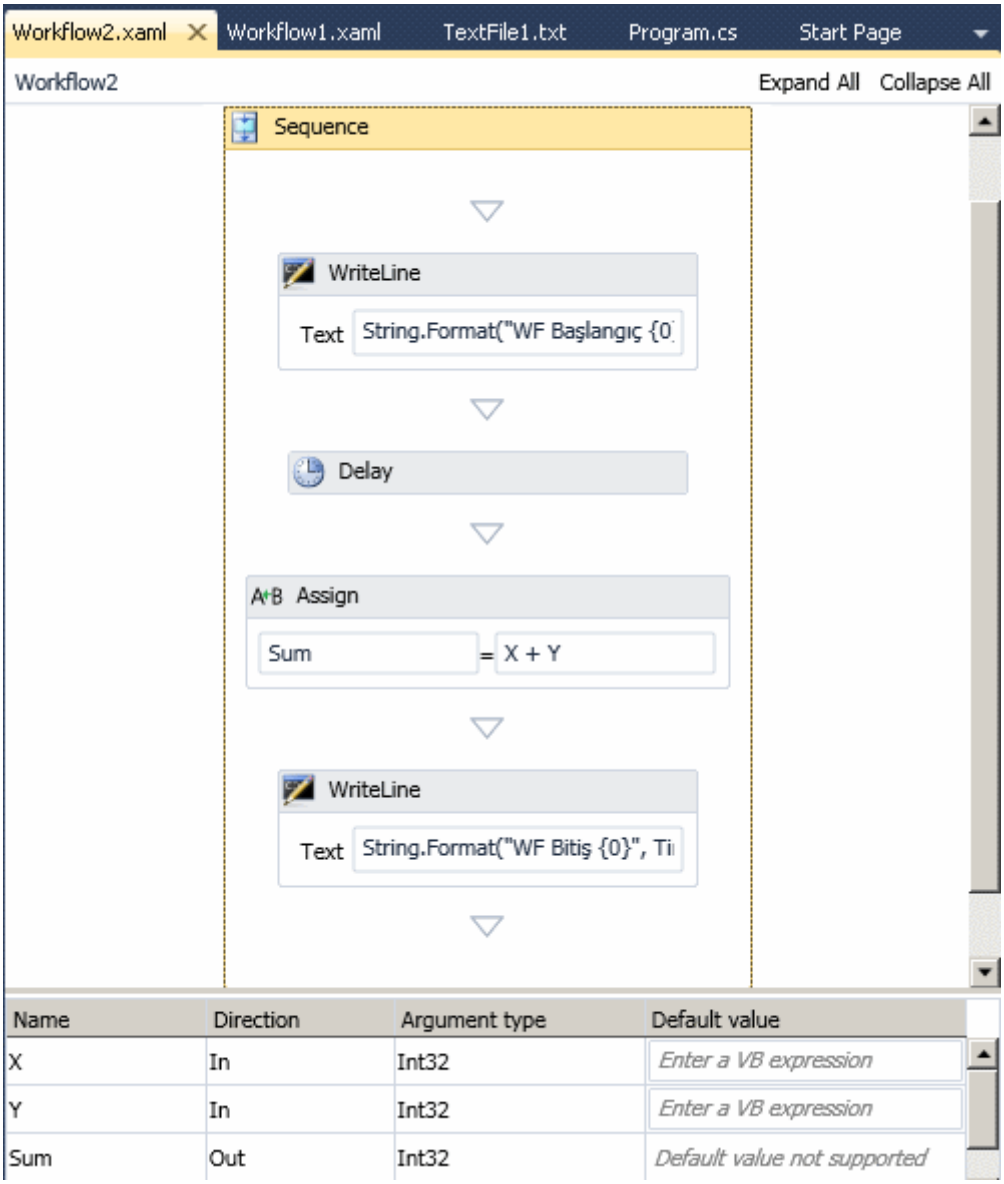
```
using System;
using System.Activities;
using System.Threading;

namespace WorkflowConsoleApplication5
{

    class Program
    {
        static void Main(string[] args)
        {
            #region Senkron çalışma(Aynı Thread içerisinde)
```

```
            WorkflowInvoker.Invoke(new Workflow1());
            Console.WriteLine("Invoke çağrısının hemen sonrası {0} Thread ID
:{1}",DateTime.Now.ToLongTimeString(),Thread.CurrentThread.ManagedThreadId.
ToString());

            #endregion
        }
    }
}
```

İlk olarak **WorkflowInvoker** sınıfının **static Invoke** metodu ile **Workflow1** nesne örneğinin çalıştırılması sağlanmaktadır. Devam eden kod satırının devreye girmesi için**Delay** süresi kadar beklenmesi gerekecektir ki bu örnekte önemli olan noktada budur. Diğer taraftan hem **Workflow1** için hemde **Program** sınıfı için aynı **ManagedThreadId**değerlerinin üretildiği gözlemlenecektir. İşte çalışma zamanı sonuçlarımız.



Görüldüğü gibi **Invoke** metodu çağrısı sonrasındaki kod satırına geçmek için **Workflow1'** in tamamlanması beklenmiştir. Bu senkron çalışmanın bir sonucu ve ispatıdır. Ayrıca bir **Workflow** örneğini çalıştırmanın en basit yolu olarak düşünülebilir. Ancak bazı zamanlarda özellikle **Long Running Process'** lerde söz konusu olduğunda, sunucu tarafındaki Workflow örneğinin farklı bir Thread içerisinde yürütülmesi ve buna bağlı olarakta Host uygulamanın paralel olarak çalışmaya devam etmesi istenebilir. Bir başka deyişle Workflow örneklerini host eden uygulamanın söz konusu akışları **Multi-Thread** olarak yürütmesi istenebilir. İşte bu durumda **WorkflowApplication** tipinden yararlanılmaktadır. Bu tip ve üyeleri sayesinde **Workflow** örneğinin farklı bir **Thread** içerisine açılması ve ana uygulama ile paralel olarak yürütülmesi sağlanabilir. üstelik çeşitli olayları sayesinde **Workflow**örneğinin tamamlanması, ele alınmamış bir istisna**(Unhandled Exception)** nedeniyle sonlanması gibi durumlar kontrol altına alınabilir. Aslında bir önceki versiyonda yer alan **WorkflowRuntime**tipinin üstlendiği misyonun aynısıdır diyebiliriz. Şimdi bu çalışmayı farklı bir **Workflow** örneği üzerinden değerlendirmeye çalışalım. Bu sefer parametre kullanımı ve geri dönüş tiplerinide söz konusu sınıf olayları içerisinde değerlendirmeye çalışacağız. İşte **Workflow2** tasarım görüntüsü.



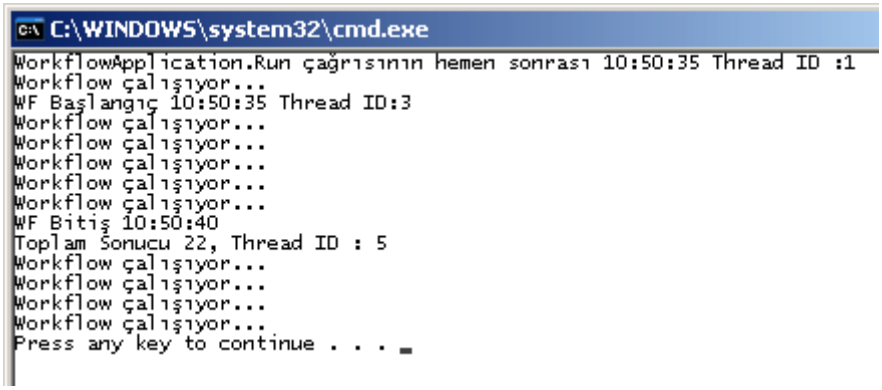
Workflow1'deki aktivitemizde olan bileşenleri kullanmakla beraber bu kez **X** ve **Y** isimli iki **In** ve **Sum** isimli bir **Out** argümanımız olduğunu belirtelim. **Main** metodu içeriğimizi ise aşağıdaki gibi geliştirdiğimizi düşünelim.

```
using System;
using System.Activities;
using System.Threading;

namespace WorkflowConsoleApplication5
{

    class Program
    {
        static void Main(string[] args)
        {
            #region Asenkron çalışma
```

```
AutoResetEvent aResetEvent = new AutoResetEvent(false);

// Workflow2 nesnesi örneklenir ve In argümanlarına ilk değerleri atanır
Workflow2 wf2 = new Workflow2
{
   X = 10,
   Y = 12
};

// WorkflowApplcation nesnesi örneklenirken parametre olarak çalıştırılmak istenen Workflow nesne örneği verilir.
WorkflowApplication wfApp = new WorkflowApplication(wf2);

// Completed olay metodu, WorkflowApplicationCompletedEventArgs tipinden olay parametresi alır.
// Workflow çalışmasını tamamladığında devreye girecek olay metodudur
wfApp.Completed = (e) =>
   {
      // e parametresinden yararlanarak Outputs özelliğine ve dolayısıyla çalıştırılmakta olan Workflow' un Out argümanlarına ilgili Dictionary koleksiyonu üzerinden erişilebilir.
      Console.WriteLine("Toplam Sonucu {0}, Thread ID : {1}", e.Outputs["Sum"].ToString(), Thread.CurrentThread.ManagedThreadId.ToString());
      aResetEvent.Set(); // Workflow' un tamamlandığı bilgisi Thread' e sinyal olarak gönderilir.
   };

// Aborted olayı WorkflowApplicationAbortedEventArgs tipinden parametre almaktadır.
// Workflow bir sebepten iptal olduğunda devreye giren olay metodudur.(örneğin, Abort metodu ile yapılan çağrı veya bir Exception nedeniyle)
wfApp.Aborted = (e) =>
   {
      // Akışın bir istisna(Exception) nedeniyle iptal olması halinde Exception tipinden olan Reason özelliği değerlendirilir.
      if(e.Reason!=null)
         Console.WriteLine("Workflow2 {0} sebebiyle iptal oldu", e.Reason.Message);
      aResetEvent.Set(); // Workflow' un tamamlandığı bilgisi Thread' e sinyal olarak gönderilir.
   };

wfApp.Run(); // İşlemler başlatılır
```

```
        Console.WriteLine("WorkflowApplication.Run çağrısının hemen sonrası {0}
Thread ID :{1}", DateTime.Now.ToLongTimeString(),
Thread.CurrentThread.ManagedThreadId.ToString());

        // Ana Thread içerisindeki işlemlerin paralel yürüdüğünü göstermek için kullanılan
hile döngüsü
        for (int i = 0; i < 10; i++)
        {
           Console.WriteLine("Workflow çalışıyor...");
           Thread.Sleep(1000);
        }

        aResetEvent.WaitOne(); // Eğer Workflow halen daha tamamlanmamışsa bekle.

        #endregion
      }
   }
}
```

örneği bu haliyle çalıştırdığımızda aşağıdaki ekran görüntüsüne benzer çalışma zamanı sonuçlarını alırız.



Görüldüğü üzere **Workflow2** örneği çalıştırıldıktan sonra ana uygulama akmaya devam etmiştir. üstelik ana uygulama akmaya devam ederken tamamlanan **Workflow** ile ilişkili**Completed** olay metodu da devreye girmiştir. Dikkat edilmesi gereken noktalardan biriside ana uygulamaya ait **ManagedThreadId** ile **WorkflowApplication** tarafından açılan **ManagedThreadId**değerlerinin farklı olmasıdır ki istediğimiz sonuçlardan bir diğeride budur. Nitekim **Multi-Thread** çalışmanın ispatıdır.
Aslında **WF4.0** öncesindeki **WorkflowRuntime** kullanımına baktığımızda neredeyse aynı işleyiş modelinin kullanıldığını görebiliriz. önceki modelde de olay bazlı olarak **Workflow** örneklerinin tamamlanması ve sonlanması durumları ele alınabilmektedir. Ancak WorkflowApplication tipinin bu şekilde kalıp kalmayacağıda henüz belli değildir. Bunu ancak son sürümde net olarak öğrenebileceğimizi ifade edebilirim. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Kişisel Not : *Tabiki halen Beta 2 sürümünde olduğumuzu hatırlatmak isterim. Yani bu özelliklerde değişiklikler olabilir, bazıları kaldırılabilir veya tekrardan geriye dönüşler yapılabilir.*

WorkflowConsoleApplication5.rar (43,99 kb)

Ado.Net Data Services 1.5 CTP2 - Web Friendly Feeds (2009-10-31T20:25:00)

ado.net data services,

 Merhaba Arkadaşlar,

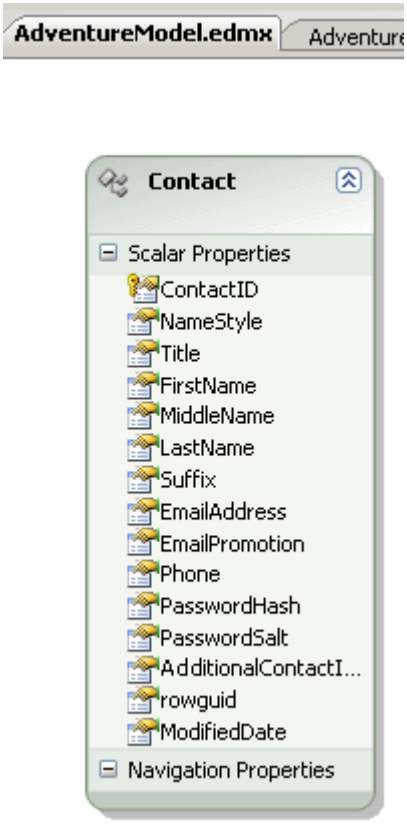
Ado.Net Data Services v1.5 CTP1 ile gelen **Web Friendly Feeds** özelliği, **CTP2** sürümünde eklenen iki yeni eşleştirme seçeneği ile genişletilmiştir. Durun bir dakika...**Web Friendly Feeds** nedir? Arkadaşlıktan farklı bir şey olsa gerek öncelikle bu konuya açıklık getirmek gerekiyor.

Web Friendly Feeds özelliği, bir **Entity**'nin herhangibir **özelliğini(Property)**, **Ado.Net Data Service'** inden çıktı olarak üretilen **Atom** içeriğindeki bir **elemente** eşleştirmekte kullanılmaktadır. Nitekim servisin ürettiği varsayılan **Atom** içeriğinde yer alan **author name, url, title** vs... gibi bilgiler zaten standart olarak kabul edilmiştir ve bu nedenle söz konusu elementleri değerlendiren yorumlayıcılara, var olan Entity içeriğindeki bazı özellik değerlerinin aktarılması istenebilir. Bir başka deyişle, servisin ürettiği içeriğin kaynağındaki özelliklerin çıktıda map edileceği yerler, Atom içeriğindeki belirli noktalar olarak belirlenebilir.

çok doğal olarak eşleştirmenin çalışma zamanında değerlendirilmesi gerekmektedir. **Ado.Net Data Servislerin**, arka planda **Entity Framework** veya **Custom LINQ Provider** kullandığı düşünüldüğünde, eşleştirme işleminin nerede bildirileceği şu anda bizim için öğrenilmesi gereken yegane noktadır. Bir geliştirici olarak söz konusu eşleştirme bildirimlerinin**nitelik(attribute)** veya **konfigurasyon** bazlı olarak yapılacağının düşünülmesi son derece doğaldır ki bunların çalışma zamanında ele alınması gerekmektedir. Biz bu yazımızda önce eksikliği anlamaya çalışacak, sonrasında ise **Entity**

Framework kullanılması halinde eşleştirmeyi nasıl gerçekleştirebileceğimizi inceleyeceğiz.

öncelikle basit bir **Asp.Net Web** uygulaması oluşturalım ve aşağıdaki **EDM** grafiğinde görülen **Ado.Net Entity Data Model** öğesini söz konusu projeye ekleyelim. Her zamanki gibi kobay olarak AdventureWorks veritabanını hedef alıyor olacağız.

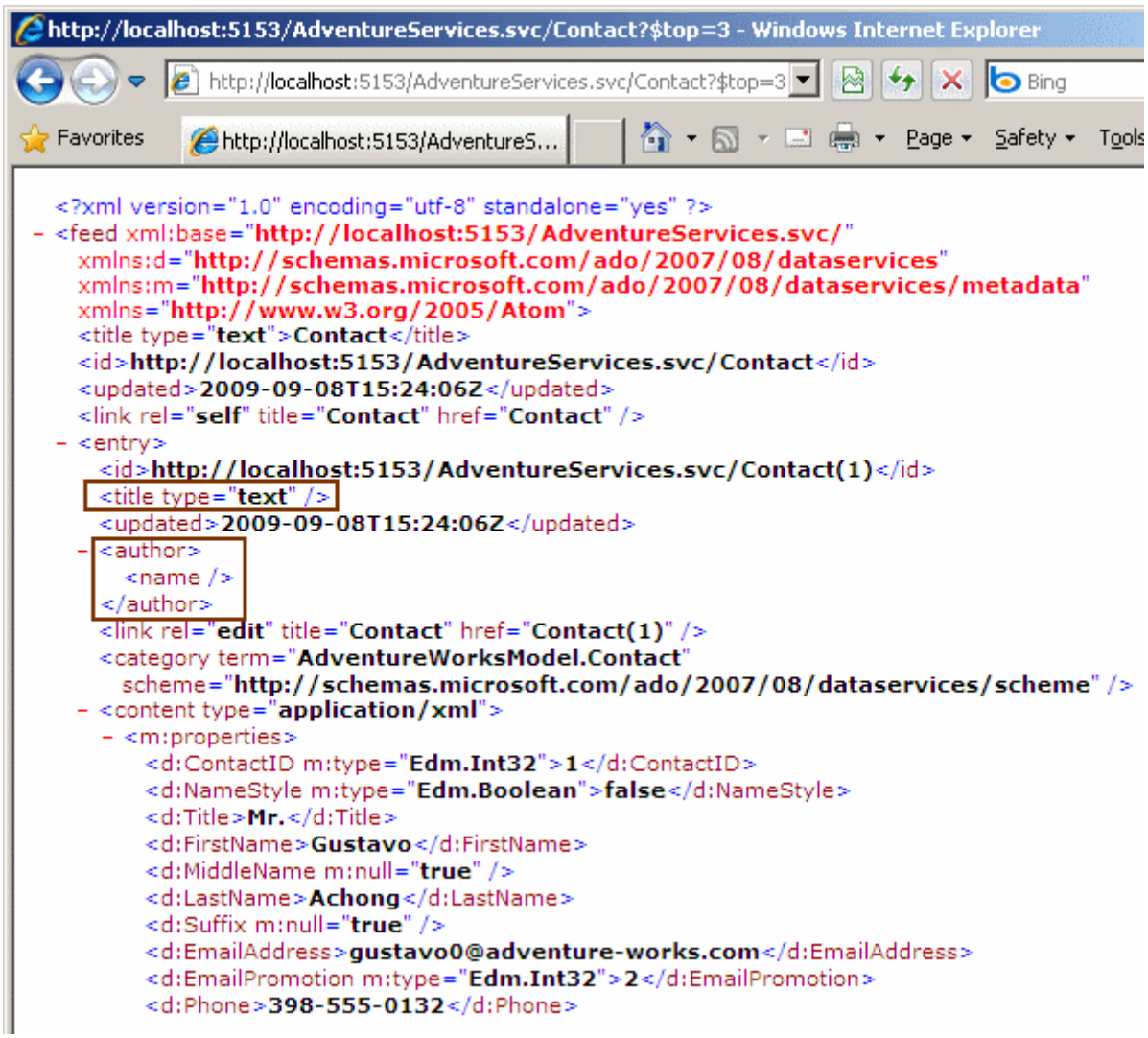


Contact tablosunu değerlendirmek amacıyla basit bir **Ado.Net Data Services v1.5 CTP2** öğesini projeye ekleyerek devam edelim. öğemizin kod içeriğini aşağıdaki gibi geliştirmemiz konumuz için yeterlidir.

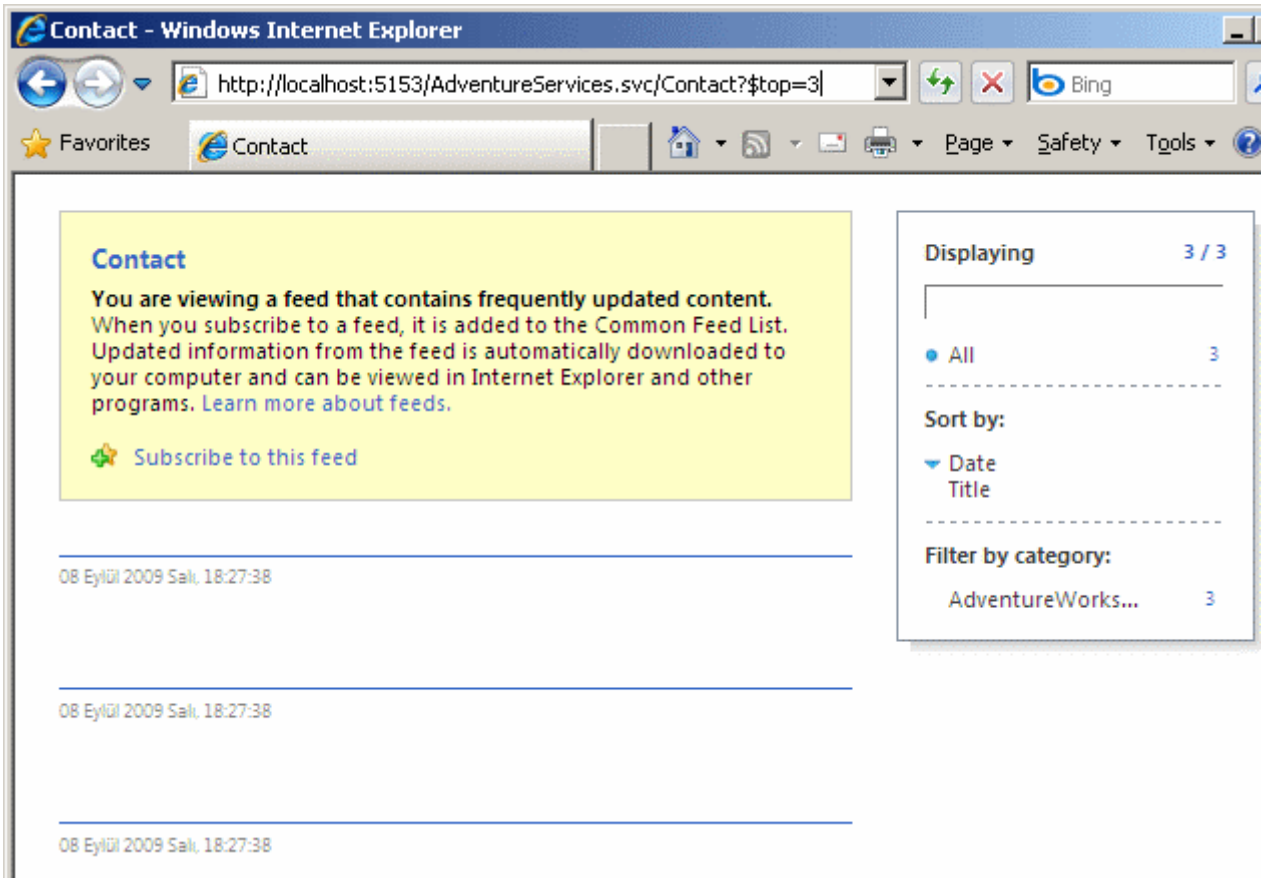
```
using System.Data.Services;

namespace WebFriendlyFeed
{
   public class AdventureServices
      : DataService<AdventureWorksEntities>
   {
      public static void InitializeService(DataServiceConfiguration config)
      {
         config.SetEntitySetAccessRule("*", EntitySetRights.AllRead);
         config.DataServiceBehavior.MaxProtocolVersion =
System.Data.Services.Common.DataServiceProtocolVersion.V2;
    }
```

```
    }
}
```

Söz konusu servis üzerinden Contact tablosundaki örneğin ilk 3 satırı talep ettiğimizde aşağıdakine benzer bir çıktı elde ederiz.



Eksiklik şudur; standart **atom feed** içeriğinde yer alan **author/name** ve **title** elementlerinin içeriği boştur. Bu bilgilerin eksik olması şu aşamada önemli değilmiş gibi görünebilir. Ama bu**atom feed** çıktısını değerlendiren bir uygulamada söz konusu alanlar belirli amaçlar ile kullanılıyor olabilir ve bu nedenden boş olmaları yararlı olmayabilir. örneğin **Internet Explorer'** ın feed içeriklerini görsel olarak yorumlama özelliğini**(Turn on feed reading view)** açtığımızda aynı sorgu aşağıdaki sonucu üretecektir.

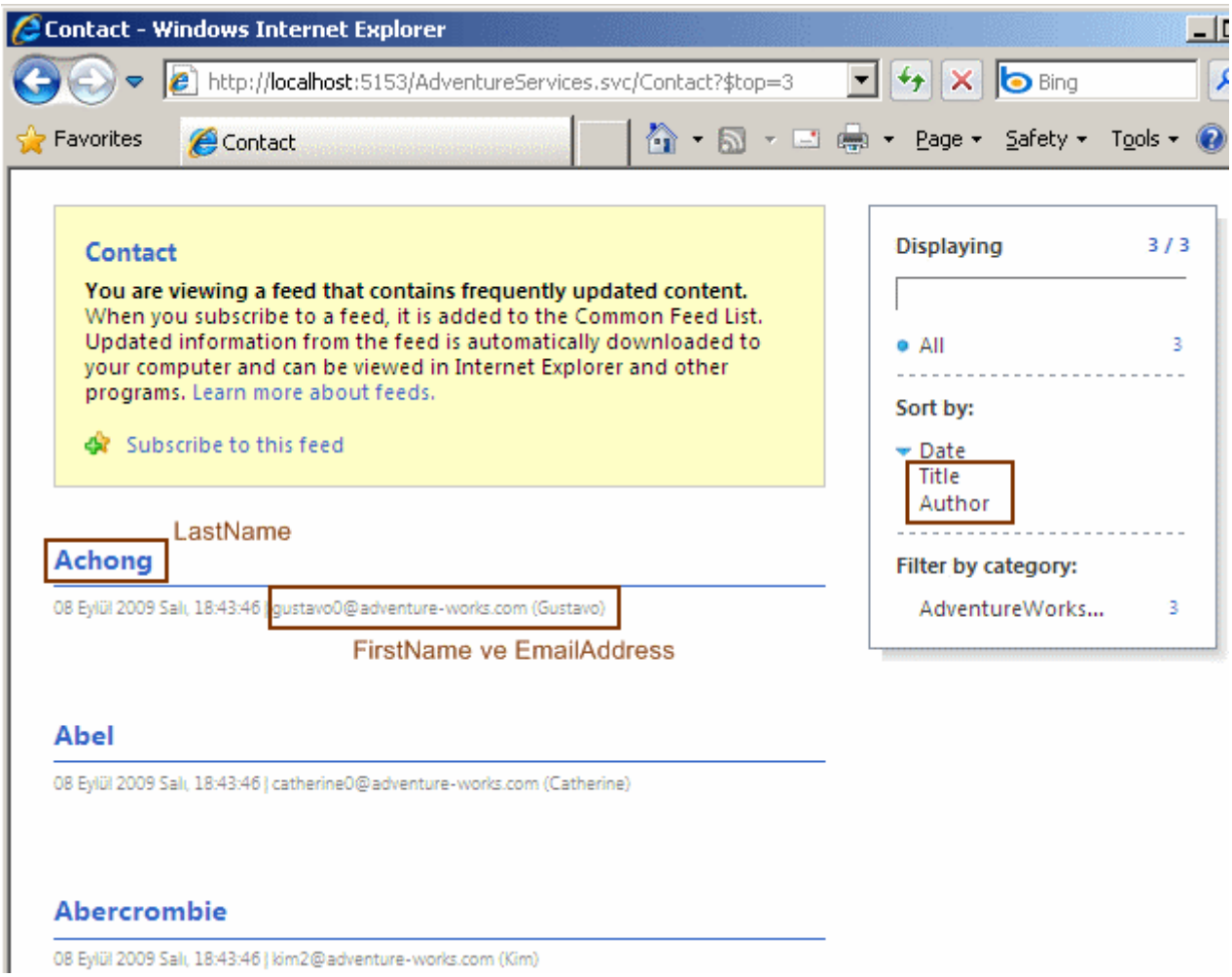


Görüldüğü gibi ilk etapta **feed entry'** leri ile ilişkili **title** veya **author/name** gibi elementler doldurulmadığı için okunabilir bir içeriğin oluşmadığı görülmektedir. üstelik **Title** gibi alanlara göre sıralama işlemi yapılmasıda mümkün değildir. İşte **Ado.Net Data Services 1.5** sürümünde getirilen **Web Friendly Feed** özelliği, söz konusu standart **entry elementlerinden** bazılarının, **entity** içeriğinden beslenmesini sağlayabilmektedir. Peki ama nasıl? 🤔 örneğimizde **Entity Framework** modeli kullanıldığından, **Conceptual Schema Definition Language(CSDL)** içeriğinde bazı ayarlamalar yapılması gerekmektedir. *(Tabi Entity modelinin **Update** edilmesi halinde bu değişikliklerin uçabileceğini hatırlatmam gerekir.)*Aşağıdaki şekilde görüldüğü gibi.

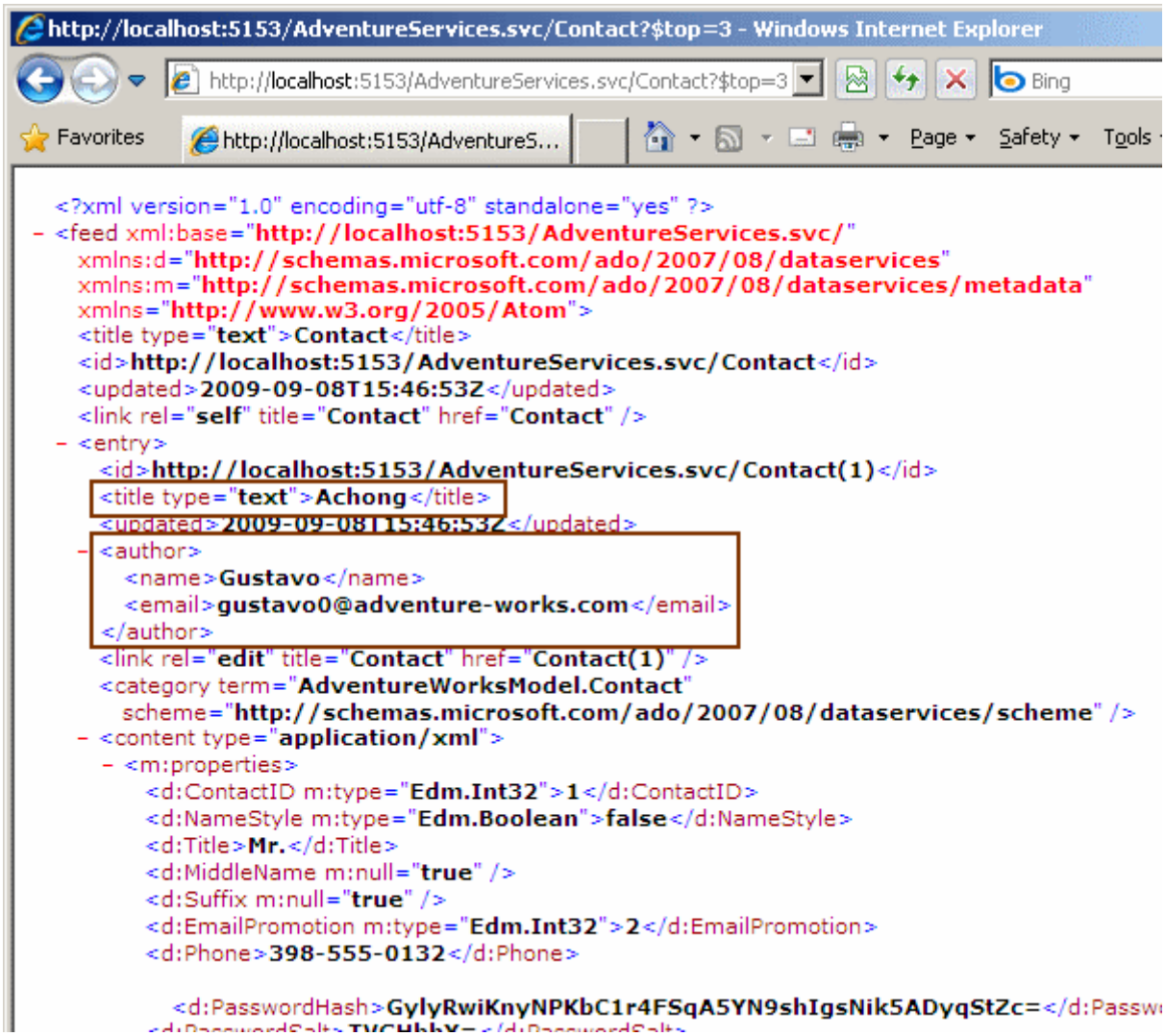
AdventureModel.edmx | AdventureServices.svc.cs | Web.config | Default.aspx

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <edmx:Edmx Version="1.0" xmlns:edmx="http://schemas.microsoft.com/ado/2007/06/edmx"
3          xmlns:friendlyFeed="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata">
4    <!-- EF Runtime content -->
5    <edmx:Runtime>
6      <!-- SSDL content -->
7      <edmx:StorageModels>...</edmx:StorageModels>
34     <!-- CSDL content = Conceptual Schema Definition Language -->
35     <edmx:ConceptualModels>
36       <Schema Namespace="AdventureWorksModel" Alias="Self" xmlns="http://schemas.microsoft.com/ad
37         <EntityContainer Name="AdventureWorksEntities">
38           <EntitySet Name="Contact" EntityType="AdventureWorksModel.Contact" />
39         </EntityContainer>
40         <EntityType Name="Contact">
41           <Key>
42             <PropertyRef Name="ContactID" />
43           </Key>
44           <Property Name="ContactID" Type="Int32" Nullable="false" />
45           <Property Name="NameStyle" Type="Boolean" Nullable="false" />
46           <Property Name="Title" Type="String" MaxLength="8" Unicode="true" FixedLength="false" />
47           <Property Name="FirstName" Type="String" Nullable="false" MaxLength="50"
48                 Unicode="true" FixedLength="false"
49                 friendlyFeed:FC_TargetPath="SyndicationAuthorName"
50                 friendlyFeed:FC_KeepInContent="false"
51                 />
52           <Property Name="MiddleName" Type="String" MaxLength="50" Unicode="true" FixedLength="fals
53           <Property Name="LastName" Type="String" Nullable="false" MaxLength="50" Unicode="true" Fixe
54                 friendlyFeed:FC_TargetPath="SyndicationTitle"
55                 friendlyFeed:FC_KeepInContent="false"
56                 />
57           <Property Name="Suffix" Type="String" MaxLength="10" Unicode="true" FixedLength="false" />
58           <Property Name="EmailAddress" Type="String" MaxLength="50"
59                 Unicode="true" FixedLength="false"
60                 friendlyFeed:FC_TargetPath="SyndicationAuthorEmail"
61                 friendlyFeed:FC_KeepInContent="false"
62                 />
```

İlk olarak bir **namespace** ilave edildiğini görüyoruz. Bu namespace ilave edilmez ise, **FC_KeepInContent** veya **FC_TargetPath** gibi nitelikleri kullanamayız. Yeri gelmişken bu niteliklerin ne iş yaptıklarını açıklayalım. **CSDL** içeriğinde yer alan **FirstName** özelliğinin değerinin **Atom Feed** içerisindeki **author/name** elementinde çıkması için **FC_TargetPath** niteliğine**SyndicationAuthorName** değeri atanmıştır. Benzer şekilde **Atom Feed** içeriğinin bir **Contact** için üretilen **Title** elementinde **LastName** özelliğinin değerinin çıkması için, FC_**TargetPath** niteliğine **SyndicationTitle** değeri atanmıştır. Aynı işlem **Email** adresi içinde gerçekleştirilmiş ve özellik değerinin **author/email** elementinde çıkması için**FC_TargetPath** niteliğine **SyndicationAuthorEmail** değeri atanmıştır. Buna göre **FC_TargetPath** niteliğinin değerinin, **entity** özelliğinin hangi **atom** alanında çıkacağını belirttiğini ifade edebiliriz. **FC_KeepInContent** niteliğine atanan **false** değeri ilede, **atom feed'** in element noktlarında çıkan değerlerin, üretilen **Contact** elementine ait **content** içeriğinde gösterilmemesisağlanmaktadır. Buna göre biraz önce yaptığımız talebi tekrarlarsak aşağıdaki çıktıları alırız.



Ve atom çıktısını içeriği;



Görüldüğü üzere **FirstName** ve **EmailAddress** bilgileri, **author** elementi altındaki **name** ve **email** alt elementlerine yazılmıştır. Ayrıca **entry'** nin **title** elementinin içeriğinede **LastName**özelliğinin değerinin yazıldığı görülmektedir. Bununla birlikte buradaki özellikler, **entry** elementi altındaki **content** elementi içeriğindende çıkartılmıştır. Süper Peki **Atom Feed Entry Elementlerinden** hangilerine atamalar yapabiliriz? İşte **Ado.Net Data Services v1.5 CTP2** eklentilerinin de yer aldığı son liste.

Entity tip özelliklerini hangi Atom Entry elementlerine eşleştirebiliriz.

author/email -> SyndicationItemProperty.AuthorEmail
author/name -> SyndicationItemProperty.AuthorName
author/uri -> SyndicationItemProperty.AuthorUri
published -> SyndicationItemProperty.Published
rights -> SyndicationItemProperty.Rights
summary -> SyndicationItemProperty.Summary *(CTP2 ile Gelmiştir)*
title -> SyndicationItemProperty.Title
Updated -> SyndicationItemProperty.Updated *(CTP2 ile Gelmiştir)*
contributor/name -> SyndicationItemProperty.ContributorName

contributor/email -> SyndicationItemProperty.ContributorEmail
contributor/uri -> SyndicationItemProperty.ContributorUri

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WebFriendlyFeed.rar (35,28 kb)

Screencast - Workflow Foundation 4.0 : Flowchart (2009-10-31T01:45:00)

wf 4.0,screencast,

Merhaba Arkadaşlar,

Bundan yaklaşık 1 sene kadar önce **Microsoft PDC 2008** sunumlarında gösterilen ve demoları yapılan **Workflow Foundation 4.0** ile ilişkili yenilikler arasında dikkat çekici olanlarından bir tanesi de, **Flowchart Workflow** modeliydi. Aslında bu yeni özellik, çoğu yazılımcının yaşam döngüsü içerisinde sıklıkla kullandığı akış diyagramlarının, **Workflow Foundation** modeli içerisinde ele alınmasından başka bir şey değildi ki **Workflow Foundation 4.0** öncesinde aradığımız ama bulamadığımız bir yenilikti. **Visual Studio 2010 Beta 2**sürümünün yayınlandığı şu günlerde, etkili **WPF** tasarım ortamınında katkısıyla, akış diagramı modeline uygun **Workflow** aktivitelerinin tasarlanması hem çok kolay hemde çok zevkli hale geldi. Bakalım görsel dersimizde bizleri neler bekliyor...

Yeni görsel derslerde görüşmek dileğiyle hepinize mutlu günler dilerim.

Screencast - Workflow Foundation 4.0 ve Unit Test (2009-10-28T23:30:00)

wf 4.0,unit test,screencast,

Merhaba Arkadaşlar,

Test güdümlü geliştirme(Test Driven Development) günümüz geliştirme süreçlerinden en popüler olanlarından birisidir. Tabiki test denince çoğu geliştiricinin aklına ilk gelen konu **birim testleridir(Unit Test). Visual Studio** gibi ortamların sağladığı hazır test araçları sayesinde, projelerin içerdiği birimlerin önce test edilerek geliştirilmeye başlanması oldukça kolaydır. Her türlü projenin birim testlerinin yapılması mümkündür.

İşte bu kavramların düşüncesiyle yürüyeceğimiz bu görsel dersimizde, **Workflow Foundation 4.0** ile geliştirilen **Activity'** lerin, **Unit Test** tipleri ile nasıl test edilebileceklerini incelemeye çalışmaktayız.

Yeni görsel derslerde görüşmek dileğiyle hepinize mutlu günler dilerim.

Screencast - Ado.Net Entity Framework 4.0 Yenilikleri (2009-10-28T05:07:00)

ado.net entity framework,screencast,

Merhaba Arkadaşlar,

Ado.Net Entity Framework 4.0 versiyonunda, daha önceki sürüme göre önemli yenilikler ve düzeltmeler bulunmakta. Bunlardan birisi **Model** diagramından yararlanarak veritabanının oluşturulabilmesi Bu sayede **Entity** tasarımlarının veritabanına aktarılmasını sağlayabilimekteyiz ki bu bir önceki sürümde olmasını istediğimiz yegane özelliklerden birisiydi. Bu önemli eklenti dışında, nesne isimlerinin çoğullaştırılması veya tekilleştirilmesi, **Foreign Key** alanlarının **Entity** tipleri içerisine istendiğinde **Property** olarak alınabilmesi veya **Complex Type** kullanımı gibi yeniliklerde yer almakta. Daha fazlası için sizleri görsel dersime davet etmek istiyorum.

Yeni görsel derslerde görüşmek dileğiyle hepinize mutlu günler dilerim.

WF 4.0 Beta 1' den Beta 2' ye (2009-10-27T15:25:00)

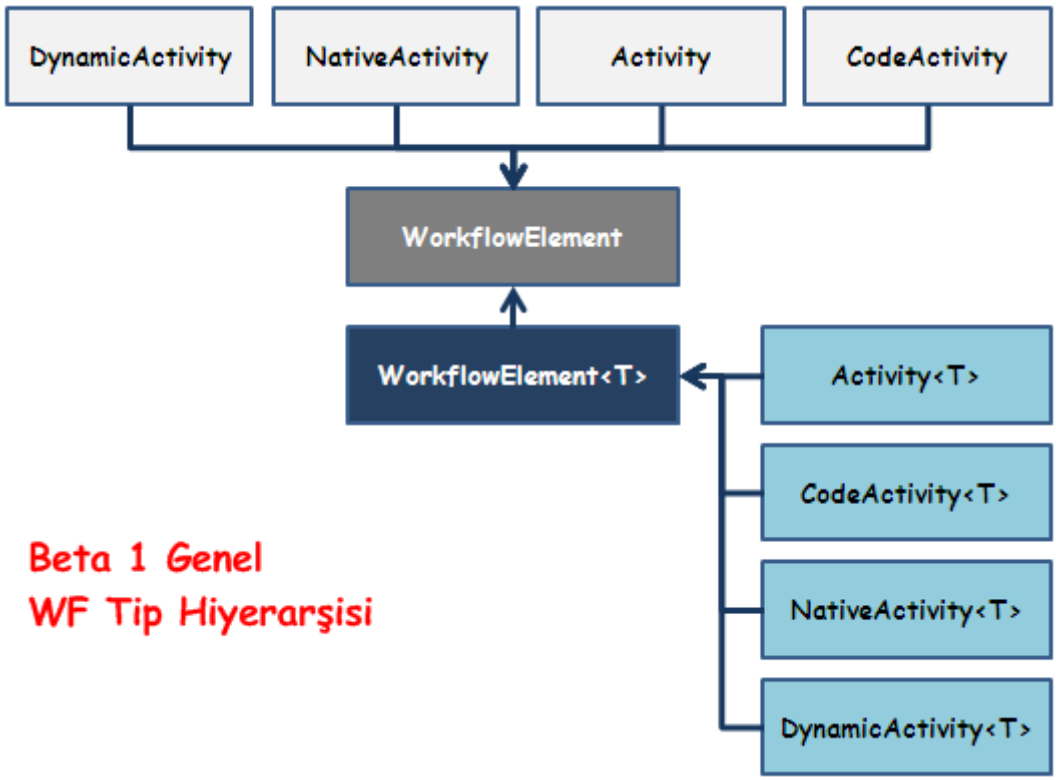
wf,wf 4.0,

Merhaba Arkadaşlar,

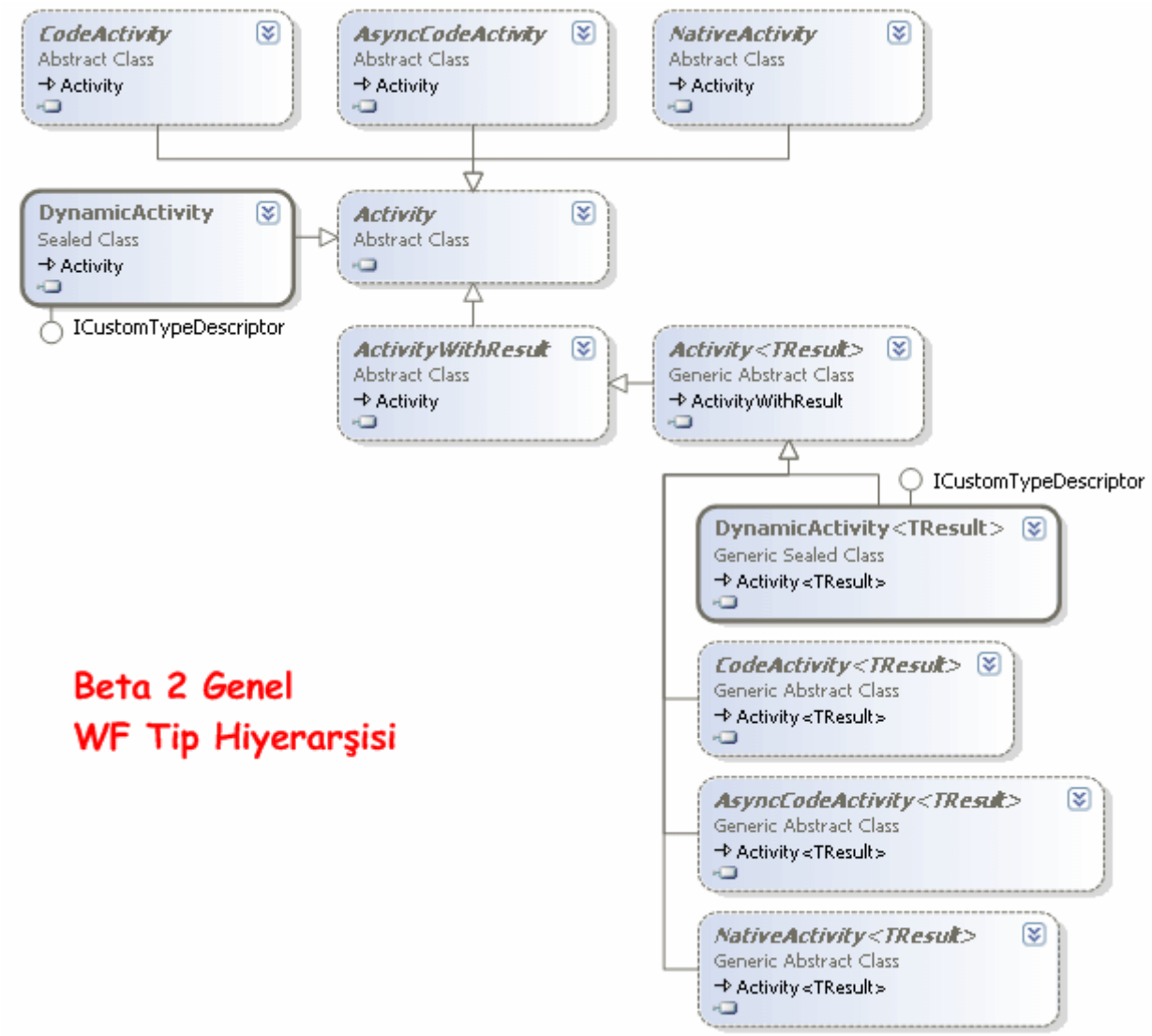
Hatırlıyorumda bundan bir kaç sene önce **Visual Studio 2005** in henüz **Build** sürümleri **CD** olarak değerli bir Microsoft çalışanı tarafından bana hediye edilmişti. **Whidbey** kod adlı sürümde pek çok yenilik olduğu gibi **IDE'** nin zaman zaman **istisnalar(Exception)** vererek çöktüğüne de sıklıkla şahit olmuştum. Tabiki **Microsoft** cephesinden gelen yenilikleri takip etmek, nasıl ilerlenildiğini izlemek adına **Alpha, Beta, RC** gibi sürümlerle çalışmanın oldukça faydası oluyor. En azından getirilen yeniliklerin neden değiştirildiğini, neden kaldırıldığını veya yeni eklenen bileşenlerin hangi amaçla düşünüldüğünü görme ve araştırma şansını buluyoruz. Bazende anlayamıyor veya bulamıyoruz 🙂 Tabi zaman zaman bulmacanın içerisinde kayıp parçalar da olabiliyor. İşte böyle bir durum kısa bir süre önce benim başımada geldi.

Workflow Service' ler ile çalışırken **Beta 1** sürümünde geliştirdiğim örneklerin **Beta 2** sürümünde ne yazık ki çalışmadığını farkettim. Bu son derece doğaldı çünkü kısa bir süre önce yayınlanan **.Net Framework Beta 2** sürümünde, **WF** tarafında özellikle tipler bazında bazı geri dönüşler ve değişimler meydana geldi. Hal böyle olunca konuyu hemen araştırmaya koyuldum. Pek çok blog yazısında **WF 4.0 Beta 1** ve **Beta 2** arasındaki farklılıkları bulabilirsiniz.

İşe ilk olarak daha önceden sıkça bahsedilen ata **Workflow** sınıfının **(WorkflowElement)** artık kaldırıldığını söyleyerek başlamak gerekiyor. Artık **Activity**isimlendirilmesine geri dönüş yapmış durumdayız. Aslında **Beta 1** sürümünde **Workflow** tip hiyerarşisinin aşağıdaki gibi olduğunu görmüştük.



Görüldüğü üzere tüm aktiviteler **WorkflowElement** veya **generic** olan versiyonundan türemektedir. Bunların haricinde özellikle paralel geliştirmeye yönelik herhangibir aktivite bileşenine yer verilmediği de görülmektedir. Ancak **Beta 2** sürümünde tip hiyerarşisi değişerek aşağıdaki sınıf diagramında görülen hale getirilmiştir.



İlk dikkati çeken noktalardan birisi **WorkflowElement** tipinin artık olmayışı ve **Activity** tipinin devreye alınışıdır. Buna ek olarak **Expression** kullanımları için **Activity<TResult>** ve türevleri getirilmiştir. Ancak belkide en önemli noktalardan birisi paralel işlemler için **asenkron** aktivite bileşenlerinin getirilişidir. **AsyncCodeActivity** ve **AsyncCodeActivity<TResult>** .

Tip hiyerarşisinin değişmesi elbetteki pek çok noktayı farklı şekilde etkilemektedir. Söz gelimi **XAML** içerikli bir **WF** dosyasının çalışma zamanında yüklenip yürütülebilmesi için **Beta 1** sürümünde aşağıdaki gibi bir kod parçası kullanılmaktaydı.

```
WorkflowElement workflow1;
using (Stream workflow1Stream= File.OpenRead("Workflow1.xaml"))
{
   workflow1= WorkflowXamlServices.Load(workflow1Stream);
}
var outputs = WorkflowInvoker.Invoke (workflow1);
```

önce **WorkflowElement** nesne örneği oluşturulmaktaydı. Sonrasında ise **XAML** içeriği bir **Stream** içerisine alınıp **WorkflowXamlServices** tipinin **static Load** metodu yardımıyla belleğe yüklenmekteydi. Sonrasında ise **WorkflowInvoker** tipinin **static Invoke** metoduna parametre olarak gönderilip, **XAML** içeriğinden **workflow'** un yürütülmeye başlanması sağlanmaktaydı. Di' li geçmiş zaman kullandığımı farketmiş olmalısınız *(Gerçi WorkflowInvoker.Invoke halen mevcut.)*

Ancak bu kodlama **Beta 2** sürümünde değişmiştir. Söz gelimi elimizde aşağıdaki uzun içeriğe sahip **Hesaplama.xaml** isimli bir **Workflow** dosyası olduğunu düşünelim.

***Kişisel Not : XAML** içeriğini kavramak için **<x:Members>** kısmından okumaya başlarsanız çok kolay bir şekilde anlaşılabileceğini görebilirsiniz*

<**Activity** mc:Ignorable="sap" x:Class="WorkflowConsoleApplication2.Hesaplama" xmlns="http://schemas.microsoft.com/netfx/2009/xaml/activities" xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006" xmlns:mv="clr-namespace:Microsoft.VisualBasic;assembly=System" xmlns:mva="clr-namespace:Microsoft.VisualBasic.Activities;assembly=System.Activities" xmlns:s="clr-namespace:System;assembly=mscorlib, Version=4.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089" xmlns:s1="clr-namespace:System;assembly=mscorlib" xmlns:s2="clr-namespace:System;assembly=System" xmlns:s3="clr-namespace:System;assembly=System.Xml" xmlns:s4="clr-namespace:System;assembly=System.Core" xmlns:sad="clr-namespace:System.Activities.Debugger;assembly=System.Activities" xmlns:sap="http://schemas.microsoft.com/netfx/2009/xaml/activities/presentation" xmlns:scg="clr-namespace:System.Collections.Generic;assembly=mscorlib" xmlns:scg1="clr-namespace:System.Collections.Generic;assembly=System" xmlns:scg2="clr-namespace:System.Collections.Generic;assembly=System.ServiceModel" xmlns:scg3="clr-namespace:System.Collections.Generic;assembly=System.Core" xmlns:sd="clr-namespace:System.Data;assembly=System.Data" xmlns:sd1="clr-namespace:System.Data;assembly=System.Data.DataSetExtensions" xmlns:sl="clr-namespace:System.Linq;assembly=System.Core" xmlns:st="clr-namespace:System.Text;assembly=mscorlib" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <x:Members>
    <x:Property Name="YourName" Type="InArgument(x:String)" />
  </x:Members>
  <mva:VisualBasic.Settings>Assembly references and imported namespaces serialized as XML namespaces</mva:VisualBasic.Settings>
  <Sequence sad:XamlDebuggerXmlReader.FileName="C:\Documents and Settings\bsenyurt\my documents\visual studio 10\Projects\WorkflowConsoleApplication2\WorkflowConsoleApplication2\Hesaplama.xaml" sap:VirtualizedContainerService.HintSize="264,342">
    <sap:WorkflowViewStateService.ViewState>

```
    <scg:Dictionary x:TypeArguments="x:String, x:Object">
      <x:Boolean x:Key="IsExpanded">True</x:Boolean>
    </scg:Dictionary>
  </sap:WorkflowViewStateService.ViewState>
  <Assign sap:VirtualizedContainerService.HintSize="242,57">
    <Assign.To>
      <OutArgument x:TypeArguments="x:String">[YourName]</OutArgument>
    </Assign.To>
    <Assign.Value>
      <InArgument x:TypeArguments="x:String">Burak Selim
Şenyurt</InArgument>
    </Assign.Value>
  </Assign>
  <Delay Duration="00:00:03" sap:VirtualizedContainerService.HintSize="242,22"
/>
  <WriteLine sap:VirtualizedContainerService.HintSize="242,59"
Text="[String.Format(&quot;Merhaba {0}&quot;, YourName)]" />
 </Sequence>
</Activity>
```

Bu **XAML** dosyasının çalışma zamanında yürütülmesi için **Beta 2** sürümüne ait kodlama aşağıdaki gibi olacaktır.

```
using System.Activities;
using System.Activities.XamlIntegration;

namespace WorkflowConsoleApplication2
{

  class Program
  {
    static void Main(string[] args)
    {
      Activity activity=ActivityXamlServices.Load("..\\..\\Hesaplama.xaml");
      var outputs=WorkflowInvoker.Invoke(activity);
    }
  }
}
```

öncelikli olarak **Activity** tipine ait bir nesne örneklenmektedir. Bu örnekleme işleminde **ActivityXamlServices** sınıfının **static Load** metodu kullanılır ve parametre olarak **XAML** dosyasını adres bilgisi verilir. Belleğe yüklenen **Activity** nesne örneğinin çalıştırılması için yapılan işlem **Beta 1** sürümündeki ile benzerdir.
Yine **WorkflowInvoker** sınıfının **static Invoke** metodu kullanılmaktadır. Ama tabi bu Invoke metodu Activity tipi ile çalışacak şekilde değiştirilmiştir.

Elbette başka farklılıklarda bulunmaktadır. Söz gelimi **WF 4.0** içerisinde önceki sürüme ait aktivite bileşenlerinin sarmalanarak çalıştırılmasına destek vermek amacıyla getirilen**Interop** aktivitesinde **doğrulama(Validation)** ve **Transaction Yönetimi** ile ilişkili iyileştirilmeler yapılmıştır.

özellikle **Workflow Service'** lerde **Send** ve **Receive** gibi aktivitelerin dış ortamdan aldıkları parametreler için **Parameters** kullanımından vazgeçilmiş bunun yerine ilgili aktivitelerin **Content**özellikleri içerisinden parametre seçimlerinin yapılabilmesine olanak sağlanmıştır.

Yine Workflow seyivesinde **tip güvenli(Type Safe)** özelliklerin tanımlanabilmesi ve bunlara Workflow nesne örnekleri üzerinden kod bazında erişilebiliyor olması söz konusudur. Bu konu ile ilişkili olarak ilgili adresteki görsel derslerimi izleyebilirsiniz.

Bunlar ve daha pek çok farklılığı veya yeniliği ilerleyen yazılarımızda veya görsel derslerimizde ele almaya çalışıyor olacağız. Tabiki **Release** sürüme yaklaştıkça çok şeyin değiştiğini de görebiliriz. Bu nedenle buradaki mimari modellerinde kalıcı olduğunu garanti etmemiz yanlış olacaktır. Ancak tüm bu yenilenmeler ve çalışmalar geliştiric olarak bizlerin iyiliği içindir. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Screencast - WF 3.5 ve WF 4.0[Beta 2] Parametre Kullanımı (2009-10-27T00:45:00)

wf 4.0,wf,screencast,

Merhaba Arkadaşlar,

Workflow Foundation 3.5 modelinde, **Workflow** bazında özelliklere değer aktarılması için **Dictionary<string,object>** tipinden generic koleksiyonlardan yararlanılmaktadır. Buna göre özelliğin adı **string** tipinden olan **key** yerine, içeriği ise **object** tipinden olan **value** yerine konumlandırılmaktadır. Bu koleksiyon sayesinde, **Workflow** üzerinde tanımlanmış olan n sayıda özelliğe ulaşmak ve değer atamak mümkündür. Ancak bu kullanım şekline göre tip güvenliği de**(Type safety)** ortadan kalkmaktadır. üstelik **Workflow** nesne örneği üzerinden ilgili özelliklere doğrudan erişim mümkün değildir.

Workflow Foundation 4.0 modelinde ise **Workflow** seviyesinde tanımlanan argumanlara tip güvenli olarak, **Workflow** nesne örnekleri üzerinden rahatlıkla erişilmektedir. üstelik argümanların eklenmesi için **WPF** tabanlı tasarım ortamı kolaylık sağlamaktadır. Ayrıca **XAML** içeriğine basit bir metin editörü kullanımı ile gerekli parametrelerin dışarıdan aktarılması ve kullanılması da sağlanabilmektedir. İşte bu görsel dersimizde söz konusu durumu önce **Visual Studio 2008** ardınan **Visual Studio 2010 Ultimate Beta 2** sürümleri üzerinden geliştireceğimiz basit örnekler ile mercek atlına alıyor olacağız.

Yeni görsel derslerde görüşmek dileğiyle hepinize mutlu günler dilerim.

Screencast - Workflow Service Geliştirmek [Beta 2] (2009-10-24T16:51:00)

wcf,wcf 4.0,screencast,

Merhaba Arkadaşlar,

WCF(Windows Communication Foundation) ve **WF(Workflow Foundation)** modellerinin bir arada kullanıldığı **Workflow Service'** lerin sunduğu kolaylıklardan birisi de, iş akışlarının servis bazlı olarak sunulabilmesi imkanıdır. Bir Workflow örneğinin çalışma ortamı içerisinde **Persistence** hizmetlerinden, **transaction** yönetiminden vb... yararlanması ve **Long Running Process'** lerin ele alınması sağlanabilir. üstelik buna **Visual Studio 2010** gibi geliştirme ortamlarında yer alan **WPF designer** desteğininde eklenmesi ve **XAML** bazlı olarak geliştirme yapılması da çok önemlidir. Bir **Workflow Service** ise, az önce bahsedilen fonksiyonellikleri taşıyan bir akışın servis bazlı olarak sunulabilmesine olanak tanımaktadır. Böylece, pek çok sistemin ortaklaşa kullandığı akışların servis bazlı olarak sunulabilmesi mümkün hale gelebilmektedir. Gerçek hayat senaryolarında **Workflow Service'** lerin değerlendirilebileceği pek çok alan bulunmaktadır. Ama öncesinde basit ve pekte işe yaramayacak bir **Hello World** örneği geliştirmekte yarar vardır. 😉

Yeni görsel derslerde görüşmek dileğiyle hepinize mutlu günler dilerim.

Screencast - .Net Framework 4.0 Beta 2 ile Gelen 5 Minik Yenilik (2009-10-23T22:10:00)

.net framework 4.0 beta 2,screencast,

Merhaba Arkadaşlar,

Ne yazıkki uzun süredir görsel derslere zaman ayıramıyordum. Ancak **.Net Framework 4.0 Beta 2** ve **Visual Studio 2010 Beta 2** sürümlerinin çıkması beni bu konuda biraz daha heveslendirmiş durumda. Şimdi sırada incelenmesi gereken onlarca konu olduğunu görüyorum. Bu amaçla ilk olarak **.Net Framework 4.0 Beta 2** sürümün ile gelen 5 kolaylaştırıcı fonksiyonelliği incelemeye çalıştım.

Görsel dersimizde öncelikle, **String** tipine eklenen **IsNullorWhiteSpace** metodunu inceliyoruz. Sonrasında **Stream** türevli tiplerin içeriklerini başka bir **Stream** türevli nesnesine kopyalamalarını sağlayan **CopyTo** metoduna bakıyoruz. **Guid** tipine eklenen yeni fonksiyonellikler ve **TryParse** metoduna değindikten sonra benzer konuyu **Enum** tipi üzerinden inceliyoruz. Son olarak **IEnumerable<T> arayüzü(Interface)** tipinden referanslar ile birlikte çalışabilen **String.Concat<T>** ve **String.Join<T>** metodlarına yakından bakıyoruz.

Kişisel Not : *Her ne kadar teknik bazı aksaklıklar nedeni ile Web Cam' deki hareketlerim akıcı görünmese de ses ve kodun senkronize olması zaten sizler için yeterli olacaktır kanısındayım.*

Yeni görsel derslerde görüşmek dileğiyle hepinize mutlu günler dilerim.

Windows Azure Service Platformu Hakkında İlk İzlenimler (2009-10-22T19:15:00)

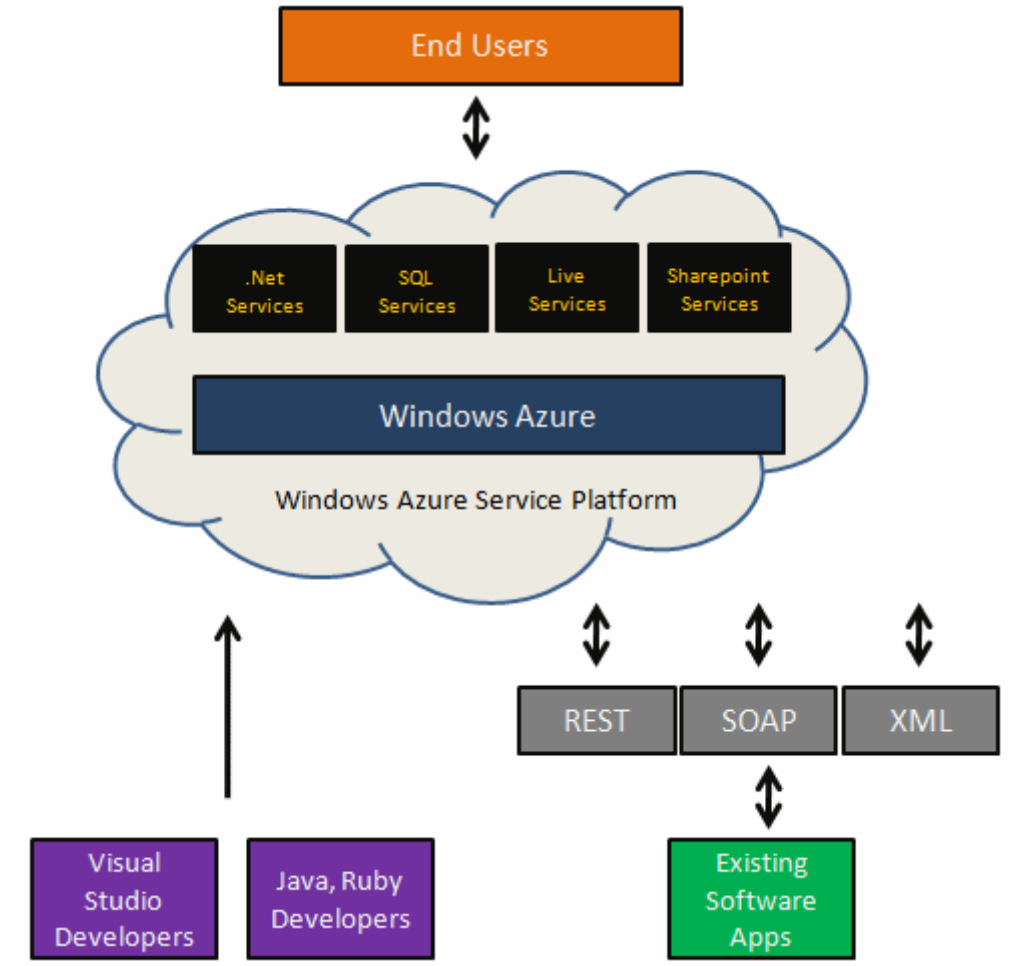
cloud computing,windows azure,

 Merhaba Arkadaşlar,

Güneşli açık bir hava ve sessiz bir gün yada gece...Deniz kenarında veya caddenin herhangibir köşesinde...Mutlaka zaman zaman gökyüzüne bakıp bulutları bazı nesnelere benzettiğimiz olmuştur. Yandaki resimde görüldüğü gibi gülen bir bulutla karşılaşma ihtimalimiz az olsada, arabaya, ördeğe veya başka bir şekle benzettiğimiz bulut sayısı oldukça fazladır. Bugünkü yazımızda bulutlar ile pek bir haşır neşir olacağımızı düşünebilirsiniz. Ancak tabiki yazılımsal anlamda

Microsoft' un son yıllarda **Cloud Computing** mimarisi için getirdiği geliştirmelerden biriside **Windows Azure Services** platformudur. Bu platformu servis bazlı bir işletim sistemi olarak düşünebiliriz. Ama bu çok basit bir yaklaşım olur. Bu konuda aslında pek çok kaynakta yazılmakta ve çizilmektedir. Ancak ürün henüz nihai halini almadığından sürekli olarak değişimlere uğramaktadır. Söz gelimi daha önce yayınlanan **SDK** içerisine**Microsoft.Net Services** bloku içerisinde yer alan **Workflow Services** modeli, **Temmuz 2009'** da yayınlanan **CTP** sürümünde kaldırılmışdır. Aslında konunun detaylarına girdiğimde çok kısa bir sürede kaybolduğumu ifade edebilirim. Bu nedenle yazımızın bundan sonraki kısımlarında mimarinin ne getirdiğini sizlere araştırma sonuçlarımdan aktarmaya çalışacağım.

Windows Azure Service Platformu iki ana parçayı içermektedir. İlk parça olan **Windows Azure**, bulut tabanlı işletim sistemidir. Bu işletim sistemi üzerinde ise çeşitli servis inşa blokları(**Building Blocks**) yer alır*(Microsoft SQL Services, Microsoft .Net Services, Live Services, Microsoft Sharepoint Services, Microsoft Dynamics CRM Services)* gibi. Aşağıdaki şekilde söz konusu mimari model daha net görülebilmektedir.



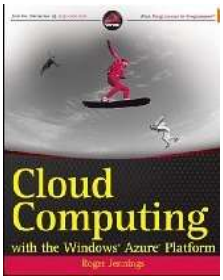
Windows Azure platformunu aslında şu cümle ile tanımlayabiliriz; **Azure**, **internet tabanlı bulut bilişim ortamını(Internet Based Cloud Computing)** sağlayarak, uygulamaların çalıştırılması veya verinin saklanması için **Microsoft veri merkezlerinin(Data Centers)** kullanılabilmesini sağlayan bir platformdur. Bu tanımlamaya göre **Windows Azure Service** platformunu bir **Cloud Computing Fabrikası** olarak düşünülebilir. öyleki bir veri merkezi olarak, uygulama veya servislerin **Internet** üzerinde **geliştirilmesi(Development)**, **dağıtılması(Deployement, yönetilmesi(Management)** mümkündür. Bu açıdan bakıldığında **Windows Azure Service** platformunun iki temel fonksiyonelliği olduğu söylenebilir. Uygulama çalıştırmak ve veri saklamak.

Peki **Internet** üzerinde yer alan böyle bir bulut kümesinin ne gibi bir faydası olabilir? Herşeyden önce uygulamaların çalıştırılacağı verinin fiziki olarak saklanacağı ortama ait **IT**karmaşıklığını düşünememize gerek kalmamaktadır. Bir başka deyişle **host** ortamının fiziki sunucu özelliklerinin ayarlanması, oluşturulması, bunlar için gerekli yazılımların tesis edilmesi gibi maliyetlerin düşünülmesine gerek kalmamaktadır. üstelik **Cloud** içerisine dahil edilen uygulama veya veri kümelerinin istenen anda istenilen yerden kullanılabilmesi de mümkündür.

Windows Azure tarafında önem arz eden kavramlardan olan **veri depolama(Data Storage)** bloku, **BLOB(Binary Large OBjects), kuyruk(Queue)** ve **basit tablo(Simple Table)** verilerinin saklanması için gerekli servisleri sunmaktadır. Ancak aksine ilişkisel veritabanları tarafından sunulan, **sorgulama(query), arama(Search),raporlama(Reporting)** vb yetenekleri sağlamamaktadır. Bu yeteneklerin, **Cloud**üzerinde saklanan veri kümelerinde kullanılabilmesi için **Microsoft SQL Services'** lerden yararlanılması gerekmektedir.

Microsoft .Net Services şu anda 2 temel fonksiyonelliği karşılamaktadır. **Uygulamalar arası bağlanabilirlik(Application Connectivity)** ve **Erişim kontrolü(Access Control).** Uygulamalar arası bağlanılabilirlik için **Microsoft .NET Services Bus** bloku kullanılmaktadır. Bu blok, çeşitli mesajlaşma desenlerine göre **Internet** üzerindeki uygulamalar arasında bir **network** altyapısının oluşturulabilmesini sağlamaktadır. Erişim kontrolü için **Microsoft .NET Access Control Service** bloku ele alınır. Bu blok ile aslında **Claims** tabanlı erişim kontrolünün Cloud üzerinde gerçeklenmesi sağlanır.

Aslında **Microsoft .Net Service'** leri, bütünüyle servis bazlı geliştirme fabrikası olarak düşünülebilir. **Microsoft .Net Service'** lerini geliştirirken kullanılan **SDK**, **WCF** ile entegrasyon da sağlamaktadır. Bu sayede var olan **.Net** geliştirme teknikleri ile tasarlanan **WCF** uygulamalarının **Cloud** servisi haline getirilmesi mümkündür. önemli olan noktalardan bir diğeride, **Microsoft .Net Service'** in sadece **.Net** geliştiriciler için tasarlanmış olmayışıdır. Desteklenen **REST, SOAP, WS*** gibi protokoller sayesinde **Java** ve **Ruby** gibi diller için geliştirilmiş olan **SDK'** lardan yararlanarak da **Cloud** servisleri geliştirilebilir.



Windows Azure platformu ile ilişkili olarak başlangıç noktamız http://www.microsoft.com/windowsazure/ adresi olmalıdır. Bu adresten gerekli **SDK** ve **Visual Studio 2008/2010 Beta X** araçlarının indirilierek kurulması geliştirilmeye başlanması için yeterlidir. Ek olarak örneğin .Net Services geliştirilmesi yapılacaksa bu konu ile ilişkili SDK' nınhttp://www.microsoft.com/windowsazure/developers/dotnetservices/ adresinden tedarik edilmesi gerekmektedir.

Ben bu konuya oldukça meraklıyım aslında. Nitekim işin içerisinde **Service** kavramı var 😎 Bu nedenle bende boş durmadım ve önümüzdeki sene **Cloud Computing** ve**Windows Azure** ile ilişkili olan araştırmalarımı daha sağlıklı devam ettirebilmek adına **Amazon'** dan **Cloud Computing with the Azure Platform** adlı kitabı

sipariş ettim. Bakalım önümüzdeki sene **Azure** ile ilişkili olarak ne gibi bir macera beni*(ve doğal olarak siz değerli okurlarımı)* bekliyor olacak. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

C# 4.0 - ExpandoObject (2009-10-21T16:14:00)

c# 4.0,dynamic language runtime,.net framework 4.0 beta 2,

Merhaba Arkadaşlar,

Bildiğiniz üzere **.Net Framework 4.0** ile birlikte gelmesi muhtemel en köklü yenilikler arasında **Dynamic Language Runtime** alt yapısı yer almaktadır. Bu anlamda daha önceden **dynamic** anahtar kelimesini inceleyerek tiplerin dinamik olarak oluşturulup kullanılmasını kavramaya çalışmıştık. Bu yazımızda nasıl bir yenilikten bahsedeceğimizi anlatabilmek için öncelikle aşağıdaki kod parçasına odaklanmanızı istiyorum.

Not : *örnek henüz yayınlanmış olan* ***Visual Studio 2010 Ultimate Beta 2*** *sürümü üzerinde geliştirilmiş bir Console uygulamasıdır.*

```
Console.WriteLine("İlk Bakış\n");

dynamic employee = new ExpandoObject();

employee.Name = "Burak";
employee.Salary = 1000.23F;
employee.Birth = new DateTime(1976, 12, 4);
employee.WorkingArea = new ExpandoObject();
employee.WorkingArea.City = "Istanbul";
employee.WorkingArea.Degree = 1;
employee.WorkingArea.CustomerCount = 190;

Console.WriteLine(String.Format("\tName:{0} City:{1} Degree:({2})", employee.Name,
employee.WorkingArea.City, employee.WorkingArea.Degree));
```

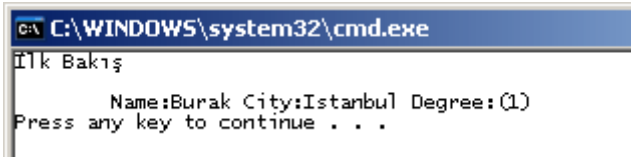
Şimdi bu kod parçası ile ilişkili olara bir kaç ipucu vermek istiyorum;

1. öncelikli olarak uygulamamızda **Name,Salary** ve diğer özelliklere sahip **Employee** gibi ismi olan herhangibir **tip(type)** tanımı bulunmamaktadır
2. İkinci kodu yazdığımız sırada **employee** isimli değişkene herhangi bir özellik eklemek istediğimizde aşağıdaki bilgi penceresi ile karşılaşırız.

```
dynamic employee = new ExpandoObject();

employee.Name = "Burak";
employee.Salary = 1000.23F;
employee.Birth = new DateTime(1976, 12, 4);
employee.WorkingArea = new ExpandoObject();
employee.WorkingArea.City = "Istanbul";
employee.WorkingArea.Degree = 1;
employee.WorkingArea.CustomerCount = 190;
employee.
          (dynamic expression)
          This operation will be resolved at runtime.
```

Ahaaaa!!! 😉

Sanıyorum olayın sizde farkına varmış durumdasınız. Aslında **employee** ismiyle tanımlamış olduğumuz değişkeni bir sınıf nesne örneği olarak inşa ettiğimizi düşünebiliriz. Ancak bunun için **Emplyee** tipini tanımlamış değiliz. Tamamen kodlama zamanında verdiğimiz bazı kararlar uygulamaktayız. örneğin **employee** ismiyle tanımlanan tipin ve içeriğinin oluşturulması aşaması tamamen **çalışma zamanına(Runtime)** bırakılmış durumdadır. **employee** isimli değişkenin **Name, Salary, Birth** gibi özellikleri dışında başka bir **ExpandoObject** tipine işaret eden **WorkingArea** isimli bir özelliği daha bulunmaktadır. Buna göre **WorkingArea** özelliği de bir tip olarak düşünülebilir ki **City, Degree, CustomerCount** gibi özellikleri yer almaktadır. Tabiki çalışma zamanında, **özelliklere(Properties)** atanan değerlere göre tiplerin ne olacağı da belirlenmektedir. Buna göre örneğin **Name**özelliğinin çalışma zamanında **string** tipinden olacağı açık ve nettir. Ancak **WorkingArea** özelliğinin kendiside aslında bir **ExpandoObject** tipidir. Kodun ilerleyen kısımlarına baktığımızda, employee değişkeninin özelliklerini kullanabildiğimizi de görebiliriz. örnek kod parçamızın çalışma zamanı çıktısı aşağıdaki gibi olacaktır.

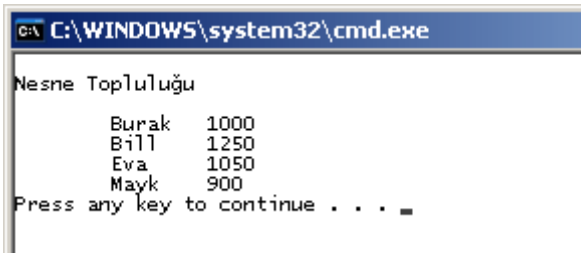


ExpandoObject tipinin aslında hangi amaçla kullanıldığını ilk etapta görmüş olduk. Peki daha neler yapabiliriz? **örneğin bir liste veya dizi oluşturulmasında, ExpandoObject nesnelerini kullanabilir miyiz?** İşte cevabımız;

```
Console.WriteLine("\nNesne Topluluğu\n");

dynamic personList = new List<dynamic>();

personList.Add(new ExpandoObject());
personList[0].Name = "Burak";
personList[0].Salary = 1000;
```

```
personList.Add(new ExpandoObject());
personList[1].Name = "Bill";
personList[1].Salary = 1250;

personList.Add(new ExpandoObject());
personList[2].Name = "Eva";
personList[2].Salary = 1050;

personList.Add(new ExpandoObject());
personList[3].Name = "Mayk";
personList[3].Salary = 900;

foreach (var person in personList)
   Console.WriteLine("\t{0}\t{1}", person.Name, person.Salary);
```

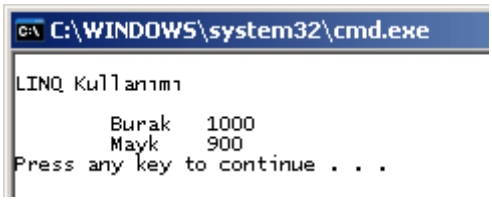
Bu sefer **List<dynamic>** tipinden bir koleksiyon oluşturulduğu ve koleksiyonun her bir nesnesinin **ExpandoObject** tipinden tanımlandığı görülmektedir. Buna göre örnek olarak **Name** ve **Salary**özellikleri olan nesnelerin **dynamic** listeye eklendiği görülebilir. Tabiki her öğe oluşturulma işleminden önce koleksiyona **ExpandoObject** tipinden bir nesne örneği eklenmesi gerekmektedir. örneğin çalışma zamanındaki çıktısı aşağıdaki gibidir.



Hatta istersek çalışma zamanında oluşturulacak bu nesneler üzerinde **LINQ** sorgularının çalıştırılmasını da sağlayabiliriz. Aşağıdaki kod parçasında yukarıdaki listenin bir **LINQ** ifadesi ile sorgulandığı görülmektedir.

```
Console.WriteLine("\nLINQ Kullanımı\n");

var result = from person in (personList as List<dynamic>)
   where person.Salary <= 1000
   select person;

foreach (var r in result)
   Console.WriteLine("\t{0}\t{1}", r.Name, r.Salary);
```

Bu kez **dynamic** liste içerisindeki nesnelerden **Salary** özelliği **1000** birime **eşit ve az** olanların çekilmesi için bir **LINQ** ifadesi kullanılmaktadır. çalışma zamanındaki sonuç aşağıdaki gibi olacaktır.

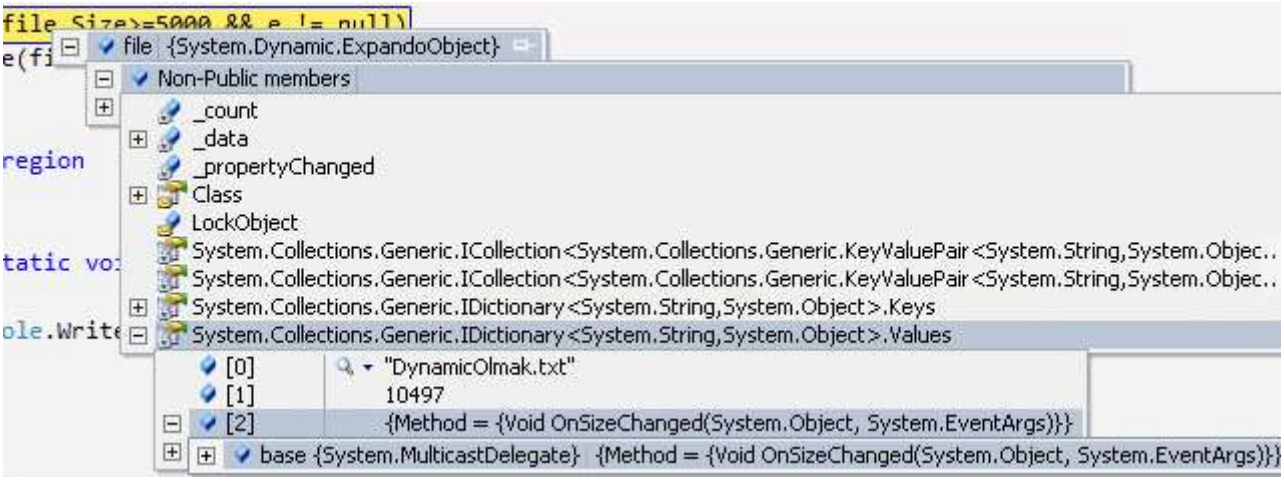


Daha neler yapabilir? **Acaba tanımladığımız ExpandoObject tipinden nesne için bir olay(Event) ekleyebilir miyiz?** Hımmm... Bu oldukça ilginç olabilir. Aşağıdaki kod parçası ile bir deneyelim bakalım;

```
static void Main(string[] args)
{
  #region Event Kullanımı

  dynamic file = new ExpandoObject();

  file.Name = "DynamicOlmak.txt";
  file.Size = 1024;

  file.SizeChanged = null; // Bu satır olmadığı takdirde hata mesajı alınmakta.
  file.SizeChanged += new EventHandler(OnSizeChanged);
  EventHandler e = file.SizeChanged;

  Random rnd = new Random();
  file.Size += rnd.Next(1000, 10000);

  if (file.Size>=5000 && e != null)
    e(file, null);

  #endregion
}

public static void OnSizeChanged(object sender, EventArgs e)
{
  Console.WriteLine("Boyut değişti olayı tetiklendi");
}
```

Bu sefer **file** isimli değişkenin işaret ettiği tipe **SizeChanged** isimli bir **olay(event)** bildirimi yapılmaktadır. **Event'** ler bildiğiniz üzere **delegate** tipleri ile ilişkilidir. örnekte basitlik açısından standart **EventHandler** temsilcisi kullanılmıştır ki dilerseniz farklı **handler'** ları veya kendi tanımladığınız **delegate** tiplerini de ele alabilirsiniz. Olayın **file** nesnesine eklenmesi sırasında birde olay metodunun işaret edildiği görülmektedir. Bildiğiniz üzere bu olay metodu, **EventHandler** temsilcisinin belirttiği parametrelere ve dönüş tipine sahip olmalıdır. Buna göre**OnSizeChanged** metodu olayın gerçekleşmesi sonrası devreye girecek olan fonksiyonumuzdur. çok doğal olarak olayın bir

sebepten tetikleniyor olması gerekir. örnekte **Size** değerinin **5000'** in üzerinde olması durumu değerlendirilmeye çalışılmıştır. Eğer böyle bir koşul oluşur ve **e** ile ifade edilen olay **file** nesnesine daha önceden ilave edilirse söz konu olay metodu tetiklenecektir. Aslında uygulamayı **Debug** ettiğimizde söz konusu olay metodu için **MulticastDelegate** kullanıldığını rahatlıkla görebiliriz.

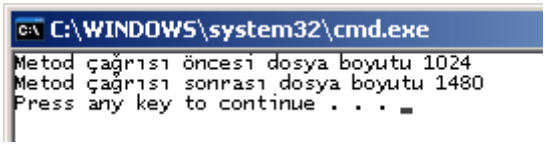


Dikkat edileceği üzere dinamik olarak üretilen nesne içerisinde yer alan **IDictionary** koleksiyonunda **Name, Size** özellikleri ile birlikte **MulticastDelegate** tipinden tanımlanmış bir metod bildirimde yer almaktadır.

Peki ExpandoObject örneklerini metodlara parametre olarakta geçirebilir miyiz? Elbette Aşağıdaki kod parçasını göz önüne alalım;

```
static dynamic file = new ExpandoObject();

static void Main(string[] args)
{
  #region Metoda Parametre Aktarımı

  file.Name = "DynamicOlmak.txt";
  file.Size = 1024;

  Console.WriteLine("Metod çağrısı öncesi dosya boyutu {0}",file.Size);
  Action(file);
  Console.WriteLine("Metod çağrısı sonrası dosya boyutu {0}", file.Size);

  #endregion
}

static void Action(dynamic obj)
{
  obj.Size += 456;
}
```

Burada **file** değişkeni sınıf seviyesinde tanımlanmıştır. Tabi örneği **Console** uygulamasında geliştirdiğimizden **dynamic** olarak tanımlanan **ExpandoObject** tipinin başına **static**anahtar kelimesininde gelmesi gerekmektedir. Bu kodu okurken biraz tuhaflığa neden olmaktadır. **ExpandoObject** örneği **Action** metoduna **dynamic** tipi üzerinden aktarılmaktadır. Dolayısıyla **Action** metodu içerisinde kullanılan **obj** değişkeninin **file** değişkeni olduğunu metod çağrısını yaptığımız yerde belirtmekteyiz. Bu nedenle **Action** metodu içerisine zaten**file** değişkeninin geldiğini biliyor olmalıyız. (*Peki hangi tipin geldiğini anlayabilir miyiz? İşte size süper bir araştırma konusu*) Metod içerisinde sembolik olarak **Size** değerini arttırmaktayız. İşte çalışma zamanı sonucu;



ExpandoObject tipi görüldüğü üzere dinamik olarak tiplerin oluşturulması, onlara üye tanımlanması(özellik, Event gibi), özelliklerine değer atanması gibi işlemleri yapabilmemize olanak sağlamaktadır. Bu anlamda test senaryolarında **stub** veya **moc** nesneler için kullanılabilecek bir yenilik olarak düşünebiliriz belkide. Aklıma ilk gelen kullanım alanın bu olması, belkide **The Art of Unit Testing** kitabını okuyor olmamadan da kaynaklanabilir. Aslında söz konusu yeniliği tanımak, kavramak ve gerçek hayat senaryolarındaki yerini anlamak için biraz daha araştırma yapmamız, çalışmamız gerektiği ortadadır. En azından benim...

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ConsoleApplication1.rar (25,51 kb)

WCF Known Types Analizi (2009-10-20T22:36:00)

wcf,

Merhaba Arkadaşlar,

Bilindiği üzere **WCF** aslında **SOA(Service Oriented Architecture)** mimarisinin uygulama modellerinden birisidir. İşin içerisinde servisler söz konusu olduğunda ağlar ve sistemler arası mesajlaşlamalar söz konusudur. Mesajlaşmalar söz konusu olduğundaysa, servis ve istemci arasında hareket eden verinin serileşebilir olması önem arz eden konuların başında gelmektedir. Ne varki serileşen veri içeriklerinin, platform bağımsızlık adına her iki tarafında kullanabileceği **tiplerden(Types)**oluşmasının sağlanması bir avantajdır. İşte bu noktada biz **WCF** geliştiricileri için anlaşılması zor olan ve dikkatle üzerinde durumlası gereken kıyıda köşede kalmış konulardan biriside **Known Types** kavramıdır.

WCF ile birlikte gelen ve serileştirmede kullanılan tipler esas itibariyle **Shared Contracts** kategorisindendir. Nitekim serileşen tipin ve kullandığı içeriğin**Interoperabilitiy** kuralları çevresinde değerlendirilebiliyor olması gerekir. **DataContractSerializer**, **JsonDataContractSerializer** ve **XmlSerializer** gibi serileştirici tipler bu kategoride yer almaktadır. Diğer yandan **BinaryFormatter**, **SoapFormatter** veya **NetDataContractSerializer** gibi tipler, **Shared Types** kategorisinde yer alan serileştiricilerdir. Genellikle serileşen tiplerin içerdiği tip bilgilerinin tanımlandığı **Assembly'** lar, uygulama ile aynı makinede yer alır. örneğin serileştirilebilir tipin içeriğinin**.Net CLR** tiplerinden oluştuğunu düşünelim. Bu durumda ters serileştirme işlemini üstelenen uygulamanında bu **CLR(Common Language Runtime)** tiplerini biliyor olması, bir başka deyişle uygun **Framework Assembly'** larına sahip olması yeterlidir. Ancak **SOA** tabanlı bir sistemde **serileştirme(Serialize)** ve **ters-serileştirme(Deserialize)** yapan tarafların aralarında taşıdıkları tiplerle ilişkili olarak ortak bir noktada buluşmaları gerekmektedir. Nitekim hem sağlayıcı hemde tüketici taraflar kendi platformlarında kendi özel veri tiplerine sahiptir ve sadece bu tipleri kullanabilir. Bu noktada tipin **XSD(XmlSchemaDefinition)** şemaları içerisinde tanımlandığını*(dolayısıyla serileşen paketler içerisinde taşındığını)* ve örneğin **Proxy'** yi üreten tarafta

bile kullanılabilecek bir tipe karşılık gelmesi gerektiğini söyleyebiliriz. Tabiki buradaki veri tipi bilgileri arada transfer edilen **XML** içeriklerinde ortaktır.

Aslında bu teorik bilgiler, yazarken bile insanın kafasını allak bullak etmeye neden olabilir. Bu yüzden kafamızda sorunsal haline gelebilecek bu konuyu örnekler üzerinden açıklamakta yarar olacağı kanısındayım. İşe ilk olarak **Visual Studio 2008** ortamında oluşturulumuş bir **Console** uygulaması ve aşağıdaki kod parçası ile başlayalım.*(System.Runtime.Serialization referansının eklenmiş olması gerektiğini unutmayın)*

```
using System.IO;
using System.Runtime.Serialization;

namespace KnownTypes
{
   class Program
   {
      static void Main(string[] args)
      {
         // DataContractSerializer nesnesi örneklenirken parametre olarak serileştirilecek tip bilgisi verilir.
         XmlObjectSerializer serializer = new DataContractSerializer(typeof(Product));
         // Serileştirme işlemi yapılır. İlk parametre ile çıktının Product.xml isimli dosyaya yaplacağı belirtilir.
         // İkinci parametrede Product nesnesi örneklenir. Dikkat edilecek nokta Information özelliğine object tipinden bir nesne örneğinin aktarılmış olmasıdır.
         serializer.WriteObject(
            new FileStream("Product.xml", FileMode.Create, FileAccess.Write)
            , new Product { Information = new object() }
            );

         // Object tipine string atama
         serializer.WriteObject(
            new FileStream("ProductV2.xml", FileMode.Create, FileAccess.Write)
            , new Product { Information = "ürün hakkında çeşitli bilgiler" }
            );

         // Exception durumu
         //serializer.WriteObject(
         //    new FileStream("ProductV3.xml", FileMode.Create, FileAccess.Write)
         //    , new Product { Information = new ProductInformation { Id=1, Summary="ürün için çeşitli bilgiler" } }
         //    );
```

```
    }
  }

  [DataContract]
  class Product
  {
    [DataMember]
    public object Information { get; set; }
  }

  // [DataContract]
  // class ProductInformation
  // {
  //    [DataMember]
  //    public string Summary { get; set; }
  //    [DataMember]
  //    public int Id { get; set; }
  // }
}
```

örneğimizde **Product** isimli serileştirilebilir bir tip tanımlandığı görülmektedir. **DataContract** niteliği ile işaretlenmiş olan **Product** tipinin **object** tipinden **Information** isimli bir özelliği de bulunmaktadır ki işleri karıştıracak olan nokta burasıdır. Program kodu içerisindeki amaç, **Product** tipinden nesne örneklerinin nasıl serileştirildiğine bakmaktır. **serializer** isimli**XmlObjectSerializer** nesne örneği üzerinden yapılan ilk **WriteObject** çağrısındai **Information** özelliğine **object** tipinden bir değer atandığı görülmektedir. İkinci **WriteObject**çağrısında ise bir **string** değer atanmıştır ki bu son derece doğaldır. *(Hatırlayalım .Net tipleri en üstte Object tipinden türemektedir. Bu nedenle her nesnenin Object tipi ile ifade edilebilmesi mümkündür.)* Bu kod parçasını çalıştırdığımızda **Product.xml** ve **ProductV2.xml** isimli iki dosya üretildiğini görürüz. önce **Product.xml** içeriğine ve şemasına bir bakalım.

Product.xml içeriği;

```
<Product xmlns="http://schemas.datacontract.org/2004/07/KnownTypes"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance"><Information/></Product>
```

Product.xsd içeriği;

```
<?xml version="1.0" encoding="windows-1254"?>
<xs:schema xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://schemas.datacontract.org/2004/07/KnownTypes"
```

```
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Product">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Information" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Tahmin ettiğimiz ve beklediğimiz gibi bir çıktı üretilmiştir.
Ancak **Information** özelliğine **string** bir değer atanmasının sonucu oluşan çıktı biraz farklıdır. Bu noktada dikkatle duralım

ProductV2.xml içeriği;

```
<Product xmlns="http://schemas.datacontract.org/2004/07/KnownTypes"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Information i:type="a:string" xmlns:a="http://www.w3.org/2001/XMLSchema">ürün
hakkında çeşitli bilgiler</Information>
</Product>
```

ProductV2.xsd içeriği;

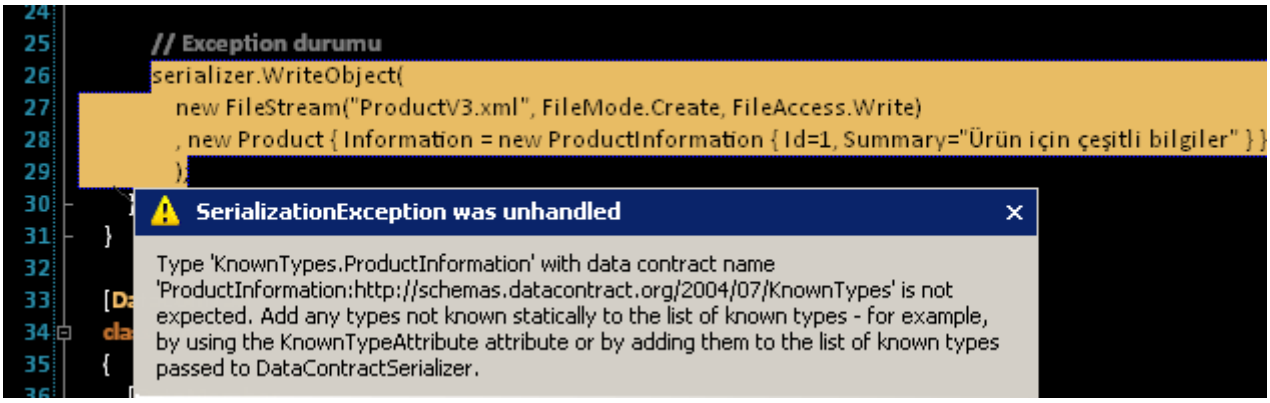
```
<?xml version="1.0" encoding="utf-8"?>
<a:schema xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:a="http://www.w3.org/2001/XMLSchema" attributeFormDefault="unqualified"
elementFormDefault="qualified"
targetNamespace="http://schemas.datacontract.org/2004/07/KnownTypes">
  <xs:element name="Product">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="Information" type="xs:string" />
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</a:schema>
```

XSD şemasında, **Information** elementi için **type** niteliğinde **xs:string** kullanıldığı görülmektedir. üstelik üretilen veri içeriğinde
de **i:type** niteliğinde **Information** elementinin içeriğinin **string** veri tipinden olduğu işaret edilmektedir. Yani **Information** özelliğine atadığımız **string** değişken, **primitive** bir tip tanımana göre XML içerisinde bildirilmiştir. Zaten**primitive** tiplere dönüşüm yapıldığı takdirde pek bir sıkıntı yoktur. Söz gelimi **Information** özelliğine örneğin **12** değerini

atadığımızda buna uygun olarak **XML**içeriğinde de **int** tipinin kullanıldığı görülebilir. Sorun yorum satırı kodlarını çalışır hale getirdiğimizde ortaya çıkmaktadır.
Yeni **ProductInformation** tipini etkinleştirip aşağıdaki kodları çalıştırdığımızda...

```
// Exception durumu
        serializer.WriteObject(
            new FileStream("ProductV3.xml", FileMode.Create, FileAccess.Write)
            , new Product { Information = new ProductInformation { Id=1,
Summary="ürün için çeşitli bilgiler" } }
            );
```

Bu sefer **Information** özelliğine serileştirilebilir*(ki **DataContract** ve **DataMember** nitelikleri nedeni ile)* **ProductInformation** tipinden bir nesne örneği atanmaktadır. Bu durumda çalışma zamanında aşağıdaki ekran görüntüsünde yer alan **SerializationException** istisnasının alındığı görülür.



Bu istisna mesajından anlamamız gereken özlü söz şudur;

"Serileştirici tiplerin, serileştirecekleri nesne örneklerinin özelliklerinin tiplerinin neler olabileceğini açık bir şekilde bilmeye ihtiyaçları vardır. "

Her ne kadar **primitive** bir tip kullanıldığında sorun olmasa da, yukarıdaki örnekte görüldüğü gibi bilinmeyen bir tipin atanmasında sorunlar yaşanabilir. üstelik belkide atanan verinin tipinin, serileştirici tarafından varsayılan olarak atanan bir tip olmaması da gerekebilir. Söz gelimi **Information** özelliğine noktasız sayısal bir değer atandığında büyüklüğüne göre varsayılan olarak **int** tipi göz önüne alınacak ve **XML** içeriğinde bu yönde bir tanımlama olacaktır. Ki karşı tarafta belkide bu sayısal değerin **string** olarak değerlendirilmesi isteniyor olabilir! Upsss...

Peki serileştiricinin serileştirdiği tipin içeriğinde kullanılabilecek tipleri kesin olarak bilmesi nasıl sağlanabilir?

Yöntemlerden birisi ve belkide en basiti, aşağıdaki kod parçasında olduğu gibi serileştirilecek tip için **KnownType** niteliğini kullanmaktır.

Kişisel Not: *Başka yöntemlerde bulunmaktadır. örneğin tip içerisinde Type[] dizisi döndüren bir metodun adı KnownType niteliğinde kullanılarak birden fazla tipin bildirimi yapılabilir. Ya da servis sözleşmesinde ServiceKnownType niteliğinde bu bildirim yapılabilir. İşte size güzel bir araştırma konusu. Bu tekniklerin nasıl uygulanabileceğini araştırabilirsiniz. özellikle ilk teknik dikkate değerdir. Yani bir metod aracılığıyla, KnowType niteliğine birden fazla tipin Type[] dizisi olarak bildirilmesi. Dikkat edilmesi gereken tek nokta Type[] dizisi döndüren metodun static olmasını sağlamaktır. Hatta buradan bir adım öteye gidip generic bir modelin KnownType niteliğine söz konusu metod yardımıyla aktarılması dahi sağlanabilir. Bu kadar ipucu yeter. Haydi klavye başına* 🙂

```
[DataContract]
[KnownType(typeof(ProductInformation))]
class Product
{
  [DataMember]
  public object Information { get; set; }
}
```

Bu şekildeki kullanım sonrasında artık **SerializationException** istisnası üretilmeyecek ve aşağıdaki çıktılar oluşacaktır.

ProductV3.xml içeriği;

```
<Product xmlns="http://schemas.datacontract.org/2004/07/KnownTypes"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
  <Information i:type="ProductInformation">
    <Id>1</Id>
    <Summary>ürün için çeşitli bilgiler</Summary>
  </Information>
</Product>
```

Dikkat edileceği üzere **ProductInformation** nesne örneği içeriği ile birlikte **Product** elementi içerisine alınmıştır. Bu durumda şema içeriğinde gerekli bilgilendirmelerin aşağıdaki gibi yapıldığı görülecektir.

ProductV3.xsd içeriği

```
<?xml version="1.0" encoding="utf-8"?>
<xs:schema xmlns:i="http://www.w3.org/2001/XMLSchema-instance"
attributeFormDefault="unqualified" elementFormDefault="qualified"
targetNamespace="http://schemas.datacontract.org/2004/07/KnownTypes"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Product">
    <xs:complexType>
      <xs:sequence>
```

```
      <xs:element name="Information">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Id" type="xs:unsignedByte" />
         <xs:element name="Summary" type="xs:string" />
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
</xs:schema>
```

Known Type sorunsalının bir sorunsal olarak değerlendirilmesinin ise iki sebebi vardır.

Birincisi, **SOA** düşünce tarzına aykırı olduğu görüşününün yaygın olmasıdır. Nitekim **Shared Contract'** ların söz konusu olduğu senaryolarda **Interoperability** sağlanırken, **XML** içerisindeki tipin karşı tarafça anlaşılabilir olması gerekmektedir. İkinci olarak serileştirme ve ters serileştirme işlemleri sırasında **Known Type'** a göre **Reflection**mekanizmasının devreye girmesi, **XML** üzerinde tip ile ilişkili bilgi edinme ve yazma gibi operasyonların söz konusu olmasından kaynaklanan performans kayıpları değerlendirilmektedir. Bu iki sebep nedeniyle zorunlu kalınmadıkça **Known Type** kullanımından kaçınılması önerilmektedir. Peki neden böyle bir konuya değindik? Aslında bir sonraki yazımıza zemin hazırlamaya çalışıyoruz. Nitekim **WCF 4.0** tarafında bu konu ile ilişkili bir yenilik gelmesi muhtemeldir. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

KnownTypes.rar (22,64 kb)

Organik Yazılım Günü 1.5 (2009-10-20T10:00:00)

seminer,

İşte uzun bir aradan sonra yeniden açık kaynak etkinliğinde buluşma fırsatına ne dersiniz? İlk etkinliğimizde yaşadıklarımız ve açıklığa getirdiğimiz ilginç bakış açısı sonrasında :) şimdilik 1.5 sürümü olarak nitelendirdiğim bu etkinliğe hepinizi bekliyorum! Güzel ve hızlı bir açık kaynak günü olacak! (Daron Yöndem)

[WCF 4.0 Yenilikleri - Workflow Services [Beta 2] (2009-10-19T22:22:00)]

wcf,wcf 4.0,wf,wf 4.0,

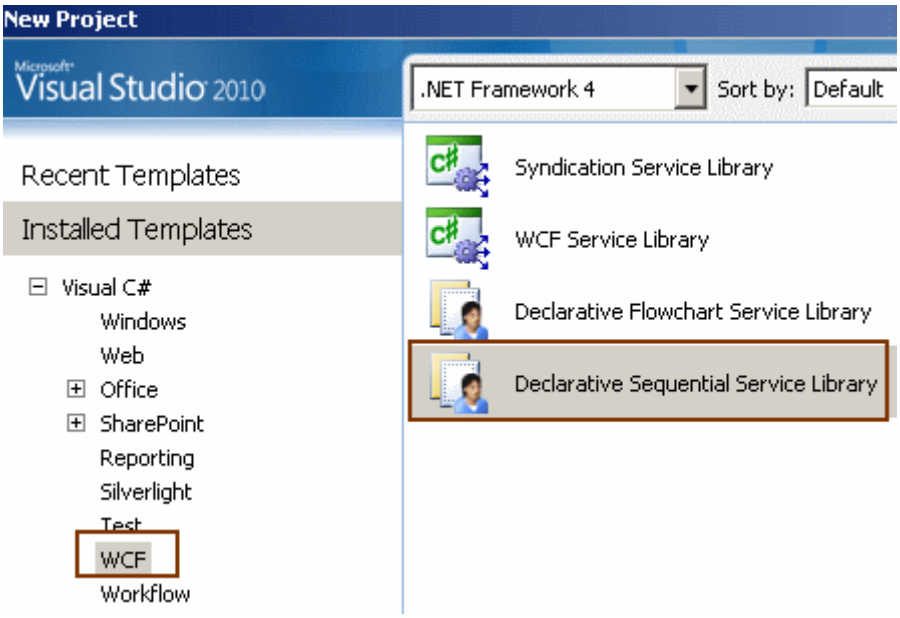
Merhaba Arkadaşlar,

WCF ve **WF** arasında ilişkiyi anlatan güzel bir cümle vardır. **.Net Framework 3.0'** da arkadaş olan **WCF** ve **WF**, **.Net Framework 3.5** sürümünde nişanlanmış, **.Net Framework 4.0** sürümünde ise evlenmişlerdir. 🙂 Bu ikilinin bir arada ele alınması

sonucu **Workflow Services** adı verilen bir ufaklıkta ortaya çıkmıştır. Aslında bir akışın servis bazlı olması son derece önemlidir. Nitekim **WF** tarafında, uzun süreli işlemlerin ele alınması**(Long Running Process),**akışın çeşitli noktalarından farklı anlar için kalıcı olarak saklanabilmesi**(Persistence),** asenkron çalışma zamanı motorunun zaten hazır olarak bulunması söz konusudur. Bu özelliklerin **WCF Servis Noktaları(Endpoints)** ile desteklenmesi, bir akışın ağ üzerindeki istemciler tarafından başlatılabilmesine olanak tanımaktadır. üstelik bu akışlar sonuçlar üreterek bunları geriye de döndürülebilir. Bu durum aslında **WCF 3.5** sürümünde de gerçeklenebilmektedir. Ne varki, **WCF 4.0** tarafında tamamen **declerative** olarak tanımlanabilen ve saf **XAML(eXtensible Application Markup Language)** içeriğinden oluşan bir **Workflow Servisinin** geliştirilebilmesi mümkündür. üstelik **Visual Studio 2010 WF Designer** sayesinde bu **XAML** içeriğinin üretilmesi için IDE desteği de sunulmaktadır.

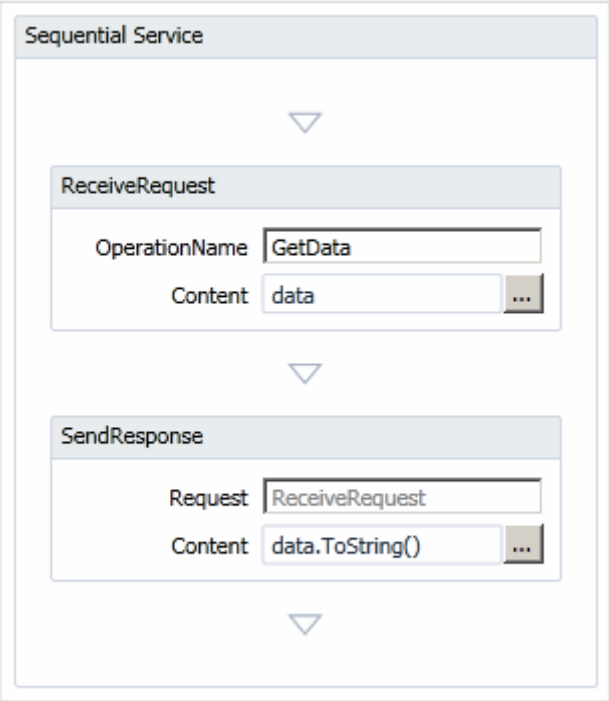
***Kişisel Not : Windows XP, Vista** ve 7 sürümlerinde kullandığım **Visual Studio 2010 Beta 1** ürününde, **WF Designer** sürekli olarak hata verip **IDE' yiRestart** işlemine zorlamaktaydı.* 😕 *Neyseki bu sorun **Visual Studio 2010 Beta 2** sürümünde ortadan kaldırılmış durumdadır.* 🙂

Dolayısıyla bu yazımızın konusu şimdiden belli oldu. çok basit ve büyük ihtimalle pek işe yaramayan bir **Workflow Service'** in nasıl geliştirilebileceğini incelemeye çalışıyor olacağız. İşe ilk olarak **Declerative Sequential Service Library** tipinden bir **WCF projesi** oluşturarak başlayacağız. Aşağıdaki ekran görüntüsünde olduğu gibi.



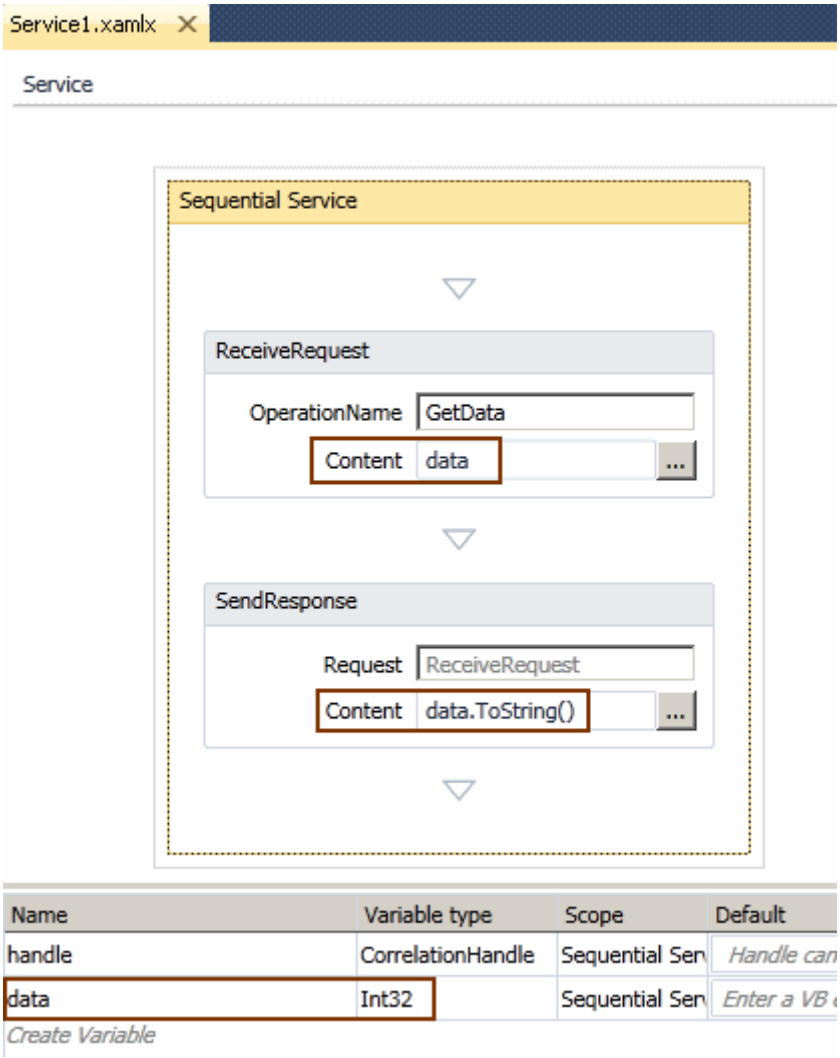
örnek olarak **AdventureWFServices** ismiyle oluşturduğumuz projede **Service1.Xamlx** dosyası hemen dikkati çekmektedir. Bu örnek olarak oluşturulan **Workflow Service** örneğidir. Yani Microsoft tarafından sunulan hazır Hello World örneği 🙂 Tamamen **XAML** içeriğinden oluşmaktadır ve istemcilere servis olarak nasıl sunulacağına ilişkin çalışma zamanı bilgileri(WCF Servis konfigurasyon bilgileri) **Web.config** dosyasında tutulmaktadır. *(Bu bir WCF kütüphanesi projesi*

*olduğundan doğrudan çalıştırılabilir ve Web.config içerisindeki ayarlar sayesinde HTTP bazlı olarak host edilir. Diğer taraftan test etmek için **WcfTestClient** aracından kolayca yararlanılabilir.)*



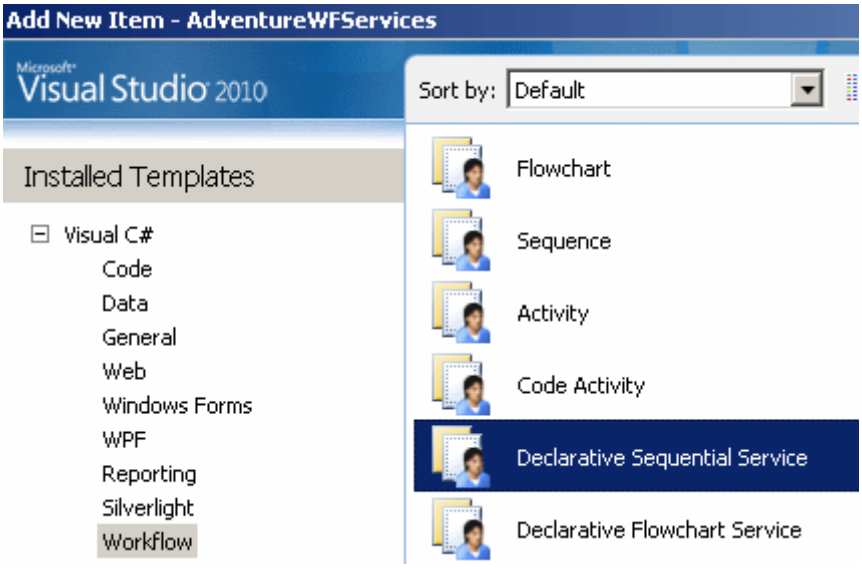
Biz tabiki bu içerik yerine kendi **Workflow Service** örneğimizi kullanacağız. Ama öncesinde **ReceiveRequest** ve **SendResponse** isimli bileşenlerin ne işe yaradıklarını kısaca anlamaya çalışmakta yarar vardır. **ReceiveRequest** isimli bileşen aslında **Receive** tipinden bir aktivitedir. Benzer şekilde **SendResponse** isimli bileşende, **SendReply** tipinden bir aktivitedir. Tahmin edileceği üzere **Receive** aktivitesi ile, istemci tarafından gelen **talep(Request)** alınmakta ve **SendReply** aktivitesi yardımıylada bir **cevap(Response)** döndürülmektedir.*(Bu iki bileşen arasında ise istenen bir akışın yürütüldüğü düşünülebilir)* Bir başka deyişle WCF bazlı mesaj alınması ve cevap döndürülmesi amacıyla kullanılan iki aktivite tipi söz konusudur.**ReceiveRequest** isimli bileşen aynı zamanda istemci tarafından çağırabilecek bir operasyon sunduğundan **OperationName** isimli bir özelliği de sahiptir. Diğer taraftan istemci tarafından gelecek olan talep içerisindeki değişkenler **Content** özelliği tarafından taşınabilirler ki bu özellik aslında bir koleksiyon olarak düşünülebilir. çünkü operasyonun birden fazla parametre alması gerekebilir. **Receive** aktivitesi için önem arz eden noktalardan biriside, **CanCreateInstance** özelliğinin değeridir. Bu değer örneğimizde **true** olarak set edilmiştir ve söz konusu aktiviteye bir talep gelmesi ile birlikte Workflow örneğinin oluşturulup oluşturulmayacağını işaret etmektedir. önemli olan noktalardan biriside, **SendResponse** isimli bileşenin **Request** özelliğinin değerinin,**Receive** tipinden olan **ReceiveRequest** isimli bileşen olmasıdır. Dolayısıyla bu iki mesajlaşma aktivitesinin birbirleriyle haberleştiklerini söyleyebiliriz. *(İlerki yazılarımızda burada sözü edilen korelasyon konusuna değiniyor olacağız ki XAML içeriğinde bu konunun izlerini görmekteyiz.)*

Bu arada **Sequential Service** tipinin üzerinde tanımlanmış bazı **değişkenler(Variables)** olduğu görülebilir.

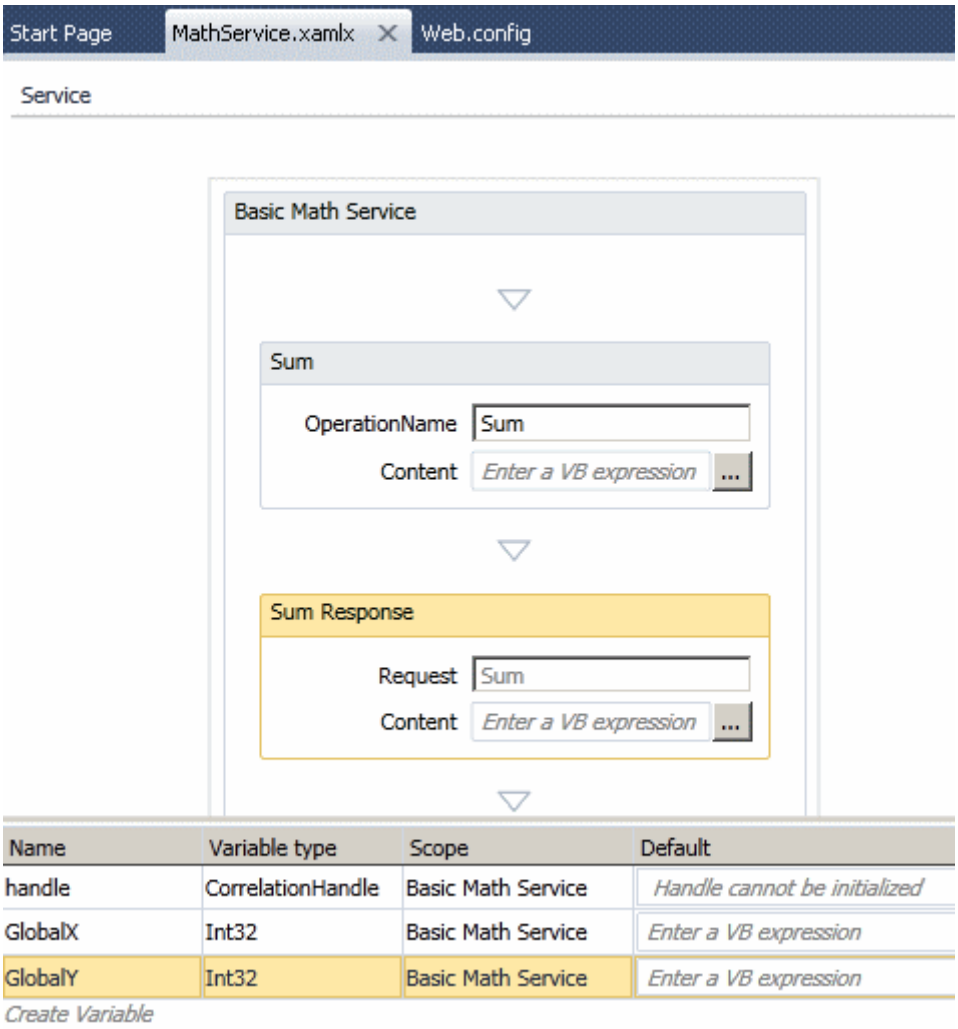


data isimli **Int32** tipinden olan değişken, **Receive** aktivitesi için **Content** değerini de temsil etmektedir. Diğer taraftan, **SendReply** aktivitesine ait **Content** özelliğinde kullanılmış ve **string** karşılığı üretilerek geriye döndürülmüştür. Buna göre akışın senaryosunu şu şekilde düşünebiliriz. İstemci talep olarak **Int32** tipinden bir değer gönderir. Bu değer**ReceiveRequest** isimli aktivite bileşeni tarafından alınır ve üst aktivitesi olan **Sequential Service'** in **data** değişkenine atanır. Bu değişken **SendResponse** isimli aktivite bileşeninin de ulaşabileceği kapsamda(Scope) olduğundan, istemciye geriye gönderilecek cevabın içeriğinde kullanılabilir. Şimdi bu bilgilerden yola çıkarak kendi **Workflow Servisimizi** oluşturalım.

örnek olarak istemciden gelen iki sayının toplamını bulup geriye bu sonucu döndürecek bir Workflow Servis geliştirmeyi planlayabiliriz. Böylece **XAML** içeriğini anlamamız daha kolay olacaktır. Bu amaçla projemize yeni bir **Declerative Sequential Service** öğesi ekleyerek devam edebiliriz.



MathService.xamlx servisinin tasarım görüntüsü aşağıdaki gibidir.



Aslında global seviyede tanımladığımız GlobalX ve GlobalY isimli iki değişken haricinde çok fazla detay görülmemektedir. Bu nedenle konuyu anlamanın en iyi yolu XAML

içeriğine bakmamız olacaktır(Zaten ilerleyen zamanlarda bu blog girişinin destekleyici görsel videosunuda yayınlıyor olacağım) İşte MathService.xamlx içeriği;

```
<Service mc:Ignorable="sad1"
xmlns="http://schemas.microsoft.com/netfx/2009/xaml/servicemodel" xmlns:a="clr-
namespace:AdventureServices;assembly=AdventureServices"
xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
xmlns:mv="clr-namespace:Microsoft.VisualBasic;assembly=System" xmlns:mva="clr-
namespace:Microsoft.VisualBasic.Activities;assembly=System.Activities"
xmlns:p="http://schemas.microsoft.com/netfx/2009/xaml/activities"
xmlns:p1="http://adventure/" xmlns:s="clr-namespace:System;assembly=mscorlib"
xmlns:s1="clr-namespace:System;assembly=System" xmlns:s2="clr-
namespace:System;assembly=System.Xml" xmlns:s3="clr-
namespace:System;assembly=System.Core" xmlns:sad="clr-
namespace:System.Activities.Debugger; assembly=System.Activities"
xmlns:sad1="http://schemas.microsoft.com/netfx/2009/xaml/activities/design"
xmlns:scg="clr-namespace:System.Collections.Generic;assembly=System"
xmlns:scg1="clr-namespace:System.Collections.Generic;assembly=System.ServiceModel"
xmlns:scg2="clr-namespace:System.Collections.Generic;assembly=System.Core"
xmlns:scg3="clr-namespace:System.Collections.Generic;assembly=mscorlib"
xmlns:sd="clr-namespace:System.Data;assembly=System.Data" xmlns:sd1="clr-
namespace:System.Data;assembly=System.Data.DataSetExtensions" xmlns:sl="clr-
namespace:System.Linq;assembly=System.Core" xmlns:st="clr-
namespace:System.Text;assembly=mscorlib"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <WorkflowServiceImplementation ConfigurationName="MathService"
Name="MathService">
    <p:Sequence DisplayName="Basic Math
Service" sad:XamlDebuggerXmlReader.FileName="C:\My Dot Net
World\Projects\2010\WCF\AdventureWFServices\AdventureWFServices\MathService.xa
mlx" sad1:VirtualizedContainerService.HintSize="303,348.553333333333"
mva:VisualBasic.Settings="Assembly references and imported namespaces serialized as
XML namespaces">
      <p:Sequence.Variables>
        <p:Variable x:TypeArguments="CorrelationHandle" Name="handle" />
        <p:Variable x:TypeArguments="x:Int32" Name="GlobalX" />
        <p:Variable x:TypeArguments="x:Int32" Name="GlobalY" />
      </p:Sequence.Variables>
      <sad1:WorkflowViewStateService.ViewState>
        <scg3:Dictionary x:TypeArguments="x:String, x:Object">
          <x:Boolean x:Key="IsExpanded">True</x:Boolean>
        </scg3:Dictionary>
      </sad1:WorkflowViewStateService.ViewState>
      <Receive x:Name="__ReferenceID0" CanCreateInstance="True"
DisplayName="Sum" sad1:VirtualizedContainerService.HintSize="257,85.27666666666
```

```
67" OperationName="Sum" ServiceContractName="p1:IMathService">
      <Receive.CorrelatesOn>
        <MessageQuerySet />
      </Receive.CorrelatesOn>
      <Receive.CorrelationInitializers>
        <RequestReplyCorrelationInitializer CorrelationHandle="[handle]" />
      </Receive.CorrelationInitializers>
      <Receive.KnownTypes>
        <x:Type Type="x:Int32" />
      </Receive.KnownTypes>
      <ReceiveParametersContent>
        <p:OutArgument x:TypeArguments="x:Int32"
x:Key="X">[GlobalX]</p:OutArgument>
        <p:OutArgument x:TypeArguments="x:Int32"
x:Key="Y">[GlobalY]</p:OutArgument>
      </ReceiveParametersContent>
    </Receive>
    <SendReply Request="{x:Reference __ReferenceID0}" DisplayName="Sum
Response" sad1:VirtualizedContainerService.HintSize="257,85.2766666666667">
      <SendParametersContent>
        <p:InArgument x:TypeArguments="x:Int32" x:Key="Result">[GlobalX +
GlobalY]</p:InArgument>
      </SendParametersContent>
    </SendReply>
  </p:Sequence>
 </WorkflowServiceImplementation>
</Service>
```

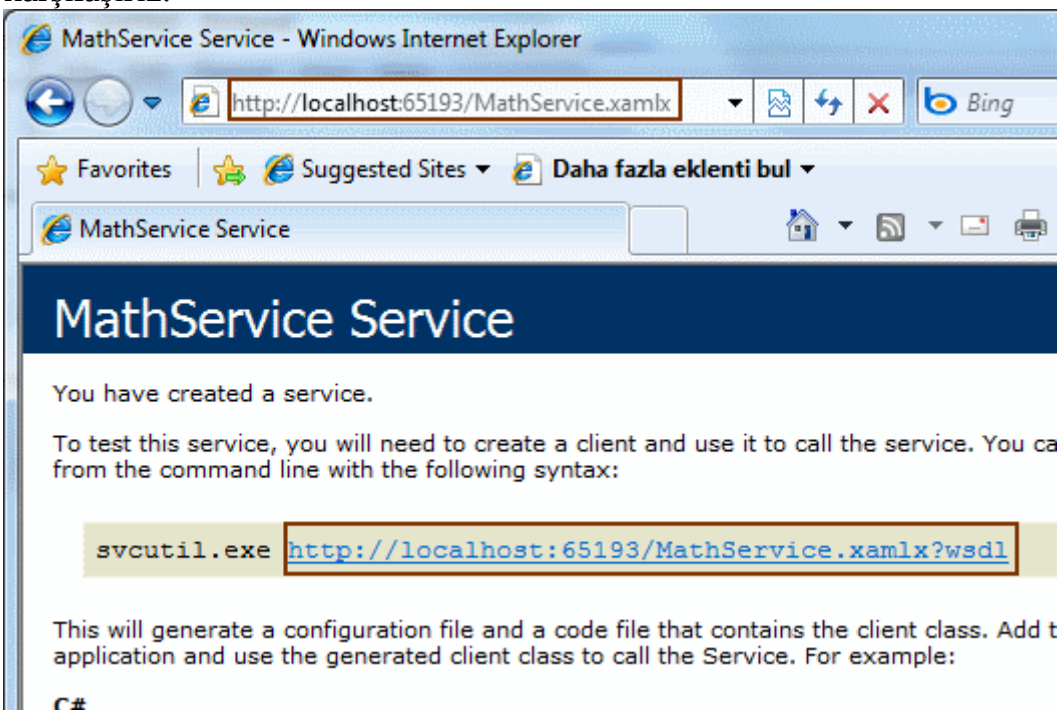
İlk etapta biraz korkutucu görünebilir. Ancak şu an için üzerinde duracağımız önemli noktalar **Bold** olarak işaretlenmiştir. **Sequence.Variables** elementi içerisinde 3 değişken tanımı görülmektedir. Bizim eklediklerimiz **GlobalX** ve **GlobalY** isimli **int** tipinden olanlardır. Bu değişkenler **Sequence** elementi içerisinde tanımlandıklarından, takip eden **Receive** ve **SendReply**elementleri veya gelecek başka aktiviteler tarafından ortaklaşa kullanılabilirler. **Receive** elementinde operasyon ile ilişkili bilgiler dışında, servisin sözleşme ismide belirtilmektedir. **Receive** aktivitesi için en önemli parça **ReceiveParametersContent** kısmıdır. Burada **X** ve **Y** isimli **OutArgument** tipinden argümanlar tanımlamıştır. Bu argümanlarda önemli olan kısım **GlobalX** ve**GlobalY** atamalarıdır. **X** ve **Y** aslında istemcinin çağıracağı servis operasyonunun parametreleridir. İstemciden gelen bu değerler, aktivitenin **GlobalX** ve **GlobalY** değerlerine set edilmektedir.**SendReply** aktivitesi içerisinde yer alan **SendParametersContent** kısmında ise **InArgument** tipinden bir argüman tanımlanmıştır. Result ismi ile tanımlanan bu argümanın içeriği, GlobalX ve GlobalY değerlerinin toplamından oluşmaktadır. Bir başka deyişle istemci tarafına döndürelecek cevap içeriği belirlenmiş olur. İçeriğin tipi ise **TypeArguments** niteliğine atanan**Int32** yapısıdır. Görüldüğü üzere herhangibir kod dosyası kullanılmamıştır. Tüm

işlemler XAML bazlı olaraktan tanımlanabilmektedir. Buda çalışma zamanı için önemli bir esnekliktir.

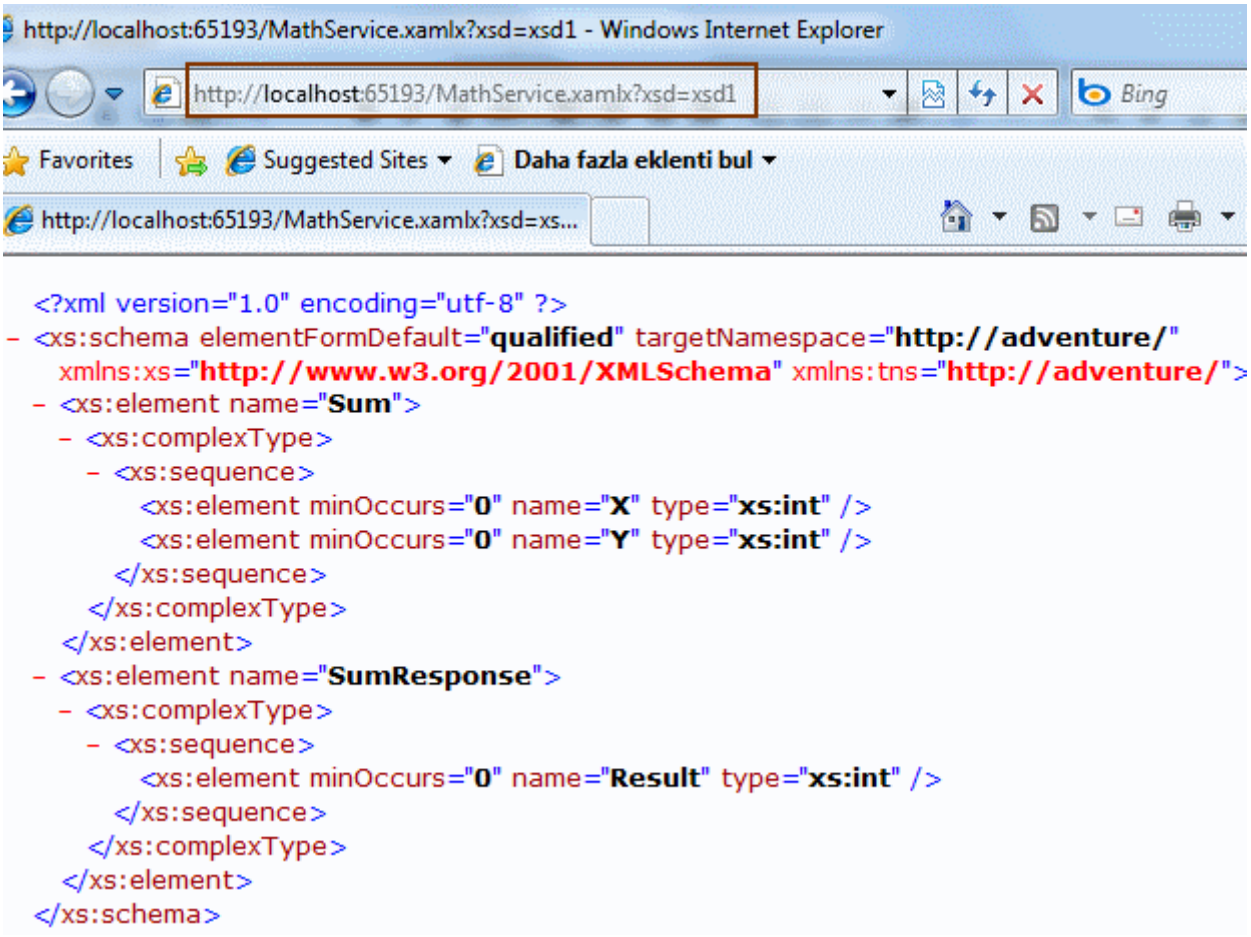
Oluşturduğumuz sınıf kütüphanesinin Web.config dosyasının içeriği ise son derece sadedir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.web>
    <compilation debug="true" targetFramework="4.0" />
  </system.web>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceDebug includeExceptionDetailInFaults="False" />
          <serviceMetadata httpGetEnabled="True"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

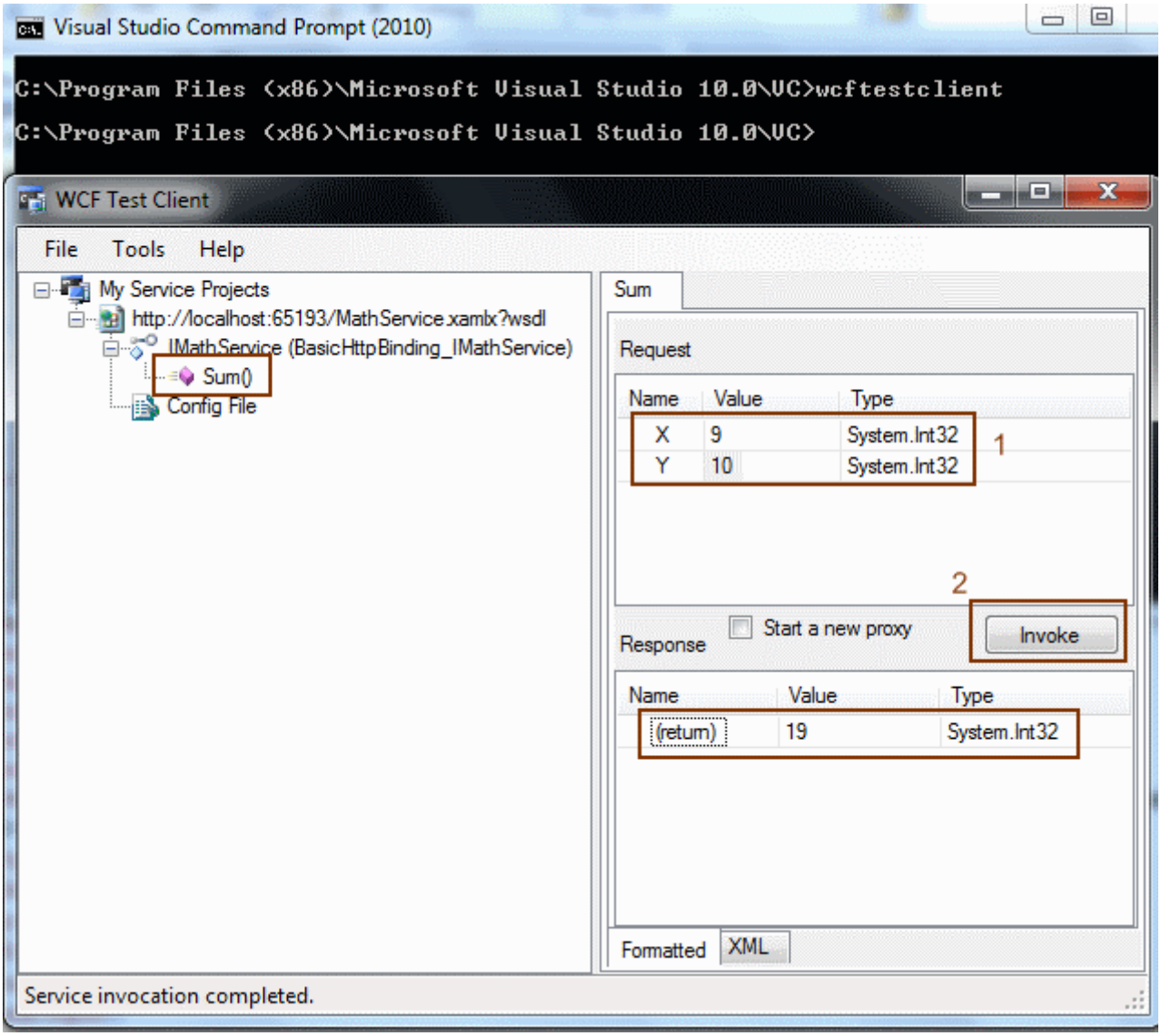
Görüldüğü gibi belirli bir **Endpoint** tanımlaması yoktur. Ancak **exception** detayının gönderilmesi ve **Metadata** bilgisinin elde edilebilmesi için gerekli **davranışlar(Behaviors)** tanımlanmıştır. Buna göre **Workflow Servisimiz**, **HTTP** bazlı olaraktan host edilecektir. Uygulamayı **F5** ile başlattığımızda ve **MathService.xamlx** içeriğini talep ettiğimizde aşağıdaki ekran görüntüsü ile karşılaşırız.



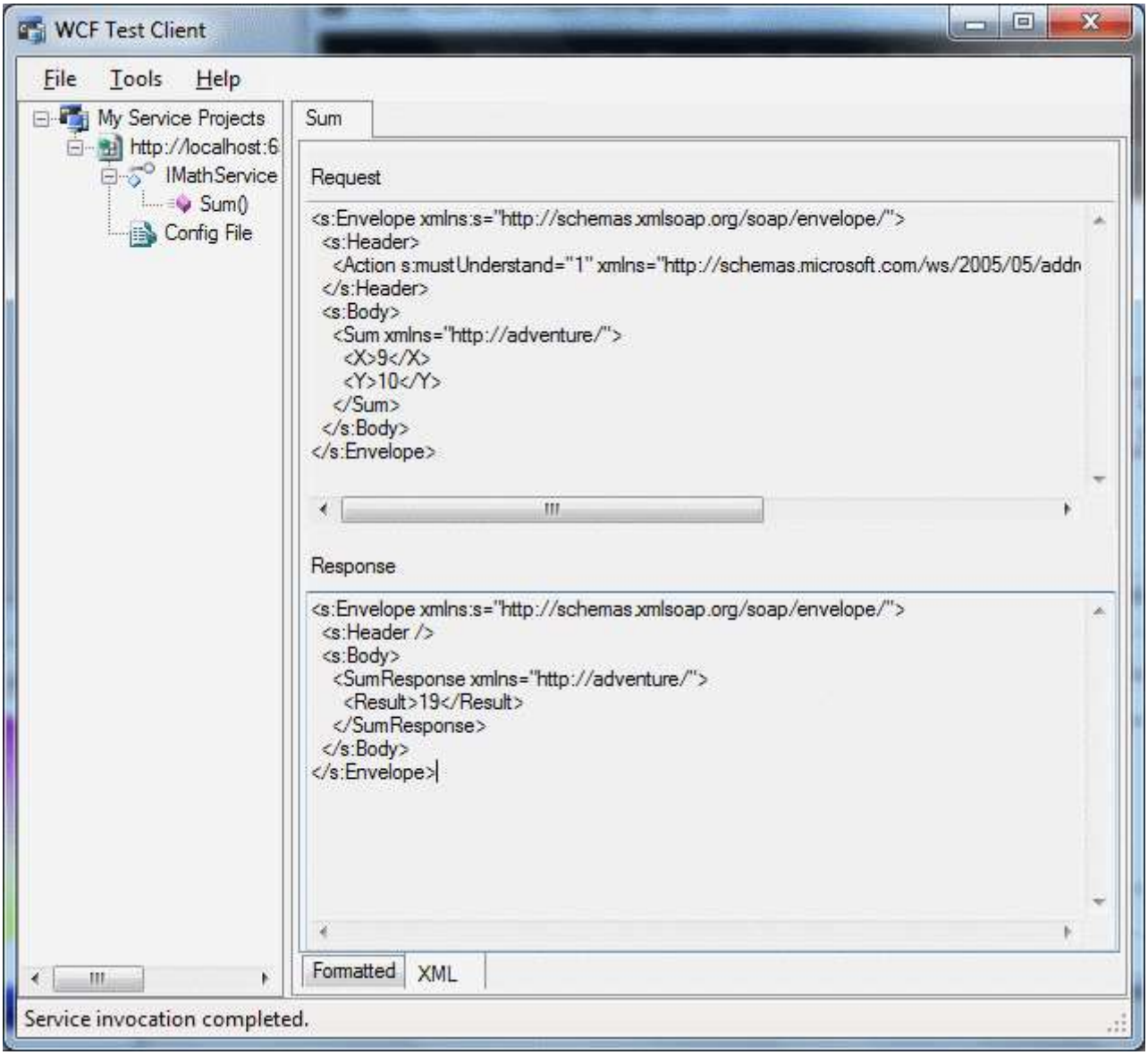
Görüldüğü üzere **WSDL** içeriğide doğrudan talep edilebilmektedir. Buda istemciler için gerekli **Proxy** üretiminin kolayca yapılabileceği anlamına gelmektedir. Burada WSDL içeriğinide inceleyebiliriz. Ancak benim özellikle göstermek istediğim nokta **http://localhost:65193/MathService.xaml?xsd=xsd1** talebinin sonucudur. Nitekim **WSDL** dökümanına baktığımızda, **Sum**isimli operasyonun kullandığı **X** ve **Y** parametrelerini göremeyiz. Ancak bu parametrelerin tanımlandığı **XSD** içeriğine bir referans bildirildiğini fark edebiliriz. Bu **XSD** içeriğinde aşağıdaki ekran görüntüsünde olduğu gibi, **X,Y** ile istemci tarafına döndürelecek Result değişkenlerine ait tanımlamalar yer almaktadır.



Dilerseniz artık servisimizi test edelim. Bu amaçla daha önceden de belirttiğimiz gibi WcfTestClient aracını kullanabiliriz. İşte örnek bir toplama işlemi;



İlk olarak Request için X ve Y değerleri girilir sonrasında Invoke düğmesine tıklanır. Bir süre beklemenin ardından servisten geri dönüş elde edilir. BasicHttpBinding tabanlı olarak host edilen Workflow Servisimiz başarılı bir şekilde çalıştırılmıştır. Yapılan Invoke işlemi sonrası üretilen XML içeriğine bakıldığında ise aşağıdaki çıktının oluştuğu görülebilir.



İşte bu kadar. **Workflow Servisleri** sayesinde bir iş akışının servis bazlı olarak başlatılabilmesi ve **WF çalışma zamanının** nimetlerinden yararlanabilmesi mümkün olmaktadır. üstelik, **.Net 4.0** tarafında gelen yeni modele göre söz konusu **Workflow Servislerinin** tamamen **XAML** içeriğinden oluşacak şekilde **dekleratif** olarak tanımlanabilmesi söz konusudur. Bu da çalışma zamanında koda müdahale etmeye gerek duymadan akış ile ilişkili bazı değişikliklerin yapılabileceğinin bir göstergesidir. Bakalım WCF&WF kardeşliğinde başka ne gibi yenilikler bizleri bekliyor. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

AdventureWFServices.rar (442,66 kb)

WF 4.0 - Kod Yoluyla Workflow Service Oluşturmak, Kullanmak [Beta 1] (2009-10-16T09:00:00)

wf,wf 4.0,wcf,wcf 4.0,

Merhaba Arkadaşlar,

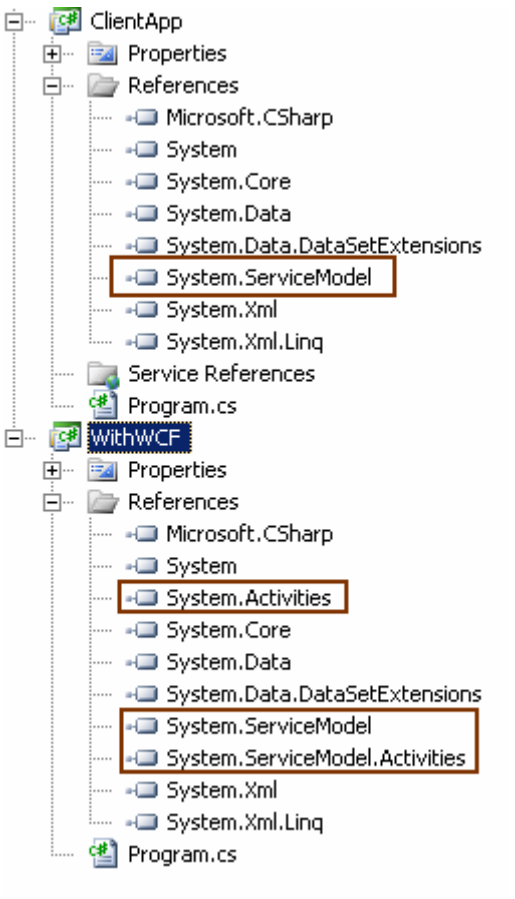
Yükseklik korkum olmasına rağmen her zaman yandaki gibi tırmanışta olanlara imrenmişimdir. Bu fotoğrafa konu olan kişinin tek yaptığı yoğun bir mücadele ve efor ile yukarı doğru tırmanmaktır. Bana göre sonuçta elde edilebilecek tek şey zirveye ulaşmak ve oranın eşsiz manzarasını izlemekten ibarettir. Tabi bunun birde inişi olduğunu düşünmek gerekiyor Hangi açıdan bakarsak bakalım bizde hayatımızda zaman zaman böyle mücadeleler içerisine gireriz. özellike yazılım geliştirirken

örneğin her zaman elimizin altında **Visual Studio IDE'** sinin sunduğu gibi gelişmiş arayüzler bulunmayabilir. örneğin **Visual Studio 2010 Beta 1** üzerinde yaşadığım sorunlardan birisi **WPF** tabanlı **Designer'** ı **Workflow** uygulamaları için kullanamıyor oluşumdu. Bu gerçekten çok üzücü bir durum. Ama çaresiz değiliz. çaresizliğin çözümü bazı işlemleri basit bir **Console** uygulamasında, gereklilikleri fark ederek*(örneğin hangi**Assembly**' ların referans edilmesinin gerektiğinin bilinmesi...),* kod bazında yapmaktan ibarettir. Bunun bize sağlayacağı pek çok fayda bulunmaktadır. Kod tarafında her ne kadar mücadeleci bir yol izlesekte, neyin nasıl oluşturulması gerektiğini, hangi durumlarda ne gibi istisnalara(Exceptions) düşebileceğimizi, nelere ihtiyaç duyduğumuzu ve arka planda aslında işlemlerin nasıl değerlendirildiğini görmek açısından yararlı bir yoldur. Bu kadar cümleyi elbetteki sizi yazının kalanına motive etmek için sarfettiğimi düşünebilirsiniz. Haydi gelin kamera arkasına bakalım.

***Kişisel Not :** Aslında **Beta 2** sürümünde(ki henüz public olarak yayınlanmadığını biliyorsunuz), **designer** tarafındaki sorunların aşıldığını ifade edebilirim. Bizzat tecrübe ile sabitlenmiştir Hatta bu konu ile ilişkili bir yazımı söz konusu sürüm public hale geldikten sonra yayınlıyor olacağım.*

Bu yazımızda **.Net Framework 4.0 Beta 1** ile bir **Workflow Service'** in oluşturulması, host edilmesi ve bir istemci tarafından kullanılması konusu irdelenmeye

çalışılacaktır. **Workflow Service** tek yönlü bir operasyon**(One Way)** için hizmet vermekte olup istemci tarafına bir geri bildirimde bulunmamaktadır. Her iki uygulamada birer **Console Application** olarak tasarlanmıştır. İstemci tarafında **Workflow Service'** in kullanılabilmesi için gerekli **Proxy** nesnesi yine kod yardımıyla*(WSDL dökümanından yararlanmadan)* oluşturulmaktadır. Aslında tamamlanmış olan uygulamalara baktığımızda servis ve istemci tarafı için gerekli olan referans **Assembly'** ların aşağıdaki şekilde görüldüğü gibi olduğunu fark edebiliriz.



Tamam çok güzel ama biz bu yazımızda tam olarak neyi hedeflemekteyiz?

Yapmak istediğimiz ilk şey **WF 4.0** bazlı olarak bir **Workflow Service** geliştirmek olacaktır. Bilindiği üzere **Workflow Service'** ler, istemcilerin uzaktan erişerek başlatabileceği akışları içerebilecek hizmetler olarak düşünülebilir. öyleyse **Workflow Service'** in istemciden gelecek talepleri alabilmesi, gerektiğinde istemciye dönüş yapabilmesi ve kendi içerisinde bir akışı barındırması gerekmektedir. Tahmin edileceği üzere istemci üzerinden gelecek taleplerin alınması veya cevapların gönderilmesi sırasında hazır aktivite bileşenlerinden yararlanılır. örneğin **Receive** veya **SendReply**...Elbette bu tip bir servisin geliştirilmesi yeterli değildir. Bu servisin istemcilere hizmet verebilmesi için ayrıca host edilmesi de gerekmektedir. İstemci tarafı ise standart bir **Console, WinForms, WPF, Asp.Net** ucu olabileceği gibi başka bir **Workflow Service** de olabilir. Biz işe ilk olarak **Workflow Service** tarafını geliştirerek başlayacağız. Bu

amaçla**WithWCF** isimli **Console** uygulamamızın kodlarını aşağıdaki gibi geliştirdiğimizi düşünebiliriz.

Workflow Service tarafı kodları;

```
using System;
using System.Activities;
using System.Activities.Statements;
using System.ServiceModel;
using System.ServiceModel.Activities;
using System.Xml.Linq;

namespace WithWCF
{
   class Program
   {
      static void Main(string[] args)
      {
         // örnekte yer alan akış tek-yönlü bir Workflow Service' idir. Sequence tipinden olan bu akış, Receive aktivitesi ile başlamakta olup istemciden gelen double tipinden değişkenin karekökünü hesaplamakta ama istemci tarafına bir bilgilendirmede bulunmamaktadır.

         #region Sequence Aktivitesi Oluşturulma İşlemleri

         // Sequence tipinden bir aktivite örneklenir
         Sequence squareRootFlow = new Sequence();

         // Aktivitede kullanılacak olan Variable tanımlamaları yapılır
         Variable<double> number = new Variable<double>(); // Receive aktivitesi tarafından alınan değişken
         Variable<double> square = new Variable<double>(); // Sonuç değişkeni
         Variable<CorrelationHandle> corHandle = new Variable<CorrelationHandle>();
         // Xml Namespace tanımlaması yapılır. XNamespace kullanılmadığı takdirde örneğin Receive aktitivesinin ServiceContractName özelliğine adres bilgisi text tabanlı olarak atandığında şekilde görülen exception üretilir.
         XNamespace ns = "http://www.buraksenyurt.com/WF4";

         // Variable tanımlamaları Sequence aktivitesinin Variables koleksiyonuna dahil edilir.
         squareRootFlow.Variables.Add(number);
         squareRootFlow.Variables.Add(square);
         squareRootFlow.Variables.Add(corHandle);
```

```
        // Akışın içerisinde yer alan ilk aktivite Receive tipindendir.
        Receive receive1 = new Receive
        {
            OperationName="SquareRoot",
            DisplayName="Square Root Calculation",
            ServiceContractName = ns + "CalculatorService",
            CanCreateInstance=true,
            Value=new OutArgument<double>(number),
            AdditionalCorrelations={
                {"ChannelBasedCorellation",new
InArgument<CorrelationHandle>(corHandle)}
             }
        };

        // Receive aktivitesi ile istemciden gelen sayısal değer number değişkenine
alındıktan sonra çalışan InvokeMethod aktivitesi ile Calculator isimli sınıf içerisindeki
FindSquareRoot metodu çağırılır.
        InvokeMethod<double> invokeMethod1 = new InvokeMethod<double>
        {
            TargetType=typeof(Calculator),
            MethodName = "FindSquareRoot",
            Result=new OutArgument<double>(square) // FindSquareRoot metodunun
çalıştırılması sonucu elde edilen sonuç square isimli değişken tarafında yakalanabilecektir.
        };
        // Metoda parametre olarak number değişkeninin değeri gönderilir. Bu değer
Receive aktivitesi içerisinde set edilmiş olup istemci tarafından gelmektedir.
        invokeMethod1.Parameters.Add(new InArgument<double>(number));

        // Ekrana bilgilendirme yapılır. Bu bilgilendirmede istemciden akışa gelen sayının
karekökü yazdırılmaktadır.
        WriteLine writeLine1 = new WriteLine
        {
           Text=new InArgument<string>(e=>String.Format("Sonuç
{0}",square.Get(e).ToString()))
        };

        // Aktivitelere sırasıyla Sequence aktivitesi içerisine ilave edilir
        squareRootFlow.Activities.Add(receive1);
        squareRootFlow.Activities.Add(invokeMethod1);
        squareRootFlow.Activities.Add(writeLine1);

        #endregion

        #region Workflow Servis Oluşturma İşlemleri
```

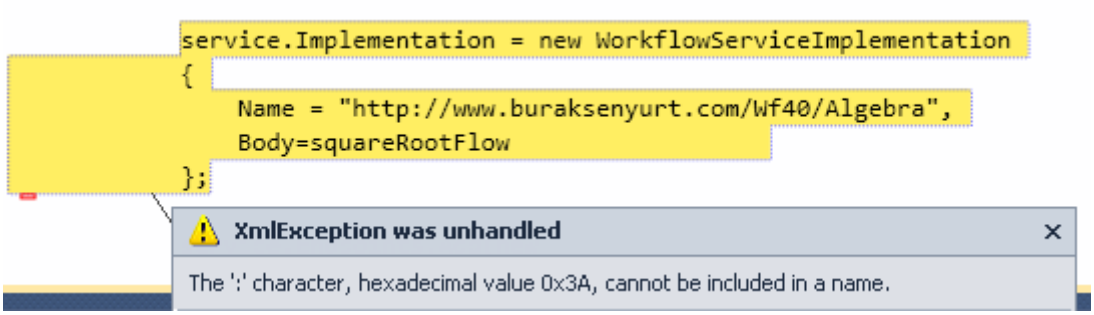
```
        // Workflow' un Service olarak host edilmesi için gerekli hazırlıklar başlar.
        // İlk olarak bir Service nesne örneği oluşturulur
        Service service = new Service();

        // Service nesne örneğinin Implementation özelliğine
WorkflowServiceImplementation türünden bir referans atanırken, Body özelliğine yukarıda
oluşturulan Workflow aktivitesi bildirilir
        service.Implementation = new WorkflowServiceImplementation
        {
           Name = ns+"CalculatorService",
           Body=squareRootFlow
        };

        // Sonuçta Workflow bir WCF servisi olarak host edileceğinden bir Endpoint
bilgisine sahip olmalıdır
        // Bu nedenle basit bir Endpoint bildirimi yapılır
        // örnekte Tcp bazlı servis iletişimi tercih edilmiştir
        service.Endpoints.Add(new Endpoint
        {
           Uri = new Uri("SquareRoot", UriKind.Relative), //Address bilgisi
           Binding=new NetTcpBinding(), // Binding bilgisi
           ServiceContractName = ns + "CalculatorService" // Contract bilgisi(Burada
verilen isim ile Receive aktivitesine ait ServiceContractName özelliğindeki değerler aynı
olmalıdır. Aksi halde Exception2 alınır)
        });

        #endregion

        // Workflow Service' i çalıştıracak olan WorkflowServiceHost nesnesi örneklenir
        // İlk parametre host edilecek olan servistir. İkinci parametre ise servisin host
edileceği adres bilgisidir.
        WorkflowServiceHost host = new WorkflowServiceHost(service, new
Uri("net.tcp://localhost:4501/Calculator/Workflows/"));
        host.Open(); // Host açılır

        Console.WriteLine(host.State.ToString()); //Hostun durumu hakkında bilgilendirme
yapılır
        Console.WriteLine("Kapatmak için bir tuşa basınız.");
        Console.ReadLine();

        host.Close(); //Host kapatılır
    }
  }
```

```
    class Calculator
    {
        public double FindSquareRoot(double number)
        {
            return Math.Sqrt(number);
        }
    }
}
```

Kod içerisinde işleyiş ile ilişkili gerekli açıklamalar bulunmaktadır.

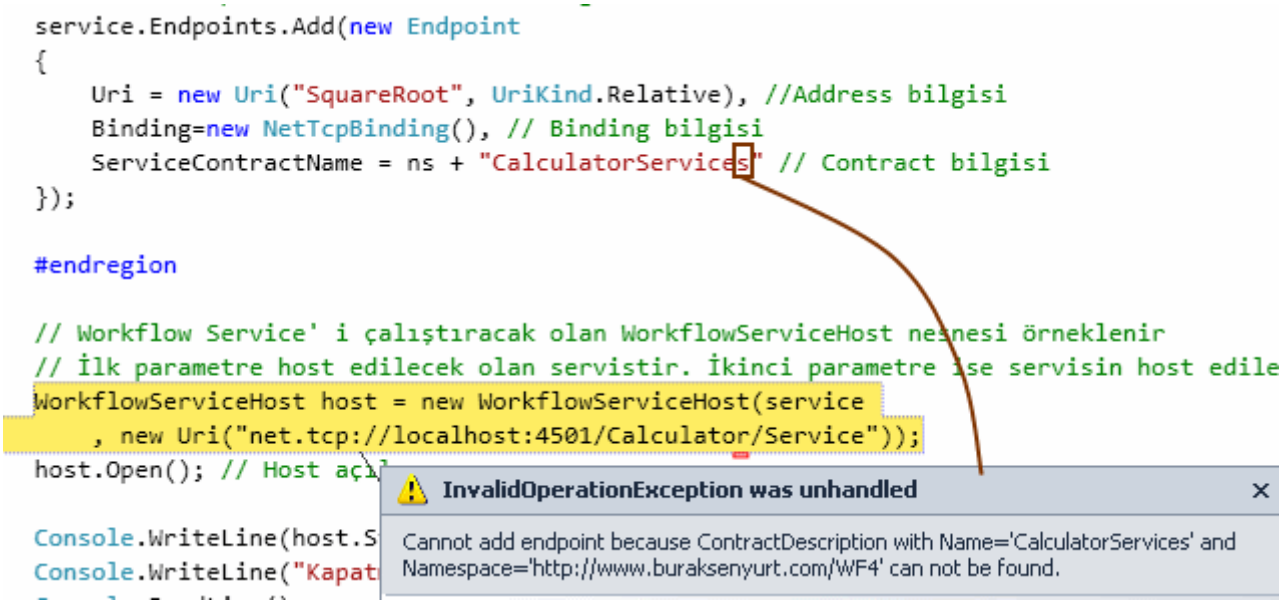
Ancak geliştirme sırasında dikkat edilmesi gereken bazı durumlar da vardır. örneğin **XNamespace** tipinin kullanılmaması, bunun yerine **URL** bilgisinin **text** tabanlı olarak atanması halinde çalışma zamanında aşağıdaki ekran görüntüsünde yer alan **XmlException** istisnası alınacaktır.

Exception 1 (Xml Namespace kullanılmaması halinde);



Diğer yandan **ServiceContractName** değerinin hem **Endpoint** hemde **Receive** aktivitesi için aynı olması gerekmektedir ki aksi durumda aşağıdaki ekran görüntüsünde yer alan çalışma zamanı istisnası fırlatılmaktadır.

Exception 2 (ServiceContractName' lerin Receive aktivitesi ve Endpoint içerisinde farklı olmaları halinde);



Gelelim istemci tarafına. Bu noktada kendimizi yine zora sokuyor olacağız. Elimizde servisin **Publish** edilen bir **WSDL** dökümanı olmadığını ve herhangibir şekilde **Proxy** üretimi için bir araç kullanmadığımızı farz edeceğiz. Bu durumda istemci tarafındaki **proxy** sınıfının ve hatta istemciden servis tarafına gönderilecek olan mesaj sözleşmesinin manuel kod ile oluşturulması gerekmektedir. Aynen aşağıda olduğu gibi;

İstemci tarafı sınıf diagramı;

ve kodları;

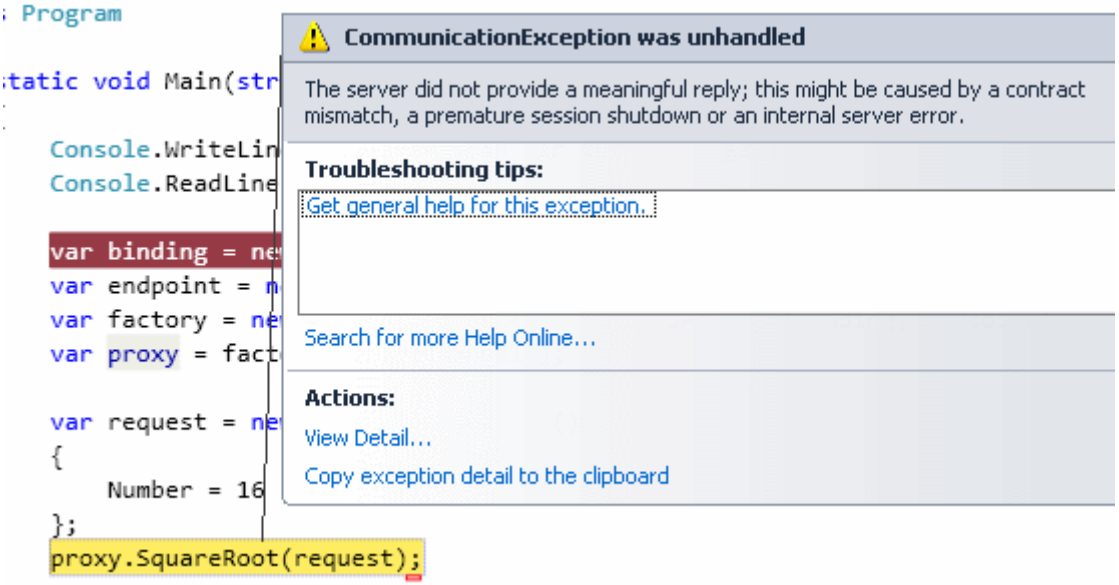
```
using System;
using System.ServiceModel;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
```

```
        {
            Console.WriteLine("Başlamak için bir tuşa basınız");
            Console.ReadLine();

            // Binding oluşturulur
            NetTcpBinding binding = new NetTcpBinding();
            // Endpoint tanımlaması yapılır. Adres bilgisinin servisin yeri ve çağırılmak istenen operasyonun adıo yer almaktadır. Bu bilgileri service tarafındakiler ile aynı olmalıdır
            EndpointAddress endpoint = new EndpointAddress("net.tcp://localhost:4501/Calculator/Workflows/SquareRoot");
            // Kanal fabrikası üretilir. İlk parametre bağlayıcı ikinci parametre ise EndPoint bilgisidir
            ChannelFactory<CalculatorService> factory = new ChannelFactory<CalculatorService>(binding, endpoint);
            // Kanal fabrikasından yararlanılarak istemcinin kullanacağı Transparant Proxy nesnesi üretilir.
            CalculatorService proxy = factory.CreateChannel();

            // Talep oluşturulur ve parametre olarak servis tarafındaki akışın ilk aktivitesi olan Receive aktivitesinin alacağı sayısal değer bildirilir
            SquareRootRequest request = new SquareRootRequest()
            {
               Number = 16
            };
            // Operasyon çağırılır
            proxy.SquareRoot(request);

            Console.WriteLine("İşlemler tamamlandı. Kapatmak için bir tuşa basınız.");
            Console.ReadLine();
        }
    }

    // Talep olarak gidecek mesaj sözleşmesi tanımlanır
    [MessageContract(IsWrapped = false)]
    public class SquareRootRequest
    {
       [MessageBodyMember(Namespace= "http://schemas.microsoft.com/2003/10/Serialization/",Name = "double")]
        public double Number { get; set; }
    }

    // Proxy tanımlaması yapılır
    // Workflow Servisi varsayılan isim alanı(namespace) ile sunulmadığından Namespace bildiriminin istemci tarafındaki proxy için açık bir şekilde yapılması gerekmektedir.
    [ServiceContract(Namespace = "http://www.buraksenyurt.com/WF4")]
```

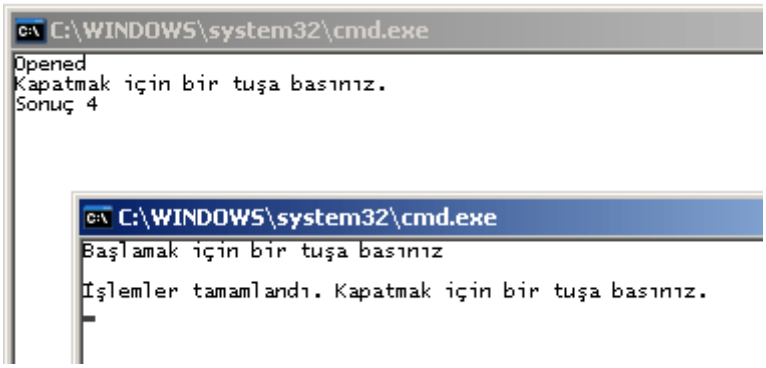
```
    interface CalculatorService
    {
        [OperationContract(IsOneWay=true)] // Operasyonun OneWay olduğunu belirtmessek Exception3' teki çalışma zamanı hatasını alırız.
        void SquareRoot(SquareRootRequest request);
    }
}
```

Yine istemci tarafı açısından olaya baktığımızda dikkat etmemiz gereken bazı hususlar olduğu ortadadır. örneğin, **Workflow Service** içerisinden geriye bir dönüş yapılmamaktadır. Bir başka deyişle tek yönlü bir istek söz konusu olabilir. Bu nedenle istemci tarafındaki **SquareRoot** metoduna uygulanan **OperationContract** niteliğinde **IsOneWay** özelliğine **true**değeri atanmıştır. Bu yapılmadığı takdirde çalışma zamanında aşağıdaki istisna mesajı ile karşılaşılacaktır.

Exception 3 (Service operasyonunun OneWay olduğunu belirtmediğimiz durumda);



Piuuuuuvvvvv!!! Biraz uğraştık ama faydalı bir çalışma oldu sanıyorum ki. Gerçi elde edeceğimiz sonuç hiç bir anlam ifade etmesede, bir Workflow Service' in kod yardımıyla nasıl oluşturulabileceğini ve kullanılacağını görmüş olduk. İşte bir yazılımcı olarak tırmandığımız dağın zirvesindeki görüntü...



Ne kadar muhteşem değil mi? Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WithWCF.rar (42,98 kb)

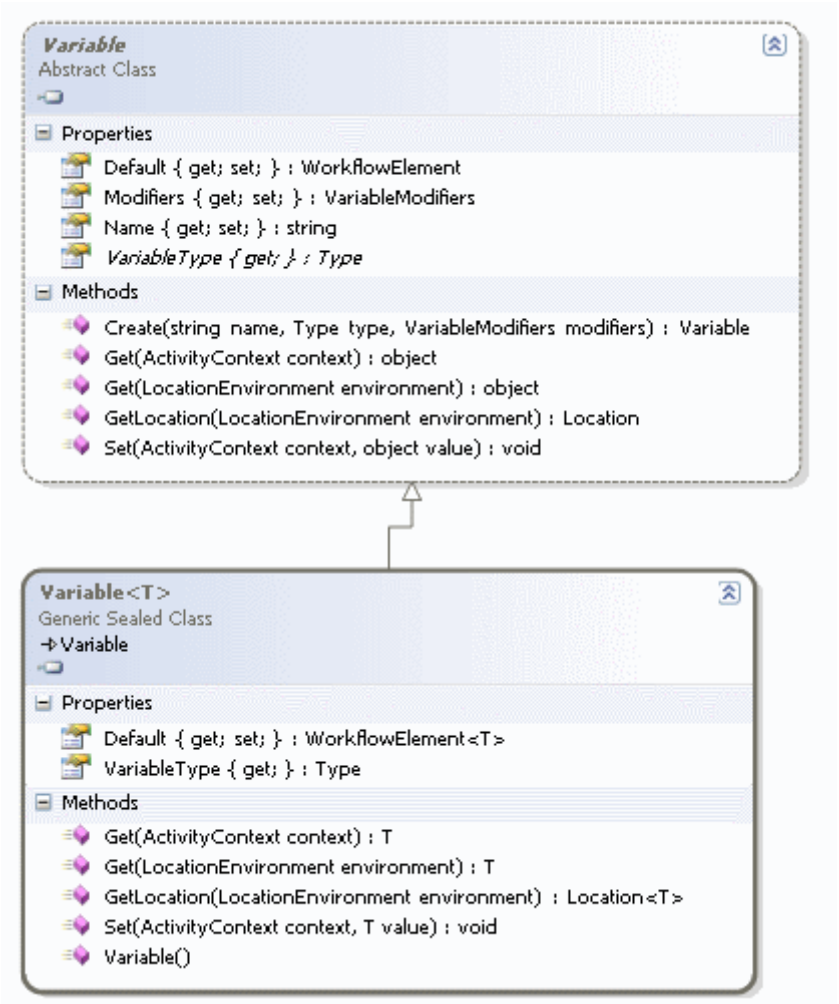
Kişisel Not : örnekler bildiğiniz üzere Visual Studio 2010 Beta 1 sürümünde ve .Net Framework Beta 1 üzerinde geliştirilmektedir. Beta 2 ile arasında farklılıklar olabilir. Hatta Relase sürümde çok daha fazla farklılık görülebilir.

WF 4.0 - Veri(Data)[Beta 1] (2009-10-12T23:48:00)

wf,wf 4.0,

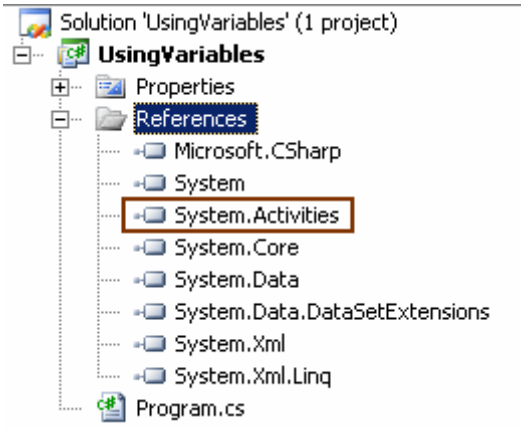
Merhaba Arkadaşlar,

Bir süredir **Workflow Foundation 4.0** ile ilişkili blog yazılarını, makaleleri ve görsel dersleri takip etmekteyim. Bu araştırmalarım sırasında **Workflow Foundation 4.0** modelinde **veriye(Data)**olan bakış açısının **WF 3.X** sürümüne göre oldukça farklılaştığını gördüm. **WF 3.X** tabanlı modelde, aktivite bazlı verileri temsil etmek için genellikle standart sınıf**özelliklerinden(Property)** veya **WPF'** ten esinlenilen **bağımlı özelliklerden(Dependency Property)** yararlanılmaktadır. **WF 4.0** modelinde ise veriyi temsil etmek amacıyla **Variable** veya **Argument**türevli tiplerden yararlanıldığı görülmektedir. Bu tiplerin türevleri veriyi doğrudan tutmazlar. Bunun yerine veriyi tanımlar ve elde edilmesini sağlarlar. Verinin içeriği **Workflow** üzerinde bir yerlerde saklanmaktadır. Bu noktada **Variable** kavramını aynen **Imperative** programlama dillerindeki kullanılış biçimi ile düşünebiliriz. Bu nedenle **Variable** veya **Argument** türevli tipler tanımlandıkları scope dahilinde kullanılabilirler. Dolayısıyla bir **Variable** bir Workflow için **kök seviyede(Root Level)** tanımlanırsa tüm alt aktiviteler tarafından kullanılabilir. Bugünkü örneğimizde bir önceki blog yazımızda olduğu gibi kod bazlı bir **Workflow** örneği geliştirecek ve **Variable** kullanımı ile veriyi nasıl ele alacağımızı görmeye çalışacağız. Aslında .Net 4.0 içerisindeki oluşuma baktığımızda **Variable<T>** için **Variable** isimli bir abstract sınıftan türetme yapıldığını görebiliriz. Aşağıdaki sınıf diyagramında bu durum görülmektedir.



Burada dikkat edilmesi gereken noktalardan birisi, **Variable<T>** tipinin **sealed** olarak tanımlanmış olmasıdır. Yani kendisinden türetme yapamayız. Diğer yandan **Variable** tipi **abstract** bir sınıftır ve bu nedenle kendisinden türetme yapılarak özel **Variable** türevlerinin üretilmesi mümkündür.

Evet bu kadar laf kalabalığından sonra yeni veri modeline kısaca bakmaya çalışalım. Bu amaçla **Visual Studio 2010 Beta 1** üzerinden açtığımız basit bir **Console** projesini kullanıyor olacağız. Projede **Workflow** alt yapısını ele almak için tek yapmamız gereken **System.Activities** isimli **assembly'** ın referans edilmesi olacaktır.



Uygulama kodlarını ise aşağıdaki gibi geliştirdiğimizi düşünebiliriz.

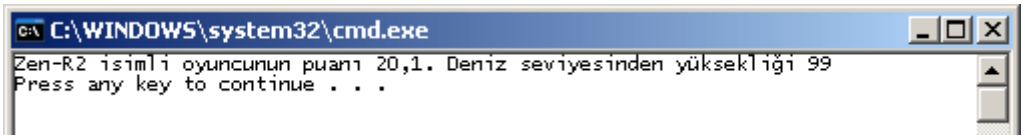
```
using System;
using System.Activities;
using System.Activities.Statements;

namespace UsingVariables
{
   class Program
   {
      static void Main(string[] args)
      {
         // Değişken tanımlanır. Tanımlanırken varsayılan olarak(Default) bir nesne örneğide verilir
         Variable<Player> firstPlayer = new Variable<Player>
         {
            Name = "FirstPlayer",
            Default = new Player { Name = "Zen-R2", Location = new Location { X = 10, Y = 12, Altitude = 100 }, PlayerType = PlayerType.Computer, TotalPoint=10 }
         };

         // Bir Sequence aktivitesi tanımlanır. Root aktivite.
         Sequence gameFlow = new Sequence();
         // Variable Sequence nesne örneğinin Variables koleksiyonuna eklenir.
         gameFlow.Variables.Add(firstPlayer);

         // Atama işlemi için basit bir Assign aktivitesi gameFlow isimli Sequence nesne örneğinin Activities koleksiyonuna eklenir.
         // To kısmında kime atama yapılacağı belirlenir. Tasarım zamanından farklı olarak bir expression kullanılır. firstPlayer isimli değişkenin işaret ettiği veri alanında TotalPoint değeri alınır.
         // Value özelliğine verilen değer ile atama yapılır. Atamada o anki variable' ın TotalPoint değeri 10.1 birim arttırılmaktadır.
```

```
            gameFlow.Activities.Add(
               new Assign<double>() {
                   To=new OutArgument<double>(v=>firstPlayer.Get(v).TotalPoint),
                   Value=new InArgument<double>(v=>firstPlayer.Get(v).TotalPoint+10.1)
               }
               );

            // örnek olarak birde Location özelliğinin işaret ettiği içerik değiştirilmektedir.
            gameFlow.Activities.Add(
               new Assign<Location>()
               {
                  To = new OutArgument<Location>(v => firstPlayer.Get(v).Location),
                  Value = new Location { X = 10, Y = 15, Altitude = 99 }
               }
               );

            // Ekrana bilgi yazdırmak için WriteLine tipinden bir aktivite daha eklenir
            // Text özelliğinde ekrana bilgi yazdırabilmek için InArgument nesne örneğinden yararlanılır ve o anki firstPlayer variable' ının Name ve TotalPoint değerleri yazdırılır.
            gameFlow.Activities.Add(
               new WriteLine {
                   Text=new InArgument<string>(v=>String.Format("{0} isimli oyuncunun puanı {1}. Deniz seviyesinden yüksekliği {2}",firstPlayer.Get(v).Name,firstPlayer.Get(v).TotalPoint.ToString(),firstPlayer.Get(v).Location.Altitude))
               }
               );

            // gameFlow isimli Workflow çağırılır ve başlatılır
            WorkflowInvoker.Invoke(gameFlow);
        }
    }

    class Player
    {
        public string Name { get; set; }
        public Location Location { get; set; }
        public PlayerType PlayerType { get; set; }
        public double TotalPoint { get; set; }
    }

    class Location
    {
        public double X { get; set; }
        public double Y { get; set; }
```

```
        public double Altitude { get; set; }
    }

    enum PlayerType
    {
        Computer,
        Human,
        Hybrid
    }
}
```

örneğin herhangibir gerçek hayat işlevselliği bulunmamaktadır ancak kullanıcı tanımlı bir tipten*(örneğimizde Player isimli sınıf)* oluşturulan **Variable'** ın **Sequence** tipinden bir nesne örneği içerisinde nasıl değerlendirilebildiği açık bir şekilde görülmektedir. Aslında konsept son derece basittir. **Variable<T>** tipinden bir nesne örneği tanımlanırken **Default** özelliği ile ilk değer ataması yapılmaktadır ki zorunlu değildir. Yani **Variable'** ın içeriği akışa dışarıdan gelebilir. Sonrasında bu değişkenin kullanılmak istenen scope içerisine bildirilmesi gerekmektedir.
örnekte **gameFlow** isimli **Sequence** aktivite örneğinin **Variables** koleksiyonuna yapılan ekleme ile bu bildirim gerçekleştirilmektedir. Nitekim söz konusu değişkeninin, içeride yer alan tüm aktiviteler tarafından kullanılması istenmektedir.

Kodun ilerleyen kısımlarında sembolik olarak **firstPlayer** isimli **Variable<Player>** tipine ait özelliklerin bazılarında değişiklikler yapılmıştır. Bu işlemler için **Expression'** lardan yararlanılmaktadır. Değiştirme işlemlerinde dikkat edileceği üzere **Assign<T>** isimli aktiviteden yararlanılmaktadır. İlk Assign aktivitesine
ait **To** ve **Value** özelliklerinde **InArgument** ve **OutArgument**tiplerinden yararlanılarak verinin elde edilmesi ve değer ataması işlemleri yapılmaktadır. İkinci Assign aktivitesinde ise generic olarak Location tipi belirtildiğinden Value kısmında doğrudan yeni bir Location nesne örneğine atama yapılmaktadır. Her iki aktivite içinde dikkat edilmesi gereken noktalardan birisi, değeri alınmak istenen özelliğe erişilirken **firstPlayer** isimli değişkenin **Get**metodundan yararlanılıyor olmasıdır. Zaten kodlama sırasında **intelli-sense** özelliği devreye girmekte ve **Get** metodu sonrasında kullanılabilecek tüm **Player** tipi özellikleri gösterilmektedir. Programda son olarak **Workflow** örneğinin çalıştırılması sağlanmaktadır. Uygulamanın çalışma zamanı görüntüsü aşağıdaki gibi olacaktır.



Dikkat edileceği üzere başlangıçta 10 olan puan 10.1 birim arttırılmış ve Location tipi üzerinde tutulan Altitude değeride 1 birim azaltılmıştır.

Elbette **Workflow 4.0'** ın **WPF** tabanlı bir **IDE** kullanıyor olması göz ardı edilemez. Bir başka deyişle söz konusu **Variable** ekleme işlemleri aslında tasarım zamanında çok daha

kolay bir şekilde gerçekleştirilebilmektedir. Ancak yazıyı hazırladığım zaman diliminde kullandığım ve public olan **Visual Studio 2010 Beta 1** sürümüne ait **WF designer** ne yazıkki IDE tekrardan başlatılmasına nedenl olmaktadır. Bu nedenle şimdilik kod tarafı ile idare etmeniz gerekiyor. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

UsingVariablesV2.rar (26,44 kb)

WF 4.0 - Workflow Yapısı ve Object Initialization[Beta 1] (2009-10-05T08:29:00)

wf 4.0,

Merhaba Arkadaşlar,

Workflow Foundation 4.0 ile ilgili yenilikleri araştırdığım şu günlerde, yaptığım araştırmalar sırasında ilgimi çeken noktalardan biriside, bir **Workflow'** un kod tarafında tek bir**ifade satırı** ile oluşturulabiliyor olmasıydı. Burada **Workflow** sisteminin hiyerarşik yapısının, **Object Oriented** seviyede etkili bir kullanımının söz konusu olduğunu belirtmek isterim. Ancak konuya çekirdek bilgilerden başlayarak yaklaşmakta yarar var.

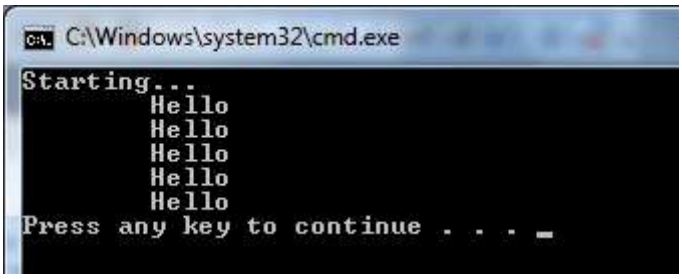
Workflow içerisindeki lego parçalarının temelini **aktiviteler(Activities)** oluşturmaktadır. **WF 4.0** mimarsinde tüm aktiviteler **WorkflowElement** bileşeninden türemektedir. Aktivitleri aslında**Workflow'** ların **iş birimleri(Work Units)** olarak düşünebiliriz. İki ve daha fazla iş biriminin bir araya gelerekten aktivite oluşturmaları da çok doğal olarak mümkündür. Aslında buradan ilginç olan bir tespit vardır. Bir aktivite, hiyerarşinin en üstünde yer alıyorsa bir **Workflow** halini alır ve kendi içerisinde pek çok aktiviteyi barındırabilir. Bunu şu şekilde de düşünebiliriz; **"Bir metodun kendi içerisinde birden fazla metodu çağırması".** Nitekim metodun kendi içerisinde çağırdığı ardışıl fonksiyonlar, yukarıdan aşağıya doğru hareket eden bir iş akışını oluşturmaktadır. Bu açıdan bakıldığında **Workflow Foundation** olmasa dahi, kod bazında iş akışlarının en temel seviyede fonksiyonlar yardımıyla gerçekleştirilebileceğini bilmek gerekir*(Hatta çalıştığım son projede bu tip bir mimari kullanılmaktadır).* **Workflow Foundation 3.X** sürümünden beridir **Sequence** gibi**üst seviye(Top Level)** aktiviteleri barındırmaktadır. Bunlar **Top Level** olarak kullanıldıklarında bir Workflow' u ifade etmektedir.
Tabiki **Sequence** örnekleri kendi içlerinde başka **Sequence**örnekleri de barındıralabilirler.

Burada ilginç olan noktalardan birisi, **Workflow'** ları sadece tek bir ifade içerisinde oluşturabilmemizdir. Bunun için **nesne başlatıcılarından(Object Initializers)** yararlanılmaktadır. Nasıl mı? Gelin **Visual Studio 2010 Beta 1** üzerinde basit bir **Console Application** oluşturup **Program.cs** içeriğini aşağıdaki gibi oluşturduğumuz düşünelim.

***Not: Console** uygulaması üzerinde **Workflow** aktivitileri kullanılacağından, projeye **System.Activities.dll Assembly'** ının referans edilmesi gerekmektedir.*

```
namespace WorkflowStructure
{
   using System;
   using System.Activities;
   using System.Activities.Statements;

   class Program
   {
      static void Main(string[] args)
      {
         // Yeni bir Sequence aktivitesi oluşturulur. Top Level olduğu için workflow' un kendisidir.
         // Tek satırlık ifade içerisinde bir Workflow tanımlandığına dikkat edilmelidir
         Sequence flow1 =new Sequence
         {
            DisplayName = "Hello Workflow World",
            // Activity tipinden elamanlar taşıyan koleksiyonun içerisinde alt aktiviteler tanımlanır
            Activities =
            {
               new WriteLine{ DisplayName="Workflow Start", Text="Starting..."},//Basit bir WriteLine aktivitesi
               new InvokeMethod{ DisplayName="5 Times Say Hello", MethodName="SayHello", TargetType=typeof(Logic)} // Logic sınıfından SayHello metodunu çağıracak olan InvokeMethod aktivitesi tanımlanır
            }
         };

         WorkflowInvoker.Invoke(flow1); // Workflow örneği çalıştırılır :)
      }
   }

   // Harici bir sınıf ve metod
   class Logic
   {
      public static void SayHello()
      {
         for (int i = 0; i < 5; i++)
         {
            Console.WriteLine("\tHello");
         }
      }
   }
}
```

Bu örnek kod parçasında **Sequence** tipinden bir nesne örneği oluşturulmaktadır. Nesne oluşturulurken, **Activities** özelliği içerisinde alt aktivite örnekleri kullanılmıştır. **Activities**özelliği **Collection<Activity>** tipinden bir koleksiyonu işaret etmektedir. Dolayısıyla herhangibir **Activity** referansına ait örnek, söz konusu koleksiyona eklenerek akışın gövdersi oluşturulabilir. örnekte ilk olarak **System.Activities.Statements** isim alanında yer alan **WriteLine** aktivite bileşeni kullanılarak ekrana basit bir çıktı verilmektedir. Takip eden adımda,**InvokeMethod** aktivite bileşeni kullanılmaktadır. Bu bileşenin özellikleri ile, **Logic** sınıfı içerisinde yer alan **SayHello** metodunun çalıştırılacağı belirtilmiştir*(Burada **TargetType**veya **TargetObject** özelliklerinden birisinin mutlaka set edilmesi gerekir. Böylece **MethodName** özelliğinde belirtilen fonskiyonun çalışma zamanında hangi tipe ait nese örneği içerisinden çağırılacağı belirtilmiş olur)* Tabiki **flow1** isimli **Sequence** nesne örneğinin çalıştırılması için **WorkflowInvoker** tipi üzerinden **static Invoke** metodu kullanılmıştır. Olaya noktalı virgüller açısından baktığımızda sadece iki satırda bir Workflow' un tasarlanıp yürütüldüğünü ifade edebiliriz. Ancak görsel bir tasarım ortamının olması elbetteki çok daha mühimdir ve tercih edilmelidir. Nitekim gerçek hayat çözümlerindeki iş akışlarının çoğu bu kadar basit Workflow örnekleri ile ifade edilememektedir. Geliştirdiğimiz**Workflow** herhangibir anlam içermesede 🙂 önemli olan, tek satırlık bir ifade ile **object initializer** kavramından da yararlanarak, bir **Workflow** örneğinin tesis edilebileceğinin farkında olmaktır. örneği yürüttüğümüzde çalışma zamanı için aşağıdaki sonuçları aldığımızı görürüz.



Görüldüğü üzere akış başarılı bir şekilde işletilmiştir. Bu kısa yazımızda **Workflow** yapısının kod tarafında **Object Initializers** yardımıyla ele alınışını incelemeye çalıştık. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WorkflowStructure.rar (19,38 kb)

5000 Feet Yüksekten Workflow Foundation 4.0[Beta 1] (2009-10-01T22:54:00)

wf 4.0,

Merhaba Arkadaşlar,

Paraşütle atlamak gerçekten zevkli olsa gerek. Yerden binlerce feet(1 feet=30,48 cm) yüksekten atlayıp özgür bir şekilde kendinizi yer çekimi gücüne bırakıp, saniyeler boyunca serbest düşüşü yaşamak...Size yandaki resimde atlayan kişinin ben olduğumu söylemek isterdim ama ne yazık ki değilim. Olmayı istermiydim bilemiyorum. Oldukça yüksek görünüyor. Bir paraşütçü için en güzel duygulardan birisi sanıyorum ki atladığı noktadan itibaren altındaki Dünyayı görebildiği kadar yüksekten izleyebilmenin verdiği mutluluktur. Tabiki atlanılan noktadan aşağıya doğru düştükçe ve paraşütü çekme noktasına yaklaştıkça alttaki Dünyanın resimlerinin daha da büyüdüğü çok açık bir gerçektir. Büyük bir dikdörten alan...Sonrasında içerisinde başka geometrik şekiller...Sonrasında bu geometrik şekiller içerisinde daha da netleşen çayırlar, dağlar, kayalar, yollar, binalar...Sonrasında bir anlık yavaşlama ve sakin bir şekilde(bazende hızlı bir şekilde) yere ayak basmak. Kısaca yaşanan bu duruma **"yüksekten görülebileceği kadar büyük bir alanı görüp, yaklaştıkça daha fazla detay fark edebilmek durumu"** adını verebiliriz. Şimdi bu noktaya nereden vardım diyebilirsiniz. Hemen açıklayayım.

Yazılım ile ilişkili pek çok kaynakta şu tip başlıklar görmüşsünüzdür. **50 bin feet yukarıdan X mimarisi**...İşte bizde bu felsefeyi bu günkü blog yazımızda kullanıyor olacağız. Ama biraz daha alçak mesafeden

5000 feet yukarıdan fotoğraflandığında, WF 4.0 modelinin ne gibi özellikleri dikkat çekiyor? İşte görülen tespitler...

- **WF 3.X**' teki kısıtlı olan **XAML** bazlı **Workflow** modeli yerini **tam desteklenen(Full XAML Based) Workflow** modeline bırakıyor. Böylece bir Workflow' un basit bir text editorü yardımıyla tamamen **dekleratif(declerative)** olarak geliştirilmesi ve çalışma zamanına devredilmesinin mümkün olabildiğini *(Bu özelliğin en önemli açılımlarından biriside, çeşitli 3ncü parti araçların **XAML** içeriklerini ele alarak akışları koda girmeden değiştirebilecek olmaları. **Oslo, Dublin** ve **Quadrant** üçlemesinin önemle üzerinde durduğu noktalardan biriside zaten bu dekleratif açılım.)*

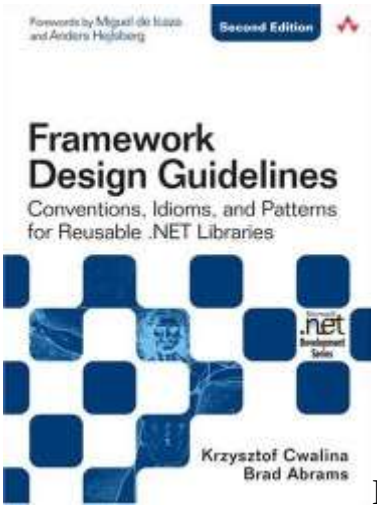
- **WF 3.X'** te daha zor olan **özel aktivite(Custom Activity)** geliştirme yeteneğinin **WF 4.0** için daha da basitleştirildiğini ve bu amaçla temel workflow hiyerarşisinde de değişikliklere gidildiğini ve ata tip olarak **WorkflowElement** ve kendisinden türeyen **Activity, CodeActivity, NativeActivity** gibi alt tiplerin geliştirildiğini,
- **WF 3.X'** te ilkel ve daha az genişletilebilir olan **kurallar motorunun(Rules Engine)** dahada zenginleştirilmiş olduğunu,
- Yeni **Workflow** bileşenlerinin **System.Activities.* assembly'** ları altında olduğunu ama **.Net Framework 4.0** içerisinde yer alan ve **geriye uyumluluk(Backwards Compatibility)** amacıyla kullanılan **Workflow** bileşenlerininse **System.Workflows.* assembly'** ları içerisinde yer aldığını, bu anlamda WF 4.0' da geriye uyumluluğa da büyük önem verildiğini,
- Tamamen **WPF(Windows Presentation Foundation)** temelli bir tasarım ortamının söz konusu olduğunu ve bu sayede geliştirici deneyiminin dahada zenginleştiğini,
- Bir **Workflow** içerisine veya dışarısına yapılan **veri akışlarının(Data Flow)** çok daha kolay ele alınması için **Designer** desteği ile birlikte **Arguments** kavramının geldiğini,
- Bir aktivitenin kendi içerisinde veriyi saklaması ve farklı seviyedeki alanlarda(Scope) kullanabilmesinde rol oynayan **Variables** kavramını ayrıca **Arguments** kavramında olduğu gibi, Variables içinde **Designer** desteğinin olduğunu,
- Bir veya daha fazla **input** argümanı alıp bunlar üzerinde çeşitli operasyonlar gerçekleştiren ve geriye değer döndürebilen **ifadeler(Expressions)** yazılabildiğini, üstelik bunların**XAML** bazlı olabildiğini,
- **FlowChart, ForEach, Parallel, ParallelForEach** *(Parallel versiyonların Ekimdeki PDC' de yayınlanacak sürümde olması bekleniyor)* ve daha pek çok aktivite tipi ile zengineştirilmiş olan**temel aktivite kütüphanesi(Base Activity Library)** ni,
- **Sequential** ve **State Machine** arasında duran ama geliştirici ve iş analistlerinin, bilinen iş akışı tasarım modeline çok yakın olması nedeniyle kolayca kullanabildiği yeni **Flow Chart**modelini,
- **Workflow** ve **Activity'** ler için **Unit Test'** lerin kolayca geliştirilebiliyor olduğunu,
- **Workflow'** ların, dışarıdaki **Activity'** ler ile haberleşmenin dahada kolaylaştırılmış olduğunu,
- **4.0** versiyonunda evlenebilmerleri için **WCF** ve **WF** tarafında;
    - Workflow tarafında yenilenen çalışma zamanı motoru bulunduğunu,
    - Workflow Service' lerin **host** edilebiliyor olduğunu,
    - Workflow' lar içerisinde **XAML** bazlı olarak **WCF** servis materyallerinin tanımlanabiliyor olduğunu,*(Service Contract, Data Contract, EndPoint vb...)*
    - Visual Studio' da **Workflow Service'** ler için **Add Service Reference** desteğinin getirildiğini,
    - Yeni mesajlaşma aktivitileri***(SendAndReceiveReply, ReceiveAndSendReply gibi)*** ve bunların **mesaj korelasyon(Message Corellation)** desteğine de sahip olduğunu,

- **WF 3.X** ile yazılmış olan **Workflow'** ların **4.0** çalışma zamanı tarafından da yürütülebildiğini,
- **WF 3.X** tarafından yazılmış olan aktivitelerin **sarmalanarak(Wrap) 4.0** içerisinde de kullanılabildiğini,***(Interop activity)***
- Ayrıca WF 3.0' dan geçiş yapacaklar için bir klavuzun bulunduğunu,
- Daha detaylı bilgiler içinse şu adrese başvruabileceğimizi,

görüyoruz. Vooovvvvv!!!! Artık paraşütümüzü açalım mı ne dersiniz? 😎 Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Kitap - Framework Design Guidelines: Conventions, Idioms, and Patterns for Reusable .NET Libraries (2009-10-01T13:00:00)

kitap,



Merhaba Arkadaşlar,

.Net tabanlı yazılım geliştiricilerin hemen hepsi yaşam döngülerinin belirli dönemlerinde, var olan Framework' lerin genişletilmesi(Extend) veya en başından itibaren tasarlanıp(Design), yazılması(Development) gibi işlerle eminim ki haşır neşir olmuştur. Eğer bu süreçlerle uğraşırken sağımızda veya solumuzda daha önceden Framework geliştirilmesini tecrübe etmiş deneyimli personel bulunuyorsa, çok şanslı olduğumuzu da bilmeliyiz. 😃

Yinede profesyonel bir yazılım geliştirici ister Framework genişletme ister baştan sona geliştirme işlerinden hangisi ile uğraşmış olursa olsun şu an sizlere tanıtmaya çalıştığım kitabı en azından referans olarak almak, neleri yanlış veya doğru yaptığını anlamak için okumalıdır.

Addison Wesley yayınlarından ikinci versiyonu 1 Kasım 2008 tarihinde*(Neredeyse 1 sene önce, bu kitabı okumak için geç bile kalındığını itiraf etmeliyim* 😕 *)*çıkmış olan 480 sayfalık bu kitap henüz dün elime ulaşmış olmasına rağmen, daha Introduction bölümünde göze çarpan net ve açıklayıcı anlatımı ile şimdiden beni kendine bağlamış durumda.

Krzysztof Cwalina*(Microsoft' ta .Net Framework takımında ürün müdürü)* ve **Brad Abrams***(CLR geliştirme takımının üyeleri arasındada yer almıştır. Aynı zamanda[blogunuda](#) takip etmekteyim)* tarafından kaleme alınmış bu kitap, kılavuz niteliği taşımakla birlikte Framework geliştirmek isteyenler için oldukça önemli tavsiyelere yer vermekte. Kitapta Framework tasarlamanın ve geliştirmenin temel prensiplerinden, genel Framework tasarım kalıplarına(Design Patterns), tip(Type) ve tip üyelerinin(Type Members) genişletilmesi ve tasarlanmasından, Framework' ün çeşitli tasarım parçalarının isimlendirilmesine(Naming) kadar daha pek çok konuya yer verilmekte.

Tüm bunlar bir yana kitabı tavsiye edenler arasında**Miguel De Icaza** ve **Anders Hejlsberg** yer almakta, daha ne olsun . [Kitabı](#)bitirmek için gerçekten sabırsızlanıyorum. Sizlerede şiddetle tavsiye ederim.

[Ado.Net Data Services 1.5 - Projections (2009-09-30T09:00:00)](#)

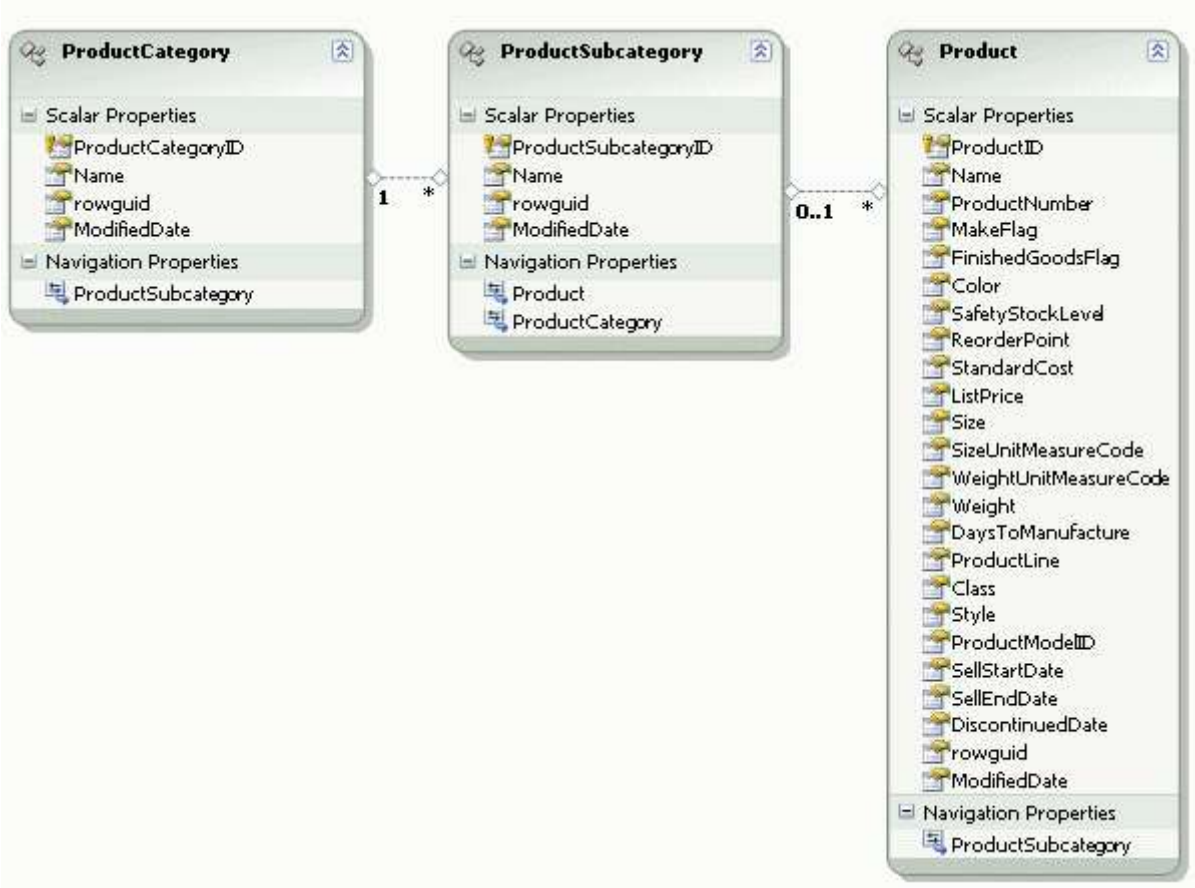
ado.net data services,wcf,rest,

Merhaba Arkadaşlar,

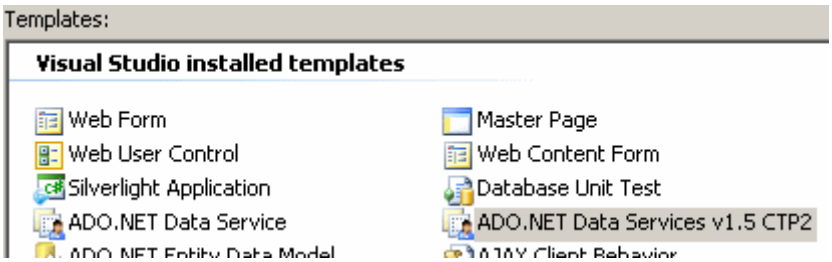
Gün geçmiyorki yazılım teknolojilerinde bir yenilik, bir güncelleme, bir genişletme çıkmasın...özellikle dünyanın dev yazılım şirketlerinin en büyüğü olarak görebileceğimiz Microsoft tarafında bu gelişme ve güncelleme hızı oldukça yüksek. Gerçektende heyecan verici yenilikler, özellikler ile karşılaşmıyor değiliz. Bu konuya nereden mi geldim? çok zaman değil daha bir sene öncesine kadar **Astoria** kod adlı**Ado.Net Data Services** konusunu incelemeye başlamıştım. **Entity Framework** veya **Custom LINQ Provider'** ları ile sunulan veri kümelerine, **REST**bazlı olarak **URL** sorgular atılabilmesini sağlayan ve özellikle **Silverlight** gibi **RIA** içeriklerinde son derece kıymetli olan bir servis uygulaması olarak değerlendirebileceğimiz bu konu ile ilişkili ilk paylaşımlarımı yaptıktan sonra araya **WCF 4.0, WF 4.0, Design Patterns, Design Principles, .Net RIA Services** gibi konular girdi. Bu konulardaki incelemelerimi ve paylaşımlarımı devam ettirirken bir baktım ki **Ado.Net Data Services** konusuna çok uzun zaman ara vermişim. Ara vermeklede iyi yapmamışım

Nitekim program yöneticisi olan **Mike Flasko** boş durmamış ve Ado.Net Data Services v1.5 versiyonu için CTP2 sürümünü duyurmuş*(.Net Framework 3.5 Service Pack 1 ve Silverlight 3.0' ı hedefleyen ama .Net Framework 4.0 içerisinede dahil edilecek olan özellikleri içeren bir sürüm olarak düşünülebilir)*. Duyurulması ile birlikte hem blog sitesinde hemde çeşitli kaynaklarda konu ile ilişkili yazılar yayınlanmaya da başlanmış.

Bu versiyonda bazı yenilikler ve daha önceki sürüme ait çeşitli düzeltmeler**(bug-fix)** yer almakta. Gelen yeni özelliklerden birisi de **Projections** kullanımı. Bu yeniliğe göre servis üzerinde gerçekleştirilen **URL** bazlı sorguların sonuçları kırpılabiliyor ve sadece ilgilenilmek istenenlerin istemci tarafına çekilmesi sağlanabiliyor. Bir başka deyişle, istemcinin yapmış olduğu bir **talebin(Request)** sonuçlarında sadece ilgilendiği özelliklerin getirilmesi sağlanabilmekte. Bunu tam olmasada, LINQ sorguları sırasında anonymous type kullanımına benzetebiliriz. Söz konusu özellik içerisinde **primitive/complex** tipleri veya **navigation** özelliklerini de kullanabilmekteyiz. özelliğin getirisi, istemcinin talebi sonrası tüm **Entity** kümesinin işlenmesi ve ağ üzerinde hareket etmesi yerine, sadece istediği özellikleri içeren kümenin/kümelerin değerlendirilebilmesi olarak görülebilir. Bu çok doğal olarak istemci ile sunucu arasındaki trafiği boyutsal olarak azaltmaktadır.**Projections** kullanımı son derece basittir. Bunun için **$select** operatöründen yararlanılmaktadır. Tabiki konuyu anlamamızın en iyi yolu basit bir örneği adım adım geliştirmek ve üzerinde ilerlemekle olacaktır. Bu nedenle kolları sıvayıp işe koyulalım. İlk olarak **Visual Studio 2008** ortamında*(Service Pack 1 yüklü olan)* basit bir Asp.Net Web Uygulaması oluşturarak işe başlayabiliriz. Sonrasında servisimiz için gerekli Entity kaynağını oluşturmamız gerekiyor. Bu amaçla Ado.Net Entity Framework' ten yararlanabilir ve yine kobay veritabanımız olan AdventureWorks' ü değerlendirebiliriz. örneğimizde aşağıdaki **EDM** şemasını kullanıyor olacağız.



AdventureWorks veritabanındaki **Production** şemasında yer alan **ProductCategory, ProductSubCategory** ve **Product** tablolarını kullanmaya çalışıyoruz. **Entity** modelimizi oluşturduktan sonra, projemize yeni bir **Ado.Net Data Services** öğesi ekleyerek devam edebiliriz. Tabi bu seferki örneğimizde **v1.5 CTP2** sürümüne ait öğeyi kullanmamız gerekiyor.

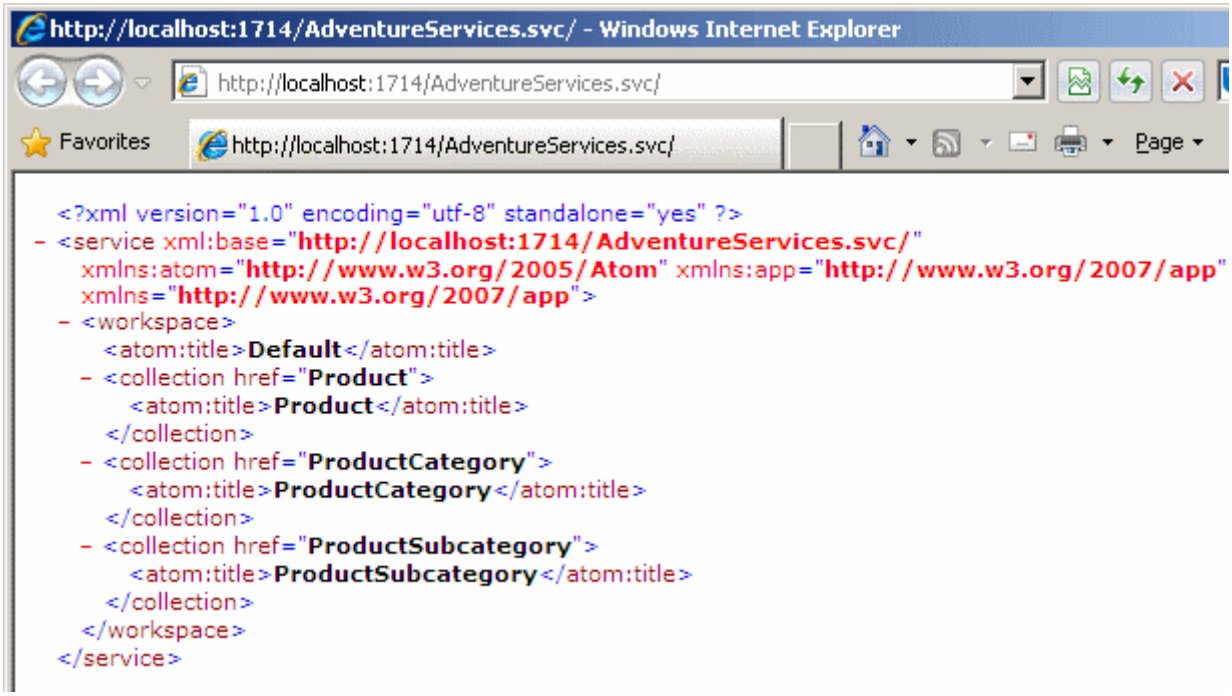


Ado.Net Data Services öğemizin kod içeriğini ise aşağıdaki gibi değiştirmemiz yeterli olacaktır.

```
using System.Data.Services;

namespace Projections
{
   public class AdventureServices
      : DataService<AdventureWorksEntities>
   {
```

```
        public static void InitializeService(DataServiceConfiguration config)
        {
            // Tüm Entity' leri sadece okuma amaçlı açıyoruz
            config.SetEntitySetAccessRule("*", EntitySetRights.AllRead);
            // İstemciden gelecek olan Projection taleplerinin değerlendirileceğini belirtiyoruz
            config.DataServiceBehavior.AcceptProjectionRequests = true;
            // Versiyon 2 için geliştirme yapacağımızı belirtiyoruz. Bu versiyon belirtilmediği
takdirde select operatörü ve projection fonksiyonelliği çalışmayacaktır.
            config.DataServiceBehavior.MaxProtocolVersion =
System.Data.Services.Common.DataServiceProtocolVersion.V2;
    }
   }
}
```

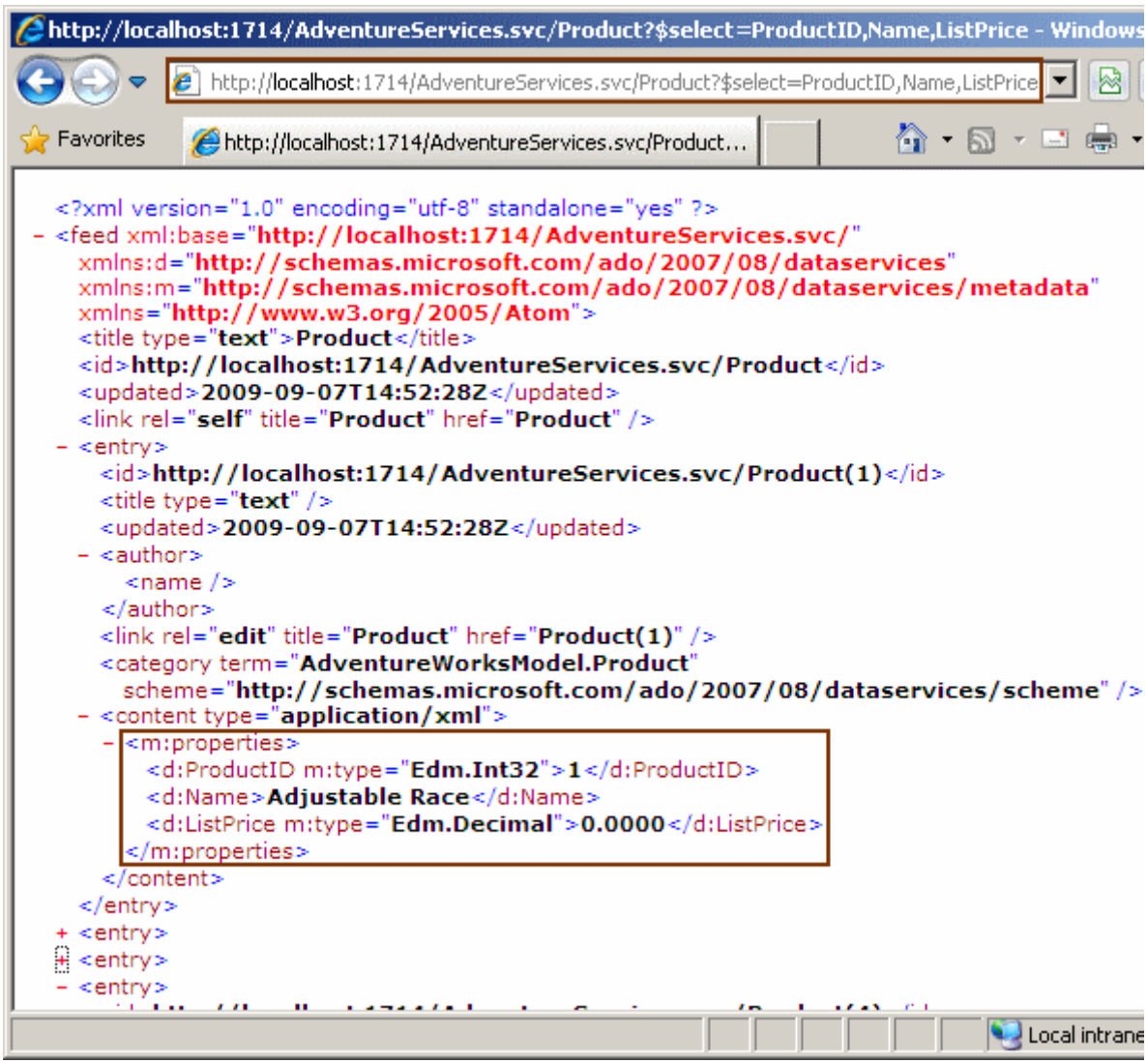
Dikkat edilmesi gereken noktalardan birisi **AcceptProjectionRequest** ise **MaxProtocolVersion** özelliklerine atanan değerlerdir. Bu değerlere göre servisimiz, istemcilere **Projection** fonksiyonelliğini sunabilecektir. **AdventureServices.svc** dosyasını bir tarayıcı yardımıyla talep ettiğimizde, başlangıç için aşağıdakine benzer bir ekran görüntüsü ile karşılaşırız.



Görüldüğü üzere Product, ProductCategory ve ProductSubcategory Entity' leri kullanılmaya hazırdır. Evetttt...Gelelim yazımızın önemli olan kısmına. Tarayıcı üzerinden aşağıdaki sorguyu talep ettiğimizi düşünelim.

http://localhost:1714/AdventureServices.svc/Product**?$select=ProductID,Name,ListPrice**

Dikkat edileceği üzere **Product** Entity' si üzerinden select sorgusu atılmış ve sadece **ProductID,Name,ListPrice** alanları talep edilmiştir. Bu sorgunun çalışma zamanı çıktısı aşağıdaki gibi olacaktır.



Dikkat edileceği üzere **Product** tablosundaki tüm ürünlerin sadece **ProductID,Name** ve **ListPrice** alanları çekilmiştir. İşin güzel yanı, bu **URL** talebi için arka planda çalıştırılan**SQL** sorgusuda sadece istenen alanları değerlendirmektedir. İşte URL' imize ait SQL sorgusunun **SQL Server Profiler'** dan yakalanan içeriği.

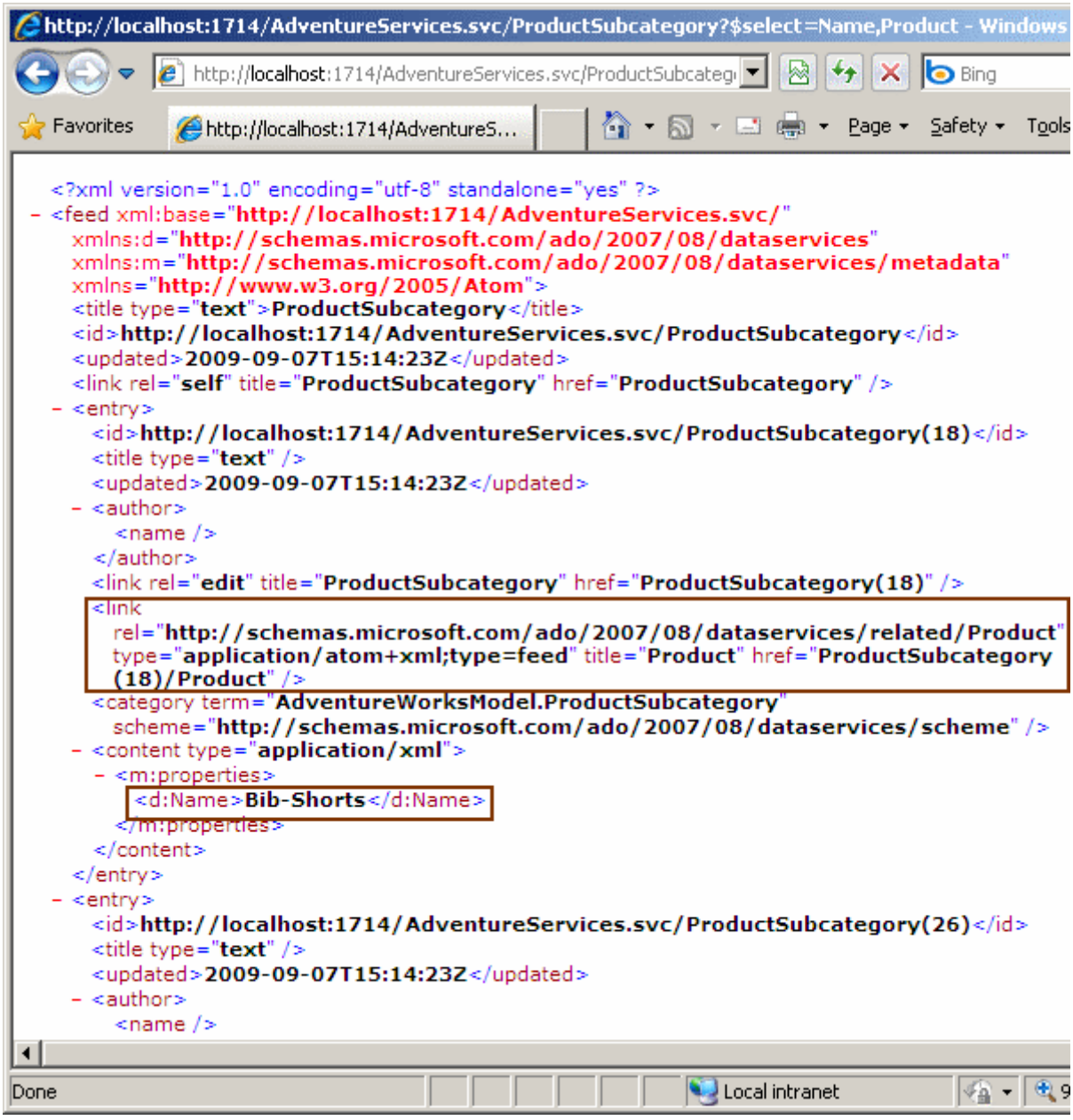
```
SELECT
1 AS [C1],
CASE WHEN ([Extent1].[ProductID] IS NULL) THEN N'' ELSE
N'AdventureWorksModel.Product' END AS [C2],
N'ProductID,Name,ListPrice' AS [C3],
[Extent1].[ProductID] AS [ProductID],
[Extent1].[Name] AS [Name],
[Extent1].[ListPrice] AS [ListPrice]
FROM [Production].[Product] AS [Extent1]
```

Dolayısıyla **Projection** kullanılaraktan, bir **Entity** üzerinden sadece istenen alanları içeren çıktıların alınması sağlanabilir. Bu kullanım aynen **SQL** tarafı içinde geçerli olduğundan, performans adına da bazı kazanımların elde edildiği ortadadır.

select operatörünü dilersek **navigasyon özellikleri(Navigation Properties)** ilede bir aradada kullanabiliriz. örneğin aşağıdaki gibi bir **URL** talebinde bulunduğumuzu düşünelim.

http://localhost:1714/AdventureServices.svc/**ProductSubcategory?$select=Name,Product**

Buna göre **ProductSubcategory Entity'** sinden sadece **Name** alanlarının değerlerini isterken, her alt kategoriye bağlı ürünleri tutan **Product Entity** örneklerini de talep etmekteyiz. Bu **URL** talebinin çıktısı aşağıdaki gibi olacaktır.



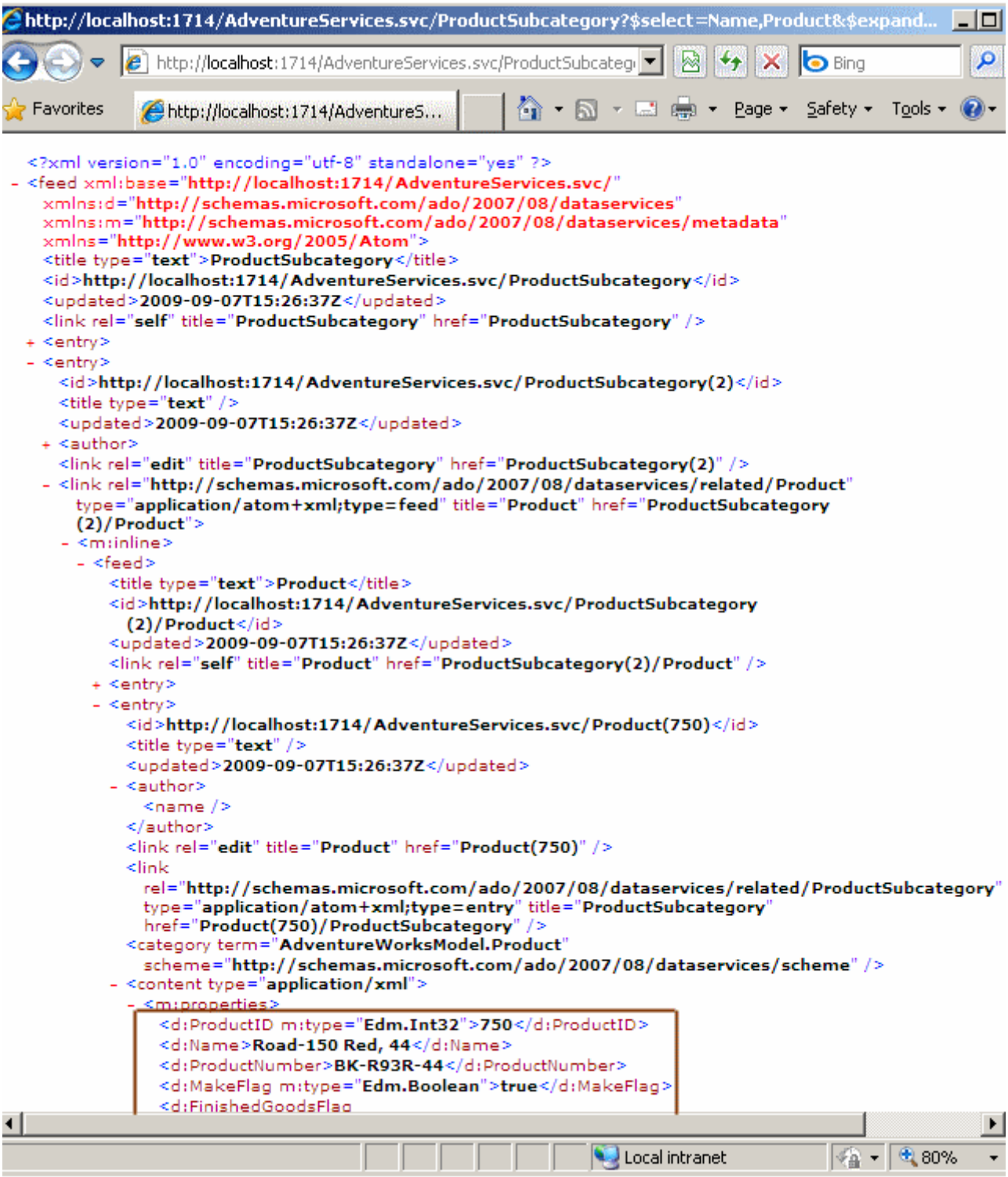
Dikkat edileceği üzere alt kategoriye bağlı olan **Product** kümeleri için sadece bağlantı bildirimi yapılmaktadır. Söz konusu **URL'** ın çalıştırılması sonucunda **SQL** tarafında da aşağıdaki sorgunun yürütüldüğü görülecektir.

```
SELECT
1 AS [C1],
CASE WHEN ([Extent1].[ProductSubcategoryID] IS NULL) THEN N'' ELSE
N'AdventureWorksModel.ProductSubcategory' END AS [C2],
N'Name,ProductSubcategoryID' AS [C3],
[Extent1].[Name] AS [Name],
[Extent1].[ProductSubcategoryID] AS [ProductSubcategoryID]
FROM [Production].[ProductSubcategory] AS [Extent1]
```

Fark edilebileceği gibi, **Product** tablosu ile ilişkili bir sorgu ifadesi yer almamaktadır. Diğer yandan sadece **Name** alanını talep etmemize rağmen, **PrimaryKey** olan **ProductSubcategoryID** alanı da getirilmektedir. Bu son derece doğaldır nitekim, belirli bir **ProductSubcategory**' nin çekilmesinde **primary key** alanı ayırt edici özelliklerdendir üstelik entry/id elementleri içerisinde gereklidir. Diğer yandan, **URL** satırını aşağıdaki gibi değiştirirsek,

http://localhost:1714/AdventureServices.svc/**ProductSubcategory?$select=Name,Product&$expand=Product&$top=2**

Hımmm... Bu sorguya göre **ProductSubcategory** içeriğinden sadece **Name** alanını almakla kalmıyor, aynı zamanda alt kategoriye bağlı olan ürünleride çekiyoruz. üstelik sadece ilk 2**ProductSubcategory** tipini ele alıyoruz*(Sondaki **top=2** sorgusu nedeniyle)*. İşte çalışma zamanı çıktımız.



Peki bu **URL** talebi sonrası arka planda nasıl bir **SQL** sorgusu çalışıyor?

SELECT
[Project2].[ProductSubcategoryID] AS [ProductSubcategoryID],[Project2].[Name] AS [Name], [Project2].[rowguid] AS [rowguid], [Project2].[ModifiedDate] AS [ModifiedDate], [Project2].[C1] AS [C1], [Project2].[C2] AS [C2], [Project2].[C3] AS [C3], [Project2].[C4] AS [C4], [Project2].[C5] AS [C5], [Project2].[C6] AS [C6], [Project2].[C7] AS [C7], [Project2].[ProductID] AS [ProductID], [Project2].[Name1] AS

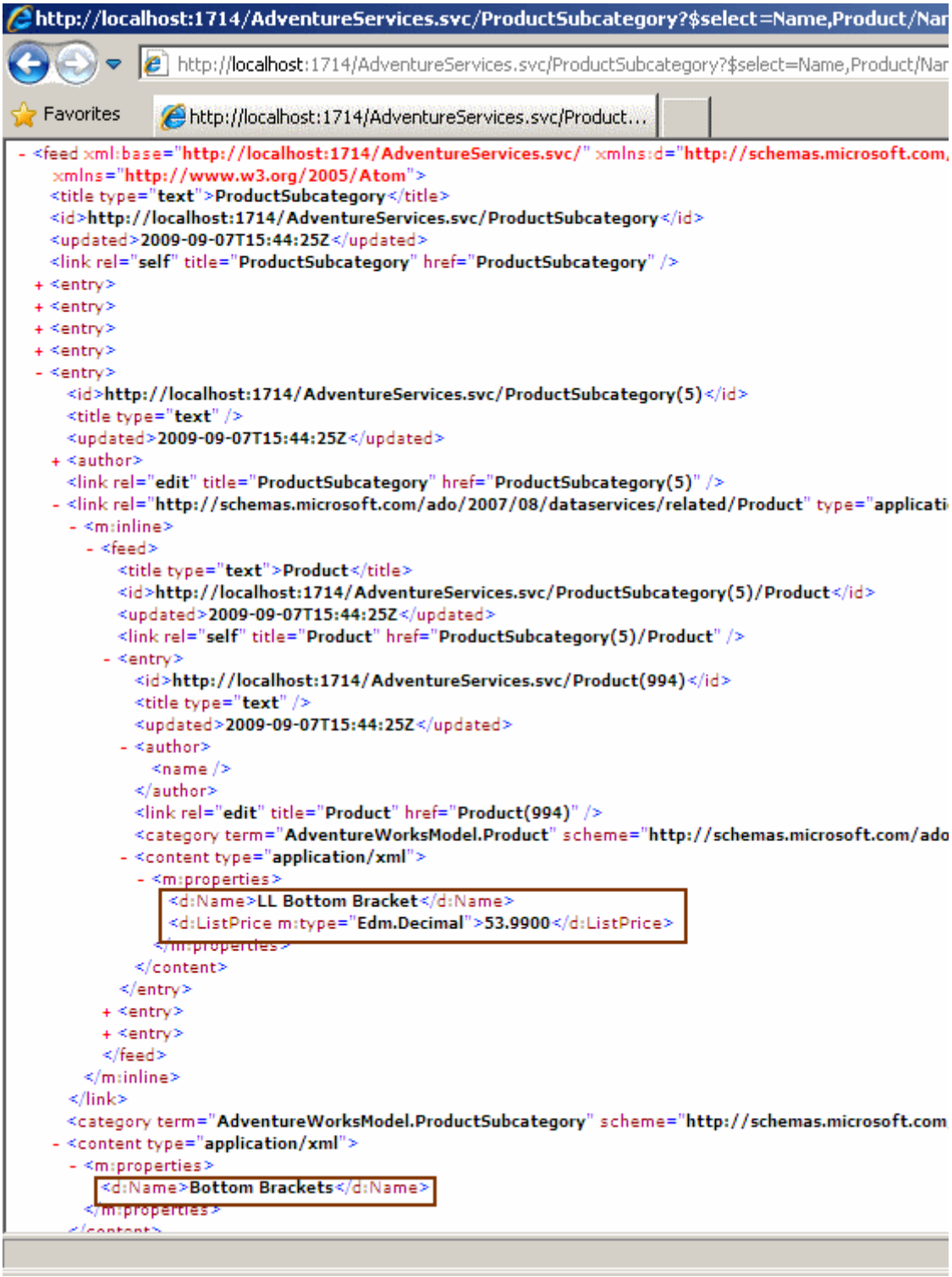
```
[Name1], [Project2].[ProductNumber] AS [ProductNumber], [Project2].[MakeFlag] AS
[MakeFlag], [Project2].[FinishedGoodsFlag] AS [FinishedGoodsFlag], [Project2].[Color]
AS [Color], [Project2].[SafetyStockLevel] AS [SafetyStockLevel],
[Project2].[ReorderPoint] AS [ReorderPoint], [Project2].[StandardCost] AS
[StandardCost], [Project2].[ListPrice] AS [ListPrice], [Project2].[Size] AS [Size],
[Project2].[SizeUnitMeasureCode] AS [SizeUnitMeasureCode],
[Project2].[WeightUnitMeasureCode] AS [WeightUnitMeasureCode], [Project2].[Weight]
AS [Weight], [Project2].[DaysToManufacture] AS [DaysToManufacture],
[Project2].[ProductLine] AS [ProductLine], [Project2].[Class] AS [Class],
[Project2].[Style] AS [Style], [Project2].[ProductModelID] AS [ProductModelID],
[Project2].[SellStartDate] AS [SellStartDate], [Project2].[SellEndDate] AS [SellEndDate],
[Project2].[DiscontinuedDate] AS [DiscontinuedDate], [Project2].[rowguid1] AS
[rowguid1], [Project2].[ModifiedDate1] AS [ModifiedDate1]
FROM ( SELECT
 [Limit1].[ProductSubcategoryID] AS [ProductSubcategoryID],  [Limit1].[Name] AS
[Name],  [Limit1].[rowguid] AS [rowguid],  [Limit1].[ModifiedDate] AS
[ModifiedDate],  [Limit1].[C1] AS [C1],  [Limit1].[C2] AS [C2],  [Limit1].[C3] AS
[C3],  [Limit1].[C4] AS [C4],  [Limit1].[C5] AS [C5],  [Limit1].[C6] AS
[C6],  [Extent2].[ProductID] AS [ProductID],  [Extent2].[Name] AS
[Name1],  [Extent2].[ProductNumber] AS [ProductNumber],  [Extent2].[MakeFlag] AS
[MakeFlag],  [Extent2].[FinishedGoodsFlag] AS [FinishedGoodsFlag],  [Extent2].[Color]
AS [Color],  [Extent2].[SafetyStockLevel] AS
[SafetyStockLevel],  [Extent2].[ReorderPoint] AS
[ReorderPoint],  [Extent2].[StandardCost] AS [StandardCost],  [Extent2].[ListPrice] AS
[ListPrice],  [Extent2].[Size] AS [Size],  [Extent2].[SizeUnitMeasureCode] AS
[SizeUnitMeasureCode],  [Extent2].[WeightUnitMeasureCode] AS
[WeightUnitMeasureCode],  [Extent2].[Weight] AS
[Weight],  [Extent2].[DaysToManufacture] AS
[DaysToManufacture],  [Extent2].[ProductLine] AS [ProductLine],  [Extent2].[Class] AS
[Class],  [Extent2].[Style] AS [Style],  [Extent2].[ProductModelID] AS
[ProductModelID],  [Extent2].[SellStartDate] AS [SellStartDate],  [Extent2].[SellEndDate]
AS [SellEndDate],  [Extent2].[DiscontinuedDate] AS
[DiscontinuedDate],  [Extent2].[rowguid] AS [rowguid1],  [Extent2].[ModifiedDate] AS
[ModifiedDate1],
 CASE WHEN ([Extent2].[ProductID] IS NULL) THEN CAST(NULL AS int) ELSE 1
END AS [C7]
 FROM   (SELECT TOP (2) [Project1].[ProductSubcategoryID] AS
[ProductSubcategoryID], [Project1].[Name] AS [Name], [Project1].[rowguid] AS
[rowguid], [Project1].[ModifiedDate] AS [ModifiedDate], [Project1].[C1] AS [C1],
[Project1].[C2] AS [C2], [Project1].[C3] AS [C3], [Project1].[C4] AS [C4], [Project1].[C5]
AS [C5], [Project1].[C6] AS [C6]
  FROM ( SELECT
   [Extent1].[ProductSubcategoryID] AS [ProductSubcategoryID],
   [Extent1].[Name] AS [Name],
   [Extent1].[rowguid] AS [rowguid],
```

```
   [Extent1].[ModifiedDate] AS [ModifiedDate],
   1 AS [C1],
   1 AS [C2],
   CASE WHEN ([Extent1].[ProductSubcategoryID] IS NULL) THEN N'' ELSE
N'AdventureWorksModel.ProductSubcategory' END AS [C3],
   N'Name,ProductSubcategoryID' AS [C4],
   N'Product' AS [C5],
   1 AS [C6]
   FROM [Production].[ProductSubcategory] AS [Extent1]
  )  AS [Project1]
  ORDER BY [Project1].[ProductSubcategoryID] ASC ) AS [Limit1]
 LEFT OUTER JOIN [Production].[Product] AS [Extent2] ON
[Limit1].[ProductSubcategoryID] = [Extent2].[ProductSubcategoryID]
)  AS [Project2]
ORDER BY [Project2].[ProductSubcategoryID] ASC, [Project2].[C7] ASC
```

Amanınnnn!!! 😬 Aslında biraz can sıkıcı ama doğal olarak tüm **Product** alanlarının değerlendirildiğini görüyoruz. Nitekim aksini belirtmedik. Peki belirtebilir miyiz? Yani**ProductSubcategory** kümesinden ve genişletilebilen **Product** kümesinden bir kaç alanı almayı başarabilir miydik? İşte örnek bir cevabı 😎

http://localhost:1714/AdventureServices.svc/**ProductSubcategory?$select=Name,Product/Name,Product/ListPrice&$expand=Product&$top=5**

Görüldüğü gibi **EntityAdı/AlanAdı(örneğin Product/Name)** stilinde yapılan bildirimlerle, üretilecek olan çıktıda birden fazla **Entity'** den gelebilecek alanları ayrı ayrı belirtebiliyoruz. *(Buna göre sizlerde ProductCategory,ProductSubcategory ve Product kümelerinin tamamının bir arada bulunduğu örnek URL üzerinde çalışabilirsiniz. çalışmanızı öneririm.)*



Görüldüğü üzere alt kategori ile ilişkili **Feed** girişlerinde **Name** alanı yer almaktayken, o alt kategoriye bağlı **Product** tipleri için sadece **Name** ve **ListPrice** değerleri getirilmektedir. Dolayısıyla **SQL** sorgusuda buna göre aşağıda görüldüğü gibi oluşacaktır.

SELECT
[Project2].[**ProductSubcategoryID**] AS [ProductSubcategoryID], [Project2].[**Name**] AS [Name], [Project2].[C1] AS [C1], [Project2].[C2] AS [C2], [Project2].[C3] AS [C3],

```
[Project2].[C4] AS [C4], [Project2].[C5] AS [C5],
[Project2].[C9] AS [C6], [Project2].[C6] AS [C7], [Project2].[C7] AS [C8], [Project2].[C8]
AS [C9], [Project2].[Name1] AS [Name1], [Project2].[ListPrice] AS [ListPrice],
[Project2].[ProductID] AS [ProductID]
FROM ( SELECT
 [Limit1].[ProductSubcategoryID] AS [ProductSubcategoryID],  [Limit1].[Name] AS
[Name],  [Limit1].[C1] AS [C1],  [Limit1].[C2] AS [C2],  [Limit1].[C3] AS
[C3],  [Limit1].[C4] AS [C4],  [Limit1].[C5] AS [C5],  [Extent2].[ProductID] AS
[ProductID],  [Extent2].[Name] AS [Name1],  [Extent2].[ListPrice] AS [ListPrice],  CASE
WHEN ([Extent2].[ProductID] IS NULL) THEN CAST(NULL AS int) ELSE 1 END AS
[C6],
 CASE WHEN ([Extent2].[ProductID] IS NULL) THEN CAST(NULL AS varchar(1))
ELSE CASE WHEN ([Extent2].[ProductID] IS NULL) THEN N'' ELSE
N'AdventureWorksModel.Product' END END AS [C7],
 CASE WHEN ([Extent2].[ProductID] IS NULL) THEN CAST(NULL AS varchar(1))
ELSE N'Name,ListPrice,ProductID' END AS [C8],
 CASE WHEN ([Extent2].[ProductID] IS NULL) THEN CAST(NULL AS int) ELSE 1
END AS [C9]
 FROM   (SELECT TOP (5) [Project1].[ProductSubcategoryID] AS
[ProductSubcategoryID], [Project1].[Name] AS [Name], [Project1].[C1] AS [C1],
[Project1].[C2] AS [C2], [Project1].[C3] AS [C3], [Project1].[C4] AS [C4], [Project1].[C5]
AS [C5]
  FROM ( SELECT
   [Extent1].[ProductSubcategoryID] AS [ProductSubcategoryID], [Extent1].[Name] AS
[Name], 1 AS [C1], 1 AS [C2],
   CASE WHEN ([Extent1].[ProductSubcategoryID] IS NULL) THEN N'' ELSE
N'AdventureWorksModel.ProductSubcategory' END AS [C3],
   N'Name,ProductSubcategoryID' AS [C4],
   N'Product' AS [C5]
   FROM [Production].[ProductSubcategory] AS [Extent1]
  )  AS [Project1]
  ORDER BY [Project1].[ProductSubcategoryID] ASC ) AS [Limit1]
 LEFT OUTER JOIN [Production].[Product] AS [Extent2] ON
[Limit1].[ProductSubcategoryID] = [Extent2].[ProductSubcategoryID]
)  AS [Project2]
ORDER BY [Project2].[ProductSubcategoryID] ASC, [Project2].[C9] ASC
```

Görüldüğü üzere **Ado.Net Data Services v1.5 CTP2** ile gelen **Projection** özelliği performans kazanımı elde etmemizi sağlayacak derecede önemli bir özellik olarak karşımıza çıkmaktadır. Bu yazımızda kullandığımız sorgular aşağıdaki gibidir.

- http://localhost:1714/AdventureServices.svc/Product**?$select=ProductID,Name,ListPrice ->**(Product kümesinden ProductID, Name ve ListPrice alanları alınır)

- http://localhost:1714/AdventureServices.svc/**ProductSubcategory?$select=Name,Product** -> (ProductSubcategory kümesinden Name alınır, her bir alt kategoriye bağlı Product kümelerinin sadece linkleri getirilir.)
- http://localhost:1714/AdventureServices.svc/**ProductSubcategory?$select=Name,Product&$expand=Product&$top=2** -> (Bir önceki sorgu değerlendirilir ama Product kümesinin tüm üyeleri ve sadece ilk iki alt kategori tipi çekilir)
- http://localhost:1714/AdventureServices.svc/**ProductSubcategory?$select=Name,Product/Name,Product/ListPrice&$expand=Product&$top=5** -> (Bir önceki sorgu çalışır ancak Product kümesinden sadece Name ve ListPrice alanları hesaba katılır. Alt kategorilerinde ilk 10 adedi getirilir.)

Bakalım **Ado.Net Data Services 1.5 CTP2** tarafında bizleri başka ne gibi sürprizler beklemekte. Bu konularıda ilerleyen yazılarımızda değerlendirmeye çalışıyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim .

Projections.rar (53,71 kb)

WCF 4.0 Yenilikleri - DataContractResolver ile Dinamik Tip Çözümleme(Dynamic Type Resolution) [Beta 1] (2009-09-27T01:30:00)

wcf,wcf 4.0,

Merhaba Arkadaşlar,

Hatırlayacağınız üzere bir önceki yazımızda, WCF serileştirme işlemlerinde **Known Types** sorunsalını değerlendirmeye çalışmıştık. Bu sorunsalın giderilmesinde ele alınan tekniklerden biriside **KnownType niteliğinin(Attribute)** kullanılmasıyıdı. Ama istersek servise uygulanacak **ServiceKnownType** niteliği ve başka diğer teknikleri de değerlendirebileceğimizden bahsetmiştik. Ne varki tüm bu teknikler **static** bir model sunmaktadır. **WCF 4.0** ile birlikte, **tip çözümlemelerinin(Type Resolution) dinamik** olarak ele alınmasını sağlayan **DataContractResovler** isimli **abstract** bir sınıfın geldiği görülmektedir. Bu sınıf **System.Runtime.Serialization.dll assembly'** ının **.Net Framework 4.0** versiyonunda yer almaktadır. Abstract bir sınıf olması, **türetmede(Inheritance)** kullanıldığı takdirde anlam kazanacak bir tip olduğunu ifade etmektedir.

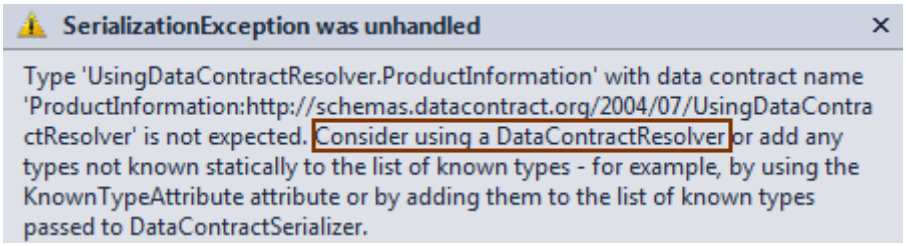
Aslında teori basittir. **DataContractResolver** sınıfı iki **abstract** metod tanımlaması içerir. **ResolveType** ve **ResolveName**. Bu metodlar tahmin edileceği üzere, tip çözümlemesinde**serileştirme(Serialization)** ve **ters-serileştirme(DeSerialization)** işlemlerinde bir veya daha fazla **Known Type'** ın ele alınması gerektiği durumlarda devreye girmektedir. çok doğal olarak metodların uygulanması için bir sınıfın **DataContractResolver** tipinden türetilmesi gerekir. O halde **"türetilen ve tip çözümlemesi işlerini üstlenen sınıf nerede kullanılır?"**sorusu da ortaya çıkmaktadır Bunun için farklı teknikler olmasına rağmen belkide en

basiti, **DataContractSerializer** nesne örneği oluşturulurken yapılan bildirimdir. Böylece,**DataContractSerializer** nesne örneğinin uygulayacağı serileştirme ve ters-serileştirme işlemleri sırasında karşılaşılabilecek olası **Known Type** sorunlarında başvurulabilecek bir yardımcı belirlenmiş olmaktadır ki bu yardımcı, **DataContractResolver** türevi olan bir sınıftır. İşe bu açılardan bakıldığında, **DataCotractResolver** sayesinde **dinamik tip çözümleme yeteneğine(Dynamic Type Resolution)** sahip olduğumuzu görebiliriz. Aslında kafalarımızı dahada karıştırmadan önce dilerseniz bir önceki yazımızda ele aldığımız ve Known Type sendromuna neden olan örneğimizi ele alıp ilerlemeye çalışalım. *(örneklerimizi Visual Studio 2010 Beta 1 ve .Net Framework Beta 1 üzerinde geliştirdiğimizi hatırlatmak isterim. Yani bir sonraki sürümde Beta 1 pek çok farklılık olabilir)*

```
using System;
using System.Runtime.Serialization;
using System.Xml;

namespace UsingDataContractResolver
{
   [DataContract]
   class Product
   {
      [DataMember]
      public object Information { get; set; }
   }

   [DataContract]
   class ProductInformation
   {
      [DataMember]
      public string Summary { get; set; }
      [DataMember]
      public int Id { get; set; }
   }

   class Program
   {
      static void Main(string[] args)
      {
         XmlObjectSerializer serializer = new DataContractSerializer(typeof(Product));

         serializer.WriteObject(new XmlTextWriter(Console.Out) { Formatting =
Formatting.Indented }
               , new Product { Information = new ProductInformation { Id = 1000,
Summary = "özet bilgi" } });
```

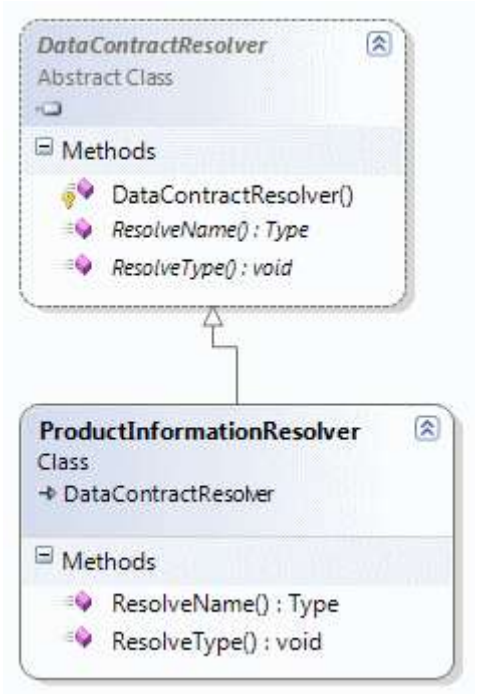
```
    }
  }
}
```

Hatırlayacağınız üzere object tipinden tanımlanmış olan Information özelliğine, ProductInformation tipinden bir nesne örneği atandığında ve Known Type ile ilişkili bir bildirimde bulunmadığımızda, çalışma zamanı hatası almaktaydık. Bu sefer örneğimizi **.Net Framework 4.0** tabanlı olarak derleyip çalıştıracağız. çok doğal alarak**SerializationException** tipinden bir istisna mesajı almayı bekliyoruz. Ancak bu sefer, **SerializationException** mesajı içerisinde **DataContractResolver** kullanılmasının da önerildiği görülmektedir.



Peki ya çözüm?

Zaten **.Net 3.5** sürümünde **KnownType** niteliği gibi materyalleri kullanarak bu sorunu aşabilmekteyiz. Ancak **WCF 4.0** ile birlikte sunulan **DataContractResolver** abstract sınıfı sayesinde, söz konusu sorunu çalışma zamanında dinamik olarak değerlendirme şansına sahibiz. Şimdi örneğimizi buna göre revize edeceğiz. İlk yapmamız gereken **DataContractResolver** türevli bir sınıfın tasarlanması olacaktır. Aynen aşağıda görüldüğü gibi.



```
class ProductInformationResolver
      :DataContractResolver
   {
      public override Type ResolveName(string typeName, string typeNamespace,
DataContractResolver knownTypeResolver)
      {
         if (typeName == "ProductInfo"
            && typeNamespace ==
"http://www.adventure.com/resolver/productInformationType")
            return typeof(ProductInformation);
         else
            return knownTypeResolver.ResolveName(typeName, typeNamespace, null);
      }

      public override void ResolveType(Type dataContractType,
DataContractResolver knownTypeResolver, out XmlDictionaryString typeName, out
XmlDictionaryString typeNamespace)
      {
         if (dataContractType == typeof(ProductInformation))
         {
            XmlDictionary dictionary = new XmlDictionary();
            typeName = dictionary.Add("ProductInfo");
            typeNamespace =
dictionary.Add("http://www.adventure.com/resolver/productInformationType");
         }
         else
         {
            knownTypeResolver.ResolveType(dataContractType, null, out typeName, out
typeNamespace);
         }
      }
   }
```

Güzel... Şimdi bir kaç noktayı açıklığa kavuşturmaya çalışalım. öncelikli olarak **DataContractResolver** tipine ait iki metodun **ezildiğini(override)** görmekteyiz. **ResolveType**metodu serileştirme işlemi sırasında devreye girmektedir ve içeride kontrol edilen tipin **XML'** de nasıl ifade edileceğini belirtmektedir***(xsi:type tanımlaması)***. Metodda ilk olarak **dataContractType**parametresinin çalışma zamanında **ProductInformation** olup olmadığı kontrol edilir. Eğer **ProductInformaion** tipindense yeni bir **XmlDictionary** nesnesi örneklenir ve **typeName** ile**typeNamespace** değerleri bu nesne üzerine **Add** metodu ile set edilir.
Zaten **typeName** ve **typeNamespace** parametrelerinin **out** tipinden oldukları gözden kaçmamalıdır. Bir başka deyişle bu parametre değerleri, **ResolveType** metodunun çağırıldığı ortama aktarılmaktadır.***(Out ve Ref parametrelerini hatırlıyorsunuz değil mi***

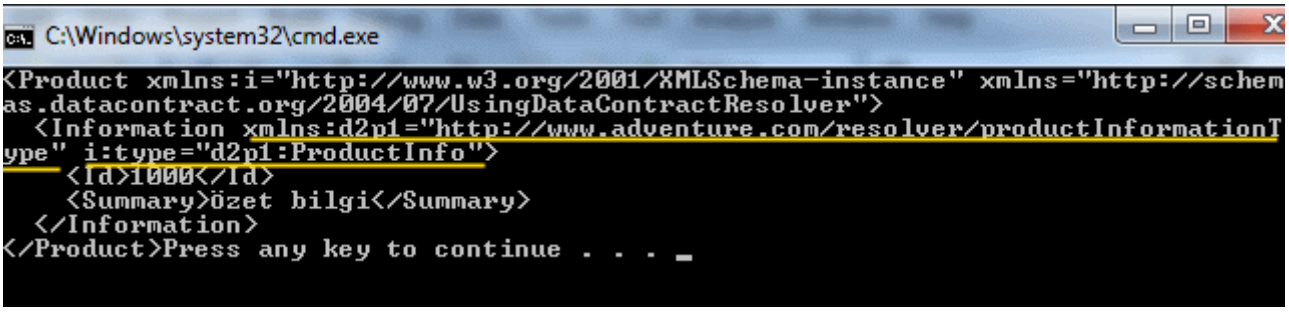
? ***)*** Kısacası, serileştirme işlemi sırasında eğer **ProductInformation** tipi ile karşılaşılırsa, tip çözümlemesinin nasıl yapılacağı geliştiricinin isteği doğrultusunda tanımlanabilmektedir. Burada yer alan**typeName** veya **typeNamespace** değerlerinin herhangibir dış ortamdan alınabileceğini***(örneğin parametrik bir XML tablosu birden fazla tipin çözümlenmesi sırasında değerlendirilebilir)*** belirtmekte yarar olduğu kanısındayım.

ResolveName metodu ise tahmin edileceği üzere **ters serileştirme(DeSerialization)** işlemi sırasında devreye girmektedir. Metod içerisinde ilk olarak **typeName** ve **typeNamespace** değişkenleri kontrol edilir ve buna göre geriye döndürülecek **tip(Type)** belirlenir. Bu metod içerisinde de **nesne tipi(Object Type)** ve **xsi:type** eşleştirmeleri için bir referans veri kaynağı*(örneğin bir XML içeriği)* kullanılabilir.

Peki ya bundan sonrası? **çalışma zamanı ProductInformationResolver tipini kullanacağını nereden bilecek?** İşte cevap...

```
XmlObjectSerializer serializer = new
DataContractSerializer(typeof(Product),null,Int32.MaxValue,false,false,null,new
ProductInformationResolver());
        serializer.WriteObject(new XmlTextWriter(Console.Out) { Formatting =
Formatting.Indented }
            , new Product { Information = new ProductInformation { Id = 1000, Summary =
"özet bilgi" } });
```

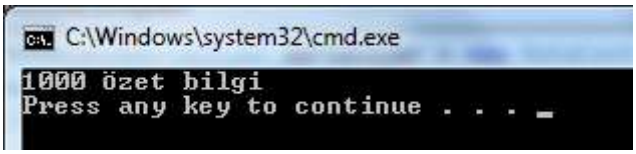
Dikkat edileceği üzere **DataContractSerialiazer** nesne örneği oluşturulurken son parametre olarak **ProductInformationResolver** örneği verilmektedir. Buna göre **serializer** nesnesinin yapacağı serileştirme ve ters-serileştirme işlemleri sırasında, **ProductInformationResolver** nesne örneği devreye girecektir. örneğimizi bu haliyle deneyecek olursak, çalışma zamanında aşağıdaki sonuçların üretildiğini görebiliriz.



Görüldüğü üzere **Information** elementi içerisinde, **DataContractResolver** türevli olan **ProductInformationResolver** nesne örneğine ait **ResolveType** metodu içerisinde belirlenen, **typeName**ve **typeNamespace** değerleri yer almaktadır. Elbetteki serileştirilen nesnenin ters-Serileştirme işlemi sırasında da işlemlerin başarılı bir şekilde yürütüldüğü gözlemlenebilir. Ama yinede kodumuzu aşağıdaki gibi revize edip ters serileştirme işleminin çalıştığından emin olmalıyız.

```
static void Main(string[] args)
      {
         FileStream fs=new FileStream("Product.xml",FileMode.Create,FileAccess.Write);
         serializer.WriteObject(fs
            , new Product { Information = new ProductInformation { Id = 1000, Summary =
"özet bilgi" } });
         fs.Close();

         Product product=(Product)serializer.ReadObject(new
FileStream("Product.xml",FileMode.Open,FileAccess.Read));
         ProductInformation information=(ProductInformation)product.Information;
         Console.WriteLine("{0} {1}",information.Id,information.Summary);

      }
```

Bu kez **Product.xml** dosyası içerisinde serileştirme işlemini yaptıktan sonra **ReadObject** metodu yardımıyla **XML** kaynağından okuma işlemini gerçekleştirmekteyiz. **ReadObject** metodu geriye **object** türünden bir referans döndürdüğü için, **bilinçli bir tür dönüşümü(Explicitly Type Cast)** yapılmaktadır. Sonrasında ise **product** nesne örneği üzerinden**Information** özelliğine gidilmekte ve **ProductInformation** tipine dönüştürülen referansın, **Id** ve **Summary** değerlerine bakılmaktadır. İşte çalışma zamanı sonucu.



WCF 4.0 son sürümü ile gelmesi muhtemel olan bu yenilik sayesinde, **Known Type** durumlarının çalışma zamanında dinamik olarak değerlendirilmesi sağlanmaktadır. Bu özelliğin geliştiricilere daha büyük bir esneklik sunduğuda ortadadır. Böylece geldik WCF 4.0 ile ilgili bir yeniliğin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WF - ExternalDataExchange, Local Services ve CallExternalMethodActivity (2009-09-25T16:23:00)

wf,

Merhaba Arkadaşlar,

Artık yazın bittiği, okulların açıldığı, şehrin kalabalığının arttığı bu günlerde birde sağnak yağışlar işin içerisine girince, insan ister istemez tatilde üzerinden denize atladığı bir iskelede olmak istiyor. Artık o iskelenin etrafında fazla insan yok ve yağmur yüzünden tahtaların üzerinde gizemli bir şekilde akan su birikintileri var; diyerek yaptığımız duygusal girişimizin aslında yazımızın ilerleyen kısmı ile bir alakası yok. Ama yine böyle yağmurlu bir günde cama vuran damlacıkları izlerken **Workflow Foundation** ile ilişkili düşündüğüm ve aklıma gelen bir konunun çözümünü sizlerle paylaşmak niyetindeyim.

İhtiyaç : Birden fazla aktivitenin aynı fonksiyonları ortaklaşa kullanabilmeleri nasıl sağlanır? Yani bir fonksiyonun birden fazla aktivite içerisinde kullanılması gerektiği durumlarda nasıl bir yol izleyebiliriz?

çözüm : Böyle bir ihtiyaçta metodların kod içeriklerini tüm aktivitelerde örneğin **CodeActivity** bileşenleri içerisinde değerlendirebiliriz. Ama bu durumda merkezileştirilmemiş ve güncelleştirmeler sırasında kullanıldığı tüm aktivitelerde düşünülmesi gereken bir çözüm üretmiş oluruz. Aslında bir yol olarak söz konusu fonksiyonellikleri ortak bir kütüphane içerisinde toplayabilir ve yine CodeActivity' ler içerisinden çağırabiliriz. Lakin bu noktada değerlendirebileceğimiz başka bir çözüm daha vardır ve gerçekten araştırılmaya değerdir. Buna göre, **Local Service** olarak çalışma zamanına eklenmiş bir arayüzden yararlanılabilir ve ortak fonksiyonelliklerin bu arayüz üzerinden aktiviteler ile mesajlaşması sağlanabilir.

Burada kritik olan
nokta **ExternalDataExchange niteliği(attribute)** ile işaretleniş bir **arayüzü(Interface)** im plemente eden bir tipin fonksiyonelliklerinin, herhangibir aktivite tarafından kullanılabilir hale gelmesidir. Tabi bu kullanımı sağlamak için **CallExternalMethodActivity** aktivite tipinden yararlanılması gerekir. Geliştirici olarak çalışma şeklini iyice kavramak yakalayacağımız kavramlar açısından önemlidir.
öncelikle **CallExternalMethodActivity** bileşeninin bir aktivite tipi olarak harici bir metodu işaret edebileceğini göz önüne almalıyız. Bu durumda **tasarım zamanında(Design Time), CallExternalMethodActivity** bileşeninin çağıracağı harici metodun imzasını ve nerede olduğunu bilmesi gerekmektedir ki çalışma zamanında bu bilgilerden yararlanarak,

içinde bulunduğu aktivite ile harici metod arasında bir mesajlaşma sağlayabilsin. Diğer yandan, tasarım zamanında **IDE**' nin **CallExternalMethodActivity** bileşenine kullanabileceği tipleri göstermesi, basit bir plug-in düzeneğine benzetilebilir. Söz konusu bileşen kullanabileceği tipleri bulmak konusunda, **ExternalDataExchange** niteliğini uygulamış interface tiplerini baz almaktadır. Buna göre arayüz tipinin çalışma zamanında gerçek işlevleri içeren bir uygulayıcısı da olmalıdır. Yani söz konusu arayüzü implemente eden bir tipten bahsediyoruz. Aktiviteler birden fazla **CallExternalMethodActivity** bileşeni içerebileceği gibi, birden fazla **ExternalDataExchange** nitelikli arayüz implementasyonunu da değerlendirebilir.

Artık konuyu örnekleyerek devam etmekte yarar olacağı kanısındayım. örneğimizi **Visual Studio 2008** ortamında ve **.Net Framework 3.5** odaklı olarak geliştiriyor olacağız. İlk olarak**System.Workflow.Activities** assembly' ını referans eden bir **Class Library** projesi oluşturarak işe başlayalım. Bir sınıf kütüphanesi tasarladığımızdan, herhangibir **Workflow**projesinde kullanılabilir ve tek merkezden güncellenebilir bir ürünümüz söz konusudur. Bu kütüphane, **ExternalDataExchange** nitelikli arayüz ve implementasyonlarını yapan tipleri barındırabilir ki örneğimizde bu amaçla aşağıdaki sınıf diagramında görülen tipler değerlendirilecektir.

Kod içeriğimiz;

```
using System;
using System.Workflow.Activities;

namespace CommonOperations
{
   // ICommonAccounting arayüz tipinin yerel servislerden(Local Service) birisi olduğu
belirtilir
 [ExternalDataExchange]
 public interface ICommonAccounting
 {
       void IncreaseRate(double rate, int categoryId);
       void DecreaseRate(double rate, int categoryId);
 }

   // Yerel servis metodlarının uygulandığı yer
   public class CommonAccounting
      :ICommonAccounting
   {
      // Host uygulamanın değerlendirebileceği basit bir olay
      public event EventHandler<AccountingResultsEventArgs> OnCompleted;

      #region ICommonAccounting Members

      public void IncreaseRate(double rate, int categoryId)
      {
         Console.WriteLine("{0} kategorisindeki maaşlar % {1} oranında
arttırılacak",categoryId.ToString(),rate.ToString());
         if (OnCompleted != null)
            OnCompleted(this, new AccountingResultsEventArgs { StepType = "Increase",
Rate = rate,StepOk=true });
      }

      public void DecreaseRate(double rate, int categoryId)
      {
         Console.WriteLine("{0} kategorisindeki maaşlar % {1} oranında azaltılacak",
categoryId.ToString(), rate.ToString());
         if (OnCompleted != null)
            OnCompleted(this, new AccountingResultsEventArgs { StepType = "Decrease",
Rate = rate,StepOk=true });
      }

      #endregion
   }
```

```
   // OnCompleted olayı içerisinde kullanılan ve olay metoduna bilgi taşıyan sınıf
   public class AccountingResultsEventArgs
      : EventArgs
   {
      public string StepType { get; set; }
      public double Rate { get; set; }
      public bool StepOk { get; set; }
   }
}
```

ICommonAccounting isimli arayüze **ExternalDataExchange** niteliği uygulanmıştır. Arayüzümüzde, işlevleri bizim için şu aşamada çok önemli olmayan iki basit operasyon tanımlaması yer almaktadır. Diğer taraftan bu arayüzü implemente eden **CommonAccounting** tipi içerisinde operasyonların uygulaması yer almaktadır. **CommonAccounting** sınıf ayrıca, kendisini kullanan aktivitelere bilgi taşıyabilmekte kullanılabilecek bir olay bildirimi de***(OnCompleted)*** içermektedir. Bu olay içerisinde kullanılan **AccountingResultEventArgs** isimli **EventArgs** türevli tip, çalışma zamanındaki **CommonAccounting** nesne örneğinden, **OnCompleted** olayına abone olan aktiviteye **StepType, Rate** ve **StepOk** gibi bazı yardımcı bilgiler döndürmektedir.**IncreaseReate** ve **DecreaseRate** metodları içerisinde, **OnCompleted** olayının yüklü olması halinde çalıştırılması işlemi gerçekleştirilmektedir.

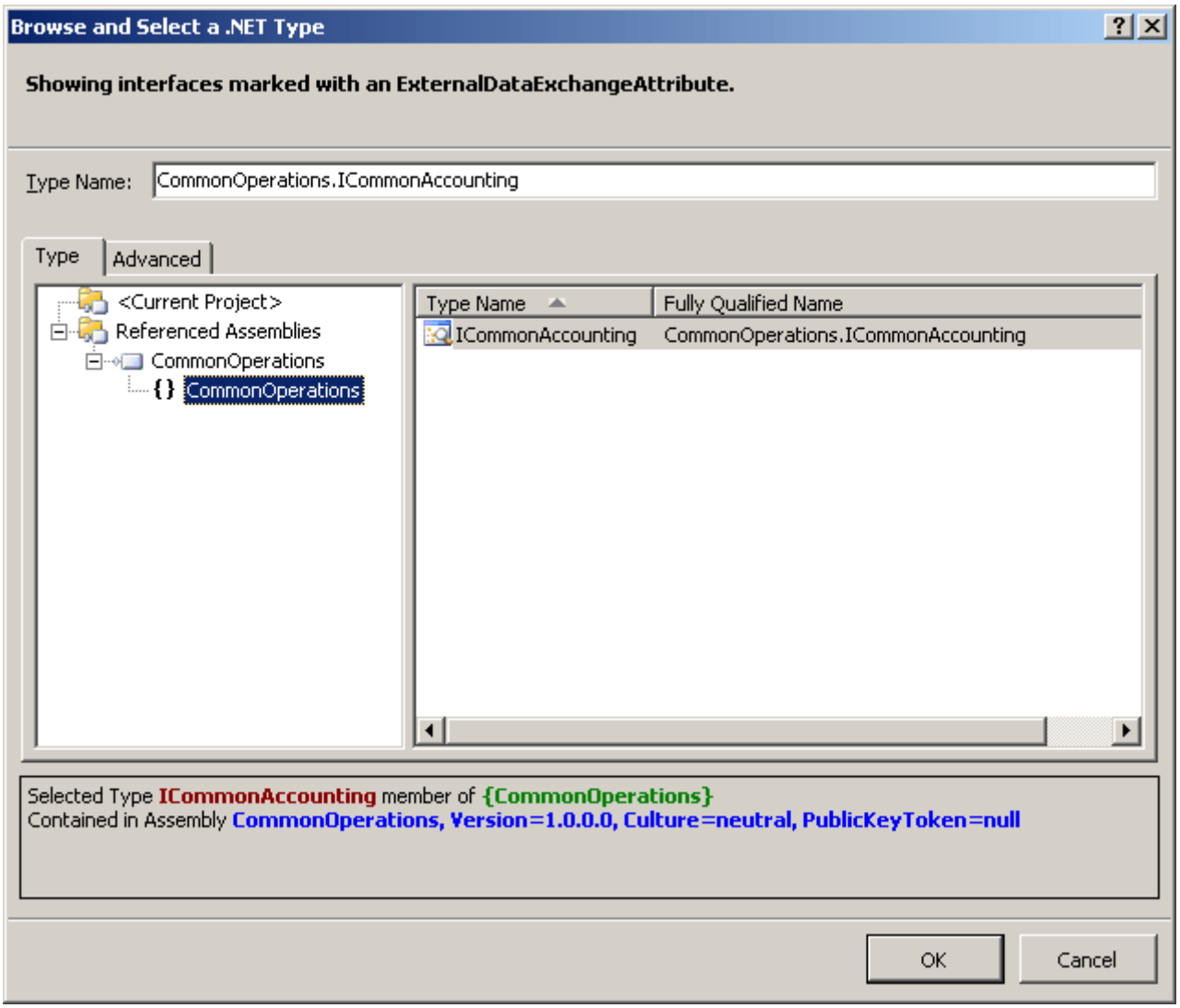
Kişisel Not : *Olayları daha net kavrayabilmek için eski bir makalemden faydalanabilirsiniz.*

Artık bu sınıf kütüphanesini kullanacak basit bir **Workflow** projesi geliştirebiliriz. Bu amaçla bir **Sequential Workflow Console Application** projesi oluşturduğumuzu ve geliştirdiğimiz**CommonOperations** isimli sınıf kütüphanesini buraya referans ettiğimizi düşünelim. Boş bir **Activity** öğesini projeye ekledikten sonra içeriğini aşağıdaki gibi kodlayalım.

```
using System.Workflow.Activities;

namespace HostApp
{
   public partial class Activity1
      : SequenceActivity
   {
      public double IncreaseRate { get; set; }
      public double DecreaseRate { get; set; }
      public int CategoryId { get; set; }

      public Activity1()
      {
```

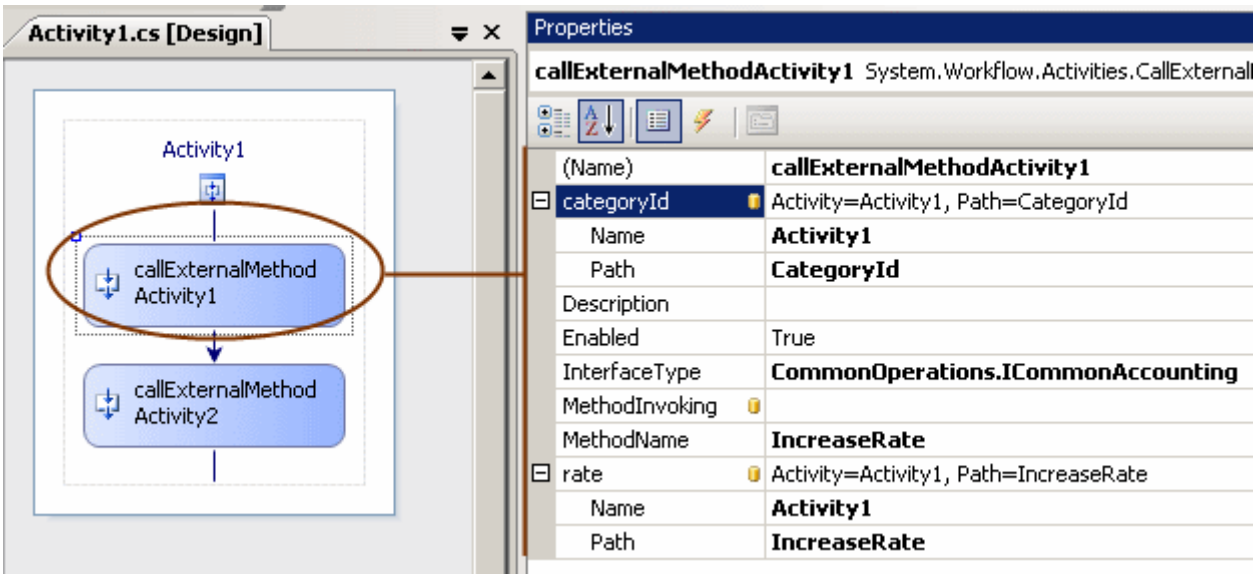
```
            InitializeComponent();
        }
    }
}
```

Burada tanımlanan **IncreaseRate,**
DecreaseRate ve **CategoryId** özellikleri, **CallExternalMethodActivity** bileşenlerinin kullanacağı harici metodlara aktarılacak aktivite seviyesindeki değerleri taşımak üzere kullanılmaktadır. Şimdi tasarım zamanında, **Activity1** içerisine örnek bir **CallExternalMethodActivity** bileşenini sürükleyerek devam edebiliriz. Bu işlemin ardından bileşenin**InterfaceType** özelliğinden yararlanarak hangi arayüzü kullanacağını aşağıdaki şekilden görüldüğü gibi seçebiliriz.

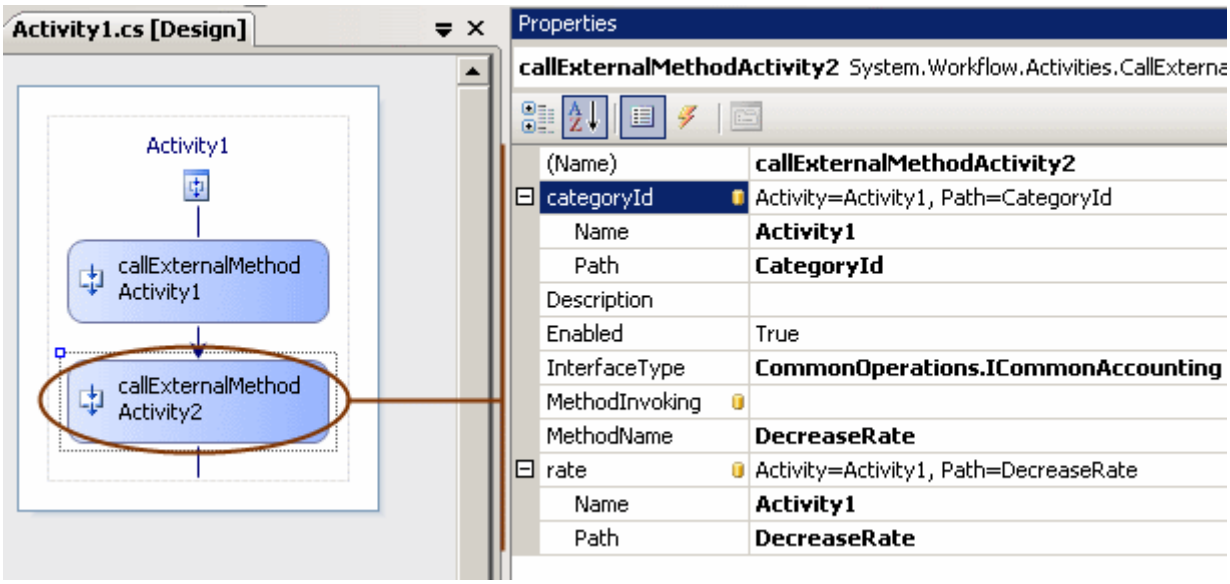


Görüldüğü gibi **ICommonAccounting** arayüzü otomatik olarak gelmiştir. Böylece hangi operasyonların kullanılabileceği, bu operasyonlara hangi parametrelerin verilmesi gerektiği bilinmektedir. Bizde akışımıza örnek olarak iki **CallExternalMethodActivity** bileşeni ekleyip özelliklerini aşağıdaki gibi ayarlayarak devam edebiliriz.

callExternalMethodActivity1 bileşeninin özellikleri;



callExternalMethodActivity2 bileşeninin özellikleri;



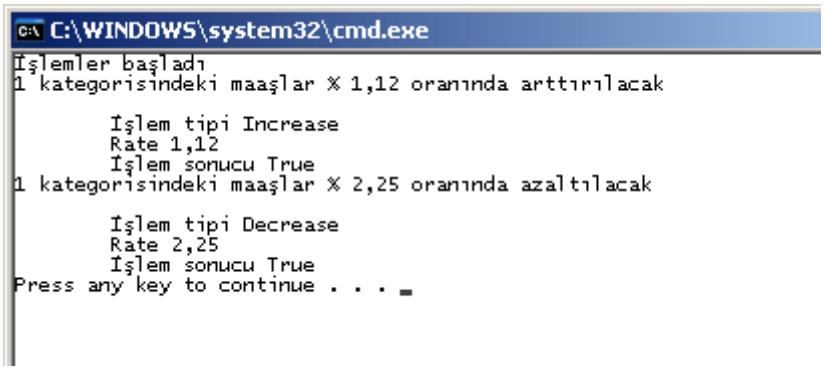
Görüldüğü gibi her iki bileşen için **ICommonAccounting** arayüzü seçilmiş, buna göre sırasıyla **IncreaseRate** ve **DecreaseRate** operasyonlarının kullanılacağı belirtilmiştir. Ayrıca söz konuzu operasyonların parametreleri, otomatik olarak özellikler penceresine gelmiştir**(rate ve categoryId).** Bu özelliklerde aslında, **Activity1** tipi içerisinde tanımlanmış olan**IncreaseRate,DecreaseRate** ve **CategoryId** özelliklerini işaret edecek şekilde bizim tarafımızdan ayarlanmaktadır. Dolayısıyla **WF çalışma zamanında**, **Activity1** içerisindeki ilgili özelliklere atanabilecek olan değerler, **CallExternalMethodActivity** bileşenleri ile **CommonAccounting** nesnesinin ilgili metodlarına gönderilerek işlenebilecektir. Eğer WF çalışma zamanını host eden sınıf, **CommonAccounting** tarafından tanımlanmış **OnCompleted** olayınıda yüklerse, **CallExternalMethodActivity** bileşenlerinin çalıştırdığı harici metodlardan bazı

bilgileri kendi ortamına alarak değerlendirebilecektir**(AccountingResultEventArgs yardımıyla)**. Bu yapı için WF çalışma zamanına özel bazı kodlamaların yapılmasıda gerekmektedir. İşte uygulama kodlarımız;

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Workflow.Activities;
using System.Workflow.Runtime;
using CommonOperations;

namespace HostApp
{
   class Program
   {
      static void Main(string[] args)
      {
         using(WorkflowRuntime workflowRuntime = new WorkflowRuntime())
         {
            #region Yerel Servisi Bildirme İşlemi

            // Yerel servisler için eklenmesi gereken servistir
            ExternalDataExchangeService service = new
ExternalDataExchangeService();
            // ExternalDataExchangeService örneği Workflow çalışma zamanına eklenir
            workflowRuntime.AddService(service);

            // ExternalMetadaExchange nitelikli interface tipini implemente eden asıl nesne
örneklenir
            CommonAccounting accounter = new CommonAccounting();
            // HostApp uygulamasının ele alacağı OnCompleted olayı yüklenir ve
anonymous method yardımıyla değerlendirilir.
            accounter.OnCompleted += delegate(object sender,
AccountingResultsEventArgs e)
            {
               // örnek metodlardan gelen sonuçlar listelenir.
               Console.WriteLine("\n\tİşlem tipi {0}\n\tRate {1}\n\tİşlem sonucu
{2}", e.StepType, e.Rate.ToString(),e.StepOk.ToString());
            };

            //accounter isimli ExternalMetadaExchange nitelikli interface tipini implemente
eden asıl nesne örneği, ExternalDataExchangeService örneğine eklenir.
            service.AddService(accounter);

            #endregion
```

```
            AutoResetEvent waitHandle = new AutoResetEvent(false);
            // Workflow tamamlandığında devreye giren olay metodu
            workflowRuntime.WorkflowCompleted += delegate(object sender,
WorkflowCompletedEventArgs e) {
               waitHandle.Set();
            };
            // Exception gibi nedenlerle Workflow sonlandığında devreye giren olay metodu
            workflowRuntime.WorkflowTerminated += delegate(object sender,
WorkflowTerminatedEventArgs e)
            {
               Console.WriteLine(e.Exception.Message);
               waitHandle.Set();
            };

            // Yerel servis içerisindeki metodların kullanacağı parametrelere işaret eden
özellikler yüklenir.
            Dictionary<string, object> parameters = new Dictionary<string, object>
            {
               {"IncreaseRate",1.12},
               {"DecreaseRate",2.25},
               {"CategoryId",1}
            };

            // Aktivite nesnesi örneklenir, özellikleri için ilk değerleri yüklenir.
            WorkflowInstance mathActivity =
workflowRuntime.CreateWorkflow(typeof(HostApp.Activity1),parameters);
            // Aktivite başlatılır
            mathActivity.Start();
            // Asenkron işleyişi ispat etmek için
            Console.WriteLine("İşlemler başladı");
            // İşlemler tamamlanmadıysa bekle
            waitHandle.WaitOne();
         }
      }
   }
}
```

Görüldüğü üzere **Local Service'** in tanımlanmasını takiben, **ExternalDataExchange** nitelikli tipe ait nesne örneklenmiş ve yerel servise bildirilmiştir. Ayrıca **CommonAccounting** nesnesinin**OnCompleted** olayı yüklenmiş ve **Program'** ın bu olaya abone olması sağlanmıştır. **Activity1** nesnesine ait özellikleri set etmek için **Dictionary<string,object>** koleksiyonundan yararlanılmış ve son olarak aktivitemiz başlatılmıştır. İşte çalışma zamanı sonuçları.



Evet...önce belirli oranda arttırım yapıp sonra azaltım yapmak son derece saçma gözükmektedir Ancak yakalamamız gereken nokta elbetteki bu değildir. önemli olan, bir aktivite' nin kendi sınırları dışındaki fonksiyonellikleri kullanabilmek için yerel servislerden nasıl yararlanıldığı ve bunun için **ExternalDataExchange** niteliğinin nasıl değerlendirildiğidir. üstelik bu değerlendirme, WF tasarım zamanı içinde önem arz eder. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

UsingExternalCode.rar (50,98 kb)

WCF 4.0 Yenilikleri - HTTP Cache Desteği [Beta 1] (2009-09-22T22:07:00)

wcf,wcf 4.0,

Merhaba arkadaşlar,

Performans pek çok uygulama geliştirme ortamında önem arz eden konuların başında gelmektedir. özellikle Web tabanlı uygulamalarda performans arttırmak adına göz önüne alınan kriterlerden biriside farklı tipteki **önbellekleme(Caching)** işlemleridir. En basit ve popülerlerinden birisi olan**Output Caching,REST** tabanlı **WCF** servisleri içinde kullanılabilmektedir. WCF' in önceki sürümünde **WebOperationContext** tipinden yararlanılarak ekstra kod eforu ile ele alınabilen **Output Cache** özelliği, **4.0** sürümünde tamamen **dekleratif** olarak değerlendirilebilmektedir. Aslında bu yenilik bilindiği üzere **WCF Rest Starter Kit Preview 2** ile birlikte **.Net Framework 3.5** üzerinde de uygulanabilmektedir. **Output Cache** özelliği performans için önemli bir kriter olduğundan, **WCF 4.0** versiyonunda doğrudan ele alınmaktadır.

Bu yazımızda ön bellekleme işleminin **WCF 4.0** içerisinde, **REST** tabanlı servisleri için nasıl geliştirilebileceğini ele almaya çalışacağız. İşe ilk olarak bir **WCF Service Application**projesi oluşturarak ve içerisine aşağıdaki servis sözleşmesi ve uygulayıcı tipi ekleyerek başlayabiliriz.

Servis Sözleşmesi(Service Contract)

```
using System.ServiceModel;
using System.ServiceModel.Web;
using System.ServiceModel.Web.Caching;

namespace Calculus
{
   [ServiceContract(Namespace="http://calculus/BasicMathService")]
   public interface IBasicMathService
   {
      [AspNetCacheProfile("ShortCache")] // Config dosyasındaki outputCacheProfile girdilerinden parametre olarak verilen isimdekini işaret eder
      [OperationContract]
      [WebGet]
      string Sum(double x, double y);
   }
}
```

Uygulayıcı tip;

```
using System;
using System.ServiceModel.Activation;

namespace Calculus
{
   [AspNetCompatibilityRequirements(RequirementsMode=AspNetCompatibilityRequirementsMode.Allowed)]
   public class BasicMathService
      : IBasicMathService
   {
      public string Sum(double x, double y)
      {
         return string.Format("{0} zamanlı hesaplama {1}+{2}={3}", DateTime.Now.ToLongTimeString(), x, y, x + y);
      }
   }
}
```

Servisimize ait svc içeriği;

<%@ ServiceHost Language="C#" Debug="true" Service="Calculus.BasicMathService" CodeBehind="BasicMathService.svc.cs" **Factory="System.ServiceModel.Activation.WebServiceHostFactory"** %>

IBasicMathService isimli servis sözleşmesi içerisinde yer alan **Sum** metoduna **WebGet** ve **OperationContract** dışında **AspNetCacheProfile** isimli bir niteliğin(attribute) daha uygulandığı görülmektedir. Bu nitelik parametre olarak **string** bir bilgi alır. Bu bilgi ise biraz sonra yazacağımız **Web.config** dosyası içerisindeki bir **Cache** profilini işaret etmektedir. Dolayısıyla bir operasyonun çıktısının ön belleklenmesi için gerekli özellikler, konfigurasyon dosyasında tanımlanır. Servis kodunda dikkat çekici noktalardan biriside, **BasicMathService** tipinin,**AspNetCompatibilityRequirements** isimli niteliği uygulamış olmasıdır. Bu durumu biraz sonra değerlendiriyor olacağız nitekim uygulanmadığı hallerde başımıza iş açacaktır 😕

Tabikide üzerinde durmamız gereken en önemli kısım config dosyası içeriğidir.

```
<?xml version="1.0"?>
<configuration>
 <system.web>
  <compilation targetFrameworkMoniker=".NETFramework,Version=v4.0" debug="false"/>
 </system.web>
 <system.web>
  <caching>
   <outputCacheSettings>
    <outputCacheProfiles>
     <!-- Süreleri farklı olan iki ayrı Cache profili tanımlanmıştır -->
     <add name="ShortCache" duration="20" varyByParam="none"/>
     <add name="LongCache" duration="600" varyByParam="none"/>
    </outputCacheProfiles>
   </outputCacheSettings>
  </caching>
 </system.web>
 <system.serviceModel>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>
  <!--WebServiceHostFactory tipini kullandığımız için aşağıdaki davranışı eklememize gerek kalmamıştır-->
  <!--<behaviors>
     <endpointBehaviors>
       <behavior>
         <webHttp enableHelp="true" />
       </behavior>
     </endpointBehaviors>
   </behaviors>-->
```

```
  <services>
   <service name="Calculus.BasicMathService">
    <endpoint binding="webHttpBinding" contract="Calculus.IBasicMathService"/>
   </service>
  </services>
 </system.serviceModel>
</configuration>
```

Görüldüğü üzere **outputCacheSettings** elementi içerisinde iki farklı önbellekleme profili tanımlanmıştır. Buna göre, **Sum** isimli operasyonumuz **20** saniyelik ön bellekleme yapan ve**Querystring** parametrelerini hesaba katmayan bir yapı sunmaktadır. *(Tabiki önbellekleme profilini oluştururken farklı özellikleride değerlendirebiliriz. örneğin ön bellekleme lokasyonunu değiştirebilir sunucu tarafı, istemci tarafı veya her iki taraf gibi değerler verilebilir.)* **Config** dosyasında **bold** olarak işaretlediğimiz diğer kısımlarda önemlidir. **WCF 4.0** tarafına getirilen**Output Cache** özelliği aslında **Asp.Net** ortamının hazır olarak sahip olduğu **Output Cache** kabiliyetlerini kullanmaktadır. Bu nedenle Asp.Net uyumluluğu önemlidir.

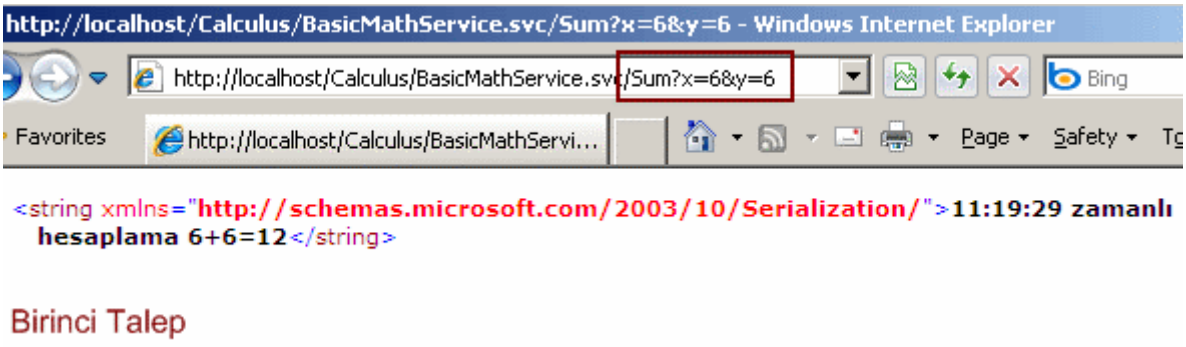
Uygulama basit bir kaç ayarlamaya sahip olmasına rağmen geliştirme ve testler sırasında beklenmedik pek çok hata ile karşılabiliriz. İşte karşılaşabileceğimiz bir kaç hata ve önerilen çözümler*(ki bu çözümlerin bir kısmı must olarak görülmelidir)*

- **Web.config** dosyasında **targetFrameworkMonikor** değerinin set edildiğinden emin olmalıyız. Edilmediği takdirde çalışma zamanında alınacak hata mesajı : **"The application domain or application pool is currently running version 4.0 or later of the .NET Framework. This can occur if IIS settings have been set to 4.0 or later for this Web application, or if you are using version 4.0 or later of the ASP.NET Web Development Server. The <compilation> element in the Web.config file for this Web application does not contain the required 'targetFrameworkMoniker' attribute for this version of the .NET Framework (for example, '<compilation targetFrameworkMoniker=".NETFramework,Version=v4.0">'). Update the Web.config file with this attribute, or configure the Web application to use a different version of the .NET Framework."**
- **aspNetCompatibilityEnabled** niteliğinin değeri **true** olmadığı sürece, **AspNetCacheProfile** özelliğini kullanamayız. Alınacak çalışma zamanı hata mesajı :**"AspNetCacheProfileAttribute is supported only in AspNetCompatibility mode. "**
- **aspNetCompatibilityEnabled** niteliğinin **true** olarak atamış olması yeterli olmayacaktır. Nitekim servis sınıfı için **AspNetCompatibilityRequirements** niteliğinin de set edilmesi gerekir. Edilmediği takdirde alınacak çalışma zamanı hata mesajı : **"The service cannot be activated because it does not support ASP.NET compatibility. ASP.NET compatibility is enabled for this application. Turn off ASP.NET compatibility mode in the web.config or add the**

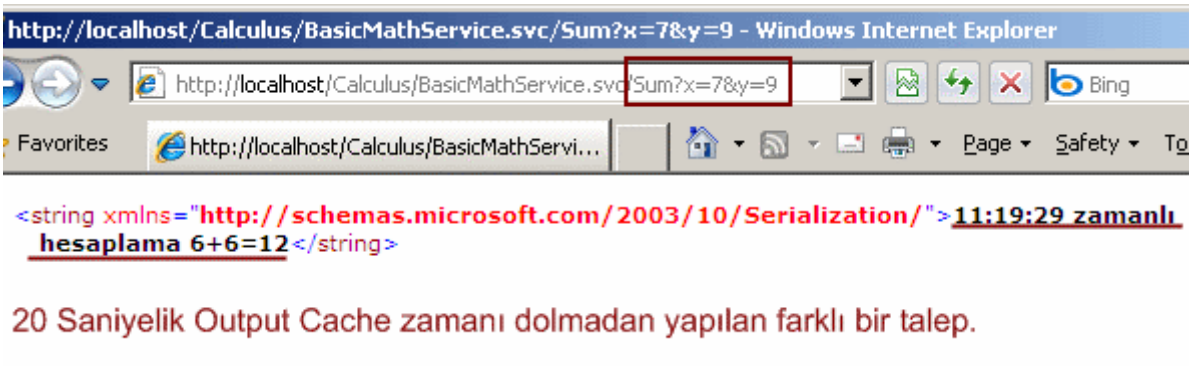
AspNetCompatibilityRequirements attribute to the service type with RequirementsMode setting as 'Allowed' or 'Required'"

- **Asp.Net Development Server** üzerinden geliştirme yapıldığında **Output Cache** özelliği test edilememektedir. **Output Cache** özelliğinin çalışabilmesi için, servis uygulamasınınIIS üzerine dağıtılması gerekmektedir.
- **IIS** üzerine atılan WCF servis uygulamasının özelliklerinden **Target Framework'** ün **4.0** versiyonunu işaret edecek şekilde değiştirilmesi gerekir. *(örneği geliştirdiğim zamanda 4.0.20506 versiyonu idi)*
- Güvenlik ile ilişkili bir sıkıntı yaşanabilir. özellikle varsayılan olarak **Anonymous Access** ve **Integrated Windows Authentication** modlarından her ikiside seçili gelebilir. Bu noktada WCF çalışma zamanı sadece bir tanesinin seçili olmasını isteyebilir. Bu durumda sadece **Anonymous Access** seçeneğini işaretleyerek testleri yapabiliriz. Yapılan değişiklik sonrası yinede hata mesajı alınıyorsa, IIS' in bir kere reset edilmesi gerekebilir.
- **varyByParam** değerini kullanmıyorsak bile **none** olarak atamalıyız*(Asp.Net tarafından bildiğimiz bir kural)*. Yapmadığımız takdirde alacağımız çalışma zamanı hata mesajı : **"The cache profile, 'ShortCache', must include value for 'VaryByParam'. "**

Gelelim çalışma zamanı sonuçlarına. **IIS** üzerine atılan **WCF** servisimizi çalıştırdıktan sonra **Sum** operasyonu için yapılan ilk **HTTP GET** talebi sonrası aşağıdaki örnek ekran görüntüsü elde edilmiştir.



Görüldüğü gibi ilk talep karşılanmıştır. Hemen zaman bilgisine dikkat edelim ve **20** saniyelik **Output Cache** süresi dolmadan önce ikinci bir talepte bulunduğumuzu hatta x ve y değerlerini farklı olarak verdiğimizi düşünelim. Bu durumda aşağıdaki sonuçlar alınacaktır.



Görüldüğü gibi istemciye gönderilen içerik değişmemiştir. Nitekim şu andaki cevap, **Asp.Net Output Cache** mekanizması tarafından karışlanmış bir önceki talep için üretilen hazır çıktıdır. Ancak **Output Cache** süresini aştıktan sonra tekrar talepte bulunursak ikinci talep için güncel sonuçları aşağıdaki örnek çıktıda görüldüğü gibi alabiliriz.

Böylece sistemin çalıştığını ispat etmiş olduk  **Output Cache** özelliği **REST** tabanlı **WCF** servislerinde, perfomansı arttırıcı bir unsur olarak görülebilir. Nitekim sunucu ve istemci arasındaki gidiş gelişlerde, sürekli hesaplanması gerekmeyen ve çoğunlukla sabit olan içeriklerin tekrardan üretilmesi yerine belirli zaman dilimleri boyunca ön bellekten karşılanması hem servisin işlem yükünü azaltacak hem de hızlı cevap verme sürelerini doğuracaktır. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

RESTSupport.rar (37,47 kb)

WCF 4.0 Yenilikleri - Automatic Help Page [Beta 1] (2009-09-17T01:02:00)

wcf,wcf 4.0,

Merhaba Arkadaşlar,

WCF 4.0 tarafında beklenen gelmesi yüksek olan yenilikleri sizlere aktarmaya çalıştığım yazılarımızın yavaş yavaş sonlarına gelmekteyiz. Elbette incelemeyemediğimiz bir çok detay var. Bunları ilerleyen dönemlerde ürün son halini alırken tartışma ve araştırma fırsatımız olacak. Bu yazımızda **WCF 4.0** tarafına entegre olarak gelen **REST** geliştirme modeline yönelik yeteneklerden bahsedeceğiz. Aslında bu yeniliklerin çoğunu **WCF Rest Starter Kit** ile birlikte, **.Net Framework 3.5** platformu üzerinde kullanabiliyoruz. Ne varki, ek bir pakete ihtiyaç duyulmadan kullanılabilen iki özellik, **WCF 4.0** içerisine entegre edilmiş durumda. Bunlardan birisi otomatik yardım sayfaları**(Automatic Help Page)**. **WCF 4.0** içerisindeki **WebServiceHost** fabrikasını kullandığımızda otomatik olarak her **RESTful** servis için gelen ve varsayılan olarak açık olan yardım sayfaları, istenirse konfigurasyon dosyasındaki bir nitelik yardımıyla kapatılabilirde.

Yardım sayfaları özellikle **HTTP** protokolünün **GET,POST,PUT** veya **DELETE** gibi metodları yardımıyla erişilen **RESTful** servis operasyonlarının tüketiciler tarafından kolayca anlaşılmasını hedeflemektedir. Nitekim, tüketici tarafını yazan geliştiricilerin bu modelindeki bir servisin operasyonlarını çağırırken, **HTTP** metoduna göre nasıl bir paket içeriği veya **URL** hazırlayıp göndermeleri gerektiğini bilmeleri gelmektedir. Otomatik olarak üretilen yardım sayfalarının tek ve yegane amacı bu ihtiyacı karşılamaktır.

Şimdi bu konuyu basit bir örnek üzerinden değerlendirmeye çalışarak sonuçları görmeyi hedefleyeceğiz. Bu amaçla **Visual Studio 2010 Beta 1** ortamında ve **.Net Framework 4.0 Beta 1** odaklı olarak hazırlayacağımız bir **WCF Service Application** üzerinden ilerliyor olacağız. Bu örnekte **System.ServiceModel.Web.Activation** isim alanında yer alan **WebServiceHostFactory** fabrikasından yararlanmayacağız*(Bir sonraki konumuz olan HTTP Cache desteğine ait örnekte ise kullanacağız)*. Servisimize ait sözleşmemizi**(Service Contract)** aşağıda görüldüğü gibi geliştirdiğimizi düşünelim.

```
using System.ServiceModel;
using System.ServiceModel.Web;

namespace GeoService
{
   [ServiceContract(Namespace="http://GeoServices/LocationService")]
```

```
    public interface ILocationService
    {
        [OperationContract]
        [WebGet]
        string FindLocation(string gsmNumber);

        [OperationContract]
        [WebGet(ResponseFormat = WebMessageFormat.Json)]
        string FindLocationInJson(string gsmNumber);

        [OperationContract]
        [WebInvoke(Method = "DELETE")]
        string Delete(string location);

        [OperationContract]
        [WebInvoke(Method="POST")]
        bool Insert(Customer customer);

        [OperationContract]
        [WebInvoke(Method = "PUT")]
        bool Move(Customer customer);
    }
}
```

Servis sözleşmesinde örnek olarak **HTTP GET, POST, PUT** ve **DELETE** metodlarına göre kullanılabilen bazı operasyonlar tanımlanmıştır. **FindLocationInJson** operasyonu, **JSON** formatında **cevaplar(Response)** üretmektedir. Bilindiği üzere **WebGet** metodu için yapılan çağrılarda **URL** satırından gelen talepler söz konusudur. **WebInvoke** niteliği ile imzalanmış operasyonlarda ise HTTP içeriğinin paket olarak gönderilmesi söz konusudur. Bu operasyonlardan **Insert** ve **Move** metodları, parametre olarak serileştirilebilir bir tip kullanmaktadır. Dolayısıyla, servisi **REST** modele göre kullanmak isteyen geliştiricilerin bu kritik noktalara göre paket içeriği veya URL bilgilerini nasıl hazırlayacaklarını bilmeleri son derece yararlı olacaktır. Gelelim servis sözleşmesini uygulayan tipimize;

```
using System;

namespace GeoService
{
    public class LocationService
        : ILocationService
    {
        public string FindLocation(string gsmNumber)
        {
```

```
            return String.Format("({0}:{1})-({2}:{3})", 36, 42, 26, 45);
        }

        public string FindLocationInJson(string gsmNumber)
        {
            return String.Format("({0}:{1})-({2}:{3})", 36, 42, 26, 45);
        }

        public string Delete(string location)
        {
            return string.Format("{0} ----> {1}",location);
        }

        public bool Insert(Customer customer)
        {
            return false;
        }

        public bool Move(Customer customer)
        {
            return true;
        }
    }

    public class Customer
    {
        public string GsmNo { get; set; }
        public string Name { get; set; }
        public string Location { get; set; }
    }
}
```

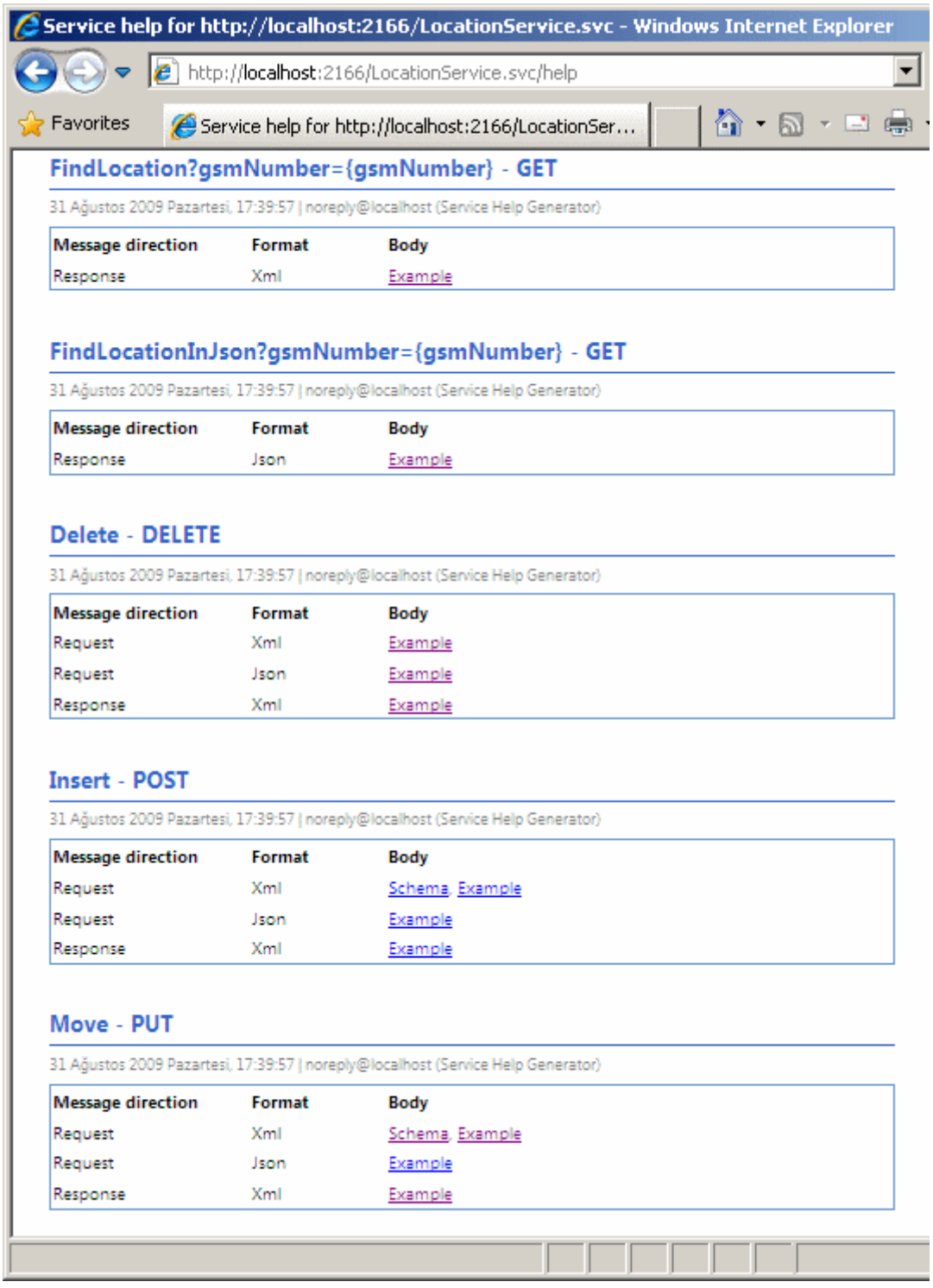
Operasyonalrın uygulanışında herhangibir özel durum söz konusu değildir aslında. Asıl üzerinde duracağımız nokta **Web.config** dosyasının içeriğidir. **Web.config** dosyasında yer alan **system.ServiceModel** element içeriğini aşağıdaki gibi düzenleyebiliriz.

```
<system.serviceModel>
    <services>
      <service name="GeoService.LocationService">
        <endpoint
address="" binding="webHttpBinding" contract="GeoService.ILocationService"/>
      </service>
    </services>
    <behaviors>
      <endpointBehaviors>
```

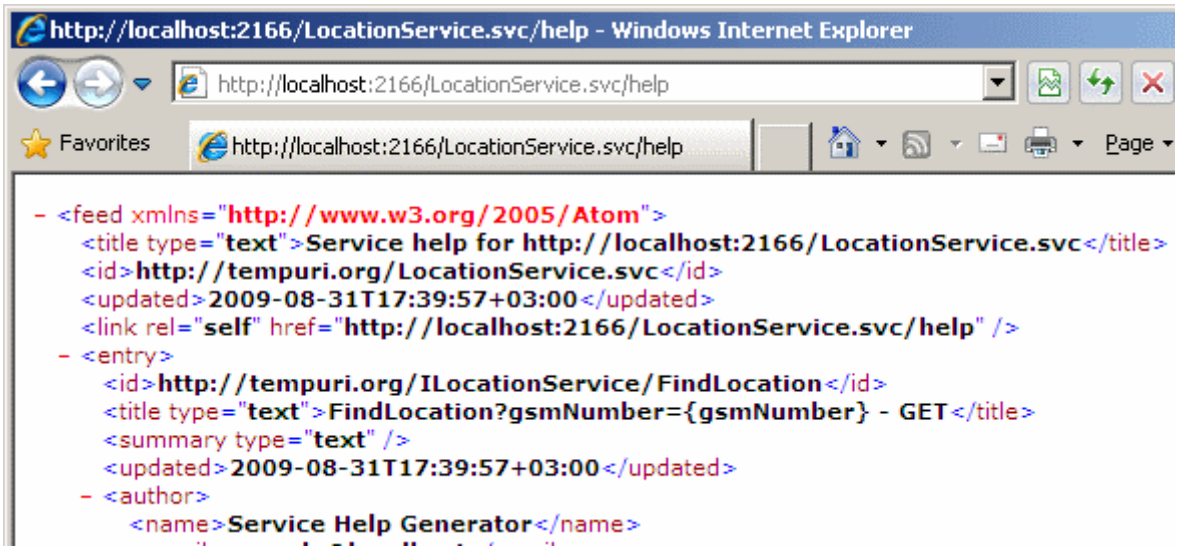
```
      <behavior>
        <webHttp enableHelp="true"/>
      </behavior>
    </endpointBehaviors>
  </behaviors>
</system.serviceModel>
```

Bağlayıcı tip(Binding Type) olarak **webHttpBinding** kullanmamızın sebebi elbetteki servisimizin **RESTful** özelliklerine göre hizmet vermesinin sağlanmasıdır. Diğer yandan **endPoint** noktasına eklenen **webHttp** isimli davranışın **enableHelp** özelliğine **true** değeri atanmıştır. Buna göre çalışma zamanında **help** takısı ile servis talep edildiğinde aşağıdaki çıktı ile karşılaşılır.

URL : http://localhost:2166/LocationService.svc/**help**

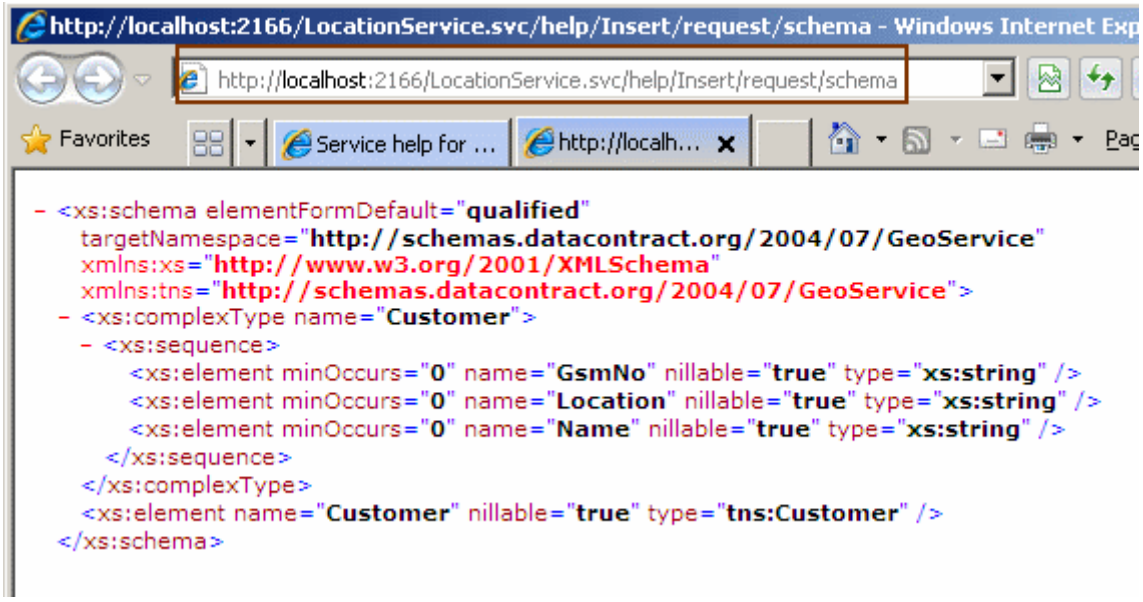


Görüldüğü üzere tüm operasyonlar için nasıl çağırılacaklarına, ne tür cevaplar döndüreceklerine dair bilgiler ve hatta kullanılan serileştirilebilir tipler varsa bunların şemalarına ait detaylar bu yardım sayfasında yer almaktadır. Tabi şu anda help sayfası için, **Internet Explorer'** ın **Feed** özelliğine göre bir çıktı elde edilmektedir. Normal şartlarda kaynağa baktığımızda aslında varsayılan olarak **ATOM** formatında bir **feed** içeriğinin üretildiği görülebilir.



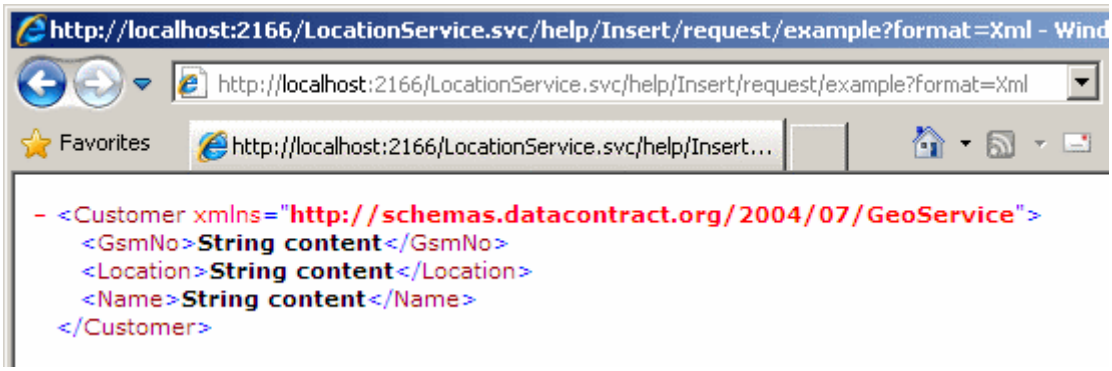
Hemen hatırlalatım örneğimizde **WebServiceHostFactory** kullanmadığımızdan, yardım sayfaları **enableHelp** niteliğine **true** değerini atamadığımız sürece çalışmayacaktır. **GET** metodlarının çağrıları dikkat edileceği üzere doğal olarak bir **URL** formatındadır. Diğer taraftan örneğin **Insert** operasyonunu çağırmak istediğimizi göz önüne alalım. Bu durumda geliştirici olarak tek yapmamız gereken **Request** ve örnek talep formatı için **Example** isimli bağlantılardan elde edilen sonuçlara bakmak olacaktır. Insert operasyonu için üretilen şema aşağıdaki gibidir.

URL : http://localhost:2166/LocationService.svc/**help/request/schema**



Dikkat edileceği üzere şema bilgisinden yararlanılarak operasyona parametre olarak nasıl bir tip gönderilmesi gerektiği, üyelerinin ne olacağı açık bir şekilde görülmektedir. **Insert**operasyonu için **Example** linkine tıkladığımızda ise örnek bir **POST** çağrısının içeriğinin nasıl olması gerektiğinin örneklendiği görülecektir.

URL :
http://localhost:2166/LocationService.svc/**help/Insert/request/example?format=Xml**

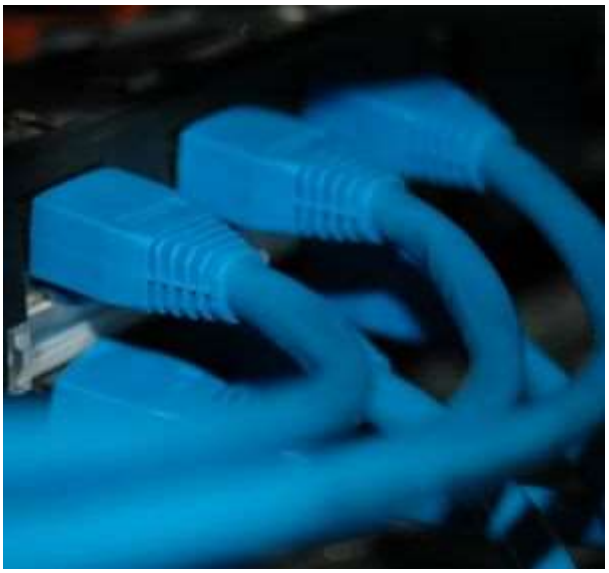


İşte bu kadar. Son derece basit bir özellik olmakla birlikte otomatik yardım sayfaları aslında tüketiciyi yazan geliştiriciler için bulunmaz bir nimettir. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HelpSupport.rar (18,55 kb)

[WCF 4.0 Yenilikleri - Routing Service - MatchAll Filtresi [Beta 1] (2009-09-14T09:00:00)]

wcf,wcf 4.0,

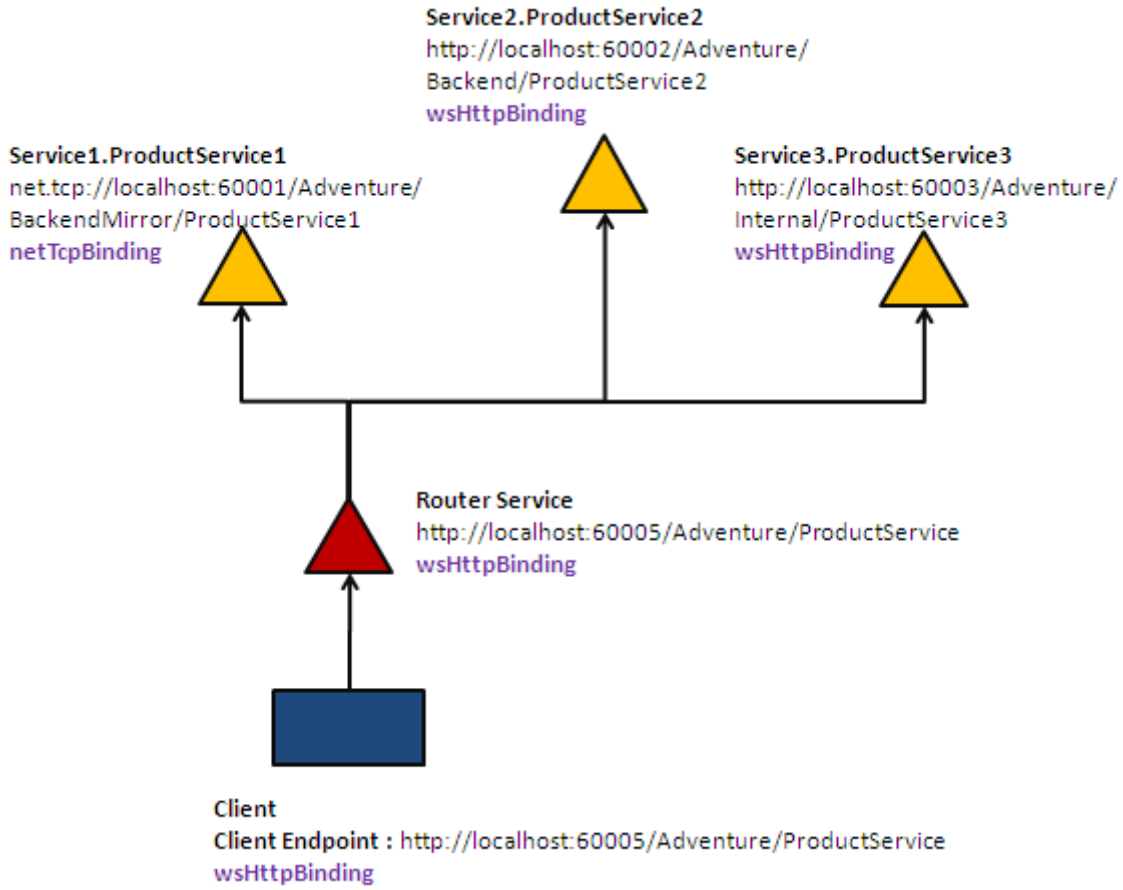


Merhaba Arkadaşlar,

Bundan önceki yazılarımızda **WCF 4.0** için **yönlendirme servislerinin(Router Service)** nasıl yazılabileceğini incelemeye çalışmıştık. Fark edeceğiniz üzere yönlendirme servislerinin en önemli noktaları arasında filtreleme tablosu ve filtrelerin olduğunu gördük. Bununla birlikte sadece**Action** tipinde bir filtreleme

kullanıp, istemciden gelen **SOAP** paketinin **Action** kısmından yararlanılarak bir yönlendirme yapılmasını inceledik. Oysaki filtreleme tipi olarak **Action** dışında, **Address, AddressPrefix, StrictAnd, EndpointName, MatchAll, XPath** gibi seçeneklerimiz de bulunmaktadır. İşte bu yazımızda **MatchAll** seçeneğini incelemeye çalışıyor olacağız. **MatchAll** seçeneğine göre, istemciden gelen mesajın içeriği ne olursa olsun, söz konusu talebin tanımlanan birden fazla **DownStream** servise yönlendirilmesi mümkündür. Ancak önemli bir kısıtlama vardır. Bu kısıtlamaya göre sadece **One-Way** veya **Duplex** modeldeki **iletişim(Communication)** desteklenir. Dolayısıyla **Request/Reply** modelde olan iletişimi ele alan tipleri yönlendirme servisi üzerinde kullanamayız. Bu kısıtlamaya rağmen bazı senaryolarda*(örneğin asenkron modellerde)*, istemciden gelen talebin birden fazla **DownStream'** e aktarılmasının **WCF 4.0** ile gelen özellikler sayesinde kolaylaştırılmış olması, geliştiriciler açısından oldukça heyecan vericidir.

öyleyse vakit kaybetmeden basit bir örnek üzerinden ilerlemeye ne dersiniz. Ben yazıyı gecenin geç bir vaktinde yazdığım için yanımda bir adet sıcak kahveyi bulundurmayı ihmal etmedim. örnek senaryomuzda aynı **servis sözleşmesini(Service Contract)** implemente eden 3 farklı alt servisimiz bulunmaktadır. **Router Service**, istemciden gelen talebi alıp ne olduğu ile ilgilenmeden doğrudan bu 3 servise aktarma işlemini üstlenmektedir. Dolayısıyla ispat etmemiz gereken noktalardan birisi, istemciden gelen talebin sonrasında operasyonun 3 servis üzerinde de çalışıyor olmasıdır. **OneWay** veya **Duplex** kısıtlamasından dolayı biz örneğimizde **OneWay** olarak imzalanmış basit bir servis operasyonu kullanıyor olacağız. öncelikle örneğimize ait mimari modelimize bir göz atalım.



Görüldüğü üzere sadece **Endpoint** tanımlamaları açısından farklı olan*(ve gerçek hayat senaryolarında istenirse farklı makinelerde bulunabilecek olan)* ama aynı sözleşmeyi uygulayan **3 Downstream** servisimiz bulunmaktadır. Servislerimizin uyguladığı sözleşme aşağıdaki kod parçasında olduğu gibidir.

```
using System.ServiceModel;
using System.Runtime.Serialization;

namespace AdventureContracts
{
   [ServiceContract(Namespace="http://adventure/productService")]
   public interface IAdventureContract
   {
      [OperationContract(IsOneWay=true)]
      void ProcessProduct(Product product);
   }

   [DataContract]
   public class Product
   {
      [DataMember]
      public int ProductId { get; set; }
```

```
        [DataMember]
        public string Name { get; set; }
        [DataMember]
        public double ListPrice { get; set; }
        [DataMember]
        public int Amount { get; set; }
    }
}
```

Servislerimizin üçünün kodlarını burada ayrı ayrı yazmamıza gerek olmadığını düşünüyorum. Nitekim hem odaklanmamız gereken nokta **Router** servis tarafıdır hemde yazımızın okunurluğunun zorlaşmaması gerekmektedir. Tabiki örneği indirip incelemenizi şiddetle öneririm. Gelelim yönlendirme servisimize. Yönlendirme servisimizin **App.config** dosyası içeriği aşağıdaki gibidir.

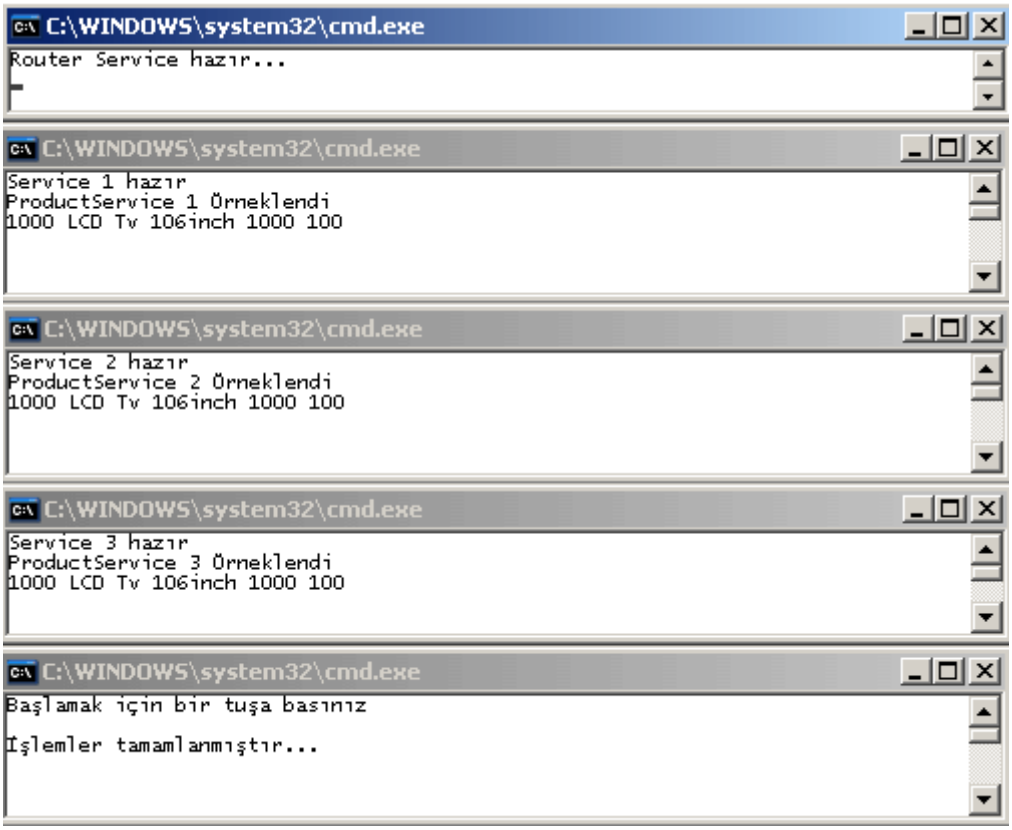
Router Service konfigurasyon içeriği(App.config);

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <routing routingTableName="RTable"/>
          <serviceDebug includeExceptionDetailInFaults="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <client>
      <endpoint
address="net.tcp://localhost:60001/Adventure/BackendMirror/ProductService1"
binding="netTcpBinding" contract="*" name="ProductServiceEndpoint1" />
      <endpoint address="http://localhost:60002/Adventure/Backend/ProductService2"
binding="wsHttpBinding" contract="*" name="ProductServiceEndpoint2"/>
      <endpoint address="http://localhost:60003/Adventure/Internal/ProductService3"
binding="wsHttpBinding" contract="*" name="ProductServiceEndpoint3" />
    </client>
    <services>
      <service name="System.ServiceModel.Routing.RoutingService">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:60005/Adventure/ProductService"/>
          </baseAddresses>
        </host>
        <endpoint binding="wsHttpBinding"
```

```
contract="System.ServiceModel.Routing.ISimplexSessionRouter"/>
    </service>
   </services>
   <routing>
    <filters>
     <filter filterType="MatchAll" name="ProductFilter"/>
    </filters>
    <routingTables>
     <table name="RTable">
      <entries>
       <add endpointName="ProductServiceEndpoint1"
filterName="ProductFilter"/>
       <add endpointName="ProductServiceEndpoint2"
filterName="ProductFilter"/>
       <add endpointName="ProductServiceEndpoint3"
filterName="ProductFilter"/>
      </entries>
     </table>
    </routingTables>
   </routing>
  </system.serviceModel>
</configuration>
```

Dikkat edileceği üzere **fiterType** niteliğine **MatchAll** değeri verilmiştir. Bu değere göre, istemciden gelecek olan **talep(Request) name** niteliğine atanan değere eş düşen **endPoint** noktalarına iletilmelidir ki bu bildirimlerde yine **entries** elementi içerisinde yapılmaktadır. **entries** elementi içerisindeki **filterName** niteliğinin değeri ile, **filter** elementi içerisindeki **name** niteliğinin değerlerinin aynı olduğuna lütfen dikkat edelim. Konfigurasyon dosyasında önem arz eden noktalardan bir diğeride, yönlendirme servisi için kullanılan sözleşme tipidir**(ISimplexSessionRouter)**. Hatırlayacağınız gibi, **Request/Reply** modelin desteklenmediğinden bahsetmiştik. Bu nedenle daha önceki örneklerimizde kullandığımız **IRequestRepyleRouter** built-in sözleşme tipini bu senaryoda kullanamayız.

örneğimizi test ettiğimizde çalışma zamanında aşağıdaki ekran görüntüsüne benzer sonuçları alırız.



Görüldüğü gibi tüm **DownStream** servisleri, istemciden gelen **Product** tipini ele almış ve basit bir şekilde kullanmıştır. Biz örneğimizde sadece gelen bilgiyi ekrana yazdırıyoruz. Aynı istemci paketinin, n sayıda **DownStream** servisi tarafından değerlendirilip üzerlerinde farklı şekillerde işlemler uygulanması söz konusu olduğunda, **MatchAll** filtereleme modelini göz önüne alabiliriz.
Tabiki **Request/Reply** kısıtlamasını unutmamak gerekir. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Router Project 3.rar (249,84 kb)

WCF 4.0 Yenilikleri - Routing Service - Hata Yönetimi [Beta 1] (2009-09-10T01:02:00)

wcf,wcf 4.0,

Merhaba Arkadaşlar,

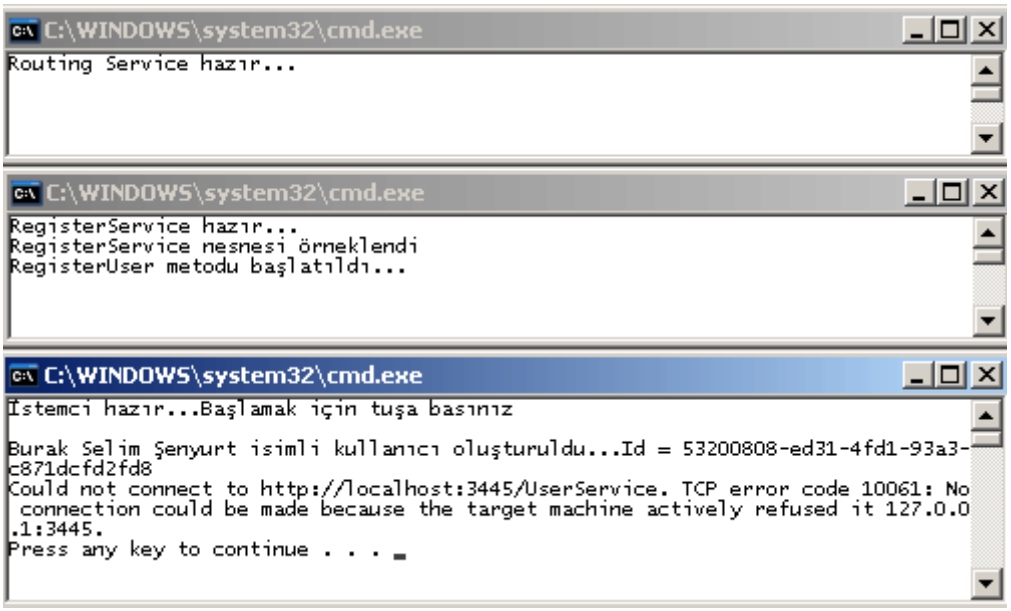
Bir önceki blog yazımızda **WCF 4.0** ile basit bir **yönlendirme servisinin(Router Service)** nasıl yazılabileceğini incelemeye çalışmıtık. Tabi bu tip bir sistemde dikkat edilmesi gereken vakalardan biriside, **Downstream** servislerde **istisnaların(Exceptions)** oluşması halinde nasıl davranılacağıdır. Peki ne gibi durumlardan bahsediyoruz? örneğin, Router servisine gelen paketin yönlendirildiği bir alt servis çalışmıyor olabilir. Bu durumda bir **TimeoutException** oluşması muhtemeldir. Benzer şekilde **CommunicationException** ve türevi olan istisna tiplerinin fırlatılmasıda söz konusudur. Bu gibi istisnaların ortaya çıkması halinde en azından işleyişin devamlılığını sağlamak ve sistemin çökmesini engellemek için, WCF 4.0 tarafı konfigurasyon dosyasında **Alternatif Endpoint** tanımlamaları yapılmasına izin vermektedir. Buna göre **Downstream** servislerinden bahsedilen tipteki istisnalardan birisi alınırsa, istemciden gelen talebin karşılanmak üzere alternatif olarak tanımlanmış olan bir servise yönlendirilmesi sağlanmış olunur. Bu alternatif servise olan yönlendirme tamamen çalışma zamanında ve router servisin yönetimi altında gerçekleşmektedir. Konuyu daha net kavrayabilmek adına bir önceki blogumuzda yazdığımız örneği aşağıdaki vakaya göre test ettiğimizi düşünelim.

İlk etapta Router Servisimiz ile tüm DownStream servislerimiz çalıştırılır. Sonrasında istemci uygulama açıkken ve henüz taleplerini iletmeden önce Downstream servislerinden herhangibiri kapatılır. örneğin **UserService** servisinin kapatıldığını düşünebiliriz. Bu durumda istemci tarafının bir**CommunicationException** istisnası ile sonlanması gerekmektedir.

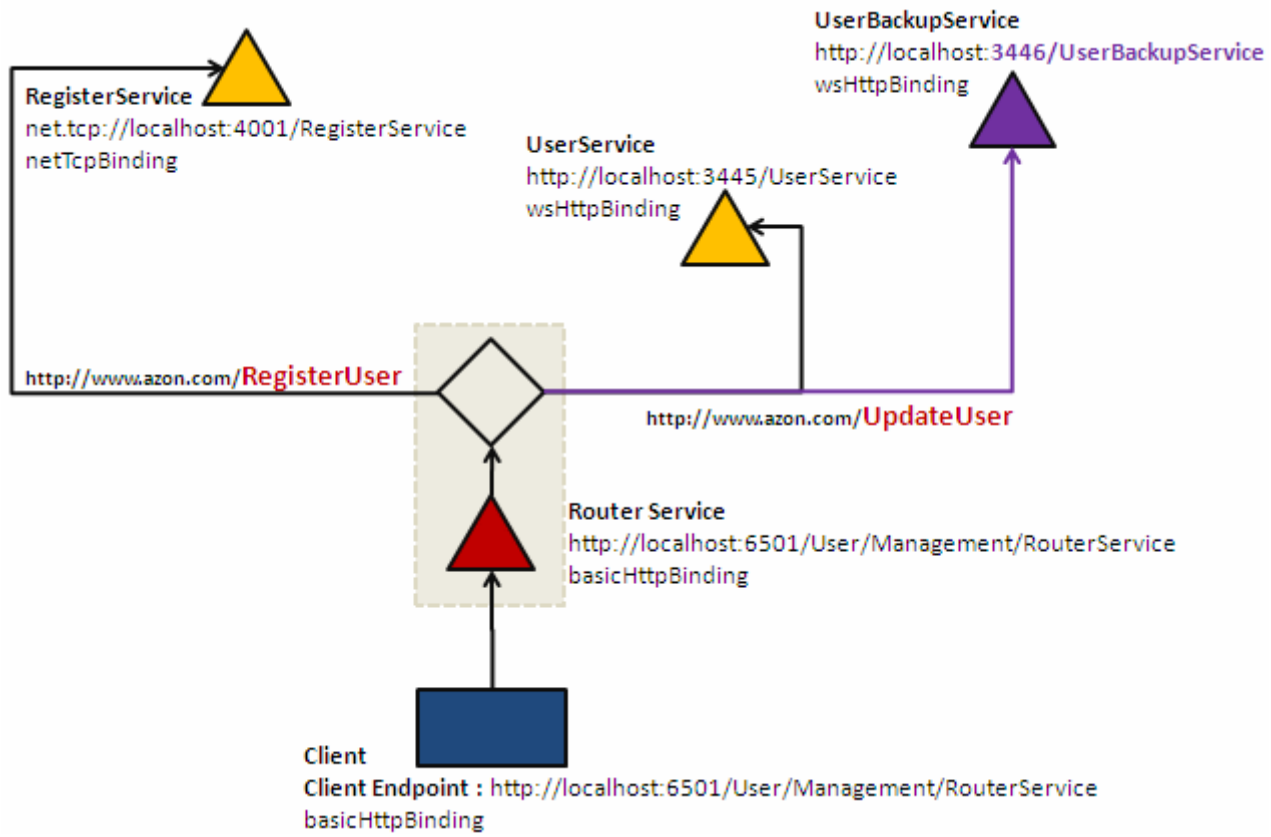
örneği kolay bir şekilde canlandırabilmek için Router servisimizde **includeExceptionDetailInFaults** özelliğine **true** değeri atayıp, istemci tarafındaki kod içeriğini ise aşağıdaki gibi güncelleştirmemiz yerinde olacaktır.

```
using System;
using System.ServiceModel;
using ClientApp.MemberManagementSpace;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("İstemci hazır...Başlamak için tuşa basınız");
         Console.ReadLine();

         try
         {
            MemberManagementServiceClient client = new
MemberManagementServiceClient();

            User burak = new User { Name = "Burak Selim Şenyurt" };

            string registerResult = client.RegisterUser(burak);
            Console.WriteLine(registerResult);

            string updateResult = client.UpdateUserName(burak, "Burki");
            Console.WriteLine(updateResult);

            Console.ReadLine();
            client.Close();
         }
         catch (CommunicationException excp)
         {
            Console.WriteLine(excp.Message);
         }
      }
   }
}
```

Test sonrasında aşağıdaki gibi bir sonuçla karşılaşmamız muhtemeldir.



Görüldüğü gibi **RegisterService** üzerinden yapılan kullanıcı kayıt operasyonu başarılı bir şekilde çalışmış ancak, **UserService** üzerinden yapılan çağrı için ortama bir **CommunicationException** döndürülmüştür. Bu son derece doğaldır, nitekim söz konusu servis kapalıdır. Peki alternatif bir yolumuz var mıdır? Yazımızın başındada belirttiğimiz gibi, **Downstream** servislerinden birisinin çökmesi halinde en azından istemci talebinin bir başka yedek servis üzerine pas edilmesi sağlanabilir. İlk önce bu **yedek servisi(Backup Service)** geliştireceğiz. Sonrasında ise, **Router** servisimize ait konfigurasyon dosyasında bazı değişiklikler yapmamız gerekmektedir. örneğimizin yeni modeli grafiksel olarak aşağıdaki gibi düşünülebilir.



UserBackupService isimli yedek servisimizin konfigurasyon dosyası ve kod yapısı aşağıdaki gibidir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="UserBackupService.UserService">
        <endpoint address="" binding="wsHttpBinding"
contract="ContractLibrary.IManagementContract" />
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:3446/UserBackupService" />
          </baseAddresses>
        </host>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

ve kod içeriği;

```
using System;
using ContractLibrary;
using System.ServiceModel;

namespace UserBackupService
{
   class UserService

      : IManagementContract
   {
      public string RegisterUser(User newUser)
      {
         throw new NotImplementedException();
      }

      public string UpdateUserName(User oldUser, string newName)
      {
         Console.WriteLine("UpdateUserName metodu başlatıldı...");
         return String.Format("{0} isimli kullanıcı adı {1} olarak değiştirildi",
oldUser.Name, newName);
      }

      public UserService()
      {
         Console.WriteLine("UserBACKUPService nesnesi örneklendi");
      }
   }

   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(UserService));
         host.Open();

         Console.WriteLine("UserBACKUPService hazır...");
         Console.ReadLine();

         host.Close();
      }
   }
}
```

Aslında **UserService'** in bire bir kopyası olan sadece farklı bir port üzerinden sunulan bir servis geliştirmiş bulunuyoruz. Elbetteki bu servisi farklı bir makine üzerinde

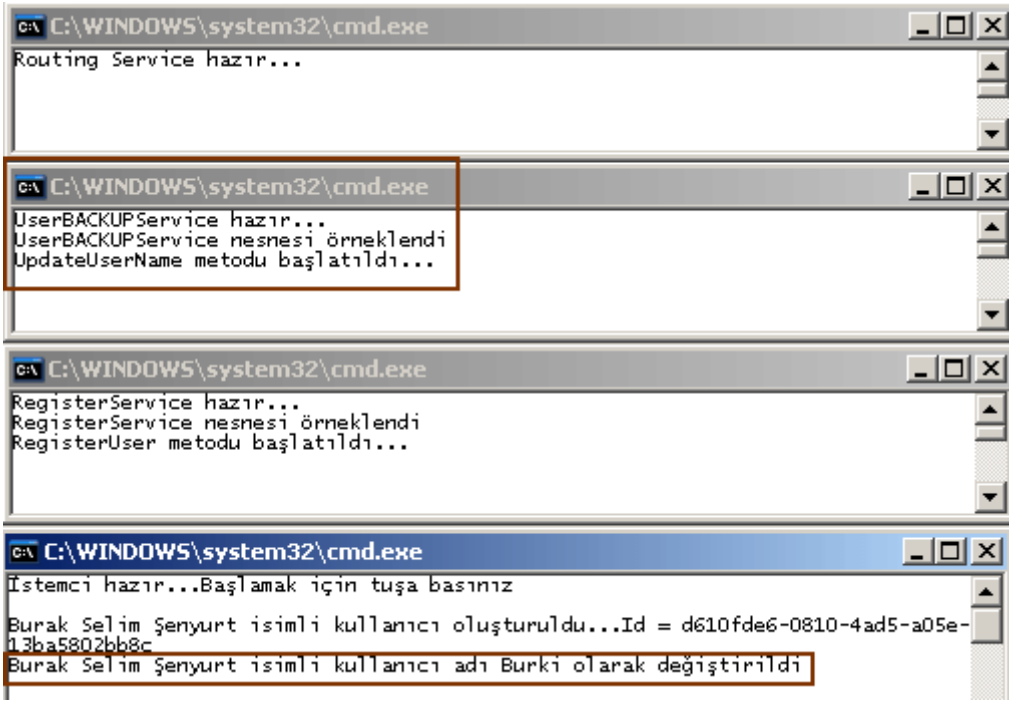
farklı **Endpoint** kuralları ilede sunabilir ve alternatif Endpoint olarak kullanabiliriz. Gelelim bu yazının en can alıcı noktasına. Router servisine ait konfigurasyon dosyasının içeriği...

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <routing routingTableName="RTable"/>
          <serviceDebug includeExceptionDetailInFaults="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <client>
      <endpoint address="http://localhost:3445/UserService" binding="wsHttpBinding"
contract="*" name="UserServiceEndpoint" />
      <endpoint address="http://localhost:3446/UserBackupService"
binding="wsHttpBinding" contract="*" name="UserBackupServiceEndpoint"/>
      <endpoint address="net.tcp://localhost:4001/RegisterService"
binding="netTcpBinding" contract="*" name="RegisterServiceEndpoint" />
    </client>
    <services>
      <service name="System.ServiceModel.Routing.RoutingService">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:6501/User/Management/RouterService"/>
          </baseAddresses>
        </host>
        <endpoint binding="basicHttpBinding"
contract="System.ServiceModel.Routing.IRequestReplyRouter"/>
      </service>
    </services>
    <routing>
      <filters>
        <filter name="RegisterFilter" filterType="Action"
filterData="http://www.azon.com/Registration"/>
        <filter name="UpdateUserFilter" filterType="Action"
filterData="http://www.azon.com/UpdateUser"/>
      </filters>
      <routingTables>
        <table name="RTable">
          <entries>
            <add filterName="RegisterFilter" endpointName="RegisterServiceEndpoint"/>
```

```
          <add filterName="UpdateUserFilter"
endpointName="UserServiceEndpoint" alternateEndpoints="alternateEndpointList"/>
        </entries>
      </table>
    </routingTables>
    <alternateEndpoints>
      <list name="alternateEndpointList">
        <endpoints>
          <add endpointName="UserBackupServiceEndpoint"/>
        </endpoints>
      </list>
    </alternateEndpoints>
  </routing>
 </system.serviceModel>
</configuration>
```

İlk etapta, **Backup** servisi içinde bir **Endpoint** bildirimi yapıldığı ve yedek servisin işaret edildiği farkedilmektedir. Diğer yandan **routingTables** içerisinde yapılan **entries** bildirimlerinden**UpdateUserFilter** isimli olanında **alternateEndpoints** isimli bir **nitelik(attribute)** dikkati çekmektedir. Bu nitelik, dosyanın ilerleyen kısımlarında yer alan **alternateEndpoints** elementi altındaki listeyi işaret etmektedir. Bu liste içerisinde **n** sayıda alternatif **endPoint** ismi belirtilebilir. Bir başka deyişle bir **endPoint'** in karşılayamadığı istekleri, birden fazla **endPoint** noktasına denenmek üzere aktarabiliriz. Tabi bu durumu henüz test etme şansım olmadı. Ki beklenen sırasıyla servislerin denenmesi ve başarılı olandan sonrakilere geçilmemesi yönünde olmalıdır. Ancak entries/add elementi içerisinde priority isimli bir nitelikte bulunmakta ve öncelik seviyesini belirlemektedir. İşte size bir garajda araştırma ödevi.😎

Artık vakamızı tekrardan test edebiliriz. Yine tüm servisleri(Backup servisimiz dahil) çalıştıracak, ancak istemci talepte bulunmadan önce **UserService'** ini kapatacağız. İşte sonuçlar;



Görüldüğü gibi, **UserService'** in kapalı olması ve **Exception** üretmesi durumunda, **Router** servisimiz talebi bu kez alternatif **endPoint** listesinde belirtilen UserBackupService isimli yedek servise doğru yönlendirmiş ve istemcinin talebinin buradan karşılanmasını sağlamıştır. Tabiki burada ele alınan **alternatif Endpoint'** lerin işaret ettiği servisler farklı makinelereden, farklı**bağlayıcı tiplerle(Binding Types)**, farklı iletişim protokolleri ile dağıtılabilir. Bu tamamen yedek servis stratejimize bağlıdır. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Router Project 2.rar (128,25 kb)

Kitap - SQL Server 2008 ve Veritabanı Programlama(Yaşar Gözüdeli) (2009-09-03T09:00:00)

Merhaba Arkadaşlar,

Beni yakından takip edenler tam anlamıyla bir .Net Developer olduğumu, olmakta ısras ettiğimi 🙂, SQL konusunu genellikle uzmanlarına bıraktığımı, bırakmak istediğimi bilirler. Ancak yazılım geliştirme işindeyseniz ve Türkiye' de yaşıyorsanız **Uzmanlık** kavramının çok farklı uygulanabildiğini ve pek çok şirkette olmadığını görebilirsiniz.

Bu anlamda bir .Net geliştiricisinin yeri geldiği zaman SQL üzerinde de çok iyi işler çıkartması beklenebilir. Ki bu benim her zaman karşı olduğum durumlardan birisidir. Bir yazılım şirketinde Database Developer, Database Admin gibi pozisyonlar olmalıdır ve örneğin bir uygulamanın veritabanı modellemesi veya tasarımı gibi işlemlerini üstlenmelidir.*(Bakınız MSF modelinde bile Database Developer, Database Administrator rolleri vardır)*

Ancak bu durumda dahi bir .Net geliştiricisinin, veritabanı geliştiricileri veya yöneticilerinin en azından ne dediklerini anlaması ve yeri geldiğinde önerilerini, taleplerini onların anlayacağı şekilde ifade edebilmesi(ve bazen yazabilmesi) de önemlidir. İşte bu açığı kapatmak için size değerli yazarlarımızdan Yaşar Gözüdeli' nin SQL Server 2008 & Veritabanı Programlama kitabını tavsiye etmek isterim. Türkçe olarak yazılmış başarılı SQL Server kaynaklarından birisi olarak karşımıza çıkan bu kitabı ben çalışma masamdan eksik etmiyorum. Hatta SQL konusuna bulaşmak konusunda tereddütlerim olsada her zaman, bu kitabı alıp bir süre boyunca ciddi anlamda okuduğumu itiraf edeyim.

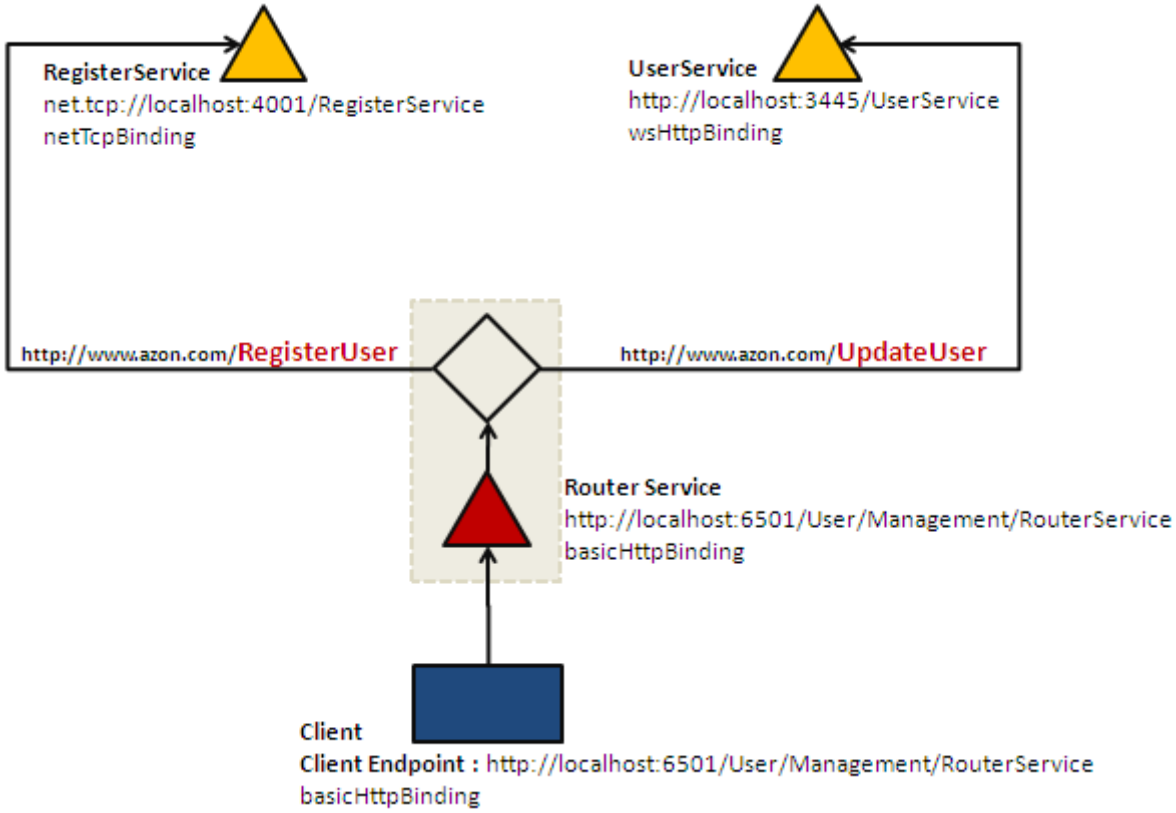
Yaklaşık olarak 670 sayfalık bu kitabın içerisinde aynı zamanda örnek görsel dersler, pdf ve uygulamalarıda içeren bir CD' de yer almaktadır. Şu anda 4ncü baskısını tedarik ettiğim bu güzel kitap için Yaşar Gözüdeli' ye ne kadar teşekkür etsek azdır diye düşünüyorum.

WCF 4.0 Yenilikleri - Routing Service Geliştirmek - Hello World [Beta 1] (2009-08-27T03:03:00)

wcf,wcf 4.0,

Merhaba Arkadaşlar,

Routing Service konusu ile ilişkili bir önceki yazımızda modelin sunduğu alt yapıya kısaca değinmeye çalışmış ancak bir örnek geliştirme girişiminde bulunmamıştık. Bu yazımızda ise bir **Hello World** örneğini geliştirmeye çalışacağız. *(örneğimizi **.Net Framework Beta 1** ve **Visual Studio 2010 Beta 1** ile geliştirdiğimizi bir kere daha hatırlatmak isterim.)* İlk olarak sizlere, örnek senaryomuzdan bahsetmek isterim. **Router** servisimizin arkasında genellikle **Downstream** olarak adlandırılan servislerimiz yer almaktadır. Bu servislerden birisi, kullanıcı kayıt işlemlerini**(Register)** üstlenirken, diğeride kullanıcı adını güncelleştirme işlemini ele almaktadır.*(Tabiki bu örnekteki amaç yönlendirme servisini devreye almak olduğundan sadece iki basit operasyonun, farklı servislere dağılması üzerine yoğunlaşılmıştır)* İstemci uygulama, **Router** servisi üzerinden yeni bir kullanıcıyı kayıt etmek veya güncellemek ile ilişkili işlemler için talepte bulunabilir. Gelen talep aslında bir **SOAP** paketidir. Geliştirdiğimiz örnekte, **Router** servis tarafında yer alan filtreleme içerisinde, **SOAP Action**içeriğine göre bir ayrıştırma yapılacak ve arka planda uygun olan servis metodlarına yönlendirme işlemi gerçekleştirilecektir. Buna göre **Router** servisimizin, istemciden gelen paketin **Action**değerine bakarak bir karar vereceğini söyleyebiliriz. Elbette bunu birde programatik ortamda söyleyebilmemiz gerekmektedir örneği tamamladıktan sonra oluşturduğumuz mimari tasarıma bakarsak*(ki burada yazımızın başında veriyorum)*, ne yapmak istediğimizi daha net görebiliriz.



Downstream servislerimizden olan **RegisterService** hizmetine **TCP** bazlı bir iletişim ile erişilebilmektedir. **UserService** isimli diğer servisimiz ise **Ws HTTP** protokolüne göre hizmet sunmaktadır.**RouterService** isimli yönlendirme servisimiz ise **Basic HTTP** tabanlı bir iletişim kanalı sağlamaktadır. Dikkat edileceği üzere **RouterService** üzerinden ayrılan iki dalın içerisindeki **URL**adresleri birbirlerinden farklıdır. Bu adresler aslında, **RegisterService** ve **UserService** isimli hizmetlerin ortaklaşa uyguladıkları **servis sözleşmesi(Service Contract)** içerisinden tanımlanan**Action** değerleridir. Dolayısıyla işe ilk olarak her iki servisinde uyguladığı ortak sözleşmeyi tasarlayarak başlamamız gerekmektedir. Söz konusu sözleşme ContractLibrary isimli bir sınıf kütüphanesi içerisinde tanımlanmış olup sadece Downstream servisler tarafından kullanılmaktadır. İşte **servis sözleşmemiz(Service Contract)**.

```
using System.ServiceModel;
using System.Runtime.Serialization;

namespace ContractLibrary
{
    // Downstream servislerin tamamının uygulayacağı ortak servis sözleşmesi
    // Namespace elementinin içeriği filtrelemelerde Action kısmına yazılacak bilgiler için önem arz etmektedir.
    [ServiceContract(Name="MemberManagementService",Namespace="http://www.azon.com/Membership/Management")]
    public interface IManagementContract
    {
```

```
        // Action değerlerini biz belirliyoruz
        [OperationContract(Action =
"http://www.azon.com/Registration",ReplyAction="http://www.azon.com/RegistrationResponse")]
        string RegisterUser(User newUser);

        [OperationContract(Action = "http://www.azon.com/UpdateUser", ReplyAction
= "http://www.azon.com/UpdateUserResponse")]
        string UpdateUserName(User oldUser, string newName);
    }

    [DataContract]
    public class User
    {
        [DataMember]
        public string Name { get; set; }
        [DataMember]
        public string Id { get; set; }
    }
}
```

IManagementContract isimli **arayüz(Interface)** içerisinde tanımlanan **RegisterUser** ve **UpdateUserName** metodlarının **OperationContract** nitelikle rine dikkat edilmelidir. Bu niteliklerde yer alan **Action**ve **ReplyAction** değerleri ile, **SOAP** paketleri ve **WSDL** içerisindeki bazı tanımlamalar doğrudan etkilenmektedir. çok doğal olarak, **Router** servisi içerisindeki filtreleme tablosunda yer alan **Action**kriterlerinde, bu operasyonlar için tanımlanan **Action** değerleri ele alınmalıdır. Şimdi sırasıyla **RegisterService** ve **UserService** servislerinin olduğu örnek Console projelerimizi geliştirelim.

UserManagementService projesine ait **App.Config** dosyası;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="UserManagementService.UserService">
        <endpoint address="" binding="wsHttpBinding"
```

```
contract="ContractLibrary.IManagementContract" />
      <endpoint address="Mex" kind="mexEndpoint" />
      <host>
        <baseAddresses>
          <add baseAddress="http://localhost:3445/UserService" />
        </baseAddresses>
      </host>
    </service>
  </services>
 </system.serviceModel>
</configuration>
```

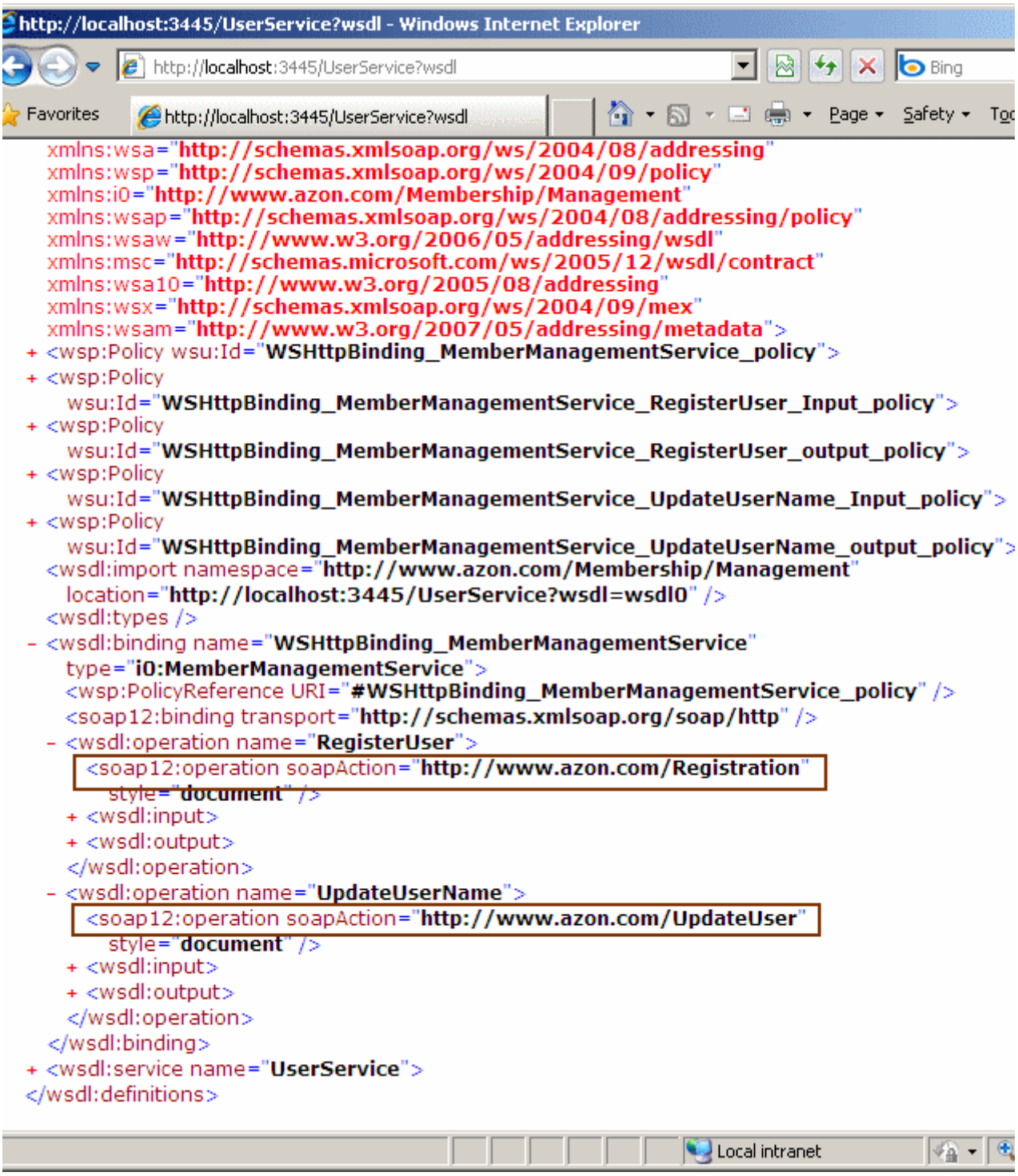
wsHttpBinding bağlayıcı tipini kullanılan bu servisin üzerinden standart **mexEndpoint** yardımıyla **WSDL** çıktısıda**(Metadata Publishing)** sağlanmaktadır. Aslında bu bir zorunluluk değildir. örnekte bu özelliği açmamın nedeni, istemci için gerekli olan **proxy** tipinin üretimini kolaylaştırmaktır; ki **proxy** tipini ürettikten sonra istemcinin **config** dosyasını da çok farklı bir şekilde değerlendirdiğimizi söyleyebilirim. Nitekim, istemci uygulama **Downstream** servislerine değil, **Router** servisine talepte bulunmalıdır.

UserManagementService projesine ait **UserService** sınıfı;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ContractLibrary;
using System.ServiceModel;

namespace UserManagementService
{
   // Senaryoya göre RegisterService sadece RegisterUser operasyonunu üstlenmek üzere tasarlanmıştır.
   class UserService
      : IManagementContract
   {
      public string RegisterUser(User newUser)
      {
         throw new NotImplementedException();
      }

      public string UpdateUserName(User oldUser, string newName)
      {
         Console.WriteLine("UpdateUserName metodu başlatıldı...");
         return String.Format("{0} isimli kullanıcı adı {1} olarak değiştirildi",
```

```
oldUser.Name, newName);
        }

        public UserService()
        {
            Console.WriteLine("UserService nesnesi örneklendi");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ServiceHost host = new ServiceHost(typeof(UserService));
            host.Open();

            Console.WriteLine("UserService hazır...");
            Console.ReadLine();

            host.Close();
        }
    }
}
```

UserService sınıfı, **ContractLibrary** sınıf kütüphanesi içerisinde yer alan **IManagementContract** isimli servis sözleşmesini implemente etmektedir. Dikkat edileceği üzere **RegisterUser**metodu için bir implemantasyon gerçekleştirilmemiş ve hatta bilinçli olarak **NotImplementedExcetion** tipinden bir **istisna(Exception)** nesnesi fırlatılmıştır. Nitekim bu istisna mesajını hiç almayacağımızı garanti edebilirim. Servisimizi tamamladıktan sonra çalıştırıp **WSDL** çıktısına baktığımızda aşağıdaki ekran görüntüsünde yer alan sonuçlar ile karşılaşırız.



Görüldüğü gibi kutucuk içine alınan bölgelerde, **OperationContract** niteliğinin **Action** özelliklerine atanan değerler yer almaktadır. Ben örnekte ilerlerken tam bu noktada bir istemci uygulama oluşturup **Add Service Reference** ile söz konusu **WSDL** çıktısının karşılığı olan **Reference.cs** dosyasının ürettirilmesini tercih ettim. Ancak sonrasında istemci uygulama tarafında sadece **Reference.cs** dosyasının içeriğini bıraktım. Yani **config** dosyasının içeriğini ve **Service Reference** klasörünün tamamını(Reference.cs hariç) sildim. Bu durumda istemci tarafının **RegisterUser** ve **UpdateUserName** metodlara çağrı

yapabilmesi **managed** olarak mümkün hale geldi. Her neyse...Vakit kaybetmeden **InternalService** isimli **Console** uygulamamızı ve **RegisterService** isimli servisimizi tasarlayarak yolumuza devam edelim.

InternalService projesine ait **App.config** içeriği;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="InternalService.RegisterService">
        <endpoint address="" binding="netTcpBinding"
contract="ContractLibrary.IManagementContract" />
        <host>
          <baseAddresses>
            <add baseAddress="net.tcp://localhost:4001/RegisterService" />
          </baseAddresses>
        </host>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

RegisterService isimli servis **netTcpBinding** bağlayıcı tipini kullanmakla birlikte aynen **UserService'** te olduğu gibi **ContractLibrary** sınıf kütüphanesi içerisindeki **IManagementContract**arayüzünü uygulamaktadır.

InternalService projesinde yer alan **RegisterService** sınıfı içeriği;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ContractLibrary;
using System.ServiceModel;

namespace InternalService
{
    // Senaryoya göre RegisterService sadece RegisterUser operasyonunu üstlenmek üzere
tasarlanmıştır.
    class RegisterService
        :IManagementContract
    {
        public string RegisterUser(User newUser)
        {
```

```
            Console.WriteLine("RegisterUser metodu başlatıldı...");
            return String.Format("{0} isimli kullanıcı oluşturuldu...Id = {1}", newUser.Name,
Guid.NewGuid().ToString());
        }

        public string UpdateUserName(User oldUser, string newName)
        {
            throw new NotImplementedException();
        }

        public RegisterService()
        {
            Console.WriteLine("RegisterService nesnesi örneklendi");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ServiceHost host = new ServiceHost(typeof(RegisterService));
            host.Open();

            Console.WriteLine("RegisterService hazır...");
            Console.ReadLine();

            host.Close();
        }
    }
}
```

Bu kez, **RegisterUser** metodu uygulanmış ancak **UpdateUserName** metodu içerisinden **NotImplementedException** istisna örneğinin fırlatılması sağlanmıştır.

Artık yönlendirme servisinin yazılmasına başlanabilir. Yönlendirme servisi için belkide en önemli nokta konfigurasyon içeriğidir. Bununla birlikte yönlendirme servisinin, **Downstream** servislerine ait referansları bilinçli olarak**(örneğin Add Service Reference yardımıyla)** eklemesine de gerek yoktur. Sadece **Client Endpoint** tanımlamalarını yapması yeterlidir. Bir başka deyişle hangi servise, hangi mesajlaşma tipi ile erişeceğini bilmesi yeterlidir. İşte yazımızın kalbini oluşturan yere geldik...

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
```

```
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <routing routingTableName="RTable"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <client>
      <endpoint address="http://localhost:3445/UserService"
binding="wsHttpBinding" contract="*" name="UserServiceEndpoint" />
      <endpoint address="net.tcp://localhost:4001/RegisterService"
binding="netTcpBinding" contract="*" name="RegisterServiceEndpoint" />
    </client>
    <services>
      <service name="System.ServiceModel.Routing.RoutingService">
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:6501/User/Management/RouterService"/>
          </baseAddresses>
        </host>
        <endpoint binding="basicHttpBinding"
contract="System.ServiceModel.Routing.IRequestReplyRouter"/>
      </service>
    </services>
    <routing>
      <filters>
        <filter name="RegisterFilter" filterType="Action"
filterData="http://www.azon.com/Registration"/>
        <filter name="UpdateUserFilter" filterType="Action"
filterData="http://www.azon.com/UpdateUser"/>
      </filters>
      <routingTables>
        <table name="RTable">
          <entries>
            <add filterName="RegisterFilter"
endpointName="RegisterServiceEndpoint"/>
            <add filterName="UpdateUserFilter"
endpointName="UserServiceEndpoint"/>
          </entries>
        </table>
      </routingTables>
    </routing>
  </system.serviceModel>
</configuration>
```

Aslında bu konfigurasyon içeriğine bir kaç dakika gözle bakmakta ve kafamızda gerekli bağlantıları yaparak neyin ne olduğunu anlamaya çalışmakta yarar olduğu kanısındayım. İlk olarak bir **routing** davranışını belirlendiğini hemen görebiliriz. Bu davranışın **routingTableName** niteliğine atanan değer ile, hangi filtreleme tablosuna bakılacağı belirlenmektedir. Yönlendirme ile ilgili eşleştirmelerin tamamı, **routing** elementi içerisinde yapılır. Dikkat edileceği üzere iki adet filtre belirlenmiştir. Bunların her ikiside **Action** tipindedir. Yani **filterData**niteliğine atanan değer, gelen taleplerin **SOAP Action** kısımlarında aranır. Peki bulunduklarında ne olur? **table** elementi altında yer alan **entries** alt boğumunda, bir filtrenin belirttiği kritere uyulması halinde hangi istemci **endPoint** noktasının devreye sokulacağı belirtilmektedir. çok doğal olarak çalışma zamanı, gelen **Action** bilgisinin eş düştüğü **Endpoint'** i bulduktan sonra, yönlendirmeyi hangi **Downstream** tipine doğru yapacağını kolayca bilecektir. önemli olan noktalardan bir diğeride, servisin **Endpoint** bilgisidir. Burada görüleceği üzere daha önceki yazımızdan da hatırlayacağınız **built-in routing** sözleşmelerinden birisi seçilmiştir. Buna göre operasyonlarımızda **request/reply** modeli söz konusu olduğundan **IRequestRepyleRouter**servis sözleşmesinden yararlanılmaktadır. Artık yönlendirme servisinin kodlarını aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Routing;

namespace Router
{
   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(RoutingService));
         host.Open();
         Console.WriteLine("Routing Service hazır...");
         Console.ReadLine();
         host.Close();

      }
   }
}
```

Tek dikkat edilmesi gereken, **System.ServiceModel.Routing assembly'** ından gelen **RoutingService** tipinin kullanılmış olmasıdır. Bu sayede çalışma zamanında yönlendirme işlemleri için gerekli alt yapının oluşturulması sağlanacaktır. Artık geriye istemci tarafını tamamlamaktan başka bir şey kalmamaktadır. Yupiiii!!! 😎 İşte istemci tarafının **App.config** dosyası içeriği;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
     <client>
       <endpoint address="http://localhost:6501/User/Management/RouterService"
             contract="MemberManagementSpace.MemberManagementService"
             binding="basicHttpBinding"/>
     </client>
   </system.serviceModel>
</configuration>
```

Görüldüğü üzere **Endpoint** tanımlamasında, **RoutingService** adresi belirlenmiş ve sözleşme tipi olarak daha önceden projeye eklediğimiz **Reference.cs** içerisine otomatik olarak üretilen**MemberManagementSpace.MemberManagementService** atanmıştır.

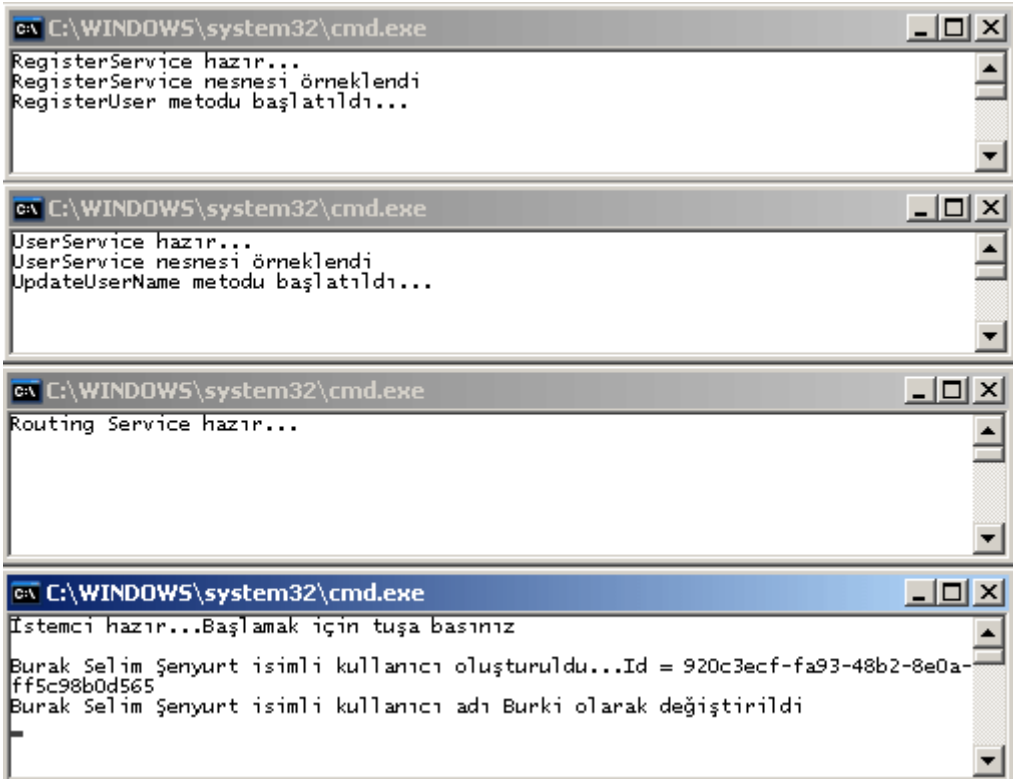
ve **Main** metodu kodları;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ClientApp.MemberManagementSpace;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("İstemci hazır...Başlamak için tuşa basınız");
         Console.ReadLine();

         MemberManagementServiceClient client = new
MemberManagementServiceClient();

         User burak = new User { Name = "Burak Selim Şenyurt" };

         string registerResult=client.RegisterUser(burak);
         Console.WriteLine(registerResult);

         string updateResult=client.UpdateUserName(burak, "Burki");
         Console.WriteLine(updateResult);

         Console.ReadLine();
         client.Close();
```

```
        }
    }
}
```

Görüldüğü gibi istemci uygulama çok normal olarak servislerin uyguladığı ortak sözleşme şemasına sadık kalacak şekilde taleplerde
bulunur. **RegisterUser** ve **UpdateUserName**metod çağırıları. Ancak hangisi olursa olsun, tüm bu operasyon çağrılarına ilişkin talepler **Routing Servisine** uğrayacaktır. Görüldüğü gibi istemcinin, **Downstream** servislerini bilmesine gerek yoktur. Zaten istemci uygulamada, söz konusu **Downstream** servislerine ait referansların ve **config** içeriğinin olmayışı bunu kanıtlamaktadır. **Action** talepleri, yönlendirme servisi tarafından değerlendirilip alt servislere iletildikten sonra, üretilen cevaplar yine **Routing** servisi üzerinden istemci tarafına gönderilir.

Artık örneği test etmeye ne dersiniz? İşte benim aldığım sonuçlar;



Umarım sizlerde benzer sonuçları elde edebilirsiniz. Herşey yolunda görünüyor.

örnekte özetle neler yaptık?

A	**DownStream Services**	
	UserManagementService	
	Adres	http://localhost:3445/UserService

	Binding	wsHttpBinding
	Metadata Publishing	true *(Sadece istemci için gerekli Reference içeriğinin kolay elde*
	Sözleşme	IManagementContract *(ContractLibrary içerisindeki servis sözl*
	Action	http://www.azon.com/UpdateUser
	InternalService	
	Adres	net.tcp://localhost:4001/RegisterService
	Binding	netTcpBinding
	Sözleşme	IManagementContract *(ContractLibrary içerisindeki servis sözl*
	Action	http://www.azon.com/RegisterUser
B	**Router Service**	*(DownStream servislerinin Endpoint bilgilerini barındırır ama*
	Service Endpoint Adresi	http://localhost:6501/User/Management/RouterService
	Binding	basicHttpBinding
	Client Endpoint Adresleri	http://localhost:3445/UserService net.tcp://localhost:4001/RegisterService
C	**Client Uygulama**	
		Sadece Action bilgilerine sahip olan proxy tipini içerir.
		Proxy tipinin üretimi için UserManagementService üzerinden a oluşturulan sınıfın istemci tarafına verilmesi yolu da tercih edile
		Downstream servislerin ait Endpoint bilgilerini içermez, bunun
		Proxy nesnesi üzerinde Register ve UpdateUser çağrılarını, Rou

Elbetteki bu örnekte en kritik noktalardan birisi filtrelemelerdir. Biz örneğimizde **Action** içeriğine bakarak bir filtreleme işlemi gerçekleştirdik. Ancak **XPath** kullanımı gibi senaryolarında mümkün olduğundan bahsetmiştik. Yani talebe ait içerik üzerinden XPath sorguları ile koşula uyan durumlarıda yönlendirme işlemlerinde kullanabiliriz. Bu gibi ince noktalarıda ilerleyen yazılarımızda sizlere aktarmaya çalışıyor olacağım. Şimdilik bu kadar. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Router Project.rar (104,09 kb)

Merhaba Bing API 2.0 (2009-08-25T23:15:00)

xml web services,c#,

Merhaba Arkadaşlar,

Bir süredir **WCF 4.0** ile birlikte gelen yenilikleri sizlere aktarmaya çalışıyorum. Son olarak **Routing Service** ile ilişkili bir giriş yazımız olmuştu. Bu konu ile ilişkili örnek en kısa sürede sizlerle olacak. Ne varki konu biraz zorlu. Bu yüzden bende yüksek lisans eğitimi aldığım yıllarda çok sevgili hocam **Halil Seyidoğlu'** nun bir açıklamasını uygulamaya karar verdim. Kendisi bize **"Bilimsel Araştırma ve Yazma"**dersinde şöyle seslenmişt; **"Bir tez konusunu araştırırken çok zorlu yollardan geçersiniz. Tezin bir noktasında tıkandınız mı? O zaman ara verin...Tatile çıkın...Bir süreliğine uzaklaşın..."**

Her ne kadar **WCF 4.0** ile gelen yenilikleri araştırmak bir tez hazırlamak kadar zorlu ve çetin olmasada sıkıldığım noktada hemen bir kaçış aradım ve bakım ne buldum.

Bing API 2.0

Microsoft' un arama motoru **Bing'** i duymayan olmamıştır sanırım. Peki **Bing'** in kendi uygulamalarımızda kullanılabilmesi için dışarıya bir **API** sunduğunu biliyor muydunuz? Ta ta ta taaaa... İşin içerisinde bir developer API'si, helede servis bazlı bir sunum olunca, değmeyin keyfime dedim ve yola koyuldum. Dolayısıyla bu yazımda sizlere Bing API' si ile ilişkili ilk izlenimlerimi ve çıkarımlarımı aktarmaya çalışacağım.

Bing API' si, kendi web sitesinden sunduğu arama özelliklerinin tamamını, farklı iletişim protokollerine göre istemci tarafına servis bazlı olarak sunmaktadır. Buna göre dilersek **Bing** üzerinden gerçekleştirilen arama kabiliyetlerini ve sonuçlarını, kendi uygulamalarımıza entegre ederek kullanabiliriz. **Bing** hizmetinden yararlanabilmek için öncelikli olarakhttp://www.bing.com/developers/ adresindeki formu doldurmamız ve yeni bir **App Id** almamız gerekmektedir. Nitekim Live servisi ile olan haberleşmede **App Id** değerinden yararlanılmaktadır. Teori oldukça basittir. Arama kutucuğundan yapılan kabliyetleri, kendi uygulamamızdan bir şekilde request olarak göndermemiz gerekmektedir. Bu noktada aslında, **Bing API** ile neler yapabileceğimiz kararının nasıl verildiğine bakmamızda yarar vardır. Söz konusu karar verilirken aslında aramanın

tipini/modelinide belirlemiş oluruz. Yada var olan aramayı genişletmiş oluruz. İşte burada a bahsedilen arama modelleri belirlenirken **SourceTypes** isimli tip değerlerinden yararlanılmaktadır. **SourceTypes'** ın değerleri **managed code** tarafında verilebileceği gibi, örneğin **HTTP Get**metoduna bağlı olarak **URL** formatında da yazılabilir.
Genel **SourceTypes** değerleri ve uygulayabileceğimiz arama modelleri aşağıdaki gibidir;

- Web sayfaları,
- Resimler(Image),
- Videolar(Video),
- Dil çevirileri,
- Lokasyonlar,
- Uygulamanız ile alakalı reklamlar. Güncel SDK dökümanına göre sadece US sınırlarında geçerli.(Ads),
- MSN Encarta Online Encyclopedia' den anlık cevaplar. örneğin What is 100*37? sonucunun bulunması(Instant Answer),
- XHTML veya WML formatında Mobile cihazlar için daha az yer harcayan sonuçlar(MobileWeb),
- Güncel arama ile ilişkili olan aramalar(Related Search),
- Haber içeriklerinin aranabilmesi(News),
- Hava durumu ile ilişkili aramalar(Weather)
- vb...

Güzel. Şimdi kafamızda bir şeyler şekillenmeye başladı. En azından arama modelini nasıl seçebileceğimizi anladık. Peki talepler nasıl iletilecekler? İstemciler taleplerini **Bing API**servisine 3 farklı formatta iletebilirler.

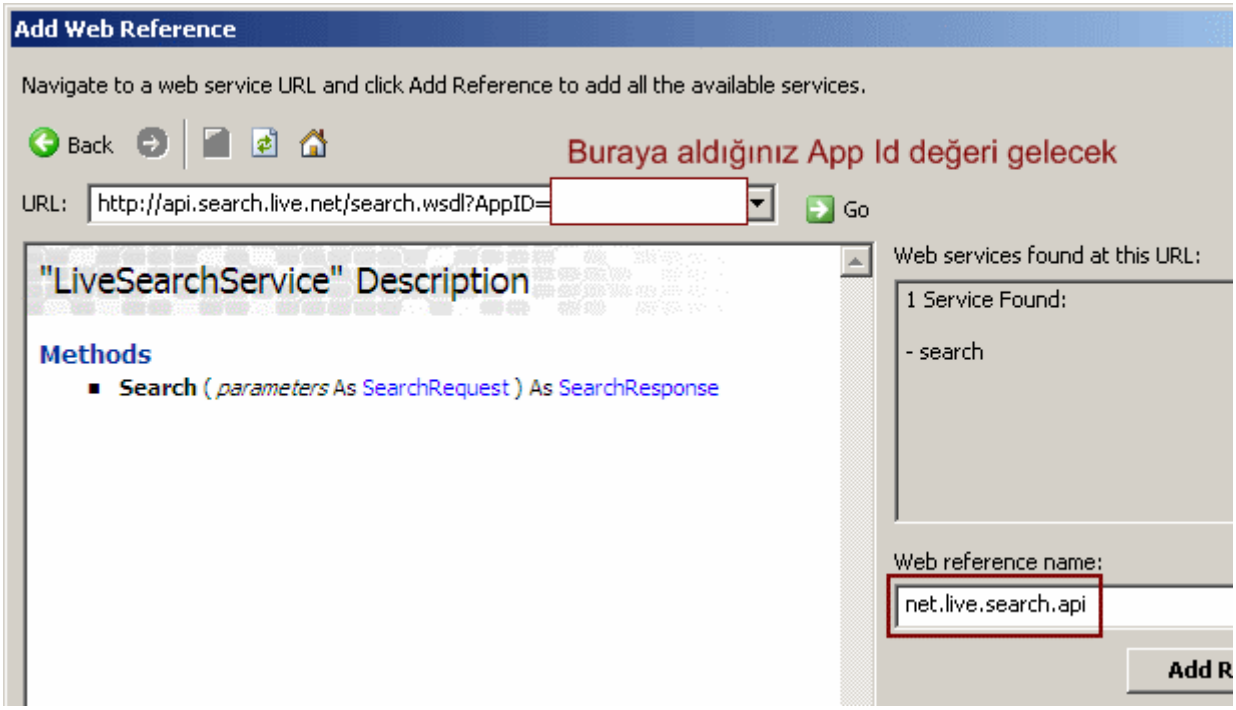
Format	**özet**
JavaScript Object Notation(JSON)	**Ajax** tabanlı uygulamalarda kullanılması tercih edilen bu tipe göre istemciye**Raw, Callback** ve **Function** formatlarında cevap döner.
eXtended Markup Language(XML)	**SOAP** formatını desteklemeyen veya **Siverlight** gibi uygulamalarda tercih edilir. İstemcinin talepleri **HTTP Get** metoduna göre gideceğinden URL sınırı en büyük handikapı olarak görülebilir.
Simple Object Access Protocol(SOAP)	**XML** modelindeki gibi URL sınır kısıtı yoktur. Ayrıca **karmaşık tiplerin(Complex Type)** ifade edilebilmesi, **request/response** nesne modelinin sağlanması gibi avantajları vardır. özellikle masaüstü uygulamalar(Desktop Applications) veya servis bazlı uygulamalar için idealdir. C# gibi yüksek seviyeli dillerle kullanımı son derece kolaydır.

Görüldüğü gibi, Bing API' si için değerlendirilecek istemci talepleri, **Json formatında, HTTP Get metodunda** gönderilebilmektedir. Ama burada altı çizilmesi gereken ve benimde en çok ilgimi çeken **SOAP** modelidir. öyleki, bu modelin uygulanması için istemci tarafının bir **XML Web Service** referansını kullanması yeterlidir. Bu, istemci tarafında managed bir kodun uygulanabilmesi anlamına gelmektedir. Asenkron çağrılar

gerçekleştirebilir, strong tipler kullanabilir, hatta sonuç kümeleri üzerinde LINQ sorguları dahi yapılabilir.

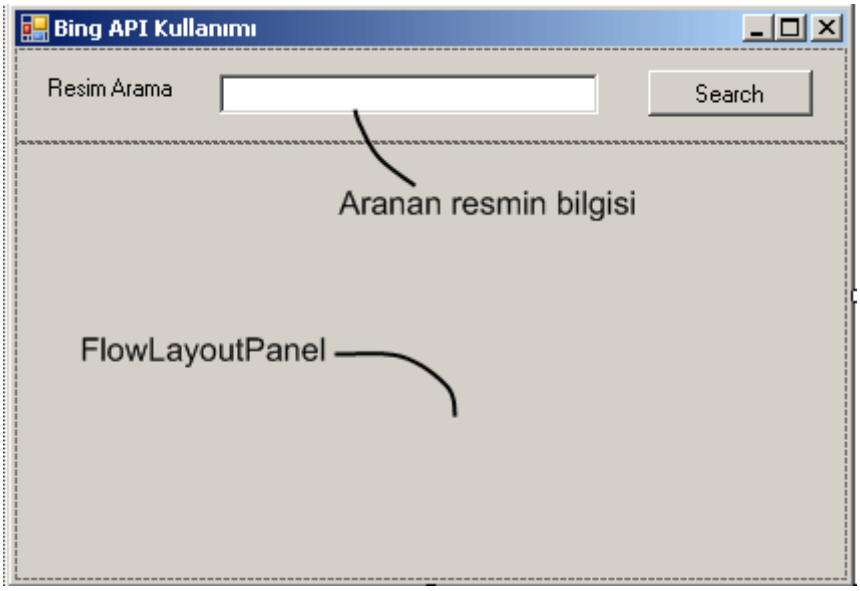
Kişisel Not: *Bir zamanlar .Net üzerine eğitmenlik yapardım. İlk yıllarımda .Net 1.0 vardı ve Xml Web Service konusunda gerçek hayat örnekleri bulmakta zorlanırdık. Genellikle kendi servislerimizi yazar, çağırır ve ele alırdık. Yada popüler hava durumu servisi örneği. Ama gerçek hayat senaryolarında, çok basit olan ve bizim tarafımızdan yazılmamış bir Web Servisi nasıl değer kazanabilir,artık pek çok örneği ile görebilmekteyiz. İşte küçük bir örnek, Bing API tarafından kullanılan Live servisi...*

öyleyse hiç vakit kaybetmeden **acele acele** bir örnek yapalım. Bu acele örneğimizde basit bir **Windows** uygulamasına, aradığımız kritere uyan **20** resmi çekmeye çalışacağız. Bir başka deyişle **SourceTypes.Image** tipinden bir arama gerçekleştireceğiz. Yapmamız gereken ilk şey, **Live Search** servisine ait **Xml Web Service** referansını uygulamamıza eklemek olmalıdır. Aşağıdaki görüntüde olduğu gibi. Dilerseniz benim yaptığım gibi **Web reference name** alanının değerini aynen bırakabilirsiniz.



Kişisel Not : *Referans eklemesinden sonra **Class Diagram** görüntüsüne bakmanızı öneririm*

Uygulamamızın Form tasarımını aşağıdaki gibi düzenleyebiliriz. Ben arama kutucuğunun sonucu olarak gelecek resim bilgilerini, alt tarafta yer alan **FlowLayoutPanel** bileşeni içerisinde**PictureBox** kontrolleri ile ifade etmeyi tercih ettim.



PictureBox kontrolümüzde, resmin arama sonuçlarından gelen tüm bilgilerinide saklamak istediğimden, aşağıdaki kod parçasında görülen **ThumbImage** isimli bir bileşen kullanmayı uygun gördüm.

```
using System.Windows.Forms;
using WinClient.net.live.search.api;

namespace WinClient
{
   class ThumbImage
      :PictureBox
   {
      public ImageResult Result { get; set; }
   }
}
```

Dikkat edileceği üzere **ImageResult** tipinden bir özellik yer almaktadır.
Bu **özellik(Property)**, arama sonucu servisden gelen sonuç kümesinde yer alan resim bilgilerini taşıyan tiptir. Kendi içerisinde, resmin **Thumbnail Url, Media Url, Title, Width, Height** vb... bilgilerini taşımaktadır. Biz bu bilgilerden faydalanıyor olacağız. Peki ama nasıl? İşte Form sınıfımızın tüm kod içeriği;

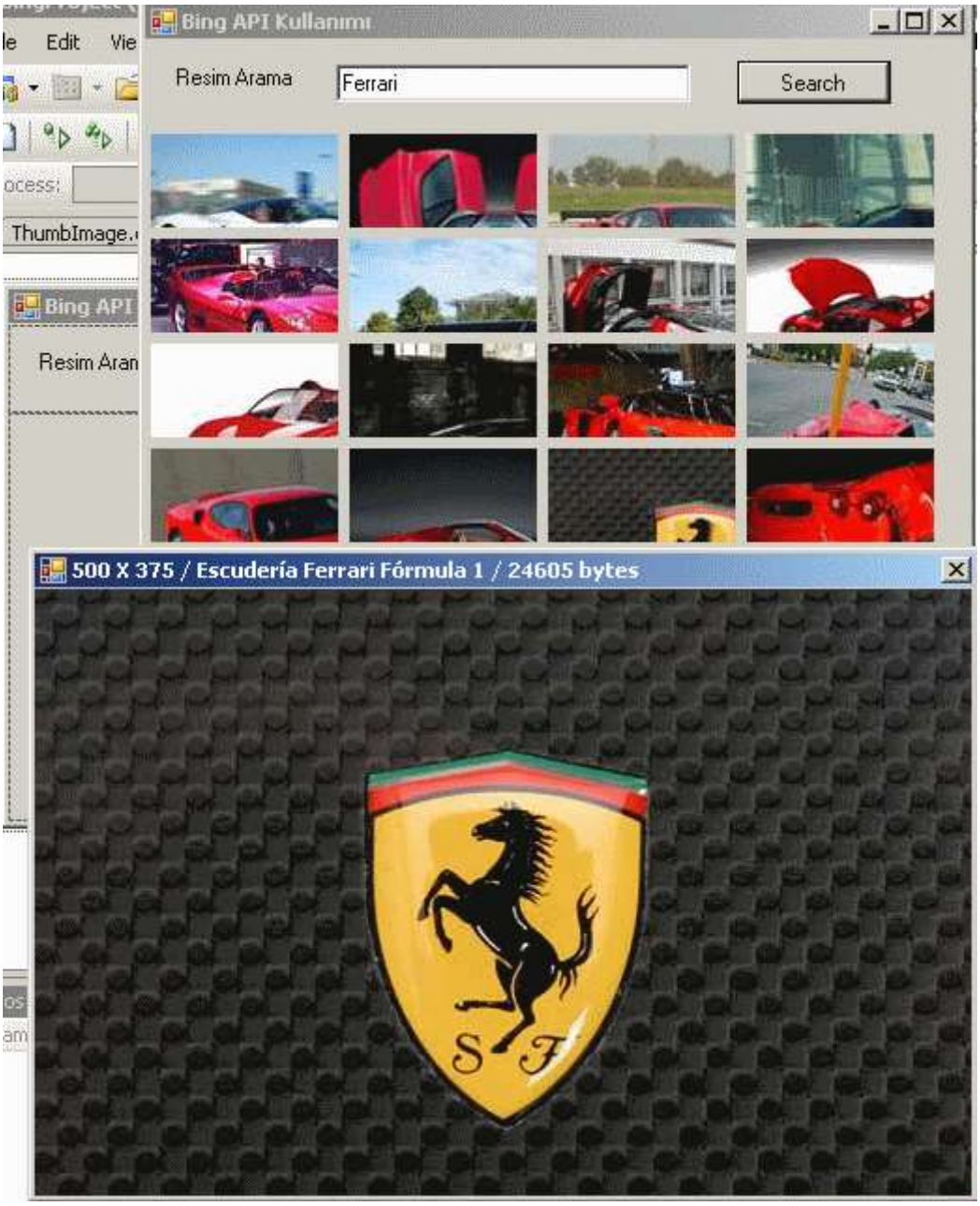
```
using System;
using System.Windows.Forms;
// Varsayılan olarak SOAP tabanlı Live servisinin eklenmesi ile gelen namespace
using WinClient.net.live.search.api;

namespace WinClient
{
   public partial class Form1 : Form
```

```
{
    public Form1()
    {
        InitializeComponent();
    }

    private void btnSearch_Click(object sender, EventArgs e)
    {
        pnlImages.Controls.Clear();

        if (!String.IsNullOrEmpty(txtSearch.Text))
        {
            using (LiveSearchService searchService = new LiveSearchService())
            {
                #region Arama talebi oluşturulur

                SearchRequest request = new SearchRequest
                {
                    AppId = "{Size verilen AppId değeri}",
                    Query = txtSearch.Text,
                    Sources = new SourceType[] { SourceType.Image },
                    Adult = AdultOption.Strict,
                    AdultSpecified = true,
                    Image = new ImageRequest { Count = 20, CountSpecified = true, Offset
= 0, OffsetSpecified = true }
                };

                #endregion

                #region Arama sonucunun değerlendirilmesi

                SearchResponse response = searchService.Search(request);

                if (response.Image!=null &&
                    response.Image.Results.Length > 0)
                {
                    foreach (ImageResult imgResult in response.Image.Results)
                    {
                        ThumbImage img = new ThumbImage
                        {
                            Result = imgResult,
                            ImageLocation = imgResult.Thumbnail.Url
                        };
```

```
                    img.Click += delegate(object obj, EventArgs args)
                    {
                       Form frm = new Form()
                       {
                          ControlBox=true
                          , MaximizeBox=false
                          ,MinimizeBox=false
                          , Text=String.Format("{0} X {1} / {2} / {3}
bytes",img.Result.Width,img.Result.Height,img.Result.Title,img.Result.FileSize)
                       };
                       PictureBox pb = new PictureBox {
                          ImageLocation =img.Result.MediaUrl
                          ,Dock= DockStyle.Fill
                       };

                       frm.Controls.Add(pb);
                       frm.Show();
                    };

                    pnlImages.Controls.Add(img);
                 }
              }
              else
              {
                 MessageBox.Show("Herhangibir sonuç bulunamadı", "Sonuç",
MessageBoxButtons.OK, MessageBoxIcon.Information);
              }
              #endregion
           }
        }
        else
        {
           MessageBox.Show("Lütfen aradığınız resim ile ilişkili bir bilgi giriniz","Sonuç",
MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
     }
  }
}
```

İlk olarak **LiveSearchService** nesnesi örneklenir. Bu örnek tahmin edileceği üzere **Search** operasyonunu yerine getirecek olan **proxy** tipimizdir. Diğer yandan arama işlemi için başlangıç kriterlerinin belirtilmesi gerekir. Bu amaçla **SearchRequest** tipinden bir nesne örneği oluşturulmaktadır. Dikkat edileceği üzere **Image** tipinden bir arama istendiği belirtilmiş ve buna göre **Image**özelliğine yeni bir **ImageRequest** nesnesi atanmıştır. **ImageRequest** nesnesinde **20** resimlik bir sonuç kümesinin talep edildiği

belirtilmektedir. **SearchRequest** sınıfı örneklenirken **App Id**değeri verilmektedir. Hatırlayınız, bu değeri siz formu doldurduktan sonra alıyorsunuz. önemli atamalardan biriside **Query** özelliği için yapılandır. Bu özelliğin değeri aranacak içeriği taşımaktadır. Bundan sonrası son derece kolaydır. **LiveSearchService** nesne örneğinin **Search** metoduna parametre olarak **SearchRequest** referansı atanır. Sonuçlar **SearchResponse** nesne örneğine gelir. Ardından **SearchResponse** nesne örneğinin **Image** özelliğinin **Results** koleksiyonundaki her bir **ImageResult** değerlendirilerek resim bilgilerinin alınması sağlanır. Elde edilen sonuçların her biri için bir **ThumbImage** bileşeni oluşturulur ve **FlowLayoutPanel** bileşeninin **Controls** koleksiyonuna eklenir. Uygulamanın çalışma zamanındaki örnek çıktısı aşağıda görüldüğü gibidir. Ben **Ferrari** kelimesi ile ilişkili resim dosyalarını arattım 😎.



Görüldüğü gibi minik resimlerden herhangirine tıklandığında orjinal halide yeni bir **Form** içerisinde gösterilebilmektedir. Buna ek olarak resim ile ilişkili bir kaç basit bilgide Form' un başlığında gösterilmektedir. Resmin boyutları, başlığı ve büyüklüğü. Ne kadar basit öğle değil mi? Bu arada **Bing API** ile ilişkili dökümanı indirdiğinizde içerisinde**JSON, XML ve SOAP** modellerinin her biri için ayrı ayrı yapılmış detaylı örnek anlatımları ve projeler olduğunu göreceksiniz. Bunları incelemenizi şiddetle tavsiye ederim. Peki bu acele örnekte yapmadıklarımız?

- **Exception kontrolü***(örneğin bağlantı problemelerinde yada resmin elde edilememesinde yaşanabilecek sıkıntıları handle etmek gerekir)*
- Asenkron arama metodu uygulanabilir ama bu durumda **Illegal Cross Thread Exception** hatasından kaçınmak gerekir.

Bunlarda size görev olsun. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WCF 4.0 Yenilikleri - Routing Service Geliştirmek - Giriş [Beta 1] (2009-08-24T18:15:00)

wcf,wcf 4.0,

Merhaba Arkadaşlar,

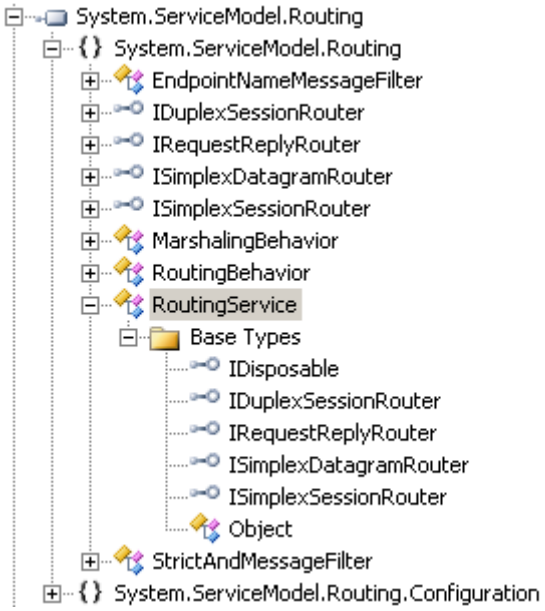
Servis Yönelimli Mimari(Service Oriented Architecture) çözümlerinde zaman zaman yönlendirme amaçlı servislerin yazılması gerekmektedir**(Router Service).** Bu servislerin genel kullanım amacı çoğunlukla, istemcilerden gelecek olan talepleri değerlendirip asıl işi yapacak olan servislere devretmek ile ilişkilidir. Ancak, gelen taleplere ait içeriğinin(Message Content, Header vb...) filtrelenerek ele alınması gibi ileri seviye teknikleride içerebilir. Yönlendirme işlemleri için kullanılan pek çok donanımsal cihaz ve hatta yazılım zaten mevcuttur. Bu nedenle öncelikli olarak yönlendirme servislerine neden ihtiyaç duyulabileceğini kavramakta yarar vardır.

WCF tarafında Routing Service geliştirilmesi hangi durumlarda tercih edilir?

- özel bir **Load Balancing** yapısı için*(genellikle donanımsal veya yazılımsal yük dengeleyici sistemlerin yetersiz kaldığı yada özelleştirilmek istendiğin durumlarda).*
- İstemciden gelen mesajın içeriğine göre servis yönlendirilmesi yapılmak istendiğinde**(Content Based).**
- önceliğe göre servis yönlendirmesi yapılmak istendiğinde**(Priority Based)**
- Versiyonlama senaryolarında.
- İstemciler ve yönlendirilen servisler arasında bir güvenlik geçidi kurulmak istendiğinde*(ki bu geçit genellikle **DMZ-demilitarized** zone arkasında asıl servislere olan akışı güvenlik kontrolüne alır).*

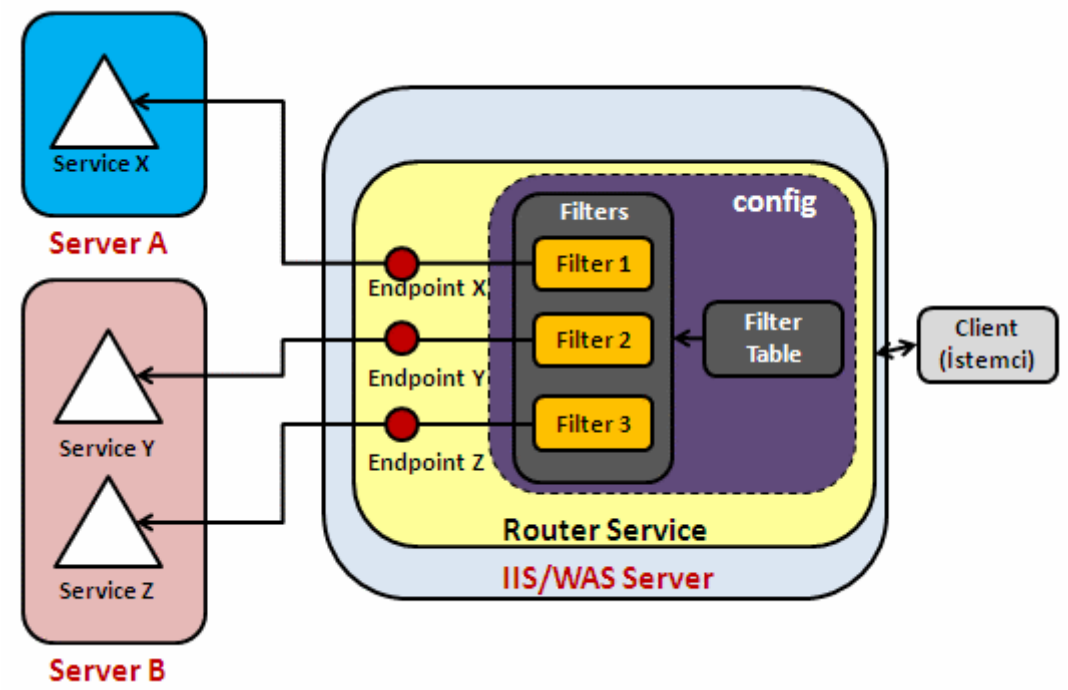
WCF 3.X tarafında yönlendirme servisi geliştirebilmek için belirli kodlama eforu sarfetmek gerekirken*(ki bu konuda daha önceden yayınladığım bir yazımı incleyebilirsiniz)*, **WCF 4.0** tarafında bu işlemler belirli tipler ve konfigurasyon özellikleri ile **one-way,two-way** ve **duplex** iletişim seviyesinde olduçka kolaylaştırılmıştır. **WCF 4.0** tarafında**System.ServiceModel.Routing assembly'** ı içerisinde yine aynı adlı isim alanında yer alan **RoutingService** isimli sınıf, söz konusu yönlendirme servisi için gerekli çalışma zamanı ortamının hazırlanmasını sağlamakta ve ayrıca istemci taleplerinin filtrelenerek uygun alt servislere aktarılmasında önemli bir rol oynamaktadır.

Not : *.Net Framework Beta 1 sürümünde **RoutingService** olarak geçen yönlendirme sınıfı çeşitli internet kaynaklarında(örneğin Michele Leroux Bustamante' nin bloğunda)**RouterService** olarak geçmektedir. Dolayısıyla final sürümde servisin adında farklılıklar olabilir. örneklerimizi **.Net Framework Beta 1** üzerinde geliştirdiğimizi hatırlatmak isterim.*



Object Browser yardımıyla elde edilen yukarıdaki görüntünden farkedeceğiniz gibi, **RoutingService** sınıfı 4 farklı **servis sözleşmesini(Service Contract)** uygulamaktadır. Bu

anlamda **IDuplexSessionRouter**, **IRequestReplyRouter**, **ISimplexDatagramRouter** ve **ISimplexSessionRouter** gibi önemli **arayüzleri(Interfaces)** implemente etmektedir. Dolayısıyla gerekli**MEP(Message Exchange Patterns)** modellerinin tümü desteklenmektedir. Buna göre servisin **one-way, two-way** veya **duplex** temelli isteklere göre çalışabilmesi sağlanmaktadır. **RoutingService**,**ServiceHost** nesne örneklemesi sırasında parametre olarak kullanıldığından, normal **WCF** host kurallarına tabidir. Yani **IIS** veya **Self** modellerde host edilebilir. Genellikle bir **Windows Service, IIS**yada duruma göre **WAS** üzerinden host edilmesi tercih edilmektedir. Genel olarak yönlendirme modelini aşağıdaki şekilde görüldüğü gibi özetleyebiliriz.



İstemciden(Client) gelen talepler **yönlendirme servisine(Router Service)** ulaştığında belirli filtrelerden geçmekte ve bu filtrelere göre belirlenmiş alt servis noktalarına aktarılmaktadırlar. Görüldüğü üzere önemli olan noktalardan biriside filtrelemedir. **Filtreleme tablosu(Filter Table)** ve içerdiği **filtreler(Filters) config** dosyası içerisinde depolanır. Bu filtrelerde;

- **XPath** gibi sorgular kullanılabilir. Sonuç itibariyle mesaj içeriğinin **XML** tabanlı olduğu düşünüldüğünde bu son derece doğaldır.
- Sadece mesajın **Header** veya **Soap Action** kısımlarına bakılabilir.
- Birden fazla filtrenin mantıksal ve(And) işlemine tabi tutulması sağlanabilir. Nitekim, daha önceden tanımlanmış iki farklı filtreye olan uygunluğun bir arada sağlanması istenebilir.
- İstenirse programlanmış bileşeneler ile özel filtrelemeler yapılabilir.

Filtrelemeler konfigurasyon içerikli olarak tutulduğundan, geliştiricilerin(Developers) söz konusu filtreleme davranışlarını koda bulaşmadan değiştirebilmesi, güncellemesi veya yenilerini eklemesi mümkündür.

Yönlendirme servisi çok doğal olarak istemciden gelen talepleri belirli kurallara göre işletmektedir. Yönlendirme hizmetinin istemcilere sunacağı bir **Endpoint** olması kaçınılmazdır. Tabi istenirse birden fazla endpoint sunabilir*(örneğin built-in gelen **ISimplexDatagramRouter**, **IRequestReplyRouter** gibi servis sözleşmelerinden yararlanarak...)* Diğer yandan istemcilerden gelen mesaj filtrelerde yer alan koşullardan birisine uyduğunda, ilişkili olan alt servise yönlendirilmelidir. Buna göre yönlendirme servisi, istemciden gelen ve filtreden geçen mesajları uygun alt servislere iletmesi gerektiğinden aynı zamanda bir istemci olarak düşünülmelidir. Dolayısıyla şekildende görüleceği gibi üzerinde her alt servis için en az bir **Endpoint** bulunmaktadır. Alt servisler istemcinin asıl işini yapmakla yükümlü olmakla birlikte, aynı sunucuda veya farklı sunucular üzerinde konuşlandırılmış olabilirler. Bu nedenle, yönlendirme servisi arkasında **Web Farm** gibi yapılara sıklıkla rastlandığını söyleyebiliriz.

Peki yönlendirme servisinin içerisinde yer aldığı basit bir sistemi nasıl tasarlayabiliriz? Burada belkide en kritik konu filtrelemelerdir. özellikle filtrelerde gelen mesaj içeriği üzerinde **XPath** ile sorgular atılması önemli olan ve zor noktalardandır. Bu gibi konuları bir sonraki yazımızda ele almaya çalışıyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WCF 4.0 Yenilikleri - Managed WS-Discovery [Beta 1] (2009-08-22T08:13:00)

wcf,wcf 4.0,

Merhaba Arkadaşlar,

Yandaki resimdeki gibi çok çok uzun bir yolun başında ve ulaşmanız gereken yere yüzlerce kilon
Sabırlı bir şekilde bu yolu gidebilmek için çok iyi bir disipline sahip olmanız gerekir. Yazılım ge
de bu tip yollar ile karşılaşmaz mıyız? Hemde sıklıkla karşılaşırız. Yılmadan yola devam edenler

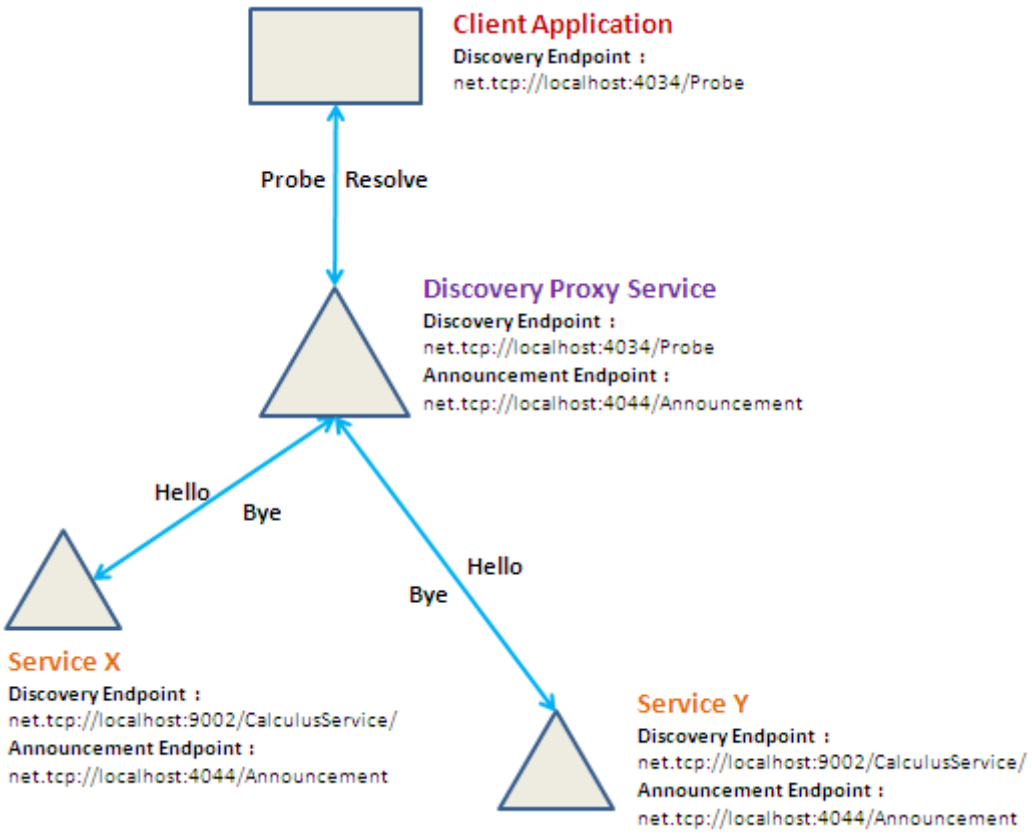
Ama belkide ulaşmazlar. Bu tamamen zamanın o andaki çevresel koşullarına bağlı olarak değişir. İşte bu yazımızda hakikaten sadece geliştirme aşaması dahi insanı çileden çıkarabilen zahmetli bir yola baş koyuyor olacağız. Hedefimiz, **WS-Discovery** tabanlı WCF sistemlerinde **Managed Discovery** modelini uygulayabilmek.

Konuyu **MSDN** ve diğer internet kaynaklarından araştırırken **Ad Hoc** modeli ile ilişkili tonlarca yazı olduğunu ama **Managed** tarafa pek kimsenin bulaşmak istemdiğini farkettim. Nedenini anlamam yaklaşık olarak 2,5 Litre kahve içmeme ve uykusuz bir Cumartesi gecesi geçirmeme neden oldu. Ama sonunda deydi. Aslında teorik olarak Managed modelin açıklaması son derece basit. İstemcilerin kullanmak isteyipte, farklı zamanlarda

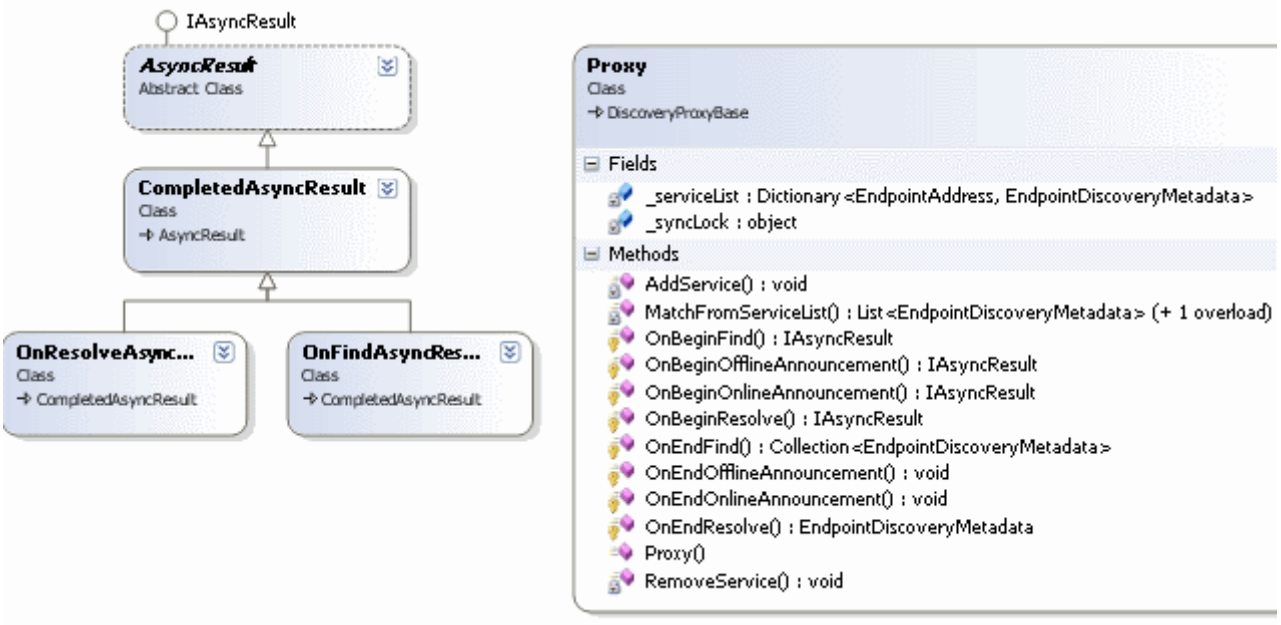
farklı lokasyonlardan ağa/ağlara dahil olan veya ayrılan servislerin keşfedilmesi görevi, istemci uygulamalardan alınıp istemci ile söz konusu servisler arasındaki başka bir Proxy servisine verilmektedir. Proxy servisi aslında hem **announcement** mesajları hemde istemcilerden gelecek olan **Probe** taleplerini dinlemektedir. **Announcement** mesajların dinlenmesi, online veya offline olan servislerin, Proxy servisi üzerinde bir saklama alanında tutulmasınıda gerektirir. Nitekim proxy servisi, ağa bağlı olan veya ayrılan tüm servislere ait ortak bir listeyi barındırıp istemci taleplerini bu listedeki durumlara göre karşılamalıdır. Diğer taraftan kendiside, istemciler tarafından keşfedilebilir olmalıdır. Bu nedenle tüm istemciler için ortak bir **Discovery Endpoint** noktasına sahip olmalıdır. Proxy servisini bu nedenlerden dolayı sürekli online halde kalan bir hizmet olarak düşünebiliriz. Online kalması önemlidir; çünkü online olduğu sürece, ağı dinleyerek katılan servisleri listesine alabilir ve istemcilerden gelen Probe veya Resolve gibi çağrılara cevap verebilir. Peki işi zorlayan nokta nedir?

Herşeyden önce Proxy servisinin, çalışma zamanındaki hareketliliği normal bir servis gibi değildir. Yani standart **ServiceHost** tipi tek başına yeterli değildir. Bu nedenle,**DiscoveryProxyBase** isimli **abstract** sınıftan bir türetme işlemi yapılarak üretilen bir servis tipi kullanılmalıdır. çok doğal olarak bu base içerisinden **override** edilmesi gereken bir takım üyelerde gelmektedir. Ayrıca, Proxy servisi tek bir örnek olarak**(Single Instance)** oluşturulmalı fakat eş zamanlı olarak gelecek istemci ve announcement taleplerine de cevap verebilmelidir. Bu noktada**IAsyncResult** arayüzünüde içeren asenkron modeli uygulayıp Thread yönetimini üstlenen yardımcı bir takım tipler kullanması gerekmektedir. Zaten işin zorlaştığı nokta burasıdır. Neyseki MSDN' de bu konu ile ilişkili olan örnekte, Asenkron desenin uygulanması için standart olarak sunulan sınıflar hazırdır. Dolayısıyla bu yapının aynısı kullanılarak gerekli geliştirmeler biraz daha kolayca yapılabilir. Biz örneğimizde sadece asenkron iletişimi ele alan tipleri alırken, DiscoveryProxyBase tabanlı türetmeyi kendimize göre düzenleyeceğiz.

öyleyse başlamaya ne dersiniz. Herkes sıcak kahvesini veya çayını yada yazın şu sıcak günlerinde gidecek serin bir içeceğini alsın ve benimle birlikte adım adım ilerlemeye gayret etsin. Başlamadan önce hedef modelimizin ne olduğunu kabaca aktarmak isterim. Aşağıdaki şekilde görülen senaryoyu ele almaya çalışacağız.



Şeklimizden anlaşılacağı üzere **Discovery Proxy** Servisimiz, **Service X** ve **Service Y'** nin **online/offline** olma durumlarını izlemektedir. Ayrıca istemci uygulama/uygulamalar aramak istediği servise ait talebi doğrudan **Discovery Proxy** servisine göndermektedir. İşte tam olarak gerçekleştirmek istediğimiz test senaryosu budur. İşe ilk olarak Discovery Proxy Servisinin yazımı ile başlayabiliriz. Dana öncedende belirttiğimiz gibi, bu servis içerisinde asenkron bir yapı kullanılması söz konusu olduğundan işimiz pek kolay değil. Ben sadece **Proxy** isimli **DiscoveryProxyBase**abstract sınıfından türeyen tipin uygulanışını burada göstermek istiyorum. örnek uygulama kodlarını indirdiğinizde **OnResolveAsyncResult** ve **OnFindAsyncResult** gibi tiplerin detaylarınıda bulabilirsiniz ki bunlarda standart olarak kullanılan tiplerdir ve MSDN tarafından yayınlanmıştır. Discovery Proxy servisinin içerisindeki sınıf modeli en basit haliyle aşağıdaki şekilde olduğu gibidir.



Burada bizi daha çok ilgilendiren kısım DiscoveryProxyBase türevli olan Proxy sınıfının kodlamasıdır. İşte kodlarımız;

```
using System;
using System.Collections.Generic;
using System.Collections.ObjectModel;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Discovery;

namespace DiscoveryProxyService
{
   // T anından sadece tek bir Proxy servis nesne örneğinin olabileceğini ve aynı andan birden fazla çağrıyı karşılayabilecek şekilde kullanılabileceğini belirtiyoruz
   [ServiceBehavior(InstanceContextMode = InstanceContextMode.Single, ConcurrencyMode = ConcurrencyMode.Multiple)]
   class Proxy
      : DiscoveryProxyBase
   {
      // Endpoint ve metadata bilgilerini tutacağımız bir koleksiyon tanımlanır
      Dictionary<EndpointAddress, EndpointDiscoveryMetadata> _serviceList = null;
      // _serviceList' in tutarlığını(Consistency) sağlamak için tanımlanan yardımcı değişkendir
      object _syncLock = null;

      public Proxy()
      {
         _serviceList = new Dictionary<EndpointAddress, EndpointDiscoveryMetadata>();
```

```
    _syncLock = new object();
}

#region Yardımcı Metodlar

// Online olan bir servis bilgisini listeye eklemek için kullanılır
// Proxy her bir announce mesajı aldığını çalıştırılacaktır.
void AddService(EndpointDiscoveryMetadata metadata)
{
    // eş zamanlı thread senkronizasyonunu sağlamak için lock kullanılmıştır
    lock (_syncLock)
    {
        // Address key değerine sahip bir value var ise güncelleme yoksa ekleme yapar.
        _serviceList[metadata.Address] = metadata;
        Console.WriteLine("{0} adresli endpoint eklendi",metadata.Address.ToString());
    }
}
// Offline olan bir servisi listeden çıkartmak için kullanılır
void RemoveService(EndpointDiscoveryMetadata metadata)
{
    // eş zamanlı thread senkronizasyonunu sağlamak için lock kullanılmıştır
    lock (_syncLock)
    {
        _serviceList.Remove(metadata.Address);
        Console.WriteLine("{0} adresli endpoint çıktı", metadata.Address.ToString());
    }
}

// Bu metod ve aşırı yüklenmiş(overload) versiyonu Resolve ve Probe mesajlarında
kullanılır
// FindCriteria tipinden olan parametre ile gelen kriterler uyan servisleri, listeden
çekmektedir
List<EndpointDiscoveryMetadata> MatchFromServiceList(FindCriteria
findCriteria)
{
    List<EndpointDiscoveryMetadata> result = null;

    lock (_syncLock)
    {
        result = (from epMetadata in _serviceList.Values
                  where findCriteria.IsMatch(epMetadata)
                  select epMetadata).ToList<EndpointDiscoveryMetadata>();
    }
```

```
        return result;
    }

    // ResolveCriteria tipinden gelen parametrenin Address bilgisine eş düşen servisi
listeden bulup Discovery Metadata bilgisini döndürür
    EndpointDiscoveryMetadata MatchFromServiceList(ResolveCriteria rCriteria)
    {
        EndpointDiscoveryMetadata result = null;

        lock (_syncLock)
        {
            result = (from epMetadata in _serviceList.Values
                      where epMetadata.Address == rCriteria.Address
                      select epMetadata).Single();
        }

        return result;
    }

    #endregion

    #region Override edilen metodlar

    // Online Announcement mesajı alındığından devreye giren metoddur
    protected override IAsyncResult
OnBeginOnlineAnnouncement(AnnouncementMessage announcementMessage,
AsyncCallback callback, object state)
    {
        AddService(announcementMessage.EndpointDiscoveryMetadata);
        return base.OnBeginOnlineAnnouncement(announcementMessage, callback, state);
    }

    // Online announcement mesajının işlenmesi bittiğinde devreye girer
    protected override void OnEndOnlineAnnouncement(IAsyncResult result)
    {
        base.OnEndOnlineAnnouncement(result);
    }

    // Offline announcement mesajı geldiğinde devreye giren metoddur
    protected override IAsyncResult
OnBeginOfflineAnnouncement(AnnouncementMessage announcementMessage,
AsyncCallback callback, object state)
    {
        RemoveService(announcementMessage.EndpointDiscoveryMetadata);
```

```
        return base.OnBeginOfflineAnnouncement(announcementMessage, callback, state);
    }

    // Offline announcement mesajının işlenmesi bittiğinde devreye giren metoddur
    protected override void OnEndOfflineAnnouncement(IAsyncResult result)
    {
        base.OnEndOfflineAnnouncement(result);
    }

    // Bir Find talebi geldiğinde devreye giren metoddur
    protected override IAsyncResult OnBeginFind(FindRequest findRequest,
AsyncCallback callback, object state)
    {
        return new OnFindAsyncResult(
            MatchFromServiceList(findRequest.Criteria)
            , callback
            , state);
    }

    // Find talebinin işlenmesi sona erdiğinde devreye giren metoddur
    protected override Collection<EndpointDiscoveryMetadata>
OnEndFind(IAsyncResult result)
    {
        return new
Collection<EndpointDiscoveryMetadata>(OnFindAsyncResult.End(result));
    }

    // Resolve mesajı geldiğinde devreye giren metoddur
    protected override IAsyncResult OnBeginResolve(ResolveRequest
resolveRequest, AsyncCallback callback, object state)
    {
        return new
OnResolveAsyncResult(MatchFromServiceList(resolveRequest.Criteria)
            , callback
            , state);
    }

    // Resolve mesajının işlenmesi bittiğinde devreye giren metoddur
    protected override EndpointDiscoveryMetadata OnEndResolve(IAsyncResult
result)
    {
        return OnResolveAsyncResult.End(result);
    }
```

```
        #endregion
    }
}
```

Sizi bu kod parçası ile bir süre yanlız bırakmak isterim Aslında sınıfımızın görevi basittir. çevre ağlar üzerinde **announcement** mesajı yayınlayarak **online** veya **offline**olduğunu bildiren servisleri tutmakta ve buna ek olarak, istemciden gelen arama kriterlerine uygun olanlarını yine istemci tarafına yönlendirmektedir. Sınıfımız, yardımcı metodların yanı sıra **DiscoveryProxyBase** tipinden gelen bazı **sanal metodlarıda(Virtual Method)** ezmektedir. özellikle eş zamanlı isteklerde oluşabilecek senkronizasyon sorunlarını aşmak için basit **lock**tekniğinden yararlanılmaktadır. Proxy servisini geliştirmek tek başına yeterli değildir. Bu servisin bir uygulama tarafından host edilmesi gerekmektedir. Bu anlamda basit bir Console uygulaması aşağıdaki kodlar ile tasarlanabilir.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Discovery;

namespace DiscoveryProxyService
{
    class Program
    {
        static void Main(string[] args)
        {
            // ServiceHost nesnesi DiscoveryProxyBase türevli Proxy tipi ile oluşturulur.
            ServiceHost host = new ServiceHost(new Proxy());

            // İstemcilerin Probe mesajları için bir DiscoveryEndpoint noktası tanımlanır
            DiscoveryEndpoint discoEndpoint = new DiscoveryEndpoint(
                new NetTcpBinding()
                , new EndpointAddress("net.tcp://localhost:4034/Probe"));
            discoEndpoint.IsSystemEndpoint = false;
            // DiscoveryEndpoint host' a eklenir
            host.AddServiceEndpoint(discoEndpoint);

            // Online veya Offline olan servislerin kendilerini Proxy servisine bildirebilmeleri amacıyla bir AnnouncementEndpoint noktası oluşturulur ve servise ilave edilir
            AnnouncementEndpoint announceEndpoint = new AnnouncementEndpoint(
                new NetTcpBinding()
                , new EndpointAddress("net.tcp://localhost:4044/Announcement"));
            host.AddServiceEndpoint(announceEndpoint);
```

```
            host.Open();
            Console.WriteLine("Managed Discovery Servis Durumu : ");
            Console.ReadLine();
            host.Close();
        }
    }
}
```

Proxy servisini bu şekilde host ettikten sonra, kendisine bildirimde bulunabilecek bir servisin nasıl tasarlanabileceğine de bakmamız yerinde olacaktır. Bu anlamda örneğimizde **ServiceX** ve **ServiceY** isimli iki farklı servis uygulaması bulunmaktadır. Bu servislerin en önemli görevlerinden biriside, ağa dahil olmaları veya ayrılmaları halinde bu durumlarını **Proxy** servisine bildirmeleridir. Her iki servis arasındaki fark ise tabiki sundukları hizmettir.

ServiceX içeriği;

```
using System;
using System.ServiceModel;
using System.ServiceModel.Description;
using System.ServiceModel.Discovery;

namespace ServiceX
{
    [ServiceContract]
    interface ICalculus
    {
        [OperationContract]
        double Sum(double x, double y);
    }

    class CalculusService
        : ICalculus
    {
        public double Sum(double x, double y)
        {
            return x + y;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ServiceHost host = new ServiceHost(
```

```
            typeof(CalculusService)
            , new Uri("net.tcp://localhost:9002/CalculusService/" +
Guid.NewGuid().ToString());
         host.AddServiceEndpoint(
            typeof(ICalculus), new NetTcpBinding(), string.Empty);
         // Bir announcement endpoint noktası oluşturulur ve proxy servisine bu sayede
bildirim yapılması sağlanır
         AnnouncementEndpoint announcementEndpoint = new
AnnouncementEndpoint(
            new NetTcpBinding()
            , new EndpointAddress("net.tcp://localhost:4044/Announcement"));
         // Servisin keşfedilebilir olması sağlanır
         ServiceDiscoveryBehavior serviceDiscoveryBehavior = new
ServiceDiscoveryBehavior();
         serviceDiscoveryBehavior.AnnouncementEndpoints.Add(announcementEndpo
int);
         host.Description.Behaviors.Add(serviceDiscoveryBehavior);

         host.Open();
         Console.WriteLine("Service X açıldı");
         Console.ReadLine();
         host.Close();
      }
   }
}
```

ServiceY içeriği;

```
using System;
using System.ServiceModel;
using System.ServiceModel.Description;
using System.ServiceModel.Discovery;

namespace ServiceY
{
   [ServiceContract]
   interface IAdventure
   {
      [OperationContract]
      double FindExpensiveProduct(int categoryId);
   }

   class AdventureService
      : IAdventure
   {
```

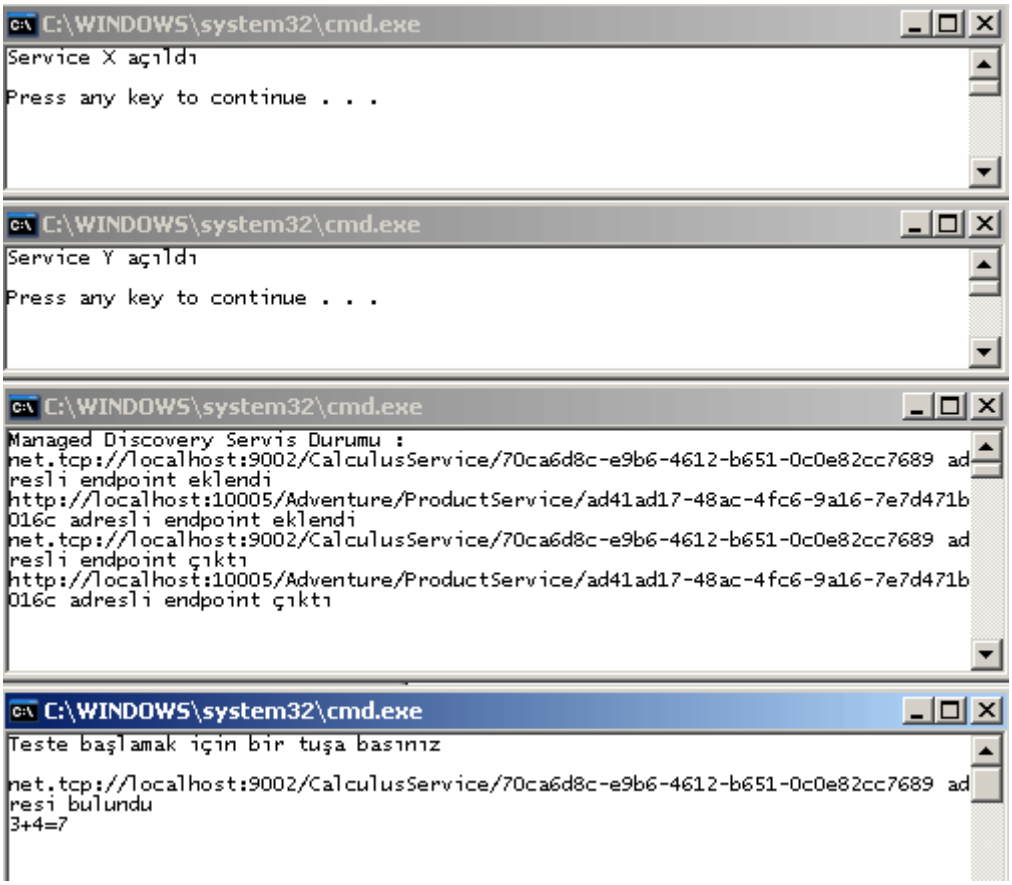
```
    public double FindExpensiveProduct(int categoryId)
    {
      return 1000;
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      ServiceHost host = new ServiceHost(
        typeof(AdventureService)
        , new Uri("http://localhost:10005/Adventure/ProductService/" +
Guid.NewGuid().ToString()));
      host.AddServiceEndpoint(
        typeof(IAdventure), new WSHttpBinding(), string.Empty);
      // Bir announcement endpoint noktası oluşturulur ve proxy servisine bu sayede
bildirim yapılması sağlanır
      AnnouncementEndpoint announcementEndpoint = new
AnnouncementEndpoint(
        new NetTcpBinding()
        , new EndpointAddress("net.tcp://localhost:4044/Announcement"));
      ServiceDiscoveryBehavior serviceDiscoveryBehavior = new
ServiceDiscoveryBehavior();
      // Servisin keşfedilebilir olması sağlanır
      serviceDiscoveryBehavior.AnnouncementEndpoints.Add(announcementEndpo
int);
      host.Description.Behaviors.Add(serviceDiscoveryBehavior);
      host.Open();
      Console.WriteLine("Service Y açıldı");
      Console.ReadLine();
      host.Close();
    }
  }
}
```

Piuuuuuvvvvv!!! 😃 İşimiz bitti diye düşünebilirsiniz. Ama hayır... Birde istemcilerin nasıl yazılabileceğine bakmamız gerekiyor. İstemci tarafında tabiki olmassa olmazlardan biriside, kullanmak istediği servislere ait proxy referanslarına sahip olmaları gerekliliğidir. Bunu göz önüne alarak ilerlediğimizi düşünürsek istemci tarafında da aşağıdaki gibi bir kodlama yapmamız yeterlidir.

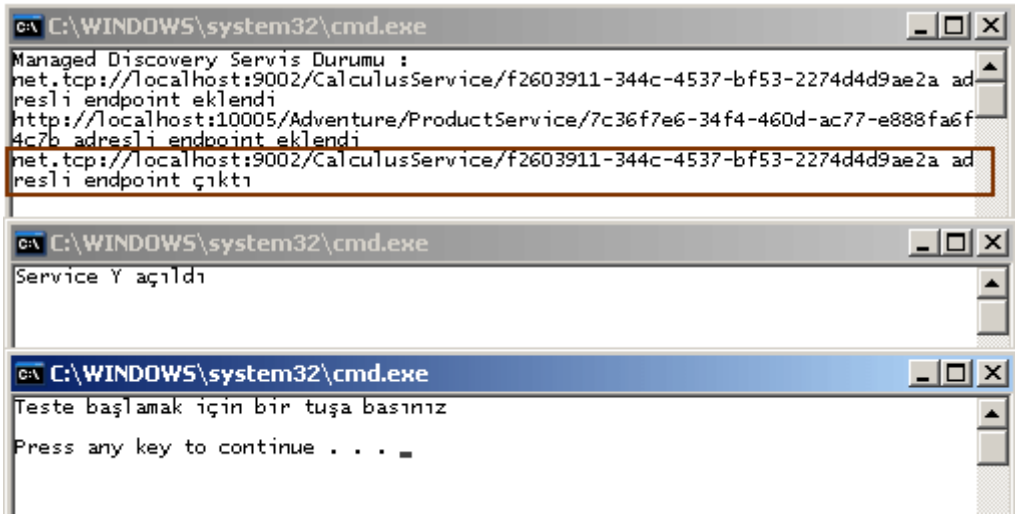
```
using System;
using System.ServiceModel;
using System.ServiceModel.Discovery;
```

```
namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Teste başlamak için bir tuşa basınız");
         Console.ReadLine();

         // Proxy servisini keşfedebilmek için bir DiscoveryEndpoint oluşturulur
         DiscoveryEndpoint disEndpoint = new DiscoveryEndpoint(
            new NetTcpBinding()
            , new EndpointAddress("net.tcp://localhost:4034/Probe")
            );
         // İstemci tarafının kullanılabilir servisleri keşfetmesini kolaylaştıran
DiscovertClient tipine ait nesne örneği DiscoveryEndpoint ile oluşturulur
         DiscoveryClient disClient = new DiscoveryClient(disEndpoint);
         disClient.Open();
         // Bir arama kriteri uygulanır ve dönen cevaptan kullanılabilir servis adresi tedarik
edilir
         FindResponse response = disClient.Find(new FindCriteria(typeof(ICalculus)));
         EndpointAddress epAddress=response.Endpoints[0].Address;

         // Eğer arama kriterine uygun servisler bulunmuşsa
         if (response.Endpoints.Count > 0)
         {
            // İlkinin adres bilgisini al
            EndpointAddress epAddress = response.Endpoints[0].Address;

            Console.WriteLine("{0} adresi bulundu", epAddress.ToString());

            // İstemci için gerekli proxy referansı örneklenir ve Probe mesajı ile bulunan
Endpoint adresi kullanılır.
            CalculusClient client = new CalculusClient(new NetTcpBinding(),
epAddress);

            // Servis operasyonu çağrılır
            Console.WriteLine("{0}+{1}={2}", 3, 4, client.Sum(3, 4).ToString());
            Console.ReadLine();
         }

         disClient.Close();
      }
   }
}
```

Nihayet test yapabilmek için gerekli ortamı hazırladığımızı ifade edebilirim. İlk olarak **Discovery Proxy** servisinin, sonrasında istemcinin kullanmak istediği servislerin ayağa kaldırılması gerekir. Son olarak istemci uygulamanın çalıştırılması ve test edilmesi yeterlidir. Yapılan ilk testler sonucunda aşağıdaki sonuçlar elde edilmiştir.



Görüldüğü gibi, **ServiceX** ve **ServiceY** isimli servislerin açılmaları ve kapatılmaları, **Managed Discovery Proxy** servisi tarafından tespit edilebilmiştir. **ServiceX'** in online olduğu zaman dilimi içerisinde, istemciden gelen talep başarılı bir şekilde karşılanabilmiştir. Farklı bir testide şu şekilde yapmak gerekir. İstemci uygulama, **ServiceX** için talepte bulunmadan önce, ServiceX kapatılır. Bu durumda istemcinin aradığı kritere uyan bir servis ayakta olmadığı için, istemcinin bir işlem yapamıyor olması gerekir. Olayı istisna ile sonlandırmayı engellemenin yolu ise **if** ile yapılan **Count** kontrolüdür. Bu tip bir testin sonucunda çalışma zamanı görüntüsü aşağıdaki gibi olacaktır.



Her ne kadar sadece iki çalışma zamanı testi yapılmış olsada, örneğin iyi bir şekilde değerlendirilmesi ve olası tüm hataların önüne geçilmesi gerekmektedir. Söz gelimi, **Proxy**servisinin kapatılmasından sonra, kendisine bağlı olan başka servislerin kapatılmaya çalışılması esnasında, söz konusu servislere ait ortamlarda **çalışma zamanı istisnaları(Runtime Exception)** oluşması kaçınılmazdır. Bu gibi noktaları dikkat almanızı ve geliştirmenizi buna göre yapmanızı öneririm.

Nihayet, uzun saatlerin, gidilen kilometrelerce yolun so alanında bir konuyu öğrenirken bu tip zorlu yollardan g çekinmeyecek ve ödül olarak şehrin parlak ışıkları ile k

ManagedWSDiscovery.rar (96,14 kb)

WCF 4.0 Yenilikleri - Announcement Kullanımı [Beta 1] (2009-08-21T13:00:00)

wcf,wcf 4.0,

Merhaba Arkadaşlar,

WCF 4.0 tarafında **WS-Discovery** tabanlı olarak gerçekleştirilen uygulamalarda önem arz eden noktalardan biriside, servislerin **online** veya **offline** olma durumlarını, bulundukları ağ üzerindeki**dinleyicilere(Listeners) bildirmeleridir(Announce)**. Bildiri şeklinde yapılan yayınlamalar aslında istemcinin ağ üzerine yaydığı **multicast** mesajların yoğunluğunu azaltmak gibi olumlu bir etkiye de sahiptir. Şimdi bu bildirim işlemlerinin nasıl yapılacağını incelemeye çalışalım. **Ad Hoc** modelinin uygulanması ile ileişkili

yazımızdaki örneğimizi bu amaçla devam ettirebiliriz. Servis tarafında konfigurasyon dosyasında sadece aşağıdaki eklemeleri yapmamız yeterli olacaktır.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceDiscovery>
            <announcementEndpoints>
              <endpoint kind="udpAnnouncementEndpoint"/>
            </announcementEndpoints>
          </serviceDiscovery>
          <serviceMetadata/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="ServerApp.CalculusService">
        <endpoint address="" binding="basicHttpBinding" contract="ServerApp.ICalculus"/>
        <endpoint address="Mex" kind="mexEndpoint" />
        <endpoint name="udpDiscovery" kind="udpDiscoveryEndpoint" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

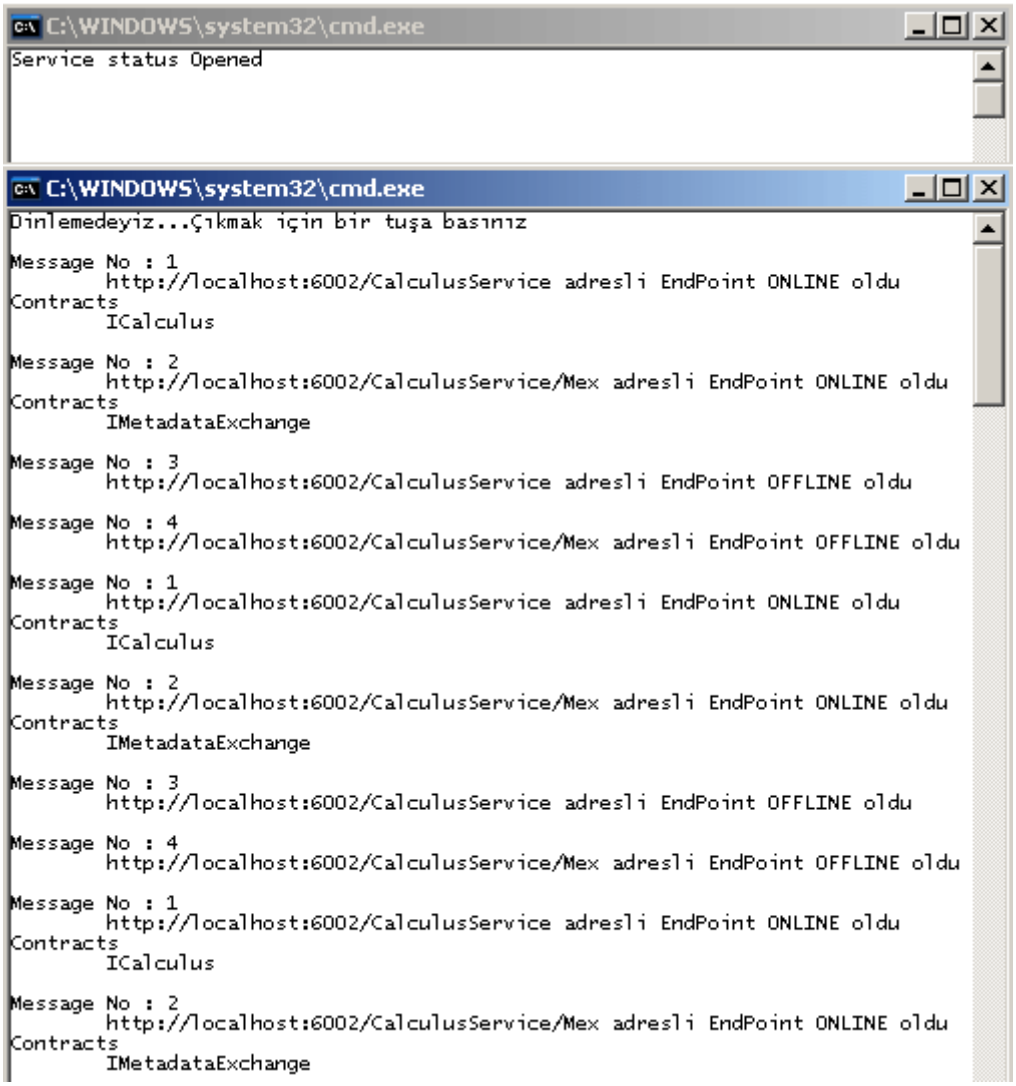
Konfigurasyon dosyasında, Servis tarafına yeni bir **davranış(Service Behavior)** eklenmiş ve bu davranış için **udpAnnouncementEndpoint** tipinden bir **Endpoint** kullanılacağı belirtmiştir. Bu endpoint tipi çalışma zamanında, servisin ağ üzerindeki dinleyicilere mesaj gönderebilmesi için gerekli alt yapının oluşturulmasını sağlamaktadır. Bir bakşa deyişle işimizi oldukça kolaylaştırmaktadır 🙂 Ancak istemci tarafında biraz kod eforu sarfedilmelidir. İstemci tarafı bir dinleyici olarak, servisin ortama gönderdiği "ben geldim" veya "ben gittim" tadındaki mesajları yakaladığında devreye girecek olan iki olay metodunu ele alabilmelidir. Tabi bunlardan daha önemlisi çalışma zamanı için gerekli alt yapı hazırlıklarınıda gerçekleştirmelidir. Şimdi istemci tarafındaki kodlarımızı aşağıdaki gibi geliştirdiğimizi düşünelim.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Discovery;

namespace ClientV2
{
```

```
class Program
{
   static void Main(string[] args)
   {
      // Servisin online/offline olma durumlarında yaptığı bildirimleri yakalayan nesne
örneği
      AnnouncementService aService = new AnnouncementService();

      // Servis online olduğunda devreye giren olay metodu
      aService.OnlineAnnouncementReceived += delegate(object sender,
AnnouncementEventArgs e)
      {
         // Online olan servisin etkinleştirilmiş Endpoint noktalarının Address bilgileri ve
o anki mesajın numarası yazdırılır.
         Console.WriteLine("\nMessage No : {0}\n\t{1} adresli EndPoint ONLINE oldu",
            e.AnnouncementMessage.MessageSequence.MessageNumber,
            e.AnnouncementMessage.EndpointDiscoveryMetadata.Address.ToString()
);

         // Etkinleşen Endpoint' ler üzerinden sunulan servis sözleşmeleri yazdırılır
         Console.WriteLine("Contracts ");
         foreach (var contractType
in e.AnnouncementMessage.EndpointDiscoveryMetadata.ContractTypeNames)
         {
            Console.WriteLine("\t{0}",contractType.Name);
         }
      };

      // Servis offline olduğunda devreye giren olay metodu
      aService.OfflineAnnouncementReceived += delegate(object sender,
AnnouncementEventArgs e)
      {
         // Kapatılan servis üzerindeki Endpoint bilgileri yazdırılır.
         Console.WriteLine("\nMessage No : {0}\n\t{1} adresli EndPoint OFFLINE
oldu",
                  e.AnnouncementMessage.MessageSequence.MessageNumber,
                  e.AnnouncementMessage.EndpointDiscoveryMetadata.Address.
ToString());
      };

      // AnnouncementService örneği kullanılarak bir ServiceHost nesnesi örneklenir
      ServiceHost host = new ServiceHost(aService);
      // Service yeni bir UdpAnnouncementEndpoint eklenir
      host.AddServiceEndpoint(new UdpAnnouncementEndpoint());
```

```
            // Dinleme işlemleri için servis açılır
            host.Open();

            Console.WriteLine("Dinlemedeyiz...çıkmak için bir tuşa basınız");
            Console.ReadLine();

            // Servis kapatılır
            host.Close();
        }
    }
}
```

Her ne kadar istemci tarafını geliştiriyor olsakta pek istemci tarzında olmadığını eminimki farketmişsinizdir. Nitekim istemci tarafında **ServiceHost** nesnesi örneklenmekte ve kullanılmaktadır. Aslında bu son derece doğaldır. Nitekim online veya offline olan servislerin, istemciler üzerinde tetikleyebildiği iki olay söz konusudur. Buda istemcinin bir anlamda servis gibide davranış gösterebilmesini gerektirmektedir. *(Normal şartlar altında servisin, istemciler üzerinde olay tetikletmesi gerektiği durumlarda özellikle **.Net Remoting** gibi modellerde çok kafa karıştırıcı kodlamalar yapılması gerektiğini hatırlatmak isterim.)* **WCF 4.0** tarafında ise tek yapmamız gereken bu iş yükünü **AnnouncementService** tipine atmaktır. Dikkat edileceği üzere**ServiceHost** nesnesi örneklenirken parametre olarak **AnnouncementService** referansı verilmektedir. Sonrasında ise **ServiceHost** nesnesine, **UpdAnnouncementEndpoint** tipinden bir **Endpoint** ilave edilmiştir. örnekle ilişkili ilginç noktalardan biriside istemci tarafında **App.config** dosyasının bulunmayışıdır.*(örnekten bu dosyası bilinçli bir şekilde çıkarttığımı belirtmek isterim)* İstemci uygulama dinlemede kaldığı süre boyunca, online veya offline olan tüm endPoint noktalarına ait **announce** mesajlarını yakalayabilmektedir. Bunlara ek olarak, istemcinin belirli bir servise odaklanmadığı da görülmektedir. Yerel ağ üzerindeki herhangibir servisten gelen **announce** mesajlarını dinleyebilmektedir. Modeli test etmek için istemci uygulama açık iken, bir veya daha fazla servisin*(tabiki bunların **WS-Discovery** tabanlı olarak geliştirilmiş olma şartları vardır)* kapatılıp açılması yeterlidir. Ben test sırasında aşağıdaki ekran görüntüsünde yer alan sonuçları aldım.



Görüldüğü üzere servisin bir kaç kere açılması ve kapatılmasının ardından istemci tarafındaki **OnlineAnnouncementReceived** ve **OfflineAnnouncementReceived** olayları tetiklenmiş ve gerekli bildirimler yakalanmıştır. Artık bu noktadan sonra istemcinin sadece online olan **Endpoint** noktalarına göre proxy nesnelerini oluşturması ve kullanması yeterli olacaktır.

Bir sonraki yazımızda Ad Hoc modelini terkedip, **Managed Discovery** modelini incelemeye çalışacağız. Bildiğiniz üzere **Ad Hoc** modelde yerel/alt ağlar söz konusudur ve ağın ötesine geçilmesi halinde proxy tabanlı bir sistemin kullanılması gerekmektedir. Bakalım bizi ne gibi sürprizler bekliyor olacak... Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

AdHocDiscoveryForAnnouncement.rar (82,45 kb)

WCF 4.0 Yenilikleri - Discovery için Scope Kullanmak [Beta 1] (2009-08-21T01:57:00)

wcf,wcf 4.0,

Merhaba Arkadaşlar,

Bir önceki yazımızda **WCF 4.0** tabanlı servislerde **WS-Discovery** protokolünün, **Ad Hoc** modeline göre nasıl uygulanabileceğini görmüştük. **Ad Hoc** modelinde istemcinin, yerel ağ üzerine dahil olan bir servisi aramak için kullanabileceği kriterleri önceden belirlemesi ve bunları kullanması gerektiğinden bahsetmiştik. Bu amaçla kod tarafında **FindCriteria** tipinden yararlanılmaktadır. Bir önceki örneğimizde, arama kriterinde sadece **servis sözleşmesini(Service Contract)** kullanmıştık. Ancak, arama alanını biraz daha dar tutmak amacıyla **Scope** bildirimlerinde de bulunabiliriz. Bir başka deyişle, ağ üzerinde birden fazla servisin arandığı durumlarda kapsama alanımızı, ekleyeceğimiz **Scope** kriterlerine göre azaltma şansımız bulunmaktadır.
Bir şekilde istemcinin ilgi alanınıda daha kesin çizgilerle belirlemiş olmaktayız. Peki bunu nasıl uygulayabiliriz?

Konunun servis tarafında yine konfigurasyon seviyesinde değerlendirilmesi gerekmektedir. Bu amaçla **endpointDiscovery** isimli bir **endpoint** davranışının kullanılması ve içerisinde gerekli **scope**tanımlamasının yapılması gerekmektedir. Bu amaçla daha önceden geliştirmiş olduğumuz servis örneğinde yer alan **app.config** dosyasına aşağıdaki davranış eklemelerini yaptığımızı düşünelim.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceDiscovery />
          <serviceMetadata/>
        </behavior>
      </serviceBehaviors>
      <endpointBehaviors>
        <behavior name="epBehavior">
          <endpointDiscovery>
            <scopes>
              <add scope="http://www.adventure.com/Math/Calculus"/>
            </scopes>
          </endpointDiscovery>
        </behavior>
      </endpointBehaviors>
    </behaviors>
```

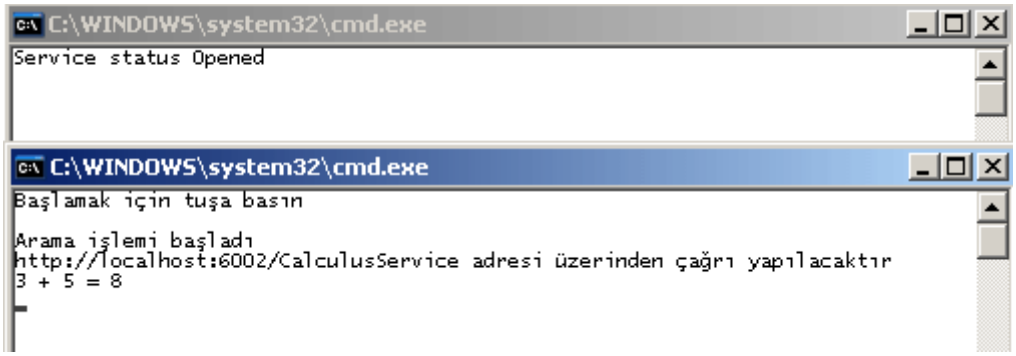
```
   <services>
     <service name="ServerApp.CalculusService">
       <endpoint address="" binding="basicHttpBinding"
contract="ServerApp.ICalculus" behaviorConfiguration="epBehavior" />
       <endpoint address="Mex" kind="mexEndpoint" />
       <endpoint name="udpDiscovery" kind="udpDiscoveryEndpoint" />
     </service>
   </services>
  </system.serviceModel>
</configuration>
```

Dikkat edileceği üzere **endpoint** davranışlarının tanımlandığı alanda, **endpointDiscovery** elementi içerisinde basit bir **scope** tanımlaması yapılmaktadır. **scopes** elementi içerisinde **n**sayıda **scope** tanımlaması olabilir. Tanımlamaların artması elbetteki kapsama alanının aranması için daha dar bir kriterin oluşturulmasına neden olacaktır. Bu daralma istemcinin arama operasyonu için aslında bir avantaj olarak düşünülebilir. Tanımlanan bu **discovery** davranışının hangi **endpoint** için ele alınacağı ise yine **endpoint** elementi içerisindeki**behaviorConfiguration niteliği(attribute)** yardımıyla sağlanmaktadır. Peki buna göre istemci tarafını nasıl kodlamalıyız? İşte istemci tarafındaki kod yapısının yeni hali...

```
using System;
using System.ServiceModel;
using System.ServiceModel.Discovery;
using ClientApp.CalculusSpace;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Başlamak için tuşa basın");
         Console.ReadLine();

         DiscoveryClient disClient = new DiscoveryClient("udpDiscovery");

         // Arama kriteri oluşturuluyor. Parametre olarak servis sözleşmesini içeren Interface
verilmekte
         FindCriteria findCriteria = new FindCriteria(typeof(ICalculus));
         findCriteria.Scopes.Add(new
Uri("http://www.adventure.com/Math/Calculus"));

         #region Asenkron erişim
```

```
            // Standart olay bazlı asenkron erişim tekniği kullanılır.

            disClient.FindCompleted += delegate(object sender, FindCompletedEventArgs e)
            {
               // Hata varsa bildir
               if (e.Error != null)
               {
                  Console.WriteLine(e.Error.Message);
               }
               else if (e.Cancelled == true) // İşlem iptal edilmişse bildir
               {
                  Console.WriteLine("İşlem iptali");
               }
               else // Aksi durumda işlemleri yürüt ve servis operasyonunu elde edilen adres
üzerinden çalıştır
               {
                  FindResponse findResponse = e.Result;
                  EndpointAddress epAddress = findResponse.Endpoints[0].Address;

                  // Bulunan endPoint adresi, proxy' nin üretilmesinde kullanılıyor
                  CalculusClient client = new CalculusClient("CalculusEndpoint", epAddress);

                  Console.WriteLine("{0} adresi üzerinden çağrı yapılacaktır",
epAddress.Uri.ToString());
                  double result = client.Sum(3, 5);
                  Console.WriteLine("{0} + {1} = {2}", 3, 5, result.ToString());
               }
            };

            disClient.FindAsync(findCriteria);
            Console.WriteLine("Arama işlemi başladı");
            Console.ReadLine();

            #endregion
         }
      }
   }
```

Görüldüğü gibi tek yaptığımız **Scopes** koleksiyonuna yeni bir **Uri** bilgisini, FindCriteria nesne örneği üzerinden eklemektir. Aslında buradaki metod parametresinin **Uri** tipinden olması, servis tarafındaki **scope** niteliğine neden bir **url** formatı yazdığımızı açıklamaktadır. Uygulamamızı bu haliyle çalıştırdığımızda yine bir önceki örnekte olduğu gibi, servisin keşfedilip bulunduğunu ve başarılı bir şekilde çalıştırıldığını görürüz.



özetle **Scope** eklentileri sayesinde istemcinin, servis keşfi yapması için gerekli ayarları tabir yerinde ise akord etmesi ve gerçekten ilgilendiği alanlara ait servisleri araması mümkün hale gelebilmektedir. Konu ile ilişkili olarak örneğin son halini link olarak vermiyorum. Lütfen burada yazılanları oraya taşıyıp denemekten üşenmeyiniz 😀 Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WCF 4.0 Yenilikleri - Ad Hoc WS-Discovery [Beta 1] (2009-08-18T18:23:00)

wcf,wcf 4.0,

Merhaba Arkadaşlar,

Uzun süredir macera dolu bir rüya görmüyordum. Geçtiğimiz gece askeri bir birlikte görev yapmaktaydım ve gerideki topçu birliklerime hedeflere ait koordinatları bildiriyordum. Tabi gerçek hayatta yedek subak eğitimimi Topçu sınıfında, Ateş Destek üzerine aldığım için keşif, arama ve bulma gibi konularda azda olsa bilgi sahibiydim. Rüyamda da bu bilgilerimi kullandığımı itiraf edebilirim. Peki konumuz nedir?

Bu yazımızdaki konumuzun keşif yapmak ile aslında çok yakın bir ilişkisi bulunmaktadır. WS-Discovery modeli için WCF 4.0 ile birlikte gelen kolaylıklar.

Ağ üzerinde bulunan servis noktalarının **çalışma zamanında keşfi(Runtime Discovery), Servis Yönelimli Mimarilerde(Service Oriented Architecture)** karşılaşılan en önemli ihtiyaçlardan birisidir. öyleki; bazı servislerin ağa dahil olması, ağdan ayrılması gibi zaman içerisinde yerlerinin sıklıkla değiştiği durumlarda, söz konusu servislerin istemciler tarafından dinamik olarak keşfedilmesi gerekebilir. **WS-Discovery** bu tip

durumlar için **OASIS** tarafından kabul görmüş bir mesajlaşma standardı sunmaktadır. Bu mesajlaşma standardına göre, servislerin keşfedilmesi(Discovery) için aslında temel olan dört temel operasyon söz konusudur.

- Hello
- Probe
- Resolve
- Bye

Servislerin ağa dahil olmaları sırasında **multicast** mesajlar yardımıyla "Merhaba, ben geldim,buradayım" demeleri bu operasyonlardan birisidir**(Hello)**. Diğer yandan çok doğal olarak ağa dahil olan bir servisin ağdan ayrılması halinde, "Ben gidiyorum" şeklinde bir multicast yayınlama yapması söz konusudur**(Bye)**. Bu da **WS-Discovery** içerisinde yer alan ve servisler tarafından gerçekleştirilen operasyonlardan bir diğeridir. İstemciler açısından olaya bakıldığında ise iki farklı operasyon söz konusudur. İstemciler, kullanmak istedikleri servislerin tipi veya kapsamlarına göre arama işlemlerini yine mutlicast mesajlaşma ile gerçekleştirebilirler**(Probe).** Birde istemcilerin servisi adları ile aramasıda mümkündür**(Resolve)**. Buda istemcilerin ele aldığı ikinci operasyon olarak düşünülebilir.

WS-Discovery ile ilişkili olarak yaptığım araştırmalarda pek çok kaynakta örnek olarak ağ üzerindeki bir yazıcı sürücüsünün(Printer Device) örnek olarak verildiğini gördüm. Sanıyorumki konunun anlaşılması üzerine verilebilecek en güzel örnek...Bu vakaya göre ağa katılan ve hatta bazı durumlarda ağdan ayrılan bir yazıcı sürücüsünün(Printer Device) servis olarak değerlendirildiği düşünülmektedir. Printer ağa katıldığında, ağ üzerindeki tüm boğumlara **tek yönlü bir merhaba mesajını(One-way Hello Message)** gönderir. Bu andan itibaren istemciler(Clients), ağ üzerine **Probe** mesajlarını göndererek var olan yazıcı listelerinden örneğin belirli bir alt ağda olanlara talepte bulunabilirler. Tabiki istemci isterse **Resolve** mesajı ile belirli bir yazıcı sürücüsüne talepte de bulunabilir. Elbetteki bu vakada eksik kalan kısım yazıcının ve dolayısıyla sürücü servisinin ağdan kopartılmasıdır(örneğin kapatılması). Bu durumda printer sürücü servisi, ağ üzerine bu kez **tek-yönlü güle güle(One-Way Bye)** mesajı göndererek artık etkin olmadığını bildirmektedir.

WCF 4.0 tarafında istemcilerin özellikle servisleri keşfetmeleri amacıyla değerlendirebilecekleri **WS-Discovery** protokolünün iki uygulama modeli bulunmaktadır. **Ad Hoc** ve **Managed**. Ad Hoc modeline göre istemci, önceden belirlenmiş kriterlerine göre servisi **Probe** mesajları ile keşfetmeye çalışmaktadır. Eğer **Probe** mesajına karşılık eşleşen bir servis bulunursa istemci tarafına bir cevap gönderilir. Bu modelde istemcinin **multicast** mesajlar ile sürekli bir kontrol içerisinde olması(Polling) ağ üzerindeki trafiği arttırıcı bir etken olarak görülebilir. İşte bu noktada servisin **announce** adı verilen **Hello** ve **Bye** mesajları ile kendisinin online olup olmadığına dair bildirimlerde bulunması söz konusu yükün hafifletilmesini sağlamaktadır. Bu modelin uygulanması son derece kolaydır. Diğer yandan model sadece yerel ağlar için kullanışlıdır. Ancak ağın ötesinde yer alan servislerin keşfedilmesi söz konusu olduğunda ise Managed modelin kullanılması gerekmektedir. **Managed** modelde, ağda görülen servislerin tamamı

için merkezileştirilmiş **depolama alanı(Repository)** ve bir **proxy** servisi söz konusudur. Böylece proxy servisi ağın ötesindeki canlı **Endpoint** listelerini tutarak, ağ içerisindeki diğer istemcilerin söz konusu servislerden yararlanabilmesini sağlamaktadır. Dolayısıyla istemciler doğrudan aradaki proxy servisi ile iletişim kurmaktadır. Managed modelin uygulanması biraz daha komplekstir. O nedenle bu ilk yazımızda kolay olan **Ad Hoc** modelini inceliyor olacağız.

özetle **WS-Discovery**, **OASIS** tarafından standart olarak görülmüş ve multicast mesajlaşmayı baz alan, servislerin keşfedilmesi için kullanılan bir protokol bütünü olarak tanımlanabilir. Konuyu daha net kavrayabilmek adına basit bir örnek ile ilerlememizde fayda olacağı kanısındayım. İlk olarak bir servis uygulaması geliştirecek ve bunu **WS-Discovery** destekli olacak şekilde kuracağız. Sonrasında ise bu servisi keşfetme yeteneğine sahip olan bir istemci uygulamayı geliştireceğiz. Her iki tarafında **.Net Framework 4.0 Beta 1** üzerinde ve **Visual Studio 2010 Beta 1**yardımıyla geliştirildiğini belirtelim. Servis tarafına ait kod içeriğimizi aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.ServiceModel;

namespace ServerApp
{
   [ServiceContract]
   interface ICalculus
   {
      [OperationContract]
      double Sum(double x, double y);
   }

   class CalculusService
      :ICalculus
   {
      public double Sum(double x, double y)
      {
         return x + y;
      }
   }

   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(CalculusService)
            ,new Uri("http://localhost:5002/CalculusService"));
```

```
            host.Open();
            Console.WriteLine("Service status {0}",host.State.ToString());
            Console.ReadLine();
            host.Close();
            Console.WriteLine("Service status {0}", host.State.ToString());
        }
    }
}
```

Servis tarafı için belkide en önemli olan kısım **App.config** dosyasının içeriğidir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <!-- Service Discovery davranışı etkinleştirilir-->
          <serviceDiscovery />
          <serviceMetadata/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="ServerApp.CalculusService">
        <endpoint address="" binding="basicHttpBinding" contract="ServerApp.ICalculus"
/>
        <endpoint address="Mex" kind="mexEndpoint" />
        <endpoint name="udpDiscovery" kind="udpDiscoveryEndpoint" /> <!-- UDP
tabanlı standart Endpoint tanımlaması yapılır-->
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

BasicHttpBinding tabanlı bir **Endpoint** tanımlamasının haricinde iki adet standart **endPoint** tanımlaması daha bulunmaktadır. **Mex** tabanlı olan ve **metadata publishing** açılımına izin veren bir yana, önemli olan **udpDiscoveryEndpoint** tipinden olanıdır. Bu tanımlamaya ek olarak **servis davranışlarında(serviceBehavior)** belirtilen **serviceDiscovery** bildirimi ile, servisin istemciler tarafından keşfedilebilmesi için **UDP** tabanlı protokol üzerinden destek verebileceği set edilmiş olmaktadır.*(Bu örnekte announce mesajlaşma kısmını değerlendirmedik. Bunu ilerleyen kısımlarda ele alacağız. Ancak udpAnnouncementEndpoint tipinin bu amaçla kullanıldığını ipucu olarak verebiliriz.)* Gelelim istemci tarafına...

İstemci tarafında önem arz eden konuların
başında **System.ServiceModel** haricinde **System.ServiceMode.Discovery assembly**' ınınd a projeye referans edilmesi gelmektedir. Nitekim bu assembly içerisinde **Discovery** sistemi için gerekli tip tanımlamaları yer almaktadır**(DiscoveryClient, FindCriteria** vb...) İstemci tarafında bir proxy tipi bulunmaktadır. Bunun üretimi için standart olarak **Add Service Reference** özelliğinden yararlanılabilir. Ancak bu sefer servis tarafının adresinin ne olduğu bilinmeden istemci tarafında geliştirilme yapılması hedeflenmektedir. Nitekim sistemimize göre istemci uygulama, ağ üzerinde belirli bir kritere uyan servisi arayacak, bulduğunda ise yayınlama yaptığı adresten yararlanarak proxy nesnesini ayağa kaldıracaktır.
Dolayısıya **config** dosyası içeriğini aşağıdaki gibi geliştirmemiz gerekmektedir.

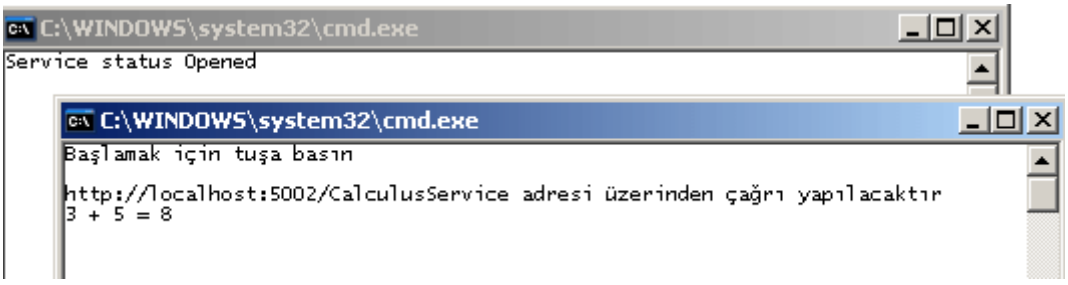
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <endpoint binding="basicHttpBinding" contract="CalculusSpace.ICalculus"
name="CalculusEndpoint" />
      <endpoint name="udpDiscovery" kind="udpDiscoveryEndpoint" />
    </client>
  </system.serviceModel>
</configuration>
```

Servis tarafındaki **app.config** dosyası içeriğine benzer olaraktan, istemci tarafında da standart **endPoint** tiplerinden olan **udpDiscoveryEndpoint** kullanımı söz konusudur. İstemci tarafının uygulama kodlarını ise aşağıdaki gibi geliştirmemiz yeterlidir.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Discovery;
using ClientApp.CalculusSpace;

namespace ClientApp
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Başlamak için tuşa basın");
            Console.ReadLine();

            DiscoveryClient disClient = new DiscoveryClient("udpDiscovery");

            // Arama kriteri oluşturuluyor. Parametre olarak servis sözleşmesini içeren Interface
verilmekte
            FindCriteria findCriteria = new FindCriteria(typeof(ICalculus));
```

```
        // FindCriteria' nun sonucunu taşıyacak olan FindResponse nesne örneği
oluşturuluyor
        FindResponse findResponse = disClient.Find(findCriteria);

        // Kritere göre bulunan ilk EndPoint adresi alınıyor
        EndpointAddress epAddress = findResponse.Endpoints[0].Address;

        // Bulunan endPoint adresi, proxy' nin üretilmesinde kullanılıyor
        CalculusClient client = new CalculusClient("CalculusEndpoint", epAddress);

        Console.WriteLine("{0} adresi üzerinden çağrı
yapılacaktır",epAddress.Uri.ToString());
        double result = client.Sum(3, 5);
        Console.WriteLine("{0} + {1} = {2}",3,5,result.ToString());

        Console.ReadLine();
      }
   }
}
```

DiscoveryClient tipine ait nesne örneğinin oluşturulması sırasında **UDP** tabanlı bir **Discovery** bağlantı noktasının kullanılacağı belirtilmektedir. Sonrasında ise bir arama kriteri oluşturulur. Bizim arama kriterimize göre servisin **arayüz sözleşmesi(Service Contract)** kullanılmaktadır. **DiscoveryClient** nesne örneğine ait olan **Find** metodu ile belirtilen kritere göre bir arama yapılmasına başlanır. Bir başka deyişle ağ üzerinde bir **multicast** mesaj yayını ile belirtilen kritere uygun servis uç noktası aranır. Servis noktası bulunduğunda ise **Find** metodundan geriye dönen **FindResponse** nesne örneğine ait **Endpoints** koleksiyonu değerlendirilir ve dizinin ilk elemanının **Address** özelliğinde servise ulaşılabilecek adres bilgisi elde edilir. Bundan sonraki kısım ise son derece tanıdıktır. Tek yapılması gereken **proxy** nesne örneğinin oluşturulması sırasında cevap olarak elde edilen **Endpoint** adres bilgisinin kullanılması gerekmektedir. Burada gözden kaçırmamamız gereken nokta, istemci tarafında herhangibir şekilde servise ait adres bilgisinin açık bir şekilde verilmemiş olmasıdır. İstemcinin elinde aslında ağ üzerindeki servisler içerisinde arama yapabileceği bir kriter bulunmaktadır**(ICalculus isimli servis sözleşmesi)**. Uygulamayı test ettiğimizde ilk etapta aşağıdaki çıktıyı elde ederiz.

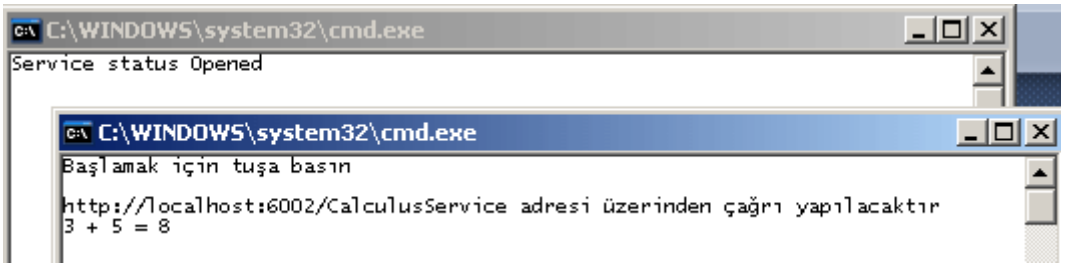


Nevarki istemci tarafında sonuç elde edilinceye kadar belirli bir süre beklendiği hemen fark edilebilir. Bu son derece doğaldır. Nitekim istemcinin kriterine uygun olan servisin arama

süresi söz konusudur. Ancak şunu belirtelim, istenirse söz konusu arama kısmı asenkron olarak değerlendirilebilir. Şu aşamada bizim için önemli olan nokta, istemcinin yayın yapılan servis adresini bulabilmiş olmasıdır. Hatta bu noktada testimize şu şekilde devam etmemizde yarar vardır. Tüm uygulamaları kapattıktan sonra servisin adresini aşağıdaki gibi değiştirdiğimizi düşünelim.

http://localhost:6002/CalculusService

Servis uygulamasını ve sonrasında istemci uygulamayı tekrardan çalıştırırsak bu kez aşağıdaki ekran görüntüsünü elde ederiz.



Mükemmel Nitekim servis tarafının adresini değiştirmiş olmamıza rağmen istemci uygulama çalışabilmektedir. Normal şartlarda **Discovery** gibi bir mekanizma kullanmadığımızda istemci uygulamalarda gerekli adres bilgilendirmelerinin değiştirilmesi gerektiğini unutmayalım.

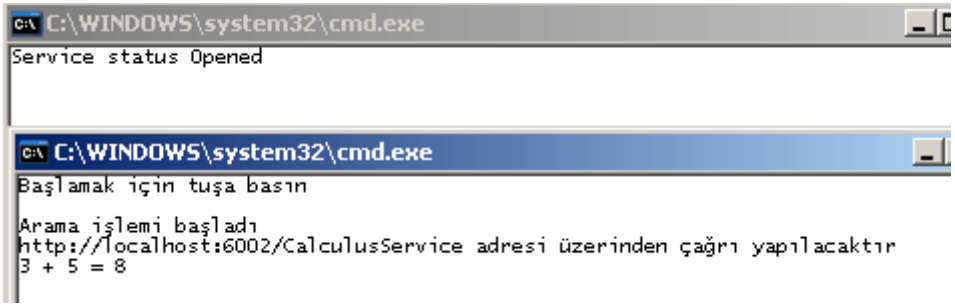
Şimdide bu işlemin asenkron olarak nasıl yapılabileceğini bakacağız. Web servislerine .Net 2.0 versiyonu ile birlikte getirilmiş olan **olay tabanlı asenkron erişim(Event-Based Asynchronous)**tekniği burada da kullanılmaktadır. **Find** metodunun asenkron olan **FindAsync** versiyonu kullanılarak, arama işleminin asenkron olarak gerçekleştirilmesi sağlanabilir. İşlemler başarılı bir şekilde tamamlandığı takdirde **FindCompleted** isimli olay metodu devreye girmektedir. İşte asenkron arama ile ilişkili örneğimizde yaptığımız değişiklikler.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Discovery;
using ClientApp.CalculusSpace;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Başlamak için tuşa basın");
         Console.ReadLine();
```

```
        DiscoveryClient disClient = new DiscoveryClient("udpDiscovery");

        // Arama kriteri oluşturuluyor. Parametre olarak servis sözleşmesini içeren Interface
verilmekte
        FindCriteria findCriteria = new FindCriteria(typeof(ICalculus));

        #region Asenkron erişim

        // Standart olay bazlı asenkron erişim tekniği kullanılır.

        disClient.FindCompleted += delegate(object sender, FindCompletedEventArgs
e)
        {
            // Hata varsa bildir
            if (e.Error != null)
            {
                Console.WriteLine(e.Error.Message);
            }
            else if (e.Cancelled == true) // İşlem iptal edilmişse bildir
            {
                Console.WriteLine("İşlem iptali");
            }
            else // Aksi durumda işlemleri yürüt ve servis operasyonunu elde edilen adres
üzerinden çalıştır
            {
                FindResponse findResponse = e.Result;
                EndpointAddress epAddress = findResponse.Endpoints[0].Address;

                // Bulunan endPoint adresi, proxy' nin üretilmesinde kullanılıyor
                CalculusClient client = new CalculusClient("CalculusEndpoint",
epAddress);

                Console.WriteLine("{0} adresi üzerinden çağrı yapılacaktır",
epAddress.Uri.ToString());
                double result = client.Sum(3, 5);
                Console.WriteLine("{0} + {1} = {2}", 3, 5, result.ToString());
            }
        };

        disClient.FindAsync(findCriteria);
        Console.WriteLine("Arama işlemi başladı");
        Console.ReadLine();

        #endregion
    }
```

```
    }
}
```

Bu kez örneğimizin çalışma zamanı çıktısı aşağıdaki gibi olacaktır.



Görüldüğü üzere **FindAsync** çağrısından sonra hemen alt satıra geçilerek kodun çalışması devam etmiştir. Servisten gerekli cevap alındıktan sonrada, Sum operasyonu başarılı bir şekilde icra edilmiştir. Discovery mekanizması ile ilişkili olaraktan WCF 4.0 tarafına gelen başka yeniliklerde söz konusudur. örneğin

- Scope kullanılarak keşif yapılması,
- announcement kullanılması,
- Managed servis keşif tekniği vb...

Bu konularıda ilerleyen yazılarımızda ele almaya çalışıyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

AdHocDiscovery.rar (51,72 kb)

WCF 4.0 Yenilikleri - Artık Svc Uzantısına Gerek Yok [Beta 1] (2009-08-18T12:45:00)

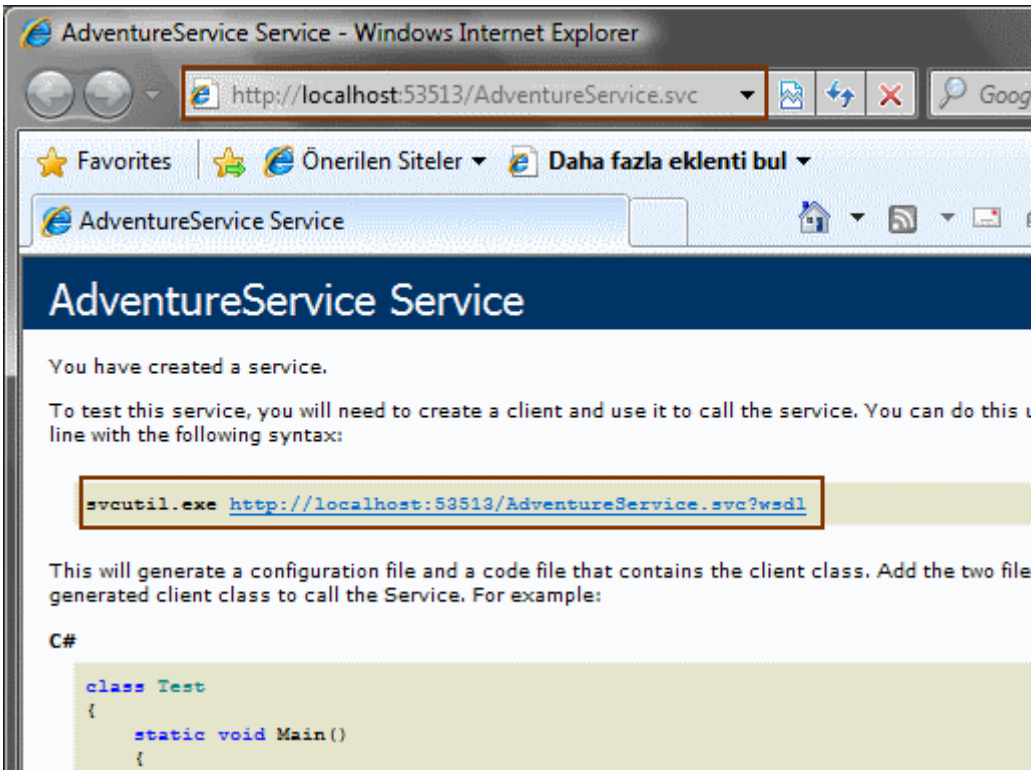
wcf,wcf 4.0,

Merhaba Arkadaşlar,

Nihayet **WCF 4.0** için **basitleştirilmiş konfigurasyon(Simplified Configuration)** yeniliklerinden sonuncusuna değineceğimiz blog girişimize ulaştık. Tabiki **WCF 4.0** tarafındaki diğer yenilikleride zaman içerisinde inceliyoruz olacağız. örneğin **Discovery, Routing, RESTful** geliştirmeleri vb...Ancak diğer köklü değişikliklere başlamadan önce konfigurasyon tarafına son noktayı koyalım artık.

Bir önceki blog yazımızdan hatırlayacağınız üzere **Asp.Net** tabanlı olarak host edilen **WCF** servislerinde, tek bir **svc** dosyası ilede yayınlama yapabileceğimizi görmüştük. **Asp.Net hosting** tarafını ilgilendiren bir diğer yenilik ise **Url Rewriting** konusunun **Svc** dosyaları için de uyarlanabilir olmasıdır. Bildiğiniz gibi

özellikle **RESTful** servislerde, **URL** satırında yer alan bilginin daha okunaklı ve anlaşılır olması söz konusudur ve önemlidir. Genellikle **IIS** tarafında veya kod yardımıyla gerçekleştirilebilecek bu işlemler için **WCF** tarafı konfigurasyon bazında bir kolaylık getirmektedir. Yine bir önceki blog yazımızda yer alan örneği göz önüne alırsak son hali ile aşağıdaki gibi çalıştırıldığını hatırlayabiliriz.



Burada görüldüğü gibi servisin adresi **http://localhost:53513/AdventureService.svc** şeklindedir. Bir başka deyişle tipik bir dosya uzantısı talebi(svc file request) ifade edilmektedir. Ancak istenirse servise olan talebin,

http://localhost:53513/**Companies/Adventure/ProductInformations**

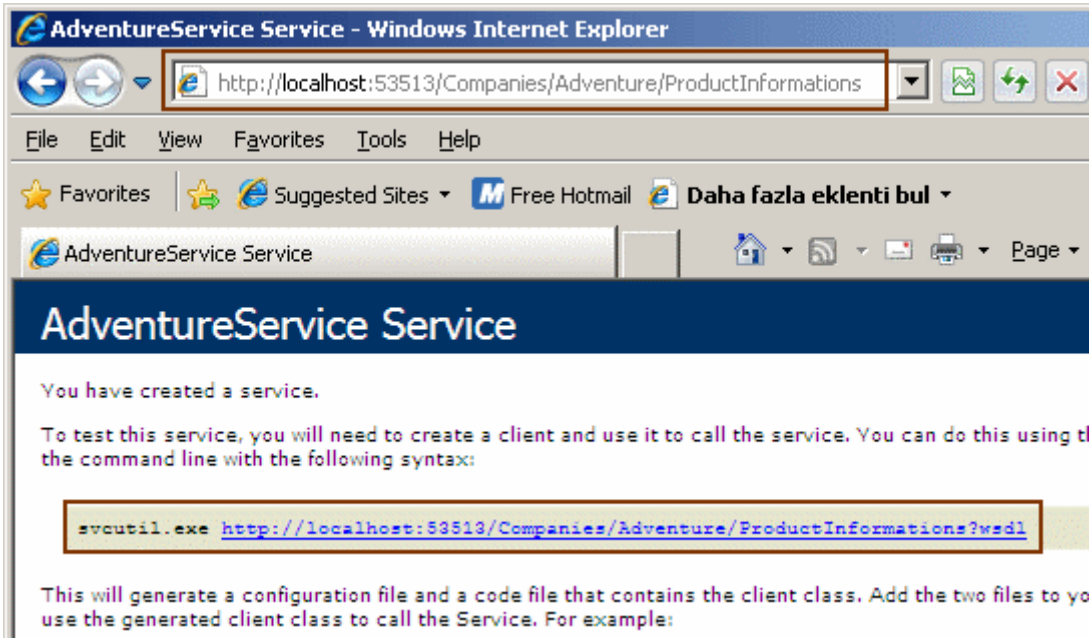
şeklindeki bir URL tanımlaması ile olmasıda sağlanabilir. Bu yazım, okunaklığı ve herşeyden önemlisi servis amacını çok daha net ifade edebilecek bir model sunmaktadır. Peki bu tanımlamanın WCF tarafındaki geçerliliği özel kod yazmadan veya IIS'e bulaşmadan nasıl sağlanabilir?

WCF 4.0 açısından olaya bakıldığında yapılması gereken tek şey **web.config** dosyası içerisinde aşağıdaki eklentilerin yapılmasıdır.

```
<system.serviceModel>
  <behaviors>
   <serviceBehaviors>
    <behavior>
     <serviceMetadata httpGetEnabled="true"/>
```

```
      </behavior>
    </serviceBehaviors>
  </behaviors>
  <serviceHostingEnvironment>
    <serviceActivations>
      <add relativeAddress="/Companies/Adventure/ProductInformations"
service="AdventureService"/>
    </serviceActivations>
  </serviceHostingEnvironment>
</system.serviceModel>
```

Görüldüğü gibi **serviceHostingEnvironment** elementi içerisinde yer alan **serviceActivations** boğumuna yeni bir **Relative Address** bilgisi eklenmiştir. Buna göre**/Companies/Adventure/ProductInformations** bilgisi aslında **AdventureService** isimli servisi işaret etmektedir. Dolayısıyla **WCF çalışma zamanı(Runtime)** gelen talepteki **URL** bilgisini değerlendirip hangi servisi ayağa kaldırması/çalıştırması gerektiğini söz konusu konfigurasyon içeriğinden anlayabilmektedir. Geliştirdiğimiz örneğin bu şekilde çalıştırılması halinde aşağıdaki ekran görüntüsünde yer alan sonuçlar ile karşılaşırız.



Hatta WSDL içeriğide aynı adresleme formatı üzerinden elde edilebilmektedir. çok doğal olarak **WSDL** içeriğinde de **SOAP** adres tanımlamaları ile ilişkili bölümlerde yeni adresleme bilgisi kullanılmaktadır.

```
- <wsdl:service name="AdventureService">
  - <wsdl:port name="BasicHttpBinding_AdventureService"
      binding="tns:BasicHttpBinding_AdventureService">
      <soap:address
        location="http://localhost:53513/Companies/Adventure/ProductInformations/AdventureService" />
    </wsdl:port>
  </wsdl:service>
```

Sırada Discovery, RESTFul ve Routing gibi yeniliklerin incelenmesi var. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

SimpleAspNetHosting2.rar (13,94 kb)

WCF 4.0 Yenilikleri - Basitleştirilmiş Asp.Net Hosting [Beta 1] (2009-08-18T12:10:00)

wcf 4.0,wcf,

Merhaba Arkadaşlar,

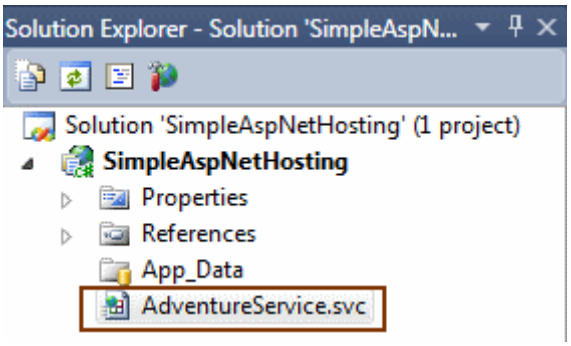
WCF 4.0 ile birlikte gelen yenilikler bitmek bilmiyor. Aslında irili ufaklı bu değişikliklerin ilk bölümünde daha çok **basitleştirilmiş konfigurasyon(Simplified Configuration)** özellikleri üzerinde durmaya çalışıyoruz. Bu değişiklikler irili ufaklı olsalarda **WCF çalışma zamanında(WCF Runtime)** ciddi geliştirmelerin yapıldığını göstermektedir. Gelen değişikliklerden biriside **Asp.Net Hosting** tarafındadır. Aslında konunun sonuna geldiğimizde "Ben bunu bir yerlerden hatırlıyorum" diyebilirsiniz.

İlk olarak **.Net Framework 4.0** öncesinde **Asp.Net Hosting** tabanlı olaraktan bir **WCF** servisini sunmak için neler yaptığımıza bir bakalım;

1. **Svc** uzantılı bir dosya oluşturulur.
2. **Svc** uzantılı dosyanın **code-behind** parçasında gerekli geliştirmeler yapılır. Burada **servis sözleşmesi(Service Contract)** için bir **interface** tanımlaması ve servisi uygulayan tipin kendisinin yazılması söz konusudur. Elbette arayüzün **ServiceContract,** operasyonlarının **OperationContract**, eğer operasyon dönüş tipi olarak kullanılan serileştirilebilir tipler var ise **DataContract** ve özelliklerininde **DataMember** nitelikleri ile işaretlendiğini hatırlayalım.
3. **Web.config** dosyası içerisinde **system.serviceModel** elementlerinde gerekli bildirimler yazpılır. (**Endpoint**, **behavior** tanımlamaları vb...)

WCF 4.0 tarafında ise sadece aşağıdaki gibi bir **svc** dosyası oluşturulması yeterlidir.

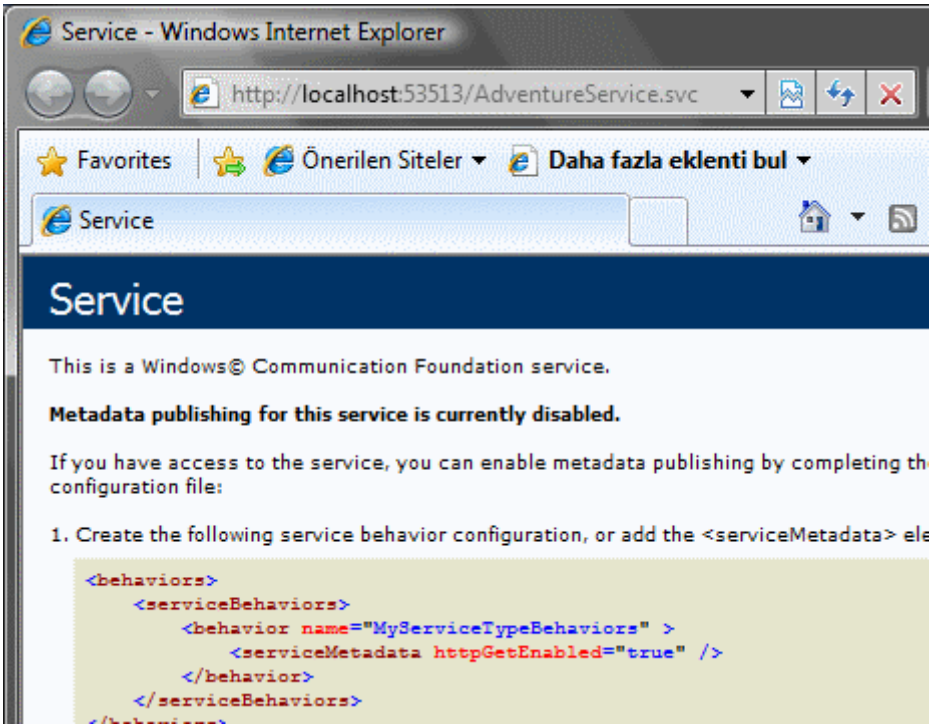
Solution içerisindeki örnek görüntü;



AdventureService.svc isimli dosya içeriği;

```
<%@ ServiceHost Language="C#" Service="AdventureService"  %>

[System.ServiceModel.ServiceContract]
public class AdventureService
{
   [System.ServiceModel.OperationContract]
   public double MostExpensiveProduct(int categoryId)
   {
      return 1000.00;
   }

   [System.ServiceModel.OperationContract]
   public Product GetMostExpensiveProduct(int categoryId)
   {
      return new Product { ProductId = 1001, Name = "Ferrari" };
   }
}

[System.Runtime.Serialization.DataContract]
public class Product
{
   [System.Runtime.Serialization.DataMember]
   public int ProductId { get; set; }
   [System.Runtime.Serialization.DataMember]
   public string Name { get; set; }
}
```

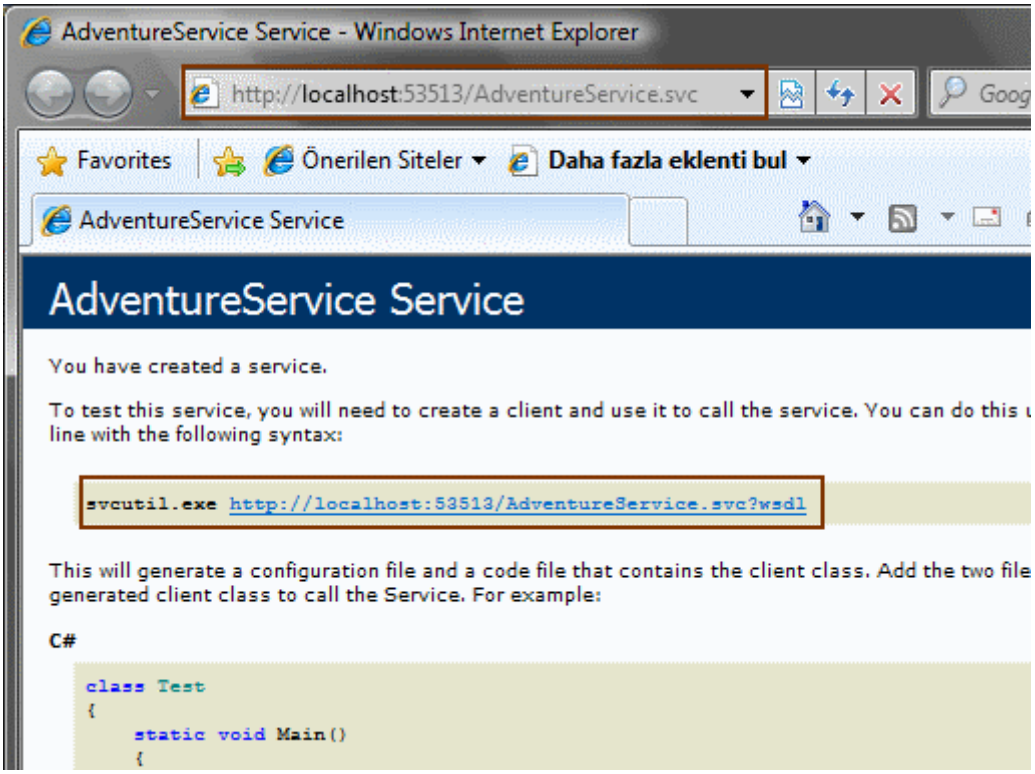
Konfigurasyon(web.config) dosyası olmasına
ve içinde **system.serviceModel** elementlerinin bildirimine gerek yoktur. **Servis sözleşmesi(Service Contract)** için herhangibir **Interface** tanımlaması yapılmamıştır. örneği bu şekilde doğrudan çalıştırabiliriz.*(örneği .Net Framework 4.0 Beta 1 üzerinde Visual Studio 2010 Beta 1 ile geliştirdiğmizi hatırlatmak isterim.)*



Gördüğünüz gibi servis başarılı bir şekilde çalışmaktadır. Elbette istenirse bazı özel ayarlar için **web.config** dosyasında **system.serviceModel** üzerinde geliştirmelerin yapılması gerekebilir. Söz gelimi şu anda çalışmakta olan servisimiz **HTTP** bazlı **metadata publishing** yapmamaktadır. Bir başka deyişle istemcilerin **proxy** üretimi için **servisin metadata**bilgilerine ulaşmaları engellenmiştir. Bu durumda yine **WCF 4.0** ile gelen kolaylıklardan yararlanabiliriz*(Name gibi niteliklerin kullanılma zorunluluğunun ortadan kaldırıldığını hatırlayalım)* Buna göre uygulamaya bir **web.config** dosyası ilave edip, **system.serviceModel** elementini aşağıdaki gibi geliştirmemiz yeterlidir.

```
<system.serviceModel>
   <behaviors>
     <serviceBehaviors>
       <behavior>
         <serviceMetadata httpGetEnabled="true"/>
       </behavior>
     </serviceBehaviors>
   </behaviors>
 </system.serviceModel>
```

Bu durumda uygulamayı tekrar çalıştırdığımızda, **proxy** üretimi için **metadata** yayınlamasının etkineştirildiğini görebiliriz.



Hatta **WSDL** içeriğine bakacak olursak, varsayılan **Endpoint** bilgisininde aşağıdaki şekilde olduğu gibi eklendiğini görebiliriz.

```
      soapAction="http://tempuri.org/AdventureService/GetMostExpensiveProduct"
      style="document" />
    - <wsdl:input>
      <soap:body use="literal" />
    </wsdl:input>
    - <wsdl:output>
      <soap:body use="literal" />
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
- <wsdl:service name="AdventureService">
  - <wsdl:port name="BasicHttpBinding_AdventureService"
      binding="tns:BasicHttpBinding_AdventureService">
      <soap:address
        location="http://localhost:53513/AdventureService.svc/AdventureService" />
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Peki bu yenilikler size neyi çağrıştırıyor? 🙂

Xml Web Servislerini hatırlayın.
Sadece **WebService** ve **WebMethod nitelikleri(attribute)** ile imzalanan tip ve üyeleri söz konusudur. **Web.config** içerisinde herhangibir bildirim yapmaya gerek yoktur.
Dolayısıyla **WCF** için **Asp.Net Hosting** tarafına, Xml Web Servislerindeki çalışma zamanı kolaylığının getirildiğini düşünebiliriz.

SimpleAspNetHosting.rar (12,40 kb)

Interpreter Tasarım Kalıbı - İkinci Randevu (2009-08-16T05:37:00)

design patterns,oop,c#,

Merhaba Arkadaşlar,

Bir süre önce tasarım kalıplarından Interpreter desenini incelemiş ve konu ile ilişkili bir kural mo
yazılabileceğini araştıracağımızdan bahsetmiştik. Interpreter tasarım kalıbında hatırlayacağınız gi
bulunmaktadır. NonTerminal tipler genellikle kural motoru gibi modellerde devreye girmektedir.
Engine), işletilmek istenen ifadelerin içerisinde sıklıkla operatörlerin kullanılması söz konusudur

örneğin **and, or, >=, <, küçüktür, eşittir** gibi düşünebiliriz. Dikkat
ederseniz **eşittir** ve **küçüktür** gibi kelimeleri de operatörler arasına kattım. Nitekim
yorumlanacak**ifade(Expression)** bütününü kendimiz oluşturduğumuz için istediğimiz
terimleri seçmemiz son derece doğaldır. Tam bu noktada sağ üstteki resmin konu ile ne
alakası olduğunu düşünebilirsiniz.

Aslında bu yazımızdaki amacımız, içerisinde değişik renklerde misketleri barındıran bir
kutu*(ki örneğimizde string tipten generic bir koleksiyon olarak ifade
edilecek)* üzerinde, **string**bazlı mantıksal bir ifadeyi işletmektir. örnek olarak aşağıdaki gibi
bir kural tanımladığımızı göz önüne alabiliriz.

"Kirmizi ve Mavi veya Mor"

Buna göre sepet içerisindeki misketlerin rengine göre yukarıdaki kurala uyan bir durum
varsa bir takım işlemlerin yapılmasını veya yapılmamasını arzu ediyoruz. Aslında ne
yapılması gerektiğinin şu aşamada bir önemi yok. çünkü önemli olan ilk aşama,
yukarıdaki kuralı söz konusu misket sepeti üzerinde işletebilmek. Peki bunu nasıl
yapacağız? Dahası yapmak için **Interpreter** tasarım kalıbını nasıl kullanacağız?

İlk etapta, kural ifadesi içerisindeki materyalleri göz önüne almamızda yarar var. Renkleri
aslında tek bir **Terminal** tipi ile ifade edebiliriz. Nitekim renklerin ayrı ayrı yapacakları bir
işlevsellik yok.(*Elbetteki kural ifadesi içerisindeki bilgilerin gerçek hayat kural
motorlarında ayrı ve farklı görevleri olabilir. Bu durumda her biri için
ayrı **Terminal** tiplerinin tasarlanması gerekir)* Diğer taraftan renkler arasında **ve,
veya** olmak üzere iki mantıksal operatör yer almaktadır. İşte bunlar **NonTerminal** tipler
olarak tasarlanmalıdır. Nitekim kendi içlerinde, **Expression** tiplerinden ikisini

taşıyacaklardır ki mantıksal olarak **ve, veya** işlemleri gerçeklenebilsin. Tabi birde içerisinde parantezler bulunmayan bir kural ifadesi ile karşı karşıyayız. Kuralın

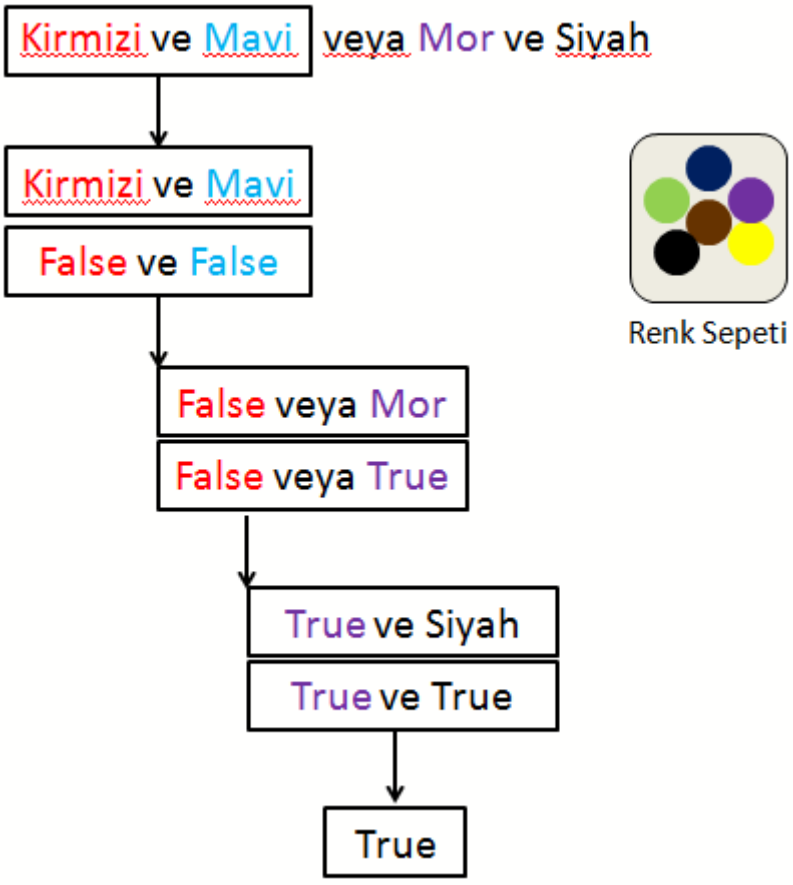
"Kirmizi ve (Mavi veya Mor)" olması ile

"(Kirmizi ve Mavi) veya Mor"

olmasının arasında işlem öncelikleri açısından farklılıklar bulunur. önce parantez içlerini çalıştırmak gerekir. Tabi bizim örneğimizde parantezleri işin içerisine şu an için katmıyor olacağız. Ama size parantezleri işin içerisine katarak geliştirme yapmaya çalışmanızı şiddetle öneririm. özellikle string biçimdeki kuralı ayrıştırırken çok zorlu bir yoldan geçeceğinizi garanti edebilirim. öyleki kuralı bir arkadaşınız yanlışlıkla şöylede yazabilir.

"(Kirmizi ve **((**Mavi veya Mor)"...Upsss!

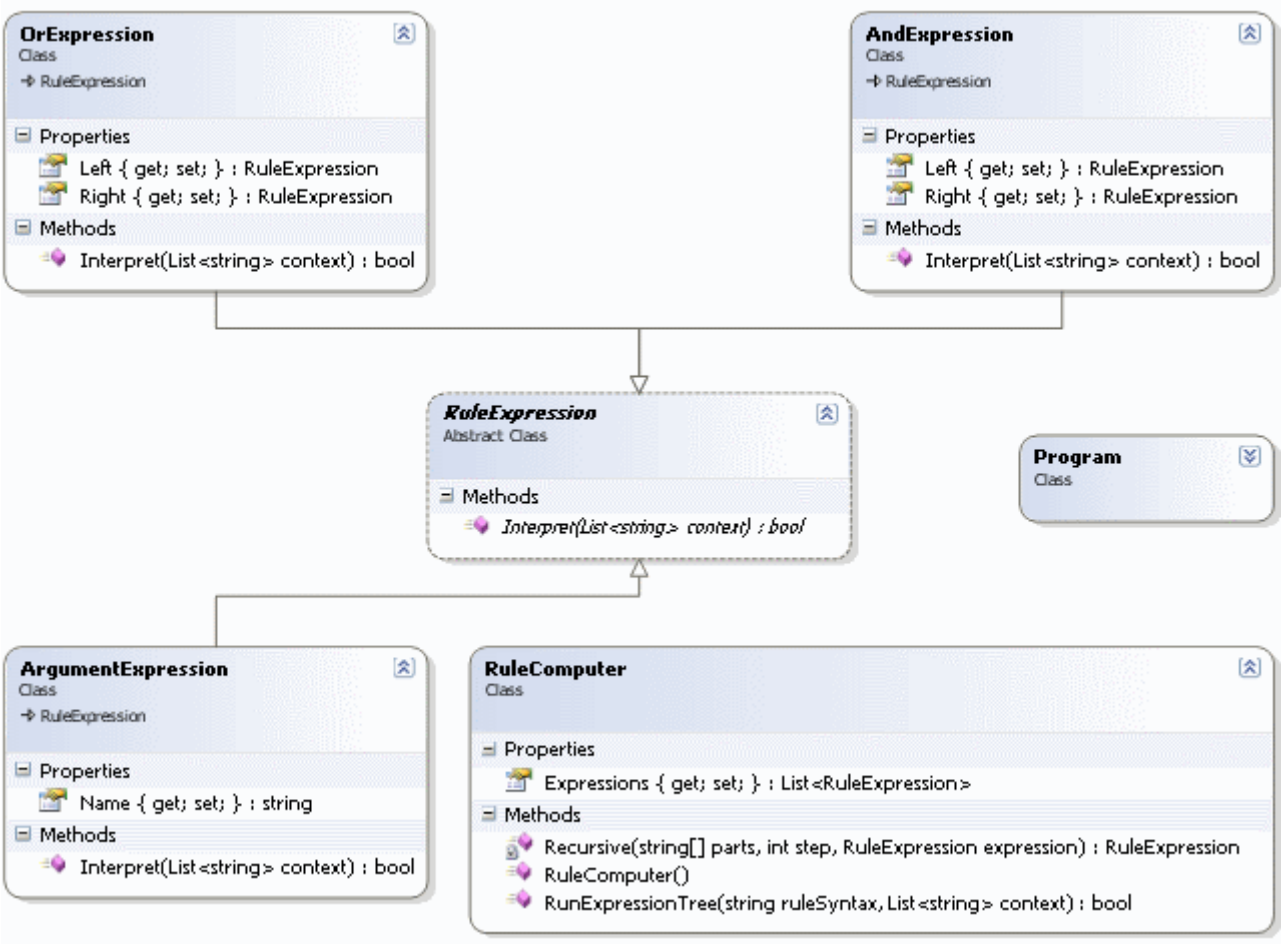
Peki biz kuralı nasıl ayrıştıralım. Aşağıdaki şekil bize bu anlamda bir fikir verebilir.



Aslında bunun programatik taraftaki karşılığını bir **ifade ağacı(Expression Tree)** olarak düşünebiliriz. Ancak yazacağımız kod içerisinde **Interpreter** tasarım kalıbının uygulanması dışında, bu şekilde bir ifade ağacının çıkartılabilmesi için **Recursive** bir fonksiyonada ihtiyacımız olacaktır. Ta ta ta taaa...

*(**Kişisel Notum :** Uzun yıllar çalıştığım eğitim firmasında verdiğim .Net derslerinde, Recursive metodları anlatırken çoğunlukla Faktoryel hesabı veya Fibonacci sayılarının bulunması problemlerini dile getirdiğimi hatırlıyorum da...Gerçek hayat çok ama çok daha farklı...Geniş düşünmek, vizyonu her zaman geniş tutmak gerekiyor. çoğu zaman göz ardı ettiğiniz bir kavram, aslında bir problemin çözümünde kritik bir rol üstlenebiliyor. Recursive bir metodun örneğimizdeki ifade ağacının çıkartılmasında üstlendiği rolde olduğu gibi...)*

çünkü ifadenin n sayıda renk ve mantık operatörü içermesi söz konusudur. Bu durumda ifade ağacı oluşturulurken ve çalıştırılırken, bir önceki ifadeyi üreten ve bunu sonraki ifadeyi üretmek için girdi olarak kullanan bir fonksiyon yazılması şarttır. Artık örneğimizi geliştirmeye ne dersiniz? Şimdi aşağıdaki sınıf diagramı ve kodları içeren Console uygulamasını yazdığımızı düşünelim.




```
using System;
using System.Collections.Generic;

namespace Interpreter
{
   // Expression Type
   abstract class RuleExpression
```

```
{
   public abstract bool Interpret(List<string> context);
}

#region Terminal Expression Types

class ArgumentExpression
: RuleExpression
{
   public string Name { get; set; }

   public override bool Interpret(List<string> context)
   {
      if(context.Contains(Name))
         return true;
      else
         return false;
   }
}

#endregion

#region NonTerminal Expression Types

class AndExpression
   : RuleExpression
{
   public RuleExpression Left { get; set; }
   public RuleExpression Right { get; set; }

   public override bool Interpret(List<string> context)
   {
      return Left.Interpret(context) && Right.Interpret(context);
   }
}

class OrExpression
   : RuleExpression
{
   public RuleExpression Left { get; set; }
   public RuleExpression Right { get; set; }

   public override bool Interpret(List<string> context)
   {
      return Left.Interpret(context) || Right.Interpret(context);
```

```
    }
}

#endregion

// Expression ağacını oluşturmak ve çaşlıştırmakla görevli olan sınıf
class RuleComputer
{
    public List<RuleExpression> Expressions { get; set; }

    public RuleComputer()
    {
        Expressions = new List<RuleExpression>();
    }

    // Expression ağacının oluşturucusu ve çalıştırıcısı olan metoddur
    public bool RunExpressionTree(string ruleSyntax,List<string> context)
    {
        bool result = false;

        // önce kural metni içerisindeki boşluklara göre elemanlar ayrılır
        string[] ruleParts = ruleSyntax.Split(' ');

        // Küçük bir kontrol. Ancak fazlasınıda yapmak gerekir :) Yazılan kural metninin geçerli olup olmadığı denetlenmelidir.
        if (ruleParts.Length < 3)
            throw new Exception("Eleman sayısı kural için yeterli değildir");

        // Expression Tree oluşturulmasına başlanır(Recursive fonksiyonu kullandığımıza dikkat edelim)
        RuleExpression longExpression = Recursive(ruleParts, 1, null);
        // Expression ağacı koleksiyona eklenir
        Expressions.Add(longExpression);

        // Koleksiyondaki her bir Expression için Interpret operasyonu çalıştırılır
        foreach (RuleExpression expression in Expressions)
        {
            result = expression.Interpret(context);
        }

        return result;
    }

    // Expression ağacının oluşturulması için kullanılan recursive fonksiyon
    // Kuralı işletmek için en soldaki ikili daldan başlayarak sağa doğru ilerliyoruz
```

```
    RuleExpression Recursive(string[] parts, int step, RuleExpression expression)
    {
      if (step == 1) // Soldan ilk operatör ile karşılaşıldığında
      {
        if (parts[step] == "ve")
        {
          expression = new AndExpression { Left = new ArgumentExpression { 
Name = parts[step - 1] }, Right = new ArgumentExpression { Name = parts[step + 1] } 
};
        }
        if (parts[step] == "veya")
        {
          expression = new OrExpression { Left = new ArgumentExpression { Name 
= parts[step - 1] }, Right = new ArgumentExpression { Name = parts[step + 1] } };
        }
      }
      else // İlk çift içerisindeki operator geçildikten sonra, her zaman bir önceki dalın, 
sonradan gelen argüman ile ve/veya işlemine sokulması sağlanır
      {
        if (parts[step] == "ve")
        {
          expression = new AndExpression { Left = expression, Right = new 
ArgumentExpression { Name = parts[step + 1] } };
        }
        if (parts[step] == "veya")
        {
          expression = new OrExpression { Left = expression, Right = new 
ArgumentExpression { Name = parts[step + 1] } };
        }
      }

      // Recursive metoddan bir notkada çıkılması gerekecektir. Bu çıkış noktası, son 
operatör ele alındıktan sonrasıdır.
      if (step == parts.Length - 2)
        return expression;

      // öteleme yapılarak sonraki çifti almak üzere aynı metod tekrar işletilir
      return Recursive(parts, step + 2, expression);
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
```

```
        // örnek kural
        string rule = "Kirmizi ve Mavi veya Mor ve Siyah";

        // Kuralın denetleneceğin veri içeriği (Context)
        List<string> myBasket =new List<string> { "Yesil", "Kahverengi",
"Lacivert", "Sari", "Mor", "Siyah" };
        RuleComputer computer = new RuleComputer();

        // Kirmizi ve Mavi = 0 && 0 => 0
        // 0 veya Mor = 0 || 1 => 1
        // 1 ve Siyah = 1 && 1 => 1
        bool result=computer.RunExpressionTree(rule,myBasket);
        Console.WriteLine(result);

        // Kirmizi ve Mavi = 0 && 0 => 0
        // 0 veya Mor = 0 || 0 => 0
        // 0 ve Siyah = 0 && 0 => 0
        myBasket = new List<string> { "Yesil", "Kahve", "Lacivert", "Beyaz" };
        Console.WriteLine(computer.RunExpressionTree(rule,myBasket));

        // Kuralı değiştirelim
        rule = "Kirmizi veya Beyaz";

        // Kirmizi veya Beyaz = 0 || 1 => 1
        Console.WriteLine(computer.RunExpressionTree(rule,myBasket));

        // Exception testidir
        // rule = "Sari";
        // Console.WriteLine(computer.RunExpressionTree(rule, myBasket));
    }
  }
}
```

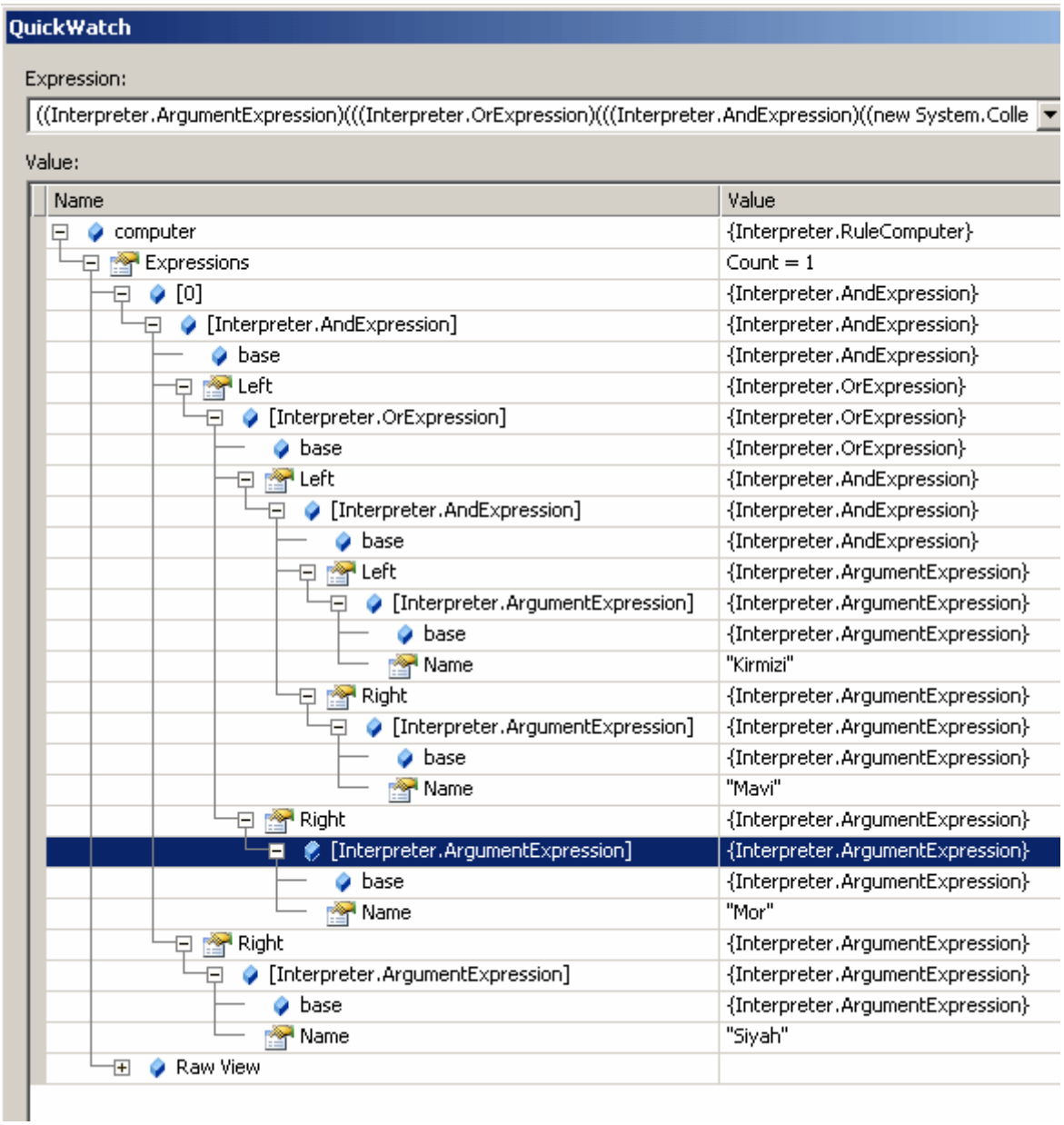
Kodu dikkatlice incelemenizi öneririm.

Tasarım kalıbımıza göre, **AndExpression** ve **OrExpre**
renklerin her birini **ArgumentExpression** tipi ile tems
taraflarındaki nesneleri kullanabilmek için **RuleExpres**
tipi **RuleComputer** sınıfı.

Tabir yerinde ise, **Interpreter** kalıbının önüne geçtiğini söyleyebiliriz. **RuleComputer** içerisinde yer alan **RunExpressionTree** metodu, ifade ağacının oluşturulması ve çalıştırılmasından sorumludur. Bu metodda kendi içerisinde **Recursive** olan başka bir fonksiyonu çağırmaktadır. Yazımızın başlarında hatırlayacağınız üzere örnek bir kuralı soldan sağa doğru yorumlayarak ele aldığımızı görmüştük. Burada kuralın n sayıda argüman ve operatörden oluşturulması söz konusu olduğundan, ifade ağacının çıkartılmasının tek yolu kendi kendini çağıran ve bir önceki çağırımda oluşturduğu ifadeyi kullanan bir metod yazmaktır.

Main metodu içerisinde bir kaç test kuralı yazıldığını ve işletildiğini görmekteyiz. Kuralları işletiş şekline göre, **ArgumentExpression** tipine ait **Interpret** metodu içerisinde yaptğımız tek şey, parametre olarak gelen **Context***(yani renk bilgilerini içeren generic List koleksiyonu)* içerisinde, söz konusu referansın taşıdığı rengin olup olmadığına bakmak ve buna göre geriye **true** veya **false** sonuç döndürmektir.

Uygulamamızı **debug** ederek çalıştırdığımızda ise son derece güzel noktalara ulaştığımızı görebiliriz Söz gelimi ilk kuralın işletilmesi sırasında **RuleComputer**içerisindeki **Expressions** özelliğinin aşağıdaki yapıda olduğunu hemen farkedebiliriz.



Dikkat edileceği üzere **string** tabanlı yazılan basit kuralın her bir parçası **Exrpression Tree** üzerinde nesnel olarak yerini almış ve birbirlerine bağlanmıştır. Bundan sonrasında kodun yapması gereken tek şey, ağacı ilk elemandan sonuncuya kadar dolaşmak ve tüm gördüğü **RuleExpression** türevli tipler için **Interpret** metodlarını çağırmaktır. Ve işte çalışma zamanı sonucu;

Peki neler yapamıyoruz?

- Herşeyden önce sadece **ve, veya** operasyonlarına hizmet veren bir sistem söz konusu. Buna **ancak** operatörünüde ekleyebiliriz.
- Diğer yandan, parantez yazımına destek verilmesi söz konusu olabilir. Bu duruma **Expression Tree'** nin oluşturulması sırasında parantez kullanımlarını değerlendirmemiz gerekecektir.
- Kural olarak yazılan ifade bütününün, gerçekten doğru bir stilde yazıldığını denetlemek gerekir. Bitişik yazımlar yada tanımlı olmayan bir operatör(ve yerine yahu yazmış olabiliriz) hatalara neden olabilir.
- ...

Maddeler elbetteki çoğaltılabilir. Ancak sonuçta ulaştığımız noktalardan birisi, belirli bir **Context** üzerinde, bizim belirlediğimiz bir kuralın işletilmesi ve sonuç olarak **true** yada **false**değere indirgenebilen bir çıktının ürettirilebilmesidir. Bir başka deyişle bu yapıyı esnetmek*(örneğin true/ false haricinde diğer tiplerin üretimine destek vermek yada =, != gibi çift taraflı karşılaştırma operasyonları hesaba katabilmek...)* tamamen klavyenin başındaki geliştiricini hayal gücü ile sınırlıdır. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

InterpreterV2.rar (26,55 kb)

WCF 4.0 Yenilikleri - Standard Endpoints [Beta 1] (2009-08-14T12:15:00)

wcf,wcf 4.0,

Merhaba Arkadaşlar,

Bir süredir **WCF 4.0** ile birlikte gelen yenilikleri tek tek incelemeye çalışıyoruz. İlk incelediğimiz noktalar konfigurasyon ayarları üzerinde yapılmış olan basitleştirmeleri içermektedir. Bu değişimlerden bir diğerini inceleyerek serimize devam ediyor olacağız. Bu anlamda konumuz **Standard Endpoints** başlığı altında gelen yeniliklerdir. Bu özelliği inceledikten sonra konuyu anlamanın en iyi yolunun bir önceki versiyonda ne olduğuna bakmak olduğuna karar verdim. Senaryomuza göre **Http** üzerinden sunulan bir servis için **mexHttpBinding bağlayıcı tipini(Binding Type)** kullanarak **Metadata Publishing** işlemini gerçekleştiriyoruz. Bir başka deyişle servisi ne yaptığı ve bunu hangi operasyonlar ile tanımladığı bilgisini istemcilere açıyoruz. Metadata üzerinden publishing işleminde anahtar noktanın servis sözleşmesi olarak **IMetadataExchange** arayüzünü kullanmak olduğunu bilmekteyiz. *(Tabi doğrudan IIS üzerinden host edilen bir WCF servisi yazmıyorsak.)* Bunu aklımızın bir köşesinde tutalım. Şimdi örneğimize ait konfigurasyon dosyasını aşağıdaki gibi geliştirdiğimizi düşünelim***(İlk örneğimizi .Net Framework 3.5 tabanlı olarak geliştirdiğimizi hatırlatalım.)***

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
```

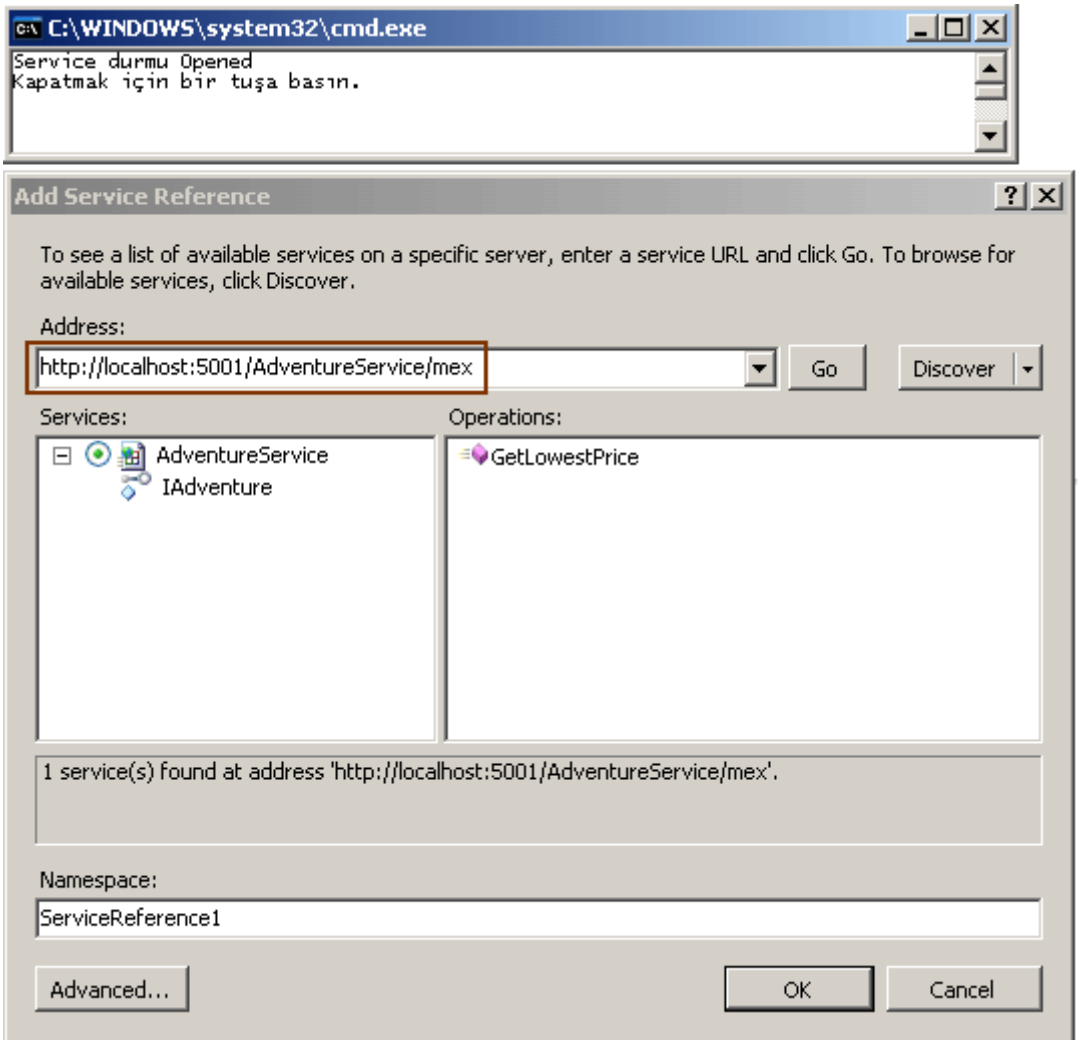
```
      <behaviors>
        <serviceBehaviors>
          <behavior name="AdventureServiceBehavior">
            <serviceMetadata />
          </behavior>
        </serviceBehaviors>
      </behaviors>
      <services>
        <service behaviorConfiguration="AdventureServiceBehavior" name="Previous
Version.AdventureService">
          <endpoint address="" binding="basicHttpBinding" bindingConfiguration=""
            name="EndPoint1" contract="PreviousVersion.IAdventure" />
          <endpoint address="Mex" binding="mexHttpBinding"
bindingConfiguration="" name="MexEndpoint" contract="IMetadataExchange" />
          <host>
            <baseAddresses>
              <add baseAddress="http://localhost:5001/AdventureService" />
            </baseAddresses>
          </host>
        </service>
      </services>
    </system.serviceModel>
</configuration>
```

Konfigurasyon dosyasında dikkat etmemiz gereken bir takım noktalar olduğu açıktır. **MexEndpoint** isimli **Endpoint** içerisinde yer alan **contract** niteliğinin değerini **case-sensitive** olarak doğru sözleşme tipini**(Contract Type)** işaret edecek şekilde yazmalıyız. Bağlayıcı tip olarak **mexHttpBinding** tipini belirtmeliyiz; nitekim **base address** ve **EndPoint1** bilgilerine göre **BasicHttp** bazlı bir yayınlama yapmaktayız. Bunlara ek olarak, **serviceMetadata** niteliğinin bir **servis davranışı (Service Behavior)** olarak mutlaka bildirilmesi gerekmektedir. Aslında **NetTcp** kullansaydıkta metadata sözleşmesi olarak **IMetadataExchange** tipini belirtmemiz gerekiyordu. İlerlemeden önce servisimize ait örnek kod içeriği aşağıdaki gibi geliştirdiğimizi düşünelim.

```
using System;
using System.ServiceModel;

namespace PreviousVersion
{
    [ServiceContract]
    interface IAdventure
    {
        [OperationContract]
        double GetLowestPrice(int category);
```

```
    }
    class AdventureService
        : IAdventure
    {
        public double GetLowestPrice(int category)
        {
            return 100;
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ServiceHost host = new ServiceHost(typeof(AdventureService));

            host.Open();

            Console.WriteLine("Service durumu {0}\nKapatmak için bir tuşa basın.",
host.State.ToString());
            Console.ReadLine();

            host.Close();
        }
    }
}
```

Buna göre servis uygulamamız çalışıyorken, istemciler **proxy** içeriklerini üretebilmek için **metadata** bilgilerini ulaşabilecektir. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.*(Dikkat edileceği üzere servis çalıştırıldıktan sonra **Visual Studio** üzerinden **Add Service Reference** iletişim penceresi ile geliştirilen servisin metadata bilgilerine ulaşılabilmektedir.)*

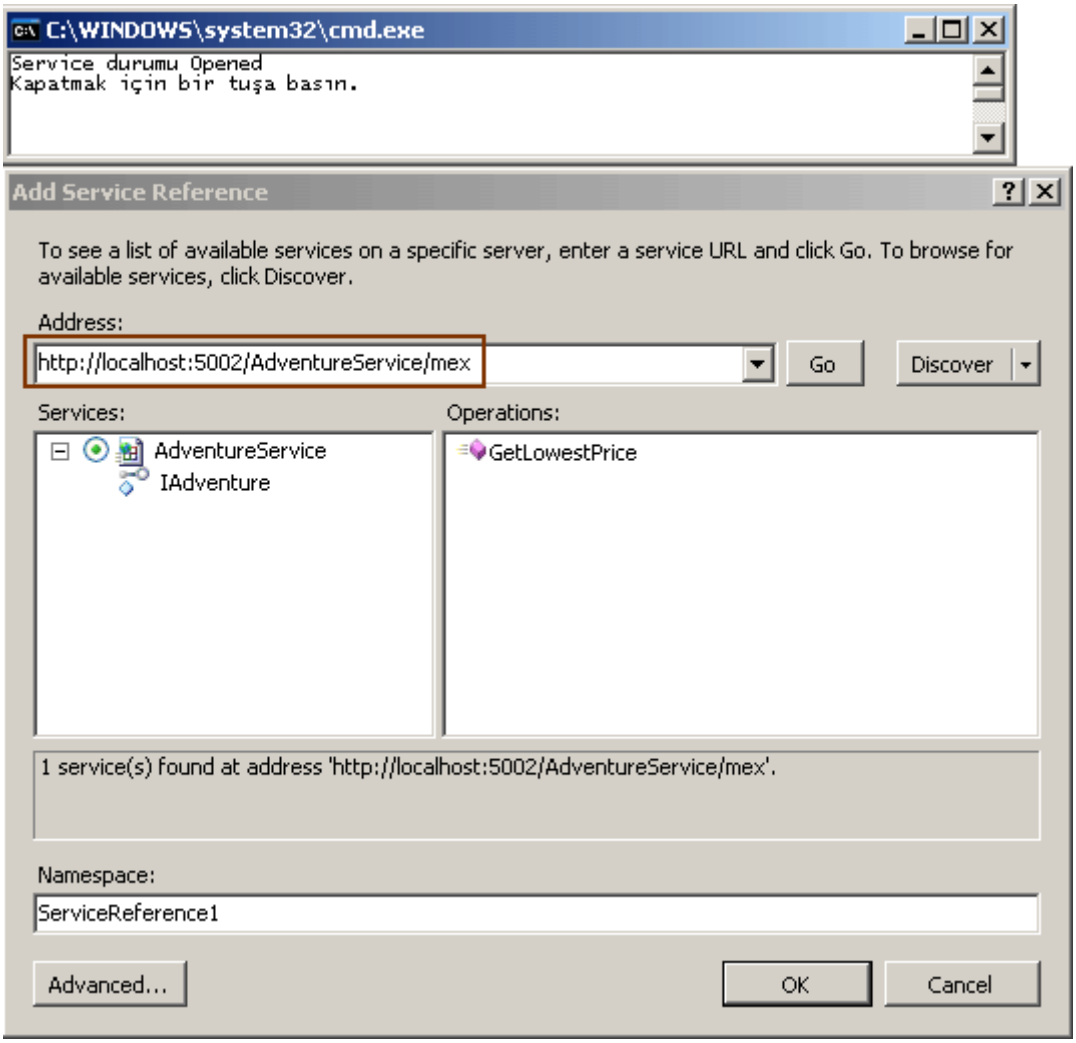


Şimdi **WCF 4.0** açısından duruma bakalım. Yeni gelen **Standard Endpoints** özelliğine göre, önceden tanımlanmış ve pek çok standart özelliği set edilmiş bazı **Endpoint** tanımlamaları gelmektedir. örneğin **Metadata Publishing** sisteminde, **IMetadataExchange** kullanımı bir standarttır. Bu nedenle **mexEndpoint** isimli tipi kullanarak yukarıdaki örneği aşağıdaki config içeriği ile **WCF 4.0** üzerinde gerçekleyebiliriz.

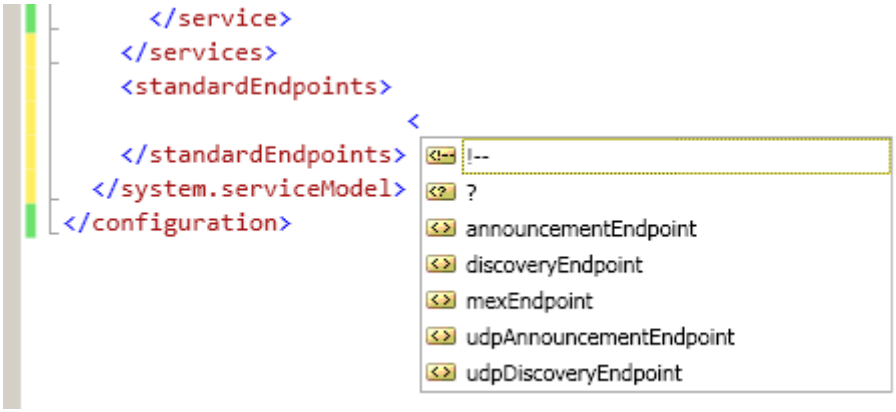
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior>
          <serviceMetadata/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service name="PreviousVersion.AdventureService">
```

```
      <endpoint address="" binding="basicHttpBinding" name="EndPoint1"
contract="PreviousVersion.IAdventure" />
      <endpoint kind="mexEndpoint" address="Mex" />
      <host>
        <baseAddresses>
          <add baseAddress="http://localhost:5002/AdventureService" />
        </baseAddresses>
      </host>
    </service>
  </services>
 </system.serviceModel>
</configuration>
```

Dikkat edileceği üzere **endpoint** elementi içerisinde yer alan **kind** niteliğine **mexEndpoint** isimli bir değer atanmıştır. Bu değer standart olarak **Metadata Exchange** yayınlaması yapacak bir**endpoint** oluşturulmasını **WCF çalışma zamanına(Runtime)** bildirmektedir. örneğin bu versiyonunu çalıştırdığımızda*(.Net Framework 4.0 ve Visual Studio 2010 üzerinde geliştirilmiştir)* bir önceki örnekte olduğu gibi istemci tarafından Metadata bilgisini çekebildiğimizi görürüz.



Ne yazıkki, **mexEndpoint** varsayılan olarak **Http** bazlı **Metadata Publishing**' i desteklemektedir. Yani **NetTcp** kullandığımız durumlarda, binding niteliğini kullanarak bildirim yapmamız şarttır. Bu durumda aslında mexTcpBinding kullanılmasını belirttikten sonra birde IMetadataExchange arayüzünü contract niteliğinde belirtmek çokda fazla bir adım olarak görülmemelidir. Ancak burada bahsedilen standartlaştırılmış endPoint' ler sadece **mexEndpoint** tipinden mi oluşmaktadır? Tabiki hayır. **announcementEndpoint, discoveryEndpoint, udpAnnouncementEndpoint, udpDiscoveryEndpoint, workflowControlEndpoint** gibi başka standart endPoint tipleride tanımlanmıştır. Dikkat çekici noktalardan biriside istenildiğinde bu standart **endPoint** tipleri içinde bazı özel ayarların yapılabileceğidir*(Bu ayarlamaları ve yeni gelen standart endPoint tiplerinden diğerlerini ilerleyen yazılarımızda detaylı bir şekilde incelemeye çalışacağım)* Bu noktada konfigurasyon içeriğinde **standardEndpoints** sekmesini ele almak yeterli olacaktır.



WCF 4.0 tarafında **basitleştirilmiş konfigurasyon(Simplified Configuration)** özellikleri ile ilişkili yenilikleri incelemeye devam ediyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

StandardEndpoints.rar (48,35 kb)

WCF 4.0 Yenilikleri - Default Behavior Configuration [Beta 1] (2009-08-13T19:01:00)

wcf,wcf 4.0,

Merhaba Arkadaşlar,

Bir kaç gece önce evde DVD keyfi yapmak için **A Fistful** Filmi seyretmeyenler için bir kaç hatırlatma yapayım. Filı bir üçlemenin ilk halkasını oluşturmakta. Hepside çok güz dövüş sahneleridir.

Kötü adam, Sarışına(ki bu üçlemenin hiç bir yerinde Client' in adını bilmeyiz. Herkes ona sarışın-Blon der) defalarca ateş eder. Kötü adam keskin nişancıdır(bu filmlerdeki her kovboy gibi) ve sürekli olarak Sarışının göğsüne ateş eder. Ama Sarışın her yere düştüğünde tekrar ayağa kalkar. Sonunda kötü adamın mermisi tükenir. İşte o an...Sarışın üzerindeki kıyati aralar ve altından saç dökümden metal bir yelek çıkar. Kötü adamın şaşkın bakışları arasında film devam eder.

Buradaki metal döküm yelek çok küçük bir ayrıntıdır ama işlevselliği çok kritiktir. Bu işlevselliğin kritik olması bir yana, Sarışının bu detayla ilintili olarak kötü adamdan gelen kurşunları düşünmesinede gerek kalmamıştır. Şimdi bu konuya nereden geldik. İki sebebimiz var.

WCF 4.0 ile gelen yeniliklerin ilk kümesi olan **basitleştirilmiş konfigurasyon(Simplified Configuration)** kabiliyetleri, geliştiricinin bazı ince detayları düşünme zorunluluğunu ortadan kaldırmaktadır. örneğin **Endpoint** eklenmesede varsayılan olanların çalışma zamanında oluşturulması, **bağlayıcılar(Binding Types)** için **name, binding configuration** gibi nitelikleri kullanma zorunluluğunun ortadan kaldırılması veya protocol eşleştirmelerinin kolayca ele alınması vb...Bu birinci nedenimiz. İkinci neden çok daha basittir. Sıkıcı olan bir blog girişi yapmamak... Bu seferki konumuz aslında bir önceki yazımızda değerlendirdiğimiz Default Binding Configuration özelliğinin Behavior için olan versiyonudur. Dolayısıyla bu kez, name ve behaviorConfiguration niteliklerine olan zorunluluğun ortadan kalktığını söyleyerek işin içerisinden çıkabiliriz. Ama tembellik etmeyip araştırmamızdan bir zarar da gelmez.

Olaya ilk olarak **davranış(Behavior)** kavramından başlamakta yarar var. WCF tarafında **Service, EndPoint, Operation veya Contract** gibi seviyelerde çalışma zamanı davranışları belirlenebilir. örneğin bir http bazlı bir servisin metadata publishind desteğinin olup olmaması, Exception detaylarının istemci tarafına gönderilip gönderilmemesi yada bir endPoint için kullanılacak sertifika bilgilendirmelerinin tasarlanması. Normal şartlar altında WCF 3.5 ile geliştirilen bir servis uygulamasında davranış tanımlamaları için config dosyalarını aşağıdaki örnekte olduğu gibi kullanırız.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="CalculusServiceBehavior">
          <serviceDebug includeExceptionDetailInFaults="true" />
         <dataContractSerializer maxItemsInObjectGraph="3"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service
```

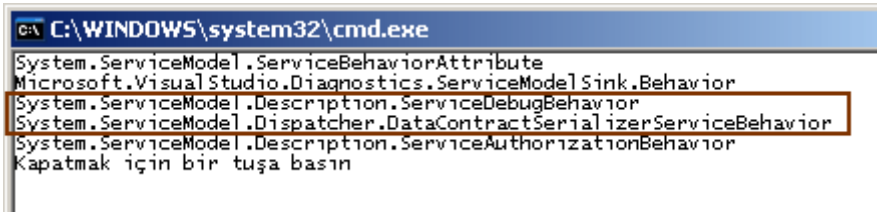
```
behaviorConfiguration="CalculusServiceBehavior" name="NewVersion.Calculus">
            <endpoint address="net.tcp://localhost:4500/Calculus" binding="netTcpBinding"
              bindingConfiguration="" name="EndPoint1"
contract="NewVersion.ICalculus" />
            <endpoint address="http://localhost:4501/Calculus" binding="basicHttpBinding"
              name="EndPoint2" contract="NewVersion.ICalculus" />
          </service>
        </services>
    </system.serviceModel>
</configuration>
```

Bu içeriğe göre servis için iki çalışma zamanı davranışı belirlenmiştir. **serviceDebug** ve **dataContractSerializer**. Bizim konsantre olacağımız nokta ise behavior elementinin **name**niteliğinin değeri ile, service elementinin **behaviorConfiguration** niteliğinin değerinin aynı olmasıdır. Dolayısıya servisler veya EndPoint' ler hangi davranışları kullanacaklarını**behaviorConfiguration** niteliklerinde bildirirler. E tabi bu config içeriğini ele alan bir uygulamayı test etmessek olmaz. Yenilikten emin olabilmemiz ve ispat edebilmemiz için bu şart. Bu nedenle sıkılmadan aşağıdaki kodları yazalım.(Tam bir Matematik Mühendisi gibi olaya yaklaşmaktan kendimi alamadığımı söyleyebilirim. İspat, ispat, ispat...

```
using System;
using System.Linq;
using System.ServiceModel;

namespace NewVersion
{
    [ServiceContract]
    interface ICalculus
    {
        [OperationContract]
        double Sum(params double[] values);
    }

    class Calculus
        : ICalculus
    {
        public double Sum(params double[] values)
        {
            return values.Sum();
        }
    }
```

```
   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(Calculus));
         host.Open();

         foreach (var behavior in host.Description.Behaviors)
         {
            Console.WriteLine(behavior.ToString());
         }

         Console.WriteLine("Kapatmak için bir tuşa basın");
         Console.ReadLine();

         host.Close();
      }
   }
}
```

örneğimizde tek yaptığımız servisin açılmasından sonra yüklü olan davranışlarını listelemektir. örneği çalıştırdığımızda aşağıdaki sonuçları elde ederiz.

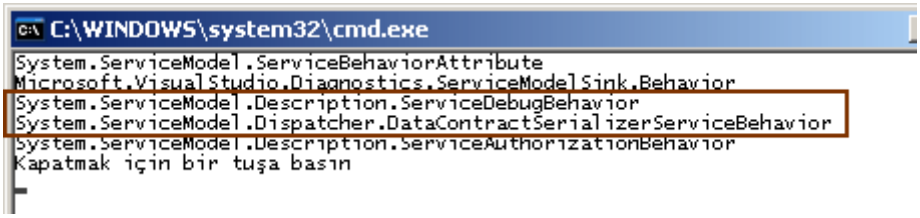


WCF 4.0 tarafında ise config dosyası içerisinde davranışlar için name ve behaviorConfiguration isimli niteliklerin kullanılma zorunluluğu yoktur. Dolayısıyla config dosyasını aşağıdaki gibi değiştirirsek,

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior><!-- name niteliği kullanılmamıştır.-->
          <serviceDebug includeExceptionDetailInFaults="true" />
          <dataContractSerializer maxItemsInObjectGraph="3"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
```

```
    <service name="NewVersion.Calculus"><!-- behaviorConfiguration niteliği
kullanılmamıştır-->
      <endpoint address="net.tcp://localhost:4500/Calculus" binding="netTcpBinding"
        name="EndPoint1" contract="NewVersion.ICalculus" />
      <endpoint address="http://localhost:4501/Calculus" binding="basicHttpBinding"
        name="EndPoint2" contract="NewVersion.ICalculus" />
    </service>
  </services>
 </system.serviceModel>
</configuration>
```

ve örneğimizi yeniden çalıştırırsak bir önceki ile aynı sonuçları elde ettiğimizi görürüz.



Tabi son örneğimizin **.Net Framework 4.0 Beta 1** tabanlı olarak **Visual Studio 2010 Beta 1** üzerinde geliştirildiğini hatırlatalım. Basit çok basit bir özellik ama merak etmeyin. WCF 4.0 ile ilişkili daha baba yeniliklerde bulunmakta. İlerleyen yazılarımızda bunlarada değiniyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

DefaultBehaviorConfiguration.rar (43,75 kb)

WCF 4.0 Yenilikleri - Default Binding Configuration [Beta 1] (2009-08-12T12:30:00)

wcf 4.0,wcf,

Merhaba Arkadaşlar,

WCF 4.0 ile birlikte gelmesi muhtemel yenilikleri incelemeye kaldığımız yerden devam ediyoruz. Bu yazımızda ele alacağımız konu, **config** dosyası içerisinde kullanılan **bağlayıcı tipe(Binding Type)** özel konfigurasyon ayarları ile ilişkili olacak. Konuyu net bir şekilde anlayabilmek için **.Net Framework 3.5** tabanlı olarak geliştirilmiş basit bir servis uygulaması ile işe başlamamız gerekiyor. Uygulamamıza ait **App.config** dosyasının içeriği aşağıdaki gibidir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <netTcpBinding>
```

```
          <binding name="TcpBindingConfig1" receiveTimeout="00:01:00"
sendTimeout="00:00:30" maxConnections="5">
            <reliableSession enabled="true" />
          </binding>
        </netTcpBinding>
      </bindings>
      <services>
        <service name="PreviousVersion.Aynstayn">
          <endpoint address="net.tcp://localhost:5001/Calculus"
binding="netTcpBinding"
            bindingConfiguration="TcpBindingConfig1" name="TcpEndPoint1"
            contract="PreviousVersion.ICalculus" />
          <endpoint address="net.tcp://localhost:5002/Calculus"
binding="netTcpBinding"
            bindingConfiguration="TcpBindingConfig1" name="TcpEndPoint2"
            contract="PreviousVersion.ICalculus" />
        </service>
      </services>
  </system.serviceModel>
</configuration>
```

Bu config dosyasında iki adet **Tcp** bazlı **EndPoint** bildirimi yapıldığı görülmektedir. **TcpEndPoint1** ve **TcpEndPoint2**. Bizim üzerinde duracağımız nokta **netTcpBinding** elementi ve içeriğidir. Dikkat edileceği üzere, **NetTcpBinding** bağlayıcı tipleri için bazı ayarlamalar yapılmıştır. Bu ayarlamalara göre **ReceiveTimeout, SendTimout, MaxConnections** ve**ReliableSession** değerleri belirlenmiştir. Söz konusu değişiklikler, **config** dosyası içerisinde tanımlı **NetTcpBinding** bağlayıcı tipini kullanan tüm **EndPoint**' ler için geçerli kılınabilir. Peki ama nasıl?

Dikkat edileceği üzere **netTcpBinding** altındaki binding elementinin **name** niteliği, hem **TcpEndPoint1** hemde **TcpEndPoint2** için **bindingConfiguration** elementlerinde kullanılmıştır. Böylece**WCF çalışma zamanı,** tanımlı olan **EndPoint**' lerin hangi binding ayarlarına bakacağını **bindingConfiguration** elementinin değerinden bulabilmektedir.

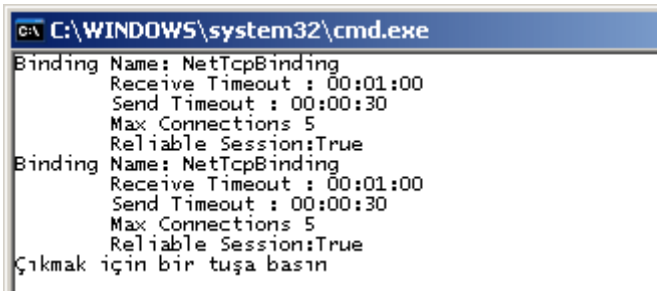
Not: *Esasında, bağlayıcı tipler için uygulamalar üzerinde configuration değerleri set edilmese dahi, WCF çalışma zamanı varsayılan bağlayıcı ayarlarını built-in olarak çekmektedir. Ancak built-in ayarlar istenirse uygulamalara ait config dosyalarında veya kod tarafında ezilebilir.*

Bu config içeriğini kullanan servis uygulaması kodlarını ise aşağıdaki gibi geliştirdiğimizi düşünelim.

```
using System;
using System.Linq;
using System.ServiceModel;

namespace PreviousVersion
{
   [ServiceContract]
   interface ICalculus
   {
      [OperationContract]
      double Sum(params double[] values);
   }

   class Aynstayn
      :ICalculus
   {
      public double Sum(params double[] values)
      {
         return values.Sum();
      }
   }

   // Client
   class Program
   {
      static void Main(string[] args)
      {
         // Service nesnesi örneklenir
         ServiceHost host = new ServiceHost(typeof(Aynstayn));

         // Servis açılır
         host.Open();

         // Tüm EndPoint' ler dolaşılır
         foreach (var endPoint in host.Description.Endpoints)
         {
            // Bu örnekte sadece NetTcpBinding kullanıldığın için Binding kontrolü
yapılmadan dönüştürme işlemi yapılmıştır.
            NetTcpBinding binding = (NetTcpBinding)endPoint.Binding;

            // Bağlayıcı tipi için ReceiveTimeout, SendTimeout, ReliableSession ve
MaxConnections değerleri ekrana yazdırılır
            Console.WriteLine(
               "Binding Name: {0}\n\tReceive Timeout : {1}\n\tSend Timeout : {2}\n\tMax
Connections {3}\n\tReliable Session:{4}"
```

```
                , binding.Name
                , binding.ReceiveTimeout.ToString()
                , binding.SendTimeout.ToString()
                ,binding.MaxConnections.ToString()
                ,binding.ReliableSession.Enabled.ToString());
            }

            Console.WriteLine("çıkmak için bir tuşa basın");
            Console.ReadLine();

            // Servis kapatılır
            host.Close();
        }
    }
}
```

Uygulamamızı çalıştırdığımızda aşağıdaki sonuçları elde ederiz.

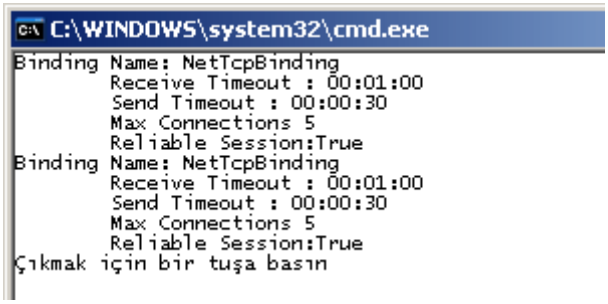


Peki **WCF 4.0** ile gelen yenilik nedir? Sakın gülmeyin ama son derece basit ve kolay **Kolaylaştırılmış konfigurasyon(Simplified Configuration)** yeniliklerine göre artık **endpoint** tanımlamalarında **bindingConfiguration** niteliğinin kullanılmasına gerek yoktur. Durumu daha net değerlendirebilmek için, yukarıdaki **config** içeriğini bu kez **.Net Framework 4.0** örneğine göre aşağıdaki gibi değiştirdiğimizi düşünelim.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <netTcpBinding>
        <binding receiveTimeout="00:01:00" sendTimeout="00:00:30"
maxConnections="5">
          <reliableSession enabled="true" />
        </binding>
      </netTcpBinding>
    </bindings>
    <services>
      <service name="PreviousVersion.Aynstayn">
```

```
            <endpoint address="net.tcp://localhost:5001/Calculus"
binding="netTcpBinding"
                name="TcpEndPoint1"
                contract="PreviousVersion.ICalculus" />
            <endpoint address="net.tcp://localhost:5002/Calculus"
binding="netTcpBinding"
                name="TcpEndPoint2"
                contract="PreviousVersion.ICalculus" />
        </service>
      </services>
   </system.serviceModel>
</configuration>
```

Dikkat edileceği
üzere **binding** elementinde **name** veya **endpoint** elementinde **bindingConfiguration** niteli kleri kullanılmamıştır. WCF çalışma zamanı **Uri** bilgilerinden yola çıkarak uygun**binding** konfigurasyonunu bulmakta ve uygulamaktadır. örneğimizi bu config ayarlarına göre **.Net Framework 4.0** üzerinden çalıştırırsak bir önceki örnek ile aynı sonuçlar aldığımız görebiliriz.



Evet. Son derece basit bir yenilik. Ancak konfigurasyon içeriğini **daha okunur** hale getirdiği ve **basitleştirdiği** ortada. Bu tekniği dilersek **machine.config** içerisindede kullanabiliriz. Yani **machine.config**içerisindeki bağlayıcı tipe özgü ayarlamalarda **name** niteliğini kullanmayabilir ve o makinedeki tüm uygulamaların da, **bindingConfiguration** niteliğini düşünmeden ilgili ayarları otomatik olarak almalarını sağlayabiliriz. **WCF 4.0** ile birlikte gelen temel yeniliklere devam ediyor olacağız. **Bu arada örneğimizi .Net Framework 4.0 Beta 1 ve Visual Studio 2010 Beta 1 üzerinde geliştirdiğimizi hatırlatalım. Dolayısıyla relase sürünmde bazı farklılıklar olabilir.** Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

DefaultBindingConfiguration.rar (47,83 kb)

WCF 4.0 Yenilikleri - Default Protocol Mapping [Beta 1] (2009-08-11T08:09:00)

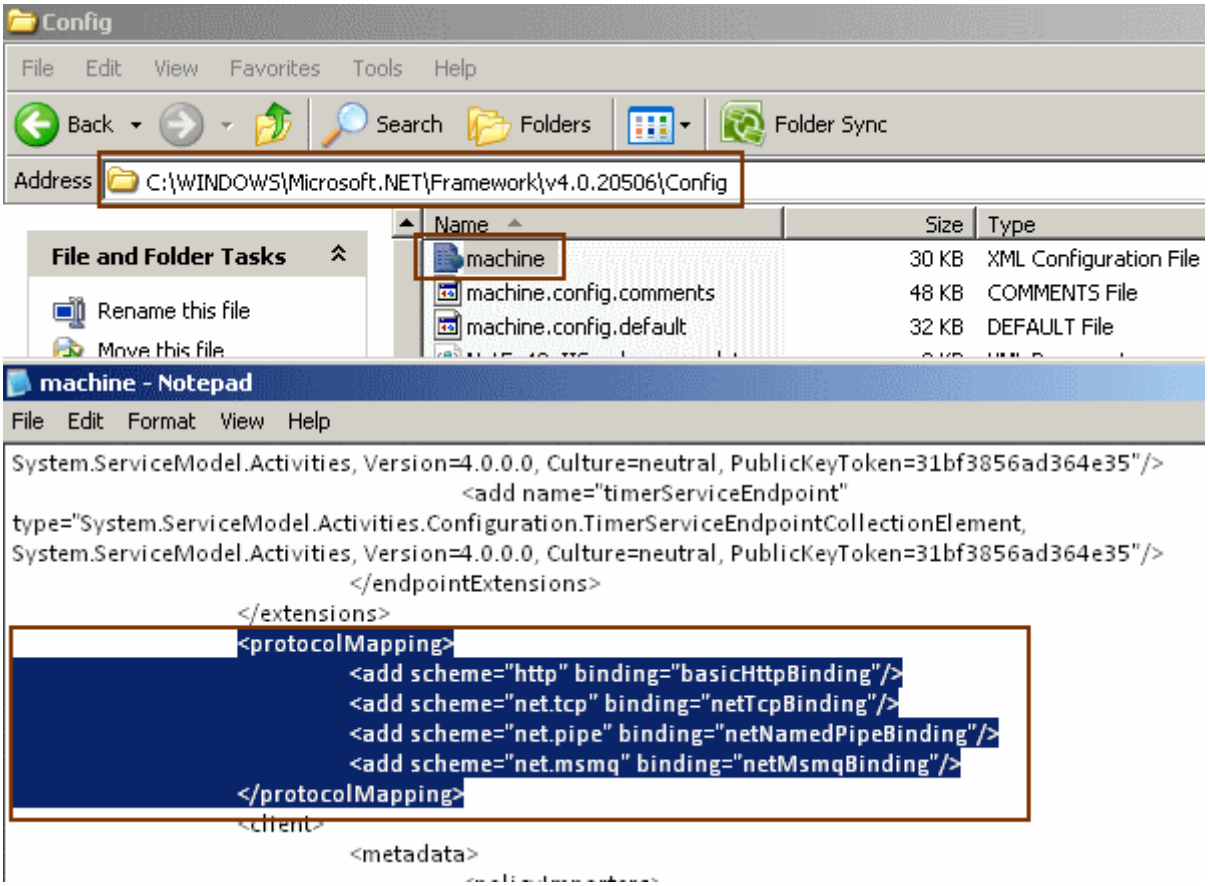
wcf,wcf 4.0,

Merhaba Arkadaşlar,

Bir önceki blog yazımızda, **WCF 4.0** ile birlikte gelebilecek özelliklerden birisi olan **Default EndPoints** kavramına değinmeye çalışmıştık. Durumu kısaca özetleyip, bu konu ile bağlantılı olan başka bir yenileğe bakarak devam edelim. **Default EndPoints** özelliği sayesinde, **WCF çalışma zamanına(Runtime)** açık bir şekilde **EndPoint** bildirimi yapma zorunluluğumuz ortadan kalkmaktaydı. Bir başka deyişle **config** dosyalarında veya kod bazında herhangibir **EndPoint** bildirimi yapmasak dahi, **WCF çalışma zamanı**, **ServiceHost** nesnesinde bildirilen **Uri**bilgilerine göre varsayılan iletişim noktalarını üretmekteydi. Ancak bir geliştirici gözü ile olaya yaklaştığımızda, **Uri** içerisinde yer alan **string** bilgiden nasıl yararlanılabildiği, yararlanıldıysada neye göre varsayılan **bağlayıcı tiplerin(Binding Type)** seçildiği bir soru işareti oluşturmaktadır.

.Net gibi gelişmiş **Framework** altyapılarında, sistemin geneline yönelik olarak kullanılan pek çok ayarlama bilindiği üzere basit konfigurasyon dosyalarında saklanmaktadır. **.Net Framework**tarafında **machine.config** dosyası ile makinedeki tüm **.Net** uygulamaları için geçerli olan konfigurasyon ayarları saklanır. Diğer taraftan uygulamalarımızda kullanabileceğimiz **config** dosyaları ile*(app.config, web.config gibi)*, **machine.config** üzerinden gelen bilgilerin bazıları ezilebilir. Hatta bildiğiniz üzere Web uygulamalarında hiyerarşik olarak yerleştirilebilen **web.config** dosyalarından yararlanılarak en alttan üste doğru ezme(Override) işlemleri gerçekleştirilebilmektedir*(Bir klasöre ayrı authorization uygulanması için web.config dosyasını nasıl kullandığımızı hatırlayalım)*. Peki buradan nasıl bir sonuca varmamız gerekiyor...

Tahmin edeceğiniz üzere **Default EndPoints** özelliğinin kullanılabilmesi için gerekli olan tüm tanımlamalar ve ayarlamalar aslında **.Net Framework 4.0'** a ait olan **maching.config** dosyası içerisinde tutulmaktadır.



Şekildende görüleceği üzere **scheme** niteliğinde çeşitli iletişim protokollerine göre bazı anahtar kelimelere yer verilmiştir. Buna karşılık olarak çalışma zamanında varsayılan olarak hangi bağlayıcı tipin kullanılacağı ise **binding** niteliğinde belirlenmektedir. Dolayısıyla **WCF çalışma zamanı**, **config** içerisinde veya kod tarafında bilinçli olarak tanımlanmış **EndPoint** verileri ile karşılaşmassa, **maching.config** içerisindeki **protocolMapping** eşleşme tablosunu baz alarak varsayılan **EndPoint** bildirimlerini, söz konusu servis için oluşturacaktır. Tam bu noktada akıllara gelen soru şu olacaktır; Acaba **maching.config** dosyasındaki bu içeriği değiştirebilir yada uygulamalarda ezebilir miyiz?

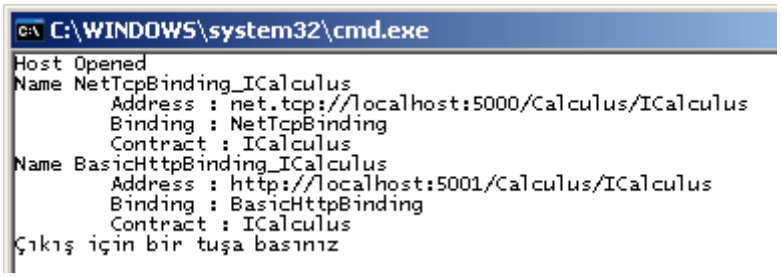
Bir geliştirici olarak bunun olmasını bekleriz. Gerçektende her iki durumda mümkündür. **Machine.config** içerisinde yer alan **protocolMapping** elementine ait alt boğumları(Child Nodes) değiştirebilir yenilerini ekleyebiliriz. örneğin tüm **http** tabanlı adreslerin aslında **WebHttpBinding** tarafından ele alınmasını sağlayabiliriz*(O makinede sadece WCF RESTful servislerin barındırıldığını düşünün).* Tabiki **machine.config** içerisinde yapılan tüm ayarlamalar, bu dosyayı kullanan makinedeki tüm **WCF** servis uygulamaları için geçerli olacaktır. Bir diğer seçenek olarakta kendi uygulamalarımız içerisinde, machinbe.config üzerinde tanımlı makine bazlı protokol eşleştirmelerini ezebiliriz. Şimdi bu durumu anlamak için aşağıdaki**Console** uygulamasını geliştirdiğimizi düşünelim.

Not : *Uygulamamızı* ***.Net Framework 4.0 Beta 1*** *yüklü bir sistemde* ***Visual Studio 2010 Beta 1*** *ile geliştirdiğimizi ve son sürümlerde farklılıkar olabileceğini hatırlatalım.*

```
using System.ServiceModel;
using System;

namespace DefaultBindings
{
   // Servis sözleşmesi
   [ServiceContract]
   interface ICalculus
   {
      [OperationContract]
      double Sum(double x, double y);
   }

   // Servis
   class Aynstayn
      :ICalculus
   {
      public double Sum(double x, double y)
      {
         return x + y;
      }
   }

   // Client
   class Program
   {
      static void Main(string[] args)
      {
         // Http ve Tcp bazlı iki adres bildirilir.
         ServiceHost host = new ServiceHost(typeof(Aynstayn),
            new Uri("net.tcp://localhost:5000/Calculus"),
            new Uri("http://localhost:5001/Calculus")
            );
         // Servis açılır
         host.Open();

         Console.WriteLine("Host {0}", host.State);

         // Varsayılan olarak eklenen EndPoint tipleri listelenir
         foreach (var endPoint in host.Description.Endpoints)
         {
            Console.WriteLine("Name {0}\n\tAddress : {1}\n\tBinding : {2}\n\tContract :
{3}",endPoint.Name,endPoint.Address,endPoint.Binding.Name,endPoint.Contract.Name);
         }
```

```
            Console.WriteLine("çıkış için bir tuşa basınız.");
            Console.ReadLine();

            // Servis kapatılır
            host.Close();
        }
    }
}
```

Bu haliyle uygulamamızı çalıştırdığımızda bir önceki blog yazımızda olduğu gibi varsayılan **EndPoint'** lerin eklendiği gözlemlenecektir.

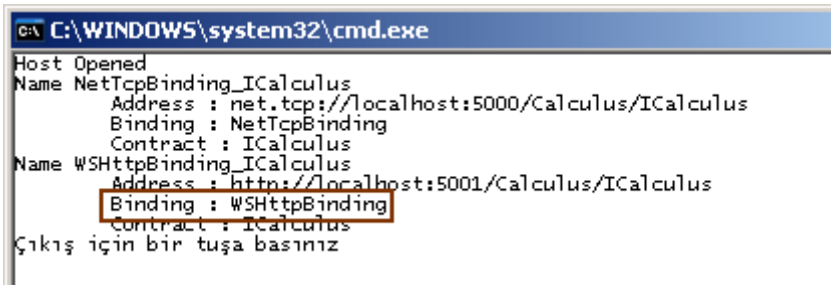


Şimdi uygulamamıza bir **app.config** dosyası eklediğimizi ve içeriğini aşağıdaki gibi geliştirdiğimizi varsayalım.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <protocolMapping>
      <!-- Eğer http protokolü ile karşılaşılırsa wsHttpBinding bağlayıcı tipini varsayılan olarak kullan. Machine.config içerisindeki http tanımlaması ezilmiştir.-->
      <!--<clear/>--> <!-- Clear kullanıdığımız takdirde, machine.config içerisindeki tüm protocolMapping eşleştirmeleri geçersiz kılınır. Dolayısıyla bu örnekte yer alan net.tcp bazlı Uri için, bu config ayarlarına göre otomatik bir EndPoint noktası üretilmez -->
      <add scheme="http" binding="wsHttpBinding"/>
    </protocolMapping>
  </system.serviceModel>
</configuration>
```

Bu tanımlamaya göre, **Uri** bilgisinde **http** protokolü ile karşılaşıldığında varsayılan olarak **WsHttpBinding** bağlayıcı tipinin kullanılması söylenmektedir. Aynı örneği, bu config ayarına göre çalıştırdığımızda aşağıdaki sonucu alırız.

Not : binding *niteliğinin(attribute) değeri* ***case-sensitive'*** *dir. Yani* ***wsHttpBinding*** *yerine örneğin* ***WsHttpBinding*** *yazıldığı takdirde çalışma zamanında****System.ServiceModel.CommunicationObjectFaultedException*** *tipinden bir* ***istisna(Exception)*** *alınır.*



Görüldüğü gibi, **http** protokolü için varsayılan olarak **WsHttpBinding** bağlayıcı tipini kullanan bir **EndPoint** üretilmiştir. Tabiki bu ayarlamalar sırasında dikkat edilmesi gereken bazı noktalarda vardır. Söz gelimi **WsHttpBinding** ile **BasicHttpBinding** arasındaki farklar göz önüne alındığında, servis tarafınında buna göre geliştiriliyor olması gerekir. Bunu en güzel açıklayan durumlardan birisi **RESTful** için **WebHttpBinding** kullanılması olarak düşünülebilir. Nitekim varsayılan bağlayıcı tipi olarak **WebHttpBinding** seçildiğinde, servis operasyonlarında **WebGet** veya **WebInvoke** gibi niteliklerin kullanılması önemlidir. Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

DefaultBindings.rar (24,20 kb)

WCF 4.0 Yenilikleri - Default EndPoints [Beta 1] (2009-08-10T01:51:00)

wcf,wcf 4.0,

Merhaba Arkadaşlar,

çok eskinden **.Net Remoting** ile ilişkili uygulamalarda sırasında, **Visual Studio.Net 2003 intelli-sense** özelliğ **sense**özelliği belirli bir elemente kadar destek veriyor a zorunda kaldığımı hatırlıyorum.

Bu durum, **.Net Remoting** tabanlı **dağıtık uygulamaların(Distributed Applications) TCP** bazlı hızlı bir iletişim sağlama avantajını kimi zaman göz ardı ettirebilen bir zorluktur. Nitekim ezbere kod yazmak, hiç bir zaman iyi bir şey değildir. özellikle işlerin arap saçına dönmesine neden olabilir.

Gel gelelim bir başka dağıtık uygulama geliştirme modeli olan **Xml Web Servislerinde, .Net Remoting** için karşılaştığımız ayarlama zorluklarını göremeyiz. öyleki, **WebService**ve **WebMethod niteliklerinin(Attributes)** kullanılması yeterli

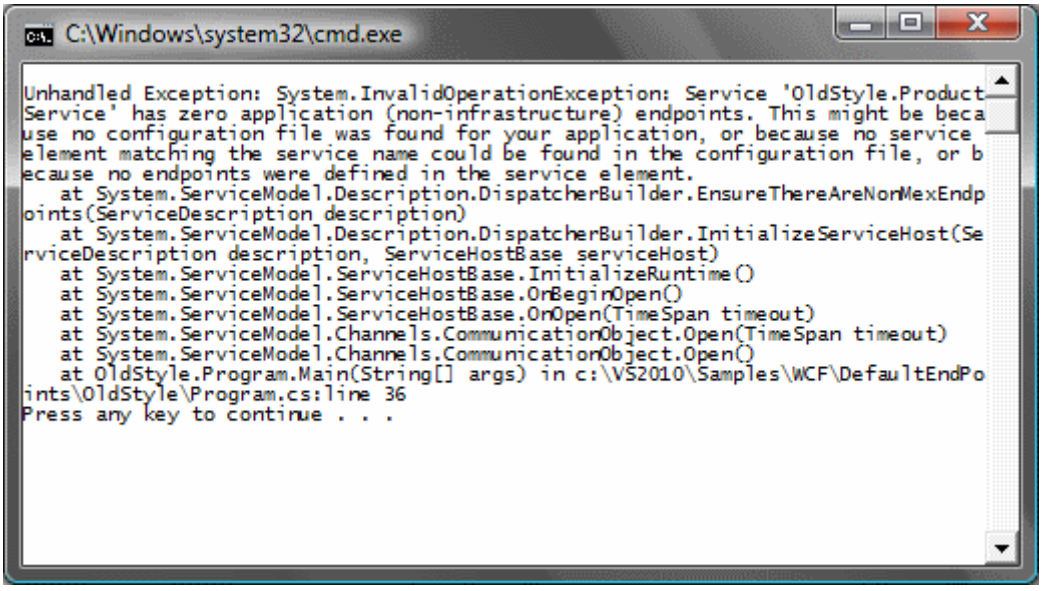
olmaktadır. çünkü çalışma zamanı, bu niteliklere göre otomatik olarak servis tarafını ayağa kaldırır.

Ancak **WCF(Windows Communication Foundation)** tarafına geçtiğimizde konfigurasyon tarafında göz önüne alınması gereken çok fazla şey olduğunu gördük. **WCF'** in pek çok dağıtık uygulama geliştirme modelini tek bir çatı altında birleştirmesinin oluşturduğu zorluklardan biriside, çok fazla ince ayarı içermesi olarak düşünülebilir. Hal böyle olunca WCF takımı boş durmamış ve **4.0** versiyonunda daha kolay konfigrasyon yapılabilmesini sağlamak adına bir takım geliştirmelerde bulunmuştur.*(örneklerimizi Visual Studio 2010 Beta 1 ve .Net Framework 4.0 Beta 1 üzerinde geliştirdiğimizden, Release sürümde bir takım değişiklikler veya farklılıklar olabileceğini hatırlatmak isterim)* Bu geliştirmelerden birisi**DefaultEndPoints** kavramıdır. Bu özelliği, varsayılan olarak **EndPoint** adreslerinin biz söylemeden çalışma zamanına entegre edilmesinin sağlanması olarak düşünebiliriz. Aslında olaya **.Net 3.0/3.5** açısından bakmamızda yarar vardır. Bu nedenle **.Net 3.5** tabanlı geliştirilmiş aşağıdaki Console uygulamasını göz önüne alalım.

```
using System;
using System.ServiceModel;

namespace OldStyle
{
   [ServiceContract]
   interface IProductService
   {
      [OperationContract]
      double GetExpensiveProduct(int categoryId);
   }

   class ProductService
      : IProductService
   {
      public double GetExpensiveProduct(int categoryId)
      {
         Random rnd = new Random();
         double price = rnd.NextDouble() * 100;
         return price;
      }
   }

   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(
```

```
            typeof(ProductService),
            new Uri("net.tcp://localhost:1500/adventure/"),
            new Uri("http://localhost:1400/adventure/")
            );

        // Herhangibir configurasyon tanımlaması ve özellikle EndPoint bildirilmeleri yapılmadığı için Open metodundan sonra çalışma zamanı istisnası alınır(Runtime Exception)
        // Dolayısıyla ya kod tarafında yada config dosyasında EndPoint bildirimleri yapılmalıdır.
        host.Open();
      }
   }
}
```

Uyulamada basit bir servis kullanılmaktadır. **ServiceHost** nesne örneğinin oluşturulması sırasında, iki farklı **Uri** bilgisi verildiğine dikkat etmeliyiz. Bunlardan birisi **Tcp** bazlı diğeri ise **Http**bazlı iletişimleri desteklemektedir. ServiceHost nesnesi, örneklenmesinin ardından istemcilerden gelecek talepleri dinlemek üzere Open metodu ile açılmaktadır. Ancka örneği çalıştırdığımızda aşağıdaki ekran görüntüsü ile karşılaşırız.



Hemen şunu belirteyim; uygulamamızda herhangibir konfigurasyon dosyası kullanmadık. Hal böyle olunca **WCF çalışma zamanı** belirtilen **Uri'** ler için
hangi **EndPoint** tanımlamalarını kullanması gerektiğini bulamadı ve bir istisna fırlatarak uygulamanın sonlanmasına neden oldu. EndPoint kavramı WCF tarafının olmazsa olmaz bütünlerinden birisidir. Basit olarak servisin nerede durduğu, hangi hizmeti ve nasıl sunacağı ile ilişkili temel bilgileri içermektedir.
Yani **AddressBindingContract** kavramından**(WCF' in ABC' si)** bahsediyoruz. Bir servis birden fazla EndPoint içerebilir. Her ne olursa olsun, EndPoint' lerin ya config dosyası

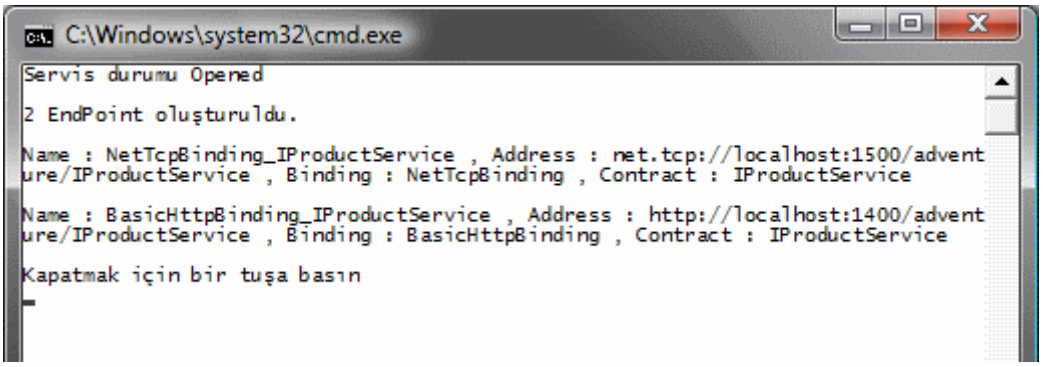
içerisinde yada kod tarafında tanımlanıp eklenmesi gerekir. Peki aynı örneği .Net 4.0 tabanlı olarak Visual Studio 2010 üzerinde geliştirseydik.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Description;

namespace DefaultEndPoints
{
   [ServiceContract]
   interface IProductService
   {
      [OperationContract]
      double GetExpensiveProduct(int categoryId);
   }

   class ProductService
      : IProductService
   {
      public double GetExpensiveProduct(int categoryId)
      {
         Random rnd = new Random();
         double price=rnd.NextDouble()*100;
         return price;
      }
   }

   class Program
   {
      static void Main(string[] args)
      {
         // ServiceHost nesnesi örneklenir
         // İki farklı adres bilgisi verilmiştir. Bunlardan birisi tcp bazlı diğeri ise http bazlıdır
         ServiceHost host = new ServiceHost(
            typeof(ProductService),
            new Uri("net.tcp://localhost:1500/adventure/"),
            new Uri("http://localhost:1400/adventure/")
            );

         // Host açılır.
         host.Open();
         Console.WriteLine("Servis durumu {0}\n",host.State.ToString());

         // Ne config içerisinde nede kod tarafında açık bir şekilde EndPoint bildirilimi
```

yapılmamıştır. Buna rağmen çalışma zamanı ServiceHost nesnesinin yapıcı metodundaki Uri bilgilerinden yararlanarak varsayılan EndPoint bilgilerini oluşturmuştur.

```
            // Servis için oluşturulan EndPoint' lerin listesi alınır
            ServiceEndpointCollection endPoints = host.Description.Endpoints;

            Console.WriteLine("{0} EndPoint oluşturuldu.\n",endPoints.Count.ToString());

            // EndPoint nesnelerinin her biri dolaşılır
            foreach (var endPoint in endPoints)
            {
                // EndPoint adı, adres(Address), bağlayıcı tip(Binding Type) adı,
sözleşme(Contract) adı yazdılır.
                Console.WriteLine("Name : {0} , Address : {1} , Binding : {2} , Contract :
{3}\n", endPoint.Name,endPoint.Address.Uri,endPoint.Binding.Name,endPoint.Contr
act.Name);
            }

            Console.WriteLine("Kapatmak için bir tuşa basın");
            Console.ReadLine();

            host.Close();
        }
    }
}
```

Bu kez 4.0 ile birlikte gelen varsayılan EndPoint kavramına güvenerek, yüklenen EndPoint' lere ait bilgileride ekrana yazdırıyoruz. Ancak yine bilinçli olarak EndPoint oluşturmadığımızı veya config dosyası kullanmadığımızı belirtelim. Uygulamayı çalıştırdığımızda aşağıdaki sonuçlar ile karşılaşırız.

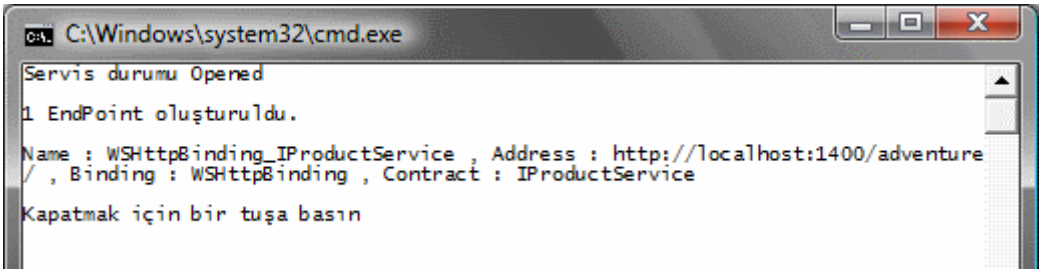


Hımmm... Harika! **Uri** bilgisindeki protokol tanımalamarına bakılarak, çalışma zamanı bizim için iki farklı **EndPoint** bilgisini otomatik olarak oluşturmuştur. **Tcp** bazlı adresleme için varsayılan olarak **NetTcpBinding**, **Http** bazlı adresleme içinse varsayılan olarak **BasicHttpBinding** bağlayıcı tipleri oluşturulmuştur. Diğer yandan **Address** özelliklerinde, **Uri** bilgisi sonuna**sözleşme tipi(Contract Type)** adının

eklendiğine dikkat edilmelidir. Buradan şu sonuca varabiliriz. Servis tarafında kaç sözleşme ve adres sunuluyorsa bunların çarpanı kadar EndPoint otomatik olarak oluşturulacaktır. Elbetteki biz **EndPoint** bildirimlerini bilinçli olarak yapmassak. Peki ya servis tarafında **EndPoint** bilgisini eklemişsek? örneğin aşağıdaki kod parçasında olduğu gibi **ServiceHost** nesnesinin örneklenmesinden sonra **AddServiceEndpoint** metodunu kullanırsak...

host.AddServiceEndpoint(typeof(IProductService), new WSHttpBinding(), "");

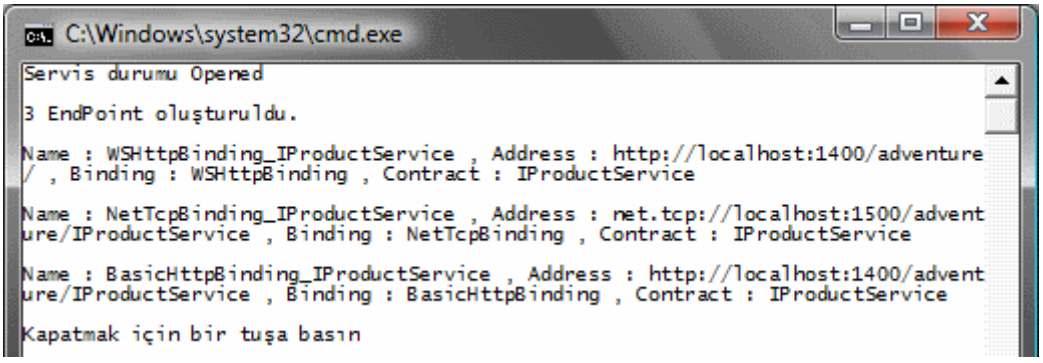
Bu durumda aynı örneğin çalışma zamanı çıktısı aşağıdaki gibi olacaktır.



Görüldüğü üzere çalışma zamanı sadece bizim eklediğimiz **EndPoint** bilgisini kullanmaktadır. Tam bu noktada **WCF 4.0** ile birlikte gelen **AddDefaultEndpoints** metodunu değerlendirmeye çalışalım. Normal şartlarda servis tarafına **EndPoint** bilgilerini eklemessek, WCF çalışma zamanı, yeni gelen **AddDefaultEndpoints** metodunu kullanmakta ve **Uri** bilgilerine göre varsayılan atamaları yapmaktadır. Peki yukarıdaki gibi **AddServiceEndpoint** metodundan sonra birde **AddDefaultEndpoints** metodunu yukarıdaki örneğe göre aşağıdaki gibi kullanırsak...

host.AddDefaultEndpoints();

Bu durumda çalışma zamanı çıktısı aşağıdaki gibi olacaktır.



Görüldüğü üzere hem bizim bilinçli olarak eklediğimiz hemde AddDefaultEndpoints metodu nedeniyle eklenen EndPoint bilgileri yer almaktadır. Yani WCF çalışma ortamı 3 EndPoint noktasını kullanıma açmaktadır.

Şüphesizki bu yenilik, varsayılan olarak standart **EndPoint** bilgilerini kullandığımız vakalarda son derece işe yarardır. Geliştiricinin işi kolaylaştırılmaktadır. Ama elbetteki pek

çok gerçek hayat senaryosunda; örneğin **WSDL** çıktılarının yasaklandığı, **iletişim seviyesinde güvenliğin(Transport Layer Security)** sağlanması gerektiği veya **çift taraflı haberleşmenin(Duplex Communication)** olduğu durumlarda varsayılan olarak atanan **EndPoint'** ler dışındakilerin kullanılması gerekmektedir.

Bu kısa yazımızda **WCF 4.0** tarafında, **basitleştirilmiş konfigurasyon(Simplified Configuration)** ayarlamalarının özelliklerinden birisi olan **Default EndPoints** kavramına değinmeye çalıştık. İlerleyen yazılarımızda diğer WCF 4.0 yeniliklerinede değinmeye çalışıyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

DefaultEndPoints.rar (40,08 kb)

VSTS 2008 için Custom Check-In Policy Geliştirmek (2009-08-07T21:56:00)

vsts 2008,

Merhaba Arkadaşlar,

Bir süredir **Team Foundation Server** üzerinde ve doğal olarak **Visual Studio Team System 200** uzun yıllar **Visual Source Safe** ile vakit geçirmenin sonucunda, **TFS** ile birlikte gelen pek çok ni Herşeyden önce TFS'in, **MSF(Microsoft Solution Framework)** ve **CMMI(Capability Maturit** geliştirme süreç modellerinin uygulanabildiği profesyonel bir çevre sağladığını bilmemiz gerekiy

İşin içerisine **Reporting Services'** ile etkili raporlama, **SharePoint** ile tutarlı, ölçeklenebilir döküman yönetimi gibi pek çok yararlı üründe giriyor. Tabiki bu tip sistemlerin uygulanması her zaman kolay değildir. Her şeyden önce bir öğrenme süreci için geliştirme ekibinin ciddi zaman ayrılması şarttır. öyleki, model içerisinde **çevik(Agile)** süreçlerin uygulanabilirliği söz konusudur ki bunlarda başlı başlına birer konsepttir.

Biz bu yazımızda **Visual Studio Team System 2008** ile çalışırken, özel **Check-In ilkelerinin(Custom Check-In Policy)** nasıl geliştirilebileceğini basit bir örnek üzerinden ele almaya çalışacağız. Genellikle yazılım projelerinde görev alan geliştiricilerin, en alt kademeden en üst kademeye doğru çıktıkça**(Junior -> Architect)** çok daha az sayıda **Check-In** yaptıkları görülebilir Bu aslında iyi yazılım geliştirme süreçlerinde bir kural olarak değerlendirilmesi gereken durumlardandır. Söz gelimi bir projenin mesai sonunda eğer derlenemiyorsa Check-In' lenmemesi*(ki bunu VSTS üzerinden vereceğiniz hazır bir Policy ile kolayca garantileyebilirsiniz)*, gün içinde belkide 1 en fazla 2 Check-In yapılmasına izin verilmesi, özellikle versiyon takibi açısından da son derece önemlidir. Peki **VSTS** tarafından bizlere sunulan hazır **Check-In** ilkeleri**(Builds, Code Analysis,**

Testing Policy, Work Items) dışında kendi özel politikalarımızı nasıl geliştirebilir ve projeye uygulayabiliriz?

Aslında burada amaç, **VSTS'** in dış ortama sunulan bazı arayüzlerini kullanarak kendisine yeni davranışlar ekleyebilmektir. Yani bir **Plug-In** modeli ile karşı karşıya olduğumuzu düşünebiliriz. Dilerseniz bu fikirden yola çıkalım. İşe ilk olarak basit bir **Class Library** geliştirerek başlamamız gerekiyor. Söz konusu sınıf kütüphanesine,**Microsoft.TeamFoundation.Version.Client** isimli referansın eklenmesi gerekmektedir. Bu referans varsayılan kuruluma göre **C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE\PrivateAssemblies** klasörü altında bulunmaktadır. Refernasların eklenmesinin arından, **PolicyBase** isimli tipten türetilen bir sınıfın geliştirilmesi ve bunun içerisinde gerekli üyelerin **ezilmesi(Override)** gerekmektedir. örneğimizde, **TFS** üzerindeki bir projeye eklenen dosyaların oluşturulduktan sonra, ne kadar süre sonra **Check-In'** lenebileceklerine dair bir ilke tanımlamaya çalışıyor olacağız. Eğer oluşturulma zamanı, gün değerinden küçük ise **Check-In** işlemi yapılırken **uyarı(Warning)** mesajı verilmesini sağlayacağız. Bununla birlikte istenirse **Check-In** için gerekli gün sayısını belirleyebileceğimiz basit bir iletişim penceremizde olacak. Böylece istenirse gün bazındaki Timeout süresi **Edit** seçeneği ile değiştirilebilir olacak. Bu tabiki tamamen sembolik bir örnek. *(Bazı kaynaklarda Check-In lenecek dosyalar içerisinde yasaklı kelimelerin bulunmasının önüne geçilmesi veya kod standartlarından çok özel olan bazılarına uyulmadığı durumlarının ele alınması için geliştirilen ilkeler yer almaktadır. Sizde projeninizin ihtiyacı olan ilkeleri modelleyebilirsiniz.)*

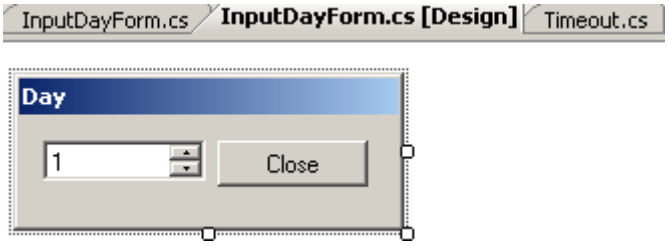
örneğimizde yer alan Timeout isimli sınıfa ait kod yapısı aşağıda görüldüğü gibidir.

```
using System;
using System.Collections.Generic;
using Microsoft.TeamFoundation.VersionControl.Client;
using System.Windows.Forms;

namespace TimeoutPolicy
{
   [Serializable] // Serileştirilebilir olma şartı
   public class Timeout
      : PolicyBase // Custom policy yazmak için PolicyBase tipinden türetme yapmak
gerekmektedir
   {
      // Minimum Check-In süresi
      public int TimeoutDay { get; set; }

      public Timeout()
      {
         TimeoutDay=1;
      }
```

```
    public override string Description
    {
        get { return "Bir Check-In işlemi yapılması için, dosyanın ilk oluşturulmasından
sonra geçmesi gereken minimum gün süresi kontrolünü yapar"; }
    }

    // Policy içerisinde yer alan gün değeri istenirse editlenebilir
    public override bool Edit(IPolicyEditArgs policyEditArgs)
    {
        // Gün değerini almak için InputDayForm açılır
        // InputDayForm sınıfının yapıcı metoduna o anki Timeout sınıfının referansı
gönderilir. Böylece form içerisinde this referansına ait gün değeri set edilebilir.
        InputDayForm inputForm = new InputDayForm(this);
        inputForm.ShowDialog();
        return true;
    }

    // Policy kontrolünün yapıldığı yerdir. Geriye var ise hata bildirimlerini döndürür
    public override PolicyFailure[] Evaluate()
    {
        List<PolicyFailure> failures = new List<PolicyFailure>();

        // Şu an beklemede olan tüm PendingChange referansları dolaşılır
        foreach (var item in PendingCheckin.PendingChanges.AllPendingChanges)
        {
            // Eğer değişiklik örneğin bir dosya ile ilgiliyse
            if(item.ItemType== ItemType.File)
            {
                // Dosyanın oluşturulma tarihi ile güncel tarih arasındaki farka bakılır
                TimeSpan distance = DateTime.Now - item.CreationDate;
                // Fark var ise
                if (distance.TotalDays < TimeoutDay)
                {
                    failures.Add(
                        new PolicyFailure(String.Format("{0} tarihli {1} için Check-In süresi
dolmamış. Lütfen {2} gün
bekleyiniz",item.CreationDate,item.FileName,TimeoutDay), this)
                    ); // Bir PolicyFailure nesnesi örneklenir ve mesaj bilgisi yazdırırılır.
                }
            }
        }

        if (failures.Count > 0)
            return failures.ToArray();
        else
```

```
            return null;
        }

        public override string Type
        {
            get { return "Chech-In Timeout(Day)"; }
        }

        public override string TypeDescription
        {
            get { return "Gün bazlı Check-In süresi"; }
        }
    }
}
```

Timeout sınıfı **serileştirilebilir(Serializable)** olarak tanımlanmalıdır.
Ayrıca **PolicyBase** tipinden türemelidir. Türeme sonucu **ezilmesi(Override)** gereken bazı üyeler olduğu görülmektedir. Bunlardan belkide en önemlisi **Evaluate** isimli metoddur. Bu metod içerisinde, senaryoda yer alan ilkenin uygulanması ve ilkenin aşılması halinde geriye **PolicyFailure** tipinden uyarı mesajlarının bir dizi şeklinde döndürülmesi sağlanmaktadır. Bizim örneğimizde dikkat edileceği üzere beklemede olan tüm **Check-In'** ler içerisinde tipi **File** olanlar ele alınmakta ve**CreationDate** özelliklerinin değerlerine bakılmaktadır. Eğer bu değer **Timeout** sınıfının kendi özelliği olan **TimeoutDay** değerinden küçük ise **PolicyFailure** oluşturulmaktadır. Ezilen diğer bir metodda **Edit** fonskiyonudur. Bu fonksiyon ile ilkenin düzenlenebilip düzenlemeyeceğine karar verilebilir. Biz örneğimizde, **TimeoutDay** özelliğinin değerinin değiştirilebileceği basit bir **Windows Form** kullanıyoruz. InputDayForm isimli bu Windows Form' un görüntüsü ve kod içeriği ise şu şekildedir;

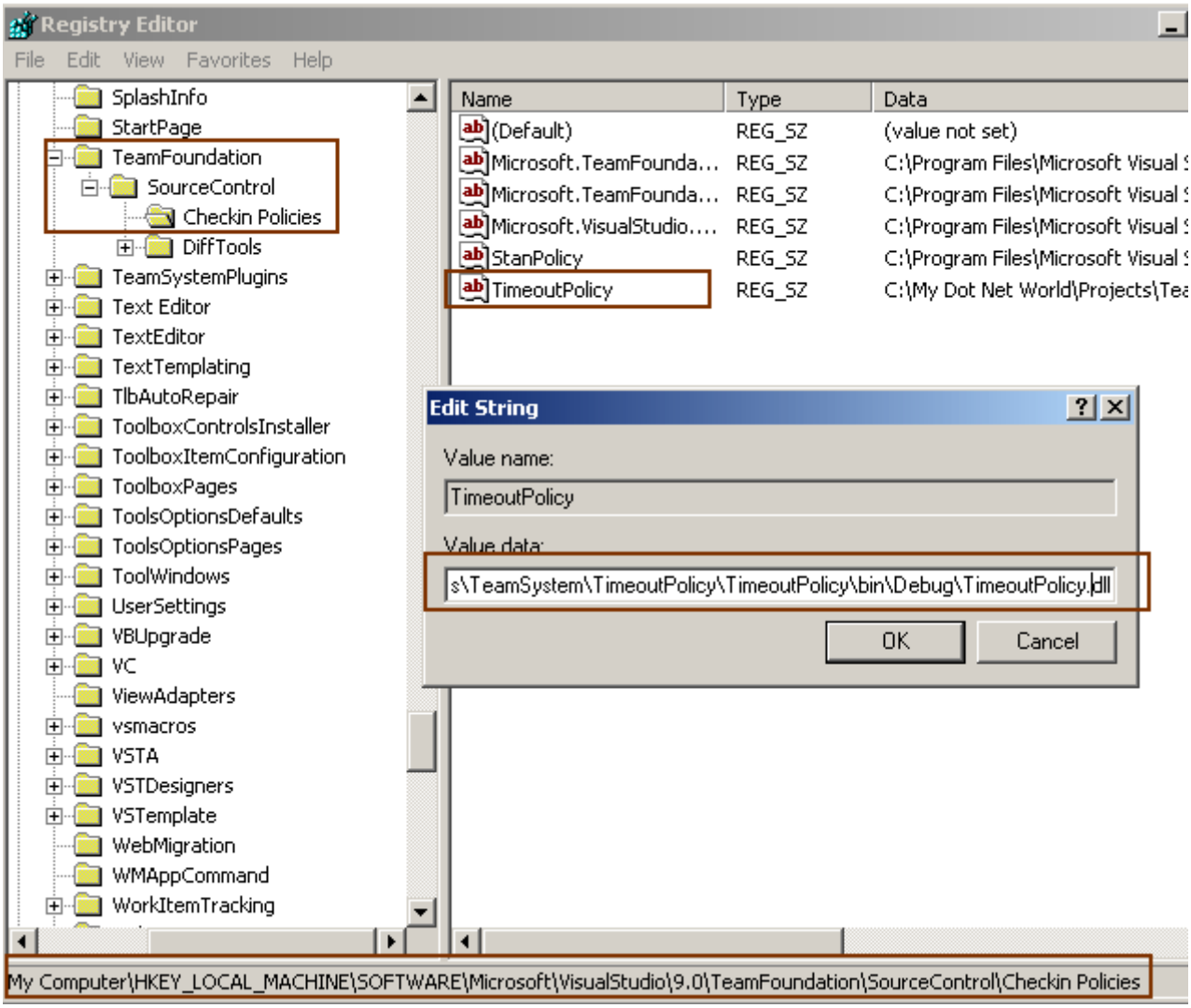



```
using System;
using System.Windows.Forms;

namespace TimeoutPolicy
{
    public partial class InputDayForm : Form
    {
        private Timeout _timeOut;
```

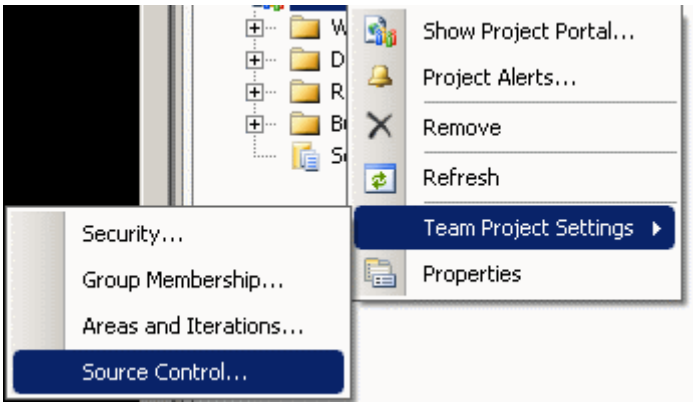
```
        public int Day
        {
            get { return Convert.ToInt16(nupDay.Value); }
        }

        public InputDayForm(Timeout timeOut)
        {
            InitializeComponent();
            _timeOut = timeOut;
            nupDay.Value = _timeOut.TimeoutDay;
            btnClose.DialogResult = DialogResult.OK;
        }

        private void btnClose_Click(object sender, EventArgs e)
        {
            _timeOut.TimeoutDay = (Int16)nupDay.Value;
        }
    }
}
```

Formumuz oluşturulurken, Timeout referansını almaktadır. Böylece Form üzerindeki NumericUpDown kontrolünde yapılan değişimlere göre, Timeout içerisindeki TimeoutDay değeri dinamik olarak değiştirilebilir.

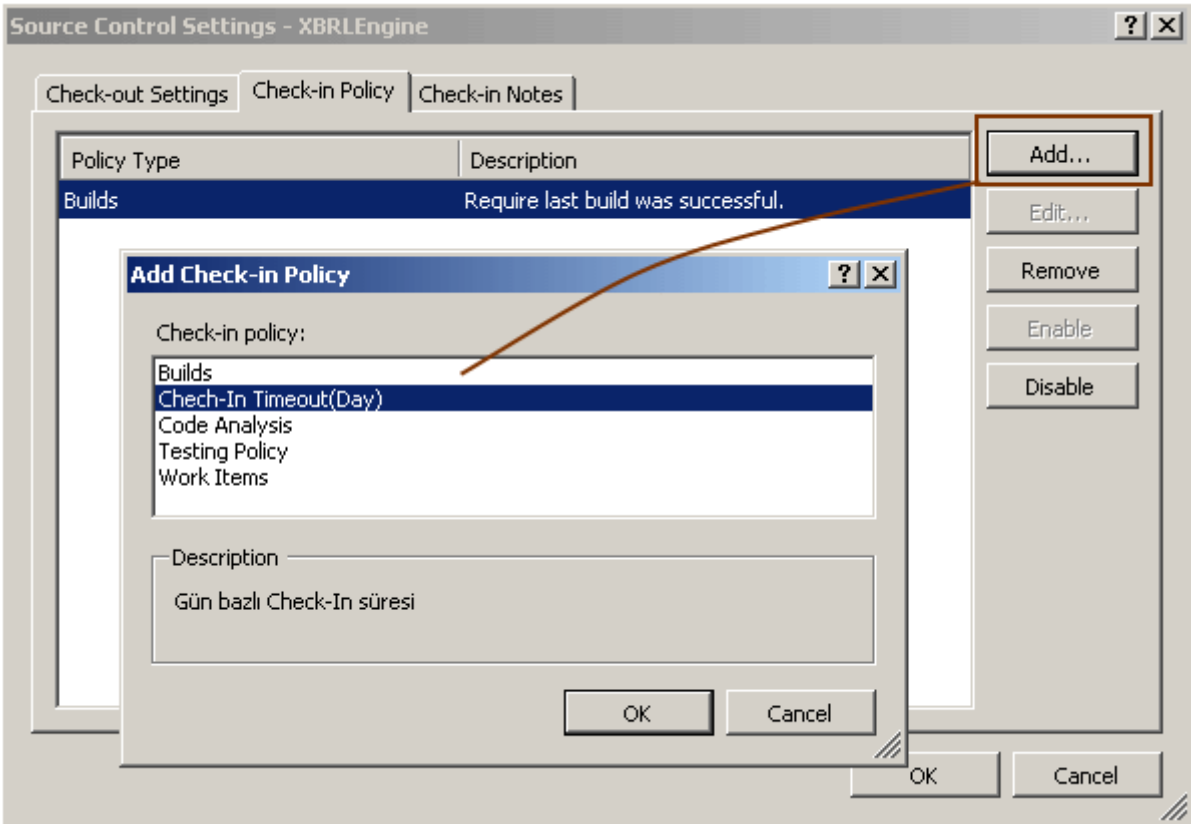
İşlemlerimiz bununla bitmiyor tabiki. Geliştirdiğimiz sınıf kütüphanesinin, **VSTS 2008** arabiriminde **Team Exlporer** üzerinden kullanılabilmesi için **Registry'** de ufak bir ekleme yapmamız gerekmektedir 😕 Aynen aşağıdaki ekran görüntüsünde olduğu gibi.



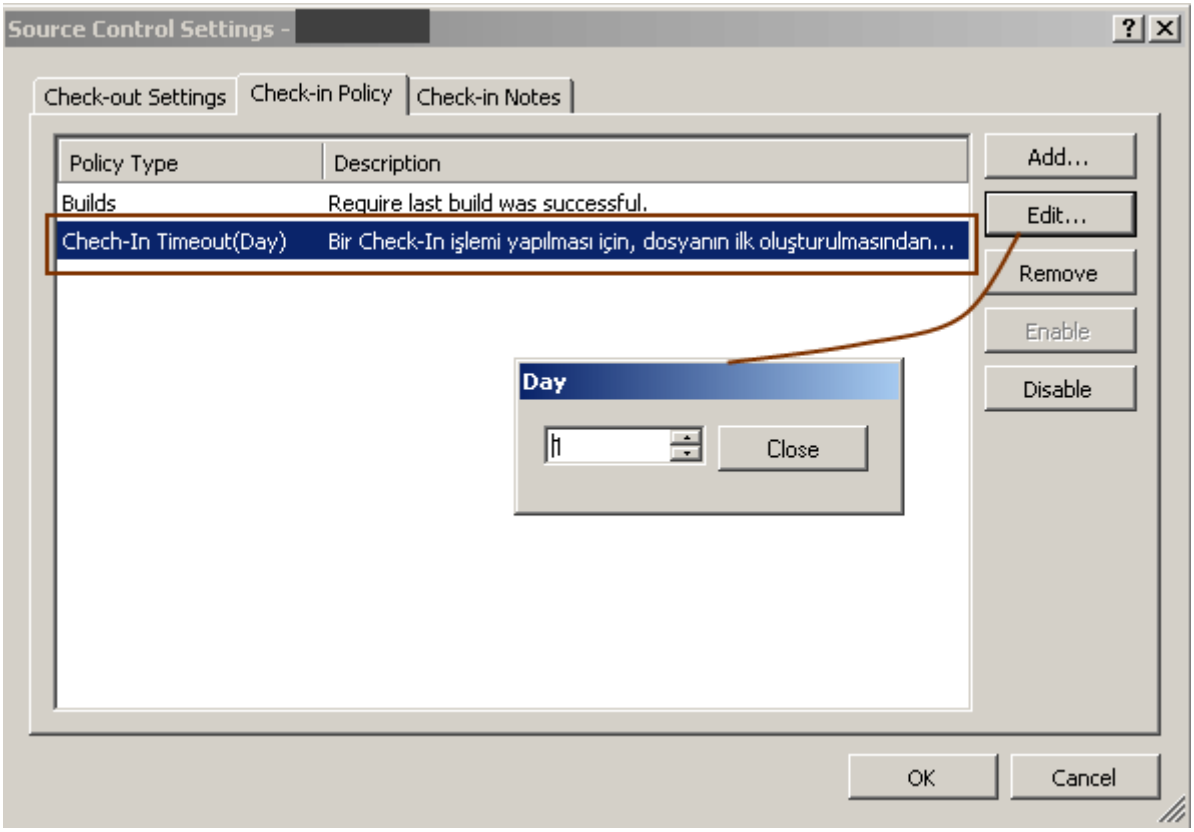
Bu ekleme işlemi sırasında dikkat edilmesi gereken noktalardan birisi **Value** adının **assembly** adı ile aynı olmasıdır. **Value Data** kısmında ise dikat edileceği üzere geliştirdiğimiz sınıf kütüphanesinin fiziki adresi bulunmaktadır. Peki şimdi elimize ne geçti. 😎 **VSTS 2008** üzerinde **Source Settings** kısmını açıp(aşağıdaki ekran görüntüsünde olduğu gibi),



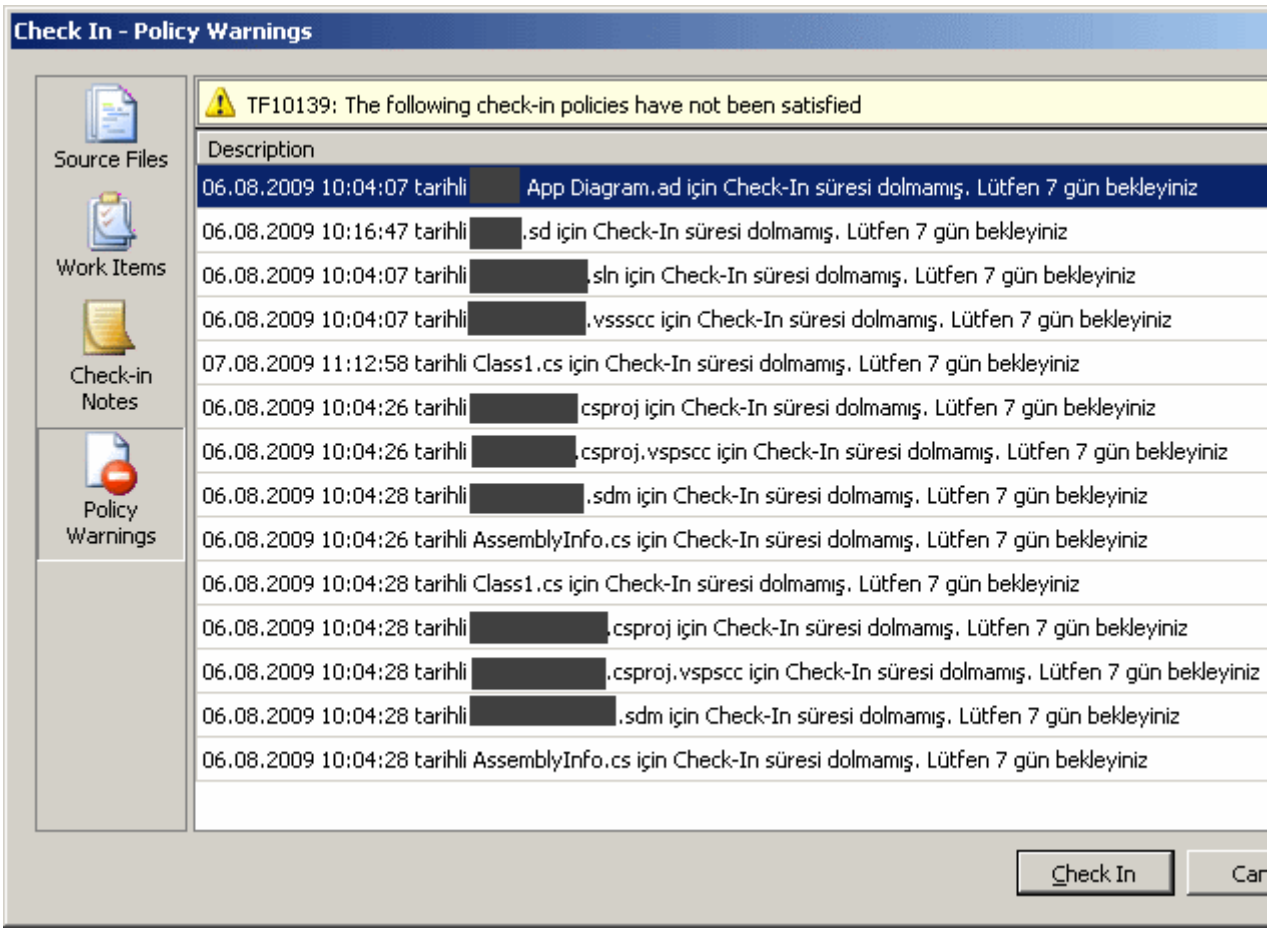
yeni bir **Check-In Policy** eklemek istediğimizde aşağıdaki durum ile karşılaşırız.



Görüldüğü gibi az önce geliştirdiğimiz **Custom Check-In Policy** tipi burada yer almaktadır. **Policy'** yi ekledikten sonra dilersek **Edit** düğmesinide kullanabilir ve **TimeoutDay** özelliğinin değerini değiştirebiliriz.



Süper. Artık durumu test edebiliriz. Ben **TFS** üzerindeki örnek bir projede bu ayarları yaptıktan ve yeni eklediğim bazı dosyaları **Check-In'** lemek istedikten sonra aşağıdaki ekran görüntüsünde yer alan uyarılar ile karşılaştım.



TimeoutDay özelliğini 7 gün olarak set ettikten sonra başıma neler gelmiş neler ? Elbette buradaki ilkeleri ezip geçebiliyoruz bildiğiniz üzere ama izleri kalıyor... Bu örnek ilkeyi uygulanızı önermem. Ancak sanıyorumki artık kendi özel Check-In ilkelerinizi nasıl geliştirebileceğinizi gördünüz. Böylece geldik bir yazımızın daha sonuna. Tekraradan görüşünceye dek hepinize mutlu günler dilerim.

TimeoutPolicy.rar (30,07 kb)

Tasarım Desenleri - State (2009-08-06T21:00:00)

design patterns,oop,c#,

Merhaba Arkadaşlar,

Bir süre öncesine kadar özel bir bankada uzman yazılım geliştirici ettiğim hususlardan biriside otomat makinesi idi. Makineyi sev gıdalar olurdu. Makinenin başına geçer, yemek istediğim ürüne ba ürünün kodunu tuşlardım. Makine, ürünü benim için ilgili yerden huyunada çok kızardım. Para üstü vermezdi Eksiği bazı ürünle ürün olmazdı. Yine mesai yaptığım akşamların birisinde makineye öylece kala kaldım.

çünkü makine sözüm ona ürünü vermişti. Ancak makinenin alt sepetinde ürün yoktu. Nitekim ürün tam bulunduğu cepten aşağıya doğru düşmek üzereyken oracıkta takılıvermişti. Para gitmişti, nitekim makinin dijital kısmında Teşekkürler yazıyordu Ben olaya klasik bir insan piskolojisi ile yaklaştım. Makineyi öne arkaya itekleyerek ürünü takıldığı yerden düşürttüm ve afiyetle yedim. Makineye pis pis bakarken aklıma şunlar geldi. Makineye yaklaşırken durağandı. öylece birbirimize bakıyorduk. Sonra paramı attıp ürünü seçtiğimde makine bir dizi kontrol yaptı ve hazırlık moduna geçti. Ardından ürünü bana teslim etmek üzere kendi içerisindeki mekanikleri çalıştırdığında ürünü teslim etme modundaydı. Peki ürün takılıp bana veremediğinde hangi moddaydı da "Teşekkürler" diyip, paramı yutup, ürünü vermemişti Her neyse konumuz bu değil tabiki. Ama bu makinenin bu senaryo içerisinde anlattığım tüm durumları aslında yazılım terminolojisinde **State Machine** tipinden bir akış ile ifade edilebilmektedir. İşte bu günkü konumuz **State** tasarım kalıbı...

Davranışsal(Behavioral) tasarım desenlerinden olan **State** kalıbı, bir nesnenin **içsel durumunda(Internal State)** meydana gelecek değişimler sonrası çalışma zamanında dinamik olarak**farklı davranışları** sergileyebilmesini sağlayan bir model sunmaktadır. Aslında **State** tasarım kalıbını, **Workflow** terminolojisinde yer alan **State Machine** kavramının **nesne yönelimli(Object Oriented)** karşılığı olarak düşünebiliriz. öyleki nesnenin durumunun değişmesi halinde farklı davranışlar sergilemesi, sahip olduğu fonksiyonların tetiklenmesi ve bunlar arasında duruma göre gerekli **geçişlerin(Transitions)** sağlanması anlamına da gelmektedir. Dolayısıyla, **State Machine** kavramına aşina olanlarımız için **State** desenini kavramak son derece kolaydır.

Farklı bir örnek ile devam edelim. Bu amaçla bir müşterinin sahip olduğu banka hesabının durumlarını göz önüne alabiliriz. Bakiyenin içeriği müşterinin para yatırmasına, çekmesine, faiz ödemesine, fon alıp satmasına vb... gibi aksiyonlara göre sürekli değişiklik gösterecektir. Bir başka deyişle hesabın kendi iç durumunda bir takım değişiklikler olması söz konusudur. Bu değişikliker oldukça hesabın farklı durumları olması*(bir başka deyişle müşterinin farklı şekillerde değerlendirilmesi)* gerekir. örneğin fazla borçlanma nedeniyle farklı bir hesap durumu olmalıdır. Yada hesabın ilk açılmasında başlangıç durumu tesis edilmeli, standart faiz oranları belirlenmelidir*(Aynen otomat makinesinin prize takıldığında ön hazırlıklar yaptığı sıradaki konumu gibi)*. Hatta müşterinin düzenli ödemelerinin ona ekstradan bir anlam katması sonucu, hesabının kolay kredi almaya uygun bir duruma geçmesi mümkün olabilir.

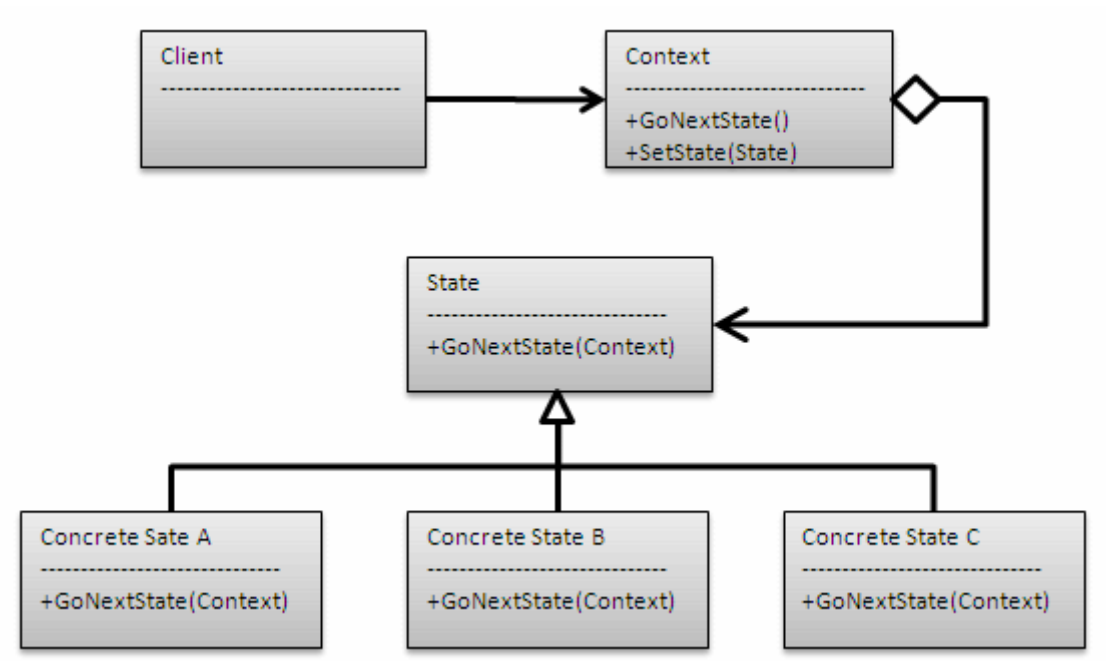
Yazılım tarafından olaya baktığımızda aslında **State** diagramları ile ifade edilebilen her nesne için **State** deseninin uygulanabileceğini düşünebiliriz. örneğin uygulamanın çalıştığı makinenin bellek durumları **State** kalıbına uygun olarak tasarlanabilir. Makinin normal seviyede olması, sistem kaynaklarının çok tüketilmesi sonucu alarm haline geçmesi veya alarm verilmeden önce uyarı moduna geçmesi söz konusu olabilir. Bu durumlar arasındaki geçişler aslında bilgisayarın bazı iç değerlerine göre gerçeklenir. Memory, CPU, Running Process ölçümleri birer kriter olabilir ve örneğin Computer isimli bir nesnenin iç durumunu ifade edebilir.

Başka bir örnek olarak oyun programlarında yer alan bazı senaryoları verebiliriz. Söz gelimi **RPG** tipinden bir oyunda yer alan herhangibi kahramanı düşünelim. Bu kahramanın duruma göre savaşması veya bir takım kontrollerde bulunması gibi davranışları, State deseninden yararlanılarak modellenebilir. öyleki, savaş halinde iken kahramanın tüm gücüyle çarpışması, aynı zamanda devriyede olması söz konusu iken, barış halinde savaşmaması ama devriyeye devam etmesi durumları söz konusudur. Bu durumların nesne yönelimli tarafta ifadesinde State kalıbından yararlanılır.

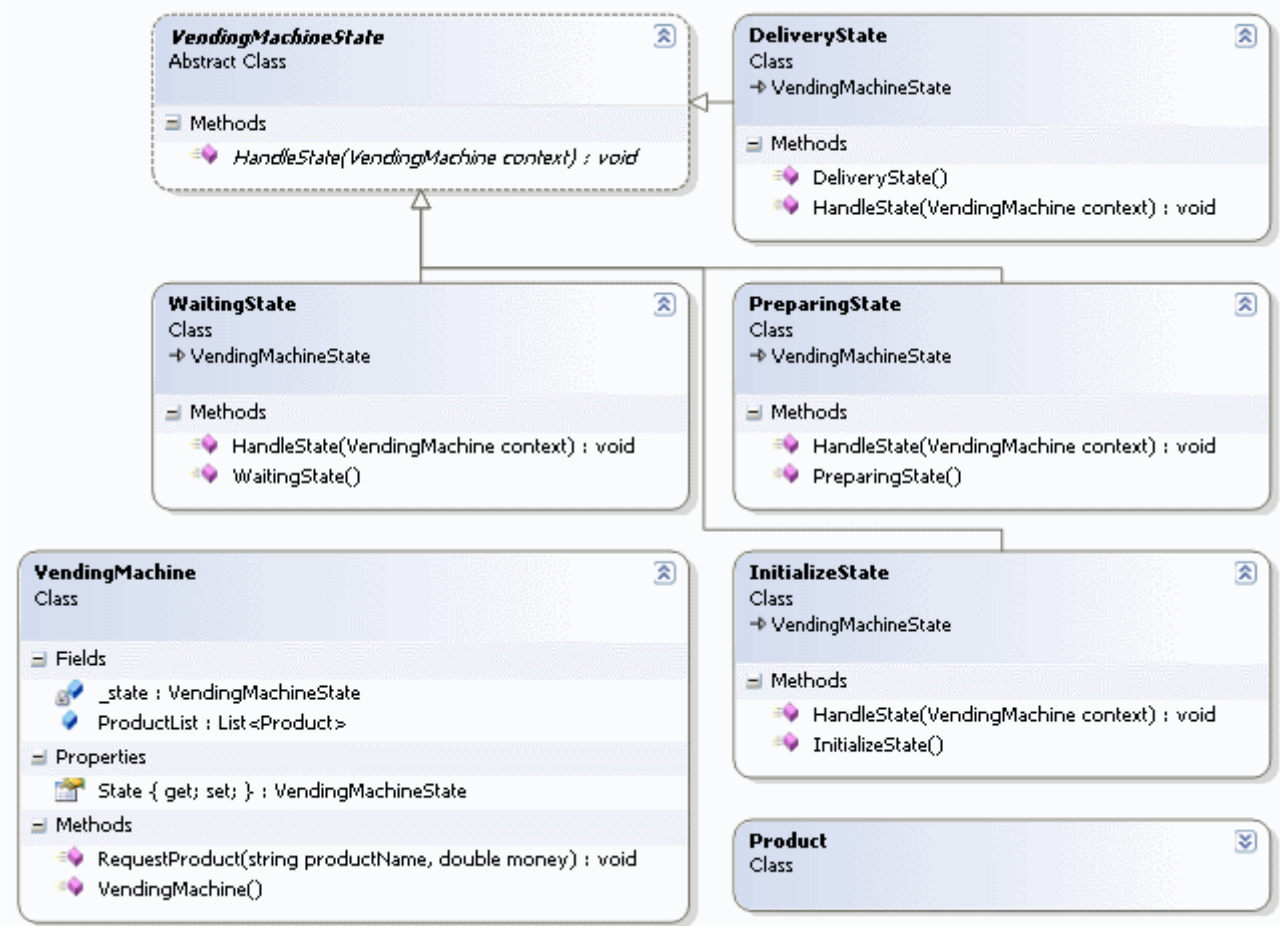
Aslında tüm bu örneklerde dikkat edilmesi gereken ortak bir notkada vardır. State tasarım kalıbında, durum değişmelerine neden olacak*(yani davranışların farklılaşmasına)* bir takım nesne içi değerler vardır. Bunların tamamı aslında davranış değişimi için takip edilecek içeriği oluşturmaktadır. Müşteri hesabı örneğinde Hesap(Account) asıl içeriği oluşturmaktadır. Bilgisayarın durumlarının ele alındığı örnekte makinenin kendisi asıl içeriği oluşturmaktadır. Hımmm... Bu durumda ortaya şöyle bir soru çıkmaktadır. İçeriğindeki veri değişimleri eğer bir nesnenin davranışlarını belirliyorsa, bu davranışların n sayıda olması ve içeriği sağlayan tip tarafından kullanılması nasıl sağlanabilir?

Bundan sonra **internal state'** i taşıyan nesneye **Context** dediğimizi düşünelim. Birden fazla davranış ve doğal olarak durum olabileceğinden, **Context'** in farklı durumlara erişebilip aralardaki**geçişleri(Transitions)** sağlayabilmesi gerekir. Bu durumda, **Context** tipinin tüm durumlar için ortak bir arayüz sunan başka bir tip ile*(buna State diyebiliriz)* **Aggregation** ilişkisini sağlaması uygundur. **State** tipinin kendisi aslında, **Context** tipinin belli bir durumu ile ilişkilendirilmiş davranışların kapsüllenmesi için bir arayüz sunmaktadır. Bu arayüz sunumu **aslı durum tipleri(Concrete**

State) tarafından değerlendirilebilir. Aslında bu yazdıklarımızdan deseninin sınıf diagramını az çok hayal edebiliriz.



E haydi öyleyse basit bir örnek ile kalıbı kavramaya çalışalım. Senaryomuzda yazımızın başında bol bol kulakları çınlayan otomat makinesini ele alıyor olacağız. 😃 Tabiki amacımız kalıbın nasıl uygulandığını ele almak olduğundan mümkün olduğunca sade(her zamanki gibi) bir örnek geliştireceğiz. Otomat makinesi için olası durumları şu şekilde düşünebiliriz. Makine elektrik şalterinden açıldığında bazı ön hazırlıklar yapar. Bu zaman diliminde makine Initialize modundadır**(InitializeState). Initialize** işlemleri başarılı ise makine bekleme moduna geçer**(WaitingState)**. Ne bekler? Tabiki bizden bir ürün almamızı 😉 Müşteri bir ürün talep ettiğinde bunu almak için makineye para atması ve sonrasında seçimi bildirmesi gerekir. Bu işlemi **Context** tipimiz içerisindeki bir metodun üstlendiğini düşünebiliriz. Eğer atılan para yeterli ise ürünün hazırlanması moduna geçilir**(PreparingState)** ve işlem başarılı bir şekilde tamamlanırsa ürün teslim edilir**(DeliveryState).** Kısaca **Context** tipi olarak düşündüğümüz **VendingMachine** sınıfı için dört farklı **durum(State)** düşünüyoruz. İşte sınıf diagramımız;



ve kodlarımız

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading;

namespace StatePattern
{
   // State tipi
   // abstract sınıf olabileceği gibi interface şeklinde de tasarlanabilir
   abstract class VendingMachineState
   {
      public abstract void HandleState(VendingMachine context);
   }

   // Concrete State tipi
   // Otomat start düğmesine basılarak çalıştırıldığında öncelikli olarak bir ön hazırlık yapacaktır.
   class InitializeState
      :VendingMachineState
   {
```

```
        public InitializeState()
        {
            Console.WriteLine("Initialize...");
        }
        public override void HandleState(VendingMachine context)
        {
            Console.WriteLine("ön hazırlıklar yapılıyor");
            Thread.Sleep(2000);
            // Makinenin durumu değiştiriliyor. Makine initialize edilmiş. Bekleme konumuna
geçebilir.
            context.State=new WaitingState();
        }
    }

    // Concrete State tipi
    class PreparingState : VendingMachineState
    {
        public PreparingState()
        {
            Console.WriteLine("Preparing...");
        }
        public override void HandleState(VendingMachine context)
        {
            Console.WriteLine("İstenilen ürün hazırlanıyor. Lütfen bekleyiniz");
            // Makienin durumu değiştiriliyor. ürün hazırlanması bitmiş. Buna göre ürünü
teslim etme durumuna geçiyor.
            context.State = new DeliveryState();
        }
    }

    // Concrete State tipi
    class WaitingState
        : VendingMachineState
    {
        public WaitingState()
        {
            Console.WriteLine("Waiting...");
        }
        public override void HandleState(VendingMachine context)
        {
            int totalProduct=context.ProductList.Sum<Product>(p => p.Count);

            Console.WriteLine("Makine bekleme konumunda. Şu anda {0} adet ürün
var.",totalProduct.ToString());
            // Makine bekleme konumundayken  aslında bir State değişikliği söz konusu değil.
```

Değişimi sağlayacak olan aslında istemcinin vereceği bir aksiyon. Context tipi üzerindeki RequestProduct metodunun çağırılması bu anlamda düşünülebilir.

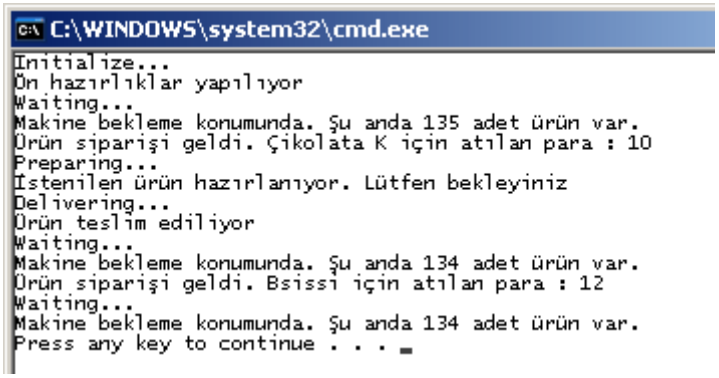
```
        }
    }

    // Concrete State tipi
    class DeliveryState
        : VendingMachineState
    {
        public DeliveryState()
        {
            Console.WriteLine("Delivering...");
        }
        public override void HandleState(VendingMachine context)
        {
            Console.WriteLine("ürün teslim ediliyor");
            // Makinin durumu değiştiriliyor. ürün teslim edildikten sonra tekrar bekleme konumuna alınıyor.
            context.State = new WaitingState();
        }
    }

    // Context tipi
    class VendingMachine
    {
        public List<Product> ProductList = new List<Product>();
        // Context tipi, kendi içerisinde State nesne referanslarını değiştirebilir. Bunun için State tipinden bir özellik sunmaktadır
        private VendingMachineState _state;

        public VendingMachineState State
        {
            get { return _state; }
            set
            {
                // State değiştiğinde, üretilen State nesne örneğinin çalışma zamanındaki referansına ait HandleState metodu çalıştırılır. Parametre olarak o anki Context gönderilir.
                _state = value;
                // Burada durum değişimleri sonucu çalıştırılacak davranışların başlatılma noktasınıda merkezileştirmiş oluyoruz.
                _state.HandleState(this);
            }
        }
```

```
        // Context nesnesi örneklenirken başlangıç durumu belirtilir.
        public VendingMachine()
        {
            // Test için makineye örnek ürünler yüklenir.
            ProductList.Add(new Product { Name = "çikolata K", ListPrice = 10,Count=50 });
            ProductList.Add(new Product { Name = "Biskuvi Bis", ListPrice = 3.45
,Count=50});
            ProductList.Add(new Product { Name = "Tuzlu mu tuzlu çıtır", ListPrice = 4.50
,Count=35});

            // Makineye ürünleri yükledikten sonra durumunu değiştir
            State = new InitializeState();
        }
        public void RequestProduct(string productName,double money)
        {
            Console.WriteLine("ürün siparişi geldi. {0} için atılan para :
{1}",productName,money);
            Product prd = (from p in ProductList
                           where (p.Name == productName && (money >= p.ListPrice && p.Count
>= 1))
                           select p).SingleOrDefault<Product>();

            // Eğer talep edilen ürün stokta var ve atılan para yeterli ise
            if (prd != null)
            {
                prd.Count--;
                // Makinenin durumunu değiştir
                State = new PreparingState();
            }
            else
                State = new WaitingState();
        }
    }

    // Yardımcı tip
    class Product
    {
        public string Name { get; set; }
        public double ListPrice { get; set; }
        public int Count { get; set; }
    }

    // Client
    class Program
    {
```

```
        static void Main(string[] args)
        {
            // Context tipine ait nesne örneği oluşturulur
            VendingMachine machine = new VendingMachine();

            // İstemci bir ürün ister
            machine.RequestProduct("çikolata K",10);

            machine.RequestProduct("Bsissi", 12); // Bu ürün olmadığı için vermeyecektir. Herhangibir aksiyon alınmayacaktır.
        }
    }
}
```

örneğimizde, **Client** yani müşteri makineyi çalıştırarak işe başlıyor. Bir başka deyişle **VendingMachineI(Context)** tipinden bir nesne örneği oluşturuluyor. Bu nesne ayağa kalkarken içerisindeki bir listeye 3 farklı üründen değişik miktarlarda aktarıyor. Bu noktada **VendingMachine** nesnesinin durumlarında da değişmeler oluyor. Sonrasında, machine isimli nesne örneği üzerinden **RequestProduct** metodu çağırılıyor. Yani müşteri makineden bir ürün istiyor. Bu sırada, yine makinenin durumları arasında bazı geçişler oluyor. özet olarak makinenin iç durumunda yapılan bazı değişikliklere göre farklı durumlara geçmesi ve farklı davranışların sergilenmesi sağlanıyor.

Ben geliştirdiğimiz örnekte pek çok durumuda göz ardı ettim. örneğin makinede talep edilen ürünün olmaması, atılan paranın yetersiz kalması veya fazla gelmesi yada makinin fişten çekilmesi hali...Bu olaylar gerçkeştiğindede aslında makinenin farklı durumlara geçmesi ve dolayısıyla **Context** tipinin farklı davranışlar sergilemesi gerekebilir. Bu kısımları, bir desen uyguladığımız için sisteme eklememiz aslında son derece basittir. örneğimizi çalıştırdığımızda aşağıdakine benzer bir sonuç ile karşılaştığımızı görebiliriz.



Tabiki yukarıdaki gibi bir deseni uygulamak yerine her şeyi **if** veya **switch** gibi kontrol deyimleri ile ele almaya çalışabiliriz. Tabi bu durumda hem kodun karmaşıklaşmasına neden olur hemde genişletilebilirliğini zorlaştırmış oluruz. Nitekim şu anda uygulanan desene göre, makine için yeni bir davranış eklemek aslında **State** arayüzünden türüyen bir tip ekleyip bunu ilgili yerlerde değerlendirmekten başka bir işlem değildir. Bunu daha iyi

anlamak için aynı örneği **if** ve **switch** yapıları ile geliştirmeye çalışmalısınız. Böylece geldik bir tasarım deseninin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

StatePattern.rar (26,49 kb)

C#Nedir? Yeni Çehresi ve Benden Size Tavsiyeler (2009-08-06T20:20:00)

c#nedir?,

Merhaba Arkadaşlar,

Yıllardır makale ve video editörlüğü yaptığım C#Nedir? sitesi yenilendi.

Serbest Köşe, Canlı Köşe, Blog Roll gibi yeni bölümleri bulunan, **Silverlight** gibi **RIA** teknolojilerine yer verilen yeni yüz bence oldukça başarılı. Ama tabiki yüz ve çehre pek bir anlam ifade etmiyor. Kaliteli olan içeriğin devamlılığını sağlamak ve istikrarı korumak lazım. Bu anlamda editor olmak isteyen arkadaşlarımıza bir kaç tavsiye vermek isterim.

- Yazmak istediğiniz konu hakkında doğru bilgiler ve örnekler verebilecek kadar iyi bilgiye sahip olun.
- Eğer sahip değilseniz o zaman çok çok iyi araştırın.
- En az 3 geçerli kaynaktan araştırmanızı yapın. Sonrasında 4ncü kaynak olarak kendi hayal gücünüzü kullanmaya gayret edin. Eğer olmuyorsa 3 elemanlı araştırma iterasyonunu farklı kaynaklarla yineleyip örneklerin kafanızda daha kolay canlandırılmasına uğraşın.
- Araştırmanızı güvenilir kaynaklardan ve ağırlıklı olarak konunun üreticisinin dökümantasyonundan temin edin(MSDN gibi).
- Anlatılanları uygularken ispat etmeye gayret edin. Bilgisayar çevresinin çok değişken parametrelerinin olduğunu duruma göre tahmin edilemeyecek veya hesaplanamayacak riskler olabileceğini unutmayın. Eğer böyle riskler var ise en azından tespit edebildiklerinizi paylaşın.
- Kaynak olarak internet ve orjinal kitaplardan(Illegal PDF değil, lütfen yazara saygı) yararlanın.
- Eğer kitap tedarik etmek istiyorsanız ve konunun Türkçe yazılı kaynaklarını bulmakta zorlanıyorsanız Amazon.com' dan tedarik etmeye çalışın.
- Internet kaynaklarından ve üreticinin dökümantasyonu dışında, konu hakkında uzmanlığını kanatlamış kişilerin blog girişlerinden faydalanın(MVP, Regional Director, Microsoft çalışanları vb...)

- Yeni duyurulan bir teknolojinin henüz release edilmemişse değişikliğie uğrayabileceğini göz önüne alın. Bunu, yazılarınızda yada videolarınızda mutlaka belirtin.
- Her zaman için konu ile ilişkili örneklerinizi açık bir dille sıkılmadan anlatın.
- örneklerinizi download edilebilir şekilde yazınıza ekleyin. Kötü niyetli program içerikleri oluşturmayın.
- Bazı konularda yazmak için sadece araştırmanın yeterli olmayacağını, gerçek hayat tecrübelerinden yararlanmanız gerektiğini ve belkide gerçekten konuyla ilgili içinden çıkılmaz durumlara düşmüş olmanızın bir nimet olabileceğini unutmayın.
- Eleştirilmekten korkmayın, yeterki okurları yanlış yönlendirmeyin(Bu pek çoğumuz için sağlanılması zor bir maddedir)
- Yazdıklarınızın mutlaka bir amacı olması gerektiğin unutmayın(Türkçe kaynak oluşturmak, konuyu yazılı anlatarak daha iyi öğrenmek, çalıştığınız iş yerine referans olarak gösterebilmek, iş arkadaşlarınızın yeri geldiğinde faydalanabileceği bir içerik oluşturmak, hayatınıza bir disiplin kazandırmak, sadece paylaşmak vb...)
- Yazınızda Türkçe dil bilgisi kurallarına uymaya özen gösterin.
- Gerekiyorsa yazınızı bitirdikten sonra en az iki kere gözle okuyup yazım hatalarını, devrik cümleleri düzeltmeyi deneyin. Ancak en iyi yazım kontrolünün ikinci bir kişinin okumasıyla olabileceğinide unutmayın.
- Araştırma yaptığınız kaynaklardan aldığınız bilgileri aynen kullanmayın. Aynen kullanıyorsanız kaynak belirtmeye özen gösterin.(Bazı konularda anlatımın zaten standart olması gerektiğinide unutmayın. örneğin bir tasarım desenini sadece bir yolla anlatabilirsiniz. Ama farklı örnekler vermek için kendinizi zorlayabilirsiniz.)
- Yazınızın sürekli iş yoğunluğunda olan editörler tarafından kontrol edileceğini/edildiğini, bu yüzden aceleci davranmayıp sabırlı olmanız gerekeceğini/gerektiğini unutmayın 

Sanıyorum ki bu maddelere dikkat ederseniz, yazacağınız/yazdığınız içeriklerin ne kadar kıymetli olacağını/olduğunu görebilirsiniz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

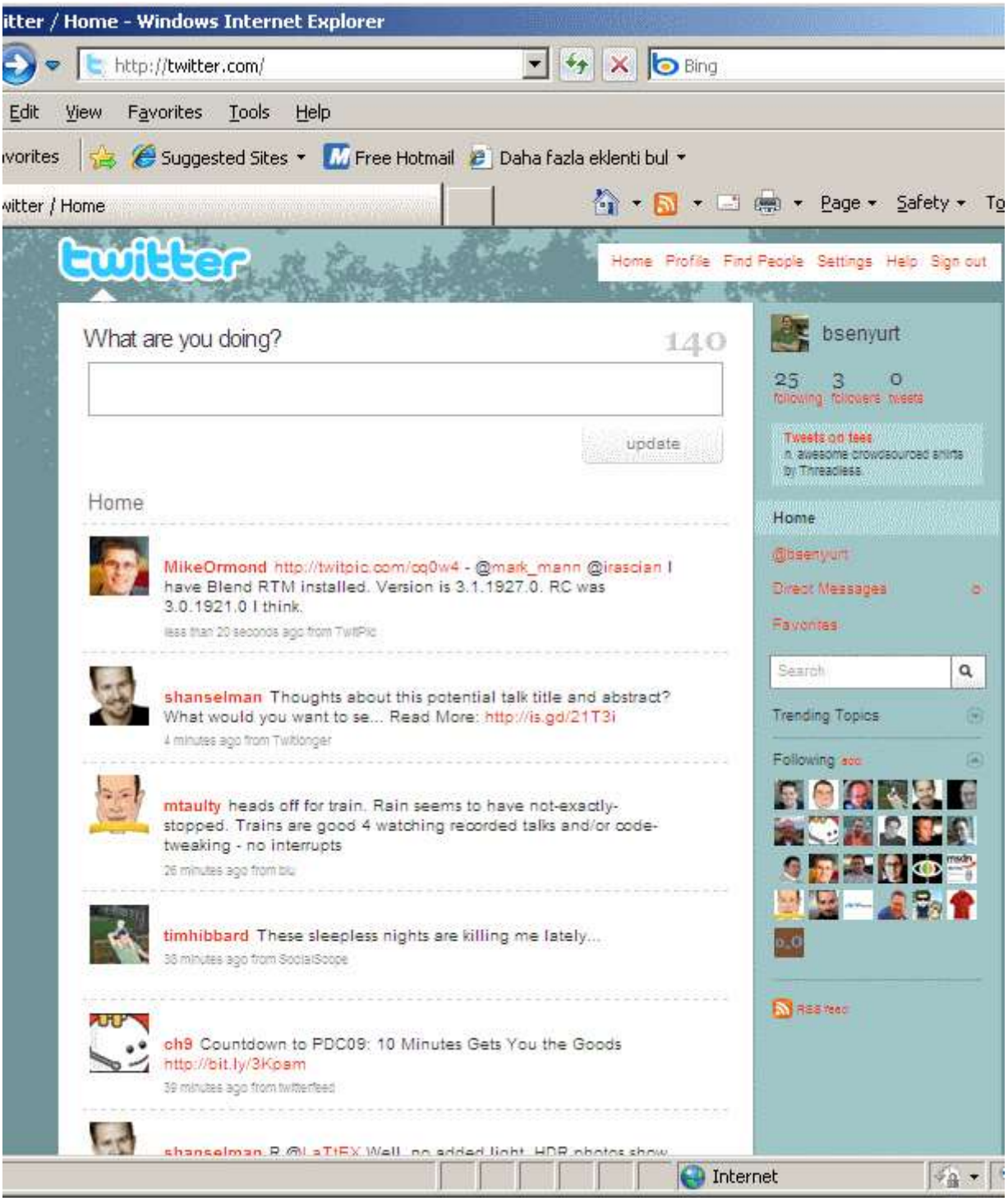
WCF Rest Starter Kit Preview 2 ile Twitter Reader (2009-08-05T08:47:00)

wcf,

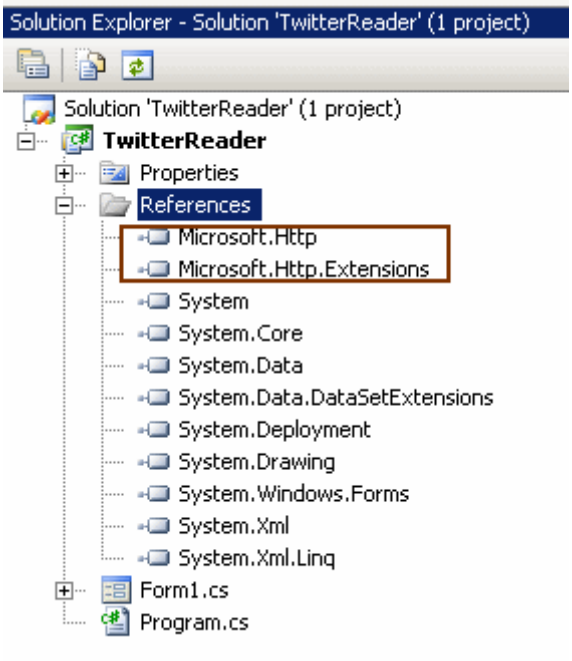
Merhaba Arkadaşlar,

Minik bir çocukken Televizyon bağımlılığı*(Malesef bu aptal kutuda* hatırlıyorum. Voltran, Transformers, Red Kit ve Daltonlar, Denver T diyen Tweety Şimdi bu konuya nereden geldiğimi düşünebilirsini *cikcik diyerek yazımıza başladığımızı anlamışsınızdır)*

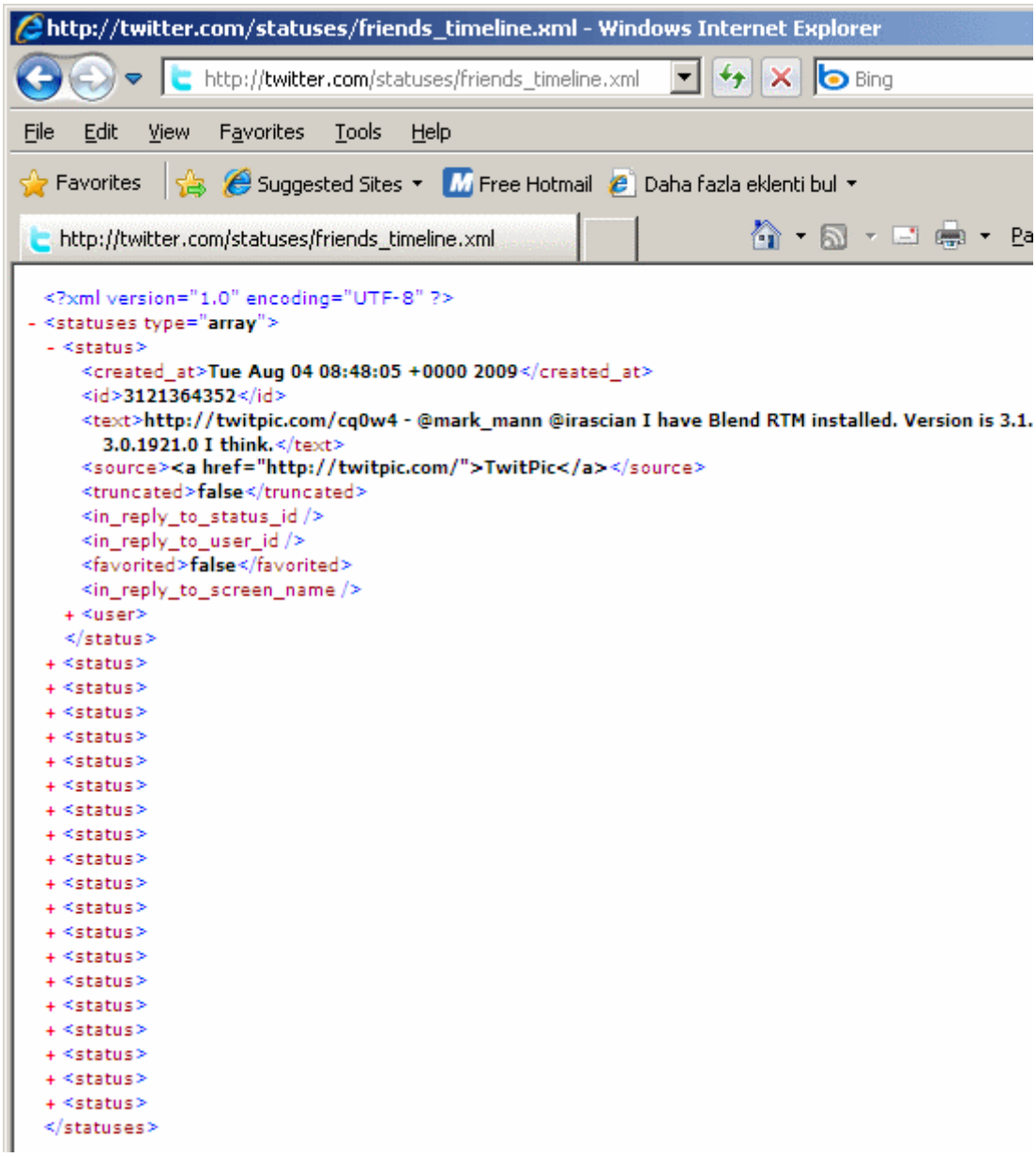
yayınlanan içeriklerin **WCF Rest Starter Kit Preview 2** ile birlikte gelen **HttpClient** sınıfı yardımıyla nasıl kolayca ele alınabileceğine dair bazı yazılar gördüm. Konunun içerisinde**REST** bazlı iletişim ve **WCF** söz konusu olunca hemen kolları sıvadım ve Windows tabanlı basit bir örnek geliştirmeye karar verdim. Tabi başlamadan önce projemizin amacından biraz bahsetmek isterim. Twitter üzerinde yayınlanan girişleri **HTTP** üzerinden **GET** metodu ile çekmeyi, buna göre eklenen güncel içerikleri uygulamamızda göstermeyi ve yenilerinide kendi **Twitter**hesabımız üzerinden, **HTTP Post** metodu ile ekleyebilmeyi planlıyoruz. Aslında olay bir RSS Reader yazmak kadar basit. Diğer yandan burada bahsettiğimiz işlevsellikleri geliştirmek için elimizde**WCF Rest Starter Kit Preview 2** olmasına da gerek yoktur. Ancak Kit' in bize sağladığı bazı avantajlar ve kolaylıklar bulunmaktadır. örneğin, XML içeriğini managed tarafta kolayca ele alabilmemiz için gerekli tiplerin üretimini kolaylaştıran **Paste XML As Types** örneği geliştirebilmek için çok sık kullanmasamda Twitter' da bir hesap oluşturdum ve bildiğim geliştiricilerin Tweet' lerini takip etmeye başladım. İşe başlamadan önce, Twitter'da ne olup bittiğine bir bakayım dedim.



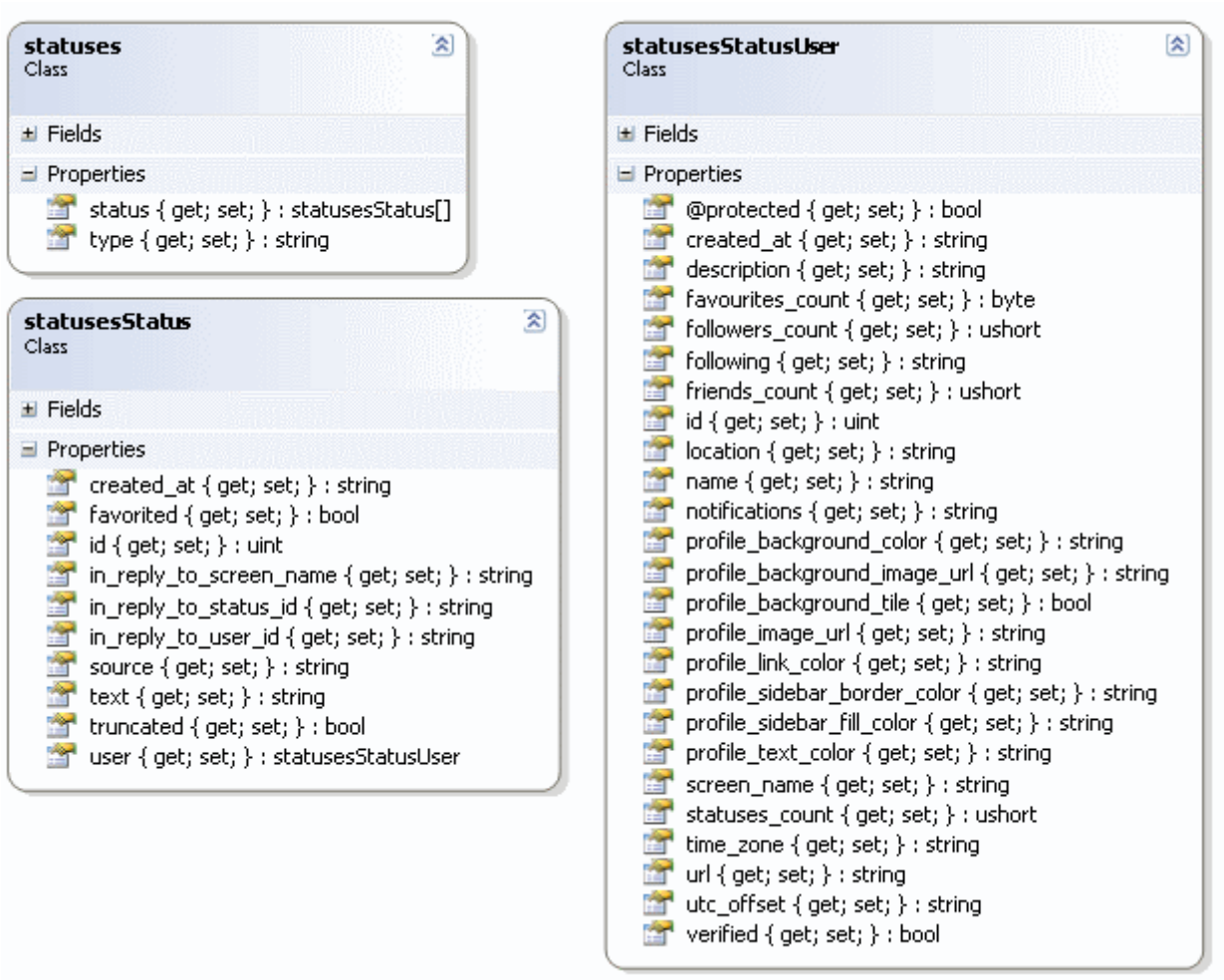
İşte buradaki içeriği Windows Uygulamasına çekmeyi hedefliyoruz. Rest Starter Kit' in nimetlerinden yararlanabilmek için, Windows uygulamasını oluşturulduktan sonra, WCF Rest Starter Kit Preview 2 ile birlikte
gelen **Microsoft.Http** ve **Microsoft.Http.Extensions** assembly' larının projeye referans edilmesi gerekmektedir.



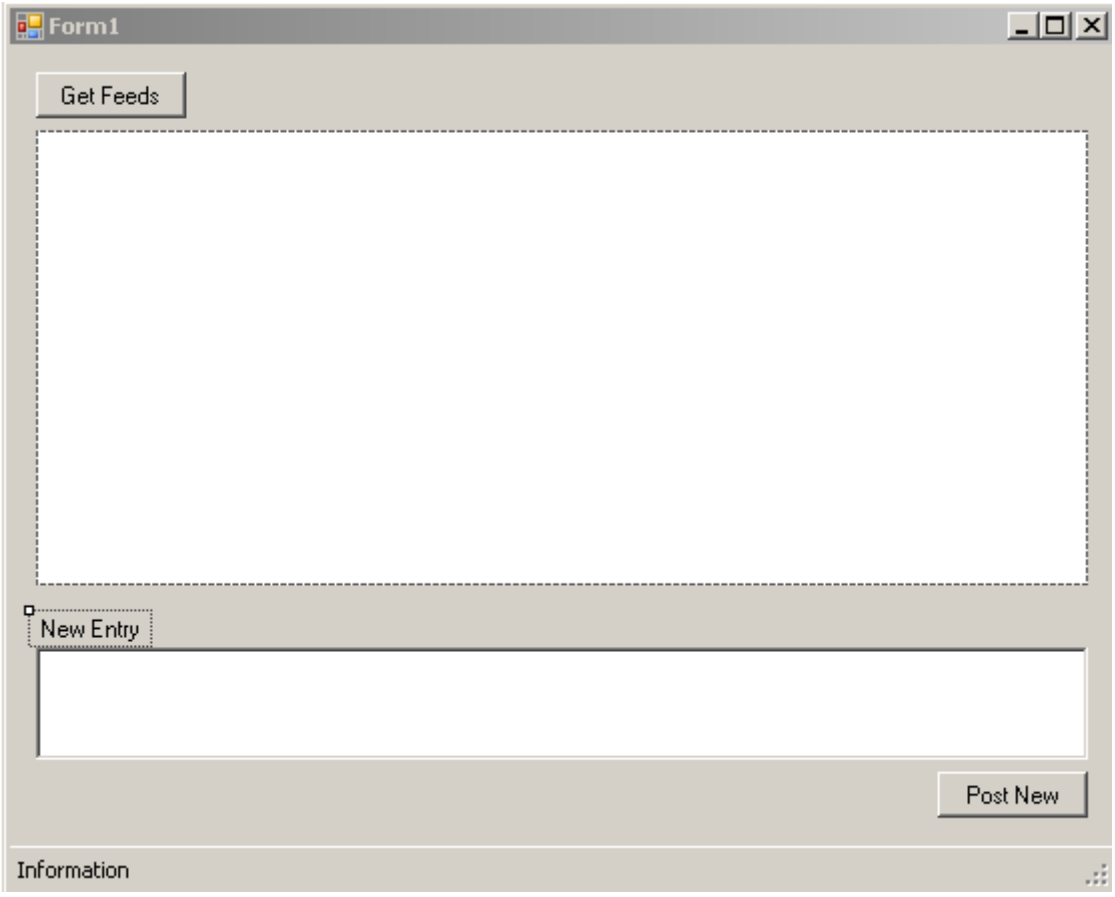
Referansların eklenmesinden sonra yolumuza, Twitter' da yayınlanan Feed içeriğinin managed taraftaki karşılıklarını oluşturarak devam edebiliriz. Bu noktada,http://twitter.com/statuses/friends_timeline.xml adresinde keni twitter hesabım ile baktığımda aşağıdaki ekran görüntüsünde yer alan XML içeriği ile karşılaştığımı gördüm.



Bu içeriği sayfanın **View Source** kısmını kullanarak kopyalayıp, **Paste XML as Types** seçeneğini yardımıyla*(WCF Rest Starter Kit Preview 2 ile Visual Studio 2008 ortamına eklenmiştir)* yönetimli kod tarafına dönüştürebiliriz. *(Burada XML içeriğinden managed tipleri oluştururken dikkat edilmesi gereken noktalardan birisi, **https** değil **http** ile talepte bulunmamızdır. Aksi durumda **View Source** seçeneği çalışmamaktadır.)* Diğer taraftan bu adres talebi sonrası var olan twitter hesabımız ile giriş yapmamız gerekmektedir. İşlem başarılı bir şekilde tamamlandığında sınıf diagramında aşağıdaki tiplerin oluştuğunu gözlemledim.



Harika! Görüldüğü üzere XML içeriğinin karşılığı olan sınıflar başarılı bir şekilde oluşturulmuştur. Dikkat edilmesi gereken noktalardan birisi, XML deki Child Node bağlantılarının sınıf bazında nasıl ifade edildiğidir. örneğin **statuses** sınıfı içerisinde **statusesStatus** tipinden bir dizi olarak tanımlanmış **status** özelliği... Bu özelliği kod tarafında değerlendirerek, **tweet** içeriğini giren **User'** a dahi ulaşabiliriz. Nitekim bunun için **statusesStatus** sınıfı içerisinde, **statusesStatusUser** tipinden **user** isimli bir özellik tanımlanmıştır. Zaten işin en önemli kısımlarından biriside bu XML içeriğinin, kod tarafınaki ifade şekli değil midir? Teşekkürler **Paste Xml As Types** Managed tiplerde oluşturulduğuna göre artık arka plan kodlarımızı geliştirebiliriz. *(Hemen şunu hatırlatalım.Paste Xml As Types ile üretilen sınıf ve üyelerinin adlarını dilediğiniz gibi değiştirebilirsiniz. özellikle yazım standartlarına uygun-CamelCasing isimlendirmeler yapılmasında yarar vardır. Şu an örneği hızlı bir şekilde geliştirme istediğinde bu noktaları atlamış bulunuyorum)* Windows Form' unu aşağıdaki gibi tasarlayabiliriz.



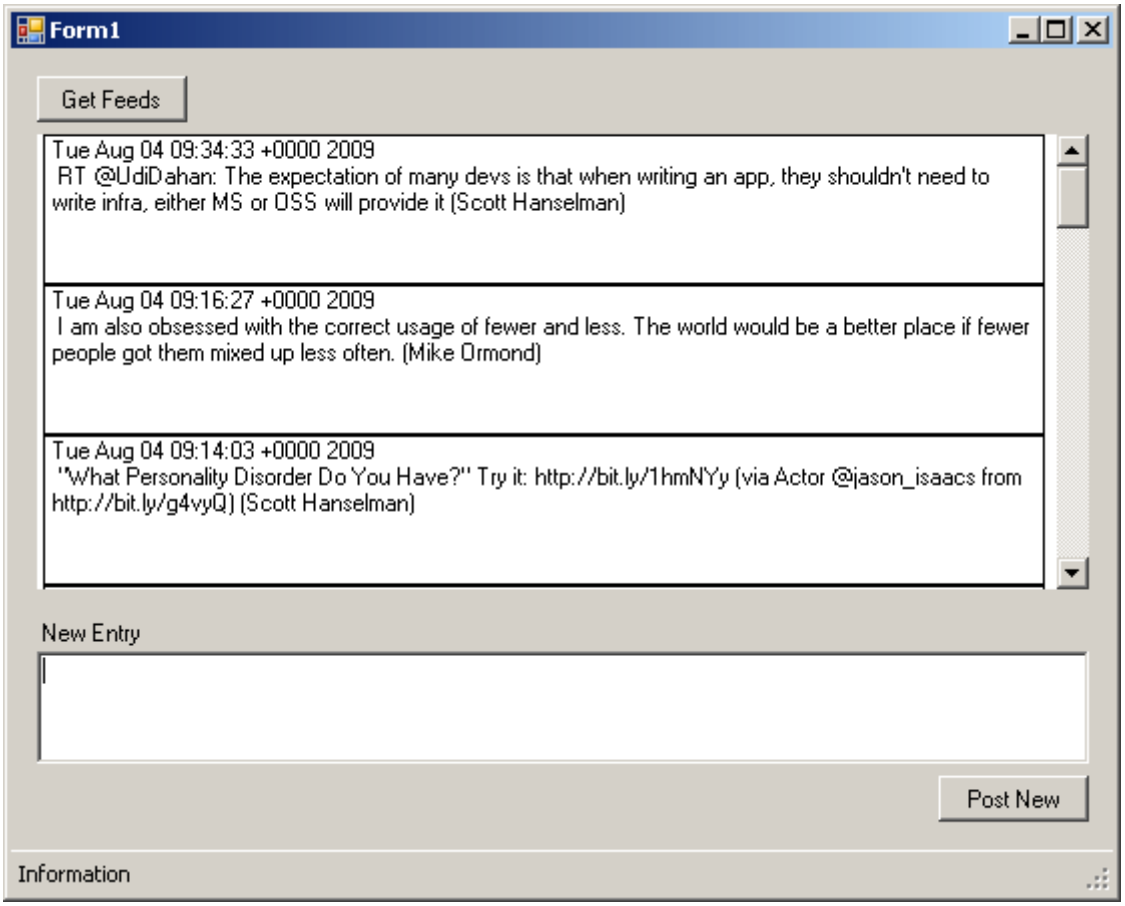
ve kodlarımız;

```
using System;
using System.Net;
using System.Windows.Forms;
using System.Xml.Serialization;  // ReadAsXmlSerializable<> genişletme metodunun çıkması için eklenmelidir.
using Microsoft.Http;// HttpClient için gerekli olan isim alanı.
using System.Drawing;

namespace TwitterReader
{
   public partial class Form1 : Form
   {
      // Siz kendi Twiteer kullanıcı adı ve şifrenizi kullanmalısınız.
      private string username = "sizin kullanıcı adınız";
      private string password = "sizin şifreniz";

      public Form1()
      {
         InitializeComponent();
      }
```

```
private void btnGetFeeds_Click(object sender, EventArgs e)
{
   try
   {
      pnlFeeds.Controls.Clear();

      // öncelikle HttpClient nesne örneği oluşturulur.
      HttpClient client = new HttpClient();
      // Xml verisini yukarıdaki adresten çekebilmek için geçerli bir Twitter hesabı ile erişmemiz gerekecektir. Bu nedenle NetworkCredential oluşturulur ve Credentials koleksiyonuna eklenir.
      client.TransportSettings.Credentials = new NetworkCredential(username, password);
      ServicePointManager.Expect100Continue = false;

      // XML içeriğine, HttpClient nesne örneği üzerinden GET talebinde bulunulur. Sonuç HttpResponseMessage nesne örneğine aktarılır.
      HttpResponseMessage response = client.Get("http://twitter.com/statuses/friends_timeline.xml");
      // İşlemin başarılı olunduğundan emin olunması sağlanır. HTTP 200 OK Kontrolu
      response.EnsureStatusIsSuccessful();

      // XML İçeriği okunur ve statuses tipinden nesne örneğine aktarılır.
      statuses stats = response.Content.ReadAsXmlSerializable<statuses>();

      // statuses nesne örneğinin status özelliği ile işaret edilen statusesStatus tipinden dizinin her bir elemanı dolaşılır.
      foreach (statusesStatus s in stats.status)
      {
         // örnek içeriği göstermek amacıyla her bir statusesStatus için birer Label üretilir

         Label lbl = new Label();
         lbl.AutoSize = false;
         lbl.Width = pnlFeeds.Width - 25;
         lbl.BorderStyle = BorderStyle.FixedSingle;
         lbl.Height = 75;
         // örnek bilgi olarak bilginin ne zaman eklendiği, içeriği ve kim tarafından oluşturulduğu ele alınır
         lbl.Text = string.Format("{0} \n {1} ({2})", s.created_at, s.text, s.user.name);

         pnlFeeds.Controls.Add(lbl);
      }
```

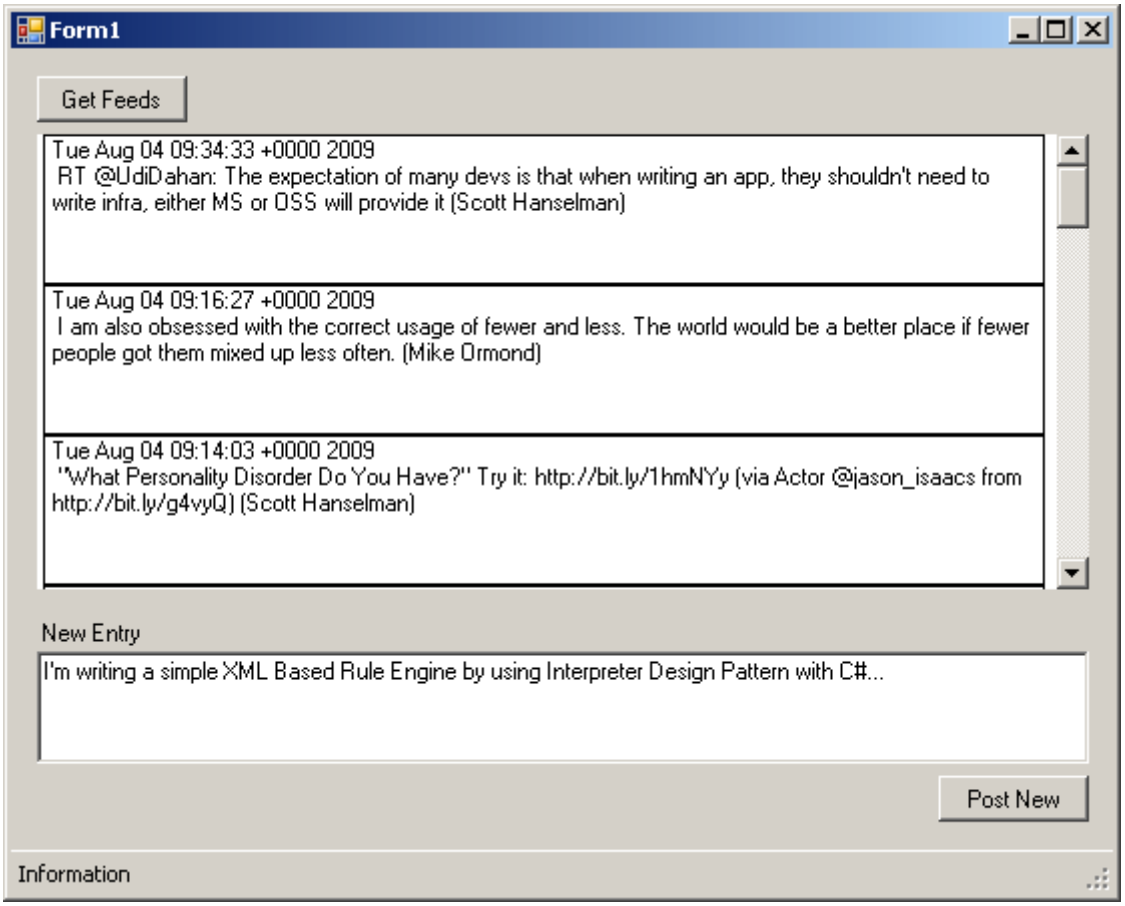
```
            }
            catch
            {
                lblStatus.Text = "Error!";
            }
        }

        private void btnPostNew_Click(object sender, EventArgs e)
        {
            try
            {
                // öncelikle HttpClient nesne örneği oluşturulur. Parametre olarak takip
edeceğimiz twitter adresi girilir.
                HttpClient client = new HttpClient("http://twitter.com/statuses/");
                // Xml verisini yukarıdaki adresten çekebilmek için geçerli bir Twitter hesabı ile
erişmemiz gerekecektir. Bu nedenle NetworkCredential oluşturulur ve Credentials
koleksiyonuna eklenir.
                client.TransportSettings.Credentials = new NetworkCredential(username,
password);
                ServicePointManager.Expect100Continue = false;

                HttpResponseMessage response = client.Get("friends_timeline.xml");

                // Yeni bilgi girişi için bir form oluşturulur
                HttpUrlEncodedForm form = new HttpUrlEncodedForm();
                // status için TextBox1 kontrolünün içeriğinin girileceği belirtilir
                form.Add("status", txtEntry.Text);
                // Bu kez HTTP Post metoduna göre update.xml adresine form içeriği gönderilir
                response = client.Post("update.xml", form.CreateHttpContent());
                // İşlemin başarılı olduğundan emin olunur
                response.EnsureStatusIsSuccessful();

                lblStatus.Text = "Entry posted.";
            }
            catch
            {
                lblStatus.ForeColor = Color.Red;
                lblStatus.Text = "Aaa...Houston...We have a problem...";
            }
        }
    }
}
```

örneğimizi çalıştırıp Get Feed başlıklı Button kontrolüne tıkladığımda aşağıdakine benzer sonuçlar ile karşılaştım.



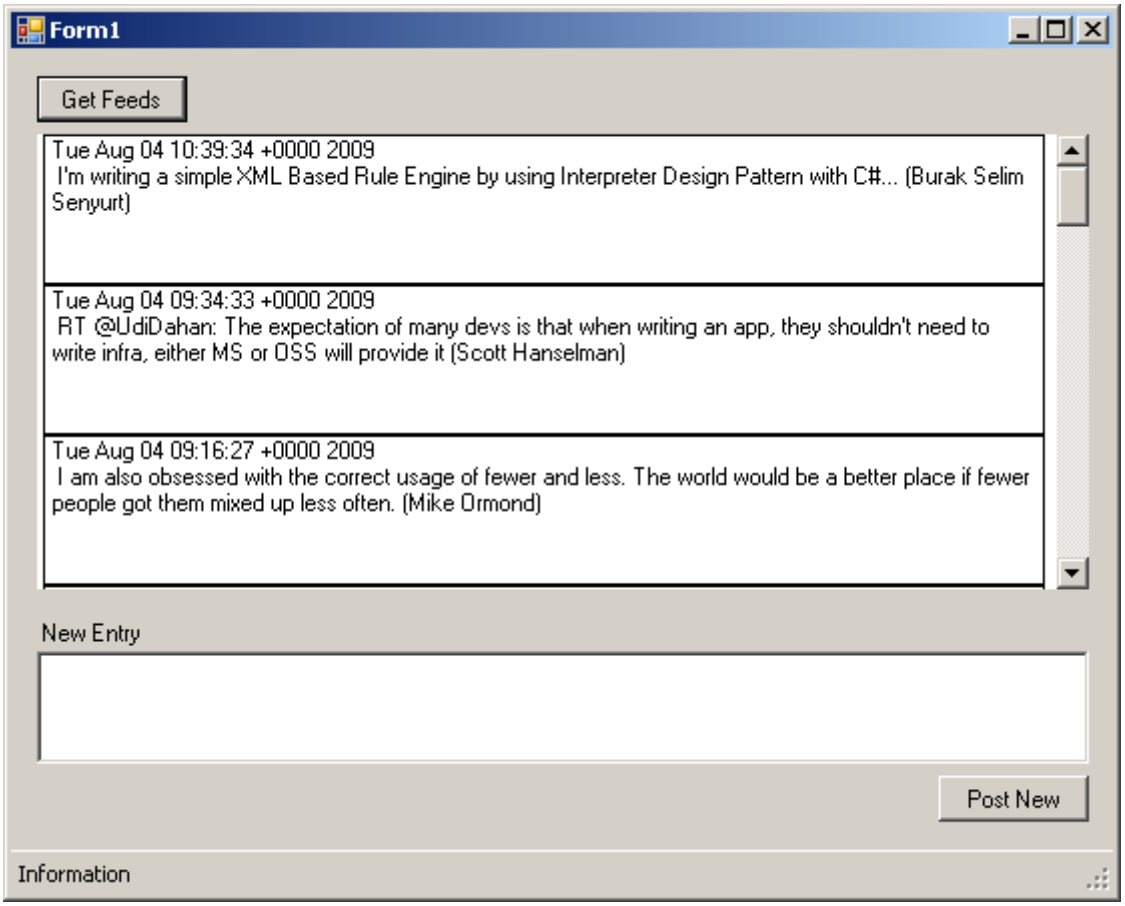
Görüldüğü üzere örneği geliştirdiğim sıradaki tüm Tweet girişlerini elde edebilmiştim. Evet, tasarım biraz kötü 😕 Hatta çok kötü 😃 Dahada güzelleştirilmesini size bırakıyorum.

Peki yeni bir Tweet girdiğimizde. örneğin aşağıdaki ekran görüntüsündeki gibi,



İşlem başarılı olduktan sonra Twitter sitesine baktığımda aşağıdaki gibi yeni Tweet' in eklenmiş olduğunu gördüm. Bu arada tweet' in kaynağı olarak **from API** yazdığına dikkat edin. Bu dış ortama sunulan **Twitter API** yardımıyla bir giriş yaptığımızı ifade etmektedir. Peki biz bu **API** için projemize bir referans ekledik mi? Hayır. Nitekim API'yi **REST** bazlı olarak kullanıyoruz. İşte işin güzel yanlarından birisi daha.

Windows uygulamasında tekrardan **Get Feed** düğmesini kullandığımda aşağıdaki gibi son eklenen Tweet bilgisinin de geldiğini gördüm.



Aslında web üzerinden takip edilen Tweet içeriğinin bir Windows uygulaması yerine zengin görselliğe sahip bir **WPF** uygulamasında ele alınması çok daha şık sonuçlar doğurabilir. Bu örnekte bizim için dikkate değer olan noktalardan biriside, **WCF Rest Starter Kit Preview 2** ile birlikte gelen **HttpClient**, **Paste Xml As Types** gibi yeniliklerin, gerçek hayat senaryosunda başarılı bir şekilde ele alınabilmiş olmasıdır.

Peki örnekte yapmadıklarım neler?

Her şeyden önce asenkron bir erişim söz konusu değildir. Bu nedenle verilerin çekilmesi veya yeni bir Tweet' in eklenmesi sırasında ekranda donmalar olmaktadır. Diğer taraftan çok güçlü bir Exception yönetimiz yok. Belki bir loglama sistemi koyarark Exception' ların uygulamayı geliştirenler için saklanması sağlanabilir. örneği geliştirirken bir Exception almamış olmama rağmen Tweet bilgilerinin çekilmesi sırasında bağlantıdaki aksaklıklar nedeni ile **Time out** istisnalarına düşülebileceğini tahmin ediyorum. Bunu kod içerisinde kontrollü bir şekilde ele almak yerinde olacaktır. Diğer taraftan **Button** kontrolü yardımıyla veri çekmek yerine, kullanıcının kendisinin set edebileceği zaman dilimleri içerisinde veri çekilmesi sağlanabilir. Bu işi bir **Timer**bileşeni kolayca halledebilir. Birde tasarım konusunda beni takip etmemenizi öneririm  Bunları deneyin ve çok daha iyisini yapmaya çalışın. Umarım yararlı bir yazı olmuştur. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

TwitterReader.rar (187,66 kb)

Tasarım Desenleri - Interpreter (2009-08-03T18:30:00)

design patterns,oop,c#,

Yandaki legoya baktığımızda sanıyorum ki hepimiz araba yarışı sahneleri geliyor. Her neyse...

Romalılar, **Mısırlıların** fikirlerinden yola çıkarak h hayatımızın bir döneminde **Roma Rakamları** ile k sonunda, çevrildikleri yıllar genellikle Roma rakam

Tabi Romalılar, geliştirdikleri bu sayı sistemlerinin bir gün gelipte GOF' un tasarım kalıplarından birisine ilham vereceklerini eminimki düşünmemiştir. *(Gerçi yazılım teknolojilerindeki pek çok sorunsalın çözümünde tarihten dersler alınmıştır. örneğin **Microsoft Solution Framework(MSF)** eğitim materyallerinde Kartaca savaşından bahsedildiğini çok iyi hatırlarım* 🙂 *)* İlhamı alan desen Interpreter tasarım kalıbıdır. Aslında kalıbın amacını anlamak için örnek senaryolara bir bakalım.

Diyelim ki çalışma ortamımız içerisinde şöyle bir bilgi yer alıyor. **"MDCLXIV".** Hatta bu bilgi, tarihsel kaynaklar ile ilişkili bir veritabanı giriş ekranından aynen bu metin formatında geliyor olsun. Ne varki, bu string bilgi yerine sayısal karşılığının bulunmasının daha önemli olduğu açıktır. Nitekim sayısal değer olması halinde bazı tarih bazlı hesaplamalar daha kolay yapılabilecektir. MDCLXIV değerinin karşılığı aslında 1664' tür. Nasıl mı? Aşağıdaki satıra geçmeden önce kağıt ve kalemi alıp hatırlamaya ve çözmeye çalışın 🙂

MDCLXVI = (M=1000)+(D=500)+(C=100)+(L=50)+(X=10)+(V=5)-(I=1) = 1664

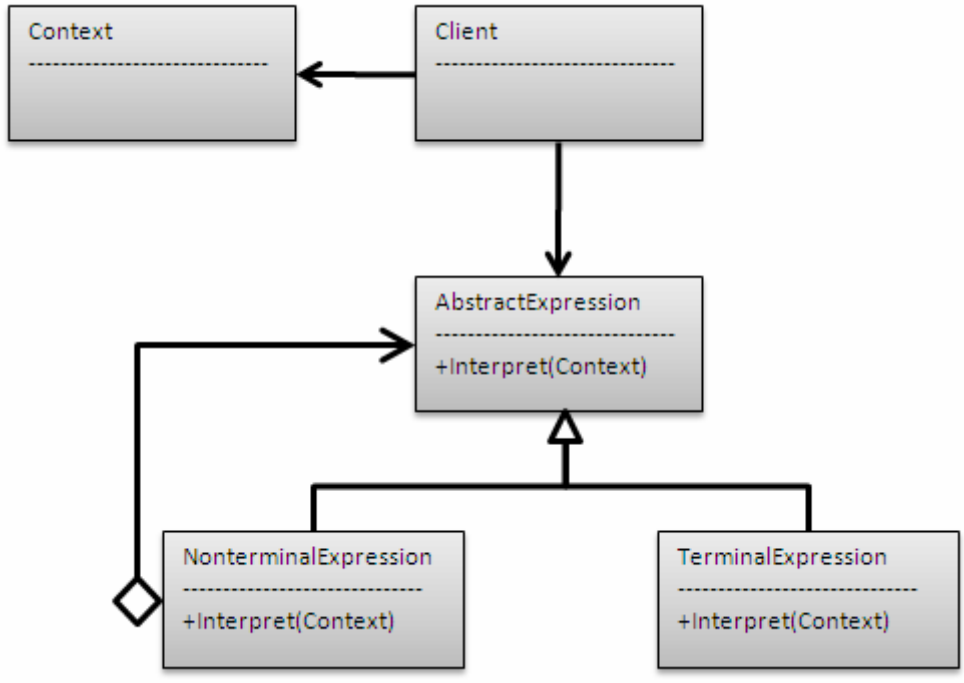
çok güzel. Peki programatik ortamda bu tip bir ifadeyi kim, nasıl yorumlayacaktır? İşte Interpreter tasarım kalıbının ana fikri bu tip ifadelerden oluşan bazı özel veri gramerlerini yorumlayabilecek bir yapının oluşturulması için bir model sunmaktır. Bu örnek son derece popülerdir. Pek çok kaynakta(başta DoFactory ve OODesign) Roma rakamlarının sayıya dönüştürülmesi işlemlerinin tasarım kalıbına örnek bir senaryo olarak sunulduğunu görebilirsiniz.

Hemen konuya farklı bir örnekle devam edelim. Söz gelimi günün tarihini sistemde **"MM - DD - YYYY"** şeklinde elde etmek istediğimizi farzedelim(Tabi hile yapıp DateTime fonksiyonlarını kullanmıyoruz). Bunun için günün tarihini, formatta gösterilen şekilde

sunmamız yeterli olacaktır. Ama olayı birde şu açıdan ele alalım. Burada tarih bilgisi için bir veri gramerimiz olduğunu düşünelim. Buna göre, günün tarihini sistemin o anki ihtiyaçları doğrultusunda **"MM - DD - YYYY", "DD - MM - YYYY", "D - MMMM - YY", "DD - YYYY", "MMMM - YY"** vb formatlara göre elde etmekte isteyebiliriz. Şimdi durum biraz değişti sanırım Hımmm... Bu durumda programatik tarafta ayrı ayrı parser yazmak çok da mantıklı olmayacaktır. Ne yapılabilir? Interpreter deseni ile bu tarih gramerini kolayca yorumlayabilir ve günün tarihinin istediğimiz formatta sunulmasını sağlayabiliriz.

Elbette başka örneklerde vermek mümkündür. Söz gelimi çok geliştirilmemekle birlikte **kural işletme motorları(Rule Engine)** söz konusu tasarım kalıbını sıklıkla kullanırlar. Buna göre kuralı oluşturan ifadeler ayrıştırılarak yorumlanır ve genellikle boolean(true/false) sonuçlar üretilir. Bir başka deyişle içeriğin, tanımlanan bir kurala uygun olup olmadığının kontrolü yapılır. Uygunluk true anlamındadır. Tabi desenin uygulanış biçiminde mantıksal değerlerin üretilmesi zorunlu değildir. Roma rakamı örneğinde bu açıkça görülmektedir.

Oldukça heyecan uyandıran bir desen olmasına karşılık, şekli belirli ve düzgün olan gramer ifadelerinde(Formal Grammer) değerlendirildiği için kısıtlı bir kullanım alanı söz konusudur. Sanıyorumki dofactory.com sitesinde desenin kullanım oranının neden %20' ler seviyesinde kaldığını bu cümle açıklamaktadır. Dilerseniz heyecanımızı kırmayalım ve desene ait UML şeması ile yolumuza devam edelim.



Aktörlerimizden **Context**, yorumlanacak içeriği taşımaktadır. Genellikle ifade bütününü ve yorumlama sonucunu kendi içerisinde taşıyan bir tip olarak düşünülebilir. **Context**içerisinde değerlendirilmesi gereken her bir parçanın yorumlanması operasyonunu ise **AbstractExpression** tipi sunmaktadır. Bu tip, abstract class olarak

tasarlanır ve grameri yorumlama kısımlarına ilişkin ana **iş mantığını(Business Logic)** üstlenebilir. Bazı durumlarda **interface** olarak tasarlandığınıda ve yorumlama işinin kendisinden türeyen tiplere bırakıldığını görebiliriz. UML şemasında dikkat çekici noktalardan birisi **TerminalExpression** ile **NonterminalExpression** isimli iki farklı Exrepssion tipi olmasıdır.

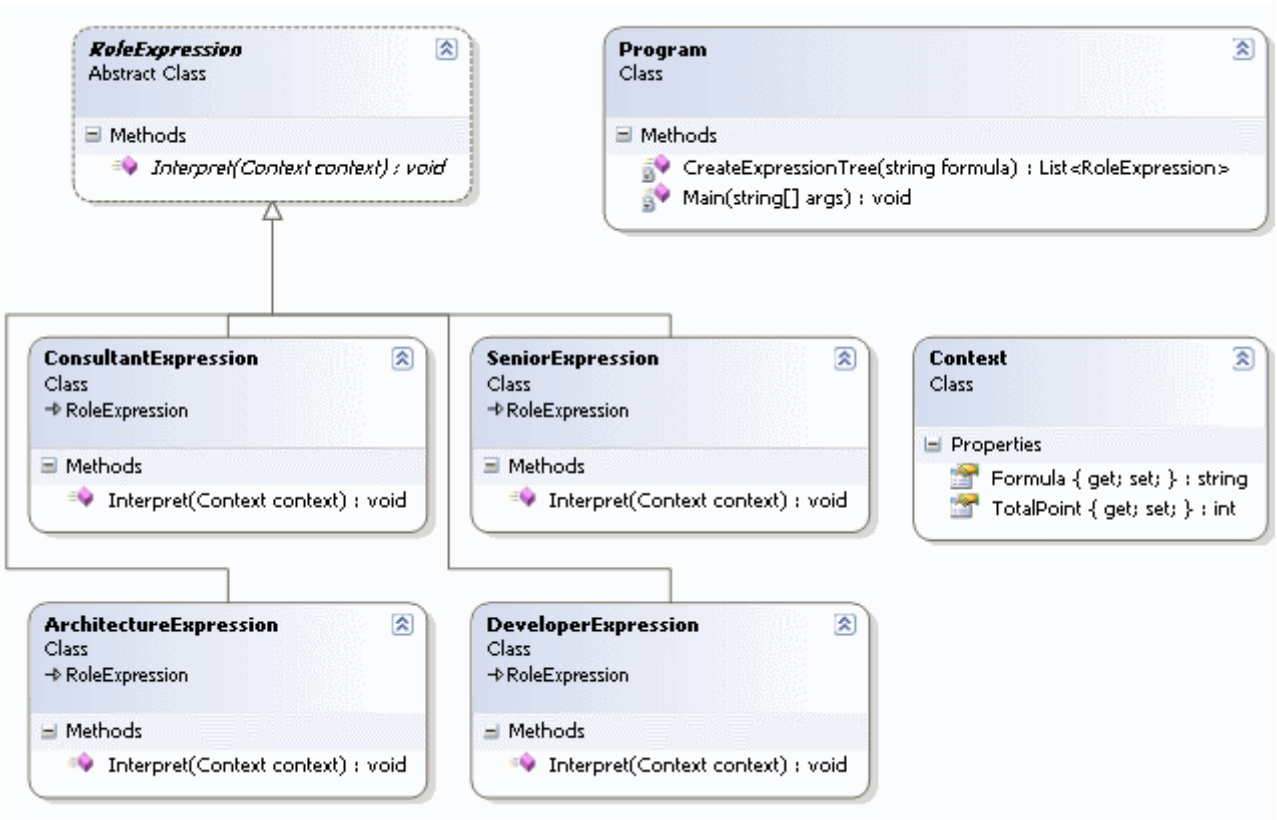
Aslında burada örnekler üzerinden hareket etmemizde yarar vardır. Roma rakamları örneğimizde her bir harf aslında **TerminalExpression** tipinden sınıflar içerisinde değerlendirilir. Ancak bir kural motorunda, ifadeler arasında bazı semboller ile işlemler yapılması gerekiyor olabilir. örneğin iki Terminal(yada Nonterminal) ifadenin or veya and ile bağlanması yada matematiksel operasyona sokulması gibi. İşte bu tip durumlarda kullanılan **ara semboller(and, or, + vb...) NonterminalExpression** tipleri içerisinde değerlendirilir. Şemadan görüleceği üzere, **NonTerminalExpression**tipinden **AbstractExpression** tipine doğru tanımlanmış bir **Aggregation** ilişki söz konusudur. Yani **NonTerminalExpression** kendi içerisinde **AbstractExpression** türevli referansları taşıyabilmelidir. Bu gereklidir, nitekim **NonTerminal** sembollerin **Terminal** ifadeleri üzerinde değerlendirilmesi söz konusudur. Söz gelimi **and** operasyonunun uygulanması istenen durumlarda **iki adet operand'ın(TerminalExpression)** olması gerekir.
Bunlar **NonTerminalExpression** içerisinde birer üye olarak bulunmalı ve **initialize** edilmelidirler. Bu nedenle bir **Aggregation** ilişkisi söz konusudur.

Artık bir örnek ile devam edebiliriz. Senaryo bulmak konusunda sıkıntımız olsada amacımızın desenin nasıl uygulandığını kavramak olduğunu bir kez daha hatırlatalım. Konuyu kolay bir şekilde ele almak için **NonTerminalExpression** nesnelerini hesaba katmayacağız. Elimizde bir projenin içerisinde yer alan çalışanların sembolik tanımlamalarını string bazlı taşıyan bir Context tipi olduğunu düşünelim. örneğin mimarlar için A, danışmanlar için C, uzman geliştiriciler için S ve geliştiriciler için D harflerini göz önüne alabiliriz. Buna göre, örneğin **ACSSDDDD** şeklindeki bir metnin bizim için anlamı,

ACSSDDDD = 1 Architecture + 1 Consultant + 2 Senior Developer + 4 Junior Developer

şeklinde olacaktır.

Bu metinsel bilgidende bir projenin adam başı maliyetini çıkartabildiğimizi düşünebiliriz. Dolayısıyla **string** ifadeyi tarayıp bize sayısal değer döndürecek bir yorumlayıcı modele ihtiyacımız var*(Burada string bir içeriği basit olarak alıp parse etmeyi hedeflemediğimizi belirtelim).* Buna göre **A, C, S** ve **D** harflerinin aslında birer **TerminalExpression** tipi olarak ifade edilebileceğini düşünebiliriz. İşte örneğimize ait sınıf diagramımız,



ve kod içeriğimiz.

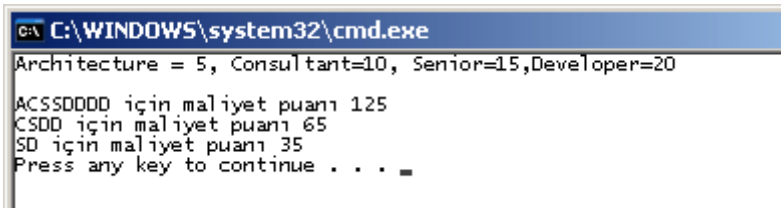
```
using System;
using System.Collections.Generic;

namespace Interpreter
{
   // Context class
   class Context
   {
      public string Formula { get; set; }
      public int TotalPoint { get; set; }
   }

   // Expression
   abstract class RoleExpression
   {
      public abstract void Interpret(Context context);
   }

   #region Terminal Expression Sınıfları

   // TerminalExpression
   class ArchitectureExpression
```

```
   : RoleExpression
{
   public override void Interpret(Context context)
   {
      if (context.Formula.Contains("A"))
      {
         context.TotalPoint += 5;
      }
   }
}

// TerminalExpression
class ConsultantExpression
   : RoleExpression
{
   public override void Interpret(Context context)
   {
      if (context.Formula.Contains("C"))
         context.TotalPoint += 10;
   }
}

// TerminalExpression
class SeniorExpression
   : RoleExpression
{
   public override void Interpret(Context context)
   {
      if (context.Formula.Contains("S"))
         context.TotalPoint += 15;
   }
}

// TerminalExpression
class DeveloperExpression
   : RoleExpression
{
   public override void Interpret(Context context)
   {
      if (context.Formula.Contains("D"))
         context.TotalPoint += 20;
   }
}

#endregion
```

```
// Client
class Program
{
    static List<RoleExpression> CreateExpressionTree(string formula)
    {
        // Expression ağacı oluşturulur
        List<RoleExpression> tree = new List<RoleExpression>();

        foreach (char role in formula)
        {
            if (role == 'A')
                tree.Add(new ArchitectureExpression());
            else if (role == 'S')
                tree.Add(new SeniorExpression());
            else if (role == 'D')
                tree.Add(new DeveloperExpression());
            else if (role == 'C')
                tree.Add(new ConsultantExpression());
        }
        return tree;
    }

    static void RunExpression(Context context)
    {
        foreach (RoleExpression expression in CreateExpressionTree(context.Formula))
        {
            expression.Interpret(context); // TerminalExpression tiplerine ait harf
sembolleri buradaki metod çağrısındada gönderilebilir.
        }
        Console.WriteLine("{0} için maliyet puanı {1}", context.Formula,
context.TotalPoint);
    }

    static void Main(string[] args)
    {
        Console.WriteLine("Architecture = 5, Consultant=10,
Senior=15,Developer=20\n");
        // 1 Architect, 1 Consultan, 2 Senior Developer , 4 Junior Developer
        Context context = new Context { Formula = "ACSSDDDD" };
        RunExpression(context);

        // 1 Consultant, 1 Senior Developer, 2 Developer
        context = new Context { Formula = "CSDD" };
        RunExpression(context);
```

```
            // 1 Consultant, 1 Senior Developer, 2 Developer
            context = new Context { Formula = "SD" };
            RunExpression(context);
        }
    }
}
```

Dikkat edileceği üzere **Main** metodunda bir **Expression Tree** oluşturulmaktadır. Bu **Expression Tree'** nin modellenmesi için **string** ifade içerisindeki tüm harfler tek tek dolaşılır ve uygun olan**TerminalExpression** nesne örnekleri üretilip, ağaca eklenir. Sornasında ise ağaç içerisindeki her bir **TerminalExpression** üzerinden **Interpret** metodu çalıştırılarak bir yorumlama işleminin gerçekleştirilmesi sağlanır. Yorumlama işlemi bu örnek için, karşılaşılan harflere göre bir puanlamanın, Context nesne örneği içerisindeki **TotalPoint** özelliğine yansıtılmasıdır. İşte uygulamanın çalışmasının sonucu.



Yaptığımız bu basit yorumlayıcı sadece metinsel bir ifade bütününü yorumlayarak ele almıştır. Aslında yapılan iş, berlirli bir grameri alıp sınıflara dönüştürmekle alakalıdır. Bu dönüştürme işlemi sırasında devreye Interpreter kalıbının aktörleri girmektedir. Grammer içerisinde yer alan herhangibir parça aslında bir **TerminalExpression** veya**NonTerminalExpression** olarak birer sınıfa dönüşür ve **AbstractExpression** tipi içerisinde veya türevlerinde işlenir. Sonrasında ise istemci uygulama, bir **Expression Tree** bütününü, kullandığı bir Context tipi üzerinde çalıştırır. Tabiki desenin bu basit uygulanış şekli dışında kural motorlarında olduğu gibi **NonTerminalExpression** tiplerininde işin içerisinde girdiği daha karmaşık uyarlamaları vardır. *(Dikkat edeceğiniz üzere birleşik bir metin var. Arada boşluklar veya aritmetiksel operatörler yok)* Bu uyarlamalardan basit bir örneğini ilerleyen blog yazılarımdan birisinde aktarmaya çalışıyor olacağım. Böylece geldik bir tasarım deseninin daha sonuna. Tekrardan görüşünceye dek hepinze mutlu günler dilerim.

Interpreter.rar (253,31 kb)

Windows Mobile 6.5 ile Widget Geliştirmek (2009-07-31T18:31:00)

webcast,

Windows Mobile 6.5 uygulama geliştiriciler ve son kullanı

Uygulama geliştiriciler tarafında bu yeniliklerin belki de en
Windows Mobile 6.5 üzerinde siz de Widget' lar geliştireb

Windows Mobile 6.5 üzerinde Widget kavramının ayrıntılı şekilde ele alınacağı ve örnek uygulamaların geliştirileceği bu seminer, aynı zamanda Türkiye'de Windows Mobile 6.5 hakkında gerçekleştirilecek ilk web semineri olma özelliğini de taşımaktadır.

Tarih : **04 Ağustos 2009 Salı Günü**
Saat : **21:00 – 22:00**

Konuşmacı: Device Application Development – MVP, Ekin özçiçekçiler.

Web seminerine kayıt olmak için buraya tıklamak ve Windows Live ID (Hotmail hesabı) ile giriş yapmak gerekmektedir.

Tasarım Desenleri - Iterator (2009-07-31T00:52:00)

design patterns,oop,c#,

Merhaba Arkadaşlar,

Küçüklüğümde pek çoğumuz gibi sahip olduğum bir pul koleksiyonum posta aracılığıyla yurt dışından arkadaşlar edinir, birbirleriyle pul değiş koleksiyonunuzu genişletiyorsunuz.

Tabiki posta mesajlaşması biraz zaman alan bir mevzuydu. Bu günkü gibi sosyal içerikli portallar veya mesajlaşma cihazları ve daha nice gelişmiş teknoloji yoktu. Acaba bu devirde yaşayan çocuklardan kaçı pul koleksiyonu yapıyor Neyse bu duygusal ortamdan çıkalım hemen. Pul koleksiyonumda yaptığım işlerden birisi zaman zaman onları baştan sonra, yada sondan başa, yada ortadan bir yerden herhangibir yöne doğru gözle taramak olurdu. Bazen kendi kafama göre sıralarını değiştirirdim. Peki nesne yönelimli dillerde kullandığımız koleksiyon veya dizi gibi veri yapıları üzerindede bu ve benzer işlemleri yapmıyor muyuz? çeşitli tipte **veri yapılarında(Data Structures)** dolaşıyor, içeriklerine bakıyoruz.

Koleksiyonlar, C# gibi bir programlama dilinde belkide en önemli **veri yapılarından(Data Structures)** birisidir. Bir koleksiyon kendi içerisinde farklı tipte veya aynı tipte nesneleri çeşitli formatlarda*(List, Stack, Queue, Dictionary vb...)* saklayabilen nesne bütünleri olarak düşünülebilir. Hatta bildiğiniz üzere **.Net 2.0** ile birlikte C# ve Vb.Net tarafına kazandırılan **generic** yetenekler ile, koleksiyonların **tip güvenli(Type Safety)** olarak ele alınmalarıda garanti edilmiştir. Hatta, C# 3.0 ve Vb 9.0 ile birlikte neler olmuştur neler Artık koleksiyonlar üzerinden **LINQ** sorguları yardımıyla sanki bir veritabanı tablosunu sorgularmışcasına filtrelemeler yapılabilmektedir. Ancak olayın en başından beri süre gelen ve bu yazımıza konu olan bir durumda söz konusudur. Bir koleksiyonun veya bir dizinin iç yapısını bilmeye gerek duymadan, başından sonunda kadar dolaşılabilmesi mümkündür. Dolayısıyla, koleksiyon veya dizi gibi bir nesne bütününün içerisindeki elemanlara erişilmesi ve dolaşılması noktasında devreye giren bir aktör olmalıdır. Aslında bu sorumluluk, bir **öteleme nesnesine(Iterator Object)** verilmiştir.

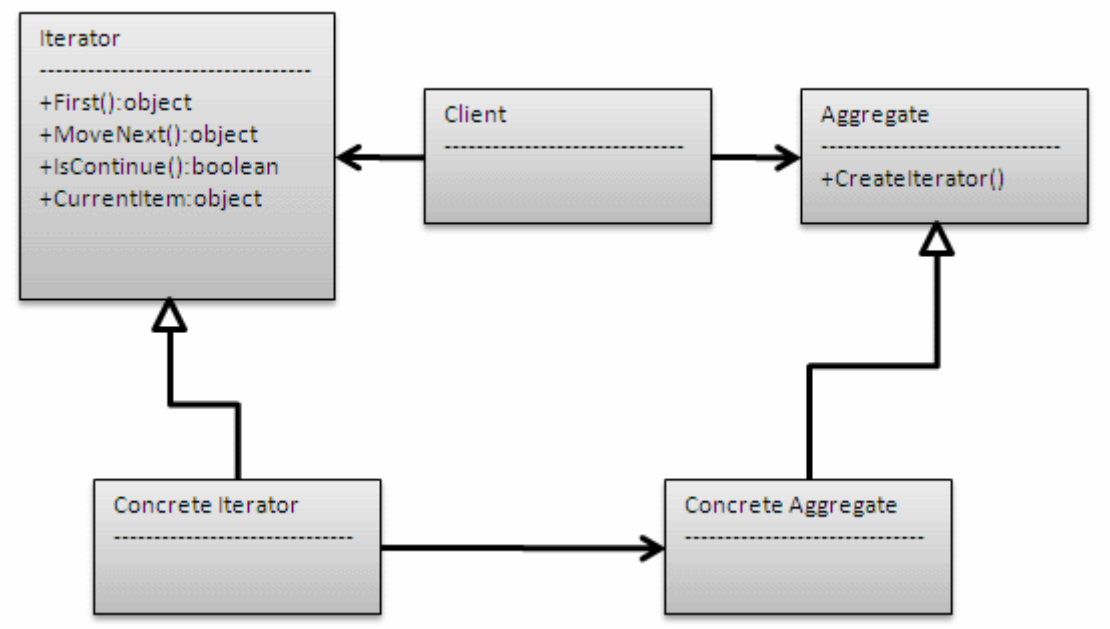
Bu açıdan bakıldığında nesne bütününün elemanlarına*(çoğunlukla koleksiyon veya dizi olarak düşünebiliriz)* erişilmesi, bu elemanların baştan sona dolaşılması, bir öteleme sırasında nerede kalındığının tutulması, hangi koşula göre devam edilmesi gerektiğinin bilinmesi, devam edilecek ise bir sonra gelen nesnenin döndürülmesi gibi sorumlulukları üstüne alan bir aktörden bahsetmekteyiz. Ki bu aktör aslında **generic programlamada** önemli bir yerede sahiptir. Nitekim, herhangibir nesne bütünün içinde dolaşılması için standart bir yol sunulması generic programlamanın gereksinimlerinden birisidir. Veri yapılarının ne kadar sık kullanıldığı düşünülünce doğal olarak ortaya, tasarımı kalıplaşmış bir uygulama biçimi çıkmaktadır. İşte bu yazımızın konusu, **Behavioral(Davranışsal)** kalıplardan olan **Iterator** tasarım deseni.

Tabiki programlama dillerine zaman içerisinde gelen bazı ek yetenekler sayesinde desenin uygulanış biçimi çok daha kolaylaşmıştır. özellikle C# tarafında, 2.0 versiyonu ile birlikte gelen **yield**anahtar kelimesinin kullanımı, C# 3.0 ile birlikte **LINQ(Language Integrated Query)** özelliklerinin gelmesi aslında nesnelerin elemanları üzerinde bir uçtan diğerine farklı filtrelemelere göre hareket edilmesini son derece kolaylaştırmaktadır. C# tarafında bu konu ile ilişkili baş aktör **IEnumerable arayüzüdür(Interface).** Kendisi doğal yollardan **Iterator** deseninin uygulanabilir olmasını sağlamaktadır. Biz bu yaklaşımları yazımızın sonlarında değerlendireceğiz. Şimdilik desenimizi kuralına uygun olaraktan geliştireceğiz. öncesinde iterator kalıbına örnek bir kaç senaryo üzerinde durmaya çalışalım.

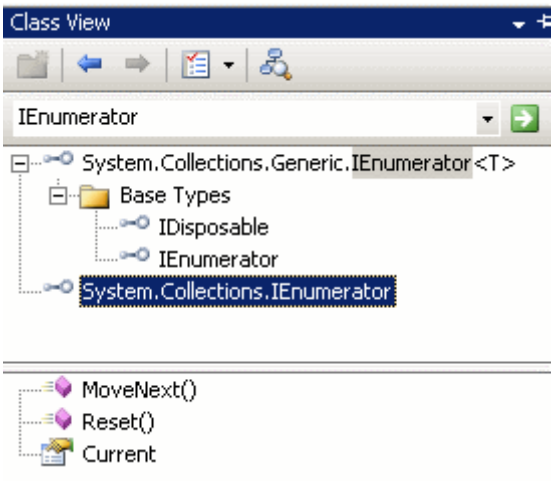
örneğin herhangibir bilgisayar sisteminde yer alan klasör yapısında bu desenin uygulanışını değerlendirebiliriz. Klasörler kendi içlerinde alt klasörler veya dosyalar içerir. Bunların ekrana beliri bir formatta listelenmesi sırasında o anki klasör ağacının tamamının bir uçtan diğerine dolaşılması gerekecektir. Ya da bir klasörün toplam boyutunun bulunması istendiğinde, alt klasör ve içlerindeki dosyaların boyutlarınında bir uçtan diğerine değerlendirilmesi gerekecektir. Buradaki klasör yapısı ve içeriği nesnel bazda düşünüldüğünde, çeşitli filtrelemelere göre değerlendirilebilmesinde sorumluluk, **Iterator** nesnesi tarafından üstlenilebilir. Unutulmaması gereken noktalardan birisi de, **Iterator** tasarım kalbında nesne bütünü içerisindeki elemanların nasıl

yapılandırıldıklarının bir öneminin olmayışıdır. Desenin amacı söz konusu nesne bütünü baştan sona dolaşabilmektir. Bir başka örnek olarak bir şirketin organizasyonel yapısını ifade eden bir nesne bütünü göz önüne alınabilir. Ağaç yapısı şeklinde ifade edilebilecek bu nesne bütünün içerisinde hareket edilebilmesi sırasında dalların dizilişleri, alt dallarda kimlerin olduğu, nasıl hareket edilmesi gerektiği gibi kriterlerin sorumlulukları Iterator nesnesine yüklenebilir.

Şimdi basit bir örnekten ilerleyerek desenimizi kavramaya çalışalım. Ama öncesinde UML şemamıza bakmakta yarar olduğu kanısındayım.

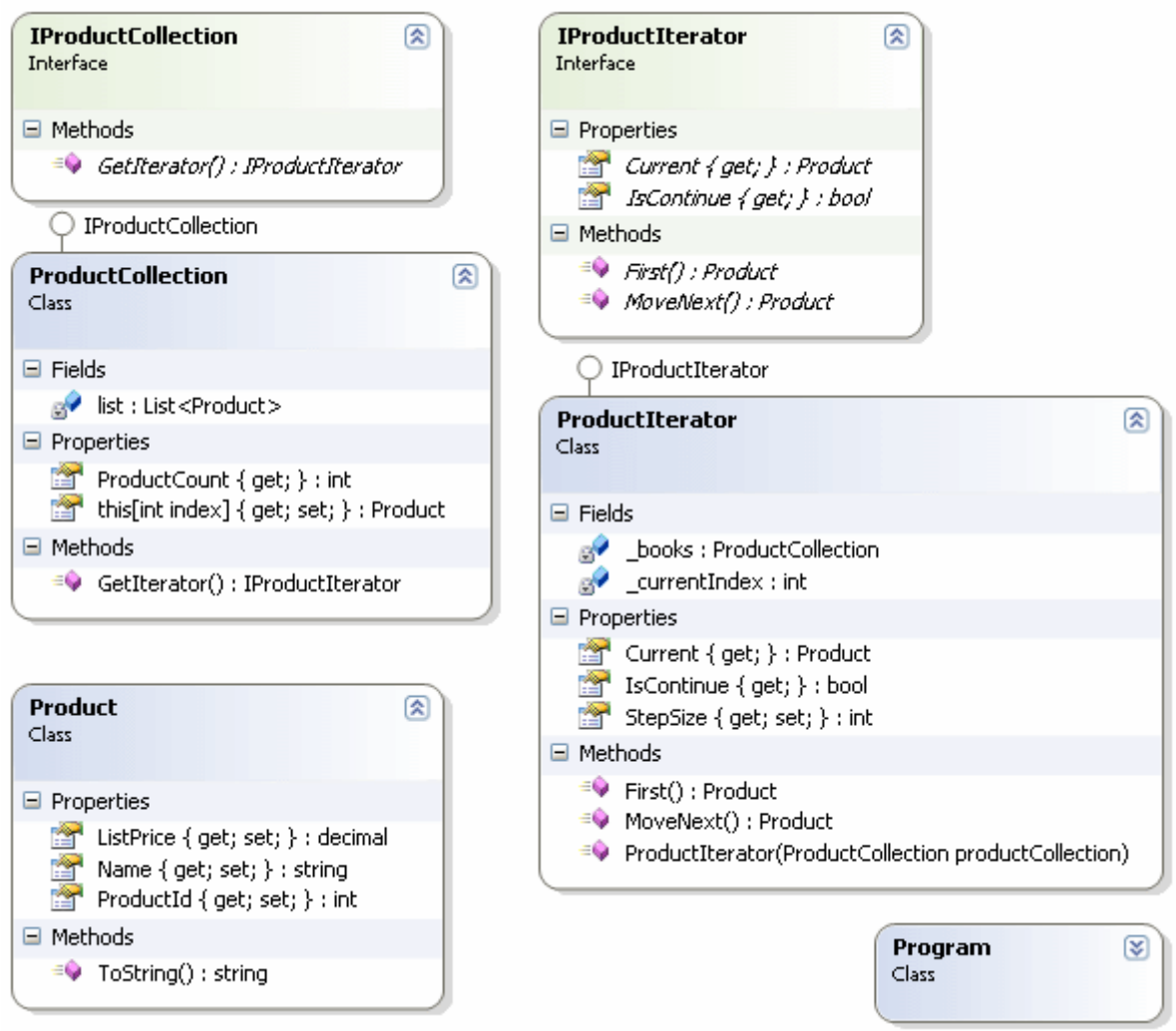


Şekildende görüldüğü üzere **Iterator** nesnesi, öteleme sorumlulukları için bir arayüz sunmakta ve nesne kümesi içerisindeki hareketlilik sırasında yapılması gereken bazı operasyonları bildirmektedir. O anki nesnenin kim olduğunun bilinmesi için **CurrentItem** gibi bir metod*(veya özellik-property olabilir)* kullanılmaktadır. Bütün içerisindeki bir sonraki elemana geçmek için **MoveNext** metodu, takip eden nesne olup olmadğını tespit etmek için **IsContinue** metodu kullanılabilir. Yada iterasyona başlarken ilk elemana gitmek için **First** operasyonu ele alınabilir. Tabiki bu metodlar tamamen semboliktir. Nitekim **IEnumerator** arayüzüde kendi içerisinde buna benzer metodları sunmaktadır.



UML şemamıza baktığımızda, istemciden nesne bütününün kendisine**(Aggregate object)** ve **Iterator** tipine doğru bir **Association** tanımlandığını görmekteyiz. Sonuç olarak istemci tarafı **Aggregate** nesnesini kullanmakta ve içerisinde dolaşmak için **Iterator** örneklerinden yararlanmaktadır. Benzer şekilde iterasyon sorumluluğunu yerine getiren nesnede**(Concrete Iterator)**çok doğal olarak **ConcreteAggregate** nesnesinin üyelerine erişmekte ve kullanmaktadır. Yani **ConcreteIterator'** dan **ConcreteAggregate'** e doğru bir **ilişki(Association)** mevcuttur.

Artık örneğimizi tasarlamaya başlayabiliriz. Kalıbın nasıl uygulandığını görmek istediğimizden çok basit bir senaryo üzerinden gideceğiz. Senaryomuzda **Product** tipinden nesne örneklerini barındıran bir nesne bütünümüz olduğunu göz önüne alacağız. Buna göre Iterator tasarım kalbının kullanaraktan, ürünleri dolaşabilmek için bir Iterator nesnesinin nasıl geliştirilebileceğini ele alacağız. Sınıf diagramımız,



şeklinde olup kodlarımızda aşağıdaki gibidir.

```
using System;
using System.Collections;
using System.Collections.Generic;

namespace IteratorPattern
{
   // Item
   class Product
   {
      public int ProductId { get; set; }
      public string Name { get; set; }
      public decimal ListPrice { get; set; }

      public override string ToString()
      {
         return String.Format("{0} {1} {2}", ProductId.ToString(), Name,
ListPrice.ToString("C2"));
```

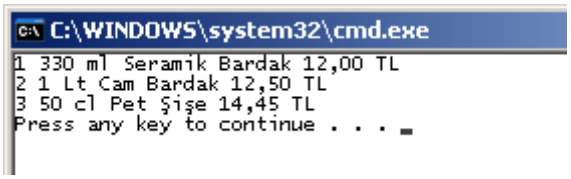
```
    }
}

// Iterator
// Nesne bütünü içerisindeki hareketlerin, yönlenmelerin gerçekleştirilebilmesi için gerekli operasyon arayüzünü tanımlar.
interface IProductIterator
{
    Product First();
    Product MoveNext();
    bool IsContinue { get; }
    Product Current { get; }
}

// Aggregate
// Nesne bütününün, iterasyon için Concrete Iterator tipinden nesne örneği döndürecek bir metodunun olmasını söyler.
interface IProductCollection
{
    IProductIterator GetIterator();
}

// Concrete Aggregate
// Nesne kümesini barındıran tipimiz.
class ProductCollection
    : IProductCollection
{
    // Product topluluğunu saklamak için generic bir List<T> koleksiyonundan yardım alıyoruz.
    private List<Product> list = new List<Product>();

    // ürün sayısını dışarıya vermek için kullanılan bir özellik
    public int ProductCount
    {
        get { return list.Count; }
    }

    // Eleman eklemek ve okumak için kullanılan bir Indeksleyici
    public Product this[int index]
    {
        get { return list[index]; }
        set { list.Add(value); }
    }

    #region IProductCollection Members
```

```
        // Iterator nesnesini örnekler
        public IProductIterator GetIterator()
        {
            // Iterator nesnesi örneklenirken parametre olarak o andaki ProductCollection nesne örneği referans olarak gönderilir.
            // Bu sayede ProductIterator isimli Concrete Iterator nesne örneği, çalışma zamanında hangi nesne bütününü dolaşacağını bilecektir.
            return new ProductIterator(this);
        }

        #endregion
    }

    // Concrete Iterator
    // Nesne bütününün bir ucundan diğerine hareket edilebilmesine olanak sağlayacak fonksiyonellikleri uygulayan asıl Iterator tipi
    class ProductIterator
        : IProductIterator
    {
        // Iterator nesne örneğinin, çalışma zamanında hangi nesne bütününü dolaşacağını bilmesi gerekmektedir.
        private ProductCollection _books;
        private int _currentIndex = 0;
        // İstemci isterse adım sayısını değiştirebilir. örneğin ikişer ikişer atlanarak gidilmesi sağlanabilir,
        public int StepSize { get; set; }

        // bu nedenle yapıcı metoda parametre olarak, ProductCollection(Concrete Aggregate) nesne örneğinin referansı gelir. Bu referansın GetIterator metodu içerisindeki çağrı ile gönderildiğini hatırlayalım.
        public ProductIterator(ProductCollection productCollection)
        {
            _books = productCollection;
        }
        #region IProductIterator Members

        // İlk elemana gidilmesini sağlayan metod
        public Product First()
        {
            // Nerede olunduğunun takibi için _currentIndex değeri set edilir
            _currentIndex = 0;
            return _books[0];
        }
```

```
        // Bir sonraki elemana geçilmesini sağlayan metod
        public Product MoveNext()
        {
            // Nerede olunduğunun takibi için _currentIndex değeri set edilir. Adım sayısı kadar
arttırılır.
            _currentIndex += StepSize;
            if (IsContinue) // Eğer takip eden bir eleman var ise geri döndürülür
                return _books[_currentIndex];
            else
                return null;
        }

        // Takip eden ürün olup olmadığını belirten read-only özellik
        public bool IsContinue
        {
            get { return _currentIndex < _books.ProductCount; }
        }

        // O anki elemanı döndüren read-only özellik
        public Product Current
        {
            get { return _books[_currentIndex]; }
        }

        #endregion
    }

    class Program
    {
        static void Main(string[] args)
        {
            ProductCollection products = new ProductCollection();

            products[0]=new Product{ ProductId=1, Name="330 ml Seramik Bardak",
ListPrice=12M};
            products[1] = new Product { ProductId = 2, Name = "1 Lt Cam Bardak", ListPrice
= 12.5M };
            products[2] = new Product { ProductId = 3, Name = "50 cl Pet Şişe", ListPrice =
14.45M };

            // Iterator nesnesi products isimli koleksiyonu kullanmak üzere oluşturulur
            ProductIterator iterator = new ProductIterator(products);

            // Adım sayısı belirlenir
            iterator.StepSize = 1;
```

```
            // First ile ilk elemana konumlanılır.
            // Koşul olarak IsContinue değerine bakılır
            // İlerleme için MoveNext metodu kullanılır.
            for (
               Product product = iterator.First()
                  ; iterator.IsContinue
                  ; product = iterator.MoveNext()
                  )
            {
               Console.WriteLine(product.ToString());
            }
         }
      }
}
```

örneğimizi çalıştırdığımızda aşağıdaki ekran görüntüsünde yer alan sonuçları elde ederiz.



Tabiki amacımız sadece kalıbın nasıl uygulandığını öğrenmek olduğundan, işimizi kolaylaştırması için aslında içeride generic bir **List<T>** koleksiyonundan yararlandık. Ama tabiki var olan koleksiyonlar ile ifade edilemeyecek bir nesne bütünü olduğunda*(özel bir ağaç yapısı olabilir)* daha farklı bir depolama modeli kullanmamız gerekebilir. **DoFactory.com**sitesinin istatistiklerine göre neredeyse kullanılmadığı görülmemiş bir tasarım deseni ile karşı karşıyayız aslında. Peki, aramızdan kaç geliştirici C# veya Vb.Net tarafında bu deseni isteyerek ve bilinçli olarak kullandı.

Ne kadar ilginç değil mi? Kodlama sırasında bir koleksiyon üzerinde dolaşırken tek yapmamız gereken çoğunlukla bir döngüyü kullanmaktır*(for, foreach, while vb...)*. Hatta basit bir**LINQ** sorgusu sonrası filtrelenmiş bir içeriği bile for, while gibi döngüler ile dolaşmamız söz konusudur. Ama hiç arka planda bu sorumluluğu alan bir **Iterator** nesnesi olduğunu ve bir kalıp uygulandığını düşünmeyiz. Şimdi bu moral bozukluğu ile aslında işleri nasıl kolaylaştırmış olduğumuza bir bakalım Yukarıda geliştirdiğimiz örneğin benzeri ile devam ediyor olacağız. İşte **IEnumerable** arayüzü ve **yield** anahtar kelimesi...

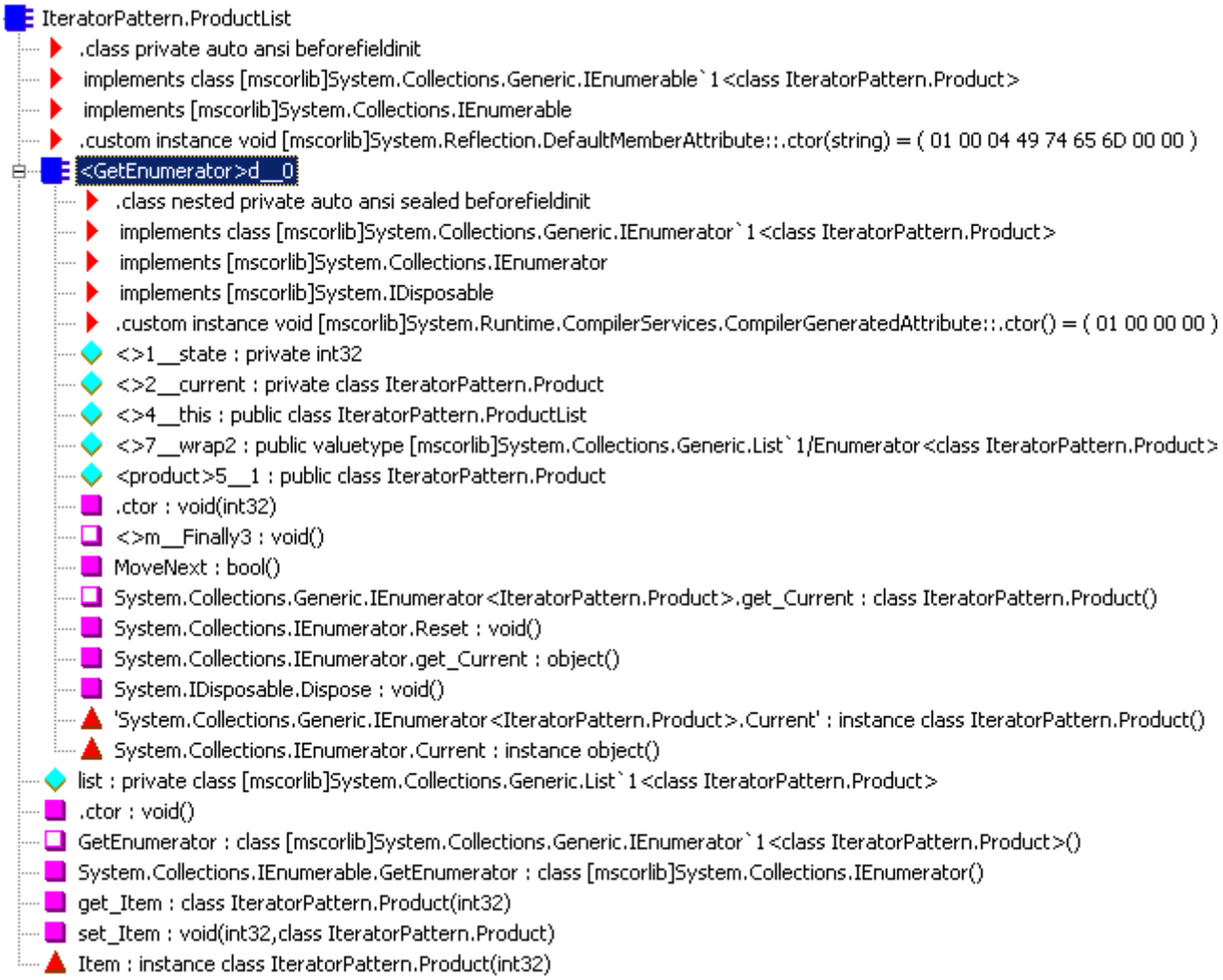
```
class ProductList
   : IEnumerable<Product>
   {
      private List<Product> list = new List<Product>();

      public Product this[int index]
      {
```

```
        get { return list[index]; }
        set { list.Add(value); }
    }

    #region IEnumerable<Product> Members

    public IEnumerator<Product> GetEnumerator()
    {
        foreach (Product product in list)
        {
            yield return product;
        }
    }

    #endregion

    #region IEnumerable Members

    IEnumerator IEnumerable.GetEnumerator()
    {
        throw new NotImplementedException();
    }

    #endregion
}
```

Dikkat edileceği üzere .Net içerisinde yer alan ve nesnelere iterasyon öğreten **IEnumerable<T>** arayüzünü **yield** ile birlikte ele alarak, **ProductList** nesne örnekleri içerisinde dolaşılabilmesini sağlayacak geliştirmeyi kolayca yapmış olduk.*(Tabi bir versiyon daha geriye gidebilir ve IEnumerator arayüzünü kullanaraktanda bu işlemleri gerçekleştirebiliriz, bunuda hatırlatayalım)* örnek kullanımı ise şu şekilde gerçekleştirebiliriz;

```
ProductList products2 = new ProductList();
products2[0] = new Product { ProductId = 1, Name = "330 ml Seramik Bardak", ListPrice = 12M };
products2[1] = new Product { ProductId = 2, Name = "1 Lt Cam Bardak", ListPrice = 12.5M };
products2[2] = new Product { ProductId = 3, Name = "50 cl Pet Şişe", ListPrice = 14.45M };

foreach (Product product in products2)
{
    Console.WriteLine(product.ToString());
}
```

Uzun uzun zaman önce, **C# 2.0** ile birlikte gelen yenilikleri anlatırken **yield** anahtar kelimesinide içeriklere kattığımı gayet net hatırlıyorum. Aslında bu anahtar kelimenin, var olan **Iterator** deseninin uygulanmasını dahada kolaylaştırdığı gün gibi ortada. Bir başka deyişle Iterator deseninin aslında .Net içerisine gömülü olduğunu söyleyebiliriz. Tabi kalıbı bizzat uygulamamış olsakta aslında **IL(Intermediate Language)** tarafındaki kodlara bakıldığında **Iterator** tasarım kalıbının izlerini görmemiz mümkündür.



Böylece geldik bir tasarım kalıbının daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

IteratorPattern.rar (30,73 kb)

Service Orientation vs Object Orientation (2009-07-30T23:11:00)

soa,

Merhaba Arkadaşlar,

Geçtiğimiz günlerde çok ilginç bir rüya daha gördüm. Ne zamandır paylaşmayı planlıyordum. yolculuğu ile **Japonya'** nın başkenti **Tokyo'** ya gidiyordum. **Antartika** üzerinden önce **Hawai'** y sonra bir anda kendimi **Tokyo'** da buldum. E rüya tabi...Bir süre Japonya' ya vizesiz gidilebildiği

Hala böylemidir bilemiyorum ama vardığımda beni sorguya çektiler. Girebilmemin tek şartı **Service** yaklaşımı ile **Object** yaklaşımı arasındaki farklardan on tanesini söyleyebilmemdi.**"Demek öyle"** dedim. **Samuray** kılıcımı çektim ve onlara şunları söyledim.

Service Orientation için Söylediklerim
Farklı sistemlerin/platformların ayrı çalışma ortamları içerisinde olması*(Heterojenlik)*...
Başka sistemler ile yapılacak haberleşmeleri ortak bir çerçevede geliştirebilmek için gereklilikler zorunluluğu*(Interoperability)*...
Asenkron erişimin daha zor sağlandığı, uzun süreli süreçlerin*(Long Running Processes)*yer aldığı edilebilirliğin azaldığı bir işleyiş...
özerk olmaları ve bu nedenle güvenlik*(Security)* ve hatalar*(Failure)* ile ilişkili izolasyonun şart ol
Şemaların*(Schema)* ve mesajlaşmanın önemli olması...
Güvenliği sağlamak için daha çok kriterin düşünülmek zorunda olması*(Ağı kimler dinler, neler y bilinmez)* ve bunun doğal sonucu paranoyaklık*(%99 güvenli bir sisteme ulaşma zorluğu)*...
Geliştirme maliyeti daha yüksek...
Sistem yaşamını devam ettirirken dağıtımların*(Deployment)* bağımsız olarak yapılabilmesi...
Farklı uygulama alanlarındaki*(Application Domain)* bileşenleri konuşturmak...
Performansı etkileyebilecek daha çok faktör olması...

Başarmıştım. Farkettiğim farkları, bilinen farkları söylemiştim. Peki ne oldu? Tokyo' ya girebildim mi? Hayır...çalar saatin sesi ile uyanıp işe koyuluverdim.

Tasarım Desenleri - Mediator (2009-07-28T21:04:00)

design patterns,oop,c#,

Merhaba Arkadaşlar,

Yandaki resimde Zurich hava alanına ait bir görüntü yer almaktadır. Hava alanının ne kadar karm yazımıza konu olarak Londra' daki Heathrow hava alanını dahil edecektim. Nitekim uzun zaman izlediğim bir belgeselde, bir iniş ve birde kalkış pistiyle bu kadar işlek bir havalimanının ne kada Ancak yaptığım araştırmalar sonrası dünyadaki en iyi hava alanları arasında olmadığını gördüm.(http://www.worldairportawards.com/). Her neyse.

Konumuz aslında kimin daha iyi olduğu değil ama tüm hava alanları için ortak olan bir sorun. İnip kalkan ve hatta aynı havasahasına giren uçakların koordine edilmesi. Hiç çok işlek hava alanlarında kontrol kulesi olmadığını hayal ettiniz mi? Sanıyorumki aşağıdaki konuşmalar ile karşılaşabilirdik.

- **AzonAir - 110 :** Ben sağdaki piste inmek üzere alçalıyorum arkadaşlar.

- **CargoL TL 101 :** Hayır hayır oraya ben inecektim.

- **AzonAir - 110 :** Eeee...önce gelen kapar.

- **öz Hawai - 444 :** Savulunnnn!!! Ben o pistten kalkış yapıyorum.

- **CargoL TL 101 :** Hangi pist, hangi pist? ...

- Cazırt cuzurt, kraşş bummm...

Abartamaya gerek yok tabiki ama bu anektodunda bir manası var. Bir kontrol kulesi temel olarak tüm iniş kalkışları düzenler ve bu işi yaparken yukarıdaki gibi, uçakların birbirleri ile konuşmasına gerek kalmaz. Bir başka deyişle birbirleriyle etkileşimde olan uçakların tüm iletişimi, kontrol kulesi içerisinde hesaplanır ve işletilir. Dahada açık bir ifade ile kontrol kulesi aslında **Mediator** nesnesinin kendisidir. Mediator??? Hımmm..

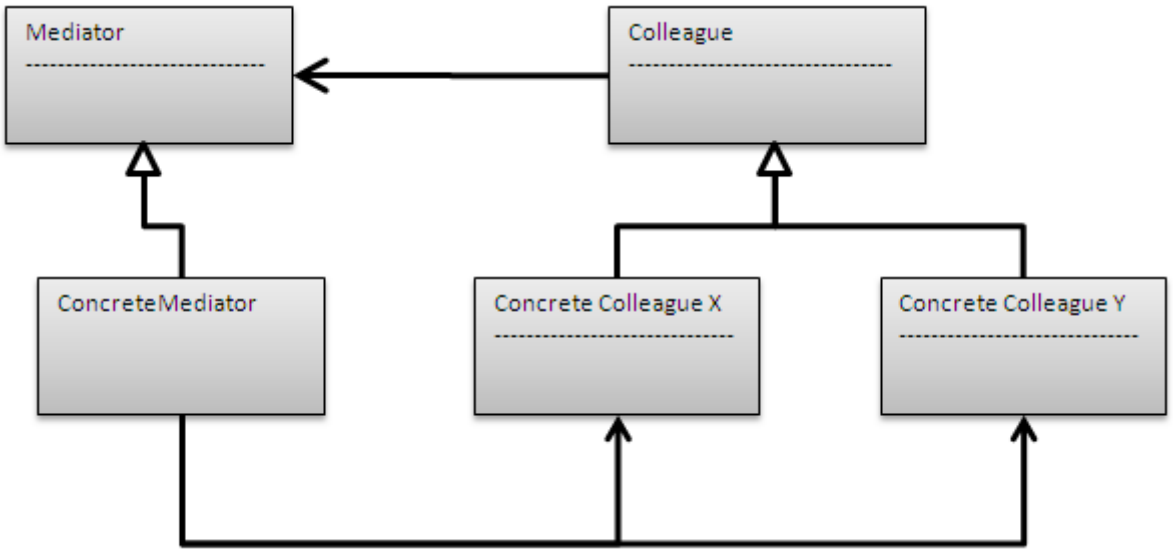
Pekala konuyu biraz daha örneklemeye çalışalım.

Bu kez bir network ağındaki kullanıcıları ve grupları göz önüne alalım. Kullanıcıların(Users) birden fazla gruba dahil olması muhtemeldir. Benzer şekilde bir grupta kendi içerisinde birden fazla kullanıcı barındırabilir. Yani kullanıcı ve gruplar arasında **çoğa çok(Many to many)** ilişki söz konusudur. Bu aktörler aslında birer nesne(Object) olarak düşünüldüklerinde, birbirlerine **sıkı sıkıya bağlı olmaları(Tghtly Coupling),** yönetimlerini zorlaştırmakla kalmaz, ileride yapılacak olan genişletmelerin çok

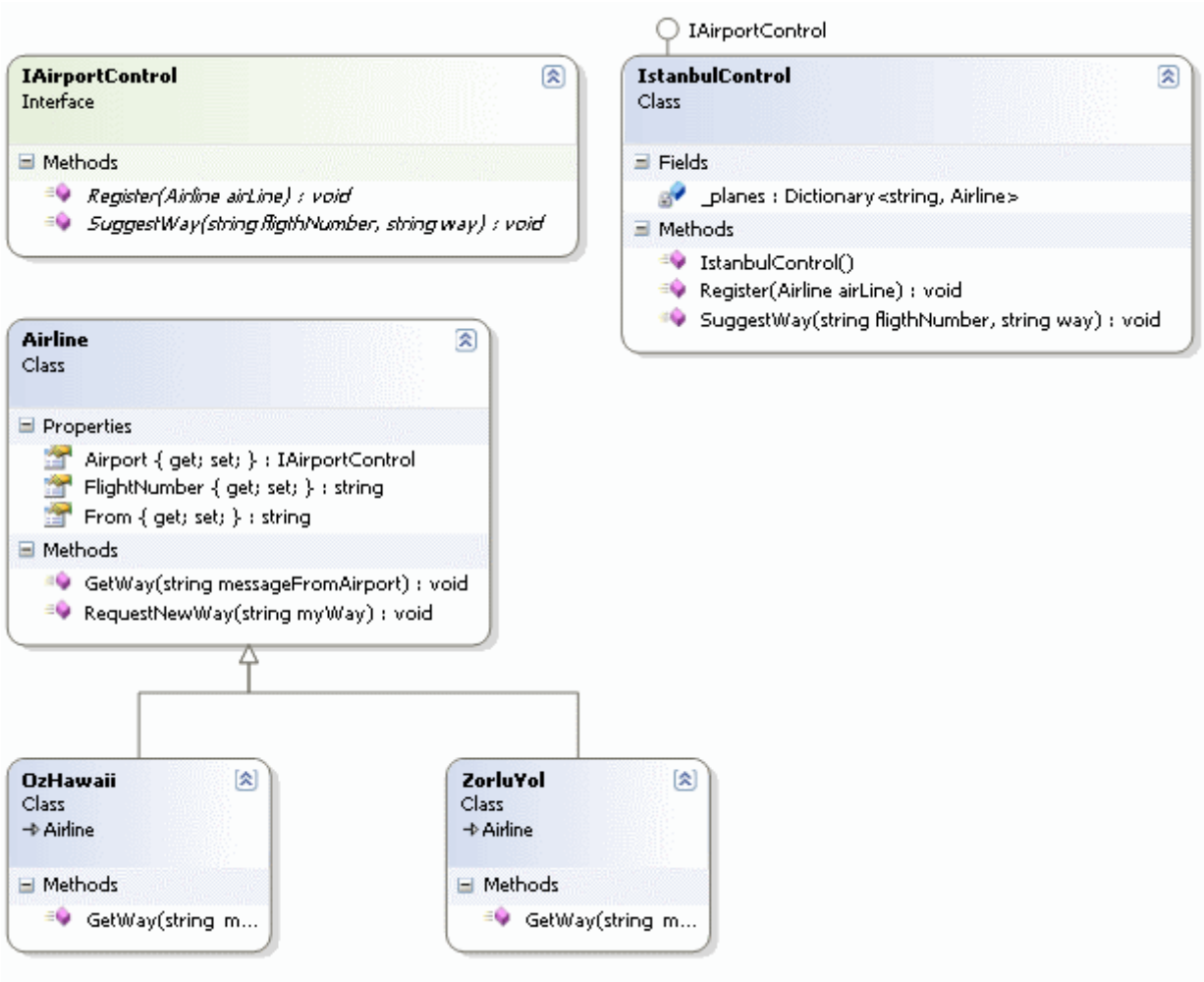
fazla nesneyi etkilemesinede neden olur. Dolayısıyla aralarındaki bağı **zayıflaştırmak(Loose Coupling)** gerekir. Bu noktada veritabanı tasarımı ile uğraşanlar için sorunu çözmek son derece kolaydır. Nitekim bir ara tablo yardımıyla çoğa çok ilişkinin tesisi kolayca sağlanabilir. Diğer yandan **Nesne Yönelimli(Object Oriented)** tarafta, kullanıcı ve gruplar arasındaki iletişimi, onlardan soyutlayarak kendi içerisinde yönetecek olan bir ara nesneye ihtiyaç vardır. Kim...**Mediator**.

Anlaşılacağı üzere konumuz **Behavioral** tasarım kalıplarından olan **Mediator** desenidir. Bu desenin kullanım amacındaki odak noktası, nesne kümelerinin birbirleriyle nasıl haberleşebileceğini soyutlayan bir ara nesnenin kullanılmasıdır. Yazılım dünyasında bu konuya ilişkin verilebilecek en güzel örneklerden biriside **Chat** uygulamalarıdır. Bir **Chat**uygulamasına dahil olan katılımcıların her birinin birbirleriyle iletişim kurarak konuşması, zaman içerisinde ağ yükünü arttıracak ve yönetilemez hale getirecektir. üstelik performans açısından da son derece kötü bir yaklaşımdır. Bunun yerine katılımcılar arasındaki iletişimin yönetimini sağlayacak bir merkezin olması önerilir. Böylece, katılımcılar hiç bir şekilde birbirlerinin nesnelerine istemeden müdahele edemez veya karmaşık hesaplamalar, karar mekanizmaları ile karşı karşıya kalamazlar. Katılımcılar isteklerini **Mediator** nesneye iletirler ve sonuçlardan haberdar edilirler. Chat uygulaması göz önüne alındığında Mediator aslında mesajlaşma işlemini üstlenen sunucu uygulama olarak düşünülebilir.

Gelin UML şemamıza bakalım ve arkasından geliştireceğimiz basit bir örnek yardımıyla konuyu irdelemeye çalışalım.



Şemadanda görüldüğü üzere Colleague' den Mediator'a doğru ve Concrete Mediator' dan, Concrete Colleague nesnelerine doğru **tek yönlü ilişkiler(Asscoiation)** söz konusdur. Yani okun solunda yer alan nesneler, okun ucundaki nesneleri ve izin verilen üyelerini kullanmaktadır. UML şemamızda yer alan nesnelerin ne işe yaradıklarını daha kolay kavrayabilmek amacıyla bir örnek üzerinden ilerleyebiliz. İşte sınıf diagramı ve kodlarımız.




```
using System;
using System.Collections.Generic;
using System.Threading;

namespace MediatorPattern
{
   // Mediator
   interface IAirportControl
   {
      void Register(Airline airLine);
      void SuggestWay(string fligthNumber, string way);
   }

   // Concrete Mediator
   class IstanbulControl
      :IAirportControl
   {
      // Concrete Colleague nesne örnekleri bu koleksiyonda depolanmaktadır.
      private Dictionary<string, Airline> _planes;
```

```
    public IstanbulControl()
    {
        _planes = new Dictionary<string, Airline>();
    }

    #region IAirportControl Members

    // Kontrol kulesine çevredeki uçakların kayıt olması için Register metodu kullanılır. Bu metod parametre olarak Colleague' den türeyen her hangibir Concrete Colleague nesne örnepğini alabilir.
    public void Register(Airline airLine)
    {
        if (!_planes.ContainsValue(airLine))
            _planes[airLine.FlightNumber] = airLine;

        // Hava yolu şirketine ait uçağın, kuleden yeni rota talep edebilmesi için, Concrete Colleague nesne örneğinin, Mediator referansının bildirilmesi gerekir.
        airLine.Airport = this;
    }

    // Concrete Colleague nesne örneklerinin yeni rota talep ederken kullandıkları metod. Bu metod o anki koşullar gereği sakladığı diğer uçakların konum bilgilerinden yararlanıp bir takım sonuçlara varmaktadır. Bu sayede n tane kombinasyonun, her bir uçak tarafından ele alınması yerine, tüm bu kombinasyonlar daha az sayıya indirgenerek Mediator içerisinde değerlendirilebilmektedir.
    public void SuggestWay(string fligthNumber, string way)
    {
        // TODO: Diğer uçakların konumlarına bakılarak flightNumber için yeni bir rota önerilir. Gerekirse diğer uçaklarada farklı rotalar önerilebilir.

        // Sembolik olarak yeni bir rota belirleniyor. Bilgilendirme rotayı talep eden Concrete Colleague nesne örneğinin GetWay metoduna yapılan çağrı ile gerçekleştiriliyor.
        Thread.Sleep(250);
        Random rnd = new Random();
        _planes[fligthNumber].GetWay(String.Format("{0}:{1}E;{2}:{3}W", rnd.Next(1, 100).ToString(), rnd.Next(1, 100).ToString(), rnd.Next(1, 100).ToString(), rnd.Next(1, 100).ToString()));
    }

    #endregion
}

// Colleague
abstract class Airline
{
```

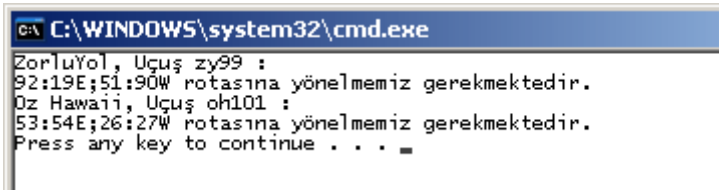
```
    public IAirportControl Airport { get; set; }
    public string FlightNumber { get; set; }
    public string From { get; set; }

    // Mediator' den yani kuleden yeni bir rota talep ederken kullanılan metod.
    public void RequestNewWay(string myWay)
    {
        // çağrı dikkat edileceği üzere Mediator tipine ait nesne referansına doğru yapılmaktadır. Peki bu referansı nerede verdik. Bknz Register metodu. :)
        Airport.SuggestWay(FlightNumber, myWay);
    }

    // Mediator tipinin, çağırıda bulunacağı GetWay metodu. Bu metodun parametre içeriği, kuleden(Concrete Mediator) üzerinden gelmektedir.
    public virtual void GetWay(string messageFromAirport)
    {
        Console.WriteLine("{0} rotasına yönelmemiz gerekmektedir.", messageFromAirport);
    }
}

// Concrete Colleague
class OzHawaii
    :Airline
{
    public override void GetWay(string messageFromAirport)
    {
        Console.WriteLine("Oz Hawaii, Uçuş {0} : ",FlightNumber);
        base.GetWay(messageFromAirport);
    }
}

// Concrete Colleague
class ZorluYol
    : Airline
{
    public override void GetWay(string messageFromAirport)
    {
        Console.WriteLine("ZorluYol, Uçuş {0} : ", FlightNumber);
        base.GetWay(messageFromAirport);
    }
}

class Program
{
```

```
        static void Main(string[] args)
        {
            // Kule nesnesi örneklenir(Concrete Mediator)
            IstanbulControl istanbulKule = new IstanbulControl();

            // Kuleden hizmet alacak tüm uçakların kendisini kuleye bildirmesi gerekmektedir.
Bu nedenle uçaklar örneklendikten sonra Concrete Mediator tipine Register metedo
yardımıyla kayıt olurlar.
            OzHawaii oh101 = new OzHawaii { Airport = istanbulKule, FlightNumber =
"oh101", From="Hawai" };
            istanbulKule.Register(oh101);
            OzHawaii oh132 = new OzHawaii { Airport = istanbulKule, FlightNumber =
"oh132", From="Roma" };
            istanbulKule.Register(oh132);
            ZorluYol zy99 = new ZorluYol { Airport = istanbulKule, FlightNumber = "zy99",
From = "Antarktika" };
            istanbulKule.Register(zy99);

            // Uçaklar yeni rotalarını talep ederler.
            zy99.RequestNewWay("34:43E;41:41W");

            oh101.RequestNewWay("34:43E;41:41W");
        }
    }
}
```

Her ne kadar bir chat uygulaması yapmış olmasakta*(ki dofactory.com' da Chat uygulaması örneğini bulabilirsiniz.)* örneğimizdeki temel amacımız konumuza giriş yaptığımız kontrol kulesi senaryosunu simule edebilmektir. İlk etapta şunu rahatlıkla itiraf edebiliriz ki, Mediator deseni aslında uygulanması zor kalıplardan birisidir. 🤔

örneğimizde **meslektaş(Colleague)** nesnelerimiz belirli hava yollarını işaret etmektedir. örnek olarak **OzHawaii** ve **ZorluYol** isimli sınıflar, **Concrete Colleague** sınıflarımızdır. Bu havayolu şirketlerine ait uçakların inişleri ve kalkışları için yeni rotaları bulmak amacıyla birbirleri ile konuşmaları yerine bu işi **Mediator** nesnesi içerisinde yer alan basit bir fonksiyon üstlenmektedir. örneğe göre hava yolu şirketleri kuleden, yaklaşma halindeyeken veya kalkıştan önce, yeni rota talebinde bulunabilirler. Bunun için **Colleague** tipi olan **AirLine** içerisindeki**RequestNewWay** metodu kullanılır. Bu metod ise aslında, **Concrete Mediator** tipi içerisinde yer alan **SuggestWay** isimli bir foksiyonu çağırmaktadır. Dikkat edileceği üzere bu metod içerisinde, **Mediator** nesne örneğine abone olan tüm hava yolu şirketi uçaklarının o anki rota, yükseklik ve diğer bilgilerinden yararlanılarak, parametre olarak gelen **Concrete Colleague** nesne örneğine bilgilendirme yapılmaktadır. Bu bilgilendirmenin yapılabilmesi için tahmin edileceği üzere, **Concrete Mediator** tipinin, **Concrete Colleague** tiplerine erişebiliyor olması gerekmektedir. Bu sayede, **Concrete Colleague** tipleri

içerisindeki **GetWay** metodları, **Mediator** nesne örneği içerisinden çağrılabilir. Bu da zaten **UML** şemasında yer alan, **Concrete Mediator'** den, **ConcreteColleague** nesnelerine olan **tek yönlü ilişkiyi(Association)** açıklamaktadır. çok doğal olarak **Concrete Mediator** tipinin, hangi nesne kümelerini değerlendireceğini bilmesi gerekmektedir. Bu amaçla örneğimizde generic bir **Dictionary<T,K>** koleksiyonundan yararlanılmaktadır. Peki, **Concrete Colleague** nesne örnekleri, **Concrete Mediator** nesne örneklerine nasıl bildirilecektir? İşte bu noktada **Register** isimli metod devreye girmektedir. Buda UML şemamızda, **Colleague'** den **Medaitor'** e doğru olan **tek yönlü ilişkiyi(Association)** açıklamaktadır. Nitekim **Register** metodu parametre olarak **AirPort tipinden(Concrete Colleague)** nesne örnekleri almaktadır. örneğimizi çalıştırdığımızda aşağıdaki ekran görüntüsündekine benzer sonuçları elde ederiz.



özet olarak herhangibir havayoluna ait bir uçak, İstanbul kulesine yaklaştığında kendisine yeni bir rota talep ederken diğer uçaklar ile haberleşmek ve onların konumlarına göre hesaplamalar yaparak bir yön tayin etmek zorunda değildir. Tüm uçaklar bir birlerinden ayrıştırılmış ve yönlerini belirlemek üzere kullanılması gereken algoritmalar **Mediator** tipi içerisine kapsüllenmiştir. Biraz karışık bir desen implemantasyonu olmasına rağmen faydalı olduğunu umuyorum. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

MediatorPattern.rar (24,85 kb)

Tasarım Desenleri - FlyWeight (2009-07-27T18:30:00)

design patterns,oop,c#,

Merhaba Arkadaşlar,

Yandaki resimde yer alan minik boksör aslında hafif siklette mücadele etmek alakası bulunmamaktadır Ancak işleyeceğimiz tasarım kalbına bu ismin ve

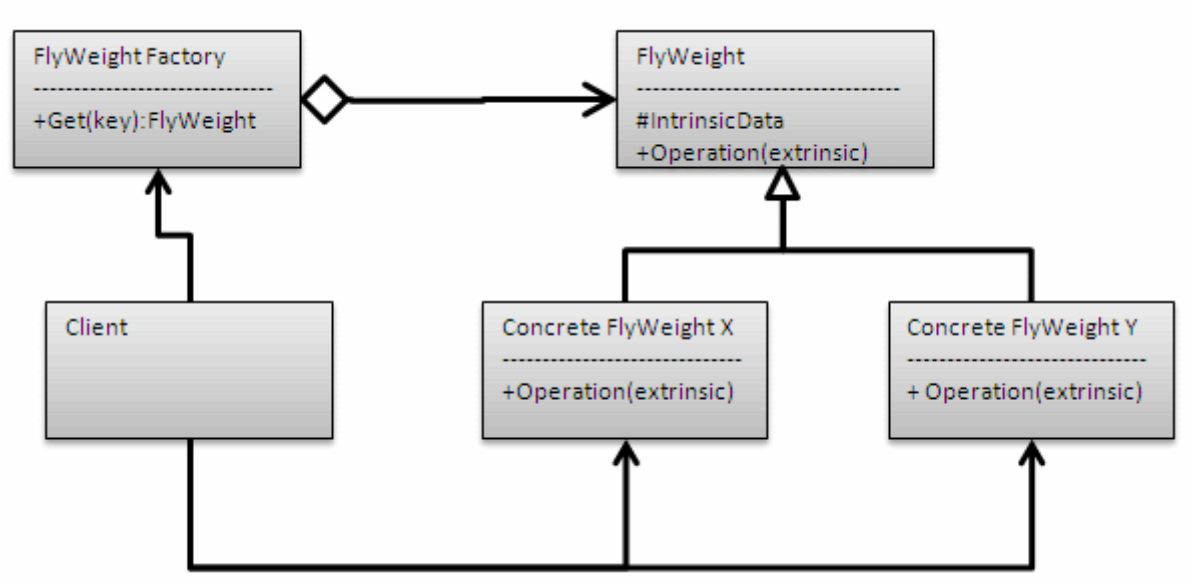
Yapısal(Structural) tasarım kalıplarından olan **FlyWeight**, bellek tüketimini optimize etmek amacıyla kullanılan bir desendir. Aslında detayına inildiğinde son derece zekice tasarlanmış ve pek çok noktada karşımıza çıkabilecek havuz mantığını içeren bir kalıp olduğu anlaşılabilir. Burada önemli olan nokta, bellek tüketiminin çok fazla sayıda nesnenin bir arada ele alınması sırasında ortaya çıkmasıdır. Buna göre söz konusu nesnelerin ortak olan, paylaşılabilen içerikleri ve bunların dışında kendilerine has durumları olduğu takdirde, nesne üretimlerini sürekli tekrar ettirmektense basit bir havuz içerisinden

tedarik ettirmek, uygulamanın harcadığı bellek alanlarının optimize edilmesi için yeterli olacaktır. Bu açıdan bakıldığında desenin, paylaşımlı nesneleri efektif olarak kullanabilmek üzerine odaklandığını söyleyebiliriz.

Aslında, **FlyWeight** tasarım kalıbını hangi kaynaktan araştırırsak araştıralım ilk etapta dikkat edilmeyen çok önemli bir özellik içermektedir. Her bir **FlyWeight** nesnesi temel olarak iki önemli veri kümesinden oluşur. Kaynaklarda çoğunlukla **intrinsic** olarak geçen **durum-bağımsız(State-Independent)** kısım parçalardan birisir. Bu kısımda, çalışma zamanındaki tüm **FlyWeight** nesneleri tarafından saklanan paylaşılmış alanlar yer almaktadır. Diğer parça ise **durum-bağımlı(State-Dependent)** olarak bilinen ve kaynaklarda çoğunlukla **extrinsic**olarak belirtilen kısımdır. Bu kümedeki veriler ise istemci tarafından saklanır, hesap edilir ve **FlyWeight** nesne örneğine, yine FlyWeight' in bir operasyonu yardımıyla aktarılırlar. Tabiki desenin kullanım amacını daha net bir şekilde kavrayabilmek için bazı örnek senaryolar üzerinden gitmeye çalışabiliriz.

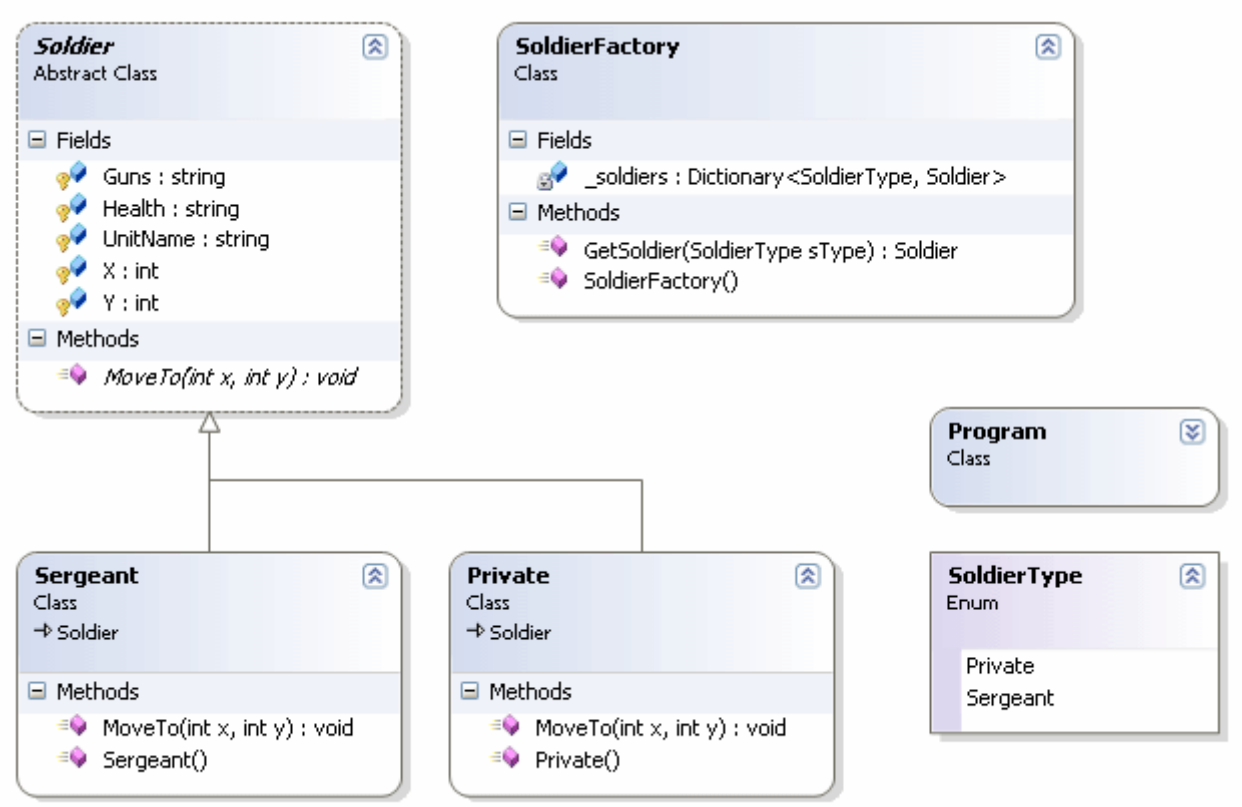
Bu desen ile ilişkili en belirgin örnek kelime işlemcilerinde ortaya çıkmaktadır. Nesneye dayalı olarak geliştirilen bir kelime işlemcisinde her bir karakterin nesne olarak oluşturulduğunu düşünelim. Her bir karakterin döküman içerisinde çok fazla sayıda kullanılabileceği ortadadır. Dolayısıyla aynı ortak özelliklere sahip olan bir çok karakter nesne örneğinin yönetimi söz konusudur ve buda doğal olarak bellek üzerinde daha fazla yer harcanmasına neden olacaktır. Aynı şekilde, bu nesnelerin tekrardan oluşturulmalarının maliyetide doğrudan performansa yansıyacaktır. Oysaki bu karakter nesnelerinin pek çoğu için ortak olan bir takım veriler söz konusudur. örneğin, karakterlerin boyutları, font tipleri, büyüklükleri vb...Bunların dışında kelime işlemci açısından önem arz eden ve karakter nesneleri tarafından ortak olarak düşünülemeyecek bir takım verilerde vardır. Söz gelimi karakterlerin lokasyonu aslında istemci tarafından*(yani kelime işlemci uygulama)* belirlenebilir. Dolayısıyla karakterleri temsil eden tipler aslında ortak özellikleri bir yerde toplanıp, örnekleride havuzlanarak hafifleştirilebilir. Hafifleştirmek aslında bu desene neden **FlyWeight** adının verildiğinide ortaya koymaktadır.

Biz örnek uygulamamızda benzer bir senaryoyu ele alıyor olacağız. Senaryomuzda bir oyun sahnesinde yer alan çok sayıda asker olduğunu düşünüyoruz. örnek olarak er ve çavuşları göz önüne alacağız. Bunların çok sayıda olduğunu ve sürekli tekrar eden sayısız örneklerinin uygulama alanında değerlendirildiğini göz önüne alırsak, oyun sahnesine getirdikleri bellek yükünü hafifletmek amacıyla, söz konusu asker nesnelerini birer **FlyWeight** tip haline getirmeyi deneyeceğiz. Sonrasında ise bu askerlerin oyun sahnesindeki yüklerini dengeleyecek, bir başka deyişle havuzu oluşturup, istemciye sunacak bir **fabrika tipi(FlyWeight Factory)** tasarlayacağız. Elbette öncesinde **FlyWeight** deseninin genel **UML** şemasına bakmamızda yarar var.



UML şemamızda gördüğümüz üzere **FlyWeightFactory** nesnesi ile **FlyWeight** nesnesi arasında bir **Aggregation** söz konusudur. Bu son derece doğaldır nitekim fabrikamız, kendi içerisinde yer alan bir depolama alanı ile **FlyWeight** nesne örneklerini havuzlamakta ve istemcinin ihtiyacı olan **FlyWeight** nesne örneklerini bu havuzdan tedarik etmektedir. Bu noktada **istemci(Client)** ile,**FlyWeight Factory** ve **Concrete FlyWeight** nesneleri arasında tek yönlü bir **Association** söz konusudur. Yani, **Client** diğerlerinin nesne örnekleri ve içeriklerini kullanmaktadır. **Concrete FlyWeight**tipi, türeyenler için **Intrinsic state** verileri ile **Extrinsic state** verilerinin ele alındığı ortak operasyonu tanımlamaktadır. **Interface** veya **abstract** sınıf tipinden tasarlanabilir.

Artık örneğimizi geliştirmeye başlayabiliriz. İşte sınıf diagramımız;



Ve kodlarımız;

```
using System.Collections.Generic;
using System;

namespace FlyWeight
{
   enum SoldierType
   {
      Private,
      Sergeant
   }

   // FlyWeight Class
   abstract class Soldier
   {
      #region Intrinsic Fields

      // Bütün FlyWeight nesne örnekleri tarafından ortak olan ve paylaşılan veriler
      protected string UnitName;
      protected string Guns;
      protected string Health;

      #endregion
```

```
        #region Extrinsic Fields

        // İstemci tarafından değerlendirilip hesaplanan ve MoveTo operasyonua gönderilerek FlyWeight nesne örnekleri tarafından değerlendirilen veriler
        protected int X;
        protected int Y;

        #endregion

        public abstract void MoveTo(int x, int y);
    }

    // Concrete FlyWeight
    class Private
        : Soldier
    {
        public Private()
        {
            // Intrinsict değerler set edilir
            UnitName = "SWAT";
            Guns = "Machine Gun";
            Health = "Good";
        }
        public override void MoveTo(int x, int y)
        {
            // Extrinsic değerler set edilir ve bir işlem gerçekleştirilir
            X = x;
            Y = y;
            Console.WriteLine("Er ({0}:{1}) noktasına hareket etti", X, Y);
        }
    }

    // Concrete FlyWeight
    class Sergeant
        : Soldier
    {
        public Sergeant()
        {
            UnitName = "SWAT";
            Guns = "Sword";
            Health = "Good";
        }
        public override void MoveTo(int x, int y)
        {
            X = x;
```

```
            Y = y;
            Console.WriteLine("çavuş ({0}:{1}) noktasına hareket etti",X,Y);
        }
    }

    // FlyWeight Factory
    class SoldierFactory
    {
        // Depolama alanı(Havuz).
        // Uygulama ortamında tekrar edecek olan FlyWeight nesne örnekleri depolama alanında basit birer Key ile ifade edilir
        private Dictionary<SoldierType, Soldier> _soldiers;

        public SoldierFactory()
        {
            _soldiers = new Dictionary<SoldierType, Soldier>();
        }

        public Soldier GetSoldier(SoldierType sType)
        {
            Soldier soldier = null;

            // Eğer depolama alanında, parametre olarak gelen Key ile eşleşen bir FlyWeight nesnesi var ise onu çek
            if (_soldiers.ContainsKey(sType))
                soldier = _soldiers[sType];
            else
            {
                // Yoksa Key tipine bakarak uygun FlyWeight nesne örneğini oluştur ve depolama alanına(havuz) ekle
                if (sType == SoldierType.Private)
                    soldier = new Private();
                else if (sType == SoldierType.Sergeant)
                    soldier = new Sergeant();
                _soldiers.Add(sType, soldier);
            }

            // Elde edilen FlyWeight nesnesini geri döndür
            return soldier;
        }
    }

    class Program
    {
        public static void Main()
```

```
        {
            // İstemci için örnek bir FlyWeight nesne örneği dizisi oluşturulur
            SoldierType[] soldiers = { SoldierType.Private, SoldierType.Private,
SoldierType.Sergeant, SoldierType.Private, SoldierType.Sergeant };

            // FlyWeight Factory nesnesi örneklernir
            SoldierFactory factory = new SoldierFactory();

            // Extrinsic değerler set edilir
            int localtionX = 10;
            int locationY = 10;

            foreach (SoldierType soldier in soldiers)
            {
                localtionX += 10;
                locationY += 5;
                // O anki Soldier tipi için MoveTo operasyonu çağırılmadan önce fabrika
nesnesinden tedarik edilir
                Soldier sld = factory.GetSoldier(soldier);
                // FlyWeight nesnesi üzerinden talep edilen operasyon çağrısı gerçekleştirilir
                sld.MoveTo(localtionX, locationY);
            }
        }
    }
}
```

Dilerseniz örneğimizi kısaca incelemeye çalışalım.

FlyWeight haline getirilen **Private** ve **Sergeant** isimli sınıflarımız **abstract Soldier** sınıfından türemektedir. **Soldier** sınıfı bu desene göre **FlyWeight** tipi görevini üstlenmekte olup kendisinden türeyen **Private** ve **Sergeant** tipleri asıl **FlyWeight tiplerinin(Concurrent FlyWeight)** modelleridir. **Soldier** tipi içerisinde bir askerin ortak alanları tutulmaktadır. Bunlardan **UnitName, Guns** ve**Health** özellikleri aslında **içsel durumu(Intrinsic State)** ifade etmektedir. Bir başka deyişle tüm benzer askerler için ortak ve paylaşılan bilgiler olarak düşünülmektedir. Tabiki senaryo gereği. Diğer yandan bir askerin, oyun sahası üzerindeki lokasyonu **X** ve **Y** isimli alanlarda tutulmaktadır. Bu alanların değerleri istemci açısından önemlidir. Nitekim oyun sahasında aynı askerin birden fazla örneği olabilmesine rağmen lokasyonları çeşitlilik gösterebilir. Bu nedenle X ve Y alanları aslında bir askerin **harici durumu(Extrinsic State)** ile alakalıdır. Peki bu durum nasıl değerlendirilir?

Desenin uygulanış biçimi gereği **Extrinsic State** içeriği, **FlyWeight** nesne örnekleri içerisine bir operasyon yardımıyla aktarılır ve değerlendirilir. İstemci tarafından gerçekleştirilecek bu operasyon çağrısı, örnek senaryomuzda **MoveTo** isimli metod ile ifade edilmektedir. Desenin belkide en önemli aktörlerinden

biriside **SoldierFactory(FlyWeight Factory)** isimli sınıftır. Bu sınıf içerisinde dikkat edileceği üzere askerlerin birer anahtar ile saklanabilmeleri ve bu sayede birden fazla sayıda olan aynı **FlyWeight** nesnesinin tek bir sembol ile ifade edilebilmeleri mümkündür. Bunun için basit bir **Dictionary<T,K>** koleksiyonundan yararlanılmaktadır. Bu koleksyion tam olarak, **FlyWeight** nesne havuzunun kendisidir. Peki istemci tarafının talep edeceği **Soldier(FlyWeight)** nesne örnekleri nasıl elde edilecektir. **GetSoldier** metodu içerisinde buna uygun bir kod yer almaktadır. Dikkat edileceği üzere daha önceden **havuz içerisinde(Koleksiyon içi),** metoda gelen parametre tipinden bir anahtar var ise bunun karşılığı olan nesne anında geri döndürülmektedir. Ancak aksi durumda, söz konusu **anahtar(Key)** için bir nesne örneklenmekte, koleksiyona*(yani havuza)*eklenmekte ve geri döndürülmetkedir. Böylece, eklenen bu **FlyWeight** nesnesinin aynısından tekrar talep edilirse havuzdan karşılanması sağlanmış olacaktır.

Main metodu aslında istemcinin mevzuyu ele aldığı yerdir. **soldiers** isimli dizi içerisinde **SoldierType enum** sabitinden pek çok değer tutulmaktadır. Dikkat edileceği üzere tekrar eden bir sürü değer vardır. **3 Private** ve **2 Sergeant** tipi tanımlanmıştır. **SoldierFactory** örneklendikten sonra ise tüm bu askerler için ortak bir operasyon gerçekleştirilmektedir. Her biri bulundukları lokasyonlardan farklı bir yere doğru hareket ettirilmektedir. Hareket edilecek yeri istemci belirlemekte ve bunu **FlyWeight** nesnelerine **MoveTo** operasyonu yardımıyla bildirmektedir. Bu noktada **for**döngüsü içerisinde **MoveTo** operasyonundan önce*(Extrinsic State değerlerinin ele alındığı fonksiyondan önce)* fabrika nesnesinden bir **Soldier** talep edildiğine dikkat edilmelidir. İşte bellek tüketiminin kontrol altına alınmaya başaldığı yer burasıdır. Eğer havuzda bir **FlyWeight** nesne var ise oradan tedarik edilecek yoksa havuza eklendikten sonra geriye döndürülecektir ki bir sonraki karşılaşmada havuzdan tedarik edilebilsin. Mutlaka farketmişsinizdir bu desen **Factory** ve **Singleton** desenelerinide kullanmaktadır. Hatta **State** ve **Strategy** nesnelerininde bu kalıp içerisinde ele alındığı görülmektedir.

Bu desen ile ilişkili görsel dersi hazırlayana dek size tavsiyem, istemci tarafındaki for döngüsünde adım adım **debug** ederek ilerlemeniz olacaktır. Bununla birlikte mutlaka **oodesing.com,**
dofactory.com ve **sourcemaking.com/design_patterns** sitelerindeki örnekleri incelemenizi öneririm. Böylece geldik bir desenin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

FlyWeightPattern.rar (23,15 kb)

Tasarım Desenleri - Chain of Responsibility (2009-07-24T23:15:00)

design patterns,oop,c#,

Merhaba Arkadaşlar,

Dün gece çok garip bir rüya gördüm. Rüyamda denize açılmak için limanda duran tekneme doğru buldum. E tabi rüya bu. Hareket etmek istedim ama bir türlü beceremedim. Sonunda sorunun ne benzeri ile karşılaştım. Geminin demir halat zinciri(zincirleri) arap saçına dönmüştü. Sabah uyan farkettim.

Acaba bu bir işaret miydi? Evet sanırım bu davranışsal tasarım desenlerinden olan **Chain of Responsibility'** yi anlatmam için bir işaretti. İşte maceramız başlıyor.

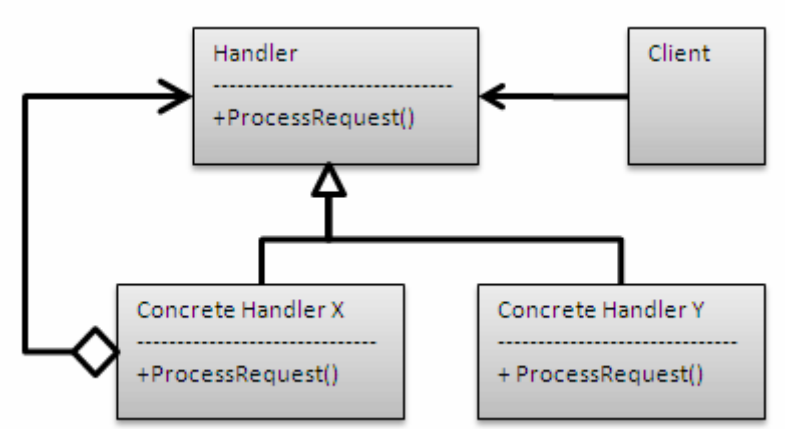
Davranışsal(Behavioral) kalıplardan olan **Chain of Responsibility** deseni, ortak bir **mesaj** veya **talebin(Request),** birbirlerine **zayıf bir şekilde bağlanmış(Loosly Coupled)** nesneler arasında gezdirilmesi ve bu zincir içerisinde asıl sorumlu olanı tarafından ele alınması gerektiği vakalarda kullanılmaktadır. **DoFactory.com** güncel istatistiklerine baktığımızda kullanım oranı %40' lar seviyesinde görünsede, yazılışı son derece basit bir desendir. Desende mesajı(talebi) işleyecek olan asıl nesne örnekleri hayali bir zincir şeklinde dizilmektedir. İstemci, işlenmesini istediği bilgiyi bu zincirin en başında yer alan nesneye gönderir. Zincir içerisinde yer alan nesne örnekleride söz konusu içeriği asıl işleneceği yere kadar göndererirler. Bir başka deyişle bir akıştan(Flow) söz etmemiz mümkündür. Zincire atılan her mesaj, zincire dahil olan tüm nesneler tarafından ele alınabilir veya bir sonrakine gönderilebilir.

Araştırma yaptığım pek çok kaynakta akılda kalıcı bir örnek olarak otomati oluşturmak aslında arka arkaya if blokları yazarak, gelen talebin anlaşılmay tek bir slot tasarlanır**(Handler).** ürünü satın almak isteyen kişinin attığı jeto olan saklama alanına**(ConcreteHandler)** düşecektir. Sonrasında ise süreç, j edilmesiyle tamamlanacaktır.

Yine gerçek hayat örneklerinden devam edersek; bir satın alma sürecinde, ödeme onayının kim tarafından verileceğinde de bu desen göz önüne alınabilir. Bu senaryoda ödeme talimatını onaylayabilecek olan yetkililer bulunur. Ancak gelen ödeme talebinin tutarına göre ilk yetkili personel, talebi bir üst yetkiliye iletmek zorunda olabilir. Bu durumda yetkililerin bir sorumluluk zincirinin parçası oldukları düşünülebilir. Burada en alt yetkiliye gelen ödeme talebi, gerektiğinde zincirin sonunda yer alan en üst yetkiliye kadar gidebilmelidir. Ayrıca bu yetkililerin her biri, birbirlerine sadece bu ödeme talepleri kapsamında bağlı olarak düşünülebilir. Bir başka deyişle ödeme onayı için her biri kendi sorumluluklarına sahip iken, farklı işlerde birbirlerinden tamamen bağımsızlardır.

Peki ya bizim dünyamızda(yani Matrix' in içerisinde) ne gibi örnekler verebiliriz? Belkide en yakın örnek **olay güdümlü programlamada(Event Based Programming)** görülür. Bazı senaryolarda bir olayın birden fazla nesne tarafından ele alınması gerektiği durumlar söz konusu olabilir. Bunu daha çok iç içe bileşenler içeren **Form'** larda veya diğer **taşıyıcı(Container)** kontrollerde görebiliriz. Nitekim hepsi için ortak sayılabilecek bir takım olaylar mevcuttur ve kullanıcının herhangibirini tetiklemesi halinde, bu kontrol zinciri içerisindeki hangi bileşenin üretilen olayla ilişkili olduğunun tespit edilmesi ve buna göre işlemlerinin yapılması gerekir. **Chain of Responsibility** deseni bu noktada devreye girerek üretilen olayın asıl sorumlusu olan bileşen tarafından ele alınmasında önemli bir rol oynamaktadır. örnekler çoğaltılabilir. Söz gelimi, wikipedia da bu tasarım kalıbı ile ilişkili olaraktan, **Loglama** örneği verilmektedir.

Gelelim desenimizin UML şemasına;



Şekildende görüleceği üzere son derece basit bir tasarım kalıbı. Dikkat çekici ilk nokta, **Handler** tipi ile **ConcreteHandler'** lar arasında **aggregation** tadında bir ilişki olmasıdır. Aktörlerimiz ise;

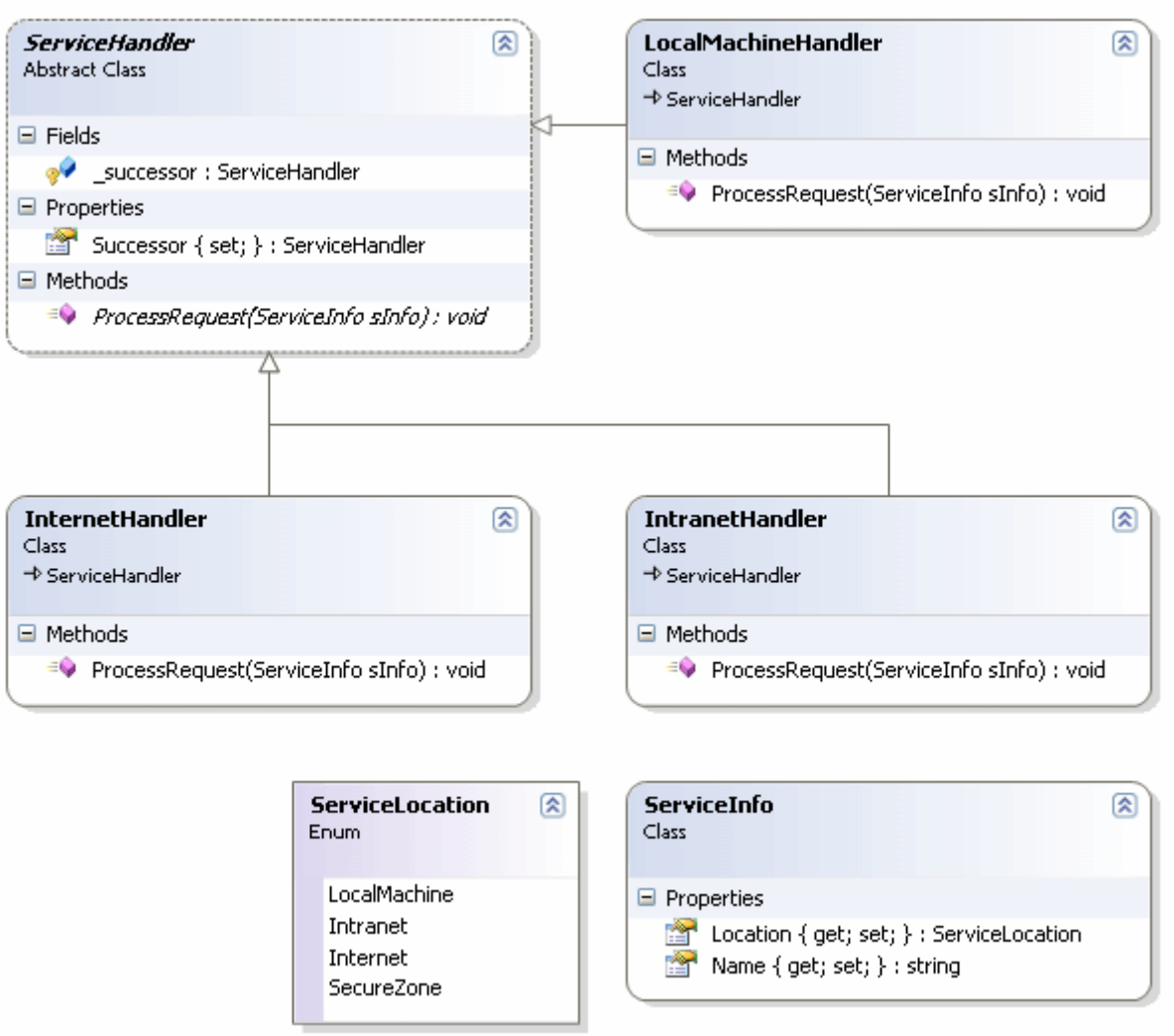
Handler : Kendisinden türeyen **ConcreteHandler'** ların, talebi ele alması için gerekli arayüzü tanımlar. **Abstract class** veya **Interface** olarak tasarlanır.

ConcreteHandler : Sorumlu olduğu talebi değerlendirir ve işler. Gerekirse talebi zincir içerisinde arkasından gelen nesneye iletir. Sonraki nesnenin ne olacağı genellikle istemci tarafında belirlenir.

Client : Talebi veya mesajı gönderir.

Artık kendi örneğimizi geliştirmemizin vakti geldi sanırım. örnek senaryomuzda sorumluluk zincirine dahil edeceğimiz bir servis bilgisi olacak. Servis bilgisini basit bir sınıf olarak tasarlayacağız. Servisin en önemli noktası lokasyon özelliğidir(Location). Servisin yerel makineden, bilgisayarın içinde bulunduğu bir network' ten veya internet üzerinden erişilebilir bir yerde olup olmama durumuna göre zincir içerisindeki sorumlu

nesne tarafından ele alınmasını sağlamaya çalışacağız. İşte örnek kodlarımız ve sınıf çizelgemiz.



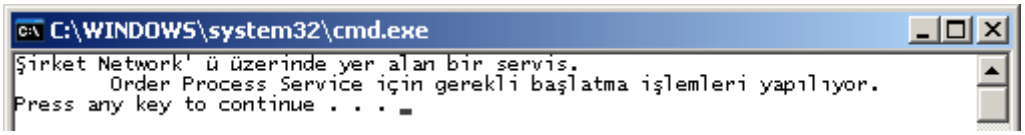

```
using System;

namespace ChainOfResponsibilityPattern
{
   // Yardımcı enum sabiti
   enum ServiceLocation
   {
      LocalMachine,
      Intranet,
      Internet,
      SecureZone,
   }

   // Zincir içerisindeki nesnelerde dolaşabilecek olan tip
   class ServiceInfo
   {
```

```
    public string Name { get; set; }
    public ServiceLocation Location { get; set; }
}

// Handler
abstract class ServiceHandler
{
    protected ServiceHandler _successor;
    public ServiceHandler Successor
    {
        set
        {
            _successor = value;
        }
    }

    public abstract void ProcessRequest(ServiceInfo sInfo);
}

// ConcreteHandler
// Servisin Internet üzerinde olduğu durumu ele alır.
// Sorumluluk zincirinin son sırasındaki tip
class InternetHandler
    : ServiceHandler
{
    public override void ProcessRequest(ServiceInfo sInfo)
    {
        // Eğer lokasyon Internet ise bu tipe ait nesnenin sorumluluğundadır Eğer Internet'
de değilse artık sernin son halkası olduğundan gidecek başka bir yer kalmamıştır. Buna
uygun şekilde bir hareket yapılmalıdır.
        if(sInfo.Location== ServiceLocation.Internet)
            Console.WriteLine("Web ortamı üzerinde yer alan bir servis.\n\t{0} için gerekli
başlatma işlemleri yapılıyor.", sInfo.Name);
        else
            Console.WriteLine("Uzaydan gelen bir servis mi bu yauv?");
    }
}

// ConcreteHandler
// Servisin Intranet üzerinde olduğu durumu ele alır.
class IntranetHandler
    : ServiceHandler
{
    public override void ProcessRequest(ServiceInfo sInfo)
    {
```

```
        // Eğer servis yerel makinede değilse zincirin bir sonraki tipi olan IntranetHandler' a gelir. Burada servis lokasyonunun Intranet olup olmadığına bakılır. Eğer öyleyse sorumluluk buradadır ve yerine getirilir.Ama değilse, zincirde bir sonraki tip olan InternetHandler nesne örneğine ait ProcessRequest metodu çağırılır.
        if(sInfo.Location== ServiceLocation.Intranet)
            Console.WriteLine("Şirket Network' ü üzerinde yer alan bir servis.\n\t{0} için gerekli başlatma işlemleri yapılıyor.", sInfo.Name);
        else if(_successor!=null)
            _successor.ProcessRequest(sInfo);
    }
}

// ConcreteHandler
// Servisin yerel makineye ait olma durumunu ele alır.
class LocalMachineHandler
    : ServiceHandler
{
    public override void ProcessRequest(ServiceInfo sInfo)
    {
        // Eğer servis yerel makinede ise sorumluluk LocalMachineHandler nesne örneğine aittir. Ancak değilse, zincirde bir sonraki tip olan IntranetHandler' a ait ProcessRequest metodu çağırılır.
        if(sInfo.Location== ServiceLocation.LocalMachine)
            Console.WriteLine("Yerel makinede yer alan bir servis.\n\t{0} için gerekli başlatma işlemleri yapılıyor.", sInfo.Name);
        else if (_successor != null)
            _successor.ProcessRequest(sInfo);
    }
}

class Program
{
    static void Main(string[] args)
    {
        // önce zincire dahil olacak nesne örnekleri oluşturulur
        ServiceHandler handlerLocal = new LocalMachineHandler();
        ServiceHandler handlerIntranet = new IntranetHandler();
            ServiceHandler handlerInternet = new InternetHandler();

        // Zincirde yer alan her bir nesne kendisinden sonra gelecek olan nesneyi belirler.
        // Bu belirleme işlemi için Successor özelliği kullanılır.
        handlerLocal.Successor = handlerIntranet;
        handlerIntranet.Successor = handlerInternet;
```

```
            // Zincir halkasındaki nesneler tarafından kullanılacak olan nesne örneği
oluşturulur.
            ServiceInfo info = new ServiceInfo { Name = "Order Process Service",
Location = ServiceLocation.Intranet };

            // Zincirin ilk halkasındaki nesneye, talep gönderilir.
            handlerLocal.ProcessRequest(info);

            // Servisi kırdığımız nokta. Minik bir bomba ve antrenman sorusu.
            // handlerInternet.ProcessRequest(info);
        }
    }
}
```

örneği çalıştırdığımızda aşağıdaki sonucu elde ederiz.



Servis lokasyonu intranet olduğundan, zincirin ilk halkasındaki **handlerLocal** isimli nesne örneği, sorumluluğu bir sonraki nesneye atmıştır. Bu nedenle **IntranetHandler** tipi içerisinde **ProcessRequest**metodu çalışmıştır ve sonraki adımda yer alan **InternetHandler** tipine bir geçiş söz konusu olmamıştır. **info** değişkenine ait **Location** özelliğinin değerini değiştirerek farklı sonuçları değerlendirebilirsiniz. Tabi mutlaka dikkatinizi çekmiştir, **Location** özelliğinin işaret ettiği **ServiceLocation enum** tipi içerisinde, zincir üzerinde ele alınmayan sabit bir değerde vardır. **SecureZone. Dın dın dın dıınnnnn** Sizce neden panik oldum acaba. Bunu bir düşünün.

örnekte görüldüğü üzere, **ServiceInfo** tipinden bir nesne örneğinin **Location** özelliğinin değerine göre bir akış gerçekleştirilmektedir. Bu akışa ait zincir halkasının ilk nesnesi**LocalMachineHandler** iken son nesneside **InternetHandler** tipine aittir. Zincirdeki tüm tipler, **ServiceHandler** isimli **abstract** sınıftan türemektedir. Bu abstract sınıf, kendi tipinden bir özelliğe sahiptir. **Successor** isimli bu özellik ile amaç, halkadaki bir nesnenin kendisinden sonra gelecek olanı işaret etmesini sağlamaktır. Zincirdeki her nesnenin*(Sonuncu hariç)* bir **Successor'u** olmalıdır. Doğal olarak ilerleyen zamanlarda zincire başka bir nesnenin eklenmesi söz konusu olabilir. Bu nedenle, **Successor** özelliğinin aslında tüm **ConcreteHandler'** ların türediği ata tipi**(Handler)**kullanması son derece mantıklıdır.

İstemci tarafındaki kod içinde de dikkat edilmesi gereken bir takım hususlar vardır. Zincir içerisindeki her bir nesne örneklendikten sonra, sıraya göre birbirlerine **Successor** özellikleri üzerinden bağlanırlar. Bu doğal olarak zincirin doğru biçimde sıralanmasını gerektirir. Aksi durumda iş mantığına uygun olmayan sonuçlar alabiliriz. öyleki asıl gidilmesi gereken yer yerine farklı bir yere gidilebilir.*(ödemenin*

onayını Genel Müdürün vermesi gerekirken, zincirdeki hatalı atama sonrası gişe memurunun trilyonlar için yetki vermesi gibi) Yada zinciri kıracak şekilde bir çağrıda gelebilir. örneğin kodun son kısmında minik bir bomba yer almaktadır. Buradaki sorunun ne olabileceğini bulmak ve bir yorum yapmak sizin göreviniz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ChainOfResponsibilityPattern.rar (25,32 kb)

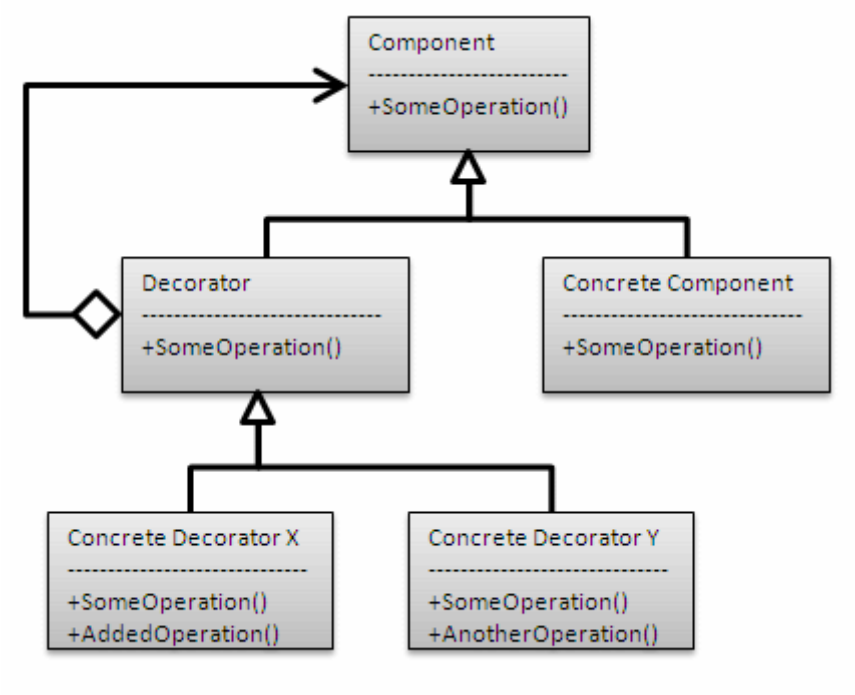
Tasarım Desenleri - Decorator (2009-07-22T17:44:00)

design patterns,oop,c#,

Merhaba Arkadaşlar,

Görsel tasarım işinden pek anladığımı söyleyemem. Hatta ne zaman büyük bir hevesle **Win Forms** yada **Asp.Net** ön yüzü tasarlamaya kalksam renkleri bir türlü tutturamayak başladığım süreci hep yarım bırakmak zorunda kalırım. Bu sebepten genellikle arka plandaki iş mantıkları ile uğraşmayı tercih ederim. Sanırım **WCF** tarafında geliştirme yapmayı sevmemin en büyük nedenide bu olsa gerek. Anlaşılacağı üzere sanatsal yeteneğim pek yok. Hatta evimizin tüm dekorasyonu sevgili eşime aittir.
Ama **Decoration** tasarım deseni deyince sanıyorumki anlatabilecek, paylaşabilecek bir kaç bilgim olabilir. İşte bu günkü konumuz **Structural** desenlerden olan **Decorator** Tasarım kalıbı.

Bu tasarım kalıbı bir nesneye dinamik olarak yeni sorumlulukların eklenmesi ve hatta var olanların çıkartılması amacıyla kullanılır. Bir açıdan bakıldığında nesneyi kendisinden türeyen alt sınıflar ile genişletmek yerine kullanılabilen alternatif bir yaklaşım olarak düşünülebilir. Desenin başlıca kahramanları ve **UML** şeması ise aşağıda görüldüğü gibidir.



Gelelim UML şemasında görülen kahramanlarımıza;

- **Component :** Dinamik olarak sorumluklar eklenebilecek olan asıl nesne için sunulan arayüzdür. **Interface** veya **abstract** sınıf olarak tasarlanabilir.
- **ConcreteComponent :** Sorumlulukların dinamik olarak eklenebilecekleri asıl bileşen sınıflarıdır. **Component** arayüzünü uyarlarlar ve **abstract** sınıf olarak tasarlanırlar.
- **Decorator : Decorator** tipi hem **Component** arayüzünü uygular hemde kendi içerisinde **Component** tipinden bir nesne örneği referansını barındırır. Bu sebepten UML şemasındanda görüldüğü gibi **Decorator** ve **Component** arasında bir **Aggregation** ilişkisi mevcuttur.
- **ConcreteDecorator :** Bileşenlere yeni sorumlulukları eklemekle görevli tiptir. Ek işlevler bu tip içerisinde tanımlanan üyelerdir.

Peki bu tasarım kalıbını hangi koşullarda kullanabilir yada tercih edebiliriz?

Aslında tanımı son derece açık olmasına rağmen zihnimizde daha iyi canlanabilmesi için bir kaç senaryoyu göz önüne almamızda yarar olduğu kanısındayım. örneğin grafiksel bir arayüzün genişletilmesini göz önüne alalım. Normal bir **Windows Form' unun** kendisini, **Scroll** ile kullanılabilir şekilde genişletmek istediğimiz bir durumda **Decorator** desenini ele alabiliriz.
özellikle **GUI(GraphicalUserInferface) Tookit'** lerinin çoğu, **Decorator** desenini ele alarak genişletilmeye imkan sağlarlar. *(Hatta O'Reilly yayınevinden çıkan C# 3.0 Design Patterns kitabında, Windows Form' larının genişletilmesine ilişkin bir Decorator örneği verilmektedir.)*

Yakın dostumuz .Net Framework içerisinde yer alan **Streaming** alt yapısındada**(Stream, FileStream, MemoryStream...) Decorator** deseninin kullanıldığını görebiliriz. Kaynak olarak **MSDN Magazine** dergisinde*(uzun bir süre abone olup posta hizmetinin eve sürekli geç getirmesi ve sonundada getirmemesi nedeniyle, online takip ettiğim ama her yazılımcının mutlaka takip etmesi gerektiğini düşündüğüm dergi)* yayınlanan makaleyi okumanızı şiddetle öneririm. Yine **.Net Framework** içerisine baktığımızda, **Asp.Net** tarafında yer alan **IHttpModule**türevli sınıf zincirinde de **Decorator** kalıbının kullanıldığını görebiliriz. örnekleri çoğaltmak mümkündür. Ancak gelin kendi örneğimizi yaparak desenin nasıl uygulandığını öğrenmeye çalışalım.

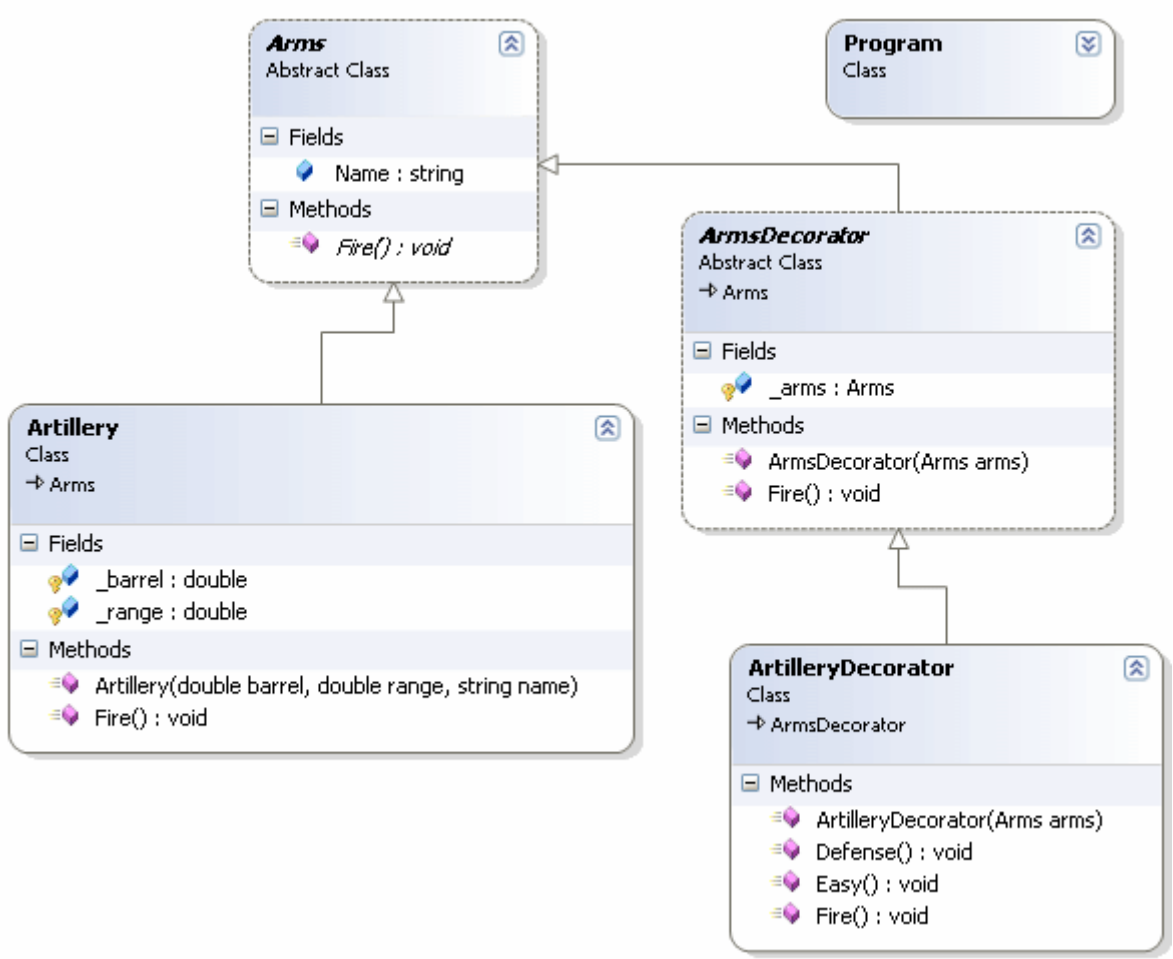
Uzun bir süredir oyunlar içerisinde kullanılan aktörlerden kendimi kurtar bu silahlara dinamik olarak yeni sorumluluklar ekleyebilmek için Decora resimdende görüldüğü üzere senaryomuzun kahramanı bir Topçu batarya bileşen olarak düşüneceğiz.

örnek Console uygulamamızın kod içeriği aşağıdaki gibidir.

```
using System;
using System.Collections.Generic;

namespace Decorator
{
   // Component
   abstract class Arms
   {
      public string Name;
      public abstract void Fire();
   }

   // ConcreteComponent
   class Artillery
      : Arms
   {
      protected double _barrel;
      protected double _range;

      public Artillery(double barrel, double range, string name)
      {
         _barrel = barrel;
         _range = range;
         Name = name;
      }
```

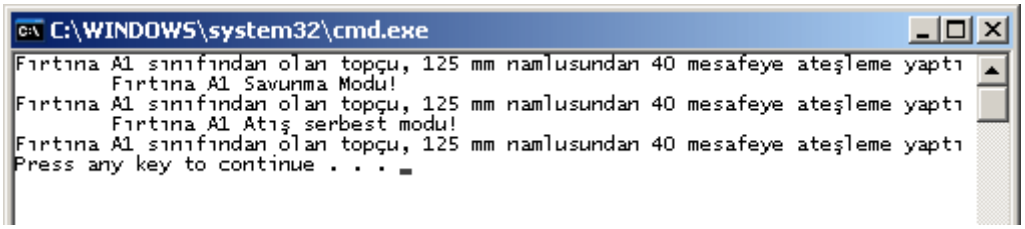
```
    public override void Fire()
    {
        Console.WriteLine("{0} sınıfından olan topçu, {1} mm namlusundan {2} mesafeye ateşleme yaptı", Name, _barrel.ToString(), _range.ToString());
    }
}

// Decorator
abstract class ArmsDecorator
    : Arms
{
    protected Arms _arms;
    public ArmsDecorator(Arms arms)
    {
        _arms = arms;
    }
    public override void Fire()
    {
        if (_arms != null)
            _arms.Fire();
    }
}

// ConcreteDecorator
class ArtilleryDecorator
    : ArmsDecorator
{
    public ArtilleryDecorator(Arms arms)
        : base(arms)
    {
    }

    public void Defense()
    {
        Console.WriteLine("\t{0} Savunma Modu!", base._arms.Name);
    }
    public void Easy()
    {
        Console.WriteLine("\t{0} Atış serbest modu!", _arms.Name);
    }
    public override void Fire()
    {
        base.Fire();
    }
}
```

```
   // Client
   class Program
   {
      static void Main()
      {
         // Bileşen örneklenir
         Artillery azman = new Artillery(125, 40, "Fırtına A1");
         azman.Fire();

         // Decorator nesnesi örneklenir
         ArtilleryDecorator  azmanDekorator= new ArtilleryDecorator(azman);
         // Decorator nesnesi üzerinden o anki asıl Component için(Artillery sınıfı) ek
fonksiyonellikler çağırılır.
         azmanDekorator.Defense();
         azmanDekorator.Fire();
         azmanDekorator.Easy();
         azmanDekorator.Fire();
      }
   }
}
```



örneğimizde **Artillery** isimli bir topçu bileşenine **Defense** ve **Easy** isimli yeni fonksiyonelliklerin eklenebilmesi için **Decorator** tasarım kalıbından yararlanılmıştır. **Arms(Component)** isimli bileşenden türeyen her gerçek tip için sisteme yeni bir **Decorator** eklenebilir. Bir başka deyişle, Tank isimli bir asıl bileşen sisteme dahil olduğunda pekala bunun içinde bir **ConcreteDecorator**tip(örneğin TankDecorator) söz konusu olabilir ki buda Tank için ek sorumlulukların dinamik olarak yüklenmesini sağlayabilir.

Tabiki bu desende kritik olan noktalardan biriside **Decorator** tipinin tanımlanış şeklidir. Bu tip kendi içerisinde, asıl bileşenleri kullanabilmek için **Component** tipinden bir üyeye sahiptir. Aynı zamanda bu üyenin **initialize(başlatılma)** işlemindende sorumludur. Diğer taraftan **Component** tipinden türediği için aslında **Component** içinde tanımlı olup, **ConcreteComponent** tipleri tarafından ezilmesi gereken kuralları kendiside uygulamak zorundadır. İşte bu noktada, kendi içerisinde sakladığı **Component** bileşeninin **abstract** üyelerini çağırarak çalışma zamanında taşıdığı asıl bileşenin ezdiği üyeleri devreye sokabilir*(ArmsDecorator içinde overrride edilmiş Fire metoduna dikkat edelim)*. Ama **puzzle'** ın eksik kalan kısmını**Decorator(ArmsDecorator)** tipinden türeyen bileşenler üstlenmektedir. örnekte yer alan **ArtilleryDecorator(ConcreteDecorator)**, **ArmsDecorator(Decorator)'** den türemekte ve kendisine çalışma zamanında verilen asıl bileşeni üst sınıfın yapıcısına iletmektedir. Bu sebepten üst sınıf üzerinden çağırılan ve asıl bileşenler tarafından ezilen tüm üyelere ulaşabilir*(ArtilleryDecorator içerisinde ezilmiş olan Fire metoduna dikkat edelim)*. Ama aynı zamanda kendisi içerisindede ek fonksiyonellikleri tanımlayabilir. Nitekim çalışma zamanında**ConcreteDecorator(ArtilleryDecorator)** tipinin çalıştığı nesne örneği, ek sorumluluklar üstlenmesi istenen asıl bileşen **ConcreteComponent(Artillery)**' den başkası değildir. Uygulamanın çalışma zamanı çıktısına baktığımızda aşağıdaki sonuçlar elde ettiğimizi görebiliriz.



Ancak bu desendede bazı eksik noktalar olabilir. özellikle, nesnelere yeni fonksiyonelliklerin çalışma zamanında eklenmesi nedeni ile sistemin fonksiyonelliğine ait hataları ayıklamak(debug) daha zordur. Decorator deseni, **Adapter** ve **Composite** kalıpları ile zaman zaman karıştırılabilir. **Adapter** deseni bir nesnenin arayüzünü değiştirirken, **Decorator** kalıbı sorumlulukları**(Responsibilities)** değiştirmektedir. Bununla birlikte decorator deseni sadece bir **component** ile ilgilendiğinden, **Composite** kalıbının çakma bir hali olarakta düşünülebilir Ancak bununla birlikte **Decorator** kalıbının bileşene ek sorumlulukla yüklediğide bir gerçektir. Böylece geldik bir yazımızın daha sonuna. Umarım sizler için yararlı olmuştur. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Decorator.rar (22,98 kb)

Tasarım Desenleri - Builder (2009-07-17T22:44:00)

design patterns,oop,c#,

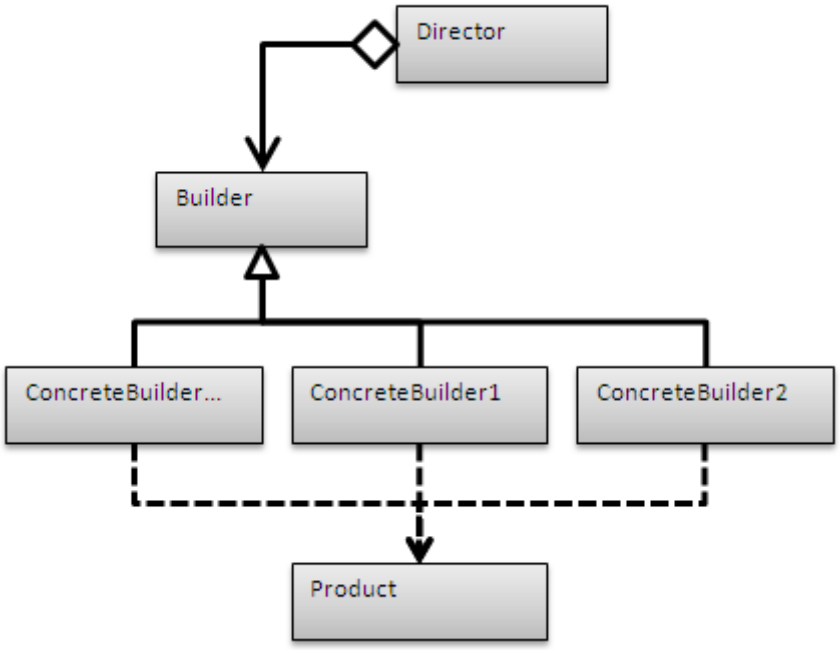
Merhaba Arkadaşlar

Zaman içerisinde geliştirdiğimiz uygulamalar son derece karmaşık bir hal alırlar. Uygulamanın çapının ve ihtiyaçlarının artması bir kenara içerisinde yer alan en küçük parçaların bile kullanımları kompleksleşebilir. Bu küçük birimlerin karmaşıklaşmasına etken olarak gösterilebilecek konulardan biriside, nesne üretimleri için kullanılan sınıfların sayılarının artması ve inşa işlemlerinin kompleks olması olarak düşünülebilir. Hal böyle olunca bazı vakalar için standartlaşmış kalıpları kullanmanın genişleyebilirlik ve ölçeklenebilirlik açısından büyük yararı vardır.

Nesne üretimi söz konusu olduğunda, **Creational** isimli kategoride yer alan tasarım kalıpları göz önüne alınmaktadır. Bunlardan birisi olan **Builder** deseni, karmaşık yapıdaki nesnelerin oluşturulmasında, istemcinin sadece nesne tipini belirterek üretimi gerçekleştirebilmesini sağlamak için kullanılmaktadır. Bu desende istemcinin kullanmak istediği gerçek ürünün birden fazla sunumunun olabileceği göz önüne alınır. Bu farklı sunumların üretimi ise **Builder** adı verilen nesnelerin sorumluluğu altındadır. Dolayısıyla **Builder** kalıbından yararlanılarak aslı ürünün farklı sunumlarının elde edilebilmesi için gerekli olan karmaşık üretim süreçleri, istemciden tamamen soyutlanabilir. Desenin önemli olan özelliklerinden biriside **Abstract Factory**tasarım kalıbı ile çok benzer yapıda olmasıdır. Ancak arada bazı farklılıklarda vardır. Herşeyden önce **Abstract Factory** kalıbına göre, fabrikanın metodları kendi nesnelerinin üretiminden doğrudan sorumludur. Builder deseninin başlıca kahramanları aşağıda sıralandığı gibidir.

- **Builder: Product** nesnesinin oluşturulması için gerekli soyut arayüzü sunar.
- **ConcreteBuilder: Product** nesnesini oluşturur. Product ile ilişkili temel özellikleride tesis eder ve **Product'** ın elde edilebilmesi için(istemci tarafından) gerekli arayüzü sunar.
- **Director: Builder** arayüzünü kullanarak nesne örneklemesini yapar.
- **Product:** üretim sonucu ortaya çıkan nesneyi temsil eder. Dahili yapısı(örneğin temel özellikleri) **ConcreteBuilder** tarafından inşa edilir.

Bu kahramanlarımızın aralarındaki ilişkileri aşağıdaki diagram üzerindeki dizilimi ise aşağıdaki gibidir.



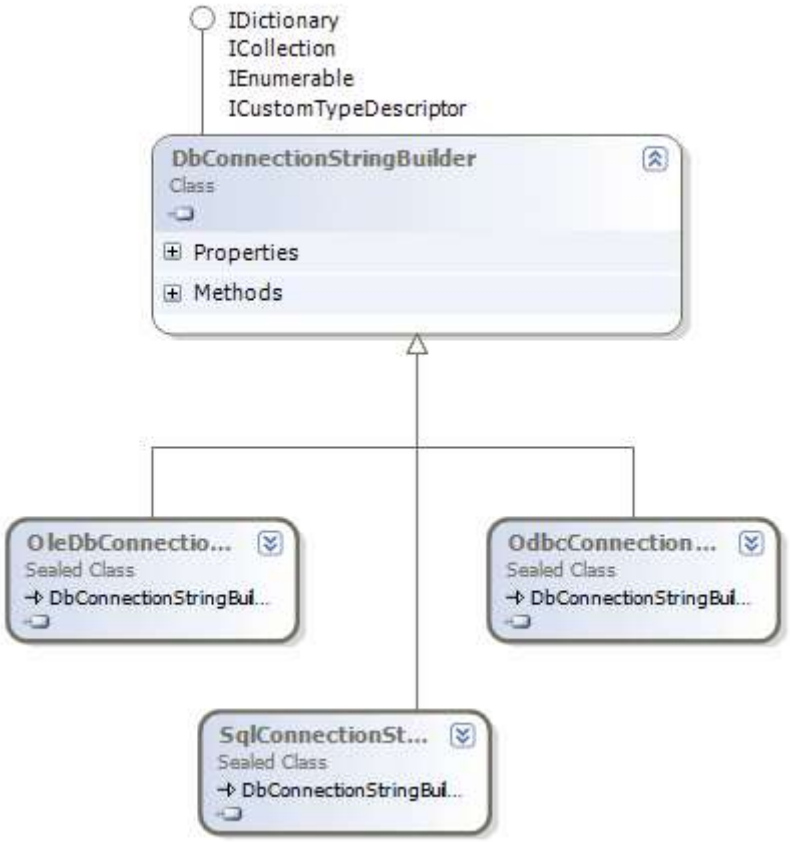
Builder tasarım kalıbının kullanım oranı **doFactory.com** sitesinin istatistiklerine göre **%40' lar** civarındadır. Bunun en büyük nedenlerinden biriside uygun senaryoların tespit edilmesinin zor olmasıdır. Yinede desenin kullanılabileceği bir kaç senaryo üzerinde konuşarak daha kolay anlaşılmasını sağlayabiliriz.

Söz gelimi **dofactory.com** tarafından verilen örneği göz önüne alalım. Bu ö istemci açısından kullanılabilir olması için üretim işleminde motorun(her ne kapıların, viteslerin vb parçalarında üretimi gerekmektedir.

Aslında bu ortak fonksiyonellikler bu ürünlerin hepsi için geçerlidir. Yani bu araçların kendisi bir **Product** olarak temsil edilebilirler. İstemci, sadece kullanmak istediği ürünün farklı bir sunumunu elde etmek isteyecektir. Bu tip bir senaryoda istemcinin asıl ürüne ulaşmak için ele alması gereken üretim aşamalarından uzaklaştırılarak sadece üretmek istediği ürüne ait tipi bildirmesi yeterli olmalıdır. Bu senaryoda araç(Vehicle) aslında üründür(Product). Motorsiklet veya araba ise araç tipleridir ve üretim işlemleri sonucu ortaya bir Vehicle çıkartırlar. Yani desendeki **ConcreteBuilder** tipleridir. Bu senaryo pekala bir oyun programı içerisindeki araçların üretimi aşamasında göz önüne alınabilir.

Diğer bir senaryoda bir firmanın çalışanlarına yılın belirli dönemlerinde farklı tip için sunumu farklı olan promosyon ürünlerinin geliştirilmesi safhasındaki üretim promosyonun kendisi ürün iken, promosyon ürününü kullanacak olan profil sahip

Peki daha gerçekçi bir örnek olamaz mı? Her zaman dediğim gibi, aslında tasarım kalıplarının çoğunun **.Net Framework** içerisinde kullanıldığını kolaylıkla görebiliriz.**Builder** tasarım kalıbı için düşünülebilecek en güzel örnek Connection String Builder operasyonudur.



Şekildende görüldüğü gibi bir istemcinin, kullanmak istediği **Connection** tipi için uygun olan bağlantı bilgisine ihtiyaç vardır. Burada bağlantının **string** şeklindeki içeriği önemlidir. Bu içeriğin sunum şekli ise **OleDb**, **SQL**, **ODBC** için farklıdır. Dolayısıyla söz konusu farklı **string** üretimleri için
bazı **ConcreteBuilder** tiplerinden(**OleDbConnectionStringBuilder** vb...) yararlanılır. Her ne kadar **DbConnectionStringBuilder abstract** bir sınıf olmasada, **Builder** tipinin görevini üstlenmektedir.

Ancak benim popüler senaryom şu anda midemden beynime doğru gelen sinyallerinde söylediği üzere **Pizzacı** örneğidir.

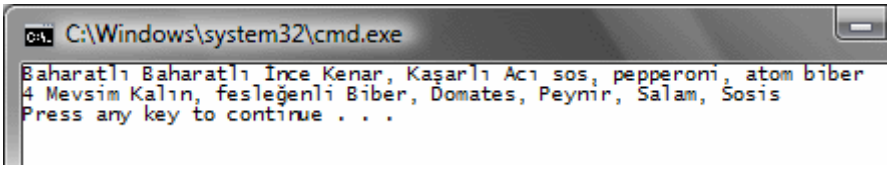
Nitekim şu aşamada heleki hafta sonuna girdiğimiz şu güzel Cuma gecesinde, bu deseni eğlenceli bir şekilde ele almamamız için hiç bir sebep bulunmamaktadır. İşte deseni ele aldığımız kod parçaları.

```
using System;

namespace Builder
{
   // Product class
   public class Pizza
   {
      public string PizzaTipi { get; set; }
      public string Hamur { get; set; }
      public string Sos { get; set; }

      public override string ToString()
      {
         return String.Format("{0} {1} {2}", PizzaTipi, Hamur, Sos);
      }
   }

   // Builder class
   public abstract class PizzaBuilder
   {
      protected Pizza _pizza;

      public Pizza Pizza
      {
         get { return _pizza; }
      }

      public abstract void SosuHazirla();
      public abstract void HamuruHazirla();
   }

   // ConcreteBuilder class
   public class BaharatliPizzaBuilder
      : PizzaBuilder
   {
      public BaharatliPizzaBuilder()
      {
         _pizza = new Pizza { PizzaTipi = "Baharatlı Baharatlı" };
      }
      public override void SosuHazirla()
      {
```

```
        _pizza.Sos = "Acı sos, pepperoni, atom biber";
    }

    public override void HamuruHazirla()
    {
        _pizza.Hamur = "İnce Kenar, Kaşarlı";
    }
}

// ConcreteBuilder Class
public class DortMevsimPizzaBuilder
    : PizzaBuilder
{
    public DortMevsimPizzaBuilder()
    {
        _pizza = new Pizza { PizzaTipi = "4 Mevsim" };
    }
    public override void SosuHazirla()
    {
        _pizza.Sos = "Biber, Domates, Peynir, Salam, Sosis";
    }

    public override void HamuruHazirla()
    {
        _pizza.Hamur = "Kalın, fesleğenli";
    }
}

// Director Class
public class VedenikliKamil
{
    public void Olustur(PizzaBuilder vBuilder)
    {
        vBuilder.SosuHazirla();
        vBuilder.HamuruHazirla();
    }
}

// Client class
class Program
{
    static void Main(string[] args)
    {
        PizzaBuilder vBuilder;
```

```
            VedenikliKamil kamil= new VedenikliKamil();
            vBuilder = new BaharatliPizzaBuilder();

            kamil.Olustur(vBuilder);
            Console.WriteLine(vBuilder.Pizza.ToString());

            vBuilder = new DortMevsimPizzaBuilder();
            kamil.Olustur(vBuilder);
            Console.WriteLine(vBuilder.Pizza.ToString());
        }
    }
}
```

İstemcinin tek derdi istediği tipte bir pizza almaktır. örneğin 4 mevsim veya Baharatlı pizza. Bu pizzaların içinde ise soslarının ve hamurlarının belirlenerek üretim işlemine dahil edilmesi gerekmektedir. Bu **VenedikliKamil** açısından kolay olmakla birlikte istemciyi ilgilendiren bir durum değildir. Bu nedenle istemcinin sadece pizza üretimini gerçekleştiren asıl **ConcreteBuilder** nesne örneğini seçmesi yeterlidir. Bu seçim işlemi **Director** sınıfı içerisindeki **Olustur** metoduna parametre olarak gönderilir. Sonrasında ise istemcinin istediği pizza üretilerek elde edilir. örneği çalıştırdığımızda aşağıdaki sonuçları elde ederiz.



Umarım sizler içinde faydalı bir anlatım olmuştur. Her zamanki gibi bu desenin görsel dersinide en kısa sürede eklemeye çalışacağım. Bu yazının üstünede şöyle güzel bir espresso içilir kanımca

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Builder.rar (23,15 kb)

Business Rule Engine ile Programlama(Biztalk Server 2006) (2009-07-15T22:30:00)

biztalk,business rule engine,c#,

Merhaba Arkadaşlar,

Yıllardır yazılım projelerinde görev almaktayım. çeşitli proje
karşılaştığım en büyük zorluklar arasında, müşterinin uygula
değiştirmek istemesi yer almaktaydı. Burada geliştirme açısı
güncelleme yapılmasına gerek bırakmadan entegre edebilen s

Tabiki burada müşterinin kastettiği iş kurallarının nasıl tanımlandığından tutunda dile getiriliş şekli daha çok büyük öneme sahiptir. Nitekim öyle kurallar olabilir ki, yada bu kurallar öyle şekillerde dile getirilebilirki, yorumlayabilmek veya uygulatabilmek için yapay zeka stratejilerinin ele alınması zorunlu hale gelir. Ben tabiki konunun bu kısmına en azından şu an için girmemeyi tercih etmekteyim 🙂 özetle büyük çaplı uygulamalarda karşılaştığımız en büyük sorunlardan birisinin, müşterinin kullandığı ürün ile ilişkili olaraktan tanımlamış olduğu **iş kurallarının(Business Rules)** koda müdahele etmeden yönetilebilmesi olarak düşünebiliriz. Bazı durumlarda, ürüne ait iş kuralları baştan bellidir ve değişmezdir. Bu tip senaryolara az rastlanmakla birlikte geliştirilmesi kolaydır. Nitekim kod içerisinde konulacak katı kurallar ile söz konusu geliştirme pekala yapılabilir.

Ancak, müşterinin uygulama üzerindeki iş kurallarını yeri geldiğinde değiştirebilmesi isteği(çok sık olmasa bile) geliştirme sürecinde bizleri bir çıkmaza düşürebilir. öyleki, çalışmakta olan sistem içerisindeki kuralların esnetilebilmesi, değiştirilebilmesi ve hatta yenilerinin eklenebilmesi demek, kodu **geliştirmeye** devam etmek**(Development),** yeniden **test(Testing)** ve tekrardan **dağıtım(Deployment)** anlamına gelmemelidir. Her geliştirici takımı bu tip durumlara karşın, uygulamasının kodunu tekrardan güncellemeye gerek bırakmadan yeni kuralları kolayca öğrenebilmesi üzerine tasarlamak ister. Lakin bu sanıldığı kadar kolay bir süreç olmayabilir. Bir noktada **XML** tabanlı olaraktan söz konusu kuralların saklanması ve kod içerisine ele alınması düşünebilir. Hatta daha önceden çalıştığım çok değerli bir şirketin iş akışları üzerine geliştirdiği bir ürün, akış tasarımları, yönetimi ve geliştirilmesi için **XML** tabanlı olan ve basit **IDE** ile çalışan sistemi, Web tabanlı uygulama olarak başarılı bir şekilde dağıtabilmiştir. Hatta **Workflow tabanlı WCF servislerinde(Workflow Based WCF Services)** bile gelinen nokta, içeriğin **XAML** olarak ifade edilebilmesi ve bu nedenle koda müdahele etmeden de değiştirilebilmesi değil midir? 😊 Tüm bunlar bir yana dursun Biztalk ailesinde, kuralları kolayca geliştirebileceğimiz, **veritabanı(database)**, **XML** veya **.NET** tipleri gibi kaynaklardan kural verilerini alıp değerlendirebileceğimiz bir **IDE** zaten mevcuttur**(Business Rule Composer).** Hatta kendi uygulamalarımız için **Biztalk'** un hazır kural moturunuda**(Business Rule Engine-BRE)** kullanabiliriz. Sanıyorum ki artık sadede gelsem iyi olacak. Bu yazımızda giriş seviyesininde altında kalmak üzere, **Biztalk Server 2006** ile birlikte gelen **Business Rule Engine** kütüphanesini nasıl kullanabileceğimizi ve iş kurallarını tanımlamak için**Business Rule Composer** aracını nasıl ele alabileceğimizi incelemeye çalışıyor olacağız.

Biztalk Server ile birlikte gelen **Business Rule Engine'** in, kendi geliştirdiğimiz .Net uygulamalarında kullanılabilmesi için, sunucu lisansına sahip ürünün yanlızca **Business Rules Components**özelliğinin kurulması yeterlidir. Tabiki burada önemli olan noktalardan birisi lisans konusudur. Lisanslı olan bir **Biztalk Server** ürünü üzerinden kurulum yapılmalıdır. Bu nedenle, kendi uygulamalarımızdan kasıt çoğunlukla sunucu tarafında çalışan servis uygulamalarıdır. Böylece, servis tabanlı **.Net** uygulamalarımız içerisinde istersek, **Biztalk Server** ile birlikte gelen kural motorunu kullanabiliriz. **Business Rule Engine** uzun uzun yıllar önce**(1974)** geliştirilmiş **RETE** algoritmasını kullanmaktadır. Tabi başlamadan önce önem arz eden bazı kavramlardan bahsetmekte yarar olduğu kanısındayım. Bunlar;

Business Rule Composer : İlkeleri(Policy), içerisindeki kuralları(Rules) ve daha fazlasını tasarla düşünülebilir. Kısaca iş kurallarını görsel olarak oluşturduğumuz programdır.

Policy : İçinde, iş kurallarını barındıran nesnedir. Bu nesne istenildiğinde versiyonlandırılabilir. İ içeren yada aynı kuralları farklı şekillerde yorumlayan birden çok versiyonu tasarlanabilir ve kull

Policy State : Policy' ler temel olarak **Editable, Saved,**
Published ve **Deployed** durumlarında bulunabilir. Bir Policy ilk kez oluşturulduğunda zaten otomatik olarak Editable moda geçer. Policy' nin kaydedilmesi sonrası Saved moda atanır. Saved modda düzenlemeler ve testler yapılabilir. Eğer Policy, publish edilirse artık düzenlenemez, değiştirilemez. Yani read-only olarak düşünülebilir. Bu aşama, söz konusu Policy Deploy edilmeden önceki zamandır. Policy, Deploy edildiğindeyse artık versiyonlanmış ve kullanılabilir hale gelmiştir. Ne varki bu moddada üzerinde düzenleme yapılamamaktadır. Dolayısıyla bu aşamadan sonra, Policy içerisinde yazılmış olan kurallarda değişim yapılamaz. Ancak yeni bir versiyonlama veya yeni bir Policy oluşturulması ile sorunlar ortadan kaldırılabilir.

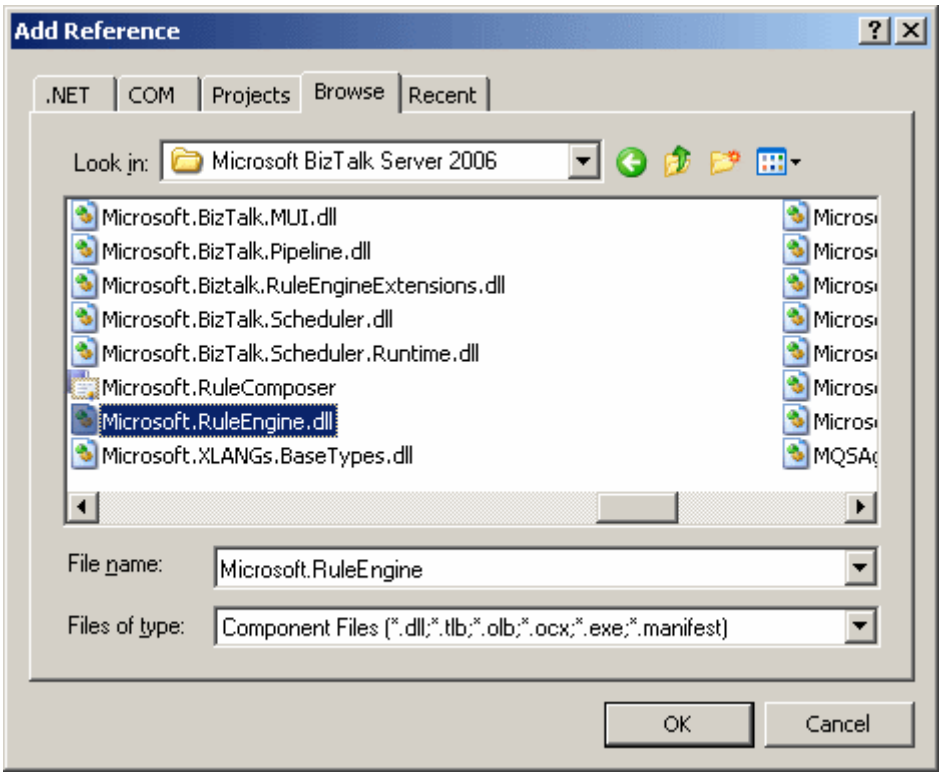
Rules : çok doğal olarak konunun ana fikri bir takım iş kurallarının uygulanmasıdır. İş kuralları, Policy' ler içerisinde Rule nesneleri ile ifade edilir. Rule' lar kendi içlerinde, koşullandırılacak olan verileri(Fact), bunlarla ilişkili Predication' ları ve aksiyonları(Action) içermektedir.

Facts : Aslında Rule içerisinde yer alan koşullar, karşılaştırmalar ve aksiyonlarda kullanılan veri birimlerini temsil etmektedir. çok doğal olarak bu nesnenin uygulanan koşul sonrası yapılacak bir takım işlemler ile kuralın bütünü oluşturulmaktadır.

Fact Source : Fact nesnelerinin içeriği, XML ve veritabanı gibi kaynaklardan gelebileceği gibi, sistemin **Global Assembly Cache(GAC)** alanında yüklü bir assembly içerisindeki .Net tipide olabilir.

Peki bir .Net tipini, **BRE** içerisinde kullanmak ve herhangibir uygulamada bu tipe ait nesne örneklerini Rules Engine içerisinde tanımlı ilkelere dahil etmek istiyorsak nasıl bir yol izlemeliyiz.

1 - İlk olarak .Net tipini içeren bir **Class Library** geliştirilir. Bu library içerisinde BRE managed nesnelerini kullanabilmek için varsayılan olarak **C:\Program Files\Microsoft BizTalk Server 2006** adresinde yer alan **Microsoft.RuleEngine.dll** assembly' ının projeye referans edilmesi gerekir.



Oluşturduğumuz CompanyRules isimli sınıf kütüphanesinde yer alan kod içeriğimiz ise ilk etapta aşağıdaki gibidir.

```
namespace CompanyRules
{
   public class Product
   {
      public int ProductId { get; set; }
      public int Count { get; set; }
      public bool StockLevelOk { get; set; }
   }
}
```

Product isimli sınıf içerisinde yer alan Count özelliğinin değerine göre bir takım kurallar tanımlayacağımızı şimdiden söyleyebilirim. örneğin Count' un belirli bir değerin altında

olması halinde StockLevelOk özelliğine false değerinin atanması bir kural olarak düşünülebilir.

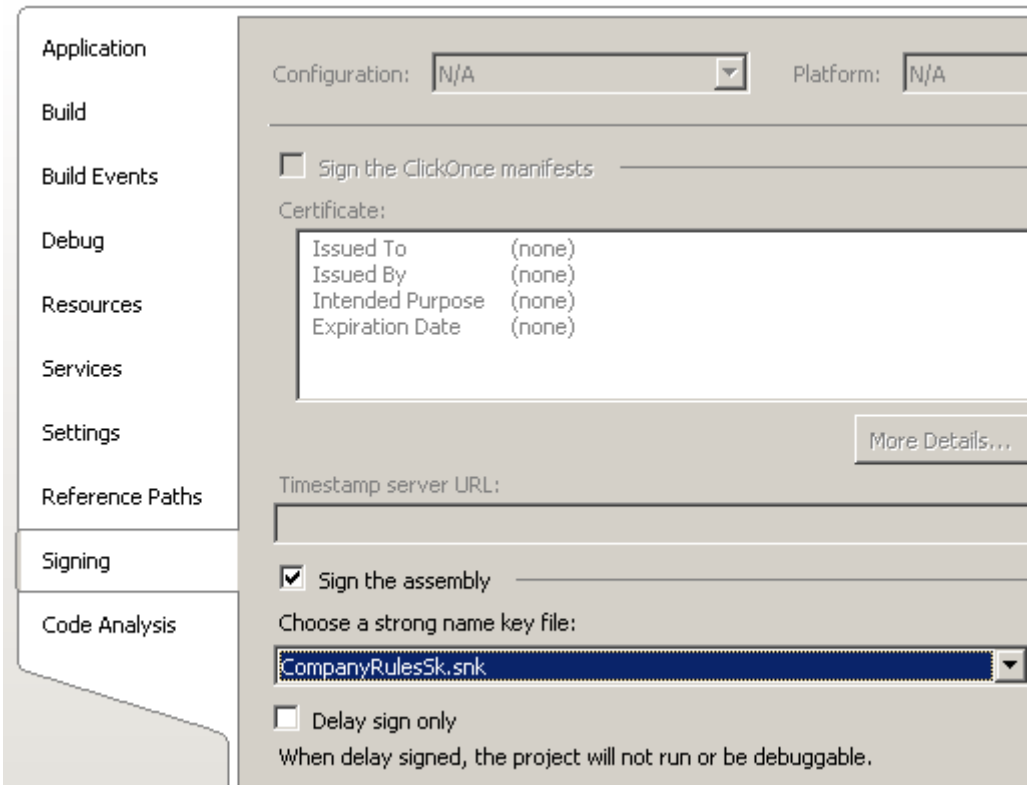
2 - Yazılan .Net tipi için mutlaka bir test tipi geliştirilmelidir. Daha önceden de bahsedildiği üzere, **Policy, Published** veya **Deployed** modlarına geçildiğinde değiştirilemez. Dolayısıyla test edilebilir olması önemlidir. Nitekim test sonuçlarına bakılarak Fact' lerin tanımlanan Rule' lar için doğru çalışıp çalışmadığı değerlendirilmelidir. Bu amaçla, yine**Microsoft.RuleEngine** isim alanı altında yer alan **IFactCreator** arayüzünden(Interface) türeyen bir tip kullanılır. CompanyRules sınıf kütüphanesinde yer alan Product tipi için, IFactCreator arayüzünde türeyen aşağıdaki tip tasarlanmıştır.

```
using System;
using Microsoft.RuleEngine;

namespace CompanyRules
{
   public class ProductFactCreator
      :IFactCreator
   {
      #region IFactCreator Members

      public object[] CreateFacts(RuleSetInfo ruleSetInfo)
      {
         Product prod = new Product
         {
             Count=100,
              ProductId=10001
         };
         return new object[] { prod };
      }

      public System.Type[] GetFactTypes(RuleSetInfo ruleSetInfo)
      {
         return null;
      }

      #endregion
   }
}
```

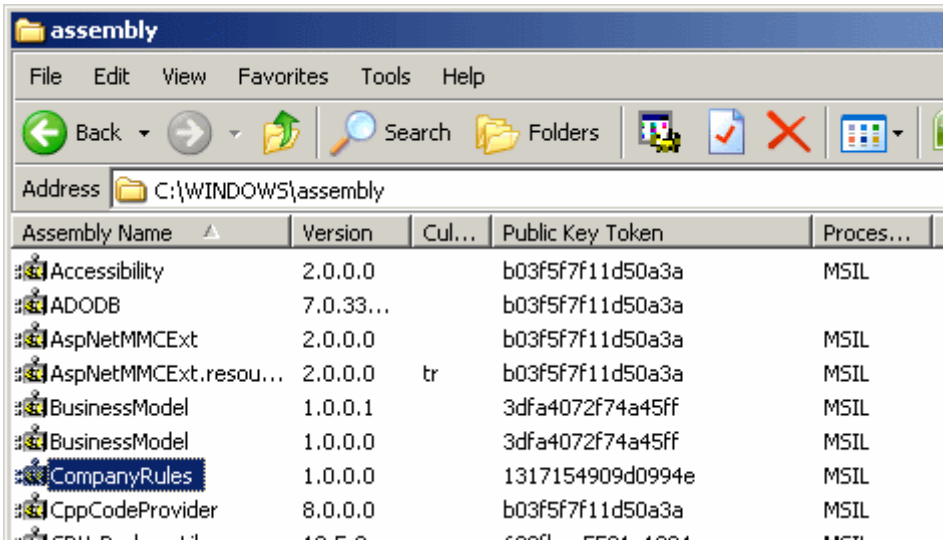
Burada yapılan aslında **Product** biriminin belirtilen bir kural için test edilebilir olmasını sağlamaktır. Bu nedenle, **CreateFacts** metodu içerisinde örnek bir **Product** nesne örneği oluşturulmuş ve geriye döndürülmüştür. Aslında burada işleyiş şekli tam anlamıyla ders niteliğindedir. **IFactCreator** arayüzü, **BizTalk** tarafında tanımlanmıştır. Bu

arayüz, **Business Rule Composer** programındaki testler için önemlidir. Nitekim dışarıdan bir tipin, var olan Biztalk uygulamasına entegre edilmesini sağlamaktadır. Yani bildiğimiz Plug-In mantığı söz konusudur.

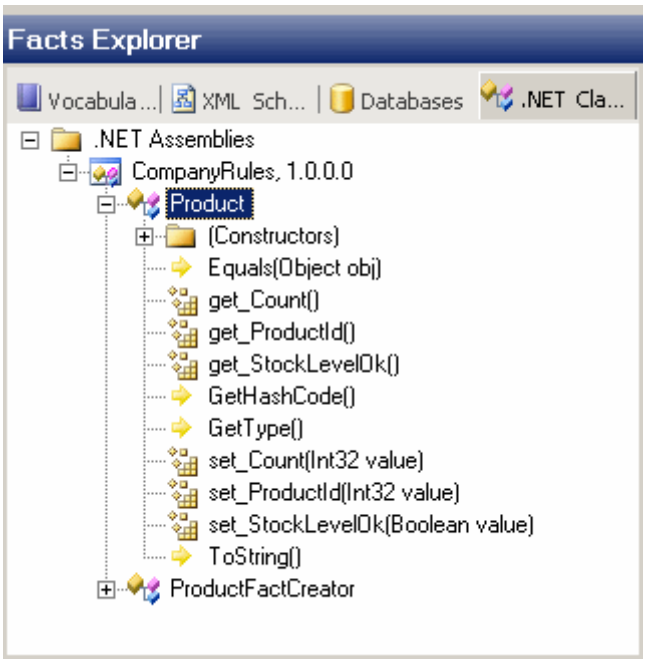
3 - Geliştirilen tiplerin yer aldığı .Net assembly' ının, Business Rule Composer içerisinde kullanılabilmesi için **Strong Name Key** ile imzalanıp **Global Assembly Cache** alanına atılmış olması gerekmektedir. Aksi takdirde **Business Rule Composer** içerisinde kullanılamaz. Tahmin edileceği üzere uygulamamızı Strong Name ile imzalamak için Visual Studio ortamında proje özelliklerinden gerekli ayarlamaları yapabiliriz.



Bu işlemin ardındada derlenen assembly, komut satırından **GacUtil** ile veya basit bir şekilde **Windows\Assembly** klasörü altına sürükle bırak yöntemi ile install edilir.

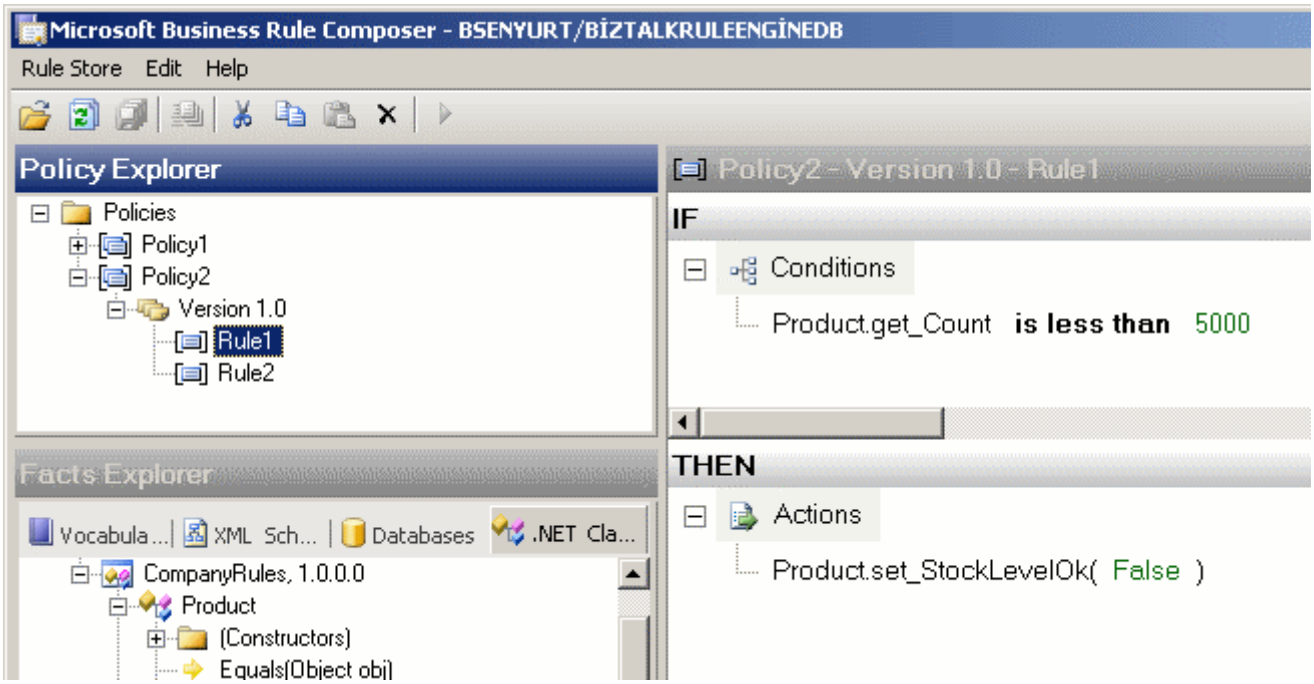


4 - Business Rule Composer aracından yararlanılarak **Policy** ve içerisinde yer alan kurallar oluşturulur. Aslında bu adımı çok fazla dert etmemiz gerek yok. Bu konuyu görsel derstede ele alacağımızdan aşağıdak şekilde görülen basit kuralları oluşturmaya çalışsak yeterli olacaktır. Tabiki unutulmaması gereken önemli noktalardan biriside, Fact Explorer kısmında, aşağıdaki ekran görüntüsünde olduğu gibi CompanyRules assembly' ının seçilmesi gerekliliğidir ki bu sayede Rule içerisindeki Fact' ler için kullanılacak özellikler ele alınabilecektir.

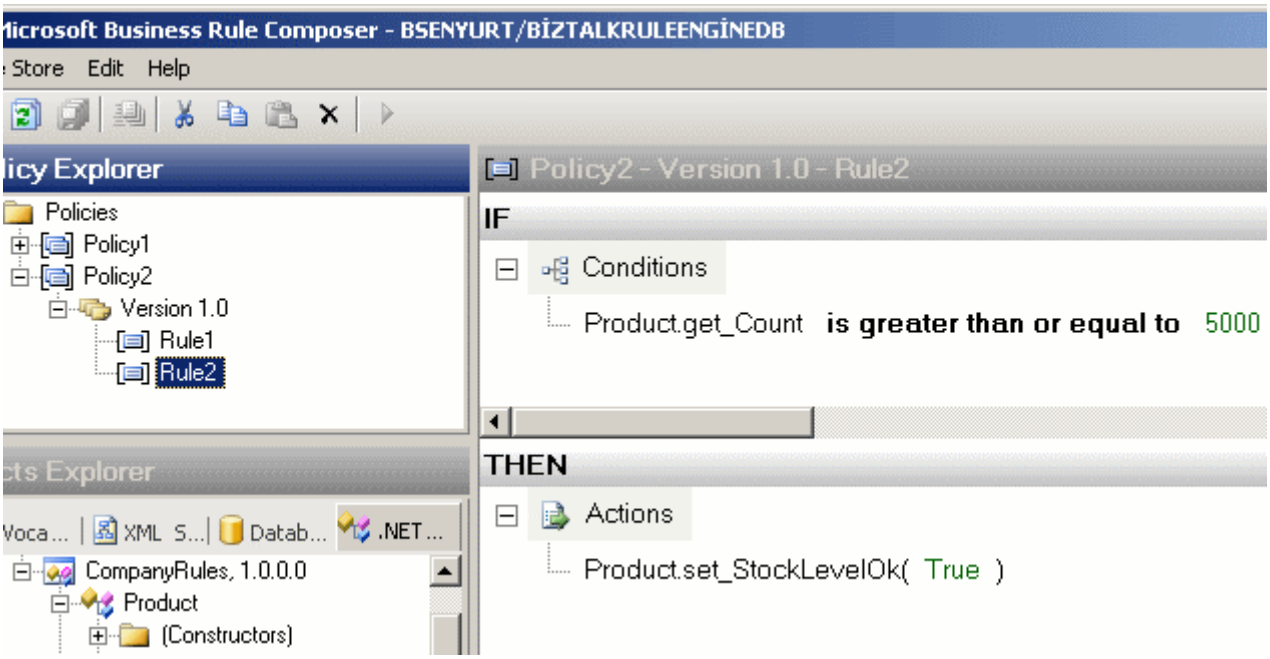


Ve örnek kurallarımız;

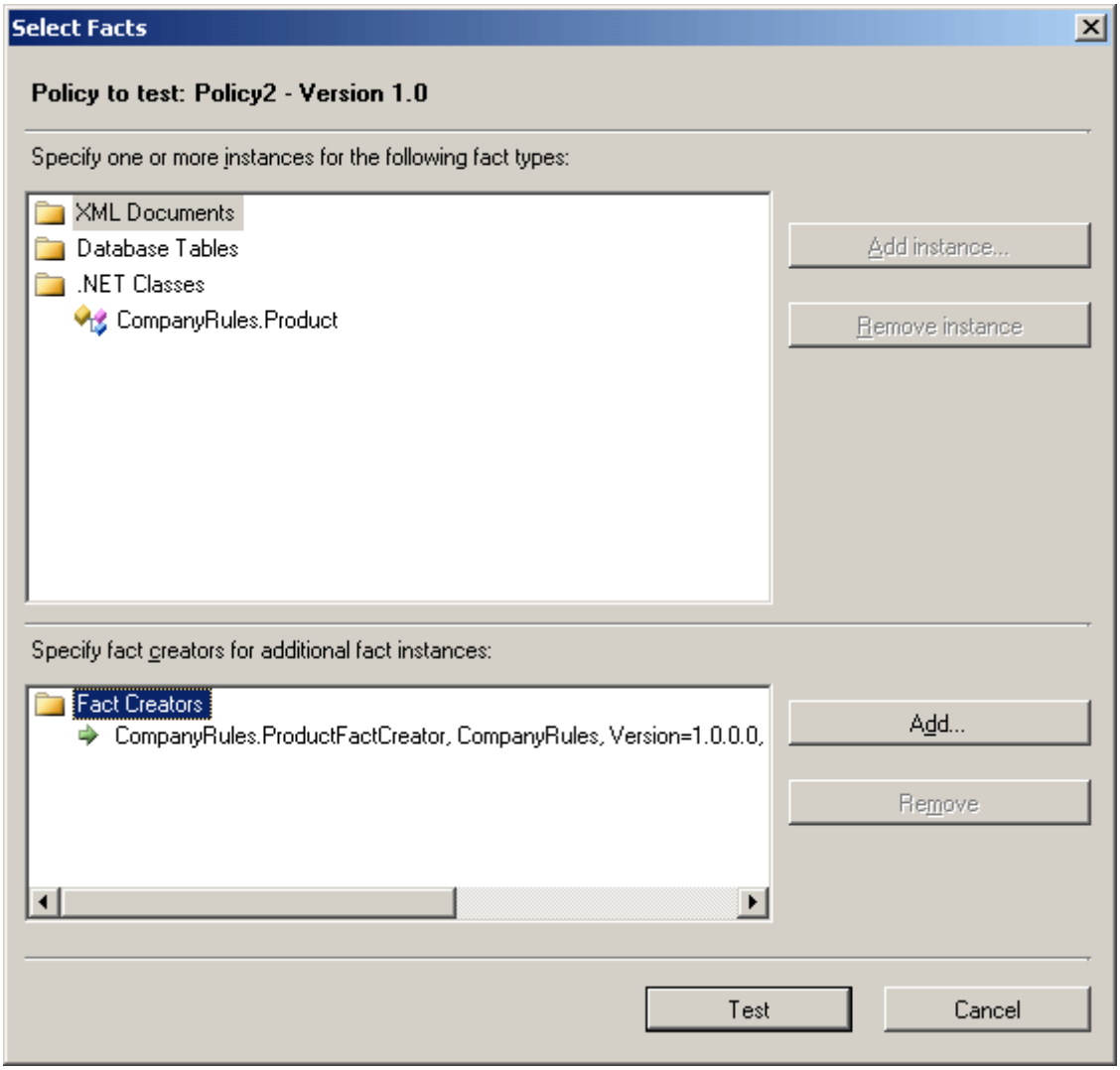
Policy2 içerisnde tanımlanan ilk kuralımız Rule1 ismindedir. Bu kurala göre Product nesne örneğinin Count özelliğinin değerinin 5000' in altında olması halinde StockLevelOk özelliğine False değeri atanmaktadır.



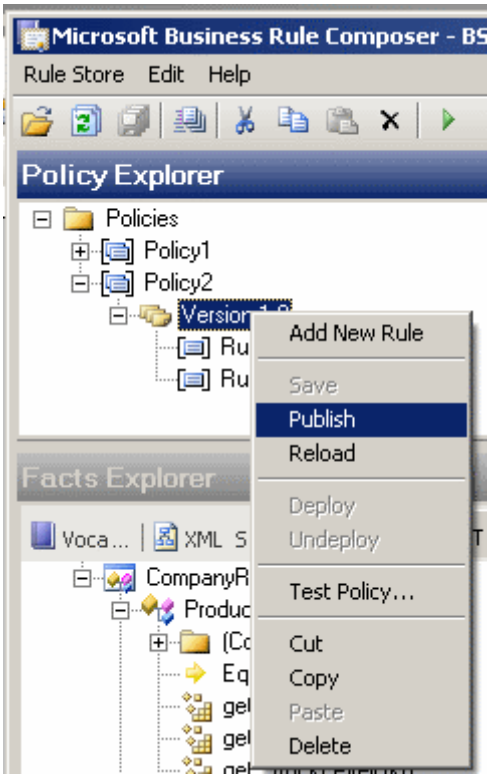
İkinci kuralımız(Rule 2)' da ise, birinci kuralın zıttı olan durum söz konusudur. Bu kez Count değerinin 5000' den büyük veya eşit olması halinde StockLevelOk özelliğine true değeri atanmaktadır.



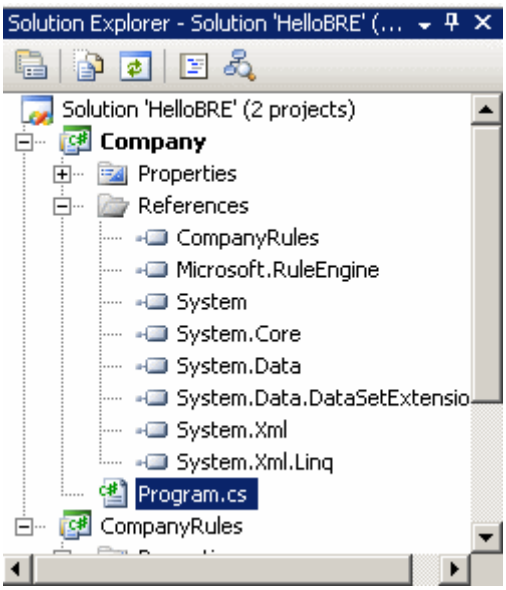
5 - Kurallar test edilir ve herşey beklendiği gibiyse, **dağıtım(Deployement)** aşamasına geçilir. **Test** için yapılması gereken ilk adım, tanımlanan **Rule'** lar üzerinde testi gerçekleştirecek olan **.Net** tipinin seçilmesidir. Buda **Test** düğmesine basıldığında bize sorulmaktadır ki yine **GAC** içerisinde duran **assembly** kütüphanemiz zaten söz konusu**IFactCreator** türevini içermektedir.



6 - Test edilen ve test sonuçları beklediğimiz gibi
çıkan **Policy** sırasıyla **Publish** ve **Deploy** işlemlerinden geçirilerek kullanıma hazır hale getirilir.



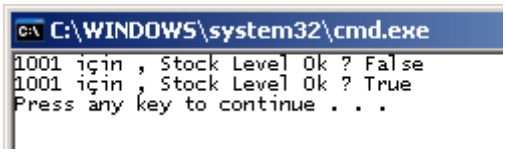
7 - Deploy edilen Policy' lerin ve içerdiği kuralların herhangibir .Net uygulamasında kullanılabilmesi için, söz konusu uyulamaya yine **Microsoft.RuleEngine.dll assembly'** ının referans edilmesi gerekir. Bu adıma gelinmeden önce, 5nci adımda yaptığımız testlerin sonuçlarının doğruluğundan emin olunmalıdır.



8 - Kural motorunu kullanacak olan .Net uygulamasında, **Microsoft.RuleEngine** isim alanı altında yer alan **Policy** tipinden yararlanılarak, **Fact** nesnesinin BRE içerisine atılması sağlanır. İşte Company isimli Console uygulamamızda yer alan kodlarımız.

```
using System;
using CompanyRules;
using Microsoft.RuleEngine;

namespace Company
{
   class Program
   {
      static void Main(string[] args)
      {
         Product prd = new Product { Count = 4999, ProductId = 1001 };

         Policy policy = new Policy("Policy2", 1, 0);
         policy.Execute(prd);

         Console.WriteLine("{0} için , Stock Level Ok ?
{1}",prd.ProductId,prd.StockLevelOk);

         prd.Count += 10;
         policy.Execute(prd);
         Console.WriteLine("{0} için , Stock Level Ok ? {1}", prd.ProductId,
prd.StockLevelOk);

      }
   }
}
```

Görüldüğü üzere ilk olarak Product nesnesi örneklenmektedir. Sonrasında bir **Policy** nesnesi örneklenir. Burada önemli olan bir noktada **Major** ve **Minor** versiyon numalarınında belirtilmesidir. Bu bize şöyle bir avantaj sağlayabilir; bir **Policy'** nin birden fazla versiyonu olması halinde, program içerisinde hangisinin kullanılacağını seçmemize olanak tanır. Hatta söz konusu değerleri(**Policy** adı, **Major** ve **Minor** numaraları) uygulamaya ait **konfigurasyon** dosyasından çekilebilir. Böylece kodun içerisinde kesinlikle girilmeden, **Policy** versiyonlaması ve hangi ilkelerin kullanılacağına karar verilmeside sağlanmış olur. *(Konfigurasyon kullanımını bu konunun görsel anlatımında gösteriyor olacağım)* Uygulamamızı çalıştırdığımızda aşağıdaki sonuçlar ile karşılaşırız.



Böylece geldik zevkli bir konunun daha sonuna. Umarım sizler içinde yararlı olmuştur. Geliştirdiğimiz örnek Biztalk Server 2006' ya ait Business Rule Engine' i kullanmaktadır. Ancak bildiğiniz üzere bir süre öncede Biztalk Server 2009 ürünü yayınlanmıştır.

Dolayısıyla bu konu ile ilişkili araştırmalarımı 2009 sürümü üzerinden devam ettiriyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HelloBRE.rar (37,09 kb)

Meraklısı için Business Rule Engine kavramı ile ilişkili detaylı bilgi

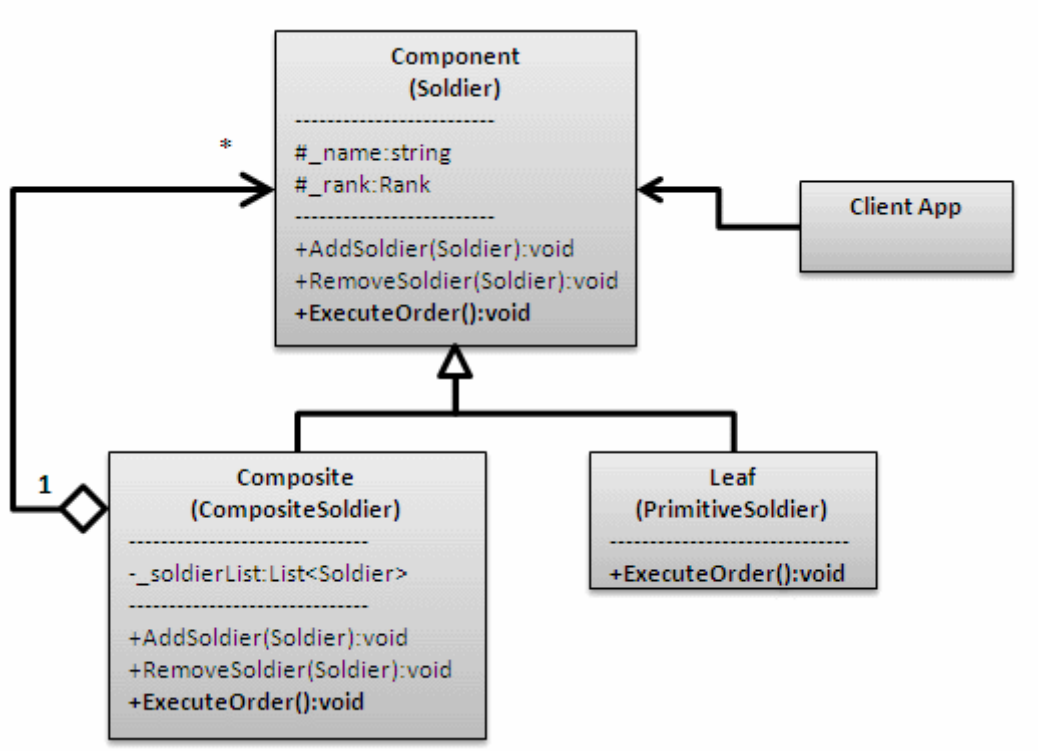
Tasarım Desenleri - Composite (2009-07-12T19:00:00)

design patterns,oop,c#,

Merhaba Arkadaşlar,

Küçüklüğümde son derece şanslı bir çocuktum. Uzun yıl
bana**Lego** oyuncaklarından göndermiştir. Evde günümün
geri kalırdım Lego oyuncakları zaman içerisinde öyle
konseptini içerdiğini görmeye başladık. Şimdi bunun kon
legolardan oluşan bir orduyu gözümde canlandırıyorum.

Aslında bu organizasyonda yer alan bireylerin hepsi birer asker. Rütbeleri farklı bile olsa. önemli olan detaylardan birisi bunların aralarındaki organizasyonel ilişkinin aslında bir **ağaç yapısı(Tree)** şeklinde ifade edilebiliyor olması. Bu ağaç ilişkisinden yararlanarak alt ve üstler arasında bilgi dolaştırılmasıda mümkün. Peki ya bu askerler **nesne yönelimli(Object Oriented)** bir programlama ortamında ifade ediliyorlarsa, organizasyonun ağaç yapısını temsil edebilecek bir kalıp mümkün olabilir mi? Tabiki olabilir ve bu kalıbın adı **Composite** tasarım desenidir. Aslında bu desenin temel amacı, nesnelerin ağaç yapısına göre düzenelenebilmesidir. Desenin içerisinde yer alan kahramanlar ise aşağıdaki örnek **UML** diagramında görüldüğü gibidir.



Component tipi içerisinde kendisinden türeyen **Composite** ve **Leaf** tiplerinin ortaklaşa kullanacağı üyeler dışında, ezmeleri gereken kurallarda tanımlanmaktadır. Bu anlamda **Component**bileşenini **abstract** bir sınıf veya **arayüz(Interface)** olarak tanımlayabiliriz. Şemadanda görüleceği üzere, **Component** tipi içerisinde yine **Component** tipinden parametre alıp ekleme ve çıkarma işlevlerini üstlenen ve **ezilen(Override)** metodlar vardır. Ancak bu metodlar Leaf tipi içerisinde ezilmemiştir. Aslında **Leaf** tiplerini, kendi içerisinde başka bir **Component** tipi içermeyecek çalışma zamanı nesnelerini örneklediğini düşünebiliriz. Bunun aksine **Composite** tipi, kendi içerisinde **Component** tipinden oluşan bir koleksiyon içermektedir. Bir başka deyişle, altında**Leaf** tiplerini veya başka **Composite** tipleri içerebilecek bir nesne üretimi sağlanabilmektedir.

Desende dikkat edilmesi gereken noktalardan biriside, **Component** tipinden tanımlanan bazı operasyonların **Composite** tipte uygulanırken **Leaf** tipinde uygulanmamasıdır. örneğe göre **AddSoldier** ve**RemoveSoldier** metodlarının **Leaf** tipi içerisinde bir anlamı yoktur. Nitekim **Leaf** kendi altında başka bir **Component** tipi içermeyecek nesne örneklerini temsil etmek üzere ele alınmalıdır. Diğer yandan **Component** tipi içerisinde tanımlanıp hem **Composite** hemde **Leaf** tipinde uygulanan ortak operasyonların, **Composite** içerisindeki uygulanış şeklide biraz farklıdır. Bu farka göre, **Composite** içerisindeki ortak operasyon(örneğimize göre **ExecuteOrder** metodu), **Composite** nesne örneğine bağlı tüm nesneleri kapsamalıdır. Sanırım kafamız iyice allak bullak oldu.

Bu nedenle olayı **XML** ağaçlarını düşünerektende ele alabiliriz. XML alt yapısını kod tarafında ifade ederken, **Composite** tasarım kalıbına göre bir ağacın **OOP'** ye uygun olacak

şekilde tasarlanması kolay olabilir. **XML** yapısı gereği, herkes birer **element(yada node)** olarak ifade edilebilirken, aslında kendilerine bağlı başka alt elementleride barındırabilirler. Böylece çalışma zamanında birbirlerine bağlı dallar üzerinde oturan **XML** ağaçları kolaylıkla tasarlanabilir. Hatta bir **Component** üzerinden varsa alt veya üst **Component** tipinede ulaşılabilir. *(Size tavsiyem bir XML verisinin içeriğindeki elementleri ifade edecek bir modeli, Composite desenine göre tasarlamaya çalışmanız olacaktır.)*

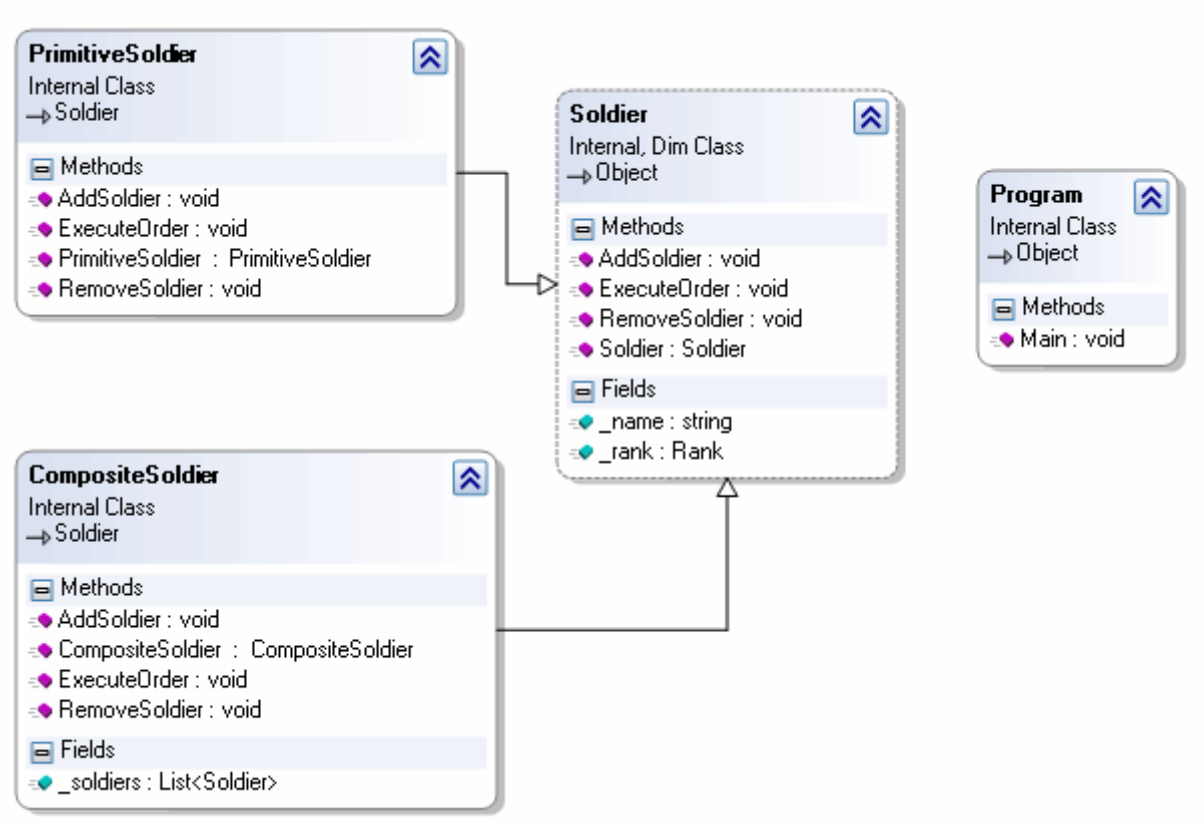
Ancak **yapısal(Stuctural)** desenelerden olan bu kalıpta dikkat edilmesi gereken en önemli nokta, ağaç içerisindeki tüm nesnelerin aslında **aynı arayüzü(veya soyut tipi)** uyguluyor olmasıdır. Bu nedenle nesne istemcileri, ağaçta yer alan **Composite** ve **Leaf** örneklerine aynı şekilde davranırlar.

Dilerseniz basit bir örnek üzerinden ilerleyelim. örneğimizde, kurduğumuz ordunun içerisindeki organizasyonel ağacı tasarlamaya çalışıyor olacağız. Buna göre

```
General
  Colonel
    LieutenantColonel
      Major
        Captain
          Lieutenant
```

şeklinde bir organizasyonumuz olduğunu göz önüne alabiliriz.

Organizasyondaki herkes bir **askerdir(Soldier)** ki buda bizim **Component** tipimiz ile ifade edilmektedir. **Composite** tipimiz(**CompositeSoldier**) isterse kendi içerisinde birden fazla başka**Component**(**PrimitiveSoldier** veya **CompositeSoldier** olabilir) tiplerini içerebilmelidir. Tüm askerlerin, ister **Leaf** ister **Composite** olsun uygulayacağı birde ortak operasyonumuz vardır(**ExcuteOrder**). İşte sınıf diagramımız ve uygulama kodlarımız.




```
using System;
using System.Collections.Generic;

namespace CompositePattern
{
 /// <summary>
 /// Askerlerin rütbeleri
 /// </summary>
 enum Rank
 {
  General,
  Colonel,
  LieutenantColonel,
  Major,
  Captain,
  Lieutenant
 }

 /// <summary>
 /// Component sınıfı
 /// </summary>
 abstract class Soldier
 {
  protected string _name;
```

```
  protected Rank _rank;

  public Soldier(string name,Rank rank)
  {
   _name=name;
   _rank=rank;
  }

  public abstract void AddSoldier(Soldier soldier);
  public abstract void RemoveSoldier(Soldier soldier);
  public abstract void ExecuteOrder(); // Hem Leaf hemde Composite tipi için
uygulanacak olan fonksiyon

 }

 /// <summary>
 /// Leaf class
 /// </summary>
 class PrimitiveSoldier
  :Soldier{

  public PrimitiveSoldier(string name,Rank rank)
   :base(name,rank)
  {

  }
  // Bu fonksiyonun Leaf için anlamı yoktur.
  public override void AddSoldier(Soldier soldier)
  {
   throw new NotImplementedException();
  }
  // Bu fonksiyonun Leaf için anlamı yoktur.
  public override void RemoveSoldier(Soldier soldier)
  {
   throw new NotImplementedException();
  }

  public override void ExecuteOrder()
  {
   Console.WriteLine(String.Format("{0} {1}",_rank,_name));
  }
 }

 /// <summary>
 /// Composite Class
```

```
/// </summary>
class CompositeSoldier
 :Soldier{

  // Composite tip kendi içerisinde birden fazla Component tipi içerebilir. Bu tipleri bir koleksiyon içerisinde tutabilir.
  private List<Soldier> _soldiers=new List<Soldier>();

  public CompositeSoldier(string name,Rank rank)
   :base(name,rank)
  {

  }

  // Composite tipin altına bir Component eklemek için kullanılır
  public override void AddSoldier(Soldier soldier)
  {
   _soldiers.Add(soldier);
  }
  // Composite tipin altındaki koleksiyon içerisinden bir Component tipinin çıkartmak için kullanılır
  public override void RemoveSoldier(Soldier soldier)
  {
   _soldiers.Remove(soldier);
  }
  // önemli nokta. Composite tip içerisindeki bu operasyon, Composite tipe bağlı tüm Component'ler için gerçekleştirilir.
  public override void ExecuteOrder()
  {
   Console.WriteLine(String.Format("{0} {1}",_rank,_name));
   foreach(Soldier soldier in _soldiers)
   {
    soldier.ExecuteOrder();
   }
  }
 }
 class Program
 {
  public static void Main(string[] args)
  {
   // Root oluşturulur.
   CompositeSoldier generalBurak=new CompositeSoldier("Burak",Rank.General);

   // root altına Leaf tipten nesne örnekleri eklenir.
   generalBurak.AddSoldier(new PrimitiveSoldier("Mayk",Rank.Colonel));
```

```
   generalBurak.AddSoldier(new PrimitiveSoldier("Tobiassen",Rank.Colonel));

   // Composite tipler oluşturulur.
   CompositeSoldier colonelNevi=new CompositeSoldier("Nevi", Rank.Colonel);
   CompositeSoldier lieutenantColonelZing=new CompositeSoldier("Zing",
Rank.LieutenantColonel);

   // Composite tipe bağlı primitive tipler oluşturulur.
   lieutenantColonelZing.AddSoldier(new PrimitiveSoldier("Tomasson",
Rank.Captain));
   colonelNevi.AddSoldier(lieutenantColonelZing);
   colonelNevi.AddSoldier(new PrimitiveSoldier("Mayro", Rank.LieutenantColonel));
   // Root' un altına Composite nesne örneği eklenir.
   generalBurak.AddSoldier(colonelNevi);

   //
   generalBurak.AddSoldier(new PrimitiveSoldier("Zulu",Rank.Colonel));

   // root için ExecuteOrder operasyonu uygulanır. Buna göre root altındaki tüm nesneler
için bu operasyon uygulanır
   generalBurak.ExecuteOrder();

   Console.ReadLine();
  }
 }
}
```

Uygulamayı çalıştırdığımızda aşağıdaki sonucu alırız.

Görüldüğü gibi emir modeli general üzerinden uygulandığı için organizasyonda generale bağlı olan herkese iletilebilmektedir. Aslında son derece kolay ve kullanım alanı geniş olan bir deseni inceledik. Ancak Console uygulamasıda olsa eksik olan kısımlar var gibi. Söz gelimi hiyerarşiye göre askerlerin ekrana girintili olarak yazdırılması sağlanabilir. Hatta bunu bir WPF veya Windows uygulamasında görsel olarak yapmaya çalışmanızı öneririm. Bu mukaddes görevleride sizlere bırakıyorum.

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Not: Desene ait görsel anlatım en yakın zamanda eklenecektir.

CompositePattern.rar (13,44 kb)

Tasarım Desenleri - Observer (2009-07-09T22:35:00)

design patterns,oop,c#,

Merhaba Arkadaşlar,

Yaklaşık olarak 10 yılı aşkın bir süredir yazılım teknolojileri ile ilg
a,**Visual Basic**' ten **Cobol**' a kadar pek çok programlama dili ile uğ
olursa olsun, hayatın ve dolayısıyla yazılım teknolojilerinin değişm

Ve birde platform, dil, cihaz gözetmeksizin var olan oyunlar.

Bende her ne kadar yazılımı bir hayat biçimi olarak benimsemiş olsamda, zaman zaman eski günlerdeki gibi strateji oyunlarını oynamıyor da değilim. **Warcraft 2** ve **"Yeş mi Lord"** ile başlayan bu serüvende, **Starcraft**' taki **Protosları**, **Red Alert**' taki **Yuri**' nin **Mind Control** makinelerini, **Generals**' taki **Stealth Fighter**' ları ve daha nicelerini kullanma fırsatım oldu. Geçen akşamda kafamda bir oyun canlandırmaya çalışırken, korumakta olduğum topraklara yapılan bir saldırıyı, filomdaki birimlere nasıl ileteceğimi düşünmeye başladım.*(çok doğal olarak telsizle bildir olsun bitsin diyebilirsiniz)* Bir saldırı anında tank, helikopter, yaya piyade veya deniz gücündeki operatörlere bu durumun iletilmesi gerekiyordu. Aslında birilerininde sürekli olarak dinlemede olması şarttı. Aslında bu birimlerin hepsi oyun sahası içerisinde yer alan birer nesne idi ve **OOP** tarafında tip olarak modellenebilirlerdi. Dolayısıyla aralarında **bire-çok(one to many)**ilişki olabilecek nesnelerin olması ve birisinde meydana gelecek değişikliklerin diğerlerine bildirilmesi gibi senaryo ortaya çıkmaktaydı.(Bir filonun komuta merkezinden, ona bağlı tüm birimlere haber gitmesi durumu)

Duruma farklı senaryolardan da bakabiliriz aslında. örneğin,

- bir stok takip sisteminde, stok hareketlerinde olan değişimlerin bayilere bildirimesi,
- haber ajanslarının, kendilerine bağlı olan bölümlere yeni başlıklar geldikçe bilgilendirmede bulunmaları,

gibi gerçek hayat senaryoları söz konusu olabilir.

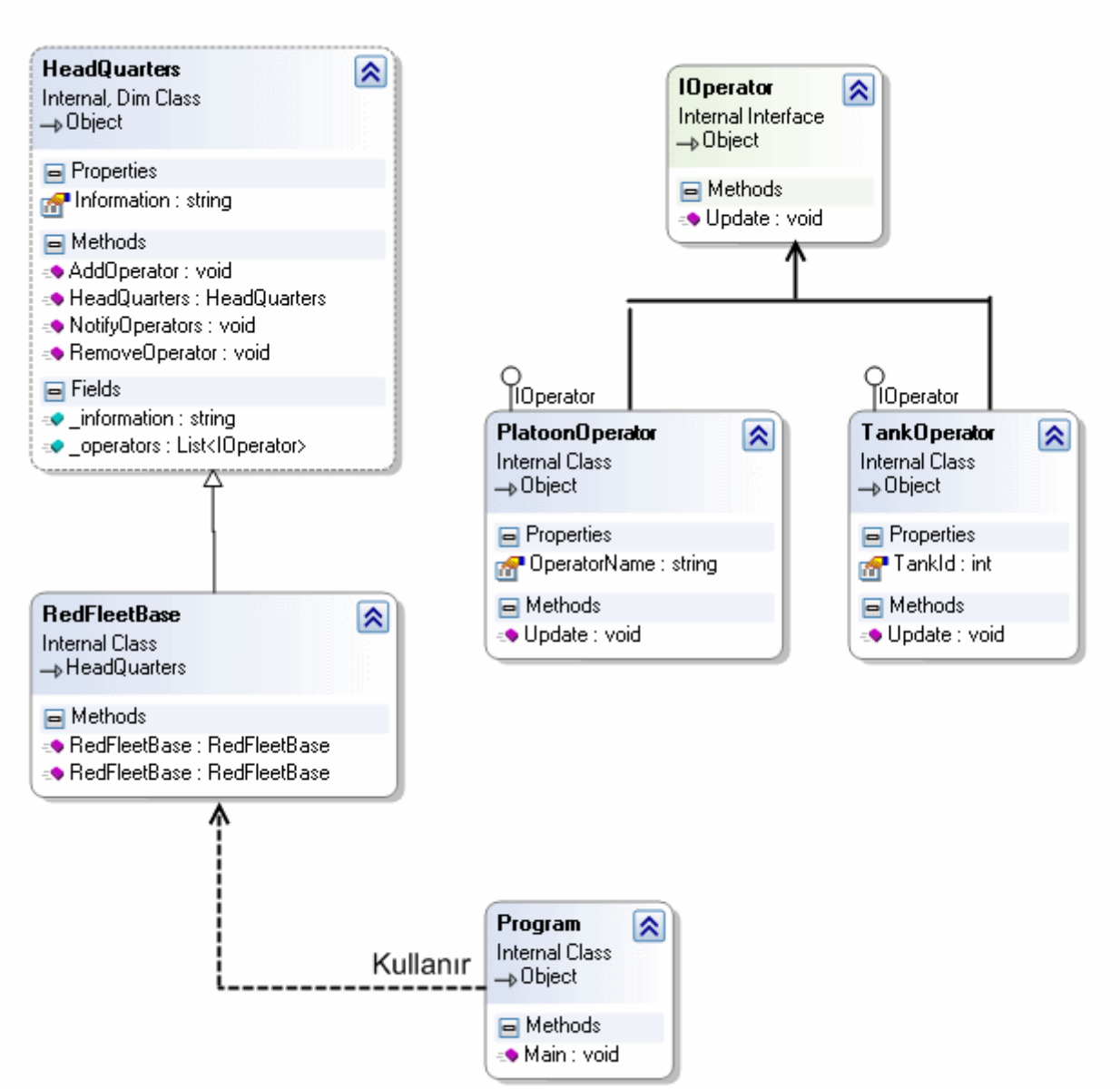
Tahmin edeceğini üzere varmakta olduğumuz nokta bu 3 basit senaryoda benzer vakanın ve ihtiyaçların olmasıdır. Buda bizi bir tasarım kalıbına götürmektedir. **Observer** tasarım deseni. Bu desen o kadar yaygındır ki onu;

- **Model View Controller(MVC)** içerisinde
- veya **.Net** ile **Java** tarafındaki **olay güdümlü programlamada(Event Based Programming)**

bulabiliriz.

Aslında bu desen servis yönelimli mimarilerde dahi karşımıza çıkmaktadır. özetle çok sık kullanılan bir kalıp olduğunu ifade edebiliriz. Desenin ilkesi az öncede belirttiğimiz üzere birbirlerine bire-çok ilişki ile bağlı nesnelerde olabilecek değişiklikleri diğerlerine iletmektir. Bu nedenle izlenmeye değer bir konu ve bununla ilişkili nesneler**(Subject, ConcreteSubject)** söz konusudur. Ayrıca konuyu takip edecek olan tiplerde**(Observer, ConcreteObserver)** bulunmaktadır.

Dilerseniz konumuza basit bir örnek ile devam edelim. Yazımıza giriş kısmında bahsettiğimiz asl alabiliriz. Senaryomuza göre konumuz, bir saldırı veya benzer durumlarda, istediğimiz bazı birlik bilgilendirmeyi yapacak olan **yayımcılar(Publishers)** ve bunu dinleyecek olan **aboneler(Subscr** deseni bir anlamda publisher/subscriber modelinin bir uygulanış biçimi olarakta düşünülebilir. İş



(Bu arada sınıf diagramı biraz farklı görünebilir. Nitekim şu anda bu yazıyı kayınvalidemin bilgisayarında yazmaktayım. Makine Visual Studio yok belki ama SharpDevelop sağolsun her ihtiyacımı görüyor. 🙂 *)*

Ve kodlarımız;

```
using System;
using System.Collections.Generic;

namespace Observer
{
   /// <summary>
   /// Subject class
   /// </summary>
```

```
internal abstract class HeadQuarters
{
   private string _information;
   private List<IOperator> _operators=null;

   protected HeadQuarters(string information)
   {
      _operators = new List<IOperator>();
      Information = information;
   }

   public string Information
   {
      get { return _information; }
      set {
         _information = value;
         NotifyOperators();
      }
   }

   public void AddOperator(IOperator opt)
   {
      _operators.Add(opt);
   }
   public void RemoveOperator(IOperator opt)
   {
      _operators.Remove(opt);
   }
   public void NotifyOperators()
   {
      foreach (IOperator opt in _operators)
      {
         opt.Update(this);
      }
   }
}

/// <summary>
/// Concrete Subject class
/// </summary>
internal class RedFleetBase
   :HeadQuarters
{
   public RedFleetBase(string information)
     :base(information)
```

```
    {

    }

    public RedFleetBase()
        :base("...")
    {

    }
}

/// <summary>
/// Observer class
/// </summary>
internal interface IOperator
{
    void Update(HeadQuarters headQuarters);
}

/// <summary>
/// Concrete Observer class
/// </summary>
internal class PlatoonOperator
    :IOperator
{
    public string OperatorName { get; set; }

    #region IOperator Members

    public void Update(HeadQuarters headQuarters)
    {
        Console.WriteLine("[{0}] : {1}",OperatorName,headQuarters.Information);
    }

    #endregion
}

/// <summary>
/// Concrete Observer Class
/// </summary>
internal class TankOperator
    : IOperator
{
    public int TankId  { get; set; }
    #region IOperator Members
```

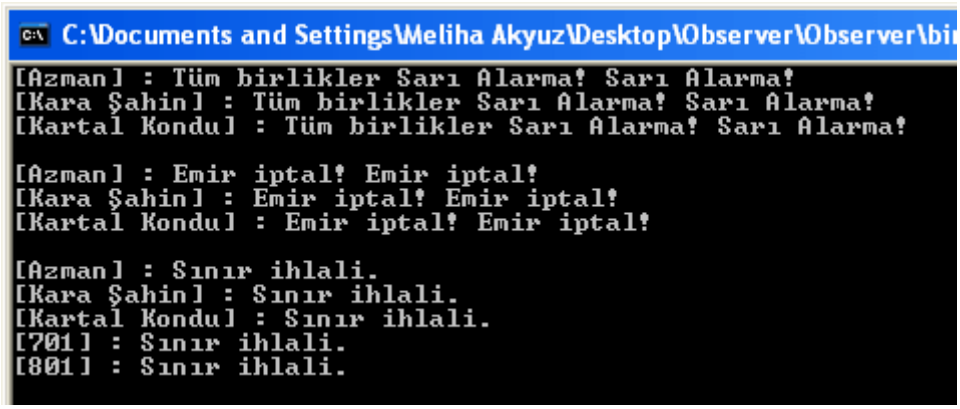
```
        public void Update(HeadQuarters headQuarters)
        {
            Console.WriteLine("[{0}] : {1}", TankId, headQuarters.Information);
        }

        #endregion
    }


    /// <summary>
    /// Client App
    /// </summary>
    class Program
    {
        static void Main()
        {
            RedFleetBase redFleetBase = new RedFleetBase {Information = "Süper işlemciler
piyasada"};
            redFleetBase.Information = "İşlemciler gelişiyor";

            redFleetBase.AddOperator(new PlatoonOperator { OperatorName="Azman"}
);
            redFleetBase.AddOperator(new PlatoonOperator { OperatorName = "Kara
Şahin"});
            redFleetBase.AddOperator(new PlatoonOperator { OperatorName="Kartal
Kondu"});

            redFleetBase.Information = "Tüm birlikler Sarı Alarma! Sarı Alarma!";

            Console.WriteLine("");

            redFleetBase.Information = "Emir iptal! Emir iptal!";

            Console.WriteLine("");

            redFleetBase.AddOperator(new TankOperator{TankId=701});
            redFleetBase.AddOperator(new TankOperator{TankId=801});
            redFleetBase.Information = "Sınır ihlali.";
        }
    }
}
```

Biraz kodları incelemekte fayda olduğu kanısındayım...

Subject tipimiz olan **HeadQuarters**, **Information** isimli bir özelliğe sahiptir. Bu özellik, **Observer** tiplerinin değerlendireceği bir bilgidir. Bir başka

deyişle, **HeadQuarters'** tan, **IOperator** türevlerine aktarılan bildirimdir. **HeadQuarters abstract** tipi içerisinde bilgilendirilme yapılacak **IOperator** türevleride bir koleksiyon içerisinde saklanmaktadır. Buna göre, **HeadQuarters** içerisindeki **Information**özelliğinde yapılacak bir değişilik sonrası devreye giren **Set** bloğundan, **NotifyOperators** isimli metod çağırılmaktadır. Dikkat edileceği üzere söz konusu metod, **IOperator** tipinden olan **List** koleksiyonundaki tüm türevlerin **Update** metodunu çağırmakta ve çalışma zamanındaki **HeadQuarters** nesne referansını alt sınıflara göndermektedir. Dolayısıyla, bilgilendirme yaplılacak **IOperator** türevlerinin söz konusu **Subject** tipi içerisinde saklanması için ekleme ve çıkarma operasyonları da **HeadQuarters** sınıfına dahil edilmiştir.

Tabiki **istemcinin(Console uygulamamız)** asıl kullanacağı tip **Subject'** ten türeyen **ConcreteSubject** sınıfıdır(**RedFleetBase**). İstemci, bu tip üzerinde abonelerini bildirmektedir. Yani**IOperator(Observer)** arayüzünü implemente eden **PlatoonOperator** ve **TankOperator** nesne örneklerini...Kodun içerisinde **RedFleetBase** nesne örneği üzerinden yapılan her bir **Information**değişikliği sonrası, ne kadar tank ve piyade operatörü varsa bilgilendirilmektedir. Aynen aşağıdaki örnek çalışma zamanı çıktısında olduğu gibi.



Modelde dikkat edilmesi gereken noktalardan biriside, Subject ve Observer tiplerinden istenildiği kadar türetme yapılabilmesi ve bunların aralarında kuvvetli bir bağ ilişkisi olmamasıdır. Burada Subject tipinin kendi içerisinde soyut bir Observer tip koleksiyonu kullanmasınında büyük bir anlamı vardır. Böylece geldik bir tasarım kalıbımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Görsel anlatımları için C#Nedir? / NedirTV? adreslerini kullanabilirsiniz.

Observer.rar (29,47 kb)

Tasarım Desenleri - Prototype (2009-07-07T08:35:00)

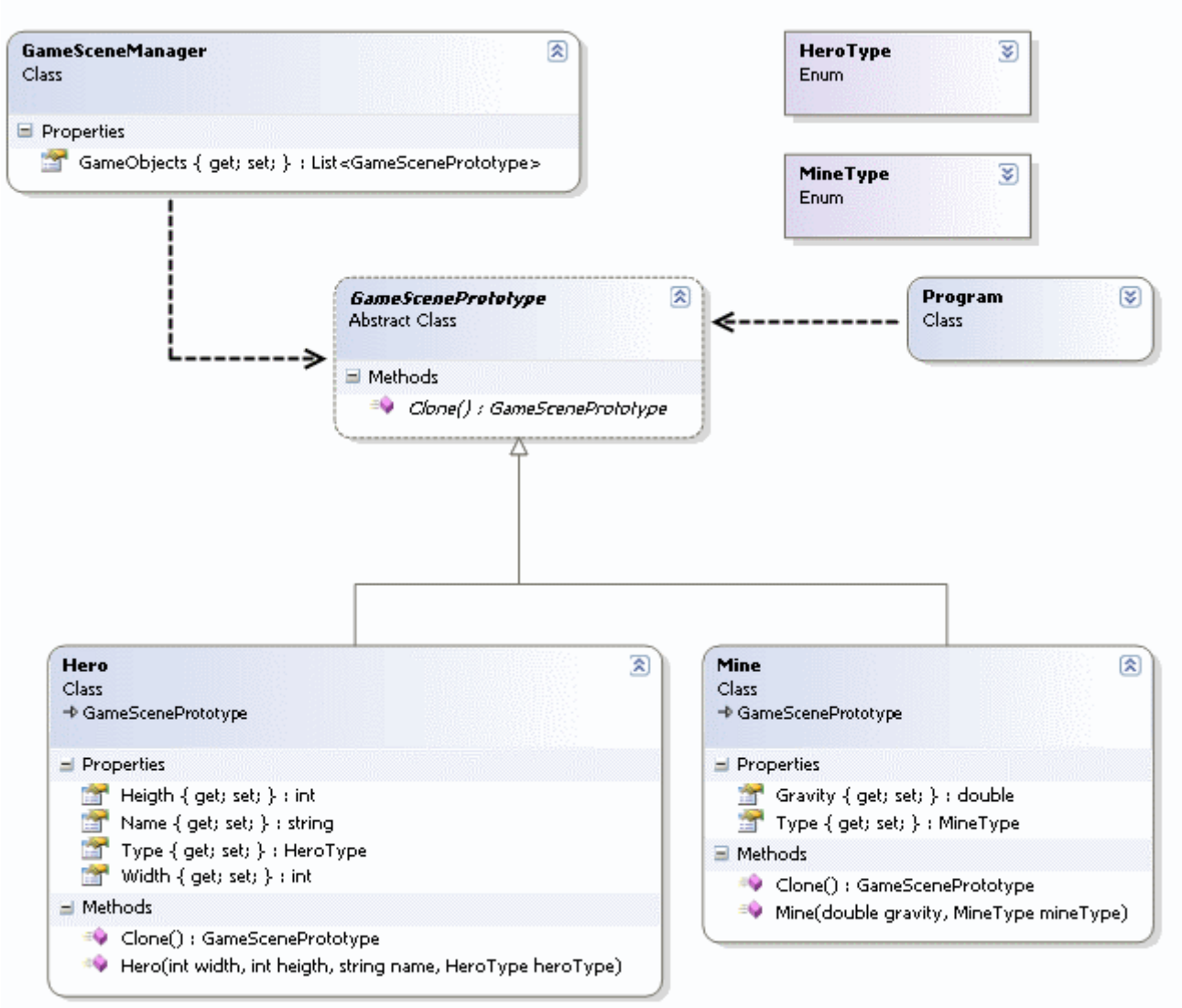
design patterns,oop,c#,

Merhaba Arkadaşlar,

Yandaki resimde **118 WallyPower** isimli tekneyi*(yada suda giden uzay mekiği)* görmektesiniz. E
hatırlamaya çalışın. Seyredenler **The Island** filmi olduğunu hemen bulacaktır. The Island filminc
görüpte ustaca çizdiği*(sanki çöpten adam çiziyormuşçasına)* bu teknenin senaryodaki adı ise **Ren**

Bu arada filmin ana fikri, ileride yarın öbür gün organ ihtiyacı olabilecek olan insanların birer klonunun üretildiği ve tutulduğu gizli bir yer altı şehri üstüne kuruluydu. Klon demişken var olan insanların birebir kopyasının üretildiği bir üs olduğunu belirteyim. E haliyle yeni klonların üretimi bir insanın doğum sürecine göre(filme gereği) çok daha kısa sürede olabilmektedir. Peki nerden geldik bu konuya...Aslında nesne yönelimli tarafta da, **üretimi pahalı olan nesneler** söz konusu olduğunda ve **new** operatörü ile oluşan **maaliyetten kaçınmak**istendiğinde, klon nesnelerin üretilmesi yolu tercih edilebilir. Bu durum zaman içerisinde pek çok nesne yönelimli projede ortaya çıkınca haliyle kalıplaşmış ve bir desen haline gelmiştir.**Creational** tasarım kalıplarından olan **Prototype** deseni. Bu kalıp gerçek hayat uygulamalarının pek çok noktasında karşımıza çıkabilir. Söz gelimi bir oyun sahnesinin tekrar eden nesne üretimlerinde, oluşturulma maaliyetlerinin azaltılmasına etki edebilir *(öyleki oyun sahnesi içerisindeki sabit olan pek çok yapının nesnel olarak ifadesi sırasında bu maliyetler oldukça yükselmektedir.)* Yada finansal veriler üzerine analiz gerçekleştiren bir sistemde, aynı veri kümesinden hareket edeceğimiz durumlarda, veriyi içeren nesne üretimlerinin maaliyetleri en aza indirgenebilir. Senaryolar çoğaltılabilir ancak özünde, nesne oluşturulma maaliyetleri vardır.

Dilerseniz durumu basit bir örnek üzerinde incelemeye gayret edelim. Az önce bahsettiğimiz senaryolardan oyun sahası problemini çok basit bir seviyede ele alıyor olacağız. Buna göre bir oyun sahasında yer alan kahraman ve mayınların birden fazla sayıda üretiminde aynı olan bazılarından yararlanıldığı düşünülmektedir. Sınıf diagramımızı aşağıdaki gibi tasarlayabiliriz.



Görüldüğü gibi klonlama operasyonu bir **abstract tip(veya interface)** içerisinde bildirilmektedir. Bu abstract tip prototipimizdir. **Prototype** tipten türeyen **Hero** ve **Mine** sınıfları ise kullanılacak asıl nesne örneklerini modellemektedirler**(Concrete Prototype)**. Buna göre oyun sahasını yöneten **GameSceneManager** sınıfı kendi içerisinde, **Prototype** tipten oluşan bir koleksiyonu kullanmaktadır. *(Bu sayede farklı prototip tiplerinin sisteme kolayca eklenebilmesinin yoluda açılmıştır.*) Peki **Clone** operasyonu nasıl gerçekleştirilecektir. Burada pek çok nesne yönelimli dilde yer alan bazı yardımcı metodlardan faydalanılabilir. Söz gelimi **.Net** tarafında **MemberwiseClone** fonksiyonu kullanılabilir. Buna göre uygulama kodlarımız aşağıdaki gibidir.

```
using System;

namespace Prototype
{
   // Prototype Class
   abstract class GameScenePrototype
```

```
{
   public abstract GameScenePrototype Clone();
}

// Concrete Prototype Class A
class Hero
   :GameScenePrototype
{
   public int Width { get; set; }
   public int Heigth { get; set; }
   public string Name { get; set; }
   public HeroType Type { get; set; }

   public Hero(int width,int heigth,string name,HeroType heroType)
   {
      Width = width;
      Heigth = heigth;
      Name = name;
      Type = heroType;
   }

   public override GameScenePrototype Clone()
   {
      return this.MemberwiseClone() as GameScenePrototype;
   }
}

// Concrete Prototype class B
class Mine
   :GameScenePrototype
{
   public double Gravity{ get; set; }
   public MineType Type { get; set; }

   public Mine(double gravity,MineType mineType)
   {
      Gravity = gravity;
      Type = mineType;
   }

   public override GameScenePrototype Clone()
   {
      return this.MemberwiseClone() as GameScenePrototype;
   }
}
```

```
// Prototype Manager class
  class GameSceneManager
  {
     public List<GameScenePrototype> GameObjects { get; set; }
     public GameSceneManager()
     {
        GameObjects = new List<GameScenePrototype>();
     }
  }

  #region Yardımcılar

  enum HeroType
  {
     Warrior,
     Employee,
     Archer
  }

  enum MineType
  {
     Gold,
     Silver,
     Bronze
  }

  #endregion

  class Program
  {
     static void Main()
     {
        GameSceneManager manager = new GameSceneManager();

        Hero hero1 = new Hero(10,20,"Bıkanyus", HeroType.Archer);
        manager.GameObjects.Add(hero1);
        Hero hero2 = new Hero(15, 35, "Wah!tupus", HeroType.Employee);
        manager.GameObjects.Add(hero2);

        Mine mine1 = new Mine(3, MineType.Gold);
        manager.GameObjects.Add(mine1);
        Mine mine2 = new Mine(5, MineType.Silver);
        manager.GameObjects.Add(mine2);
```

```
            // Var olan Mine ve Hero nesne örneklerinden klonlama yapılır
            manager.GameObjects.Add(mine2.Clone() as Mine);
            manager.GameObjects.Add(hero1.Clone() as Hero);
        }
    }
}
```

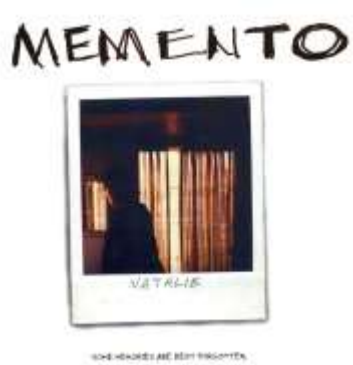
Prototype tip görevini üstlenen **GameScenePrototype abstract** sınıfı içerisinde tanımlanmış olan **Clone** metodu, kendi tipinden bir nesne referansını geriye döndürmek üzere planlanmıştır. Buna göre **GameScenePrototype** sınıfından türeyen **Mine** ve **Hero** isimli sınıflarında kendi içerisinde yer alan Clone metodlarında benzer davranışı göstermeleri gerekir. Burada **MemberwiseClone** metodundan yararlanılarak deep copy işleminin yapılması ve var olan nesnenin tüm içeriği ile birlikte klonlanması sağlanmaktadır. Tabiki MemberwiseClone metodu object tipinden bir referans döndürdüğü için **açık bir şekilde(Explicit)** tip dönüşümü yapılması gerekmektedir. çalışma zamanında örnek olaran mine2 ve hero1 isimli nesne örneklerinin klonlanması işlemi ele alınmıştır. Böylece geldik bir bölümün daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Görsel anlatımları için C#Nedir? / NedirTV? adreslerini kullanabilirsiniz.

Prototype.rar (22,60 kb)

Tasarım Desenleri - Memento (2009-07-06T08:45:00)

design patterns,oop,c#,

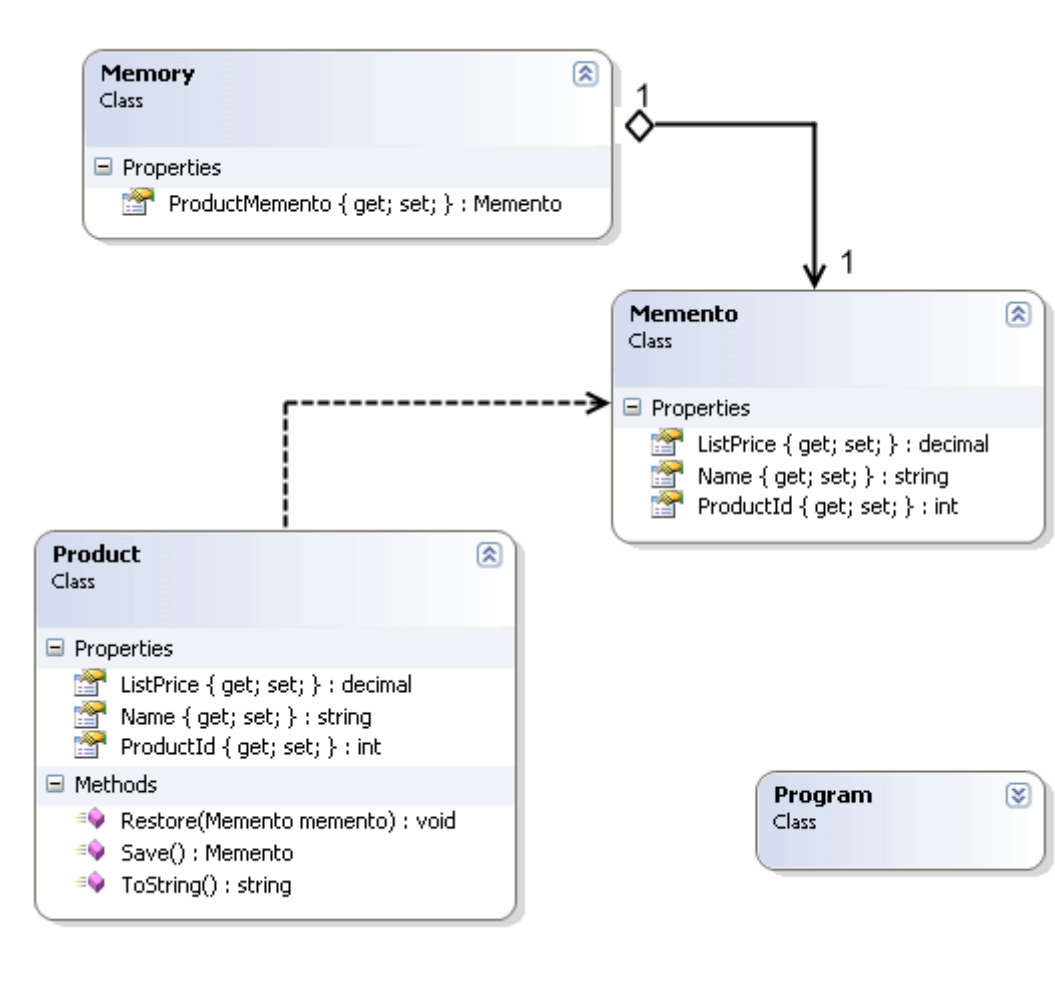


Merhaba Arkadaşlar,

Sanıyorum yandaki resmi görenler **Guy Pearce** ve **Carrie Anne Mos** arasında yer alan bu film, tersten ilerlemesi bir yana herşeyi unutan v hikayesi ile ilgiliydi. çok şükürki **nesne yönelimli(Object Oriented)** vücutlarına dövme yaptırmasına gerek yoktur. İşte bu günkü konu

çok sık kullanılmamakla birlikte*(en azından dofactory.com istatistiklerine göre %20' ler seviyesinde)* oluşturulması ve kullanılması kolay olan bu desen, **davranışsal(Behavioral)** kalıplar arasında yer almaktadır. Esas itibariyle bir nesnenin daha önceki halinin(hallerinin) saklanması ve istenildiğinde tekrardan elde edilmesi üzerine tasarlanmış bir kalıptır. Nesnelere, dahili durumları için**(Initial State)** geri alma işlemi**(Undo)** yeteneğininin kazandırılması olarak da düşünebiliriz. Bu kalıpta durumu korunmak istenen nesnenin birebir veya en azından saklanmak istenen

alanlarını(özelliklerini) tutan kopyası yer alır**(Memento)**. Diğer taraftan memento nesnesini oluşturan, bir başka deyişle kaydeden yada var olduğu son durumu yükleyerek geri getiren fonksiyonelliklere sahip olan asıl tipimiz yer almaktadır**(Originator)**. Bu temel tiplerin yanında, saklanan memento nesnesinin güvenli bir şekilde korunmasını sağlayacak bakıcı bir tipde yer almaktadır**(Caretaker)**. Şimdi tabi olaya bu şekilde bakınca(şekilsiz olarak) anlamak zor olabiliyor. Yaşasın UML şemaları diyerek yola devam etmek lazım. Bu amaçla örnek Console uygulamamızın sınıf diagramına bir bakalım.



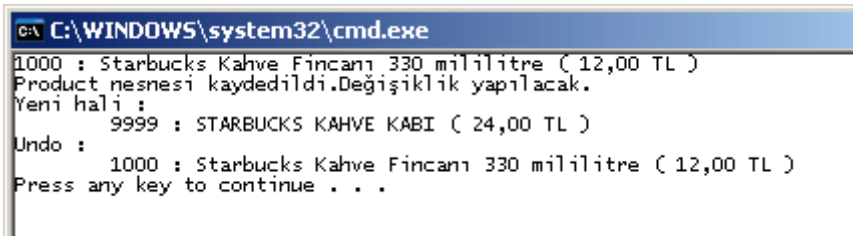
Uygulamanın çok basit bir amacı var. Product tipinden bir nesne örneğinin istenildiği durumda bir önceki haline döndürülmesi. Dikkat edileceği üzere **Product** tipinin özelliklerinin birebir aynısını içeren **Memento** isimli bir sınıf bulunmaktadır. Bu sınıf, Product nesne örneğinin herhangibir anda saklanmak istenen dahili içeriğini tutmak amacıyla oluşturulmuştur. Product isimli sınıfımız içerisinde de Memento nesne örneğini oluşturan **Save** ve tekrar geri alıp dolduran **Restore** metodları yer almaktadır. Memory tipi ise, Memento nesne örneğinin güvenli bir şekilde saklamasından sorumludur. Buna ek olarak Memento nesnesi üzerinde hiç bir operasyon gerçekleştirilmesine izin vermemektedir. Gelelim uygulama kodlarımıza.

```
using System;
```

```
namespace MementoPattern
{
   // Originator Class
   // Yaratıcı sınıf
   class Product
   {
      public int ProductId { get; set; }
      public string Name { get; set; }
      public decimal ListPrice { get; set; }

      // O anki Product nesne örneğinin içeriğini yeni bir Memento nesne örneğinde toplar ve bunu dış ortama verir.
      public Memento Save()
      {
         return new Memento {
            ProductId = this.ProductId
            , Name = this.Name
            , ListPrice = this.ListPrice
         };
      }

      // Saklanan Memento nesne örneğini alarak o anki Product nesne örneğinin dahili içeriğinin doldurulmasında kullanılır.
      public void Restore(Memento memento)
      {
         this.ListPrice = memento.ListPrice;
         this.Name = memento.Name;
         this.ProductId = memento.ProductId;
      }

      public override string ToString()
      {
         return String.Format("{0} : {1} ( {2} )", ProductId, Name, ListPrice.ToString("C2"));
      }
   }


   // Memento Class
   // Akıl defteri sınıfı
   // Product tipi içerisinden saklanmak amacıyla kullanılacak tüm özellikleri tanımlar
   class Memento
   {
      public int ProductId { get; set; }
      public string Name { get; set; }
```

```
    public decimal ListPrice { get; set; }
}

// Caretaker class
// Bakıcı sınıf
class Memory
{
    public Memento ProductMemento { get; set; }
}

class Program
{
    static void Main(string[] args)
    {
        // örnek bir Product nesnesi oluşturulur

        Product prd = new Product
        {
            ProductId = 1000,
            Name = "Starbucks Kahve Fincanı 330 mililitre",
            ListPrice = 12
        };
        Console.WriteLine(prd.ToString());

        // Caretaker nesnesi oluşturulur.
        Memory memory = new Memory();
        // Memento nesnesi içeriği o anki Product örneğinden elde edilir.
        memory.ProductMemento = prd.Save();
        Console.WriteLine("Product nesnesi kaydedildi.Değişiklik yapılacak.");

        prd.ProductId = 9999;
        prd.Name = "STARBUCKS KAHVE KABI";
        prd.ListPrice = 24;
        Console.WriteLine("Yeni hali : \n\t{0}", prd.ToString());

        // Restore işlemi gerçekleştirilir
        prd.Restore(memory.ProductMemento);
        Console.WriteLine("Undo : \n\t{0}",prd.ToString());
    }
}
}
```

Ve uygulamamızın çalışma zamanındaki durumu.



Görüldüğü gibi Product nesne örneği oluşturulup içeriği üzerinde değişiklik yapıldıktan sonra bir önceki konumuna döndürülebilmiştir. Oldukça basit ama etkileyici olan bu deseni, nesne örnekleri üzerinde Undo operasyonunun kullanılması istenen pek çok senaryoda ele alabiliriz. Tabi burada dikkat edilmesi gereken bir hususda vardır. Sadece tek bir Undo işlemi yapılabilmektedir. Oysaki istenirse **birden fazla adım geriye gidilmesi de** sağlanabilir. Bunu bir düşünmenizi ve yapmaya çalışmanızı öneririm Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Görsel Anlatımları(NedirTv? , C#Nedir?)

MementoPattern.rar (23,01 kb)

Tasarım Desenleri - Strategy (2009-07-03T10:34:00)

design patterns,oop,c#,

Merhaba Arkadaşlar,

Bir süredir **tasarım prensiplerini(Design Principles)** incelemeye çalış
çoğu prensip kendi içerisinde çeşitli **tasarım desenlerini(Design Patte**
daha önceden bakmışsak bile sık sık tekrar etmekte yarar var. Ben bu g
desenini aktarmaya çalışacağım.

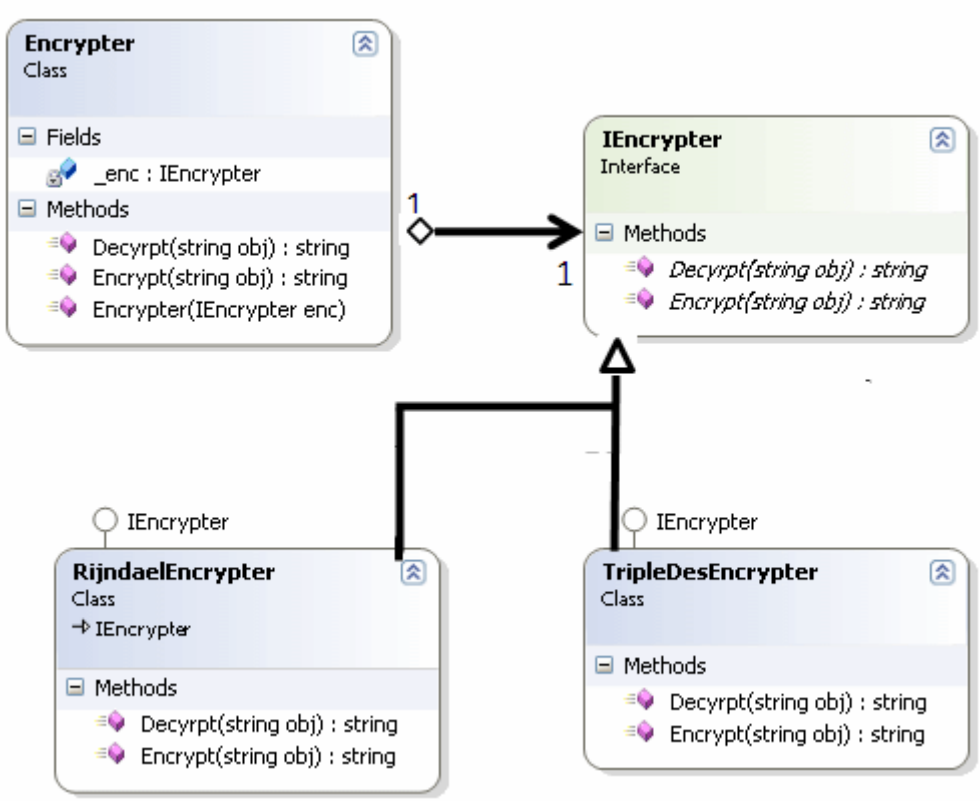
Görsel anlatımı için tıklayın, indirin.

Strategy deseni temel olarak, bir nesnenin her hangibir operasyonu gerçekleştirmek için kullanabileceği farklı algoritmaları içeren farklı tipleri kendi içerisinde ele alarak kullanması yerine, kullanmak istediği politikayı nasıl uygulandığını bilmesine gerek kalmaksızın sadece seçerek çalışma zamanında yürütmesine olanak tanımaktadır...Ughhh!!! Dediğimde sanıyorumki kafamızda pek bir şey oluşmamıştır. Hiç dert etmeyin bende bu deseni ilk öğrendiğim yıllarda bu tip cümleleri okurken etrafıma, havaya, doğaya şöyle bir bakıp kavramak için çaba harcardım. Bu nedenle gelin olayı önce senaryolaştıralım ve sonrasında bu cümleyi anlamaya çalışalım.

Elimizde string tipte verileri çeşitli algoritmalara göre şifreleyen ve tekrar eski haline getiren bir **içerik tipi(Context Type)** olduğunu varsaylım. çok doğal olarak burada iki ana operasyon söz konusudur. Bunlardan ilki

veriyi **şifrelemek(Encryption)** diğeride **çözümlemek(Decryption)** olarak düşünülebilir. Ne var bu string içeriklerin şifrelenmesi ve çözümlenmesi sırasında farklı tipte algoritmalar kullanılmak istenebilir. örneğin **Rijndael, Triple Des, SHA** vb... Bu durumda ilk akla gelen içerik tipi içerisinde söz konusu şifreleme seçeneklerini ele almaktır. Bu da bir sürü if veya switch ile olayı kontrol altına almak anlamına gelebilir. Ancak kaybettiğimiz önemli değerler vardır. Esneklik, genişletilebilirlik, test edilebilirlik vb...örneğin, veriyi yeni bir algoritmaya göre (SHA 512 mesela) şifrelemek istediğimizde, içerik tipinin üzerinde kod değişikliği yapmamız gerekecektir. Halbuki içerik tipinin çalışma zamanında kullanıldığı yerde, sadece şifreleme algoritmasını seçmesinin sağlanması bunun önüne geçebilir. Dikkat etmemiz gereken noktalardan biriside, içerik tipinin şifreleme algoritmasının nasıl yapıldığının bilmesine gerek olmayışıdır. Diğer yandan bunu bilmeyecek ise nasıl çağıracaktır. Daha da önemlisi, istediğimizde yeni bir algoritmayı içerik tipinde değişiklik yapmadan sisteme nasıl ekleyebiliriz.

Tüm bu soruların cevabı aşağıdaki sınıf diagramında ve kod parçasında yer almaktadır.



Kod içeriğimize gelince;

```
using System;

namespace StrategyPattern
{
   // Strategy type
   interface IEncrypter
```

```
{
   string Encrypt(string obj);
   string Decyrpt(string obj);
}

// ConcreteStrategy type 1
class RijndaelEncrypter
   : IEncrypter
{
   #region IEncrypter Members

   public string Encrypt(string obj)
   {
      Console.WriteLine("obj için Rijndael şifreleme");
      return obj;
   }

   public string Decyrpt(string obj)
   {
      Console.WriteLine("obj için Rijndael ters şifreleme");
       return obj;
   }

   #endregion
}

// ConcreteStrategy type 1
class TripleDesEncrypter
   : IEncrypter
{
   #region IEncrypter Members

   public string Encrypt(string obj)
   {
      Console.WriteLine("obj için TripleDES şifreleme");
      return obj;
   }

   public string Decyrpt(string obj)
   {
      Console.WriteLine("obj için TripleDES ters şifreleme");
      return obj;
   }
```

```
        #endregion
    }

    // Context Type
    class Encrypter
    {
        IEncrypter _enc=null;

        public Encrypter(IEncrypter enc)
        {
            _enc = enc;
        }

        public string Encrypt(string obj)
        {
             return _enc.Encrypt(obj);
        }
        public string Decyrpt(string obj)
        {
            return _enc.Decyrpt(obj);
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            string str =
"<app><config><sqlConnection>data....</sqlConnection></config></app>";

            Encrypter enc1 = new Encrypter(new TripleDesEncrypter());
            string encryptedStr=enc1.Encrypt(str);
            string decryptedStr = enc1.Decyrpt(str);

            enc1 = new Encrypter(new RijndaelEncrypter());
            encryptedStr = enc1.Encrypt(str);
            decryptedStr = enc1.Decyrpt(str);
        }
    }
}
```

Kodu incelediğinize göre biraz üzerinde konuşalım. Context tipimiz(Encrypter sınıfı) kendi içerisinde **IEncrypter** isimli bir interface kullanmakta ve bu arayüz üzerinde **Encrypt** ile **Decrypt**metodlarını çağırmaktadır. Bu bize şu esnekliği sağlamaktadır. **IEncrypter** arayüzünü uygulayan herhangibir tipi kendi içerisinde istediği

şifreleme algoritmasını uygulayabilir. Context sınıfının bunu düşünmesine gerek yoktur. Bir başka deyişle, **Context** sınıfı ile asıl işi yapan algoritma tipleri arasında bir bağımlılık oluşmasına engel olunmaktadır. Buna göre herhanbiri zamanda, context tipinin kullanabileceği yeni bir şifreleme algoritması sisteme eklenebilir. Tek yapılması gereken IEncrpyter arayüzünü implemente eden bir sınıfın yazılması ve çalışma zamanında bu tipin kullanılacağının söylenmesidir. Dikkat edileceği üzere Context tipi kendi içerisinde stratejik nesneye ait refaransı ele almaktadır. Dolayısıyla, arayüzlerin **polimorfik** özellikte olmaları nedeniyle, Context tipi içerisinde yer alan **Encrpyt** ve **Decrypt** metodları, **yapıcı metod(Consturctor)** ile çalışma zamanında gelen tip ne ise ona göre şifreleme ve çözümleme işlemlerini uygulayacaktır. Sanıyorumki şu anda ilk başta söylediğim o karışık cümle biraz daha anlaşılır hale gelmiştir.

Tabi tam bu noktada insanın aklına **C# 3.0** ve bazı şeytanlıklarda gelmiyor değil. öyleki **C# 3.0'** da **lambda** operatatörümüz, **Func<>** gibi temsilcilerimi bulunmakta. Bu durumda yukarıdaki desenin C# 3.0 daki yetenekler ile yazmaya çalıştığımızda belki aşağıdaki gibi bir uygulanış şeklininde söz konusu olabileceğini söyleyebiliriz. İşte Encrypter tipimizin ikinci versiyonu.

```
class EncrypterV2
   {
      public string Encrypt(Func<string,string> function,string obj)
      {
         return function(obj);
      }
      public string Decyrpt(Func<string, string> function, string obj)
      {
         return function(obj);
      }
   }
```

Görüldüğü gibi Encrypt ve Decrypt fonksiyonlarımız **Func<string,string>** tipinden bir **temsilciyi(delegate)** parametre olarak almakta ve içeride uygulamaktadır.Buna göre **EncyrpterV2** sınıfımızı kullanacağımız yerde, şifreleme ve ters şifreleme fonksiyonlarının kendimiz yazıp vermeliyiz. *(Gerçi bu durumda Context tipinin, algoritmaların nasıl çalıştığı ve yapıldığını bilmesine gerek olmayışı ilkesi ile çelişilmektedir. Bunada dikkat edelim)* Yani Context tipini çalışma zamanı için aşağıdaki gibi kullanabiliriz.

```
string str = "<app><config><sqlConnection>data....</sqlConnection></config></app>";

        EncrypterV2 v2 = new EncrypterV2();

        v2.Encrypt(s =>
        {
           Console.WriteLine("{0}\n için TripleDes şifreleme yapılıyor\n", s);
```

```
    return s;
},str);

v2.Decyrpt(s =>
{
    Console.WriteLine("{0}\n için TripleDes çözümleme yapılıyor\n", s);
    return s;
}, str);
```

Ne diyebilirim ki. **C# 3.0** sürpriz yeteneklerle dolu ve bazı temel esaslara bakış açımızı oldukça değiştiriyor. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

StrategyPattern.rar (22,61 kb)

Tasarım Prensipleri - Interface Segregation (2009-07-02T16:16:00)

design principles,

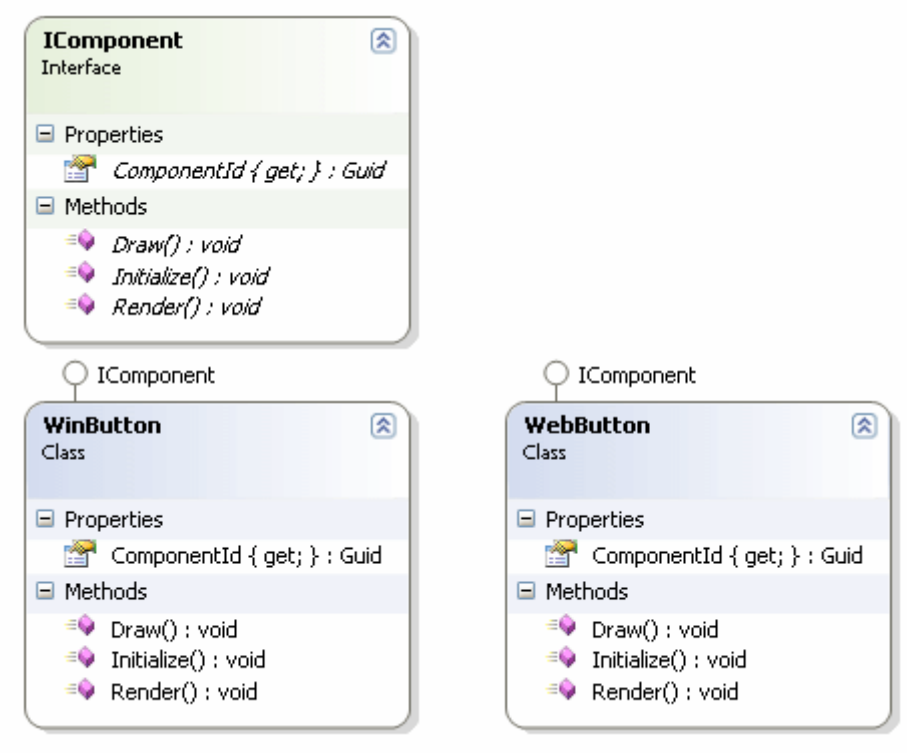
Merhaba Arkadaşlar,

Bir süredir pek çok nesne yönelimli yazılım disiplininde önem arz eden ve kullanılan **Tasarım Prensiplerini(Design Principles)** incelemeye ve öğrendiklerimi sizlere aktarmaya çalışıyorum. Şu ana kadar pek çok prensibi inceledik ve kısaltmalarına tanık olduk.

- LCP (Loose Coupling Principle)
- OCP (Open Closed Principle)
- SRP (Single Responsibility Principle)
- LSP (Liskov Substituation Principle)
- DIP (Dependency Inversion Principle)

Elbetteki önemli olan, kısaltmalarının karşılıklarını bilmek değil , söz konusu prensiplerin farkına vararak yazılım geliştirmek yada geliştirilen yazılım içerisinde bu prensiplerin uygulanabileceği, uygulanması gereken yerleri tespit edebilmektir. Bu hususları dikkate alarak ara sıra tasarım prensiplerini tekrar etmeye özen göstererekten, yeni tasarım prensibini incelemeye başlayabiliriz. **Interface Segregation Principle(ISP)**

çok hızlı bir giriş olacak ama konuya aşağıdaki sınıf diagramında görülen tipleri göz önüne alarak başlayalım.*(Her zamanki gibi ilkenin özlü sözünü kavrayabilmek için örnekle başlamakta yarar olduğu kanısındayım)*



Kod tarafından baktığımızda ise;

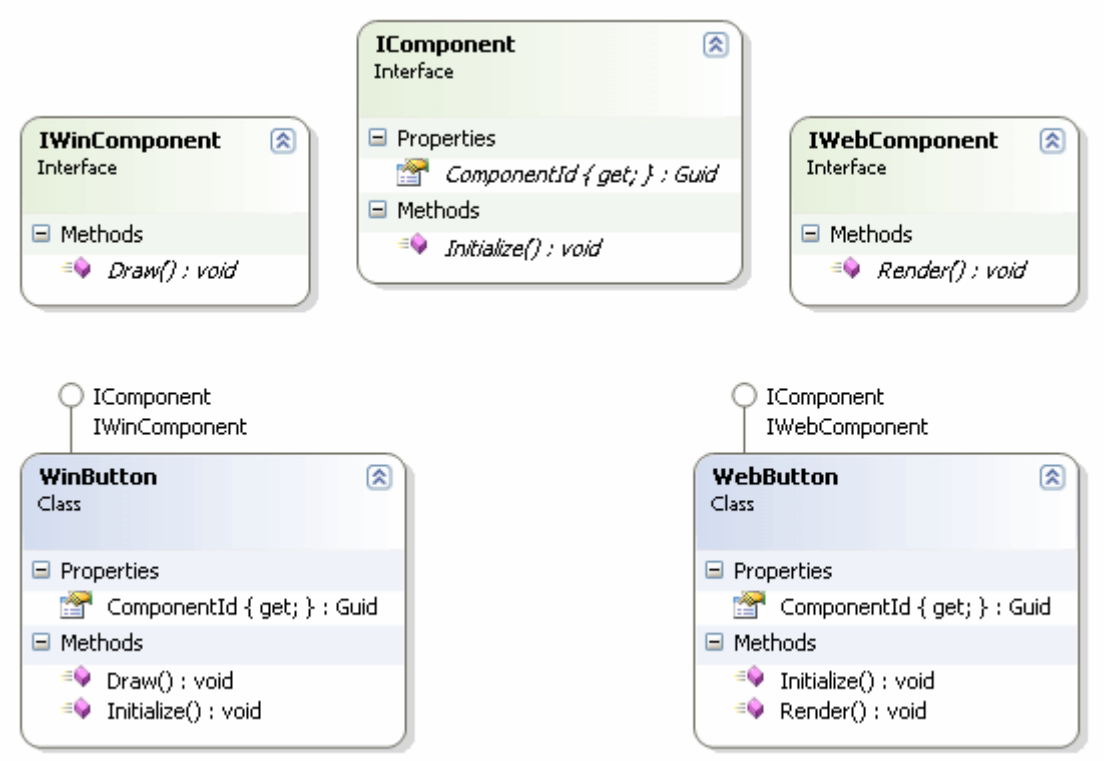
```
interface IComponent
{
    Guid ComponentId { get; }

    void Initialize();
    void Draw();
    void Render();
}

public class WinButton
: IComponent
{
    #region IComponent Members

    public Guid ComponentId
    {
        get { return Guid.NewGuid(); }
    }

    public void Initialize()
    {
        Console.WriteLine("Windows Button başlangıç işlemleri");
    }
```

```
    public void Draw()
    {
        Console.WriteLine("Ekrana çizdirme işlemleri");
    }

    public void Render()
    {
        throw new NotImplementedException();
    }

    #endregion
}

class WebButton
    :IComponent
{
    #region IComponent Members

    public Guid ComponentId
    {
        get { return Guid.NewGuid(); }
    }

    public void Initialize()
    {
        Console.WriteLine("Web Button başlangıç işlemleri");
    }

    public void Draw()
    {
        throw new NotImplementedException();
    }

    public void Render()
    {
        Console.WriteLine("HTML Render işlemleri");
    }

    #endregion
}
```

Aslında böylesine kötü bir tasarım ile konuya başlamak istemezdim ancak **ISP** ilkesinin neyi öğütlediğini bilmek adına bu yeterli bir yaklaşımdır. Hedefte bir sistemin parçası olan görsel bileşenlerin uygulaması gereken kuralları bildiren **IComponent** isimli arayüz tipi bulunmaktadır. Görsel bileşenler olarak örneğimizde, **Windows** ve **Web** tabanlı

uygulamalardaki Button kontrolleri göz önüne alınmaktadır. Ancak bir **windows** kontrolünün temel olarak ekrana çizdirilmesi ile, bir Web kontrolünün istemci tarafına **HTML** içeriği olarak **Render** edilmesi iki farklı ve ap ayrı fonksiyonelliktir. Dolayısıyla ISP ilkesine bu tespitten itibaren ters düşülmeye başlanmaktadır. Nitekim, **IComponent** arayüzünü uygulayan **WinButton** sınıfı içerisinde yer alan **Render** metodunun kesin olarak implemente edilmemesi gerekir. Hatta implemente edilmesi anlamsızdır. Bu nedenle çözüm olarak içerisinden **NotImplementedException** istisnasının fırlatıldığını görmekteyiz. Diğer taraftan benzer durum **WebButton** bileşeni içinde geçerlidir. öyleki **Web** arayüzü için **Render** işlemi önemli iken, **Draw** isimli fonksiyonelliğingerçekleştirilmemesi gerekir. Ayrıca **IComponent** arayüzüne yeni eklentiler yapılmak istendiğinde, **Liskov Substitution** ilkesine ters düşebilecek durumlarında oluşması söz konusudur. (Bknz : Tasarım Prensipleri : Liskov Substitution) Peki bu sorunlar bize neyi göstermektedir?

IComponent arayüzünü uygulayan tipler, aslında kendi içlerinde kullanmayacakları fonksiyonellikleri uyarlamak zorunda kalmışlardır. **NotImplementedException** ile istisna fırlatılmış olsa bile... İşte **Interface Segregation** ilkesi, **bir istemcinin kullanmayacağı arayüz fonksiyonelliklerini hiç bir şekilde uygulamaması gerektiğini** belirtmektedir. Buda tahmin edileceği üzere söz konusu fonksiyonellikleri farklı arayüzlere bölerek mümkün olabilir. Yani, yukarıda tasarlamış olduğumuz sistemi aşağıdaki hale getirerek ISP ilkesine sadık kalmayı başarabiliriz.



Kod içeriğimizi ise şu şekilde güncellemeliyiz;

```
interface IComponent
   {
      Guid ComponentId { get; }

      void Initialize();
   }
   interface IWebComponent
   {
      void Render();
   }
   interface IWinComponent
   {
      void Draw();
   }

   public class WinButton
   : IComponent,IWinComponent
   {
      #region IComponent Members

      public Guid ComponentId
      {
         get { return Guid.NewGuid(); }
      }

      public void Initialize()
      {
         Console.WriteLine("Windows Button başlangıç işlemleri");
      }

      #endregion

      #region IWinComponent Members

      public void Draw()
      {
         Console.WriteLine("Ekrana çizdirme işlemleri");
      }

      #endregion
   }

   class WebButton
      : IComponent,IWebComponent
```

```
{
    #region IComponent Members

    public Guid ComponentId
    {
        get { return Guid.NewGuid(); }
    }

    public void Initialize()
    {
        Console.WriteLine("Web Button başlangıç işlemleri");
    }

    #endregion

    #region IWebComponent Members

    public void Render()
    {
        Console.WriteLine("HTML Render işlemleri");
    }

    #endregion
}
```

Görüldüğü gibi **Web** tarafını ilgilendiren **Render** fonksiyonu **IWebComponent** isimli arayüzde, Windows tarafını ilgilendiren **Draw** metodu ise **IWinComponent** arayüzü içerisinde ele alınmıştır. Artık, ortak olan üyelerin IComponent, Web tarafını ilgilendirenlerin IWebComponent ve Windows tarafını ilgilendirenlerinde IWinComponent içerisinde toplanması sağlanarak ISP ilkesinin korunması sağlanabilir. Hatta söz konusu senaryoda **IWebComponent** ve **IWinComponent** arayüzlerinin **IComponent** arayüzünde n türemesi dahi düşünülebilir.

Bu ilke ile ilişkili olaraktan internet ve basılı kaynaklarda çok çok güzel örnekler yer al önüne alınmıştır. çalışanlar yemek yiyen insanlar ve yemek yemeyip sürekli çalışan Ro

Ancak ilk etapta tasarlanan herşeyi içinde barındıran **şişman arayüz(Fat Interface),** yemek yeme fonksiyonunu barındırdığı için, Robot' larında gerekmediği halde söz konusu işlevselliği uygulaması zorunlu olmuştur ki bu andan itibaren **ISP** ilkesine ters bir durum oluşmaktadır. Bu yazıyıda fikir vermesi açısından incelemenizi tavsiye ederim.

Böylece geldik bir ilkenin daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

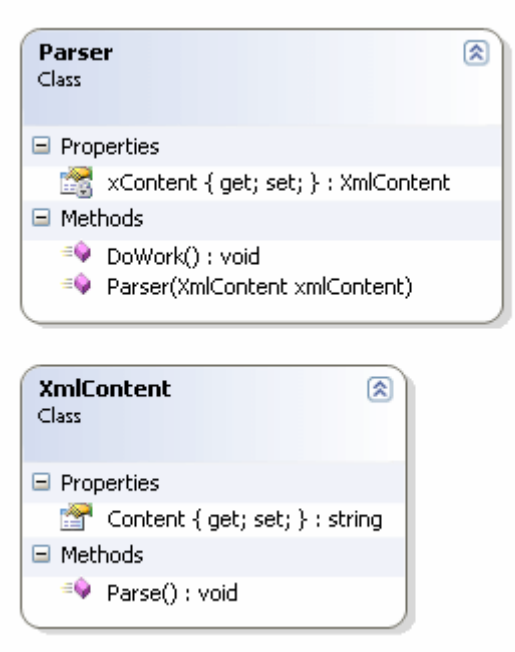
ISP.rar (31,41 kb)

Tasarım Prensipleri - Dependency Inversion (2009-06-30T22:45:00)

design principles,

Merhaba Arkadaşlar,

Bu yazımızda **Dependency Inversion** isimli tasarım prensibinden bahsediyor olacağız. Bu prensip kabaca, alt sınıflar ve üst sınıflar arasında kuvvetli bir bağ olmamasını önermektedir. Bunun en büyük gerekçesi, alt sınıflarda olabilecek sık değişiklerin, üst sınıfında değişmesine neden olabilecek olmasıdır ki bu hızla değişen yazılım ihtiyaçlarında sorunlara neden olmaktadır. Buna birde yeni alt tipler ile genişletilebilme olasılıklarınıda eklersek, üst ve alt sınıflar arasındaki bağımlılıkların ortadan kaldırılmasının *(bağımsızlık olarak düşünmek istesemde, bağımlılığın tersine çevrilmesi olarak bilmek zorundayız)* aslında ne kadar önemli olduğu anlaşılabilir. Durumu daha iyi kavrayabilmek adına basit bir örnek üzerinden ilerlemek çok daha doğrudur. öncelikli olarak aşağıdaki sınıf diagramı ve kod içeriğinde görülen örnek**Console** uygulamasını göz önüne alalım.




```
using System;

namespace Problem
{
```

```
   // Low Level Class
   class XmlContent
   {
      public string Content { get; set; }

      public void Parse()
      {
         Console.WriteLine("parsing işlemi");
      }
   }

   // High Level Class
   class Parser
   {
      XmlContent xContent { get; set; }

      public Parser(XmlContent xmlContent)
      {
         xContent = xmlContent;
      }

      public void DoWork()
      {
         // Kompleks işlemler yapıldığını varsayabiliriz.
         xContent.Parse();
      }
   }

   class Program
   {
      static void Main(string[] args)
      {
         Parser prsr = new Parser(new XmlContent { Content =
"<Kitaplar><Kitap><Ad>C#</Ad></Kitap><Kitap><Ad>VB.Net</Ad></Kitap></Kitapl
ar>" });
         prsr.DoWork();
      }
   }
}
```

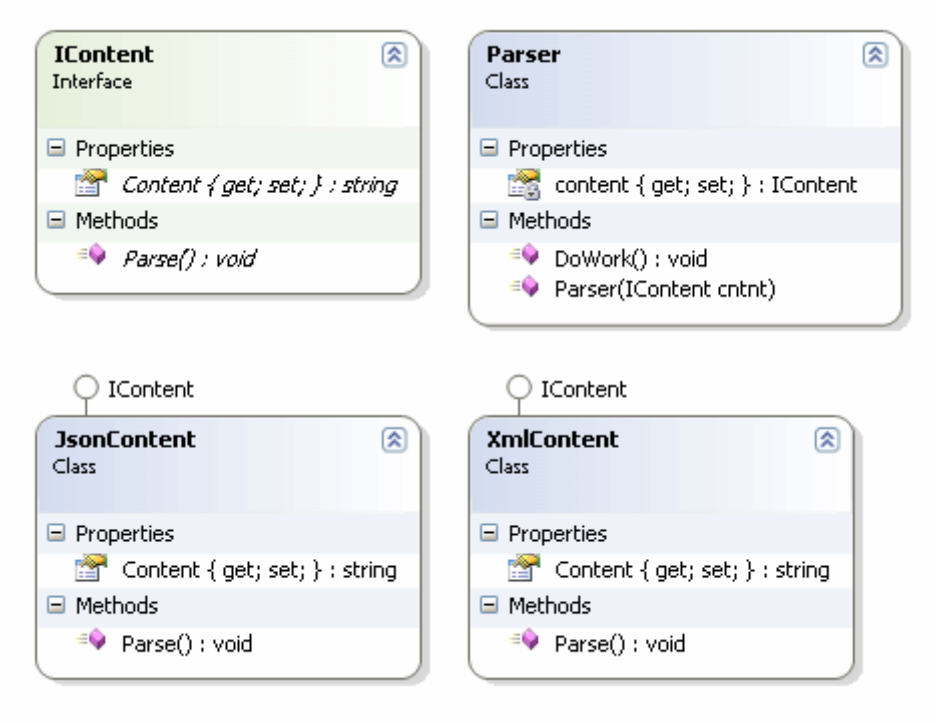
Bu örnekte yer alan **Parser** sınıfı, kendi içerisinde tanımlı olan **XmlContent** tipine bağımlıdır. XmlContent tipi temel olarak bir **Xml** içeriğini ifade etmek üzere tasarlanmış olup, ayrıştırma işlemi için özel bir metoda sahiptir. Diğer yandan **Parser** sınıfı içerisinde yer alan **DoWork** metodu, **XmlContent** tipinin **Parse** metodunu çağırmaktadır. **DoWork** metodu içerisinde aslında, **XmlContent** tipi ile ilişkili olaraktan

farklı ve hatta karmaşık bazı iş kurallarının uygulandığı farz edilebilir. Ancak **alt sınıf(Low Level Class)** olarak kabul edebileceğimiz **XmlContent** tipinin yapısında olabilecek değişikliker tahmin edileceği üzere **üst sınıfı(High Level Class)** doğrudan etkileyecektir. özellike üst sınıfta kod değişikliğine gidilmesi gerekecektir. Diğer yandan senaryoya yeni bir tip eklenmek istendiğinde de **Parser** tipinin bozulması ve içeriğinin değiştirilmesi gerekecektir. Yani genişletilebilirlik söz konusu olduğunda, üst sınıfta kod meydana gelecek kod değişimi kaçınılmaz olacaktır. Şimdi senaryomuzu genişletip sorunu ortaya koymaya çalışalım. Bu amaçla var olan sisteme, yeni bir alt sınıfın eklendiğini farzedelim. örneğin **Parser** sınıfının, **Json** formatındaki dökümanları ifade eden bir sınıf ilede çalışması istenebilir. Bu durumda 3 temel sorundan bahsedebiliriz;

- Yeni eklenen tip nedeniyle Parser sınıfının kendisine müdahele edilmesi ve DoWork metodunun kod içeriğinin değiştirilmesi gerekecektir.
- DoWork metodu veya Parser sınıfı içerisindeki bazı fonksiyonelliklerin işleyişi olumsuz yönde etkilenebilir.
- Her ne olursa olsun, Unit Test işlemlerinin tekrardan oluşturulması ve yapılması gerekecektir.

Şimdi biraz durup bu sorunların üstesinden nasıl gelebileceğinizi düşünmenizini öneririm. Hatta düşünürken bir kahve arası verebilir ve çevrenizde bu konu ile ilgili kişiler varsa onlarla durumu tarışabilirsiniz.

Dependency Inversion prensibi, yukarıda bahsediğimiz tipte bir senaryonun oluşmasını engellemek için, üst sınıf ile alt sınıf arasına bir **soyutlayıcının(abstract class veya interface)** konulmasını belirtir. Yani üst sınıf ve alt sınıf arasında bir **interface** veya **abstract** tipin kullanılması önerilmektedir. İşte örnek senaryomuzun **Dependency Inversion**prensibine uygun olarak geliştirilen son hali.




```
using System;

namespace Solution
{
   // Abstraction Layer
   interface IContent
   {
      string Content { get; set; }

      void Parse();
   }

   // Low Level Class
   class XmlContent
      :IContent
   {
      #region IContent Members

      public string Content { get; set; }

      public void Parse()
      {
         Console.WriteLine("Xml parsing işlemi");
      }

      #endregion
   }
```

```
    // Low Level Class
    class JsonContent
        : IContent
    {
        #region IContent Members

        public string Content { get; set; }

        public void Parse()
        {
            Console.WriteLine("Json parsing işlemi");
        }

        #endregion
    }

    // High Level Class
    class Parser
    {
        IContent content { get; set; }

        public Parser(IContent cntnt)
        {
            content = cntnt;
        }

        public void DoWork()
        {
            // Kompleks işlemler yapıldığını varsayabiliriz.
            content.Parse();
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            Parser prsr = new Parser(new XmlContent { Content = "" });
            prsr.DoWork();

            prsr = new Parser(new JsonContent { Content = "" });
            prsr.DoWork();
        }
    }
}
```

Görüldüğü gibi **XmlContent** ve **JsonContent** isimli tiplerimiz **IContent** arayüzünden türetilmiştir. Buna bağlı olarakta **Parser** tipi ile IContent arayüzü arasında bir bağlantı oluşturulmuştur. Bir başka deyişle, Parser tipinden XmlContent veya JsonContent tiplerine **doğrudan bir bağ mevcut değildir**. Artık alt sınıflar olan XmlContent ve JsonContent içerisinde istenilen değişiklikler yapılabilir. Söz gelimi Parse metodlarının iş mantığında değişimler olabilir yada tipler içerisine yeni üyeler eklenebilir. Sonuç olarak; **arayüzün(veya soyut tipin) yapısının değiştirilmediği düşünüldüğünde, üst sınıfın, alt sınıflardaki olası değişiklier ve sistemdeki genişletmelere karşı bağımsız olması** garanti altına alınmış olur.

Bir sonraki yazımızda görüşünceye dek hepinize mutlu günler dilerim.

DIP.rar (40,27 kb)

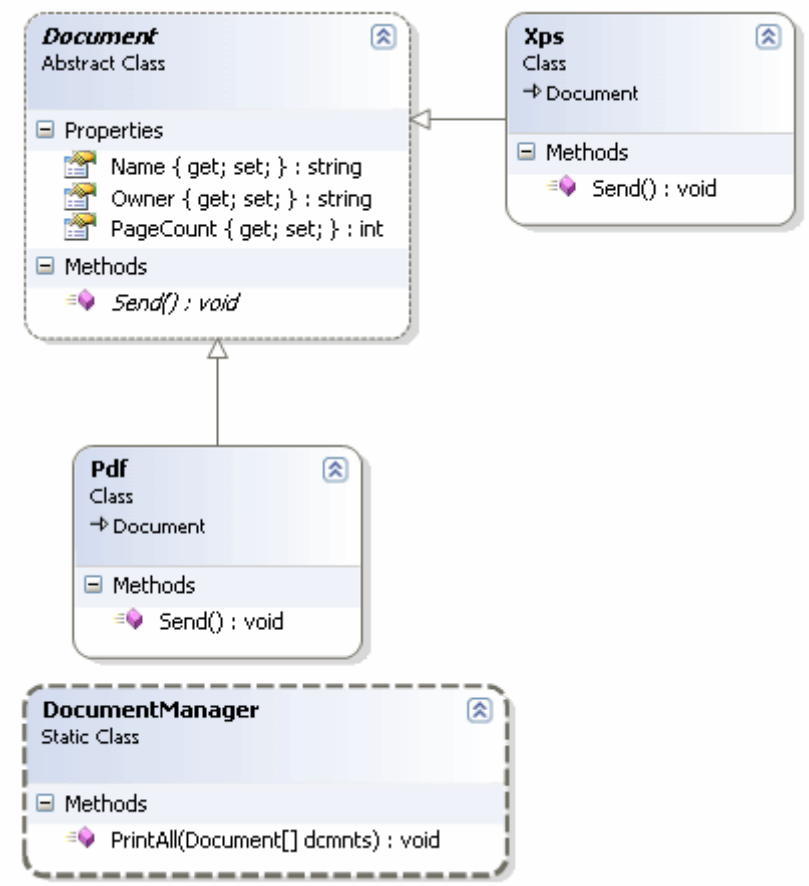
Tasarım Prensipleri - Liskov Substitution (2009-06-30T00:57:00)

design principles,

Merhaba Arkadaşlar,

Bu günkü blog yazımızın kahramanı **Barbara Liskov**(http://en.wikipedia.org/wiki/Barbara_Lisk içerisinde uygulanan disiplinlerden birisi olan **Liskov Substitution(LSP)** ilkesi. Bu ilke **üst sınıf** aldığı bir prensip olarak göz önüne alınabilir aslında. İlkenin özet cümlesini söylemeden önce bas kanısındayım. Nitekim özet cümleyi okuduğunuzda kafanızın karışmamasını garanti edemeyeceğ

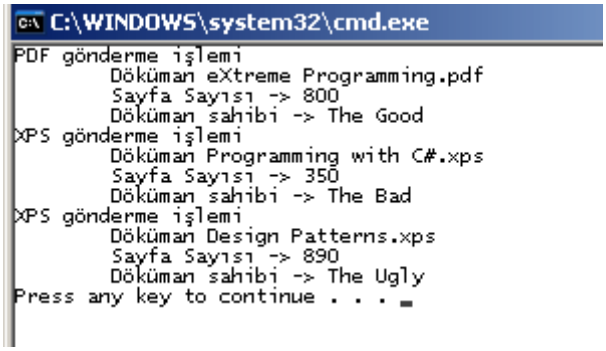
örnek **Console** uygulamamızda **Document** isimli bir **abstract** sınıfımız mevcuttur. Bu sınıf **Pdf, Xps** gibi dökümanların ortak özellikleri ile uygulaması gereken kuralları tanımlamaktadır. örneğin dökümanın network üzerinde bir yere gönderilmesi veya printer üzerinden yazdırılması bu anlamda zorunlu fonksiyonellikler olarak düşünülebilir. Diğer yandan,**DocumentManager** isimli bir başka sınıfta, **Document** sınıfına bağımlı olan bir tip olarak karşımıza çıkmaktadır. Bu tip kendi içerisinde, Document tipinden türeyen nesne örnekleri üzerinde ortak işlemlerin yapılmasını sağlamaktadır. Söz gelimi birden fazla **Document** tipinin yazdırılması veya gönderilmesi bu ortak operasyonlar olarak düşünülebilir. Burada işin önemli kısımlarından birisi, **Xps, Pdf** gibi sınıfların ve sonradan sisteme eklenecek Document türevli tiplerin, DocumentManager tarafında ortaklaşa kullanılabilecek olmasıdır. Dilerseniz örneğimize bakalım.



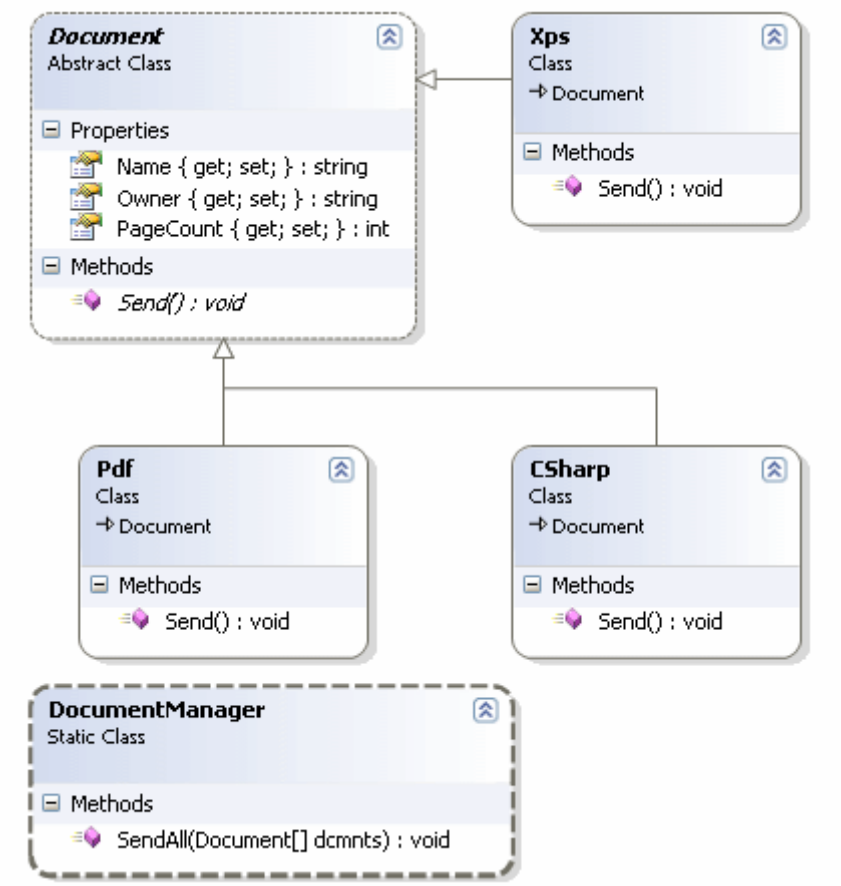

```
using System;

namespace Problem
{
   // Base Class
   abstract class Document
   {
      public int PageCount { get; set; }
      public string Name { get; set; }
      public string Owner { get; set; }

      public abstract void Send();
   }

   // Sub Class
   class Pdf
      : Document
   {
      public override void Send()
      {
         Console.WriteLine("PDF gönderme işlemi\n\tDöküman {0}\n\tSayfa Sayısı ->
{1}\n\tDöküman sahibi -> {2}", Name, PageCount.ToString(), Owner);
```

```
    }
  }

  // Sub Class
  class Xps
    : Document
  {
    public override void Send()
    {
      Console.WriteLine("XPS gönderme işlemi\n\tDöküman {0}\n\tSayfa Sayısı ->
{1}\n\tDöküman sahibi -> {2}", Name, PageCount.ToString(), Owner);
    }
  }

  // Client
  static class DocumentManager
  {
    public static void SendAll(Document[] dcmnts)
    {
      foreach (Document dcmnt in dcmnts)
      {
        dcmnt.Send();
      }
    }
  }

  class Program
  {
    static void Main(string[] args)
    {
      Document[] dcmnts ={
                          new Pdf{ Name="eXtreme Programming.pdf", PageCount=800,
Owner="The Good"},
                          new Xps{Name="Programming with C#.xps", PageCount=350,
Owner="The Bad"},
                          new Xps{Name="Design
Patterns.xps",PageCount=890,Owner="The Ugly"}
                        };

      DocumentManager.SendAll(dcmnts);
    }
  }
}
```

Console uygulaması **Pdf** ve **Xps** sınıflarına ait nesne örneklerinin toplandığı **Document** tipinden bir diziyi oluşturmakta ve bunu **DocumentManager** sınıfındaki **SendAll** metoduna göndermektedir. Xps ve Pdf tipleri, **Document** sınıfından türemiştir ve **Send** metodunun **abstract** olarak tanımlanması nedeniylede, söz konusu fonksiyonu kendi içlerinde ezmişlerdir**(Override)**. Bu nedenle Send metodu içerisindeki döngüde, Document tipinden nesne örnekleri ele alınmasına rağmen, **Send** metodu çalışma zamanında yeri geldiğinde Xps için, yeri geldiğinde de Pdf için çalıştırılacaktır. İşte nesne yönelimli olmanın güzel noktalarından birisi. Uygulamayı çalıştırdığımızda aşağıdaki sonuçlar alırız.



Ancak **Liskov Substitution** ilkesinin savunduğu da bir kural vardır. Bu kuralı göstermek için senaryomuzu şu andan itibaren biraz değiştiriyor olacağız. İlk olarak uygulamaya **Document** sınıfından türeyen **CSharp** isimli yeni bir tip eklediğimizi düşünelim. Bu tip cs uzantılı kod dosyalarını ifade etmektedir.



Yeni senaryomuza göre **CSharp** isimli tiplerin temsil ettiği dökümanların(cs uzantılı kod dosyaları olarak düşünebiliriz) herhangibir sebeple **Send** işlemine tabi tutulmaması gerekmektedir. Oysaki **Send** metodu **Document** tipi içerisinde **abstract** olarak tanımlandığından, CSharp tipi içerisinde de ezilmesi gerekmektedir. Peki ya sistemde **Send** işleminin yaptırtılması istenmiyorsa...

İki seçeneğimiz olabilir. Bunlardan birincisi **CSharp** tipi içerisindeki **Send** metodundan üst katmana doğru bir **istisna(Exception)** fırlatmaktadır. Belkide geliştirici tarafından yazılmış bir istisna tipide söz konusu olabilir. Diğer bir alternatif ise **Client** tipi içerisindeki(DocumentManager sınıfı), **SendAll** metodunda tip kontrolü yapmaktır. Buna göre gelen tip **CSharp** ise **Send** işleminin icra edilmemesi sağlanabilir. Bir başka deyişle kodlarımızı aşağıdak gibi güncelleştirebiliriz.

```
using System;

namespace Problem
{
   // Base Class
   abstract class Document
   {
      public int PageCount { get; set; }
```

```
        public string Name { get; set; }
        public string Owner { get; set; }

        public abstract void Send();
    }

    // Sub Class
    class Pdf
        : Document
    {
        public override void Send()
        {
            Console.WriteLine("PDF gönderme işlemi\n\tDöküman {0}\n\tSayfa Sayısı ->
{1}\n\tDöküman sahibi -> {2}", Name, PageCount.ToString(), Owner);
        }
    }

    // Sub Class
    class Xps
        : Document
    {
        public override void Send()
        {
            Console.WriteLine("XPS gönderme işlemi\n\tDöküman {0}\n\tSayfa Sayısı ->
{1}\n\tDöküman sahibi -> {2}", Name, PageCount.ToString(), Owner);
        }
    }

     // Sub Class
    class CSharp
        : Document
    {
        public override void Send()
        {
            // Seçeneklerden birisi exception fırlatılması olabilir. Bu durumda exception' ın üst
katmanlarda ele alınması(catch) gerekir.
            throw new Exception("Bu döküman için gönderme işlemi yapılamaz");
        }
    }

    // Client
    static class DocumentManager
    {
        public static void SendAll(Document[] dcmnts)
        {
```

```
        foreach (Document dcmnt in dcmnts)
        {
           // Bir diğer seçenek tip kontrolü olabilir. Tipe göre söz konusu alt sınıf
operasyonunun gerçekleştirilmemesi sağlanabilir.
           if(dcmnt is CSharp)
              continue;
           else
              dcmnt.Send();
        }
     }
  }

  class Program
  {
     static void Main(string[] args)
     {
        Document[] dcmnts ={
                           new Pdf{ Name="eXtreme Programming.pdf", PageCount=800,
Owner="The Good"},
                           new Xps{Name="Programming with C#.xps", PageCount=350,
Owner="The Bad"},
                           new CSharp{Name="ProductEntity.cs", PageCount=15,
Owner="SourceSafe"},
                           new Xps{Name="Design
Patterns.xps",PageCount=890,Owner="The Ugly"}
                        };

        DocumentManager.SendAll(dcmnts);
     }
  }
}
```

örnekte **SendAll** metodu içerisinde tip kontrolü yapılarak söz konusu senaryonun gerçeklenmemesi sağlanmıştır. Hatta uygulama çalıştırıldığında bir önceki ile aynı sonuçlar alınacaktır. Ancak burada ilk başka **Open Closed** ilkesine ters düşen bir durum yaşanmaktadır. öyleki, sisteme yeni bir tip eklendiğinde, **DocumentManager** içerisindede değişiklik yapılması zorunludur. Zaten, **Open Closed** prensibine uymayan durumlar **Liskov Substitution** ilkesininde bozulduğu anlamına gelmektedir. Liskov Substitution ilkesi bize şunu söylemektedir; **üst sınıf ile alt sınıf nesne örnekleri yer değiştirdiklerinde, üst sınıf kullanıcısı alt sınıfın operasyonlarına erişmeye devam edebilmelidir.** Oysaki son senaryoda, **CSharp** alt sınıfındaki operasyona erişilirken özel bir işlem yapılması gerekmiştir(tip kontrolü veya istisna yakalama işlemi). Bu örnekte **LSP** ilkesine aykırı olan durumu ortadan kaldırmak için yapılması gereken tek şey vardır oda **CSharp** sınıfını **Document** sınıfına ait **Domain** yapısından çıkartmak.

Internet ve basılı kaynaklarda **LSP** ilkesini araştırdığınızda **Rectangle** ve **Square** örneği ilede karşılaşabilirsiniz. Bu senaryoda, **Liskov Substitution** ilkesini son derece basit ve yalın bir dille anlatmaktadır. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

LSP.rar (23,83 kb)

Tasarım Prensipleri - Single Responsibility (2009-06-27T04:04:00)

design principles,

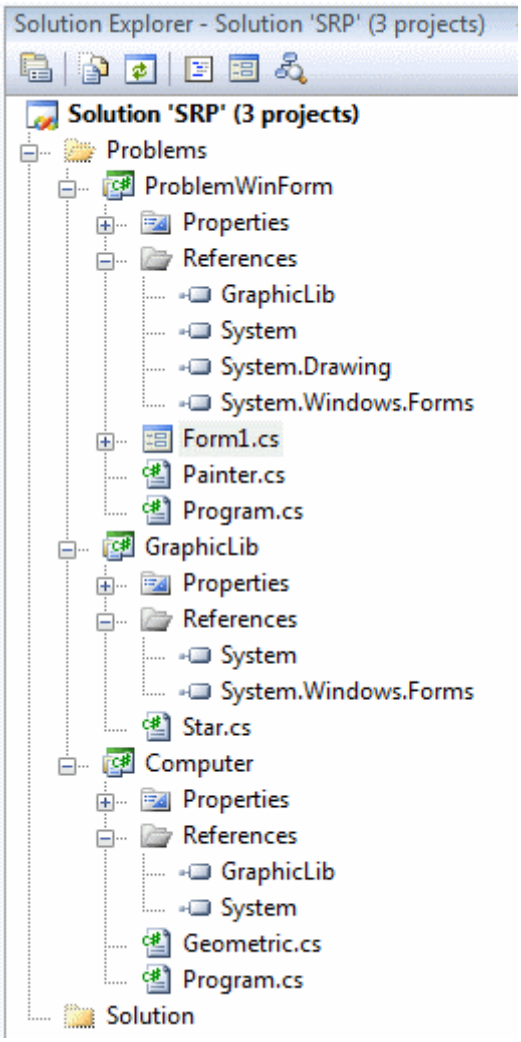

Merhaba Arkadaşlar,
Sanıyorum benim gibi eskiler, yandaki resimde yer alan değerli ressamı hatırlayacaklardır. **Bob Ross**. Küçüklüğümde(ve halen) Bob Ross'un **TRT** televizyonunda yayınlanan **Resim Sevinci** programlarını zaman zaman izler ve yarım saatlik sürede çizdiği doğa manzaralarına bakakalırdım. Rahmetli Bob bu günkü **Tasarım Prensipleri** uyarlanması sırasında **Einstein** ile birlikte küçük bir rol üstleniyor olacak. öyleyse sözü fazla uzatmadan konumuza geçelim.

Hatırlayacağınız gibi son iki blog yazımda, **nesneye dayalı tasarım prensipleri** içerisinde uyarlanan ilkelere değinmeye çalışmıştım. Bu günkü konumuz ise **Single Responsibility** prensibi. Bu prensip anlaşılması kolay ancak çoğu zaman tespit edilmesi veya gerekliliğinin ortaya çıkartılması zor bir ilke olarak karşımıza çıkmaktadır. İlkenin savunduğu tez şudur; **Bir sınıf sadece tek bir sorumluluk içermelidir.** Bir başka deyişle bir sınıfın birden fazla sorumluluğa sahip olmasına karşı bir ilkedir. Bunun en büyük nedeni olarak, sıklıkla yapılan ya da beklenen değişikliklerin, sorumluluk sayısı fazla olan sınıflar için yeniden kullanılabilirliği(Reusable), test edilebilirliği, genişletilebilirliği vb... zorlaştırıyor olmasıdır. Bu ilke aslında şu şekildede açıklanabilir; **bir sınıfın değişikliğe uğraması için birden fazla neden olmamalıdır.**

Her zamanki gibi konuyu daha kolay kavrayabilmek adına basit bir örnek üzerinden ilerlemekte yarar olduğu kanısındayım.

Bu konu ile ilişkili olaraktan çeşitli kaynaklarda oldukça farklı ve güzel örnekler bulunmakta. özellikle şu sıralar takip ettiğim, **Robert C. Martin'** in **Agile Principles, Patterns, and Practices in C#** kitabındaki örneklerden esinlendiğimi baştan belirtmek isterim.

öncelikli olarak problemi içeren Solution içeriğimizi ele alalım. örneğimizde **Star** isimli bir sınıf bulunmaktadır. Bu sınıf basit anlamda 3 boyutlu bir yıldız şeklinin bazı değerlerini tutmaktadır. Ancak dahada önemlisi içerisinde yıldızın hacmini hesaplayan ve yıldız şeklinde bir **Windows Formu** çizen fonksiyonlar yer almaktadır.*(Bu son görevleri kafamızın bir köşesinde şimdiden tutalım)* Sınıfımız bir **Class Library** içerisindedir. Bu sınıf içerisinde bir Windows Form' unun çizilmesi sağlandığından **System.Windows.Forms assembly'** ını referans etmektedir. Diğer yandan Star isimli sınıfı kullanan iki farklı uygulama söz konusudur. Bunlardan birisi bir **Windows** uygulaması olup **Star** sınıfı içerisinde **Form**çizen operasyonu ele almaktadır. Diğer uygulama ise basit bir **Console** projesidir ve sadece generic List<T> tabanlı bir **Star** nesne koleksiyonundaki hacim değerlerini kullanarak bilimsel bir hesaplama gerçekleştirmektedir. Solution içeriği aşağıda görüldüğü gibidir.



Gelelim kod içeriğimize. GraphicLib isimli sınıf kütüphanemizde yer alan Star sınıfına ait kodlar aşağıdaki gibidir.

```
using System;
using System.Windows.Forms;
```

```
namespace GraphicLib
{
   public class Star
   {
      public int CornerCount { get; set; }
      public int LineWidth { get; set; }
      public int zValue { get; set; }

      public Form Paint(string text)
      {
         Form frm = new Form();

         // Aslında yıldız şeklinde bir form çizdirildiği varsayılabilir
         frm.Text = text;
         frm.Width = LineWidth*2;
         frm.Height = Convert.ToInt32((CornerCount * LineWidth) / 3.14);

         return frm;
      }

      public double Volume()
      {
         // Tamamen hayali bir hacim hesaplaması. Normalde oluşturulan yıldızın hacminin
hesaplandığı düşünülmektedir.
         return CornerCount * LineWidth;
      }
   }
}
```

Star sınıfı içerisinde yapılan anlamsız işlemlere takılmayın. Amacımız tamamen işe yarar bir sınıfı kullanmak değil şu aşamada Ancak Paint ve Volume metodları bizim için oldukça önemlidir. Computer isimli Console uygulamamız içerisinde Geometric isimli yardımcı bir sınıf bulunmaktadır.

```
using System.Collections.Generic;
using GraphicLib;

namespace Computer
{
   class Geometric
   {
      // Sembolik bir matematiksel hesaplama yaptığı varsayılır
      public double Compute(List<Star> stars)
      {
         double total = 0;
```

```
            for (int i = 0; i < stars.Count; i++)
            {
               total += stars[i].Volume();
            }

            return total;
         }
      }
   }
```

Bu sınıf içerisinde yer alan **Compute** metodu, parametre olarak gelen **List<Star>** koleksiyonunundaki her bir eleman için **Volume** metodunu ele almaktadır. Computer isimli programa ait test kodları ise aşağıdaki gibidir.

```
using GraphicLib;
using System;

namespace Computer
{
   class Program
   {
      static void Main(string[] args)
      {
         Geometric einstein = new Geometric();

         double result=einstein.Compute(
            new System.Collections.Generic.List<GraphicLib.Star>{
               new Star{CornerCount=10,LineWidth=12},
               new Star{CornerCount=8,LineWidth=24},
               new Star{CornerCount=5,LineWidth=35}
            }
            );
         System.Console.WriteLine("Bilimsel hesaplama yapılmıştır");

         System.Console.WriteLine(result.ToString());

         Console.ReadLine();
      }
   }
}
```

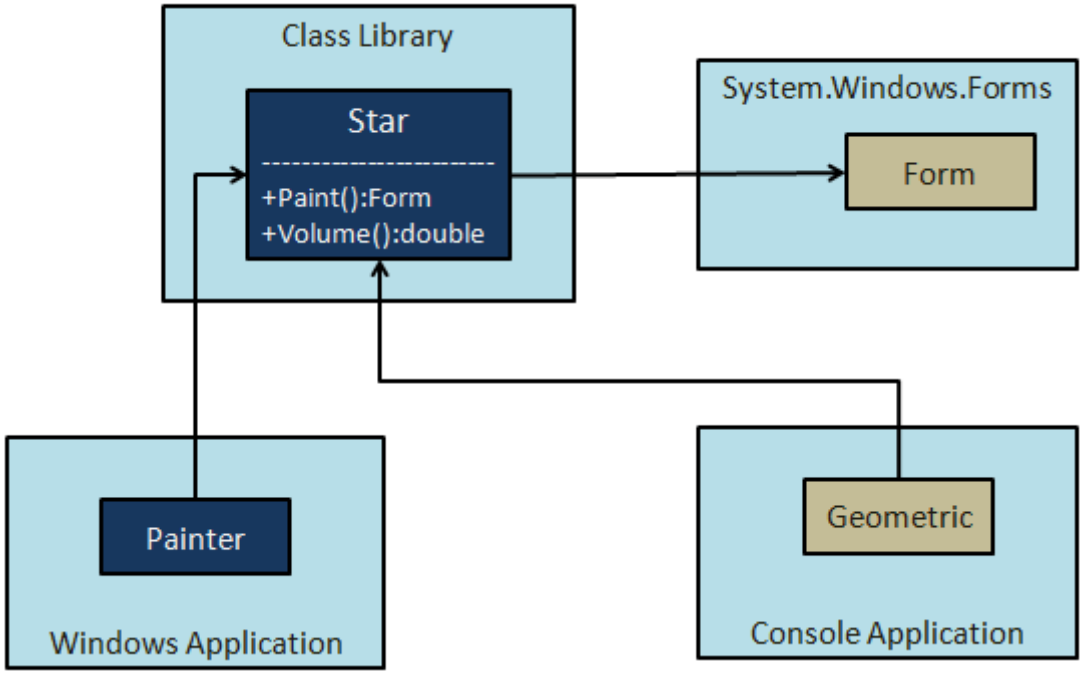
Peki ya **WinForms** uygulamamız. Bu uygulama içerisinde de **Painter** isimli bir sınıf bulunmaktadır.

```
using System.Windows.Forms;
using GraphicLib;

namespace ProblemWinForm
{
   class Painter
   {
      public void DrawScreen(Star aStar)
      {
         Form form = aStar.Paint("Yeni Form");

         form.Show();
      }
   }
}
```

Bu sınıf içerisinde yer alan **DrawScreen** metodu, parametre olarak **Star** tipinden bir referans almakta ve **Paint** metodunu çağırmaktadır. Ve sıra **Bob'** tadır. İşte Windows formumuzdaki Button kontrolüne basılınca olmasını istediklerimiz.

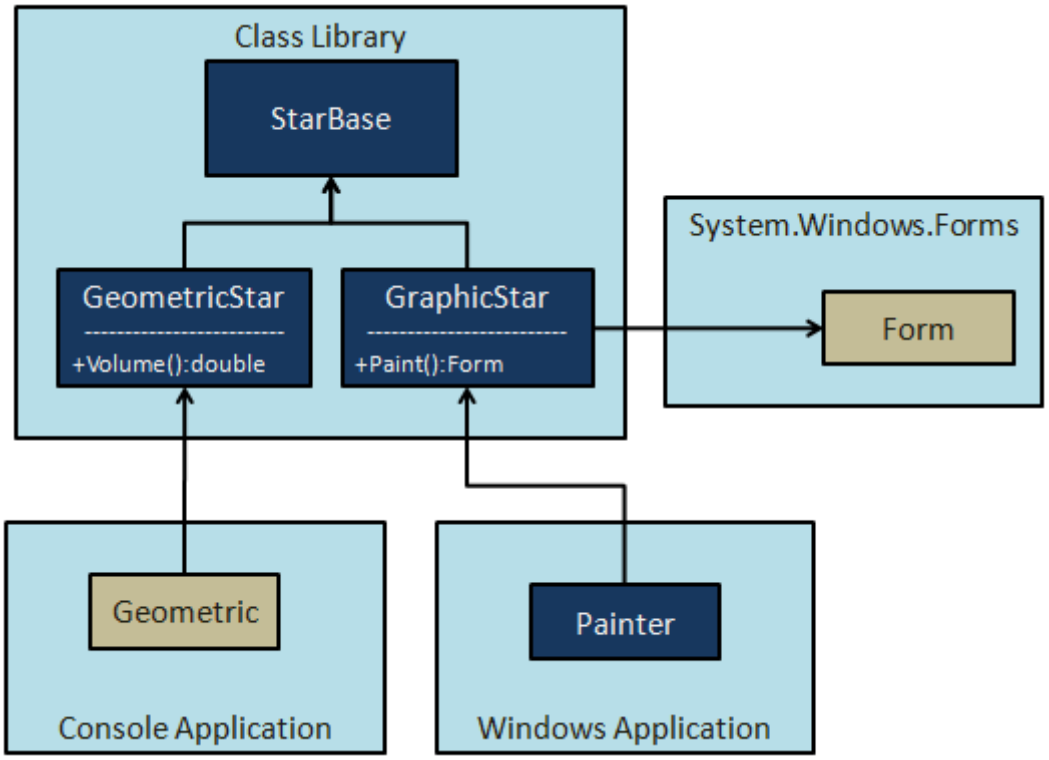
```
using System;
using System.Windows.Forms;
using GraphicLib;

namespace ProblemWinForm
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void btnDraw_Click(object sender, EventArgs e)
      {
         Painter bobRoss = new Painter();
         bobRoss.DrawScreen(new Star() { LineWidth = 100, CornerCount = 12 });
      }
   }
}
```

Peki şimdi ne oldu? Görüldüğü gibi tozu dumana kattık ve ortalığı iyice karştırdık. Karıştırdım Aslında aşağıdaki şekil zihnimizi biraz daha kolay aydınlatabilir.



Sanıyorumki durum şimdi biraz daha netleşmektedir. Yinede açıklamaya çalışayım. **Windows** uygulamasında yer alan **Painter** sınıfı için, **Star** tipi içerisinde kullanılan tek bir fonksiyonellik vardır. **Paint** isimli metod. **Painter** kesinlikle **Volume** isimli fonskiyonla ilgilenmemektedir ki işinede yaramamaktadır zaten. Aksine **Volume** metodu, **Geometric** tipinin yer aldığı **Console**uygulaması için önemlidir. Diğer yandan **Geometric** sınıfı içinde, **Star** tipindeki **Paint** metodunun bir önemi yoktur. Buda uygulamaların taşınmaları veya dağıtılmaları esnasında gereksiz olan assembly' larında taşınması anlamına gelmektedir. Bahsettiklerimizden yola çıkarak şu sonuca varabiliriz; Star sınıfı iki farklı sorumluluğu üstlenmektedir ve bu, **Single Responsibility** ilkesine ters düşmektedir.

öyleyse Single Responsibility prensibine uygun olacak şekilde nasıl bir çözüm üretebiliriz. Tek yapılması gereken sorumlulukları farklı sınıflara dağıtmak olacaktır. Yani iki farklı Star tipi tasarlanacak, bunlardan birisi Form çizme işlemini üstlenirken diğeri ise sadece hesaplama işlemlerini üzerine alacaktır. Söz gelimi **GeometricStar** ve **GraphicStar** isimli iki farklı sınıf bu amaçla tasarlanabilir.



Grafiksel işlemlere ait sorumluluklar farklı bir sınıfa, bilimsel hesaplamalar ile ilişkili sorumluluklarda diğer bir sınıfa verilmiştir. Her iki sınıf için değişmez olan ortak özellikler ise bir üst sınıfta toplanmıştır.

Görüldüğü gibi ilke son derece basit ama bazen tespit edilmesi, görülmesi ve hatta uygulanması kolay olmayabilir. Söz gelimi bu yazıdaki örneklerde farklı **Assembly'** lar yer almaktadır. Star sınıfı üzerindeki sorumlulukları farklı sınıflara dağıtmış olsak bile, **Console** ve **Windows** uygulamalarının her ikiside aynı kütüphaneyi referans etmekte ve her iki Star tipinede(dolayısıyla her iki sorumluluğada) erişebilmektedir. Belkide sadece sorumlulukları dahilindeki tiplere erişmeleri için bir takım önlemler alınması gerekebilir. Nitekim bu durumda, gereksiz tiplerinde taşınması söz konusudur ki buda ürünün çevikliğini negatif etkileyebilir. İşte hepimize kafa karıştırıcı olduğu kadar gerekli olan bir tartışma konusu. Yorumlarınız, tüm okurlarımız için değerli olacaktır.

Böylece geldik bir yazımızın daha sonuna. Bu yazımızda basit bir şekilde **Single Responsibility** prensibini incelemeye çalıştık. örnek tam olarak faydalı olmasada, **bir sınıfın tek bir sorumluluğa sahip olması** gerekliliğinin Single Responsibility prensibinin kendisi olduğunu anlamış bulunuyoruz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

SRP.rar (109,33 kb)

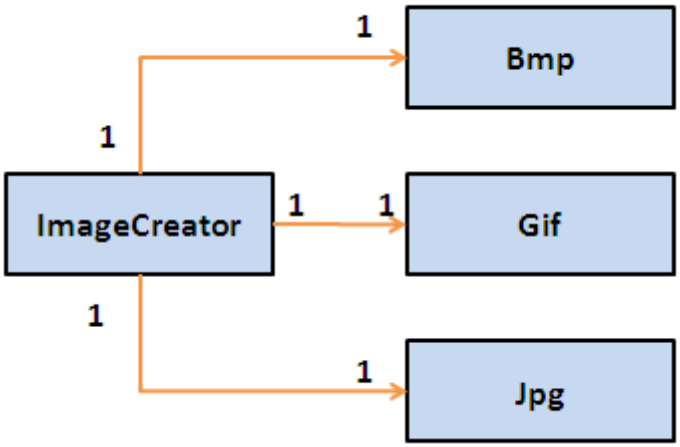
Tasarım Prensipleri - Open Closed (2009-06-25T16:22:00)

design principles,

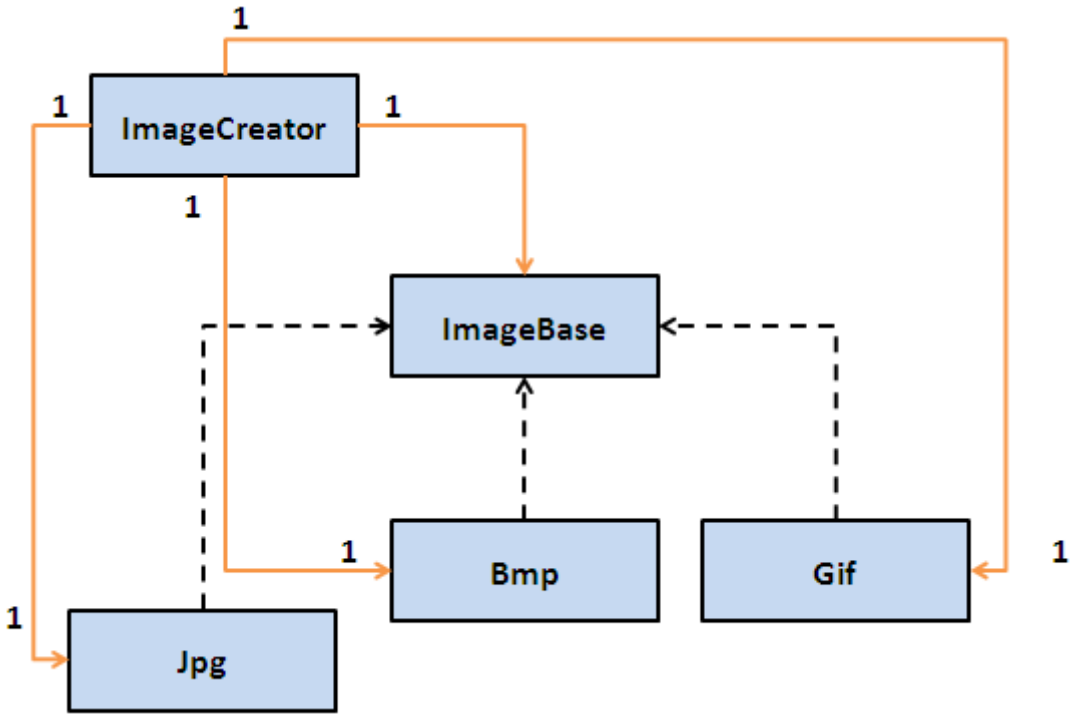
Merhaba Arkadaşlar,

Bir önceki yazımda, yazılm tasrımında benimsenen ilkelerinden birisi olan **Loose Coupling** prensibine değinmiştik. Bu yazımızda ise, **Open Closed(Açık Kapalı)** prensibine değiniyor olacağız.*(Bu prensibini pek çok yazılım disiplininde görebilirsiniz. örneğin **eXtreme Programming** veya **Aspect Oriented Programming-AOP** içerisinde.)*

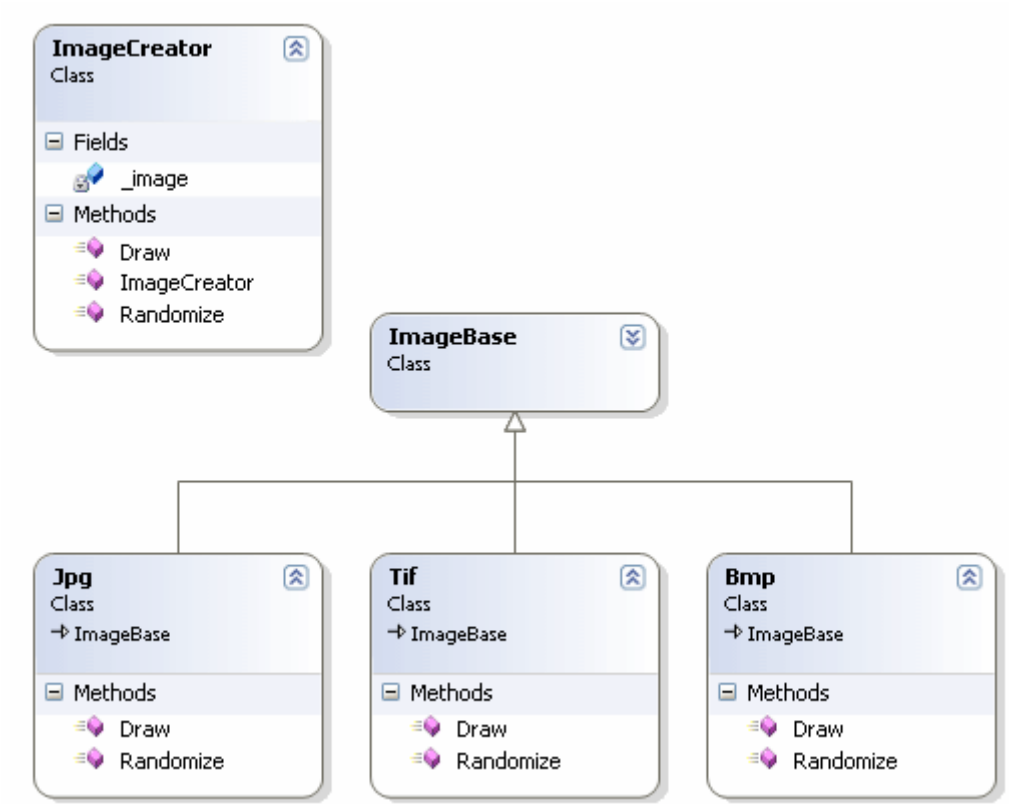
Açık kapalı prensibi aslında son derece basit bir ilkedir. Bu ilke bir sistemin sürekli olarak değişimlere maruz kalabileceğini göz önüne alaraktan*(ki örneğin çevik süreçlerde hızlı değişimler asıl odaklanılan noktadır)*, genişletilmeye açık ama modifiye edilmeye kapalı varlıkların*(Sınıf, Method vb...)* kullanılmasını önerir. Gerçekten de günümüz**Enterprise** çözümlerin çoğunda,müşteri ihtiyaçlarına göre yazılımın sürekli güncelleniyor olması gerekmektedir. Bu noktada güncelleştirme işlemleri sırasında koda dokunmadan ilerlemeye çalışmak neredeyse imkansızdır. Ancak bu risk en aza indirgenmeye çalışılabilir. **OCP(Open Closed Principle)** bu noktada devreye giren prensiplerden sadece birisidir. Tabikide bu teorik anlatım bir örnekle süslenmediği takdirde çok anlaşılır değildir Gelin önce problemli bir tasarım ile yola çıkalım ve sonrasında ise **OCP'** i nasıl uygulayabileceğimize bakalım*(ki bu noktada bir önceki blog yazısına göre bir dejavu yaşayabilirsiniz benden söylemesi)*



Yukarıdaki basit **UML** şemasında farklı formatlarda resimler üretmek için kullanılan bir yaratıcı sınıf**(ImageCreator)** görülmektedir. İlişkidende anlaşılacağı üzere**ImageCreator** sınıfı ile diğer resim sınıfları arasında kuvvetli bir bağ vardır. Bu acemi tasarımı biraz toparlamaya çalışmak istediğimizi düşünelim. Belkide aşağıdaki**UML** şemasında görülen kurguyu tasarlamış olabiliriz.



Bu kez ImageBase isimli bir ata sınıfı işin içerisine katmışız gibi görünüyor. Hatta koduda aşağıda şekilde tasarladığımızı düşünelim*(Tabi bu şekilde kod yazmamızın amacı tamam şakacıktan. Amaç bizi Open Close prensibine götüren sebepleri ortaya koyabilmek.* 🙂 *)*



using System;

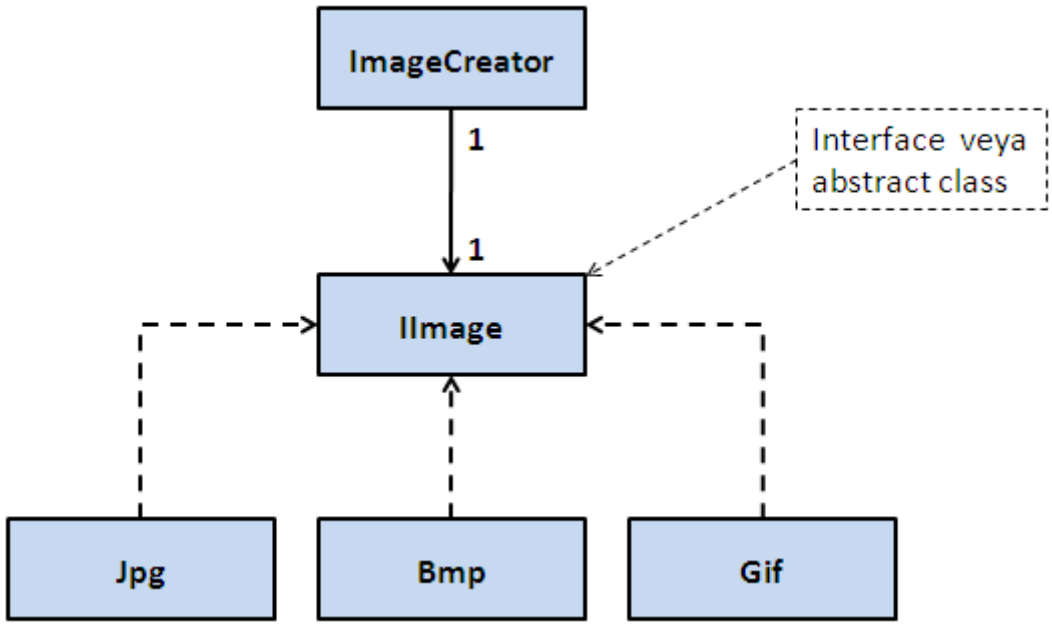
```
namespace Problem
{
   class ImageCreator
   {
      ImageBase _image = null;
      public ImageCreator(ImageBase obj)
      {
         _image = obj;
      }
      public void Randomize()
      {
         if (_image is Bmp)
            ((Bmp)_image).Randomize();
         else if (_image is Jpg)
            ((Jpg)_image).Randomize();
         else if (_image is Tif)
            ((Tif)_image).Randomize();
         else
            Console.WriteLine("Geçersiz format");
      }
      public void Draw()
      {
         if (_image is Bmp)
            ((Bmp)_image).Draw();
         else if (_image is Jpg)
            ((Jpg)_image).Draw();
         else if (_image is Tif)
            ((Tif)_image).Draw();
         else
            Console.WriteLine("Geçersiz format");
      }
   }

   class ImageBase
   {
   }

   class Bmp
      :ImageBase
   {
      public void Randomize()
      {
         Console.WriteLine("Random bitmap");
      }
      public void Draw()
```

```
        {
            Console.WriteLine("Draw bitmap");
        }
    }

    class Jpg
        :ImageBase
    {
        public void Randomize()
        {
            Console.WriteLine("Random Jpg");
        }
        public void Draw()
        {
            Console.WriteLine("Draw Jpg");
        }
    }

    class Tif
        :ImageBase
    {
        public void Randomize()
        {
            Console.WriteLine("Random Tif");
        }
        public void Draw()
        {
            Console.WriteLine("Draw Tif");
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ImageCreator creator = new ImageCreator(new Jpg());
            creator.Randomize();
            creator.Draw();

            creator = new ImageCreator(new Tif());
            creator.Randomize();
            creator.Draw();
        }
    }
}
```

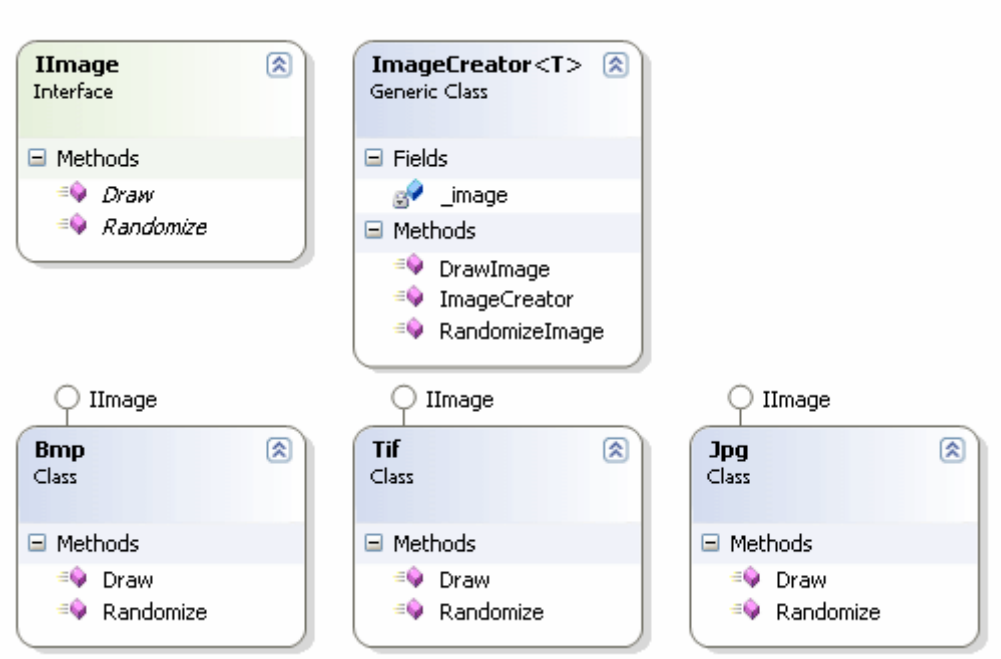
Kodun amacına göre farklı formatta Image tiplerini oluşturan bir sınıf söz konusudur. Bu sınıf içerisinde yer alan **Randomize** ve **Draw** isimli metodlar parametre olarak **ImageBase** tipinden referanslar almaktadır. Her iki metodda kendi içerisinde, gelen tipin **Jpg, Bmp** veya **Tif** olup olmadığına bakarak işlemler yapmaktadır*(Kapalılık ilkesi zaten if kısmında bozulmaktadır)*Hatta tip tespitinden sonra doğru **Randomize** yada **Draw** metodunu çağırabilmek için bir **Cast** işlemininde uygulandığını görebiliriz. Tabiki normal şartlarda bu tip bir tasarımı tercih etmeyiz, etmemeliyiz. Nitekim söz konusu tasarımın şu sorunları doğuracağı ortadadır.

- Yeni bir resim formatı sisteme eklenmek istendiğinde **ImageCreator** sınıfı içerisinde yer alan **Randomize** ve **Draw** metodlarında yer alan **if** koşullarına ilaveler yapılması gerekmektedir. Buda üretici sınıf koduna müdahele edilmesi anlamına gelmektedir.
- Her yeni imaj eklenişinde unit testlerininde(eğer hazırlanmışlarda) tekrardan tasarlanması veya oluşturulması gerekir. özellikle **UnitTest'** i yapılmış olan bir kod parçasında tekrardan değişikliğe gidilmek zorunda kalınması, testin yeniden kurgulanmasınıda gerektirecektir. En azından eski teste olan güveni sorgulatacaktır.

Bu sonuçlara göre **ImageCreator** sınıfı için **Closed** bir yapı sağlanamadığını ifade edebiliriz. Bir başka deyişle modifiyeye açık(ama olmaması gereken) bir tip söz konusudur. Peki öyleyse **Open Closed** prensibine uygun olarak kod nasıl tasarlanabilir.
önce **UML** şemasındaki düzenlememizi yapalım.



Görüldüğü gibi ImageCreator ile Jpg, Bmp, Gif ve benzeri resim sınıflar arasındaki kuvvetli bağ ortadan kaldırılmış, kurallar interface tipine yıkılmıştır. Peki ya bu şemayı C# tarafında nasıl uygulayabiliriz. İşte örnek Console uygulaması kodlarımız.




```
using System;

namespace Solution
{
   public class ImageCreator<T>
      where T:IImage
   {
      private T _image;

      public ImageCreator(T img)
      {
         _image = img;
      }

      public void RandomizeImage()
      {
         _image.Randomize();
      }
      public void DrawImage()
      {
         _image.Draw();
      }
   }

   public interface IImage
   {
      void Randomize();
```

```
    void Draw();
}

class Bmp
    :IImage
{
    public void Randomize()
    {
        Console.WriteLine("Random bitmap");
    }
    public void Draw()
    {
        Console.WriteLine("Draw bitmap");
    }
}

class Jpg
    :IImage
{
    public void Randomize()
    {
        Console.WriteLine("Random Jpg");
    }
    public void Draw()
    {
        Console.WriteLine("Draw Jpg");
    }
}

class Tif
    :IImage
{
    public void Randomize()
    {
        Console.WriteLine("Random Tif");
    }
    public void Draw()
    {
        Console.WriteLine("Draw Tif");
    }
}

class Program
{
    static void Main(string[] args)
```

```
        {
            ImageCreator<Bmp> creator = new ImageCreator<Bmp>(new Bmp());
            creator.RandomizeImage();
            creator.DrawImage();

            ImageCreator<Tif> creator2 = new ImageCreator<Tif>(new Tif());
            creator2.RandomizeImage();
            creator2.DrawImage();
        }
    }
}
```

Dikkat edileceğiz üzere Jpg, Gif ve Bmp isimli sınıflar **IImage** arayüzünü uygulamaktadır. Diğer taraftan **ImageCreator** sınıfı kendi içerisinde **IImage** arayüzünü ele alarak**Randomize** ve **Draw** operasyonlarını icra etmektedir*(Neden dejavu yaşayacağınızı anladınız sanırım* 😃 *)* Buna göre **ImageCreator** tipinin yapısını bozmadan sisteme yeni resim formatları eklenmesi sağlanabilir. Dolayısıyla ImageCreator sınıfı **OCP** uyumlu hale getirilmiştir.

Ve bu blog girişinin özlü Cümlesi: OCP ilkesi, sınıf, metod gibi **OOP** varlıklarının **genişletilmeye açık(Open)** ancak **düzenlenmeye kapalı(Closed)** olması gerektiğini savunur. özellikle müşteriden gelen istekler nedeniyle sık sık genişletilmesi gereken varlıklarda, genişletmenin kod içerisinde mümkün olduğunca az meydana gelmesine çalışmak gerekir. İlkenin amacını, yeni fonksiyonelliklerin kazandırılması için minimum kod değişikliğinin yapılması olarak düşünebiliriz.

Bu arada kod ve resimlerin bazı yerlerinde Tif bazı yerlerinde Gif formatlarını ele aldığımı farkettim. Ama son yaptıpımız OCP tasarımına göre hiç sorun değil 😉 Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

OCP.rar (42,53 kb)

Tasarım Prensipleri - Loose Coupling (2009-06-24T10:46:00)

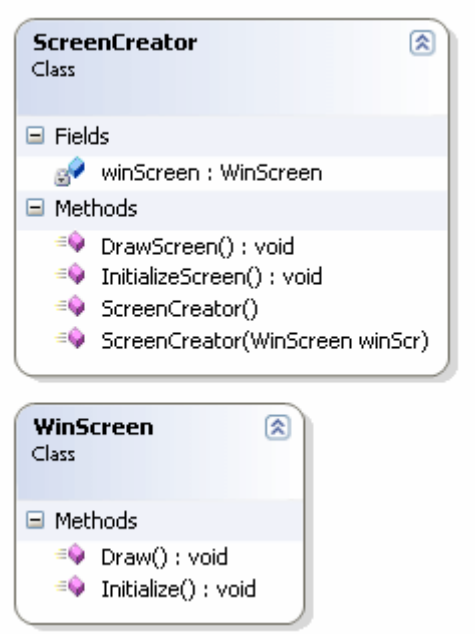
design principles,

Merhaba Arkadaşlar

Yazılım teknolojilerinde uygulanan tekniklerin çoğunda temel tasarım prensipleri sıklıkla ele alınmaktadır/Alınmalıdı. örneğin **eXtreme Programming, Aspect Oriented Programming vb...** yazılım geliştirme tekniklerinde bu prensiplerin çoğuna rastlayabiliriz. Bu yazı ile birlikte **Temel Tasarım Prensiplerinin** incelenmesine başlıyor olacağız ki özellikle büyük çaplı projelerde bu tip disiplinler büyük bir öneme sahiptir.

Enterprise yazılım süreçlerinde en çok zorlanılan noktalardan biriside müşteri ihtiyaçlarının sürekli olarak en hızlı şekilde karşılanması gerekliliğidir. Bu durumda yazılımın bir süre sonra kendinden geçerek dağılmasını engellemek gerekmektedir. Bu anlamda uygulanan süreçlerin içerisinde önem arz eden noktalardan biriside, kullanılacak yazılım disiplinleridir. Bu nedenle Hatta bu presipler içerisinde **tasarım desenlerininde(Design Patterns)** önemli bir yeri vardır. Serimizin bu ilk yazısına en kolay prensip ile başlıyor olacağız; **Zayıf Bağlılık(Loose Coupling) prensibi.**

Konuyu kavramamın en güzel yolu tahmin edeceğiniz üzere basit bir örnek üzerinden ilerlemek olacaktır. Bu amaçla aşağıdaki **Console** uygulamasını geliştirdiğimizi düşünelim.




```
using System;

namespace Problem
{
   // WinScreen tipine ait nesne örneklerinin yaratıcısı olan sınıf
   class ScreenCreator
   {
      private WinScreen winScreen = null;

      public ScreenCreator()
      {

      }
      public ScreenCreator(WinScreen winScr)
      {
         winScreen = winScr;
      }
```

```
        public void InitializeScreen()
        {
            winScreen.Initialize();
        }
        public void DrawScreen()
        {
            winScreen.Draw();
        }
    }

    // Windows tabanlı sistemleri için düşünülen bir ekran
    class WinScreen
    {
        public void Initialize()
        {
            Console.WriteLine("WinScreen initialize işlemi");
        }
        public void Draw()
        {
            Console.WriteLine("WinScreen Draw işlemi");
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Ekran üretme işlemi başladı");

            ScreenCreator creator = new ScreenCreator(new WinScreen());
            creator.InitializeScreen();
            creator.DrawScreen();
        }
    }
}
```

Bu örnek uygulama **ScreenCreator** ve **WinScreen** isimli iki sınıf ve arasındaki ilişki ele alınmaktadır. **ScreenCreator** sınıfına ait nesne örnekleri, aşırı yüklenmiş yapıcı metoda gelen **WinScreen** referansları üzerinden **Initialize** ve **Draw** isimli fonksiyonelliklerin uygulanabilmesine imkan tanımaktadır. Böylece bir Windows Formunun başlatılması ve ekrana çizilmesi işlemlerinin gerçekleştirileceği düşünülmektedir.

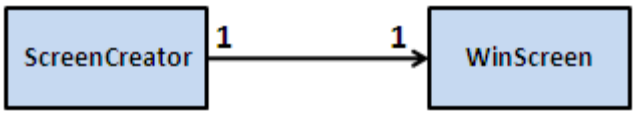
Uygulamayı çalıştırdığımızda aşağıdaki sonuçları alırız.

```
C:\WINDOWS\system32\cmd.exe
Ekran üretme işlemi başladı
WinScreen initialize işlemi
WinScreen Draw işlemi
Press any key to continue . . .
```

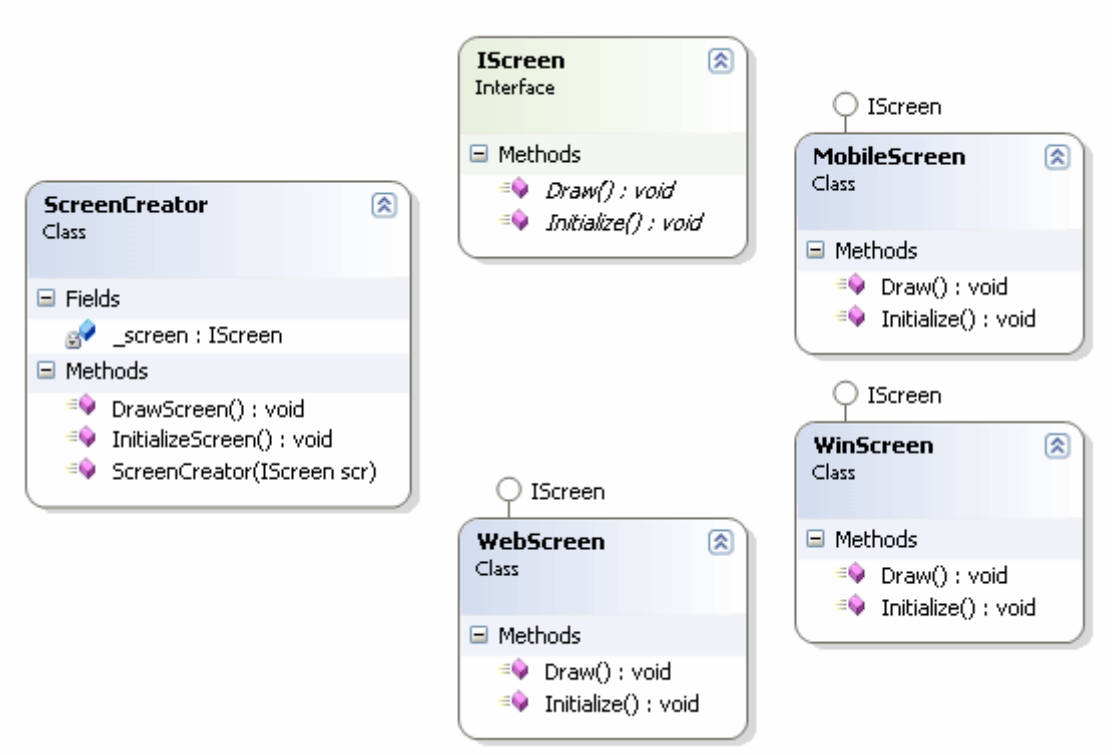
Aslında kodu dikkatlice incelediğimizde ve yazılım tasarımına baktığımızda bazı sıkıntılar olduğunu görebiliriz. örneğin;

- ScreenCreator nesne örnekleri tek başlarına anlamsızdır. Kullanışlı olmaları için WinScreen sınıfı ile birlikte ele alınmaları gerekir.
- WinScreen sınıfı içerisinde yapılacak değişiklikler ScreenCreator tipinide etikeyecektir.
- ScreenCreator sadece WinScreen tipini ele almaktadır. Oysaki web ekranları, WPF ekranları, mobile ekranlar veya x ekranlar için tasarlanmış sınıfların ScreenCreator tarafından ele alınması mümkün değildir. Kuvvetli bağ buna müsade etmemektedir.

Bu sonuçlara göre **ScreenCreator** ve **WinScreen** nesne örnekleri arasında aşağıdaki **UML** diagramında görülen ilişkinin olduğunu düşünebiliriz.



İşte **Loose Coupling** ilkesi söz konusu sorunlara çözüm getirmeyi kolaylaştırmaktadır. Aslında nesneler arasındaki bu kuvvetli bağların kaldırılması işi kökten çözebilir. Ne varki **OOP(Object Oriented Programming)** dillerinde bu mümkün değildir. Dolayısıyla bağı zayıflaştırmaya çalışmak üzere bir takım işlemler yapılabilir. **.Net** tarafından olaya baktığımızda **interface** veya **abstract** tipleri kullanarak zayıf bağlı bir ortam oluşturabiliriz. İşte Loose Coupling prensibini uygulayan yeni kod içeriğimiz.

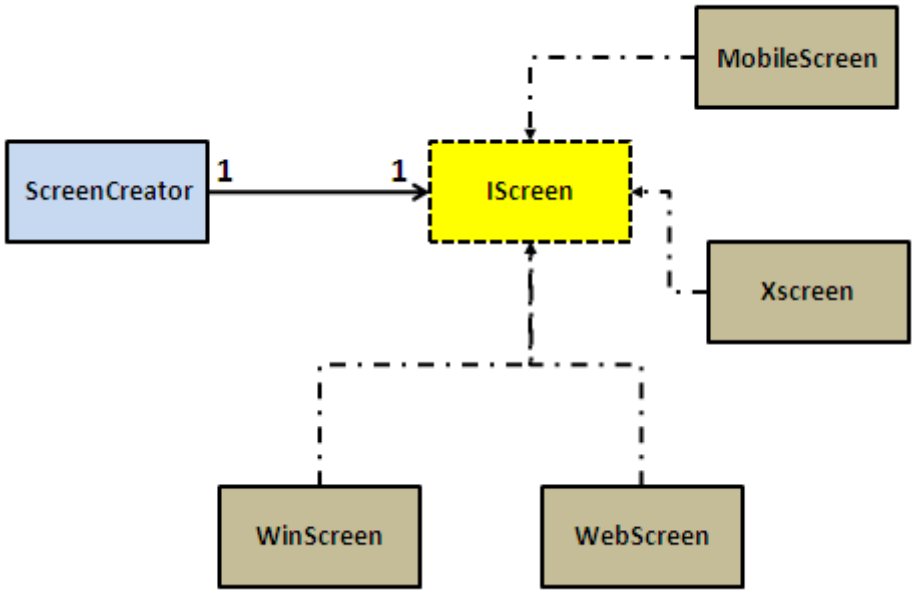



```
using System;

namespace Solution
{
   public interface IScreen
   {
      void Initialize();
      void Draw();
   }

  public class WinScreen
     : IScreen
  {
     #region IScreen Members

     public void Initialize()
     {
        Console.WriteLine("WinScreen Initialize işlemi");
     }

     public void Draw()
     {
        Console.WriteLine("WinScreen draw işlemi");
     }
```

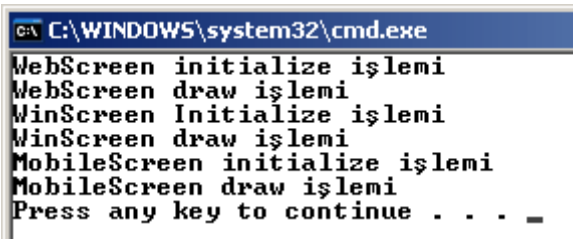
```
    #endregion
}

public class WebScreen
    :IScreen
{
    #region IScreen Members

    public void Initialize()
    {
        Console.WriteLine("WebScreen initialize işlemi");
    }

    public void Draw()
    {
        Console.WriteLine("WebScreen draw işlemi");
    }

    #endregion
}

public class MobileScreen
    : IScreen
{
    #region IScreen Members

    public void Initialize()
    {
        Console.WriteLine("MobileScreen initialize işlemi");
    }

    public void Draw()
    {
        Console.WriteLine("MobileScreen draw işlemi");
    }

    #endregion
}

public class ScreenCreator
{
    private IScreen _screen;

    public ScreenCreator(IScreen scr)
    {
        _screen = scr;
```

```
        }
        public void InitializeScreen()
        {
            _screen.Initialize();
        }
        public void DrawScreen()
        {
            _screen.Draw();
        }
    }

    class Program
    {
        static void Main(string[] args)
        {
            ScreenCreator creator = new ScreenCreator(new WebScreen());
            creator.InitializeScreen();
            creator.DrawScreen();

            creator = new ScreenCreator(new WinScreen());
            creator.InitializeScreen();
            creator.DrawScreen();

            creator=new ScreenCreator(new MobileScreen());
            creator.InitializeScreen();
            creator.DrawScreen();
        }
    }
}
```

Yeni tasarımızda **Initialize** ve **Draw** isimli operasyonlar **IScreen** isimli bir arayüz içerisinde bildirilmiş ve bu arayüzden türeyen tiplerde ayrı ayrı uygulanmışlardır. **ScreenCreator** sınıfı ise kendi içerisinde **IScreen** interface referansını ele almaktadır. Buna göre **IScreen** arayüzünü implemente eden her sınıf, **ScreenCreator** tarafından kullanılabilir. Bir başka deyişle**ScreenCreator** sınıfının, üreteceği ekran ile ilişkili herhangibir bilgiye sahip olmasınada gerek yoktur. Sadece **Interface** referansının **Initialize** ve **Draw** operasyonlarını çağırmaktadır. Diğer yandan **IScreen** arayüzünü implemente eden tipler içerisinde yapılacak değişimler, **ScreenCreator** sınıfını doğrudan etkilemeyecektir. Hatta bu etki en aza indirgenmiş olacaktır. Olaya basit bir UML şeması ile baktığımızdaysa tipler arasındaki ilişkileri daha net görebiliriz.



Programı çalıştırdığımızda ise aşağıdaki sonuçları elde ederiz.



Böylece **Temel Tasarım Prensiplerinden** birisi ve bana kalırsa belkide en basiti olan **Loose Coupling** ilkesini incelemiş olduk. İlerleyen yazılarımızda diğer prensiplerede bakıyor olacağız. örneğin **Open Closed, Single Responsibility, Liskov Substitution vb...** Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

LooseCoupling.rar (39,67 kb)

Caching Application Block Merakı (2009-06-21T02:52:00)

enterprise library,wcf,

Merhaba Arkadaşlar,

Az önce 1966 yılında çevrilmiş olan ve küçüklüğümde bol bol izlediğim nefis bir filmi belkide 179ncu kez tekrardan seyrettim. Eskiler aşağıdaki resimden hangi film olabileceğini tahmin edebilirler.

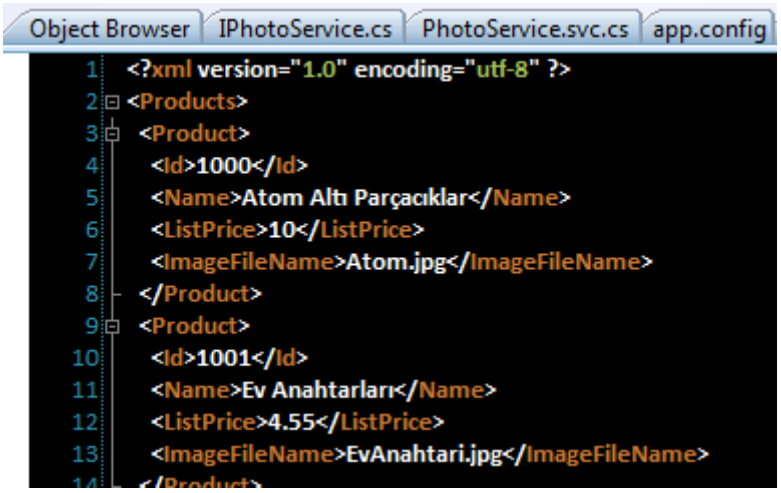
Yeni nesilden seyretmeyen varsa eğer **The Good, The Bad and The Ugly** filmini mutlaka tedarik edip izlesinler. Peki bunun anlatacağım konu ile bir ilgisi var mı? Hayır yok. 😎 Sadece off-topic bir giriş yapmak istedim.

Bu yazımda sizlere bahsetmek istediğim konu bir süredir boş vakitlerimde araştırıp incelediğim **Caching Application Block** yapısıdır. Açık kaynak olarak sunulan **Enterprise Library**ürün ailesinin bir parçası olan bu bloğu, uygulamalarımızda performansı arttırmak adına ele alabiliriz. Bilindiği üzere günümüz uygulamalarında sıklıkla tekrar eden pek çok **kıstas(Concern)** bulunmaktadır. örneğin **hata yönetimi(Error Handling), loglama(Logging), şifreleme işlemleri(Cryptography), güvenlik(Security), doğrulama(Validation)** veya **veri erişim işlemleri(Data Access)** bu kıstaslara örnek olarak gösterilebilir. Haliyle bu kavramlar çoğunlukla uygulamadan bağımsız olmaktadır. Nitekim uygulama çeşidi değişse bile, bu kıstasların bir kısmını veya tamamını kullanmak zorunda kalabiliriz. öyleyse uygulamalara bu kıstasların enjekte edilmesi sırasında gerekli hazırlıkları tekrar tekrar yapmamıza gerek bırakmayacak modellere ihtiyacımız vardır. *(**Aspect Oriented Programming** modelininde çözüm getirdiği sorunlardan birisidir bu aslında.)* Diğer taraftan, **Enterprise Library** gibi kütüphaneleri ele alaraktanda, bu kıstasların tamamını veya bir kısmını istediğimiz uygulamalarda değerlendirebiliriz. Bu sayede sürekli tekrar eden temel işlemlerle uğraşmak zorunda kalmayarak tamamen iş mantığına odaklanmamız mümkün olabilir. üstelik Enterprise Library açık kaynak kodlu olduğundan, dilersek genişletebilir veya özelleştirebiliriz.

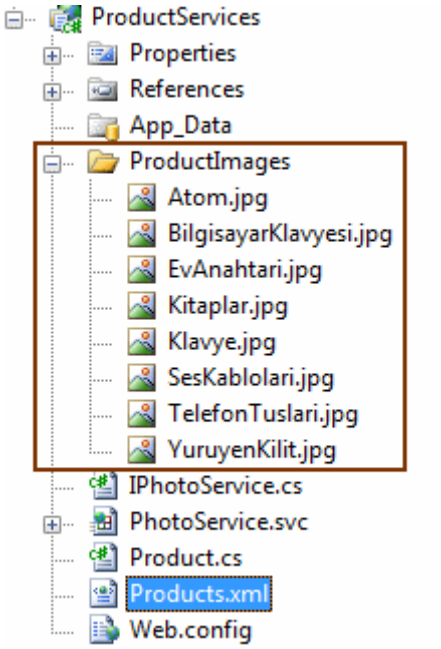
Caching Application Block' un nasıl kullanıldığını anlatmak için oldukça hevesliyim.*(Dilerseniz benden çok daha iyi bir öğretici olan Hands-On-Lab' leride kullanabilirsiniz ki şiddetle tavsiye ederim)* İlk etapta bize örnek bir senaryo gerekiyor. Ben elimde bir WCF servisi olduğunu, bu servisin XML tabanlı bir veri kaynağını kullanarak istemcilere ürün bilgilerini verdiğini düşündüm. Bununla birlikte, ürünlerin resim bilgileride fiziki dosyalarda ve sunucu tarafında yer almaktadır. İstemciler, listeleri çektikten sonra dilerlerse istedikleri bir ürünün resmini servisten talep edebilmektedir. Tabiki bu senaryoda performansı etkiliyen farklı faktörler vardır. Herşeyden önce, servis yönelimli bir uygulama söz konusu olduğundan, istemci ve sunucu arasında taşınacak resim ve boyutu iletişim hızını ve performansını doğrudan etkileyecektir. *(Burada performans için resmin MIME protokolüne göre taşınması veya parçalara bölünerek aktarılması düşünülebilir.)* Diğer taraftan, istemcinin bir ürün resmini talep etmesi durumunda, sunucu tarafında resmin fiziki dosyadan tedarik edilerek istemci tarafına gönderilecek şekilde hazırlanması durumunda da performans kaybı söz konusu olacaktır. İşte biz **Caching Application Block** yapısını, servis tarafında resmin hazırlanması

aşamasında ele alabiliriz. Yinede şunu vurgulamakta yarar vardır. WCF servisi web tabanlı olarak yayınlandığında pekala Asp.Net motorunun var olan önbellekleme modüllerinide kullanılabilir. Dolayısıyla senaryomuzu sadece **Caching Application Block'** un kullanımını öğrenmek amacıyla geliştirdiğimizi göz önüne almamızda yarar vardır. Hatta istemcinin asenkron olarak erişmesi veya servis tarafında paralel hesaplamalar yapılması gibi senaryoları hiç işin içerisine katmıyoruz bile.

önce servis tarafını ele alalım. Servis tarafında veri kaynağı olarak Products.xml isimli bir dosya kullanılmaktadır.



XML içeriğinde basit olarak ürünün Id değeri, adı, liste fiyatı ve resim dosyası adı bilgileri tutulmaktadır. Resimler ise WCF Service uygulamasında **ProductImages** isimli bir fiziki klasörde yer almaktadır.



XML içeriğini istemci tarafında sunarken Product isimli serileştirilebilir bir tipten yararlanıyor olacağız.

```
using System.Runtime.Serialization;

namespace ProductServices
{
   [DataContract]
   public class Product
   {
      [DataMember]
      public int ProductId { get; set; }
      [DataMember]
      public string Name { get; set; }
      [DataMember]
      public decimal ListPrice { get; set; }
   }
}
```

Servis sözleşmesi ise aşağıdaki kod parçasında olduğu gibidir.

```
using System.Collections.Generic;
using System.Drawing;
using System.ServiceModel;

namespace ProductServices
{
   [ServiceContract]
   public interface IPhotoService
   {
      [OperationContract]
      List<Product> GetProducts();

      [OperationContract]
      byte[] GetPhoto(int productId);
   }
}
```

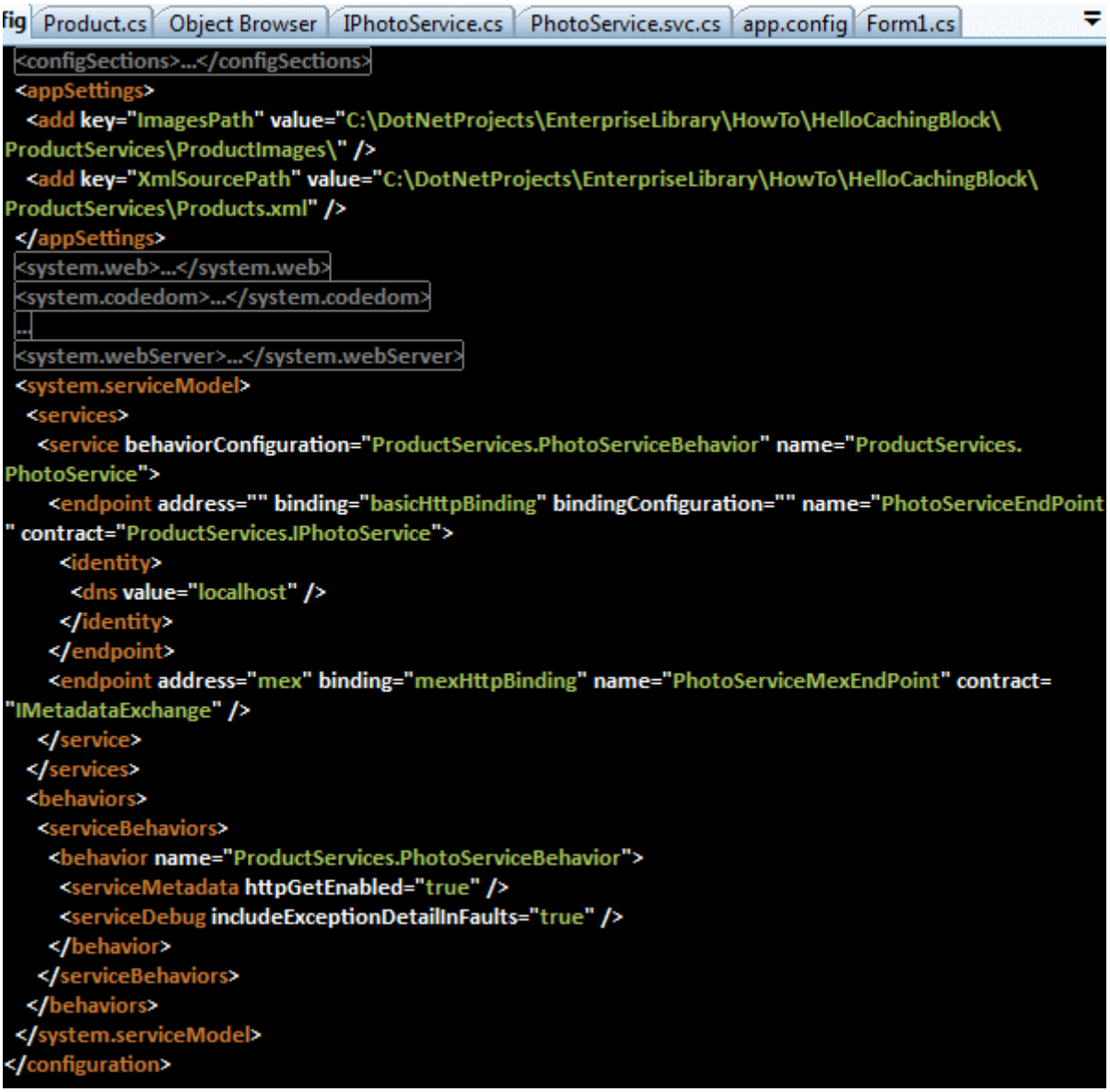
Sözleşmeyi uygulayan tipimizin kodları ise şu şekildedir.

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.IO;
using System.Linq;
using System.Xml.Linq;
```

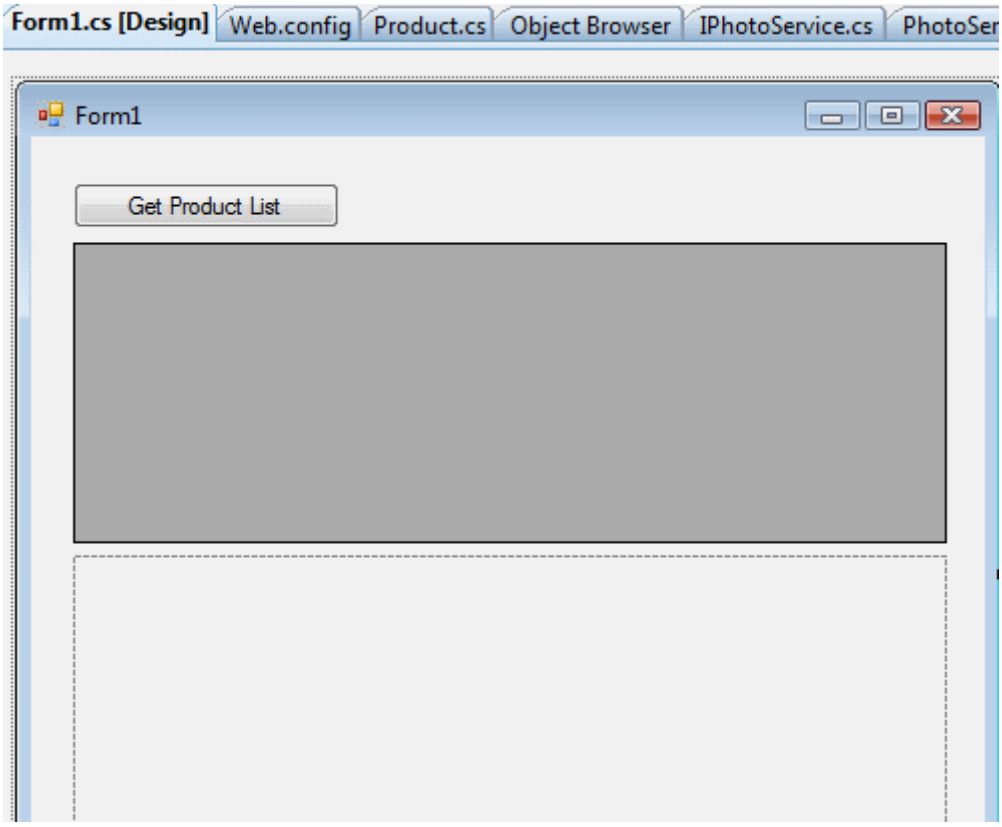
```
namespace ProductServices
{
   public class PhotoService
      : IPhotoService
   {
      public List<Product> GetProducts()
      {
         XDocument doc =
XDocument.Load(ConfigurationManager.AppSettings["XmlSourcePath"]);

         List<Product> products = (from p in doc.Element("Products").Elements("Product")
                  select new Product
                  {
                     ProductId=Convert.ToInt32(p.Element("Id").Value),
                     Name=p.Element("Name").Value,
                     ListPrice=Convert.ToDecimal(p.Element("ListPrice").Value)
                  }).ToList<Product>();

         return products;
      }

      #region IPhotoService Members


      public byte[] GetPhoto(int productId)
      {
         XDocument
doc=XDocument.Load(ConfigurationManager.AppSettings["XmlSourcePath"]);

         string imageFileName = (from p in doc.Element("Products").Elements("Product")
                        where p.Element("Id").Value == productId.ToString()
                        select p.Element("ImageFileName").Value).Single();

         string imagePath =
Path.Combine(ConfigurationManager.AppSettings["ImagesPath"], imageFileName);

         return File.ReadAllBytes(imagePath);
      }

      #endregion
   }
}
```

Görüldüğü üzere, **XML** içeriğini sorgulama kısımlarında **XLINQ** ifadelerinden yararlanılmaktadır. **GetProducts** metodu, **List<Product>** koleksiyonu tipinden bir

referans döndürmektedir. Bununla birlikte herhangibir ürünün resmi, **GetPhoto** metodu yardımıyla **byte[]** dizisi olacak şekilde üretilmektedir. **Products.xml** dosyası ve resimlerin kök adres bilgileri**web.config** dosyasında **appSettings** kısmında tutulmaktadır. Bu nedenle söz konusu konfigurasyon bilgilerin alınabilmesi için, **ConfigurationManager** tipinden yararlanıldığı görülmektedir. Servisimiz basicHttpBinding bağlayıcı tipi üzerinden sunulmaktadır. Dolayısıyla web.config dosyasındaki servis ayarları aşağıda görüldüğü gibidir.



Peki ya istemci tarafı? Bu uygulamayı aşağıdaki tasarıma sahip basit bir WinForms programı olarak düşünebiliriz aslında.

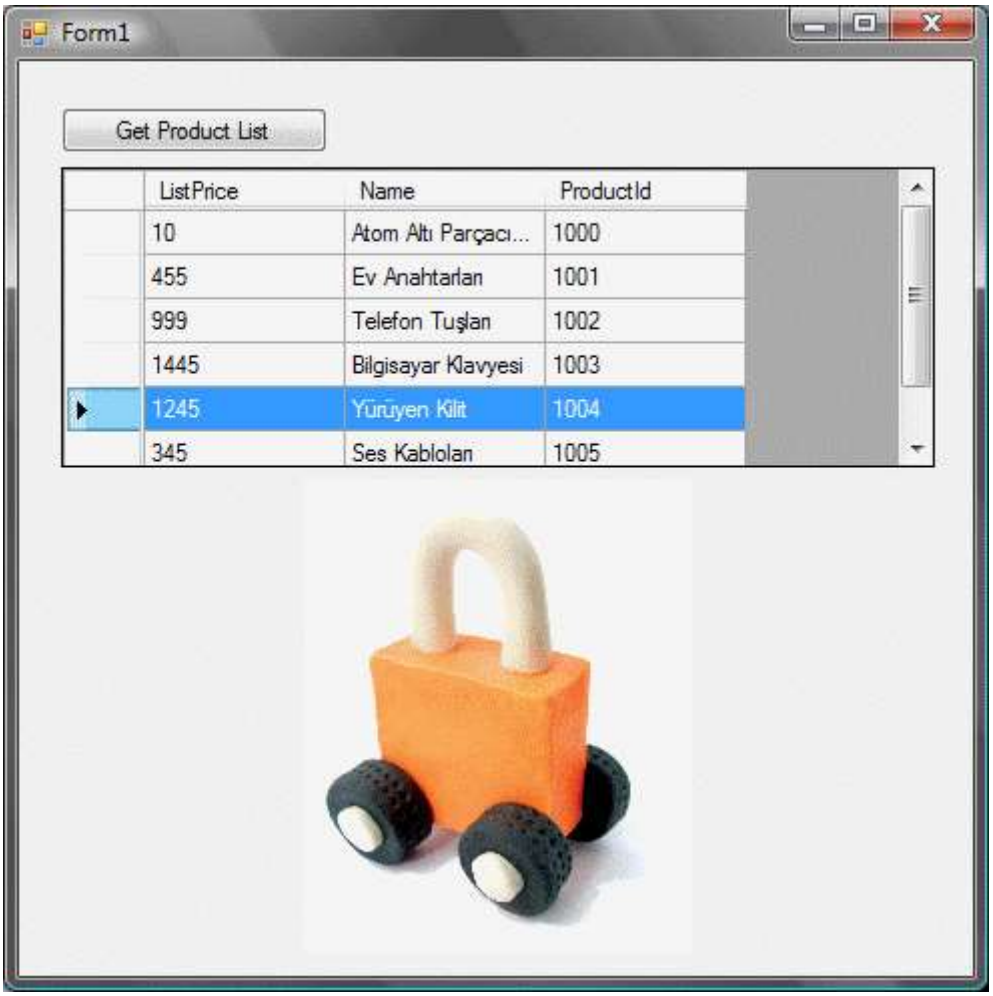


İstemci uygulamaya servis referansının eklenmesinin ardından kodlarıda aşağıdaki gibi geliştirebiliriz.

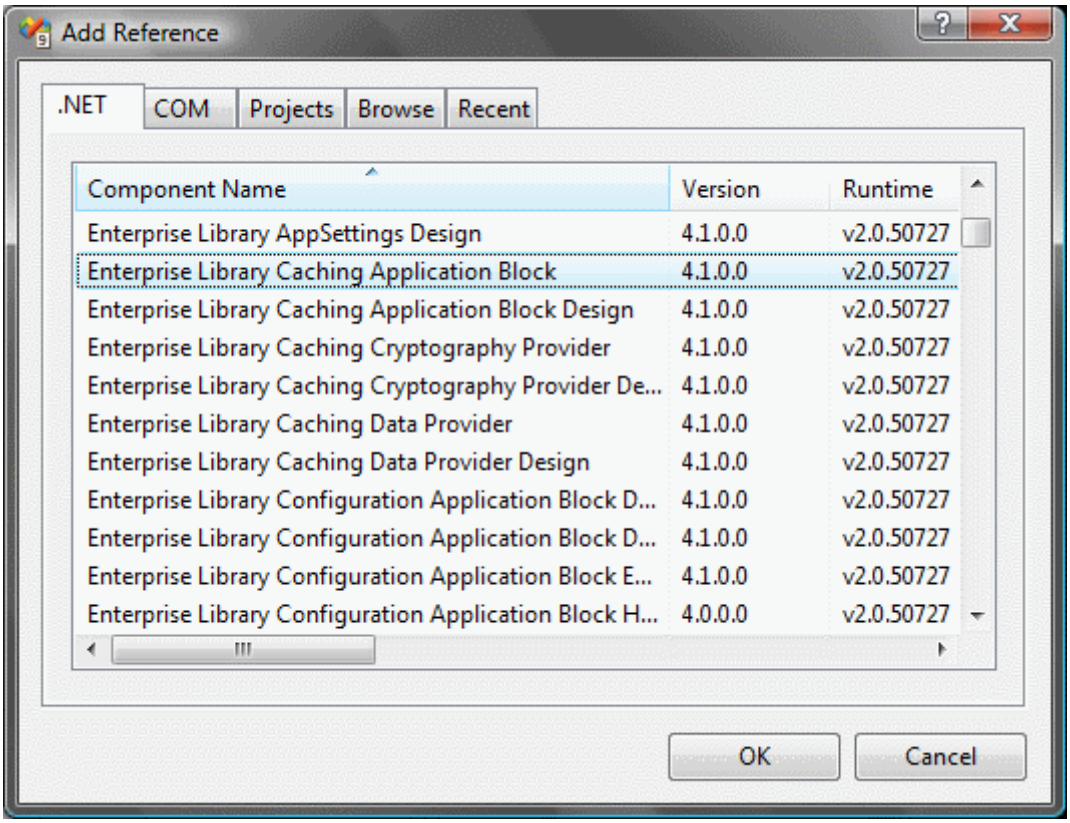
```
using System;
using System.Drawing;
using System.IO;
using System.Windows.Forms;
using ClientApp.ProductServicesRef;

namespace ClientApp
{
   public partial class Form1 : Form
   {
      PhotoServiceClient proxy = null;

      public Form1()
      {
         InitializeComponent();
         proxy = new PhotoServiceClient();
      }

      private void btnGetProductList_Click(object sender, EventArgs e)
      {
```

```
            grdProducts.DataSource=proxy.GetProducts();
        }

        private void grdProducts_CellClick(object sender, DataGridViewCellEventArgs e)
        {
            int productId = Convert.ToInt32(grdProducts[2, e.RowIndex].Value);

            byte[] byteArray= proxy.GetPhoto(productId);
            using (MemoryStream stream = new MemoryStream(byteArray))
            {
                pcbImage.Image = Image.FromStream(stream);
            }
        }
    }
}
```

Kullanıcılar **GetProductList** düğmesini kullanarak servisten ürün listeni çekmektedir. çekilen ürünler **DataGridView** kontrolüne aktarılmaktadır. Grid üzerinden herhangibir satıra tıklanıldığında ise seçilen ürünün ProductId değerine göre servisten resim bilgisi talep edilmektedir. Aşağıda çalışma zamanındaki örnek sonuçlardan birisi yer almaktadır.

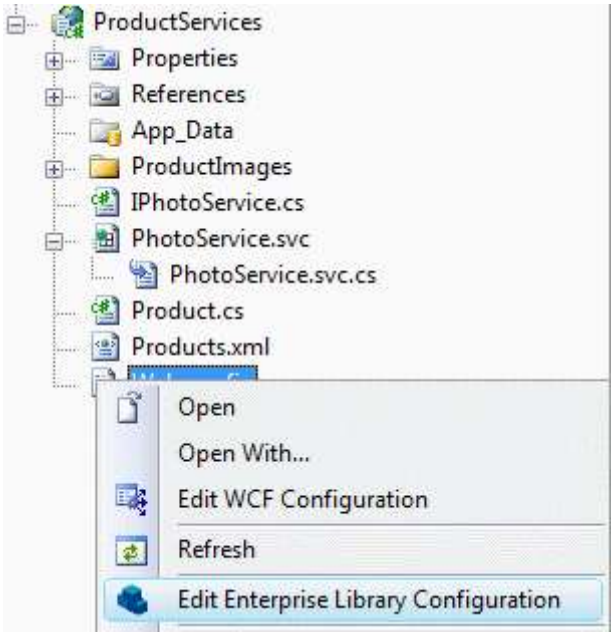


Burada hemen şu noktayı vurgulamak isterim. İstemci tarafına gönderilen **byte** dizisinin boyutu, yine istemci tarafındaki **app.config** dosyasında, **readerQuotas** elementi içerisindeki **maxArrayLength**özelliği ile sınırlandırılmıştır. Resimlerin boyutlarına göre bu değerin arttırılması gerekebilir ki benim örneğimde bu arttırım yapılmak zorunda kalınmıştır. Nihayetinde ön hazırlıklarımız tamamlanmıştır. Artık servis tarafında kullanmak istediğimiz **Caching Application Block** ile ilişkili hazırlıklara başlayabiliriz. öncelikle servis uygulamasına kullanılmak istenen Enterprise Library bloğu ile ilgili **assembly** referansının eklenmesi gerekmektedir.

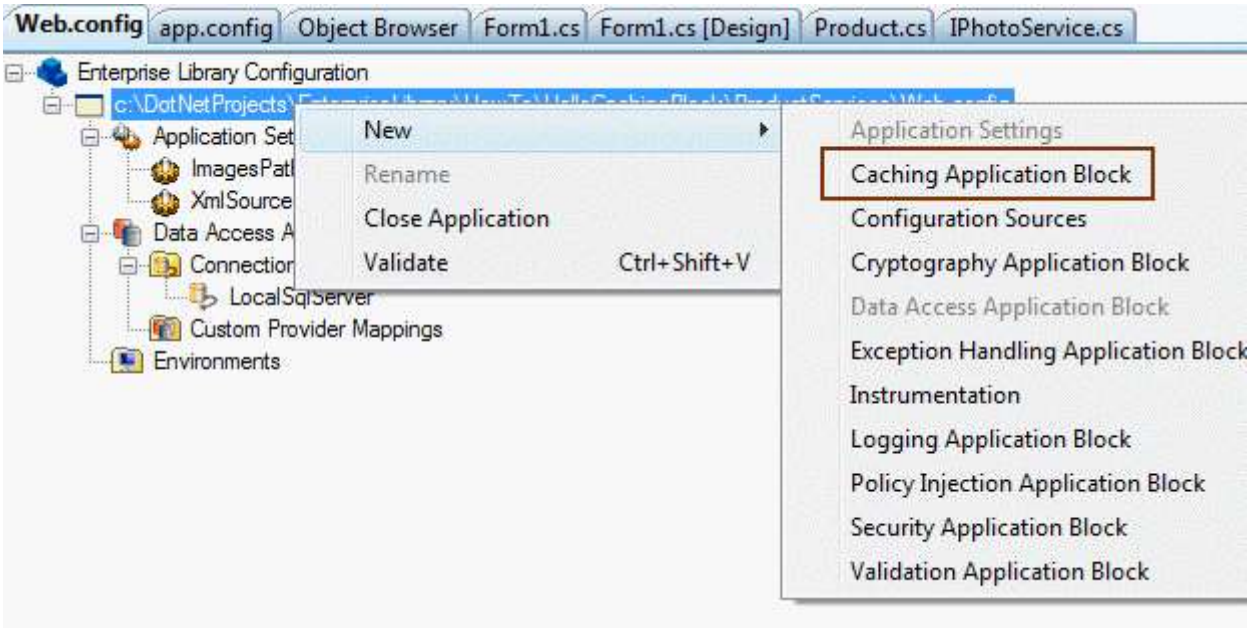


Böylece kod içerisinde, **Caching Application Block** ile ilişkili yönetimli kodlar ele alabileceğiz. **Caching Application Block**, ön belleğe alma işlemlerinde varsayılan olarak kullanıldığı host uygulamanın çalıştığı sistem belleğini ele almaktadır. Ancak istenirse saklama işlemleri için farklı bir kaynağın*(örneğin fiziki disk)* kullanılması sağlanabilir. Bunula birlikte, ön bellekte tutulacak maksimum eleman sayısınıda belirleyebiliriz. İyi ama bu ayarları nerede yapacağız? Tahmin edeceğiniz üzere host uygulamanın konfigurasyon dosyası içerisinde. Neyseki**Enterprise Library** kurulumlarından sonra, **Visual Studio 2008** için görsel bir arabirim gelmektedir. Böylece gerekli ayarları kolayca yapabiliriz.

İlk etapta servis uygulamasındaki **web.config** dosyasını **Edit Enterprise Library Configuration** ile açalım. *(İstenirse tüm ayarlamalar konfigurasyon dosyası içerisinden ellede yapılabilir.)*



Sonrasında ise **Caching Application Block** için gerekli **XML** sekmesini aşağıdaki şekildende görülebileceği gibi kolayca ilave edebiliriz.



Bu işlemlerin ardından varsayılan olarak web.config dosyasının içeriği aşağıdaki gibi olacaktır.

```
 <cachingConfiguration defaultCacheManager="Cache Manager">
   <cacheManagers>
     <add expirationPollFrequencyInSeconds="60"
maximumElementsInCacheBeforeScavenging="1000"
       numberToRemoveWhenScavenging="10" backingStoreName="Null Storage"
       type="Microsoft.Practices.EnterpriseLibrary.Caching.CacheManager, Microsoft.Practices.EnterpriseLibrary.Caching, Version=4.1.0.0, Culture=neutral,
```

```
PublicKeyToken=31bf3856ad364e35"
      name="Cache Manager" />
   </cacheManagers>
   <backingStores>
    <add encryptionProviderName="" type="Microsoft.Practices.EnterpriseLibrary.Cach
ing.BackingStoreImplementations.NullBackingStore,
Microsoft.Practices.EnterpriseLibrary.Caching, Version=4.1.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
      name="Null Storage" />
   </backingStores>
  </cachingConfiguration>
```

Tabiki özellikler penceresindende pek çok ayarlama yapılabilir. Söz gelimi **Protection Provider** ile ön bellekte tutulacak nesnelerin şifrelenmesi için hangi sağlayıcının kullanılacağı belirlenebilir*(şu andaki örneğimizde herhangibir şifreleme sağlayıcısı kullanılmamaktadır)*. Peki bunlar yeterli midir? Elbetteki değildir. ön bellekte kimi tutacağız? ön bellekte tutmak istediğimiz nesne referansını nasıl ekleyecek veya nasıl çekeceğiz? Bu durumda **GetPhoto** metodunun içeriğini aşağıdaki gibi düzenlememiz yeterli olacaktır.

```
public byte[] GetPhoto(int productId)
      {
         XDocument
doc=XDocument.Load(ConfigurationManager.AppSettings["XmlSourcePath"]);

         string imageFileName = (from p in doc.Element("Products").Elements("Product")
                        where p.Element("Id").Value == productId.ToString()
                        select p.Element("ImageFileName").Value).Single();

         string imagePath =
Path.Combine(ConfigurationManager.AppSettings["ImagesPath"], imageFileName);

         byte[] imageBytes = null;

         // İlk olarak çalışma zamanında, CacheManager referansı çekilir. Bu noktada
fabrika tipinden yararlanılmaktadır.
         ICacheManager cacheManager = CacheFactory.GetCacheManager();

         // Eğer Cache koleksiyonunda, productId ile belirtilen bir referans tutulmuyorsa
         if (cacheManager[productId.ToString()] == null)
         {
            imageBytes = File.ReadAllBytes(imagePath);
            cacheManager.Add(productId.ToString(), imageBytes);
         }
         else // Eğer Cache koleksiyonunda productId anahtarına sahip bir referans var ise
```

getir

```
            imageBytes = (byte[])cacheManager[productId.ToString()];

        return imageBytes;
    }
```

Peki sistem nasıl çalışmaktadır?

İstemci bir ürün resmi talep ettiğinde, kod parçasına göre öncelikle ön bellekte olup olmadığına bakılır. Eğer ön bellekte değilse **Add** metodu yardımıyla ön belleğe ekleme işlemi yapılır. Eğer nesne ön bellekte ise, **indeksleyiciden** yararlanılarak resmin **byte[]** dizisine **cast** edilerek elde edilmesi sağlanır. Burada önemli olan noktalardan biriside şudur; servise ait host uygulama açık olduğu sürece **productId** bazlı resimler ön bellekte saklanmaya ve korunmaya devam edecektir. Ancak host uygulamanın kapatılması durumunda, ön bellek koleksiyonuda otomatik olarak temizlenmektedir. Diğer yandan ön bellekte tutulan nesnelerin tamamını bilinçli bir şekilde temizlemek istersek, **ICacheManager** referansı üzerinden **Flush** metodunun çağırılması yeterli olacaktır. Yazımı sonlandırmadan önce son olarak şu noktayada değinmek isterim; senaryomuzda **Caching** bloğunu sunucu tarafındaki servis uygulaması için ele almış bulunmaktayız. Buna göre istemci aynı resimleri talep ettiği ve servis uygulamasıda ayakta olduğu sürece, resimler ön bellekten tedarik edilecektir. Bu işlem resmin istemciye hızlı bir şekilde iletilmesini sağlamak üzere yapılmamıştır. Buna lütfen dikkat edelim. Aksine servis tarafındaki gereksiz resim okuma işlemini ekarte etmek amacıyla kullanılmıştır(Bu bloğun başka ne tip senaryolarda kullanılabileceğini düşünmenizi tavsiye ederim) Senaryomuzda elbetteki eksik kısımlar mevcuttur. örneğin istisna yönetimi(Exception Handling) sıfırdır. Asenkron erişim ile ilişkili istemci tarafında hiç bir şey yapılmamıştır. Diğer yandan resmin değişmesi halinde cache içeriğinin güncellenmesi ile ilgili bir çalışmada yapılmamıştır ki yapmaya çalışmanızı öneririm.

Şimdilik benden bu kadar. Yeni bir western filmi sonrasında tekrardan **Enterprise Library** konulu bir örnek ile görüşmek üzere...

HelloCachingBlock.rar (5,35 mb)

Organik Yazılım Günü (2009-06-19T02:21:00)

seminer,

Merhaba Arkadaşlar,

27 Haziran 2009 Cumartesi günü mezun olduğum **Yıldız Teknik üniversitesinde**, **Microsoft'** un açık kaynak kod tarafındaki çalışmaları ile ilişkili bir seminer düzenleniyor olacak. Bu seminerde bende **WCF Rest Starter Kit** paketini aktarıyor olacağım. Kısa süreli seanslar şeklinde devam edecek etkinlikte, hem sizleri

sıkmayacak hemde **Microsoft**' un açık kaynak kod çalışmaları konusunda bilgilendirecek faydalı bir etkinlik olacağı kanısındayım. Seminerde görüşmek üzere.

Kayıt olmayı unutmayın tabikide www.inetatr.org

Parallel.For Metodu için Stop, Break Kullanımı [Beta 1] (2009-06-18T18:32:00)

parallel programming,

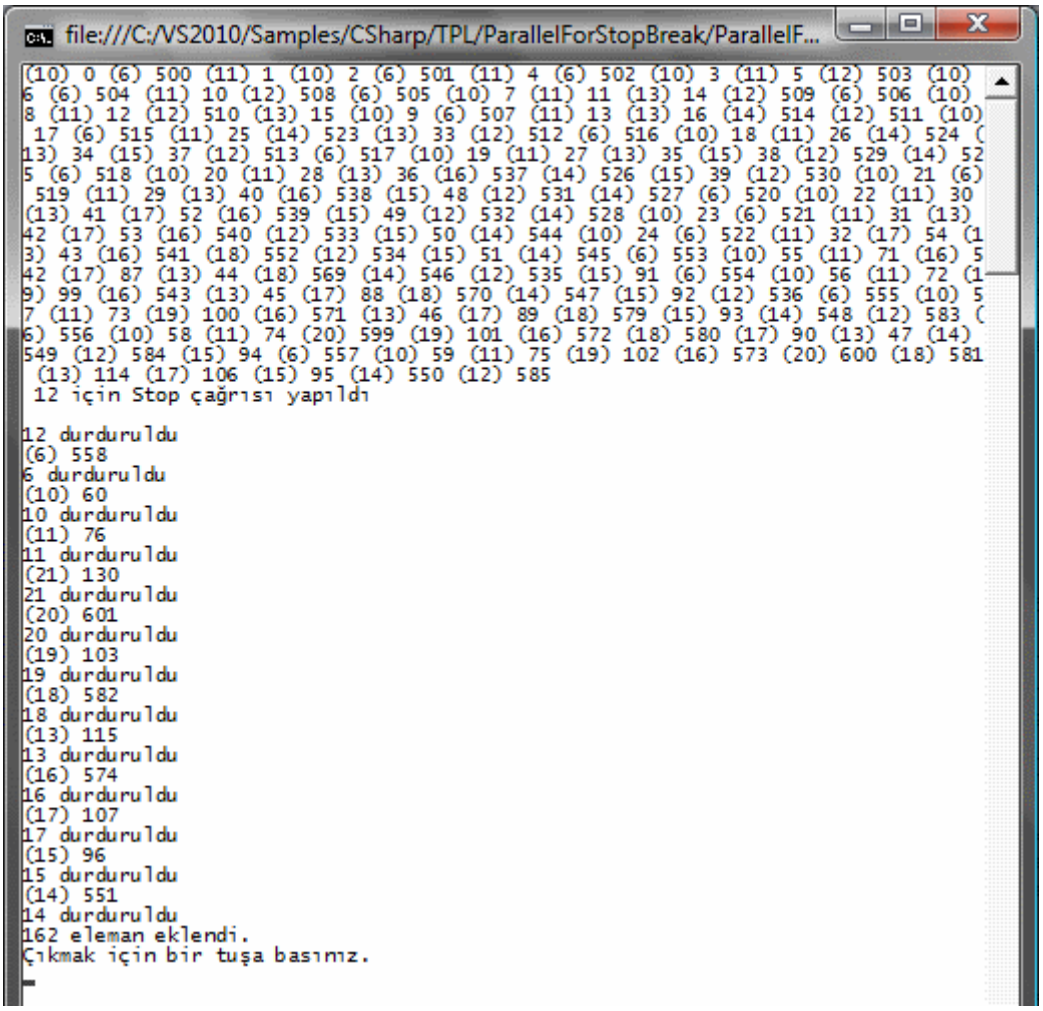
Merhaba Arkadaşlar,

Parallel.For metodu bildiğiniz gibi döngüsel işlemleri birden fazla göreve bölerek kısa sürede yapılmasına olanak sağlamaktadır. Bu yazımda, kelimeler ile ifade etmeyi bir türlü beceremediğim ancak bir örnek üzerinden sizlere aktarabileceğim **Stop** ve **Break** metodları üzerinde durmaya çalışacağım. Aslında amaç tahmin edeceğiniz üzere paralel çalışan döngü içerisinden çıkmak. Bu ardışıl çalışan bir for döngüsü göz önüne alındığında problem değil. Yada önemsenmesi gereken sorunlara yol açabilecek bir konu değil. Nitekim tek bir Thread söz konusu. Ancak Parallel.For metodu işlemleri gerçekleştirirken birden fazla Task' in başlatılmasına neden olmaktadır. Bu durumdada Stop veya Break gibi iki farklı metodun nasıl davranış göstereceğini bilmekte yarar vardır. İşte konuyu anlayabilmek için **Visual Studio 2010 Beta 1** sürümünde geliştirdiğim örnek **Console** uygulaması kodları.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections.Concurrent;
using System.Threading;

namespace ParallelForStopBreak
{
   class Program
   {
      static void Main(string[] args)
      {
         ConcurrentDictionary<int, DateTime> values = new ConcurrentDictionary<int,
DateTime>();

         // ls ParallelLoopState tipinden olup derleyici tarafından üretilmektedir.
         Random rnd = new Random();

         Parallel.For(0, 1000, (i,ls) =>
            {
               // Güncel ThreadId değerini alalım.
               string threadId=Thread.CurrentThread.ManagedThreadId.ToString();

               values.TryAdd(i, DateTime.Now);
               Thread.Sleep(500);
               Console.Write("({0}) {1} ",threadId,i.ToString());

               #region Stop Durumu

               if (rnd.Next(1, 100) == 3) // Eğer rastgele üretilen sayı 3 ise Stop metodu
çağırılır.
               {
                  ls.Stop();
                  Console.WriteLine("\t\n {0} için Stop çağrısı yapıldı", threadId);
               }
               if (ls.IsStopped) // Eğer çalışan paralel Thread durdurulmuşsa
                  Console.WriteLine("\n{0} durduruldu", threadId);

               #endregion
            }
         );

         Console.WriteLine("{0} eleman eklendi.\nçıkmak için bir tuşa
basınız.",values.Count.ToString());
```

```
            Console.ReadLine();
        }
    }
}
```

Program kodumuzda 0' dan 1000'e kadar zaman değerlerinin üretilip bir **ConcurrentDictionary<int,DateTime>** koleksiyonuna eklenmesi söz konusudur. Döngü içerisinde **Random** sınıfından yararlanılarak 3 değeri kontrol edilmektedir. Eğer 3 değerine denk gelinirse **Stop** metodu çağırılır. Program çalışması sonucu her seferinde farklı sonuçlar üretilmesi olasıdır. Bunu peşinen söyliyim. Nitekim her defasında farklı sayıda ve sırada görevler çalışmaktadır. örnek sonuçlardan birisi aşağıdaki ekran görüntüsünde olduğu gibidir.



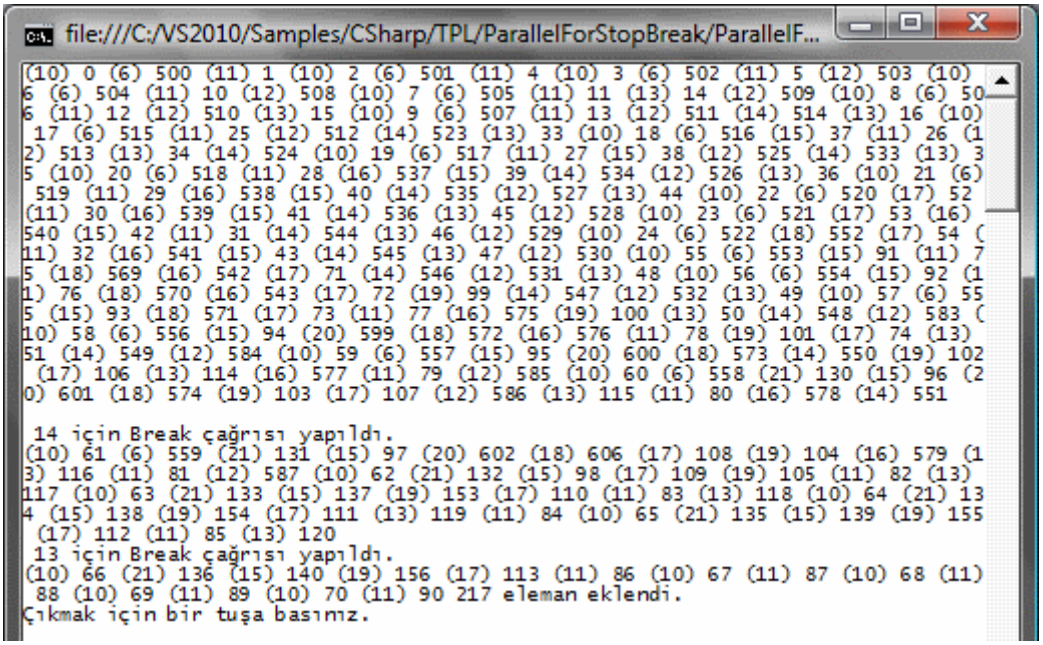
Evettt. Şimdi bu çalışma şeklini bir değerlendirelim. **Parallel.For** metodu, **12,6,10,11,21,20,19,18,13,16,17,15,14** numaralı **Th read**' leri oluşturmuştur. çalışma sırasında, 12 nolu Thread görevini yürütürkende Stop çağrısı gelmiştir. Bu durumda çalışmakta olan tüm paralel görevlere durdurulma emri gitmektedir. Ancak 12 nolu Thread sırasında Stop emri gelmesine rağmen diğer Thread' ler kısa bir sürede olsa(örneğe göre birer eleman ekleme süresi kadar) geç durmuştur.

Thread' lerin durup durmadıkları, dikkat edeceğini
üzere **ParallelLoopState**referansının **IsStopped** özelliği ile anlaşılmaktadır.

Peki ya Break metodu nasıl bir etkide bulunmaktadır. Bu amaçla Parallel.For metodu içerisine aşağıdaki kodları ekledim.

```
if (rnd.Next(1, 100) == 3) // Eğer rastgele üretilen sayı 3 ise Break metodu çağırılır.
{
 ls.Break();
 Console.WriteLine("\t\n {0} için Break çağrısı yapıldı.", threadId);
}
```

Aslında bu kez **Stop** metodu yerine sadece **Break** metodunu kullandığımızı görebiliriz. Peki ya çalışma zamanı? Her zamanki her çalışma sonrası farklı sonuçların üretildiği ortadadır. Aşağıdaki ekran görüntüsünde bu çalışmalardan birisi ele alınmaktadır.



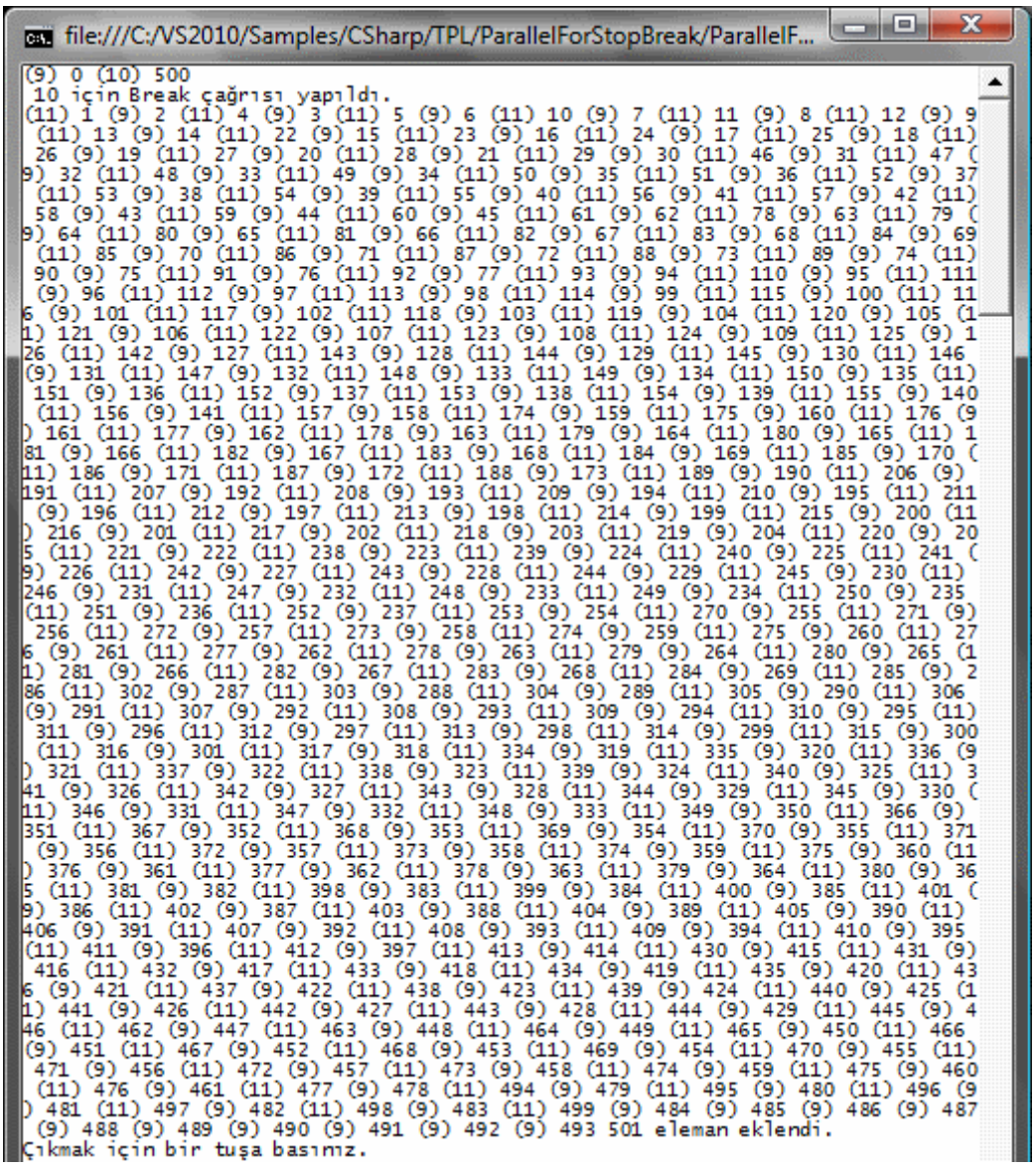
Durumu değerlendirmeye çalışalım. Herşeyden önce birden fazla **Thread'** in çalıştığı kolayca gözlemlenebilir. örnekte 10, 11, 6, 13, 12, 14, 15, 19, 21, 17 numaralı **Thread'** ler çalıştırılmaktadır. Derken çalışma zamanının bir anından, 14ncü Thread için **Break** çağrısı gelmiştir. Bunun üzerine 14 numaralı Thread durdurulmuştur. Diğer yandan, Break metodu ile karşılaşıncaya kadar başlatılan diğer Thread' ler çalışmalarına devam etmektedir. İşte Stop metodu ile aradaki önemli bir farklılık. Yinede ilerleyen kısımlarda diğer Thread' lerden bazılarının yürütülmesi esnasında **Break** çağrısı ile karşılaşılması olasıdır ki 13ncü **Thread** için bu gerçekleşmiştir. Tabiki bu çağrı sonrasında 13ncü Thread' de sonlandırılmış ama daha önceden başlatılmış diğer Thread' ler kendilerine ayrılan üst sınır değerine kadar yürümeye devam etmiştir. Nitekim ilerleyen kısımlarda diğer Thread' ler için Break komutu ile karşılaşılmamıştır.

Sanıyorumki **Stop** ve **Break** metodları arasındaki farkı biraz biraz kendini göstermeye başladı.

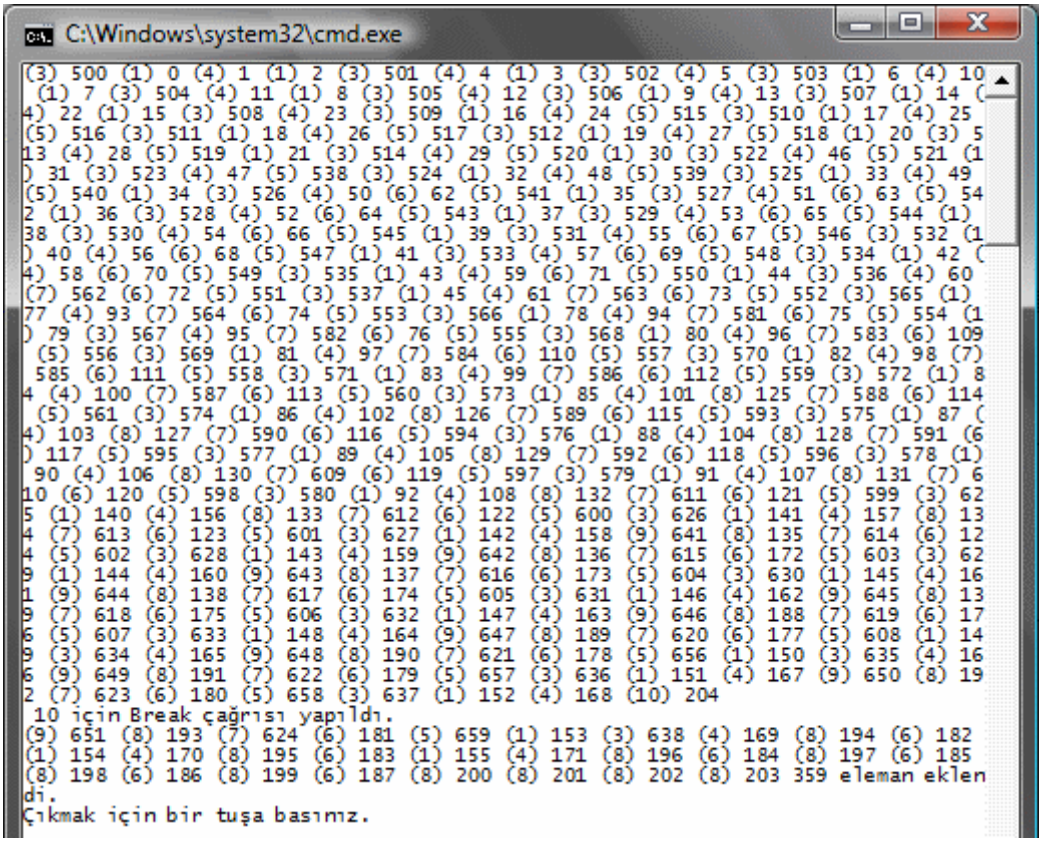
Yinede şu ana kadar yaptığım analizde havada kalan noktalar var gibi hissediyorum. Farklılığı tam olarak göremediğimi itirifat etemliyim. Bu nedenle **Break** tekniği ile ilişkili kod parçasında **if** kontrolünü aşağıdaki gibi değiştirdim ve **Thread.Sleep** süresini biraz daha kısalttım. Amaç çalışan ***Thread'*** *lerden* ***10*** *numaralı Id' ye sahip olana denk gelindiğinde****Break*** *komutu kullanmak ve diğer Thread' lere ne olacağını* anlamaktı.

if(threadId=="10")

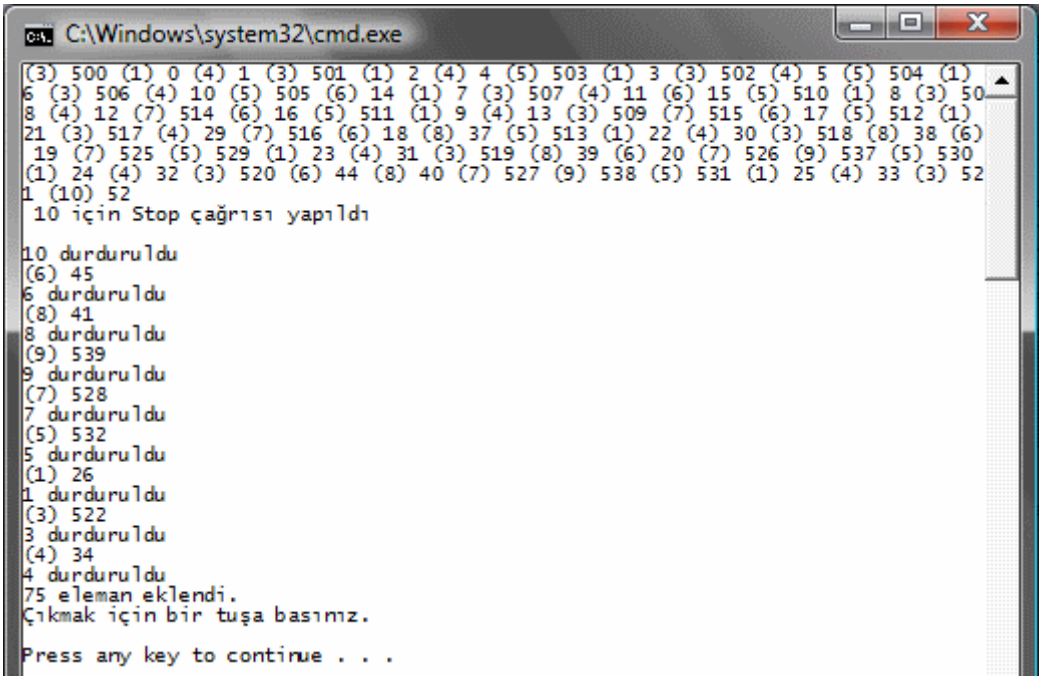
Volaaaa... Bu durumda oluşan farklı çalışma zamanı sonuçlarından birisi aşağıdaki ekran görüntüsündeki gibi oldu.



Ve diğer bir denemenin sonucu;



Bu sonuçlara ve diğerlerine baktığımda 1000 adımlık iterasyonun, **Thread**' lere farklı sayılarda bölündüğünü farkettim. Diğer yandan **10 numaralı Thread** çalışmaya başlayıp bir eleman eklendikten sonra gelen **Break** metodu çağrısı nedeniyle durdurulmuştu. Diğer Thread' ler ise çalışmalarına devam ederek kendilerine ayrılan limitler dahilinde eleman eklemeyi sürdürmüşlerdi. O zaman aynı vakada Stop metodu ne yapar diye insan ister istemez merak ediyor. Bunun üzerine Stop metodunun kullanıldığı senaryodaki if koşulunda 10 numaralı Thread' i kontrol etmeye karar verdim. Ve işte çalışma zamanı sonuçlarından birisi;



Görüldüğü gibi 10ncu **Thread** çalışmaya başlayıp 1 eleman ekledikten sonra gelen **Stop** metodu nedeniyle hem kendisi hemde diğer Thread' ler mümkün olan en kısa sürede durdurulmuştur.

Sanıyorumki artık **Stop** ve **Break** arasındaki farkı daha iyi görebiliyoruz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ParallelForStopBreak.rar (21,27 kb)

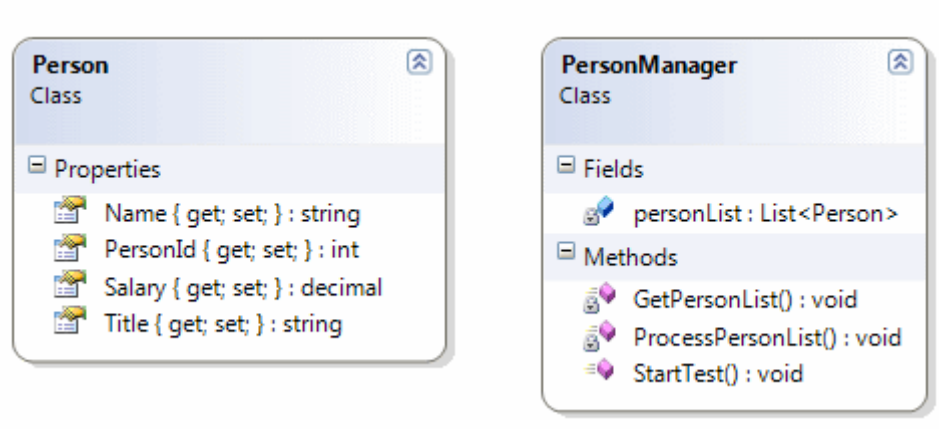
Concurrent Collections : Macera BlockingCollection ile Devam Ediyor [Beta 1] (2009-06-16T18:54:00)

parallel programming,

Merhaba Arkadaşlar,

Bir önceki blog yazımda paralel programlama kabiliyetlerinden birisi olan **Concurrent Collections(Eş Zamanlı Koleksiyonlar)** kavramını incelemeye çalışmıştım. Ne varki kendimi bunlara olan gereklilikler konusunda bir süredir ikna edebilmiş değilim. Dolayısıyla ihtiyaçları ortaya koymak adına basit bir senaryo üzerinden ilerlemeye karar verdim. Aslında eş zamanlı koleksiyonların kullanılması için en büyük gereksinim, **bir koleksiyonun elemanları üzerinde aynı anda işlemler** yapılmak istenmesi halinde ortaya çıkmaktadır. Konuyu daha net kavrayabilmek adına şöyle bir senaryoyu geliştirmeye karar verdim; Bir metin dosyasında | işaretleri ile birbirlerinden ayrılmış text tabanlı verilerin, **generic** bir **List** koleksiyonu içerisine alınması ve sonrasında ise bu koleksiyon elemanlarının içeriklerinin değiştirilmesi. Tabiki burada iki ana iş var. Metin dosyasının **ayrıştırılıp(parse)** koleksiyon içerisinde toplanması ilk adım olarak

düşünülebilir. İkinci adımda ise, bu koleksiyon üzerinde ileri yönlü bir iterasyon ile o anki nesne örneği üzerinde değişiklik yapılmaya çalışılması*(örneğin maaş bilgisinin değiştirilmesi)* durumu ele alınmalıdır. Ancak burada küçük ama önemli bir maddemiz var; **bu iki adımdaki işlemleri paralel olarak gerçekleştirebilmek** Dolayısıyla iki farklı **Thread'** in birlikte çalışarak söz konusu işlemleri yapması sağlanabilir. Bu fikirden yola çıkarak aşağıdaki bir Console uygulamasını geliştirdim. Projede yer alan ana sınıflar aşağıdaki **class diagram** çizelgesinde görüldüğü gibidir.



örnekte text tabanlı içeriği tutan Personel.txt dosyasının içeriğini ise aşağıdaki gibi tamamen atmason verilerden oluşturmuş bulunmaktayım.

ClassDiagram1.cd | Start Page | Object Browser | Personel.txt | Program.cs

```
10001|Burak Selim Şenyurt|Uygulama Geliştirme Danışmanı|900
10002|Kobi Bıraynt|Profesyonel Basketçi|9000
10003|Hido Turkulu|Profesyonel Basketçi|4500
10004|Mario Anders|Uygulama Geliştirici|3400
10005|Adrian Lam|Proje Yöneticisi|1250
10006|Kerim Mayra|Teknik Proje Lideri|4500
10007|Alonzo Rivas|Genel Müdür|10000
10008|George Ali|Finansman Müdürü|8000
10009|Eleni Grifin|İnsan Kaynakları|4500
10010|Maria Kurosova|Uluslararası İlişkiler|12000
```

Program kodları ise;

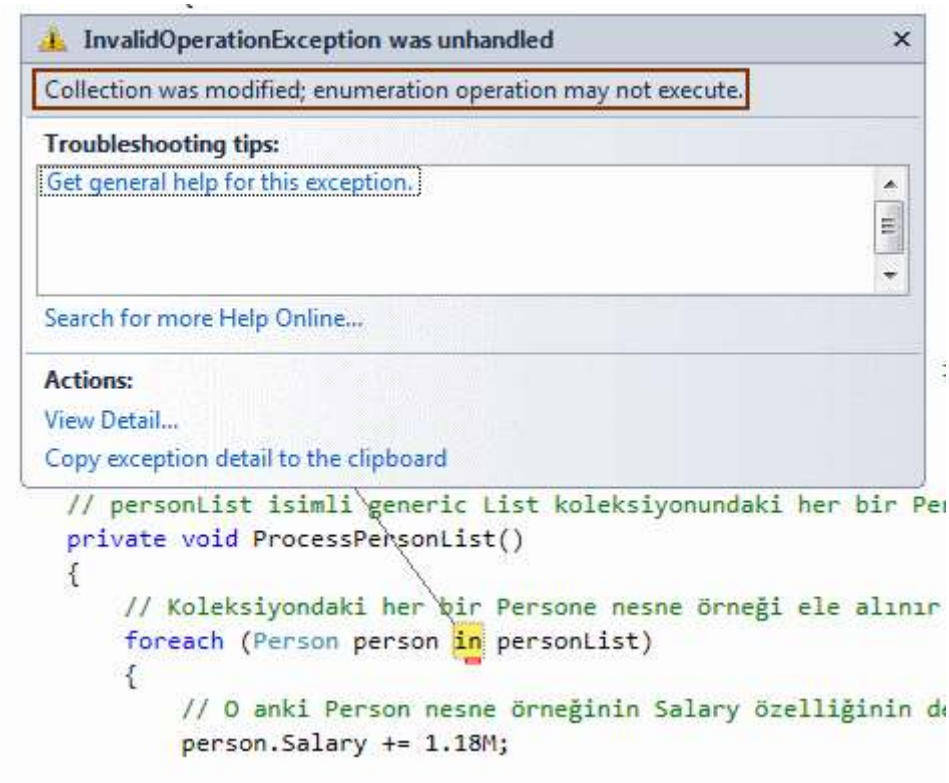
```
using System;
using System.Collections.Generic;
using System.IO;
using System.Threading;

namespace ConcurrentCollections2
{
   class Program
   {
```

```
    static void Main(string[] args)
    {
        try
        {
            PersonManager manager = new PersonManager();
            manager.StartTest();
        }
        catch (Exception excp)
        {
            Console.WriteLine(excp.Message);
        }

        Console.ReadLine();
    }
}

// Metin dosyasındaki bilgilerin nesne karşılıkları için tasarlanmış Person sınıfı
class Person
{
    public int PersonId { get; set; }
    public string Name { get; set; }
    public string Title { get; set; }
    public decimal Salary { get; set; }
}

// Test metodunu içeren Test sınıfımız
class PersonManager
{
    // Person bilgilerinin tutulacağı generic List koleksiyonu
    List<Person> personList = new List<Person>();

    public void StartTest()
    {
        // GetPersonList metodu için bir Thread tanımlanır
        Thread trd1 = new Thread(new ThreadStart(GetPersonList));
        // ProcessPersonList metodu için bir Thread tanımlanır
        Thread trd2 = new Thread(new ThreadStart(ProcessPersonList));

        // Thread' ler başlatılır
        trd1.Start();
        trd2.Start();
    }

    // Metin dosyasından okuma işlemini yaparak personList isimli generic List
koleksiyonuna Person nesne örneklerinin eklenmesi işlemini üstlenir
```

```
        private void GetPersonList()
        {
            // Personel.txt dosyasındaki tüm satırlar string[] dizisine alınır
            string[] persons = File.ReadAllLines(System.Environment.CurrentDirectory +
"\\Personel.txt");

            // Her bir satır ele alınır
            foreach (string person in persons)
            {
                // Satır | işaretine göre ayrıştırılır
                string[] values = person.Split('|');

                // Ayrıştırma sonucu elde edilen değerlere göre Person nesne örneği oluşturulur
                Person prs = new Person
                {
                    PersonId = Convert.ToInt32(values[0]),
                    Name = values[1],
                    Title = values[2],
                    Salary = Convert.ToDecimal(values[3])
                };
                // Persone nesne örneği koleksiyona eklenir
                personList.Add(prs);
                // Console penceresinden bilgilendirme yapılır
                Console.WriteLine("{0} listeye eklendi", prs.Name);

                Thread.Sleep(250); // işleyişi kolay takip edebilmek için küçük bir zaman
aldatmacası
            }
        }

        // personList isimli generic List koleksiyonundaki her bir Person nesne örneğinin
Salary bilgisini değiştirir
        private void ProcessPersonList()
        {
            // Koleksiyondaki her bir Persone nesne örneği ele alınır
            foreach (Person person in personList)
            {
                // O anki Person nesne örneğinin Salary özelliğinin değeri değiştirilir
                person.Salary += 1.18M;

                // Console ekranında bilgilendirme yapılır
                Console.WriteLine("\t {0} için maaş {1} olarak değiştirildi", person.Name,
person.Salary);
                Thread.Sleep(250); // işleyişi kolay takip edebilmek için küçük bir zaman
aldatmacası
```

```
        }
    }
  }
}
```

PersonManager sınıfı içerisinde yer alan **StartTest** metodu kendi içerisinde iki farklı **Thread** oluşturmakta ve çalıştırmaktadır. Bu **Thread**' lerden birisi **GetPersonList** fonksiyonunu kullanarak koleksiyona veri ekleme işlemini üstlenmektedir. İkinci **Thread** tarafından çağırılan **ProcessPersonList** metod ise, maaş bilgilerini düzenlemektedir. Kritik olan nokta her iki **Thread**' in aynı koleksiyon nesne örneği üzerindeki elemanları kullanmak istemesidir. Programı çalıştırdığımda aşağıdaki sonuç ile karşılaştım;



Görüldüğü gibi koleksiyon zaten farklı bir Thread içerisinde ele alındığından, düzenleme işlemi yapılmasına izin verilmemektedir. İşte eş zamanlı koleksiyonları ele almak için geçerli bir neden. Peki ama hangi eş zamanlı koleksiyon 😕 Bu noktada bir önceki blog yazımın sonunda verdiğim sözü hatırlıyorum. **BlockingCollection<T>** koleksiyonu. Bunun üzerine kodu aşağıdaki şekilde değiştirdim.

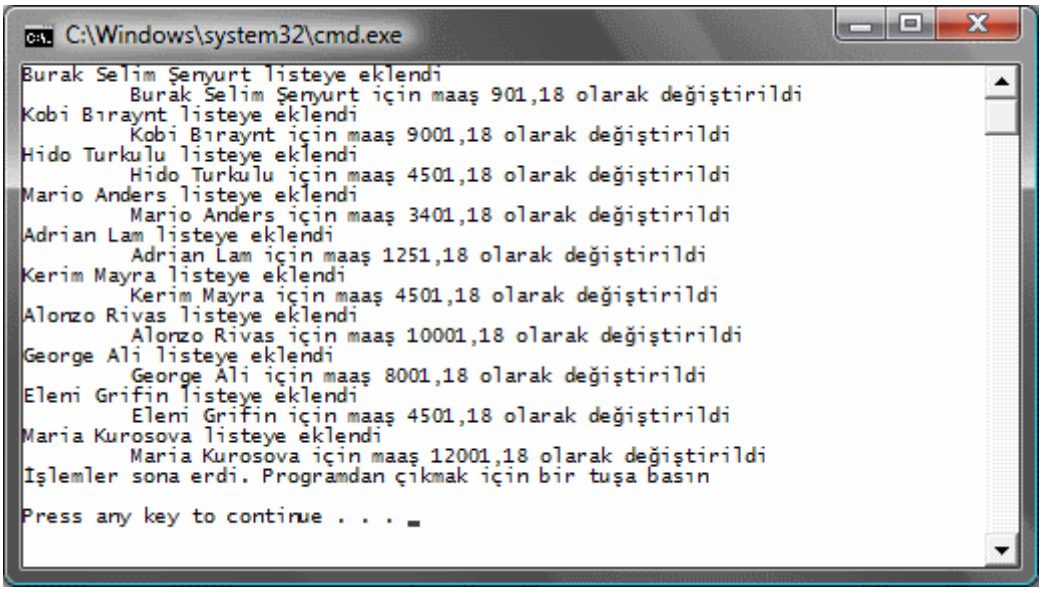
```
using System;
using System.Collections.Concurrent;
using System.IO;
using System.Threading;
using System.Threading.Tasks;
```

```
namespace ConcurrentCollections2
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            PersonManager manager = new PersonManager();
            manager.StartTestConcurrent();
         }
         catch (Exception excp)
         {
            Console.WriteLine(excp.Message);
         }
      }
   }

   // Metin dosyasındaki bilgilerin nesne karşılıkları için tasarlanmış Person sınıfı
   class Person
   {
      public int PersonId { get; set; }
      public string Name { get; set; }
      public string Title { get; set; }
      public decimal Salary { get; set; }
   }

   // Test metodunu içeren Test sınıfımız
   class PersonManager
   {
      // Person bilgilerinin tutulacağı generic List koleksiyonu
      // List<Person> personList = new List<Person>();
      BlockingCollection<Person> personList = new BlockingCollection<Person>();

      public void StartTestConcurrent()
      {
         // Task' leri başlatalım
         Task[] tasks ={ Task.Factory.StartNew(() => { GetPersonList(); }),
                         Task.Factory.StartNew(() => { ProcessPersonList(); })
                       };

         // Tüm Task' ler tamamlanıncaya kadar bekle
         Task.WaitAll(tasks);

         Console.WriteLine("İşlemler sona erdi. Programdan çıkmak için bir tuşa basın");
```

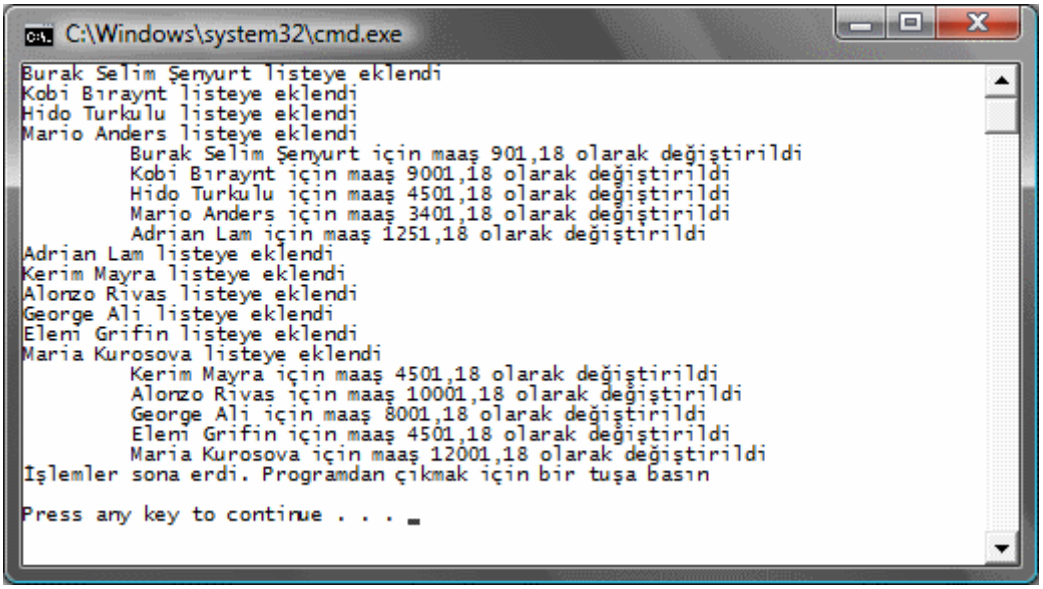
```
            Console.ReadLine();
        }

        // Metin dosyasından okuma işlemini yaparak personList isimli generic List
koleksiyonuna Person nesne örneklerinin eklenmesi işlemini üstlenir
        private void GetPersonList()
        {
            // Personel.txt dosyasındaki tüm satırlar string[] dizisine alınır
            string[] persons = File.ReadAllLines(System.Environment.CurrentDirectory +
"\\Personel.txt");

            // Her bir satır ele alınır
            foreach (string person in persons)
            {
                // Satır | işaretine göre ayrıştırılır
                string[] values = person.Split('|');

                // Ayrıştırma sonucu elde edilen değerlere göre Person nesne örneği oluşturulur
                Person prs = new Person
                {
                    PersonId = Convert.ToInt32(values[0]),
                    Name = values[1],
                    Title = values[2],
                    Salary = Convert.ToDecimal(values[3])
                };
                // Persone nesne örneği koleksiyona eklenir
                personList.Add(prs);
                // Console penceresinden bilgilendirme yapılır
                Console.WriteLine("{0} listeye eklendi", prs.Name);

                Thread.Sleep(250); // işleyişi kolay takip edebilmek için küçük bir zaman
aldatmacası
            }
            // koleksiyona daha fazla eleman eklenmeyeceğini belirt.
            // Bu metodu kullanmadan denediğinizde programın asılı kaldığını ve
kapanmadığını göreceksiniz.
            personList.CompleteAdding();
        }

        // personList isimli generic List koleksiyonundaki her bir Person nesne örneğinin
Salary bilgisini değiştirir
        private void ProcessPersonList()
        {
            // Koleksiyondaki her bir Persone nesne örneği ele alınır
            foreach (Person person in personList.GetConsumingEnumerable())
```

```
        {
            // O anki Person nesne örneğinin Salary özelliğinin değeri değiştirilir
            person.Salary += 1.18M;

            // Console ekranında bilgilendirme yapılır
            Console.WriteLine("\t {0} için maaş {1} olarak değiştirildi", person.Name,
person.Salary);
            Thread.Sleep(250); // işleyişi kolay takip edebilmek için küçük bir zaman
aldatmacası
        }
    }
  }
}
```

Bu kez **BlockingCollection<Person>** tipinden bir nesne örneğini kullanmaktayız. Bu koleksiyon kendi içerisindeki elemanlar üzerinde eş zamanlı işlemler yapılabilmesine imkan tanımaktadır. Ayrıca istenirse bir boyut verilerek, eş zamanlı çalışma sırasında maksimum eleman ekleme tavanınıda belirtebiliriz. Kodda görüldüğü gibi **Task** sınıfından yararlanarak kodu tamamen **.Net 4.0** havasına büründürmüş bulunuyoruz. **StartTestConcurrent** metodu içerisinde dikkat edilmesi gereken noktalardan biriside, **Task** sınıfının **static WaitAll** fonksiyonu ile, çalışan tüm Task' lerin tamamlanmasının beklenmesidir. Ayrıca, **GetPersonList** metodu içerisinde, text tabanlı dosyadaki tüm elemanların aktarılma işlemi tamamlandıktan sonra**CompleteAdding** fonksiyonu kullanılarak, artık daha fazla eleman eklenmeyeceği, bu nedenle aynı koleksiyon üzerinde bekleyen başka görevler var ise yollarına devam edebilecekleri belirtilmektedir. Eğer **CompleteAdding** metodunu kullanmassak, programın kapanmadığı gözlemlenecektir. Uygulamayı çalıştırdığımda aşağıdaki sonuçları aldığımı gördüm;



Harika değil mi? Artık hata mesajı yok. üstelik koleksiyon üzerinde aynı anda iki farklı gövde işlem yapabilmekte. İstenirse görev sayısı dahada arttırılabilir elbetteki. örneğin çalışmasına göre bir GetPersonList bir ProcessPersonList metodundan sonuçlar alınması Thread.Sleep sürelerinin aynı olmasından kaynaklanmaktadır. Elbetteki gerçek hayat senaryosunda bu süre aynı olmayacaktır. Bende bu düşünce ile Thread.Sleep metodlarını kaldırdığıma aşağıdaki sonuçları aldım.



Dikkat edileceği üzere dosyadan koleksiyona ekleme işlemleri gerçekleşmeden, maaş bilgilerinin düzenlenmesine izin verilmemektedir. Bir başka deyişle koleksiyon içerisinde elemanlar olduğu sürece, **ProcessPersonList** metodu içerisindeki **foreach** döngüsü çalışabilmektedir. Aksi durumlarda, koleksiyon üzerindeki iterasyon elemanlar ekleninceye kadar duraksatılmaktadır*(Tabi, maaş değişiklikerini yapan foreach döngüsü nerede duracağını nasıl bilecektir sorusunun cevabı =* ***CompleteAdding*** *metodudur).* Buda koleksiyona neden **BlockingCollection**dendiğini açıklamaktadır.

BlockingCollection<T> tipinin farklı özellikleride bulunmakta. Bunlarıda yeri geldikçe incelemeye gayret edeceğim. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ConcurrentCollections2.rar (27,67 kb)

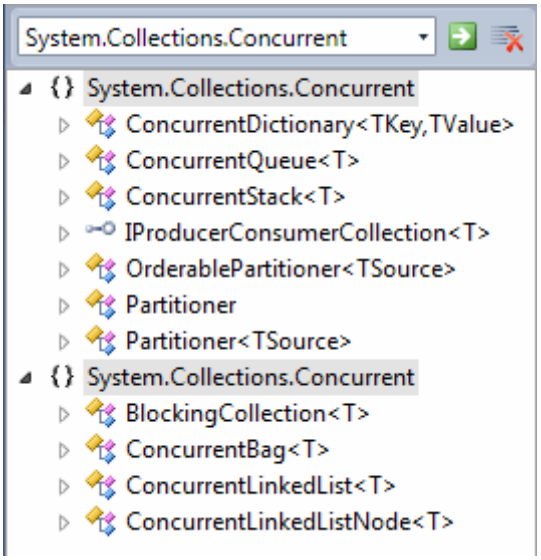
Concurrent Collections (Eş Zamanlı Koleksiyonlar) [Beta 1] (2009-06-13T01:20:00)

parallel programming,

 Merhaba Arkadaşlar,

.Net Framework 4.0 ve içerdiği **paralel genişletmeler(Parallel Extensions)** ile birlikte gelmekte olan yenilikler arasında, **eş zamanlı(Concurrent)** çalışabilen ve **Thread Safe** olan koleksiyonlarda bulunmaktadır. Bu koleksiyonlar aslında **veri yapıları(Data Structures)** ile birlikte gelen yeni tipler arasında yer almaktadır. Geçtiğimiz günlerde çok şanslı bir insan olarak hafta sonumu bir tatil beldesinde geçirirken,

bu kez gecenin derin sessizliğinde araştırmaya başladığım konulardan biriside işte bu yeni koleksiyonlar oldu. Bu koleksiyon tipleri elbetteki relase sürümünde değişikliğe uğrayabilir. Söz konusu koleksiyon tipleri esasında **System.Collections.Concurrent** isim alanı altındadır. Ancak bu isim alanı **System** ve **Mscorlib** olmak üzere iki **assembly** içerisine aşağıdaki şekilde görüldüğü gibi dağılmıştır.

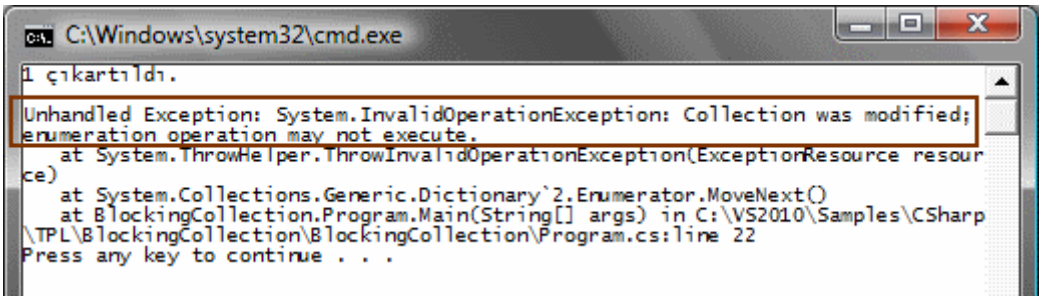


Visual Studio 2010 Beta 1 üzerindeki **object browser** yardımıyla söz konusu tiplere baktığımda bana tanıdık gelebilecek olanlar sadece **ConcurrentDictionary**, **ConcurrentQueue** ve**ConcurrentStack** koleksiyonlarıydı . Nitekim bu tipler daha önceki .Net sürümlerinden bildiğimiz **Dictionary, Queue** ve **Stack** koleksiyonlarının eş zamanlı çalışabilen versiyonlarıydı. Ancak kafamda iki önemli soru bulunmaktaydı. Bir; *diğer koleksiyon tipleri nasıl ve hangi amaçlar ile kullanılmaktaydı* ve iki; *koleksiyonların eş zamanlı olmasının ne anlamı vardı*

Paralel genişletme ile gelen koleksiyonların ataları çoğunlukla **Thread Safe** yapıda değildir. Bu nedenle geliştiricinin **Thread Safe** yapısını sağlaması gerektiği durumlarda kolları sıvaması ve kilitleme mekanizmalarını bilinçli olarak kullanması gerekmektedir. Bir başka deyişle, koleksiyon içerisine dahil edilen elemanlar üzerinde bir iterasyon yapıldığında, başka**Thread'** ler üzerinden aynı koleksiyonun elemanlarına ulaşmak güvenli değildir. Bu nedenle örneğin bir koleksiyonun elemanları dolaşılırken belirli kriterlere göre aynı koleksiyondan eleman çıkartılmasıda mümkün değildir.*(Ki bu durumda geliştiricilerin **multi-thread** yapıları içerisinde ele alınan koleksiyonlar için senkronizasyon tekniklerini kullanarak sorunu çözmesi gerekmektedir)* Hatta aşağıdaki kod parçasında olduğu gibi bir koleksiyonun üyelerinin dolaşılması sırasında,

```
static void Main(string[] args)
{
 Dictionary<int, string> numbers = new Dictionary<int, string>
 {
  {1,"Bir"},
  {2,"İki"},
  {3,"üç"},
  {4,"Dört"},
  {5,"Beş"},
  {6,"Altı"}
 };

 foreach (KeyValuePair<int,string> number in numbers)
 {
  numbers.Remove(number.Key);
  Console.WriteLine("{0} çıkartıldı.",number.Key);
 }
}
```

eleman çıkartma işlemi gerçekleştirildiğinde çalışma zamanında aşağıdaki ekran görüntüsünde yer alan **InvalidOperationException** istisnasını almamız kaçınılmazdır.

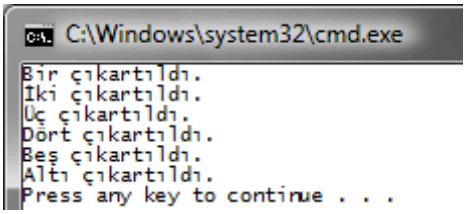


Görüldüğü gibi ilk eleman çıkartıldıktan sonra koleksiyonun boyutu değiştiğinden **InvalidOperationException** istisnasının fırlatılması söz konusu olmuştur. Oysaki**Dictionary<T,K>** koleksiyonu yerine **Concurrent** versiyonu kullanılsaydı Thread

Safe kuralları çerçevesinde herhangibir sorun ile karşılaşılmazdı. Aşağıdaki kod parçasında bu duruma ait bir kod parçası görülmektedir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Collections.Concurrent;

namespace BlockingCollection
{
   class Program
   {
      static void Main(string[] args)
      {
         #region Concurrent versiyonu

         ConcurrentDictionary<int, string> numbers = new ConcurrentDictionary<int, string>();
         numbers.TryAdd(1, "Bir");
         numbers.TryAdd(2, "İki");
         numbers.TryAdd(3, "üç");
         numbers.TryAdd(4, "Dört");
         numbers.TryAdd(5, "Beş");
         numbers.TryAdd(6, "Altı");

         foreach (KeyValuePair<int,string> number in numbers)
         {
            string value;
            bool result=numbers.TryRemove(number.Key, out value);
            if(result)
               Console.WriteLine("{0} çıkartıldı.",value);
         }

         #endregion
      }
   }
}
```

ve sonuç;



Görüldüğü gibi koleksiyon elemanları **foreach** döngüsü ile gezilirken teker teker çıkartılma işlemi yapılabilmiştir. Buna göre öyle vakalar olmalıdır ki, koleksiyonları ele alan paralel süreçlerin aynı örnek üzerindeki elemanlarda Thread Safe kuralları çerçevesinde ekleme, silmve ve güncelleme gibi işlemler yapılabilmelidir. Dolayısıyla paralel genişletmelere ait veri yapılarında yer alan **Concurrent** koleksiyonların temel kullanım amacı belkide bu şekilde ifade edilebilir. Ben tabiki hemen diğer koleksiyonları ve kullanım amaçlarını merak etmeye başladım ve incelemeye karar verdim. Ne varki içimden bir dürtü, *"bak Burakcığım, **Thread Safe** kolayca bertaraf edilmiş, eş zamanlı olarak aynı koleksiyon üzerinde birden fazla sürecin işlem yapabilmesi sağlanmış. Peki ya performanstan ne haber?"* 😀 Bu nedenle **.Net 4.0** öncesi **Dictionary** koleksiyonu ile **ConcurrentDictionary** koleksiyonu arasındaki performans farklılıklarını analiz etmeye karar verdim. Aslında ilk tahminlerimin doğru çıktığını ifade edebilirim şimdiden 😬

Thread Safe + aynı anda ilerleme,ekleme, çıkartma, düzenleme yapabilme yeteneği = pahalı maliyet

İşte test programı kodları;

```
using System;
using System.Collections.Concurrent;
using System.Collections.Generic;
using System.Diagnostics;
using System.Threading;

namespace BlockingCollection
{
   class Program
   {
      static void Main(string[] args)
      {
         //Dictionary ve ConcurrentDictionary koleksiyonları için arka arkaya 10 test yapılır
         for (int i = 0; i < 10; i++)
         {
            DictionaryTest();
            ConcurrentDictionaryTest();
            ParallelConcurrentTest();
         }
      }
```

```
static int length = 9000000;

// Dictionary<int,int> koleksiyonuna eleman ekleme ve okuma işlemlerini ele alır.
static void DictionaryTest()
{
   Stopwatch watch = Stopwatch.StartNew();

   Dictionary<int, int> collection = new Dictionary<int, int>();

   // Eleman ekleme işlemi
   Random rnd = new Random();
   for (int i = 0; i < length; i++)
   {
      collection.Add(i, rnd.Next(1, 1000000));
   }
   watch.Stop();
   Console.WriteLine("{0}",watch.Elapsed.TotalSeconds.ToString());

   // Zamanlayıcı sıfırla ve yeniden başlat.
   watch.Reset();
   watch.Start();

   // Eleman okuma işlemi
   foreach (KeyValuePair<int,int> item in collection)
   {
      int value = item.Value;
   }
   watch.Stop();
   Console.WriteLine("{0}", watch.Elapsed.TotalSeconds.ToString());
}

// ConcurrentDictionary<int,int> koleksiyonuna eleman ekleme ve okuma işlemlerini
ele alır
static void ConcurrentDictionaryTest()
{
   Stopwatch watch = Stopwatch.StartNew();

   ConcurrentDictionary<int, int> collection = new ConcurrentDictionary<int,
int>();

   // Eleman ekleme işlemleri
   Random rnd = new Random();
   for (int i = 0; i < length; i++)
   {
      collection.TryAdd(i, rnd.Next(1, 1000000));
```

```
        }
        watch.Stop();
        Console.WriteLine("\t{0}", watch.Elapsed.TotalSeconds.ToString());

        // Zamanlayıcıyı sıfırla ve yeniden başlat
        watch.Reset();
        watch.Start();
        // Eleman okuma işlemleri
        foreach (KeyValuePair<int, int> item in collection)
        {
           int value = item.Value;
        }
        watch.Stop();
        Console.WriteLine("\t{0}", watch.Elapsed.TotalSeconds.ToString());
    }

    // Parallel.For ve Parallel.ForEach kullanıldığında Concurrent koleksiyonun eleman ekleme ve okuma işlemlerini test eder.
    static void ParallelConcurrentTest()
    {
        Stopwatch watch = Stopwatch.StartNew();

        ConcurrentDictionary<int, int> collection = new ConcurrentDictionary<int, int>();

        // Eleman ekleme işlemleri
        Random rnd = new Random();

        // Paralel çalışan For döngüsü
        Parallel.For(0, length, i =>
        {
           collection.TryAdd(i, rnd.Next(1, 1000000));
        }
        );
        watch.Stop();
        Console.WriteLine("\t{0}", watch.Elapsed.TotalSeconds.ToString());

        // Zamanlayıcıyı sıfırla ve yeniden başlat
        watch.Reset();
        watch.Start();
        // Eleman okuma işlemleri
        // Paralel çalışan ForEach döngüsü
        Parallel.ForEach<KeyValuePair<int, int>>(collection, item =>
        {
           int value = item.Value;
```

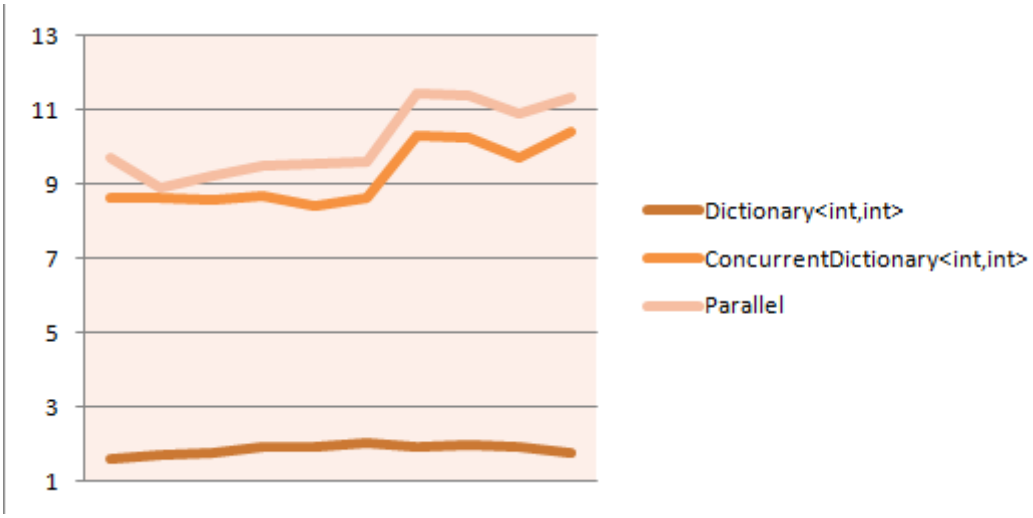
```
            }
            );
            watch.Stop();
            Console.WriteLine("\t{0}", watch.Elapsed.TotalSeconds.ToString());
        }
    }
}
```

Uygulamamızda **Dictionary<int,int>** ve **ConcurrentDictionary<int,int>** tipinden iki koleksiyon 3 farklı test metodu yardımıyla ele alınmaktadır. Testler sırasında her iki koleksiyonada rastgele sayılardan oluşan **9000000** tam sayı ilave edilmektedir. Sonrasında ise doldurulan koleksiyonlar ileri yönlü bir iterasyon ile okunmaktadır. Program kodunun temel amacı, eleman ekleme ile okuma işlemlerinde, **Dictionary** ve **ConcurrentDictionary** koleksiyonlarının söz konusu işlemleri ortalama olarak ne kadar sürelerde tamamladıklarının testini yapmaktır. **ParallelConcurentTest** isimli metod dikkat edileceği üzere **TPL(Task Parallel Library)** kütüphanesinde yer alan **Parallel.For** ve **Parallel.ForEach** metodlarını kullanarak **ConcurrentDictionary** koleksiyonunu ele almaktadır. Ben bu programı **intel** tabanlı **çift çekirdek işlemcili**, **4 Gb Ram** belleğe sahip ve **Vista Enterprise** işletim sistemi üzerinde koşturduğumda anlık koşullara göre aşağıdaki ekleme sürelerini tespit ettim.

Eleman Ekleme Süreleri

Deneme	Dictionary<int,int>	ConcurrentDictionary<int,int>	Parallel
1	1,5965157	8,6457496	9,7127165
2	1,7207327	8,6280703	8,8890291
3	1,7718992	8,6033512	9,246576
4	1,9256235	8,7227608	9,4900385
5	1,9287144	8,4039116	9,539486
6	2,0223963	8,6328307	9,6052221
7	1,9426832	10,311767	11,4428462
8	2,0062376	10,267085	11,3882937
9	1,9487786	9,7330822	10,8873102
10	1,8028344	10,415104	11,3630567

Grafik olarak baktığımızda,

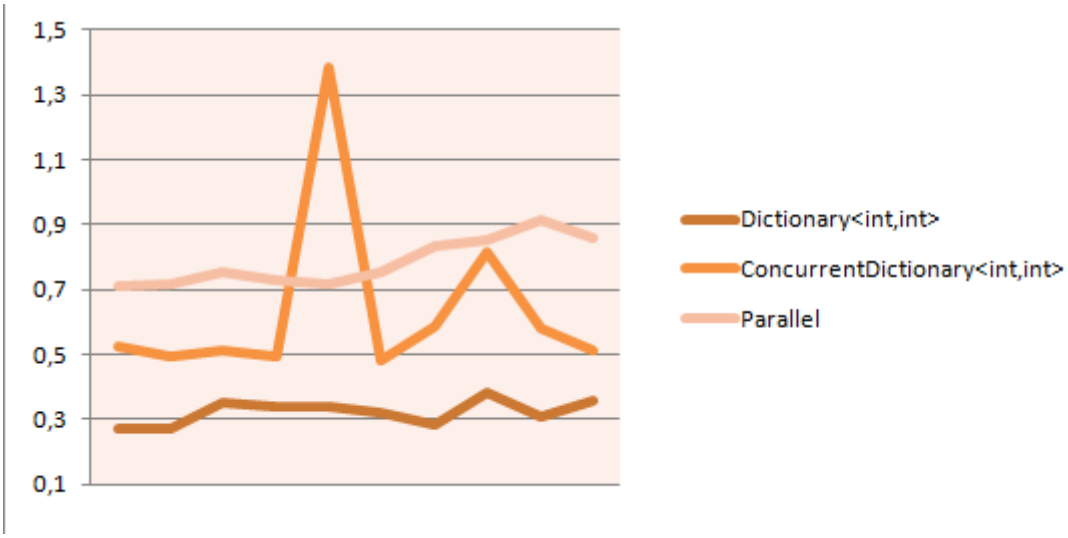


ConcurrentDictionary koleksiyonu için eleman ekleme sürelerinin gerçekten çok kötü olduğu gözlemlenebilir. Hatta durumu kurtarmak
adına **Parallel.For** ve **Parallel.ForEach** metodlarının kullanıldığı durumdaki zaman değerleride son derece kötüdür. Diğer yandan, oluşturulan bu koleksiyonların tüm elemanlarını ileri yönlü bir iterasyon ile dolaştığımızda aşağıdaki zaman değerlerini elde ettiğimi gördüm.

Eleman Okuma Süreleri

Deneme	Dictionary<int,int>	ConcurrentDictionary<int,int>	Parallel
1	0,2707316	0,5216791	0,7073974
2	0,2715216	0,4951542	0,7149783
3	0,3506021	0,5100271	0,7525682
4	0,3380284	0,4933783	0,7305076
5	0,338477	1,3850732	0,7164944
6	0,322663	0,4776662	0,7548498
7	0,2821501	0,5871846	0,8353176
8	0,3824846	0,8149798	0,8492322
9	0,305484	0,577625	0,9152573
10	0,3560983	0,5122665	0,8599752

Duruma grafiksel olarak baktığımızda,



ConcurrentDictionary ve **Dictionary** arasındaki sürelerin birbirlerine yaklaştıklarını görebiliriz. Ancak **ConcurrentDictionary** koleksiyonu için okuma sürelerinin*(işlemler paralel halde ele alınsalara dahi)* yinede **Dictonary** koleksiyonuna göre belirgin ölçüde yavaş olduğu açıktır.

Elbetteki bu testler, henüz relase edilmemiş olan **beta 1** sürümü üzerinden yapılmaktadır. Dolayısıyla zaman içerisinde iyileştirmelerin olması muhtemeldir. Hatta söz konusu uygulamanın çekirdeği yeniden yazılmış olan **Windows 7** işletim sisteminde test edilmeside mutlaka gereklidir. Ancak, **Concurrent** koleksiyonların kullanılma sebeplerinin başında hız veya performans olmadığı gayet net bir biçimde ortadadır. Tabiki bunun dışında kalan senaryolardada gerçekten performans kaybını göze almamızı gerektirecek durumlar olmalıdır. Şu anda sesli düşünüyorum; *"Bir uygulama içerisindeki birden fazla tipin ortaklaşa kullandığı bir koleksiyon üzerinde, eş zamanlı olarak ekleme, silme ve düzenleme işlemeleri yapılabiliyor olsun..."* Bilmiyorum siz ne düşünüyorsunuz. Aslında fikirlerinizi yorum olarak paylaşabilirsiniz.

Concurrent koleksiyonlar ile ilişkili araştırmalarım devam etmekte. örneğin şu sıralar göz kestirdiklerimden birisi olan ve aslında bu yazıda incelemek isteyipte, performans ve hız kriterine takıldığım için araştıramadığım **BlockingCollection**. Bunuda bir sonraki yazımda ele almaya gayret ediyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ConcurrentCollectionTest.rar (23,87 kb)

for mu, foreach mi? Yoksa Parallel.For mu, Parallel.ForEach mi? [Beta 1] (2009-06-10T01:51:00)

task parallel library,

Merhaba Arkadaşlar,

Gecenin bu saatinde uyuyamayıp blog' uma bir şeyler yazmak isteyişimin sebebi, bu gün bir okurumdan gelen şu sorudur; **"Madem Parallel.For veya Parallel.ForEach ile herşey daha hızlı oluyor, niye normal for ve foreach döngülerini bu formasyona sokmuyorlarda ek bir şeyler ilave ediyorlar".** Dolayısıyla klavyemi elime aldım ve hemen bir test programı yazmaya koyuldum. Bu kez amaç vaat edilenin tersini göstermekti. Yani performansa ve hıza ulaşmaya çalışmayacak, tam aksi yöne gitmeye gayret edecektim. Aslında bu işlemler için gayet profesyonel test araçları mevcuttur. Ancak bir araca gerek duymadanda analizimizi yapabiliriz. İşe **Visual Studio 2010 Beta 1** üzerinde, basit bir **Console** örneği geliştirerek başladım. İşte kodlarımız;

```
using System;
using System.Diagnostics;
using System.Threading;

namespace ForForEachPerformance
{
   class Program
   {
      static void Main(string[] args)
      {
         int arraySize = 1000;
         double[] array1 = new double[arraySize];

         Random rnd=new Random();
         Stopwatch watch1 = Stopwatch.StartNew();

         for (int i = 0; i < array1.Length; i++)
         {
            array1[i] =
(rnd.NextDouble()/Math.Cos(rnd.NextDouble()))*Math.Sqrt(rnd.NextDouble());
         }

         Console.WriteLine("For döngüsü eleman ekleme süresi {0}
milisaniyedir.",watch1.Elapsed.TotalMilliseconds.ToString());

         double[] array2 = new double[arraySize];

         Stopwatch watch2 = Stopwatch.StartNew();
         Parallel.For(0, array2.Length, i =>
            {
               array1[i] = (rnd.NextDouble() / Math.Cos(rnd.NextDouble())) *
Math.Sqrt(rnd.NextDouble());
            }
         );
```

```
        Console.WriteLine("Parallel.For döngüsü eleman ekleme süresi {0} milisaniyedir.",
watch2.Elapsed.TotalMilliseconds.ToString());

        Stopwatch watch3 = Stopwatch.StartNew();

        for (int i = 0; i < array1.Length; i++)
        {
           double d = array1[i];
        }

        Console.WriteLine("For döngüsü eleman okuma süresi {0} milisaniyedir.",
watch3.Elapsed.TotalMilliseconds.ToString());

        Stopwatch watch4 = Stopwatch.StartNew();

        Parallel.For(0, array1.Length, i =>
           {
              double d = array1[i];
           }
        );

        Console.WriteLine("Parallel.For döngüsü eleman okuma süresi {0} milisaniyedir.",
watch4.Elapsed.TotalMilliseconds.ToString());

        Stopwatch watch5 = Stopwatch.StartNew();

        Parallel.ForEach(array1, i =>
           {
              double d = i;
           }
        );

        Console.WriteLine("Parallel.ForEach döngüsü eleman okuma süresi {0}
milisaniyedir.", watch5.Elapsed.TotalMilliseconds.ToString());

        Stopwatch watch6 = Stopwatch.StartNew();

        foreach (double i in array1)
        {
           double d = i;
        }

        Console.WriteLine("ForEach döngüsü eleman okuma süresi {0} milisaniyedir.",
watch6.Elapsed.TotalMilliseconds.ToString());
        Console.ReadLine();
    }
```

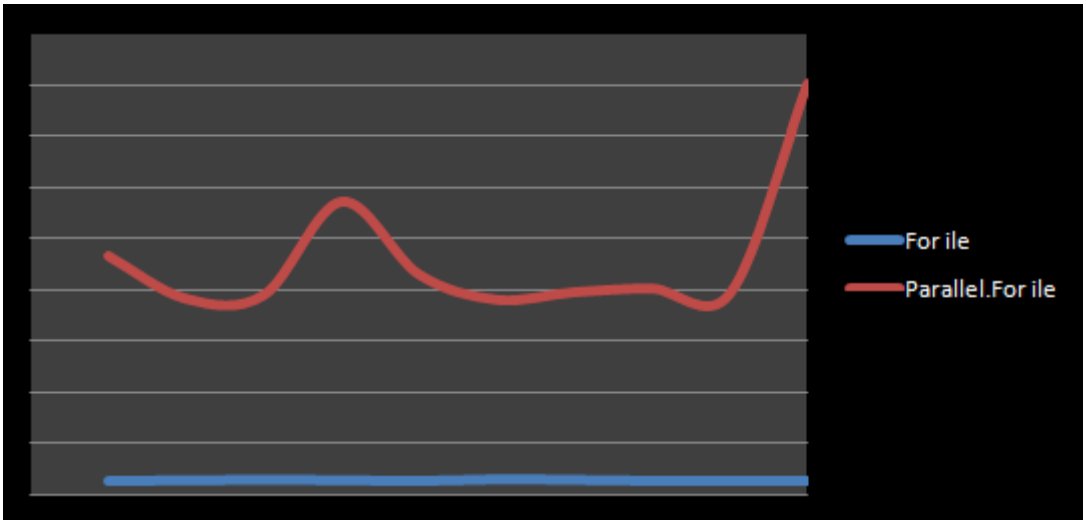
```
    }
}
```

Aslında kodumuz son derece basit. Eleman sayısını **1000** olarak set ettiğimiz **double** tipinden dizilere eleman eklemek*(ki eklerken işin uzun sürmesini sağlamak adına tamamen anlamsız bir matematik formülü içermektedir)* ve okumak için **for, foreach, Parallel.For** ile **Parallel.ForEach** metodlarını kullanmaktayız. Burada eleman sayısının bilhakis düşük tutulması son derece önemlidir aslında. Program çalıştırıldığında, for ve Parallel.For ile yapılan ekleme işlemleri ile, yine for, foreach ve Parallel.ForEach ile yapılan okuma işlemlerine ait toplam süre değerlerini bildirmektedir. Ben arka arkaya 10 deneme yaptıktan sonra ekleme işlemleri için aşağıdaki sonuçları elde ettim.

Eleman Ekleme

Deneme	For ile	Parallel.For ile
1	0,2592	4,6796
2	0,2718	3,8415
3	0,2799	3,913
4	0,2732	5,7423
5	0,2584	4,3159
6	0,291	3,8208
7	0,2782	3,9725
8	0,2612	4,041
9	0,2626	3,9412
10	0,2598	8,0166

Grafiksel olarak bakarsak çok daha acı bir gerçekle karşılabiliriz.

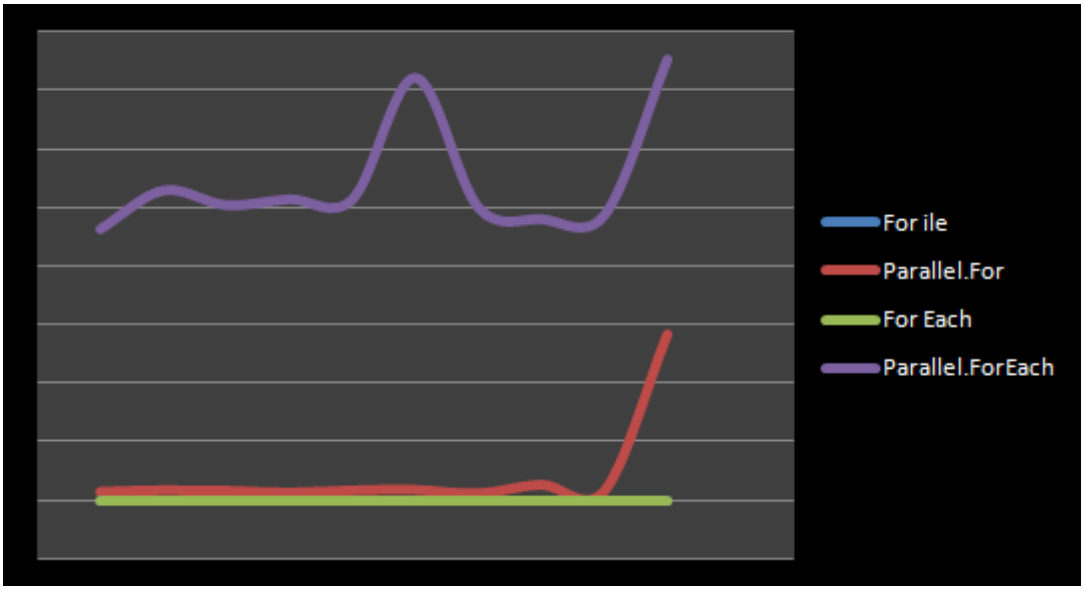


Görüldüğü üzere for ile gerçekleştirilen ekleme işlemi, eş zamanlı ve dolayısıyla paralel çalışabilen Parallel.For kullanımına göre çok daha hızlı yapılmıştır. Peki diziden veri okuma işlemi sırasındaki durum nedir? İşte sonuçlar;

Eleman Okuma

Deneme	For ile	Parallel.For	For Each	Parallel.ForEach
10	0,0128	0,6894	0,0145	23,1096
20	0,0125	0,8129	0,0134	26,3994
30	0,0128	0,7707	0,0139	25,152
40	0,0131	0,6062	0,0136	25,661
50	0,0128	0,8048	0,0139	25,6406
60	0,0131	0,842	0,0153	35,9978
70	0,0125	0,5439	0,0131	24,9442
80	0,0134	1,3035	0,0134	23,9636
90	0,0125	0,7635	0,0136	24,2947
100	0,0128	14,211	0,0142	37,5547

Okuma işlemlerinin grafiksel sonucu ise aşağıdaki şekilde görüldüğü gibidir.



Her ne kadar for süreleri grafik üzerinde görünmesede*(sıfıra çok yakın olduğu için)* en hızlı okuma süresi rekoru kendisine aittir.
Sonrasında **foreach** gelmektedir. **Parallel.For** nispeten belirli süre foreach' e yakın değerlerde seyretmesine rağmen, 10ncu denemede açılan paralel task' lerin canından bezmesi nedeni ile çok kötü bir süre üretmiştir. Ancak **Parallel.ForEach** bu teste göre sondan birinci olmuştur.

Bu testleri **Dual Core** işlemcili e **4 Gb Ram'** i olan, **Vista** yüklü bir sistem üzerinde denediğimde elde ettim. Elbetteki paralel tekniklerin burada kötü sonuçlar vermesinin en büyük nedeni işlemlerin zaten normal for veya foreach ' ler ile çok kısa sürede tamamlanabilmesidir. öyleki, bu süreler içerisinde işleyişi paralel iş parçalarına ayırmak için yapılacak tüm hazırlıklar, sürenin dahada uzamasına neden olmaktadır. Sonuç olarak şu noktayı vurgulamak gerekiyor,

Parallel.For ve **Parallel.ForEach** metodları, döngü içerisindeki işlemlerin gerçekten uzun sürelerde yapılabilldiği durumlarda kullanılmalıdır. Bu noktada kodun çalışacağı sistemin kapasitesi veya döngüler içerisinde yer alan işlemlerin maliyeti gibi pek çok etken, tamamlanma sürelerinde belirleyici rol oynamaktadır. Söz gelimi grafik tabanlı matematiksel hesaplamaların çok sayıda nesne örneği için yapılması gereken durumlarda*(**DirectX**, **OpenGL** kullanan grafik uygulamaları veya oyunlar)* paralel tekniklerden yararlanılması düşünülebilir.

Bir diğer önemli noktada aslında, **Parallel.For, Parallel.ForEach, Task, Parallel.Invoke** gibi kavramların paralel programlama genişletmesi*(Parallel Extensions)* olaraktan henüz beta aşamasında yer alan bir ürüne dahil edilmiş olmalarıdır ki **.Net Framework 3.5** ilede ek bir paket yüklenerek ele alınabilmektedir. Bu açıdan bakıldığında **Relase** sürümde farklılıklar olması veya arzu edilen iyileştirmelerin yapılmasıda muhtemeldir.

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ForForEachPerformance.rar (21,64 kb)

TPL - İptal İşlemi [Beta 1] (2009-06-08T22:19:00)

task parallel library,

Merhaba Arkadaşlar,

Bir önceki blog yazımda, TPL kullanılarak WinForms uygulamalarında paralel işlemlerin nasıl yapılabileceğini ele almaya çalışmıştım. örnekte son geldiğimiz noktaya bakıldığında aşağıdaki kazanımları elde ettiğimizi düşünebiliriz.

- **Parallel.ForEach** sayesinde resim dosyalarının iterasyonun daha hızlı gerçekleştirilebilmektedir.
- WinForms tarafındaki **Cross-Thread** ihlalinin önüne geçilmiştir.
- **Task** sınıfı üzerinden kullanılan **StartNew** metodu yardımıyla resim içeren **Button** kontrollerin üretildiği anda ekranda gösterilebilmesi sağlanmıştır.
- Yine **StartNew** metodunun kullanımı sayesinde kullanıcının paralel işlemler devam ederken, Form üzerindeki diğer kontroller ile etkileşimi sağlanmıştır.

Ancak biraz durup düşündüğümde unuttuğum önemli bir nokta olduğunu farkettim. İşlemler her ne kadar kısa gibi görünsede, kullanıcı bir yerde iptal etmek isterse ne olacak. Dolayısıyla uygulamaya bir iptal sürecininde ekleniyor olması gerekmekte. Aslında bu işlem son derece basit. Nitekim **Task** sınıfının bu işlemler için tasarlanmış olan **Cancel** isimli bir metodu bulunmaktadır. Lakin örnek dikkatlice göz önüne alındığında, resim dosyları üzerindeki iterasyonun **Parallel.ForEach** yardımıyla gerçekleştirildiği görülür. Dolayısıyla **Parallel.ForEach** içerisinde hareket edilirken, iptal talebi gelip gelmediğinin kontrol edilmesi gerekmektedir. Ki buda tek başına yeterli

değildir. Nitekim, **Parallel.ForEach** metoduda kendi içerisindeki işlemler için arka planda açtığı Task' leri kullanmaktadır. Dolayısıyla ForEach içeriğindeki task' lerinde iptal edilmesi gerekmektedir. Bu nedenle kod içeriğini aşağıdaki gibi değiştirmeliyiz.

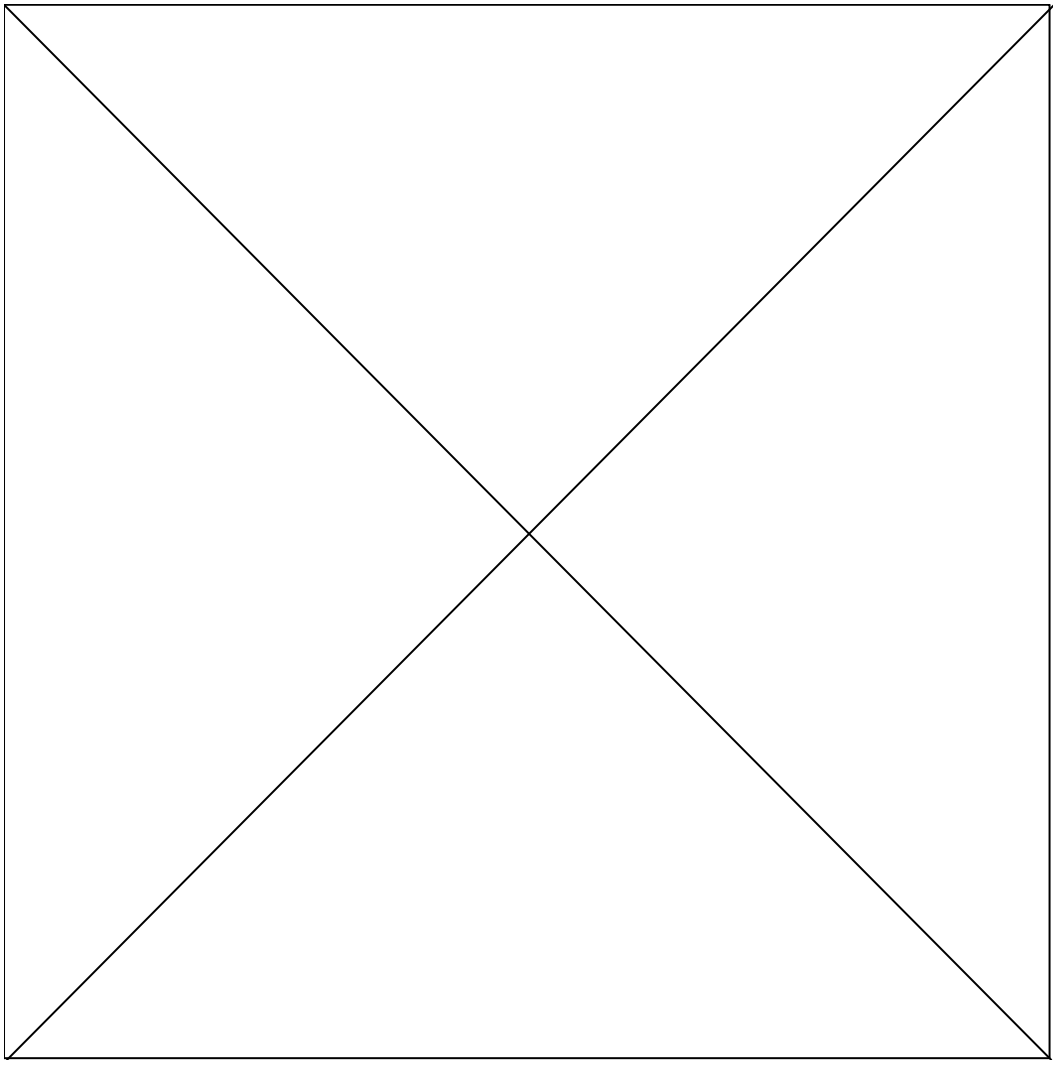
```
private Task task1 = null;

    private void btnStart4_Click(object sender, EventArgs e)
    {
       flowLayoutPanel1.Controls.Clear();
       Stopwatch watch = Stopwatch.StartNew();

       task1=Task.Factory.StartNew(() => FillImages(null));

       watch.Stop();
       lblElapsedTime.Text = String.Format("İşlemler {0} saniyede bitmiştir.",
watch.Elapsed.TotalSeconds.ToString());
    }

    private void btnCancel_Click(object sender, EventArgs e)
    {
       if (task1 != null)
          task1.Cancel();
    }

    private void FillImages(object state)
    {
       Task currentTask = Task.Current;

       Parallel.ForEach(Directory.GetFiles(imagesPath), (f, ls) =>
       {
          if (currentTask.IsCancellationRequested)
          {
             ls.Stop();
             return;
          }
          FileInfo fInfo = new FileInfo(f);
          if (fInfo.Length <= 1024 * 100
             && fInfo.Extension == ".jpg")
          {
             Thread.Sleep(100); // Bunu koymadığımızda UI istediğimiz gibi reaksiyon
vermiyor.
             Button btn = new Button();
             btn.Width = 64;
             btn.Height = 48;
             btn.BackgroundImageLayout = ImageLayout.Stretch;
```

```
            btn.BackgroundImage = Image.FromFile(f);
            AddToPanel(btn);
        }
    }
    );
}
```

Herşeyden önce iptal edilmek istenen **Task** sınıfına ait nesne örneğinin(task1 isimli değişken) sınıf seviyesinde bir değişken olarak ele alındığı görülmektedir. Kullanıcı iptal işlemi için **Button** kontrolüne tıkladığında, **task1** değişkeni üzerinden **Cancel** metodu çağırılmaktadır. Bu durumda çalışma zamanında **ForEach** döngüsü içerisinde yer alan **IsCancellationRequested** özelliği **true** değeri döndürecektir. Bu özelliğe ulaşmak için **Task** sınıfının **Current** özelliğinden yararlandığımıza dikkat edilmelidir. Bu sayede **ForEach** içerisinde **Parent** T**ask** örneğine ulaşılabilmektedir. Ardında ilginç bir kod parçası gelmektedir. **ls** isimli bir değişken üzerinden **Stop** metodu çağırılmıştır. İşte bu metod **ForEach** tarafından açılan **task'** lerin iptal edilmesini sağlamaktadır.
Aslında **Parallel** sınıfı içerisinde **ForEach** veya **For** metodları içerisinde kullanılan temsilcilere bakıldığında **ParallelLoopState** isimli bir sınıf kullanıldığı görülür. örneğin,

public static ParallelLoopResult ForEach<TSource>(IEnumerable<TSource> source, Action<TSource, **ParallelLoopState**> body);

Bu sınıf içerisinde **Stop**, **Break** gibi paralel döngünün durdurulması veya dışına çıkılması için gerekli metodlar yer almaktadır. örneğimizde **Action** temsilcisinin kullanıdığı ikinci generic parametre, çalışma zamanındaki **ParallelLoopState** nesne örneğine denk gelmektedir. Ve sonuç...



Tekrardan görüşmek dileğiyle hepinize mutlu günler dilerim.

TPL ile WinForms Macerası [Beta 1] (2009-06-07T20:53:00)

task parallel library,

Merhaba Arkadaşlar,

Dün gece **Task Parallel Library** ile ilgili olarak internette araştırma yaparken, örnekleri çoğunlukla(hatta tamamen) **Console** uygulamaları üzerinde geliştirdiğimi farkettim. Oysaki **TPL** veya**PLINQ** gibi alt yapıların, **WinForms** yada **WPF(Windows Presentation Foundation)** uygulamalarında nasıl kullanılabileceğide önemli bir konuydu. özellikle **Windows Form'** larının **TPL** çalışmalarına karşı nasıl tepkilerde bulunabileceği belkide en önemli noktaydı. Biliyorsunuz **TPL** alt yapısında, işlemci ve çekirdek gücü sonuna kadar kullanılmakta ve arka planda coşan pek çok **Thread** yer almaktadır. Fakat **WinForms** uygulamalarında herşeyin hakimi olan ana **Thread'** in genellikle bencil olduğuda bilinmektedir. Bu nedenle TPL ile çekilen bir veri içeriğinin,**Form** üzerindeki bir

kontrole doldurulması gerçekten başa bela olabilir. İşte bu düşünceler içerisinde yola çıktım ve örnek bir senaryo üzerinde durmaya çalıştım.

İlk olarak senaryodan biraz bahsedeyim; *bilgisayarımda resimlerin tutulduğu klasörde yer alan jpg dosyalarından 100 KB' ın altında olanları bulup, Form üzerindeki bir FlowLayoutPanel içerisinde Button bileşenleri ile göstermek istemekteyim. Kabaca aşağıdaki ekran görüntüsünde yer alan sonuçları elde etmek istediğimizi düşünebiliriz.*

İşe ilk olarak eski stilde başladım. Yani tek bir **Thread** ile resimleri doldurmayı denedim. Bunun için kod içeriğini ilk etapta aşağıdaki gibi geliştirdim.

```
using System;
using System.Diagnostics;
using System.Drawing;
using System.IO;
using System.Threading;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace TPLAntrenmanlari2
{
```

```
   public partial class Form1 : Form
   {
      private string imagesPath = @"C:\Users\Burak Selim Senyurt\Pictures";

      public Form1()
      {
         InitializeComponent();
      }

      private void btnStart_Click(object sender, EventArgs e)
      {
         flowLayoutPanel1.Controls.Clear();

         #region Single Thread Kullanılarak

         Stopwatch watch = Stopwatch.StartNew();

         foreach (string f in Directory.GetFiles(imagesPath))
         {
            FileInfo fInfo = new FileInfo(f);
            if (fInfo.Length <= 1024 * 100
               && fInfo.Extension == ".jpg")
            {
               Button btn = new Button();
               btn.Width = 64;
               btn.Height = 48;
               btn.BackgroundImageLayout = ImageLayout.Stretch;
               btn.BackgroundImage = Image.FromFile(f);
               flowLayoutPanel1.Controls.Add(btn);
            }
         }

         watch.Stop();
         lblElapsedTime.Text = String.Format("İşlemler {0} saniyede bitmiştir.",
watch.Elapsed.TotalSeconds.ToString());

         #endregion
      }

   }
}
```

İlk geliştirmede, resimlerin tutulduğu klasördeki dosyalar bir **foreach** döngüsü yardımıyla dolaşılmaktadır. Sonrasında ise uzantısı **jpg** olan ve **100 KB** altında olanlar belirlenmektedir. Bu kritere uyan her bir resim için bir **Button** kontrolü üretilmekte ve arka

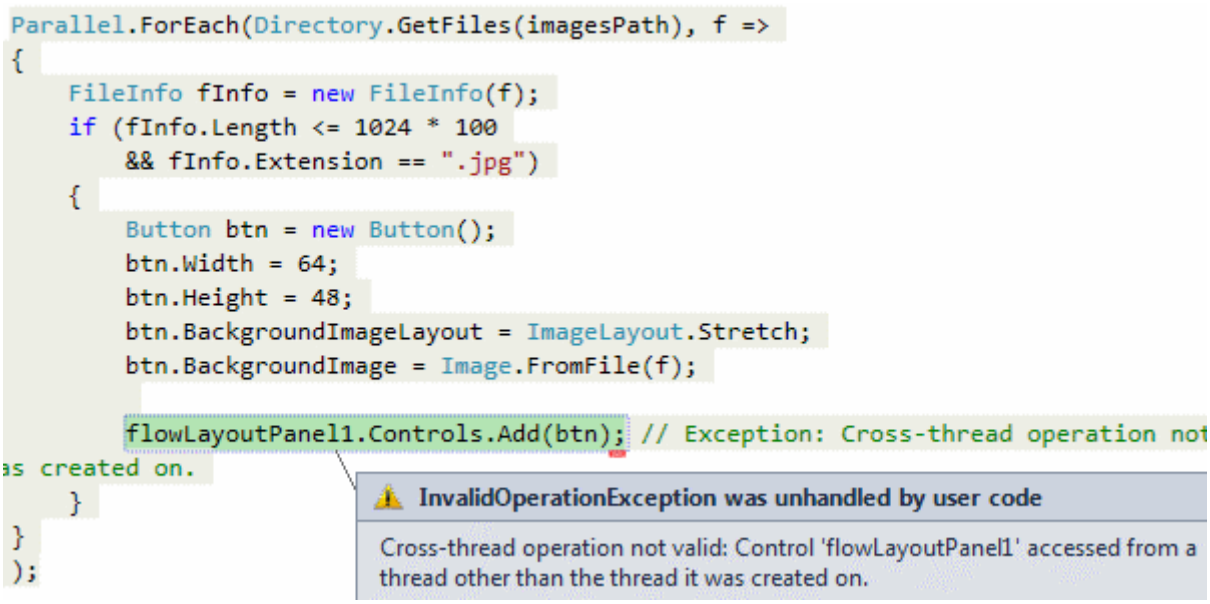
plan olarak bulunan resim kullanılmakadır. Tabiki son olarak söz konusu Button kontrolü, **FlowLayoutPanel** bileşeni içerisine eklenmektedir. Sonuçlar benim sisteminde aşağıdaki gibi gerçekleşmiştir.

Dikkat çekici nokta işlemlerin tamamlanma süresidir. Neredeyse 20 saniye. 😕 üstelik işlemler sırasında **Form'** u herhangibir yere çekiştiremediğimizi görürüz.
Ayrıca, **Button**bileşenleri oluşturulup **FlowLayoutPanel** kontrolüne eklenirken **Form** üzerinde görsel bir hareketlilik olmadığı gözlemlenebilir. Ancak tüm işlemler bittikten sonra Button' ların görülmesi mümkün olacaktır. Tabiki isteklerimizden ilki işlemlerin daha kısa sürede bitirilmesi olarak düşünülebilir. Bu amaçla btnStart2_Click kodlarında **Parallel.ForEach** kullanımını tercih ettim. İşte kodun yeni hali;

```
private void btnStart2_Click(object sender, EventArgs e)
        {
            flowLayoutPanel1.Controls.Clear();

            #region Parallel.ForEach kullanımı

            Stopwatch watch = Stopwatch.StartNew();
```

```
        Parallel.ForEach(Directory.GetFiles(imagesPath), f =>
        {
            FileInfo fInfo = new FileInfo(f);
            if (fInfo.Length <= 1024 * 100
                && fInfo.Extension == ".jpg")
            {
                Button btn = new Button();
                btn.Width = 64;
                btn.Height = 48;
                btn.BackgroundImageLayout = ImageLayout.Stretch;
                btn.BackgroundImage = Image.FromFile(f);

                flowLayoutPanel1.Controls.Add(btn); // Exception: Cross-thread operation not valid: Control 'flowLayoutPanel1' accessed from a thread other than the thread it was created on.
            }
        }
        );

        watch.Stop();
        lblElapsedTime.Text = String.Format("İşlemler {0} saniyede bitmiştir.", watch.Elapsed.TotalSeconds.ToString());

        #endregion
    }
```

Görüldüğü gibi tek fark **Parallel.ForEach** kullanımıdır. Bu sayede, **ForEach** içerisinde yer alan işlemlerin paralel iş parçalarına bölünerek gerçekleştirilmesi mümkün olacaktı. Ancak ortam **Console** değildi. Artık **WinForms** ortamındaydık. çevresel faktörler daha farklıydı. Dolayısıyla sonuç aşağıdaki gibi oldu.



İşte beklenen hayalet. Durumu şu şekilde açıklayabiliriz. **Windows** uygulaması çalıştırıldığıda yürümekte olan ana **Thread**, kendisini **Form** üzerindeki tüm kontrollerin sahibi olarak ilan etmiştir. Bu nedenle farklı bir Thread içerisinden, sahibi olduğu bir kontrole ulaşılmasına izin vermez. çözüm için pek çok farklı yol vardır. Ben bu yollardan birisi olan**Invoker'** lardan faydalanmaya karar verdim. İşte kodun son hali.

```
private void btnStart2_Click(object sender, EventArgs e)
    {
        flowLayoutPanel1.Controls.Clear();

        #region Parallel.ForEach kullanımı

        Stopwatch watch = Stopwatch.StartNew();

        Parallel.ForEach(Directory.GetFiles(imagesPath), f =>
        {
            FileInfo fInfo = new FileInfo(f);
            if (fInfo.Length <= 1024 * 100
                && fInfo.Extension == ".jpg")
            {
                Button btn = new Button();
                btn.Width = 64;
                btn.Height = 48;
                btn.BackgroundImageLayout = ImageLayout.Stretch;
                btn.BackgroundImage = Image.FromFile(f);
                AddToPanel(btn);
                // flowLayoutPanel1.Controls.Add(btn); // Exception: Cross-thread operation
not valid: Control 'flowLayoutPanel1' accessed from a thread other than the thread it was
created on.
            }
        }
        );

        watch.Stop();
        lblElapsedTime.Text = String.Format("İşlemler {0} saniyede bitmiştir.",
watch.Elapsed.TotalSeconds.ToString());

        #endregion
    }

    #region  Cross-thread operation not valid hatasına karşı mücadele

    private delegate void AddControlHandler(Button pb);
    private void AddToPanel(Button pb)
    {
```

```
        if (flowLayoutPanel1.InvokeRequired)
            flowLayoutPanel1.BeginInvoke(new AddControlHandler(RealAddToPanel),
new object[] { pb });
        else
            RealAddToPanel(pb);
    }
    private void RealAddToPanel(Button pb)
    {
        flowLayoutPanel1.Controls.Add(pb);
    }

    #endregion
```

Bu durumda kendi sistemimde aşağıdaki sonuçlar ile karşılaştığımı gördüm.

Evett...Durumu bir değerlendirelim. 20 saniyelik sürelerden yaklaşık 8 saniyelik sürelere indik. Bu çift çekirdekli bir sistem için iyi bir sonuç olarak görünüyor. *(Tabi kodu daha fazla çekirdek sayısı bir sistemde ne yazıkki test edemedim. Ama siz değerli okurlarımdan test etme fırsatı olan olursa sonuçları paylaşmasını rica edeceğim.)* Yinede herşey istediğimiz gibi değildir. Süre azalmasına rağmen, **Form'** u işlemler sırasında harekete ettiremediğimizi görürüz. Benzer şekilde resimleri içeren **Button** kontrolleri yine

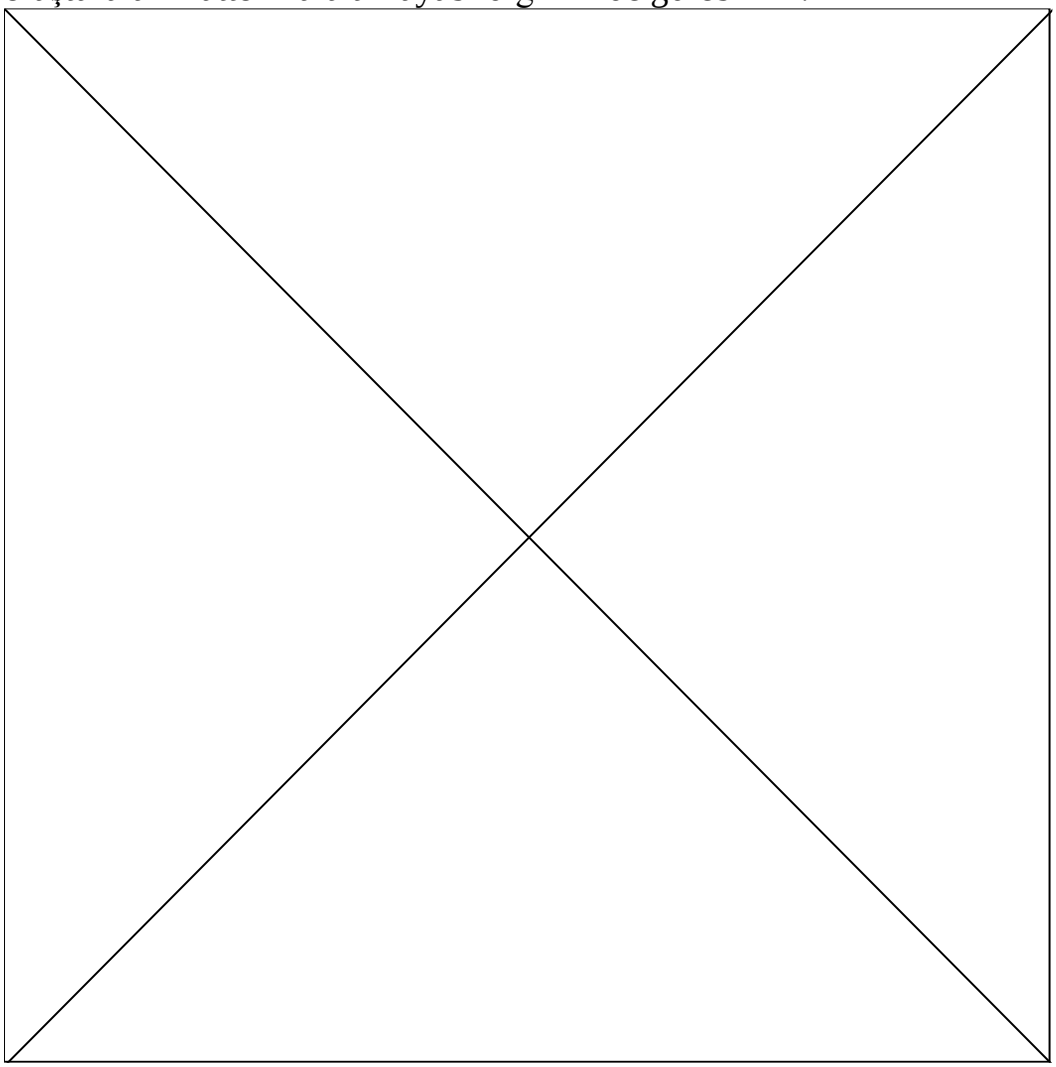
üretildikçe değil tüm işlemler bittikten sonra bir anda ekranda gösterilmektedir. Dolayısıyla **Parallel.ForEach'** in tam anlamıyla yeterli gelmediğini söyleyebiliriz. çözüm olarak **ThreadPool** sınıfından yararlanabiliriz aslında. Şimdi kodu aşağıdaki gibi değiştirdiğimizi düşünelim.

```
#region  Cross-thread operation not valid hatasına karşı mücadele

    private delegate void AddControlHandler(Button pb);
    private void AddToPanel(Button pb)
    {
       if (flowLayoutPanel1.InvokeRequired)
          flowLayoutPanel1.BeginInvoke(new AddControlHandler(RealAddToPanel),
new object[] { pb });
       else
          RealAddToPanel(pb);
    }
    private void RealAddToPanel(Button pb)
    {
       flowLayoutPanel1.Controls.Add(pb);
    }

    #endregion

    private void FillImages(object state)
    {
       Parallel.ForEach(Directory.GetFiles(imagesPath), f =>
       {
          FileInfo fInfo = new FileInfo(f);
          if (fInfo.Length <= 1024 * 100
             && fInfo.Extension == ".jpg")
          {
             Thread.Sleep(100); // Bunu koymadığımızda UI istediğimiz gibi reaksiyon
vermiyor.
             Button btn = new Button();
             btn.Width = 64;
             btn.Height = 48;
             btn.BackgroundImageLayout = ImageLayout.Stretch;
             btn.BackgroundImage = Image.FromFile(f);
             AddToPanel(btn);
          }
       }
       );
    }

    private void btnStart3_Click(object sender, EventArgs e)
```

```
    {
        flowLayoutPanel1.Controls.Clear();

        #region Parallel.ForEach kullanımı

        Stopwatch watch = Stopwatch.StartNew();

        ThreadPool.QueueUserWorkItem(new WaitCallback(FillImages));

        watch.Stop();
        lblElapsedTime.Text = String.Format("İşlemler {0} saniyede bitmiştir.",
watch.Elapsed.TotalSeconds.ToString());

        #endregion
    }
```

QueueUserWorkItem metodu parametre olarak **WaitCallback** temsilcisini kullanmaktadır. Bu temsilci ise **FillImages** metodunu işaret etmektedir.
İşlemler **FillImages** metodu içerisinde yapılmaktadır. Bu durumsa sonuçlar çok daha ilginç olacaktır. Button kontrolleri oluşturuldukça **FlowLayoutPanel** kontrolü içerisindede görünür hale gelecektir. Ayrıca, **Form'** u işlmeler sırasında sürükleyebildiğimizi veya

oluşturulan Button' lara tıklayabildiğimizide görebiliriz.



Ancak zaman ilerlemiştir ve artık Task sınıfı ve üyeleri ile aynı işlemi nasıl gerçekleştirebileceğimize bakmamız gerekmektedir. Sonuç itibaryle .Net 4.0 için aynı işleyişi aşağıdaki kod parçası ile gerçekleştirebiliriz.

```
private void btnStart4_Click(object sender, EventArgs e)
      {
         flowLayoutPanel1.Controls.Clear();
         Stopwatch watch = Stopwatch.StartNew();

         Task.Factory.StartNew(() => FillImages(null));

         watch.Stop();
         lblElapsedTime.Text = String.Format("İşlemler {0} saniyede bitmiştir.",
watch.Elapsed.TotalSeconds.ToString());
      }
```

Bir önceki yazımızdan hatırlayacağınız gibi **Task** sınıfı üzerinden **StartNew** metodunu kullanarak paralel görevlerin başlatılması sağlanabilmektedir. Burada metoda parametre

olarak**Action** temsilcisinin işaret edebileceği **FillImage** fonksiyonu verilmiştir. Sonuçlar yine yukarıdaki Flash animasyonundakine benzer olacaktır. Kullanıcılar, resimleri gösteren **Button** kontrolleri yüklenirken, Formun diğer alanları ile etkileşimde bulunubilmektedir. Ayrıca Button bileşenleri oluşturuldukça **FlowLayoutPanel** içerisinde görülebilmektedir. Ancak kod içerisinde küçük bir hile yaptığımı belirtmek isterim. Dikkat ederseniz **FillImages** metodu içerisinde o anki **Thread** için **100** milisaniye kadar bir duraksatma yapılmaktadır. Bu yapılmadığı takdirde **Button**bileşenlerinin oluşturuldukça **FlowLayoutPanel** içerisinde gösterilmelerinde bir sıkıntı olduğu gözlemlenir. Açıkçası Tutarsız bir çalışma olmaktadır. Ancak şimdilik bu gecikmenin olmasında bir sakınca yoktur. Nitekim, kullanıcı zaten paralel süreç içerisindeki işlemlerden her biri bittikçe, tamamlanan o işi ele alabilmektedir. Yani işlemlerin tamamalanmasının beklenmesine gerek kalınmadan çalışmaya devam edilebilmektedir.

Böylece geldik bir blog yazımızın daha sonuna. İlerleyen dönemlerde aynı senaryoyu bir WPF uygulaması için ele almaya çalışıyor olacağım. Görüşmek dileğiyle.

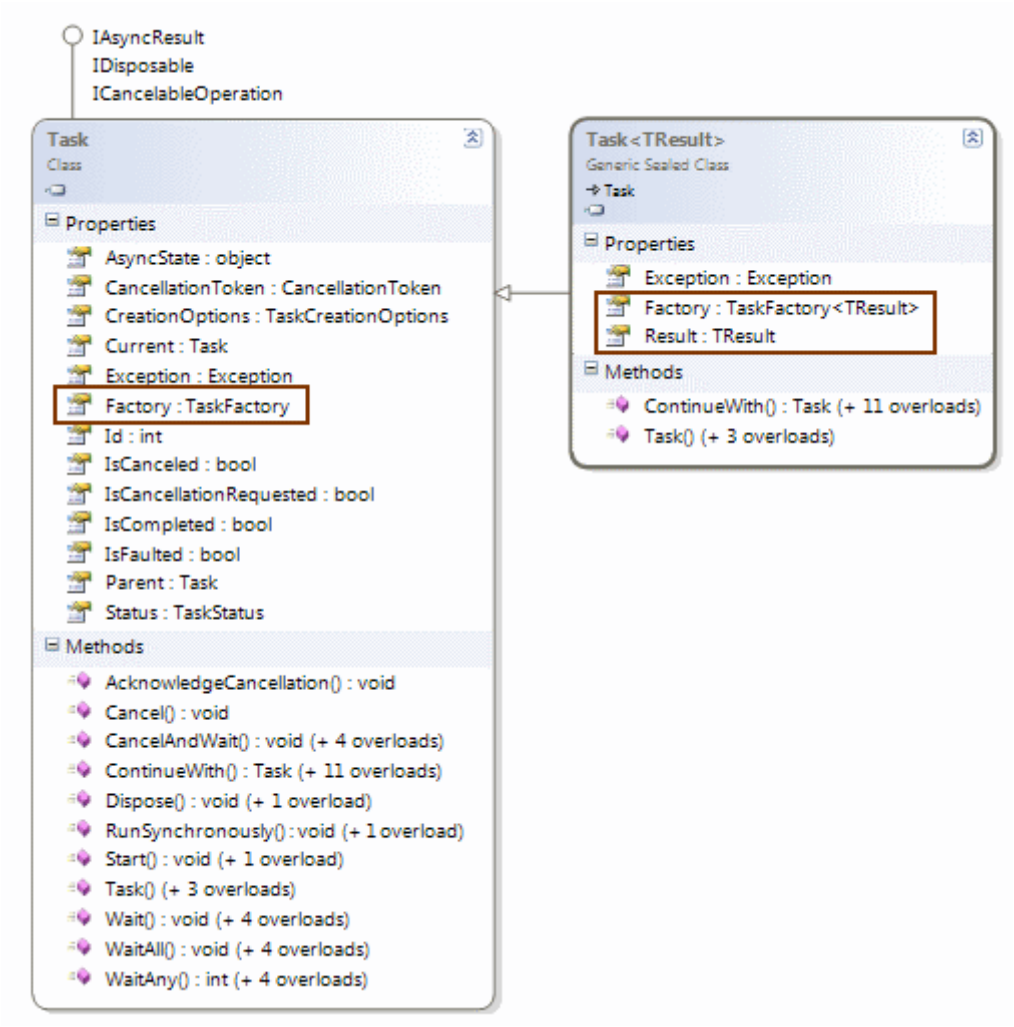
TPLAntrenmanlari2.rar (40,63 kb)

TPL için Önemli Bir Kavram : Task [Beta 1] (2009-06-05T04:05:00)

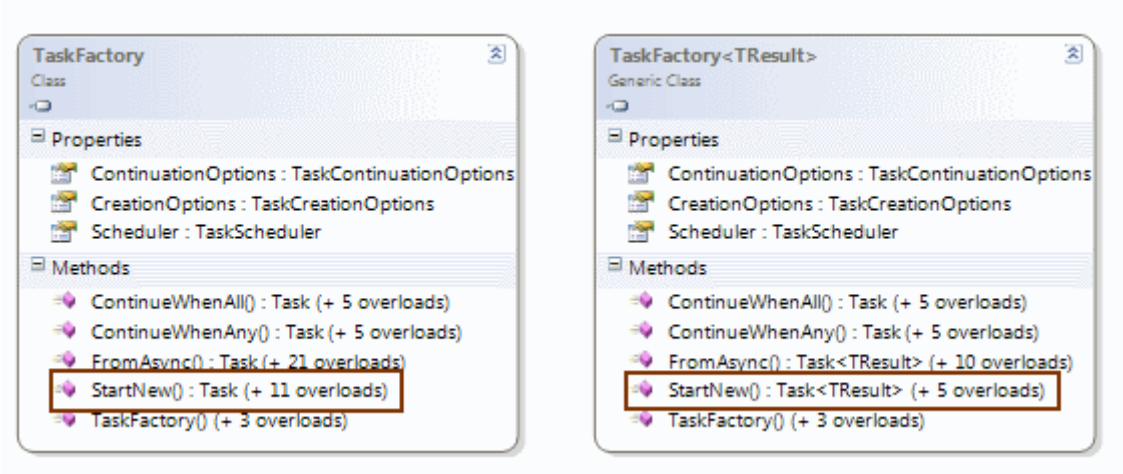
task parallel library,

Merhaba Arkadaşlar,

Bir önceki blog yazımda **Task Parallel Library** alt yapısının ne olduğunu sizlere aktarmaya çalışmıştım. Tabiki bu alt yapı üzerinde durulması gereken pek çok konu bulunmaktadır. Heyecanım çok, anlatmak içinde sabırsızlanıyorum. Ama her zamanki gibi adım adım ilerlemekte ve acele etmemekte yarar olduğu kansındayın. TPL ile ilişkili önemli konulardan birisi**Task(yada Task<T>)** sınıfıdır. **TPL** esas itibariyle görev adı verilen küçük iş parçaları üzerine kurulu bir yapı olarak düşünülebilir. Bu nedenle **Task** sınıfı son derece önemlidir. Nitekim görevlerin yönetimli kod tarafındaki ifadesidir. Bu sınıf yardımıyla, paralel çalışacak olan görevlerin başlatılması, iptal edilmesi, bekletilmesi, arka arkaya eklenerek bir süreç tesis edilmesi gibi pek çok işlem yapılabilir. Task sınıfı normal şartlarda geriye değer döndürmeyen fonksiyonelliklerin eş zamanlı olarak çalıştırılmasında ele alınmaktadır. Geriye değer döndüren metodlar söz konusu olduğunda ise, **Task<T> generic** tipinden yararlanılabilir. Buradaki **T**, paralel çalışan metodun dönüş tipi olarak düşünülebilir. Aşağıdaki sınıf diagramında söz konusu tipler ve üyeleri yer almaktadır.



Aslında Task ve Task<T> sınıflarının static **Factory** özelliği üzerinden gidildiğinde **StartNew** metodu yardımıyla görevlerin başlatılması sağlanmaktadır. Diğer yandan **Task<T>**sınıfının **Result** özelliği, geri dönüş tipini belirtmektedir. Ayrıca sınıf diagramındanda görüldüğü gibi **Task<T>** sınıfı, **Task** sınıfından türemektedir. **Factory**özellikleri, **TaskFactory** veya **TaskFactory<T>** tipinden referanslar barındırmaktadır. Bu tiplerin içeriği ise aşağıdaki şekilde görüldüğü gibidir.



Tüm tiplerde pek çok önemli üye bulunmaktadır. Bunların hemen hepsini zaman içerisinde ele almaya gayret edeceğiz, hiç merak etmeyin. Şimdi gelin **Task** ve **Task<T>** sınıflarını basit (ve her zamanki gibi tam anlamıyla gerçek hayat örneği olmayan) bir örnek üzerinden ele almaya çalışalım. İlk olarak senaryomuzdan bahsedelim. Senaryomuza göre resim dosyalarına ait 3 farklı işlemin gerçekleştiği metodların eş zamanlı ve paralel olarak çalıştırılmasını sağlamayı hedefliyoruz. Buna göre bir klasörden,

- Resim dosyalarının toplam boyutunun bulunması,
- Resimler içerisinde bmp olanların kaç adet olduklarının tespit edilmesi,
- Resimler içerisinde bmp olanların farklı bir klasöre kopylanması,

işlemlerini gerçekleştiren fonksiyonelliklerimiz bulunmakta.

Normal şartlar altında herkesin burada durup biraz düşünmesi gerekiyor. Elimizde **.Net Parallel Extensions** olmadığını varsayalım. Bu durumda ya **Multi-Thread** mimarisini kullanacağız, yada **delegate(temsilci)** tiplerinden yararlanarak **asenkron erişim modellerini(Polling, Callback, WaitHandle, Event-Based)** ele alacağız. Bunu bir düşünün ve senaryoyu bu materyaller ile yazmayı bir deneyin.

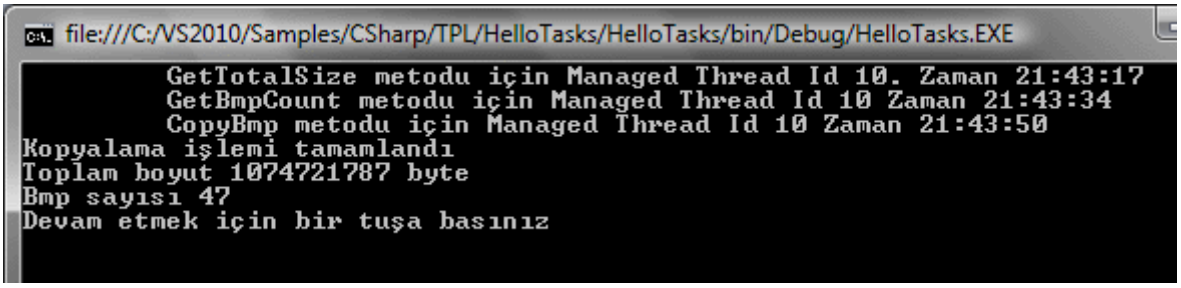
Tabi şunu biliyoruzki **TPL** alt yapısı, paralel işlemleri kolayca ele almamızı sağlayacak şekilde tasarlanmıştır. İlk etapta kodlarımızı aşağıdali gibi geliştirdiğimiz varsayalım.*(Kodlarımızı **Visual Studio 2010 Beta 1** üzerinde geliştirdiğimizi hatırlatayım)*

```
using System;
using System.Configuration;
using System.IO;
using System.Threading;
using System.Threading.Tasks;

namespace HelloTasks
{
   class Program
   {
      static string imagesPath = ConfigurationManager.AppSettings["ImagesPath"];

      static void Main(string[] args)
      {
         long totalSize = GetTotalSize();
         int bmpCount = GetBmpCount();
         CopyBmp();
         Console.WriteLine("Toplam boyut {0} byte\nBmp sayısı {1}",
totalSize.ToString(), bmpCount.ToString());
```

```
        Console.WriteLine("Devam etmek için bir tuşa basınız");
        Console.Read();
    }

    static long GetTotalSize()
    {
        Console.WriteLine("\t GetTotalSize metodu için Managed Thread Id {0}. Zaman
{1}",Thread.CurrentThread.ManagedThreadId.ToString(),DateTime.Now.ToLongTimeStri
ng());
        string[] files=Directory.GetFiles(imagesPath);
        long totalSize = 0;
        foreach (string file in files)
        {
            FileInfo fInfo = new FileInfo(file);
            totalSize += fInfo.Length;
            Thread.Sleep(10); // işlemleri biraz geciktirmek için bilinçli olarak konulmuştur
        }
        return totalSize;
    }

    static int GetBmpCount()
    {
        Console.WriteLine("\t GetBmpCount metodu için Managed Thread Id {0} Zaman
{1}", Thread.CurrentThread.ManagedThreadId.ToString(),
DateTime.Now.ToLongTimeString());
        int result = 0;

        foreach (string file in Directory.GetFiles(imagesPath))
        {
            FileInfo fInfo = new FileInfo(file);
            Thread.Sleep(10); // işlemleri biraz geciktirmek için bilinçli olarak konulmuştur
            if (fInfo.Extension.Contains("bmp"))
                result++;
        }

        return result;
    }

    static void CopyBmp()
    {
        Console.WriteLine("\t CopyBmp metodu için Managed Thread Id {0} Zaman {1}",
Thread.CurrentThread.ManagedThreadId.ToString(),
DateTime.Now.ToLongTimeString());
        foreach (string file in Directory.GetFiles(imagesPath))
        {
```

```
            FileInfo fInfo = new FileInfo(file);
            if (fInfo.Extension.Contains("bmp"))
            {
               File.Copy(file, "C:\\Bitmaps\\"+ fInfo.Name,true);
            }
         }
         Console.WriteLine("Kopyalama işlemi tamamlandı");
      }
   }
}
```

Şunu hemen belirteyim; aslında **Directory** ve **FileInfo** sınıflarının söz konusu hesaplamalar için kolaylaştırıcı metodları zaten mevcut. Söz gelimi **Directory** sınıfının **GetFiles**metoduna filtre uygulayarak zaten **bmp** dosyalarını kolayca elde edebiliriz. Yada bmp dosyalarını ele alırken kopyalama işlemlerinide yapabiliriz. Ancak yazının başında da bahsettiğim üzere bu sadece örnek bir senaryo malzemesi. önemli olan nokta **GetTotalSize**, **GetBmpCount** ve **CopyBmp** metodlarının paralel olarak çalıştırılmalarını sağlamak. Tabi şu andaki kod parçamız bu metodları **ardışık(Sequential)** olarak çalıştırmaktadır. Uygulamanın çalışma zamanı çıktısına baktığımızda ise aşağıdaki ekran görütüsündekine benzer sonuçları alırız.



Sanıyorumki metodların başlangıç zamanları ve aralarındaki farklar dikkatinizi çekmiştir. Bu zaten beklediğimiz bir sonuçtur aslında. Nitekim metod görevlerinin paralel olarak ele alınması için hiç bir şey yapmadık. Kodu paralel programlama felsefesine taşımak için aşağıdaki gibi değiştirmemiz gerekmektedir.

```
static void Main(string[] args)
      {
         Task[] tasks =
         {
            Task<long>.Factory.StartNew(GetTotalSize),
            Task<int>.Factory.StartNew(GetBmpCount),
            Task.Factory.StartNew(CopyBmp)
         };

         /* tasks isimli dizi içerisindeki Task<T> tipleri aynı generic tip ile
kullanılmadıklarında Task<T>[] gibi bir dizi üretilememiş bu nedenle 0 ve 1nci
```

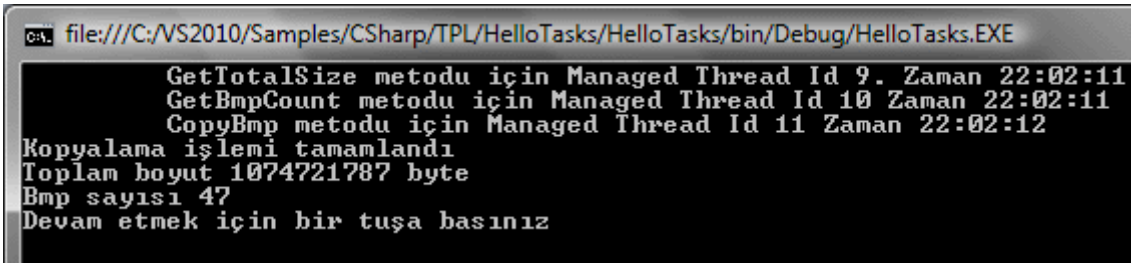
indislerdeki Task tiplerinin Result özelliklerine ulaşabilmek için bilinçli olarak Task<T> tiplerine dönüşüm yapılmıştır. */

```
        Console.WriteLine("Toplam boyut {0} byte\nBmp sayısı {1}"
            , ((Task<long>)tasks[0]).Result.ToString()
            , ((Task<int>)tasks[1]).Result.ToString()
            );

        Console.WriteLine("Devam etmek için bir tuşa basınız");
        Console.Read();
    }
```

İlk olarak **Task** tipinden bir dizi ürettildiğini görmekteyiz ki bir dizi kullanmanın bir zorunluluk olmadığını biraz sonra göreceğiz. Nihayetinde elimizde birden fazla görev var. Bu ilk kod denememizde, görevlerin tamamı bir dizi içerisinde toplanmaktadır. Dizinin her bir elemanının oluşturulması sırasında **Factory** özelliği üzerinden **StartNew** metodunun çağırıldığına dikkat edelim. Bu noktada parametre olarak belirtilen metodların, Task' ler diziye eklenirken çalıştırıldığını söyleyebiliriz. Kodun devam eden kısmında ise, generic Task tiplerinin çalıştırdığı metodlardan gelen dönüş değerleri ele alınmak istenmektedir. Dönüş değerlerinin ele alınması sırasında bilinçli olarak tür dönüşümü yapıldığına dikkat edilmelidir. Nitekim, dönüş değerleri ancak **Task<T>** generic sınıfının **Result** özellliği üzerinden ele alınabilmektedir.

*NOT : Tabiki bazı senaryolarda, tüm görevler aynı dönüş tipine sahip olabilirler. Bu durumda dizinin **Task<T>** tipinden tasarlanmış olması halinde, dönüştürme işlemlerine gerek olmadan sonuçlar alınabilir. Nitekim dizi üzerinde hareket edecek basit bir **for each** döngüsünün ele alacağı her bir eleman **Task<T>** tipinde olacağından, zaten **Result** özelliklerine otomatikman ulaşılabilecektir.*

Bu noktada şunu vurgulamaktada yarar var; bazı durumlarda paralel çalışan metodların işlemlerini tamamlamadan kodun devam etmesi istenmeyebilir. Bu durumdada Task sınıfının **staticWaitAll** veya **WaitAny** gibi metodlarını kullanarak gereken bekletmeleri yapabiliriz. örneğimizde buna gerek kalmamıştır. çünkü generic Task tiplerinin işaret ettiği metodlara ait dönüş tipleri alınmak istendiğinden, uygulama kodu zaten o anda sonuç gelmediyse mecburen beklemede kalacaktır. Peki örneği çalıştırdığımızda nasıl bir sonuç alırız.



Mutlaka dikkatinizi çekmiştir; her metod için ayrı bir **Managed Thread Id** değeri üretilmektedir. Oysaki **ardışık(Sequential)** çalışan modelde tüm metodlar

aynı **Thread** içerisinde ele alınmıştır. Bu, **Thread** bölünümünün de bir göstergesidir. Diğer taraftan, metodlar arası süre farklılıkları neredeyse sıfıra yakındır. Görüldüğü gibi gayet basit bir şekilde işlemleri paralel hale getirmeyi başardık. Kod ile ilişkili önemli bir noktayı daha vurgulamak isterim. Biraz önce bahsettiğimiz gibi, aynı dönüş tipine sahip metodların kullanıldığı senaryolarda Task dizilerini kullanmak daha mantıklıdır. Bu nedenle yukarıdaki senaryoda yer alan kodda dizi kullanımı şart değildir. Bir başka deyişle aynı amacı yerine getirden bir kod parçası, aşağıdaki şekilde olduğu gibi ele alınabilir.

```
Task<long> task1=Task<long>.Factory.StartNew(GetTotalSize);
Task<int> task2=Task<int>.Factory.StartNew(GetBmpCount);
Task task3=Task.Factory.StartNew(CopyBmp);

Console.WriteLine("Toplam boyut {0} byte\nBmp sayısı {1}"
                , task1.Result.ToString()
                , task2.Result.ToString()
                );
```

Bu sefer **GetTotalSize** ve **GetBmpCount** metodlarını kullanan **Task** tiplerine ait çalışma zamanı referansları, birer değişkene atanarak kullanılmışlardır. Bu durumda bir önceki örnekte yaptığımız gibi **Result** özelliğine erişmek için, **cast** işlemi yapılmasına da gerek kalmamaktadır ki bu oldukça doğru bir yoldur. Dolayısıyla kodu daha düzgün bir hale getirmiş bulunuyoruz. Sizlerde Task ve Task<T> sınıflarını kullanarak bir kaç antrenman yapmayı deneyebilirsiniz.

*NOT : **Visual Studio 2010** ile birlikte gelen **Parallel Tasks** ve **Parallel Stacks debugger** pencereleri yardımıyla çalışma zamanında, task ve thread' lerin durumunu daha net bir şekilde analiz edebilirsiniz. Bu konuyu bir görsel dersimizde ele almaya çalışacağım.*

Tabiki konuyu daha derinlere genişletmek mümkündür. örneğin bazı görevlerin, kendinden önceki görev(ler) tamamlandıktan sonra başlatılması istenebilir. İşte diğer blog yazımın konusunu şimdiden bulduk. Tabi başımıza dert açacak daha pek çok konuda var. Söz gelimi, **TPL** alt yapısını **WinForms** yada **WPF** gibi uygulamalarda ele aldığımızda neler olacaktır kimbilir. Malum **WinForms** yada **WPF** ekranlarında, **Main Thread** bencillik edip ekran üzerindeki kontrolleri başka **Thread'** ler ile paylaşmak istemez. Bu bencilliğe ortak olduğumuzda WinForms tarafında Illegal Cross Thread istisnalarına düştüğümüzü gayet iyi biliyoruz. Bu ve benzeri diğer konuları ilerleyen zamanlarda irdelemeye devam ediyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HelloTasks.rar (23,71 kb)

TPL(Task Parallel Library) Nedir? [Beta 1] (2009-06-03T11:50:00)

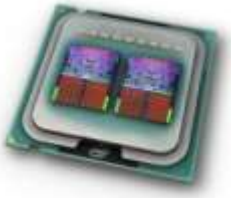
task parallel library,

Merhaba Arkadaşlar,

Uzun uzun zaman önceydi. İlk bilgisayarımı daha dün gibi hatırlıyorum.
Efsane **Commodore 64**.

Açıkçası onunla yaptığım tek şey oyun oynamaktı itiraf ediyorum. En çok sevdiğim oyunlar arasında **Grean Beret, Barbarian, Karate Kid 2, 1942, Airwolf** vardı. Gel zaman git zaman, üniversite yıllarına girince, bilgisayar işini daha ciddi düşünmeye başlamıştım. Yanlış hatırlamıyorsam yaklaşık olarak **2400** dolar değerinde (🤑) **486DX** işlemcili bir bilgisayarım daha olmuştu.

Sonrada olayların ardı arkası kesilmedi ve **Pentium MMX, Celeron** derken, **çift çekirdekli** ve hatta şu sıralarda moda olan **4 çekirdekli** işlemciler ile karşılaştık.



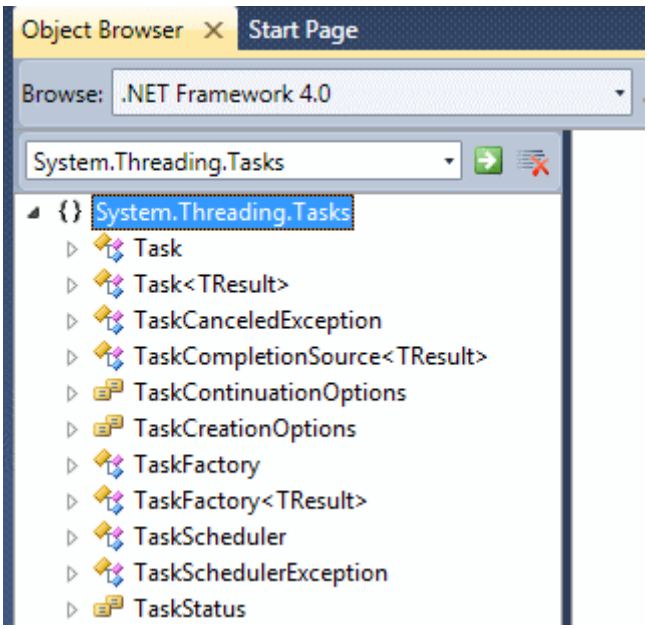
özellikle işlemcilerin bu şekilde ilerlemesine paralel olarak, yazılım geliştirme ortamlarında da pekala pek çok değişiklik ve yenilikçi fikir ortaya çıktı. Son zamanların özellikle **Microsoft .Net** cephesindeki en popüler konularından biriside **paralel genişletmeler(Parallel Extensions).** Bir başka deyişle, sistemin sahip olduğu işlemci gücünün tümünü kullanarak(Arabanın hakkını ver hakkını 🙂), paralel işlemler veya eş zamanlı yürütmelerin gerçekleştirilmesi. Bildiğiniz gibi paralel genişletmelerin önemli kısımlarından birisi olan **PLINQ(Parallel Language INtegrated Query)** alt yapısı üzerine yaptığım araştırmalarımı ve edindiğim bilgileri bir süredir sizlerle paylaşmaktayım. İşte bu yazımızda diğer önemli parça olan (belkide ile etapta incelenmesi gereken) **TPL(Task Parallel Library)** alt yapısını incelemeye başlıyor olacağız.

TPL' in en büyük amacı, eş zamanlı veya paralel olarak yürütülmek istenen işlemlerin, daha kolay ve basit bir şekilde ele alınmasını sağlamaktır. Bu anlamda günümüz işlmecilerinin çekirdek sayısı veya sistemlerdeki işlemci sayısının birden fazla olması durumunda, TPL verimli sonuçlar elde etmemizi sağlamaktadır. Bu açıdan bakıldığında TPL alt yapısına tüm sistem çekirdek gücünü verme imkanına sahip olduğumuzu belirtebiliriz. Ancak elbetteki bu güç yanlış anlaşılmamalı ve kullanılmamalıdır. Bildiğiniz gibi**"kontrolsüz güç, güç değildir"** derler

Elbetekki TPL kullanımı ile ilişkili olarak unutulmaması gereken bir noktada, işlemlerin **Multi-Threading** mantığına göre yapılıyor olmasıdır. Dolayısıyla, programın çalışma zamanı yükünü arttırıcı bir etkendir. Bir başka deyişle her işlemin, elimizde TPL var diye paralel olarak yürütülmeye çalışılması doğru değildir. Bazı süreçlerin gerçekten ve bilinçli olaraktan **ardışık(Sequential)** yürümesi gerekebilir.

NOT : *Aslında, PLINQ(Parallel Language INtegrated Query) kendi alt yapısında TPL tipleri ve üyelerinden destek almaktadır.*

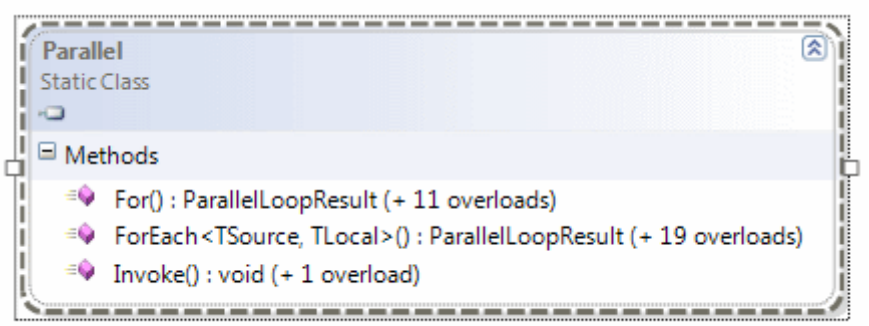
Artık olaya biraz daha teknik açıdan bakabileceğimizi düşünüyorum. TPL esas itibariyle **.Net Framework 4.0** ile birlikte gelen ve paralel işlem yapma yeteneklerini ele alan kütüphanedir.**System.Threading** ve **System.Threading.Tasks** isim alanları bu kütüphaneye ait çeşitli tipleri ve üyelerini içermektedir. TPL, içerisinde birde **Task Scheduler** içerir. Bu planlayıcı **ThreadPool**ile **TPL** tipleri arasındaki entegrasyonu sağlamaktadır. Ancak istenirse kendi özel görev planlayıcılarımızı yazabilir ve kullanabiliriz. Gerçi benim buna niyetim yok Aşağıdaki şekilde özellikle System.Threading.Tasks isim alanı altında yer alan tipler görülmektedir.



Geliştirme sürecinde özellikle **System.Threading.Tasks** isim alanı altında yer alan tipler kullanılmaktadır. TPL kütüphanesinin belkide en önemli tipi **Task** sınıfıdır. Task sınıfına ait üyeler kullanılarak aşağıdaki işlemleri gerçekleştirebiliriz;

- Yeni görev(ler) başlatılabilir, iptal edilebilir yada bekletilebilir.
- Bir görevin tamamlanması halinde, tamamlandığı yerden başka bir görev veya görevlere çağrıda bulunulabilir.
- Başlatılan görevlerden geriye değer döndürülebilir.
- Bir görev kendi içinde alt görevler başlatabilir. Bu görevler aynı Thread içerisinde veya farklı bir Thread üzerinde çalışıyor olabilir.

Tüm teorik bilgiler bir yana, konuyu ilk etapta kavramının en kolay yolu basit bir örnek üzerinden ilerlemektir. Bu anlamda, TPL' e **Hello World** demenin belkide en kolay yolu, **System.Threading** isim alanı altında yer alan **Parallel static** sınıfı ve üyelerini kullanmaktır.



Görüldüğü gibi **For**, **ForEach** ve **Invoke** isimli bizlere çok tanıdık gelen metodlar yer almaktadır. Bu fonskiyonellikleri kullanarak işlemlerin paralel olarak yürütülmesi sağlanabilmektedir.**For** ve **ForEach** metodları adlarındanda anlaşılacağı üzere, koleksiyon veya dizi yapıları üzerinde döngüsel işlemlerin paralel olarak yürütülmesini sağlamaktadır. **Invoke**metodu ise sunduğu **Action temsilcisi(delegate)** yardımıyla, birden fazla metodun aynı anda paralel olarak çalıştırılabilmesine olanak sağlamaktadır.

NOT : *For veya ForEach gibi Parallel sınıfına ait üyeleri kullandığımız hallerdede, arka planda Task sınıfı ve üyeleri gizlice devreye girerler.*

Aşağıdaki **Console** uygulamasında bu metodalara ait örnek kullanımlar yer almaktadır.

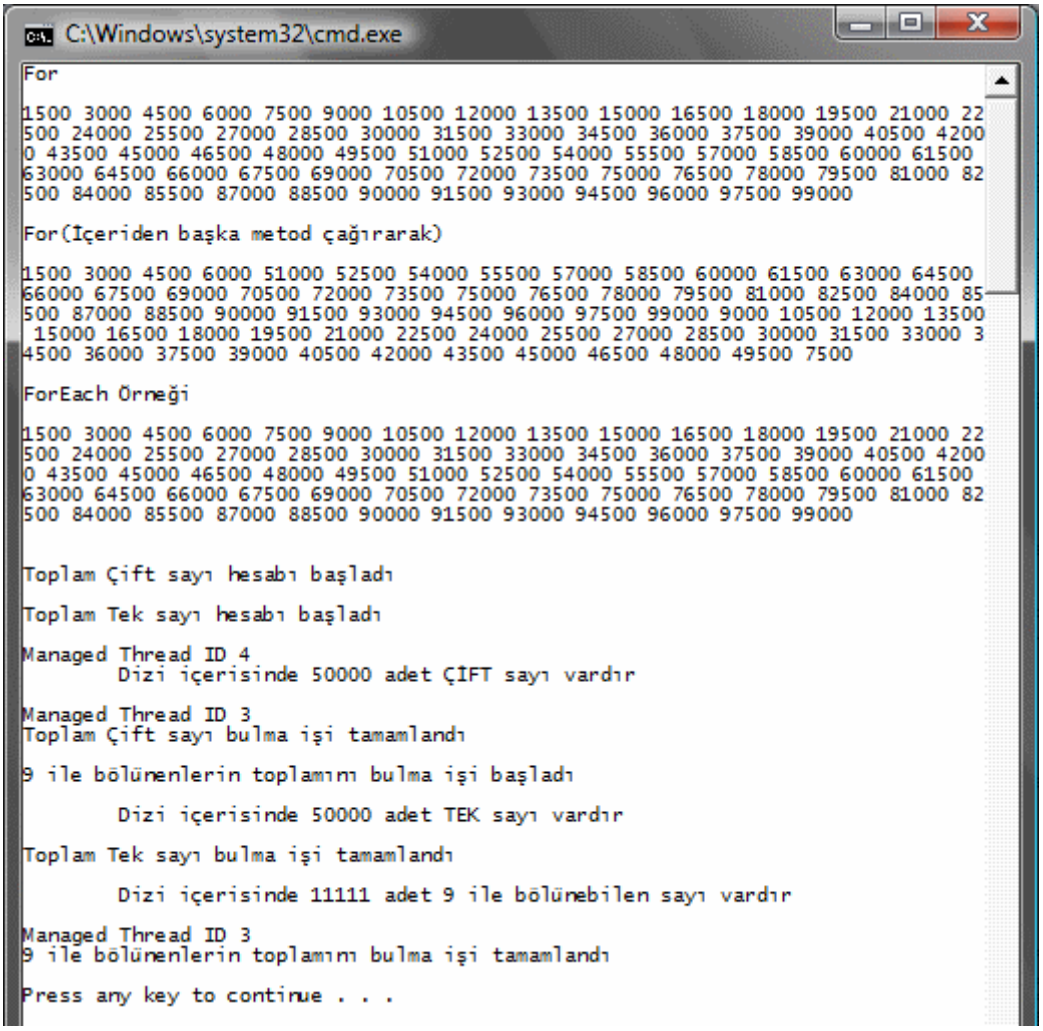
```
using System;
using System.Linq;
using System.Threading;

namespace HelloTPL
{
   class Program
   {
      static void Main(string[] args)
      {
         int[] numbers = Enumerable.Range(1, 100000).ToArray();
```

```
#region Parallel.For örneği

// Action temsilcisinin söylediği kurallara uygun olaraktan, lambda operatöründen yararlanılır.
Console.WriteLine("For\n");
Parallel.For(1, numbers.Length,
   i =>
   {
      if(i%1500==0)
         Console.Write("{0} ",i.ToString());
   }
);

Console.WriteLine("\n\nFor(İçeriden başka metod çağırarak)\n");
Parallel.For(1,numbers.Length,
   (i)=>{
      if (i % 1500 == 0)
         Task1(i);
   }
);

#endregion

#region Parallel.ForEach örneği

/* ForEach metodunun 19 aşırı yüklenmiş versiyonu vardır. İlk dikkati çeken nokta, IEnumerable<T> generic arayüzünü(interface) implemente eden referanslarıda parametre olarak almasıdır. Dolayısıyla, her koleksiyon veya diziye uygulanabilir. */
Console.WriteLine("\n\nForEach örneği\n");
Parallel.ForEach(numbers, number =>
{
   if (number % 1500 == 0)
      Console.Write("{0} ", number.ToString());
}
);

#endregion

#region Parallel.Invoke örneği

Console.WriteLine("\n\n");

// Parallel.Invoke metodu Action temsilcisi tipinden referanslar alan bir diziyi parametre olarak kullanır.
// Bu şekilde istenildiği kadar metodun paralel olarak başlatılması
```

```
sağlanabilir
        Parallel.Invoke(
            () =>
            {
                Console.WriteLine("Toplam Tek sayı hesabı başladı\n");
                Console.WriteLine("Managed Thread ID {0}
",Thread.CurrentThread.ManagedThreadId.ToString());
                OddCount(numbers);
                Console.WriteLine("Toplam Tek sayı bulma işi tamamlandı\n");
            },
            () =>
            {
                Console.WriteLine("Toplam çift sayı hesabı başladı\n");
                EvenCount(numbers);
                Console.WriteLine("Managed Thread ID {0} ",
Thread.CurrentThread.ManagedThreadId.ToString());
                Console.WriteLine("Toplam çift sayı bulma işi tamamlandı\n");
            }
            ,
            () =>
            {
                Console.WriteLine("9 ile bölünenlerin toplamını bulma işi başladı\n");
                NineCount(numbers);
                Console.WriteLine("Managed Thread ID {0} ",
Thread.CurrentThread.ManagedThreadId.ToString());
                Console.WriteLine("9 ile bölünenlerin toplamını bulma işi tamamlandı\n");
            }
        );

            #endregion
        }

        static void Task1(int number)
        {
            // Değişiklik işlemler
            Console.Write("{0} ", number.ToString());
        }

        static void EvenCount(int[] numbers)
        {
            int result = (from number in numbers
                          where number % 2 == 0
                          select number).Count();
            Console.WriteLine("\tDizi içerisinde {0} adet çİFT sayı
vardır\n",result.ToString());
```

```
        }
        static void OddCount(int[] numbers)
        {
            int result = (from number in numbers
                          where number % 2 != 0
                          select number).Count();
            Console.WriteLine("\tDizi içerisinde {0} adet TEK sayı vardır\n",
result.ToString());
        }

        static void NineCount(int[] numbers)
        {
            int result = (from number in numbers
                          where number % 9 == 0
                          select number).Count();
            Console.WriteLine("\tDizi içerisinde {0} adet 9 ile bölünebilen sayı vardır\n",
result.ToString());
        }
    }
}
```

Aslında kod son derece açıktır ancak dikkat edilmesi gereken noktalarda vardır. For, ForEach ve Invoke metodları, Action temsilcisini sıklıkla kullanmaktadır. Bunun en büyük nedeni, paralel işleme tabi tutulacak kod parçalarını taşıyan herhangibir metod veya bloğun kullanılabilmesini sağlamaktır. Bildiğiniz gibi **Action** temsilcisi, parametre almayan ve geriye döndürmeyen metodları işaret etmektedir. Diğer taraftan **Func** temsilcisinin kullanıldığı versiyonlarda bulunmaktadır. Yani geriye değer döndüren ve parametre alan metodlarında işin içerisinde katılması sağlanabilir. Elbette kullanımı kolaylaştırmak adına, **lambda operatörüde(=>)** ciddi şekilde ele alınmaktadır. Uygulamayı çalıştırdığımızda aşağıdaki ekran görüntüsünde yer alan sonuçlara benzer bir çıktı elde ederiz.



Tabiki kodu test ettiğimiz sistemin çekirdek veya işlemci sayısına göre, yada o anda çalışmakta olan programlara göre farklı sıralarda sonuçlar elde edilebilir. Ancak çalışan kod parçasında işlemlerin paralel yapıldığına dair pek çok iz vardır. Dikkatlice bakıldığında For, ForEach döngülerinin dizileri gerçekten paralel bir sırada değerlendirdiği ve işlemeri yaptığı ortadadır. Invoke metoduda benzer şekilde, çağırdığı 3 metodu mümkün mertebede paralel olarak başlatmıştır.

NOT : *TPL tarafında geliştiricinin* ***alt-seviye(Low-Level)*** *işlemlerle uğraşmasına gerek yoktur. Ancak bu durum görsel programlama tarafında,* ***Illegal Cross Thread Exception*** *gibi istisnaların olmayacağı anlamına gelmemelidir* 😕

Böylece geldik bir yazımızın daha sonuna. Bu yazımızda TPL alt yapısını en basit ve yalın haliyle tanımaya çalıştık. Elbetteki işimiz bitmedi. Bi dünya işimi var aslında 😎 Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HelloTPL.rar (25,96 kb)

.Net RIA Servisleri - Özel Doğrulama(Custom Validation) (2009-05-31T13:03:00)

.net ria services,silverlight,

Merhaba Arkadaşlar,

Bir önceki blog yazımızda, .Net RIA Servislerin kullanıldığı Silverlight uygulamalarında doğrulama(Validation) işlemlerinin nasıl yapılabileceğini incelemeye çalışmıştık. Bu yazımızda ise, **Range, Required, StringLength, RegularExpression** gibi built-in niteliklerle(attribute) gerçekleştirilen doğrulamalar haricinde kalan özel durumlar için nasıl ilerleyebileceğimizi araştıracağız. Konuyu adım adım irdelersek, aşağıdaki işlemleri yapmamız gerekmektedir.

- Sunucu uygulama tarafında(Web App) shared niteliği ile işaretlenmiş bir sınıf tasarlanır ve içerisine özel doğrulama operasyonları ilave edilir.
- özel doğrulamaların uygulanacağı sınıf veya üyelerine, CustomValidation niteliği yardımıyla geliştirilen Validator tipi bildirilir.

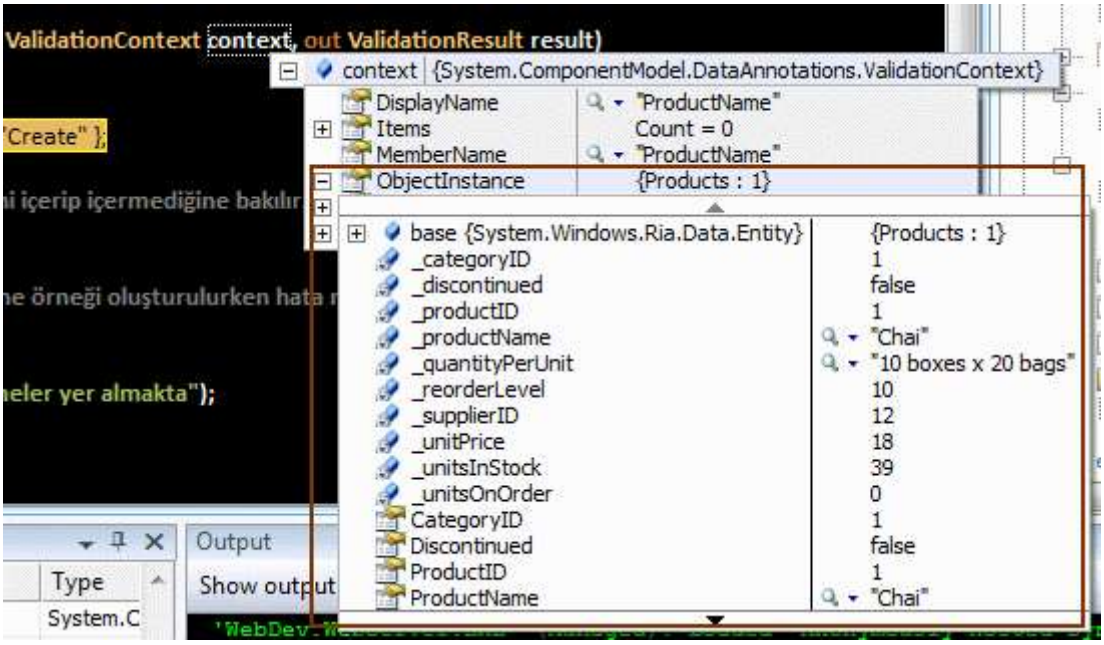
Gördüğünüz gibi gayet basit. önceki blog yazımızda geliştirdiğimiz örnek proje için bu adımları uygulamaya başlayabiliriz. örnek olarak **ProductName** alanı için özel bir doğrulama fonksiyonelliği geliştireceğiz. Bu doğrulamaya göre, ProductName ile ilişkili veri giriş alanı içerisinde **Select, Where, Delete** gibi **SQL** kelimelerinin olmamasını sağlamaya çalışacağız. Bu tabiki konunun anlaşılması için öne sürdüğümüz bir senaryo. Şu an için önemli olan, tekniğin nasıl uygulandığıdır. Bu amaçla web projesi tarafında**ProductNameValidator.shared.cs** isimli bir kod dosyası oluşturarak işe başlayabiliriz. Bu kod dosyasının adında **shared** kelimesinin eklenmesinin geliştirme ortamı(IDE) içinde özel bir anlamı vardır. **SınıfAdı.shared.cs/vb** formatında yazılan dosya adı sayesinde, istemci tarafı içinde otomatik kod üretiminin gerçekleştirilmesi sağlanmış olmaktadır.

Söz konusu sınıfın kod içeriği ise aşağıdaki gibidir.

```
using System.ComponentModel.DataAnnotations;
using System.Web.Ria.Data;

// Dosya adında shared kullanılmasının bir nedeni vardır. Bu isimlendirme standardı sayesinde, derleme zamanı alt yapısının istemci tarafı için otomatik dosya üretimi gerçekleştirmesi sağlanmış olunur.
namespace ValidationSystem.Web
{
   [Shared]
   public static class ProductNameValidator
   {
      public static bool QueryCheck(string productName, ValidationContext context, out ValidationResult result)
```

```
        {
            // sembolik olarak eklenmiş kontrol değerleri
            string[] keywords = { "Select", "Where", "Delete", "Create" };

            // ürün adının, yasaklı kelimelerden herhangibirini içerip içermediğine bakılır.
            foreach (string keyword in keywords)
            {
                // Bir tane bile içeriyorsa, ValidationResult nesne örneği oluşturulurken hata
mesajı bildirimi yapılır ve geriye false değer döndürülür
                if (productName.Contains(keyword))
                {
                    result = new ValidationResult("Tehlikeli kelimeler yer almakta");
                    return false;
                }
            }
            // Eğer doğrulama işlemi başarılıysa ValidationResult nesne örneğine null değer
atanır ve geriye true değer döndürülür
            result = null;
            return true;
        }
    }
}
```

Static olarak tanımlanan sınıfın **shared** niteliği ile imzalandığına dikkat edilmelidir. Diğer taraftan doğrulama işlemi için kullanılacak olan metod(metodlar), ilk parametre olarak doğrulanacak veri içeriğini taşıyabilecek tipte bir değişken kullanırlar. ProductName alanı tablo üzerinde **nvarchar** tipinden tanımlanmış ve bu nedenle **Entity** içerisinde **string** olarak ele alınmıştır. Dolayısıyla ilk parametrenin string tipinden tasarlanmış olması doğru bir tercihtir. Diğer taraftan ikinci parametre olarak **ValidationContext** ve üçüncü parametre olarakta**ValidationResult** tiplerinden değişkenler tanımlanmıştır. Her ne kadar örneğimizde **ValidationContext** parametresini kullanmamış olsakta, çalışma zamanında doğrulamaya tabi olan içeriğin sahibi tipe ait bilgileri içerdiğini söyleyebiliriz. Dolayısıyla bu değişken ile, doğrulamaya tabi olan ProductName değerine sahip Products nesne örneğine ulaşabilir ve doğrulamayı farklı açılardan ele alabiliriz. Aşağıdaki ekran görüntüsünde bu durum daha net bir şekilde görülebilmektedir.

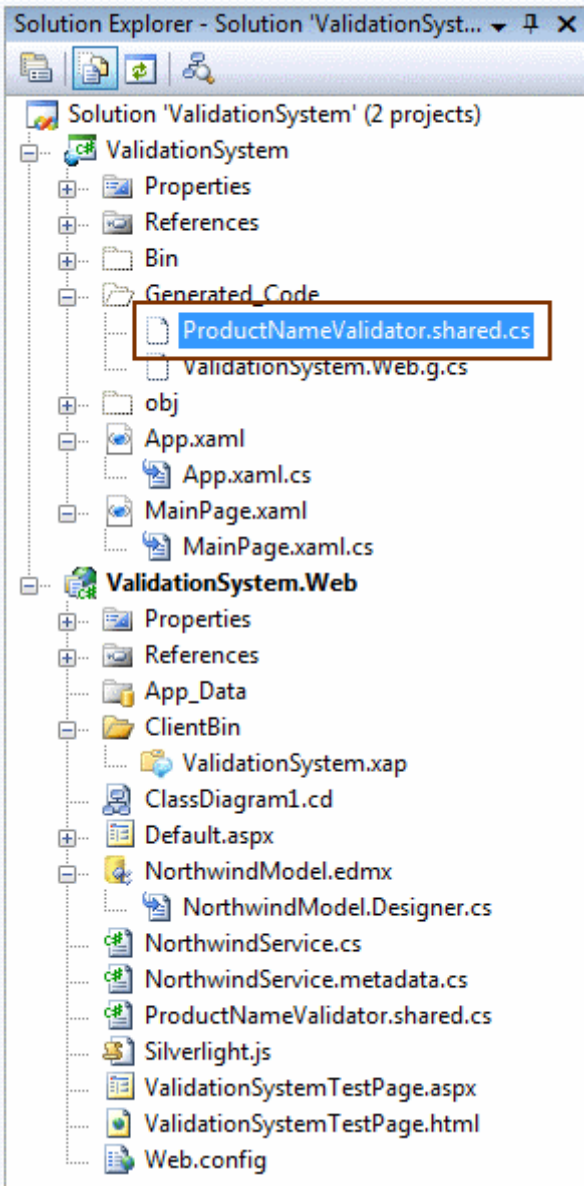


Gelelim **ValidationResult** tipine. Sonuç olarak doğrulamanın başarılı veya başarısız olma durumu söz konusudur. Başarısız olunması halinde, istemci tarafında hata mesajı gibi bilgileri içeren bir nesne örneğinin var olması gerekmektedir. İşte **ValidationResult** nesne örneğinin üretilmesi ile, doğrulamanın başarısız olması durumunda geriye nasıl bir bilgi döndürüleceği belirtilmektedir. Tabi metodun böyle bir durumda geriye **false** değer döndürmeside gerekmektedir. Elbetteki doğrulama işleminin başarılı olması halinde geriye **true** değer döndürülmesi ve ayrıca **ValidationResult** nesne örneğinin **null** olarak aktarılması sağlanmalıdır.

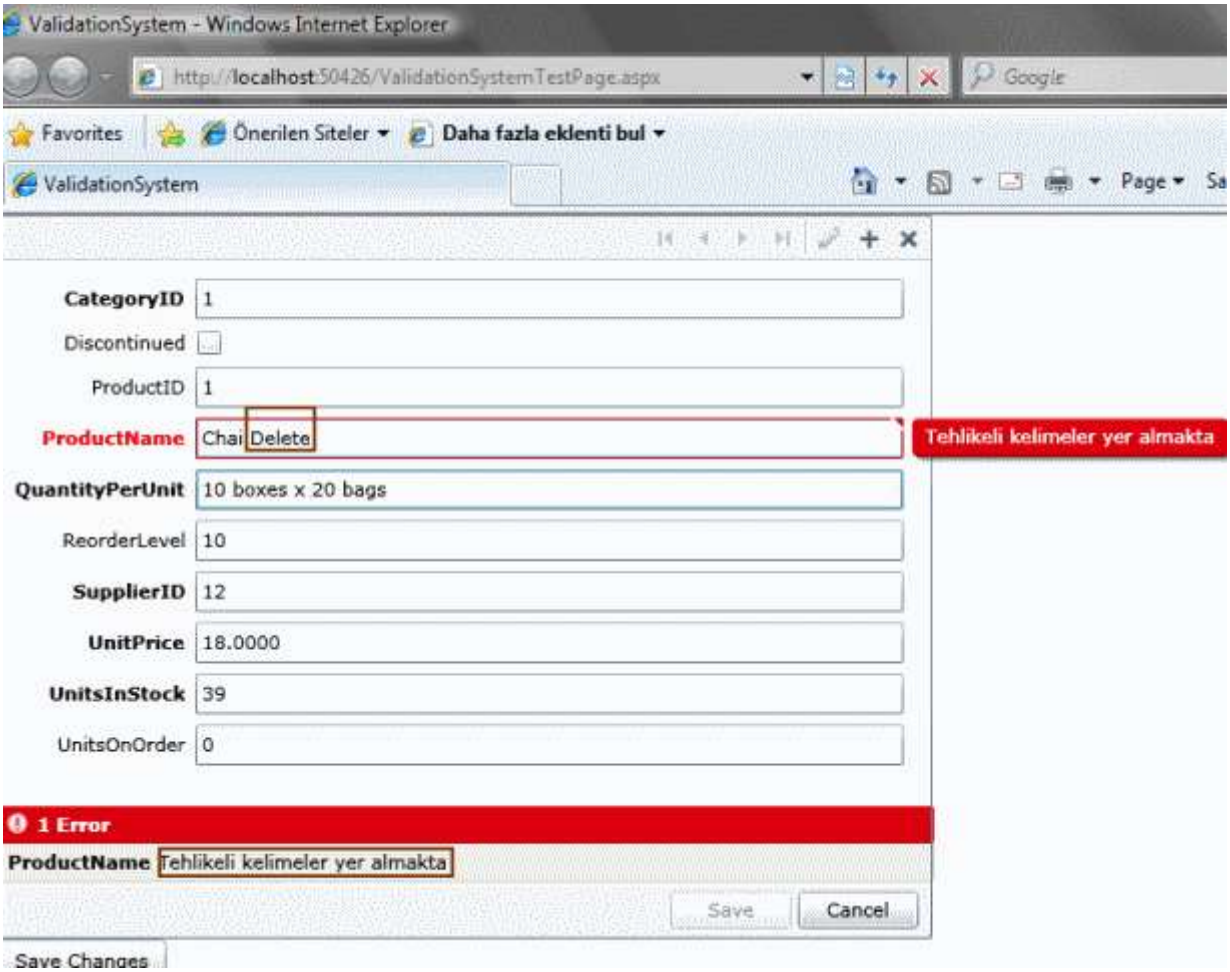
Sırada ikinci adım var. Geliştirilen bu doğrulama tipinin, çalışma zamanı tarafından ele alınması gerekmektedir. Tabiki hal böyle olunca devreye **niteliklerin(attribute)** girmeside kaçınılmazdır. Neyseki kendi niteliklerimizi yazmak yerine, herhangibir **validator** tipini, istediğimiz özellik veya sınıfa uygulamamızı sağlayan tek bir **built-in** nitelik mevcuttur. **CustomValidation**. Dolayısıyla metadata dosyası içerisinde, **ProductName** özelliğinin aşağıdaki hale getirilmesi yeterli olacaktır.

```
[Required(ErrorMessage="Lütfen ürün adını giriniz")]
[CustomValidation(typeof(ProductNameValidator),"QueryCheck")]
public string ProductName;
```

CustomValidation niteliği ilk parametre olarak doğrulama tipini almaktadır. İkinci parametrede ise, takip öden özelliğin(veya sınıfın) kontrolünü gerçekleştirecek olan metod adı belirtilmektedir. Uygulama bu son haliyle derlendiğinde, istemci projesindede aşağıdaki şekilde görülen ek dosyanında üretildiği gözlemlenebilir.



Artık uygulamayı test etmeye başlayabiliriz. Bu amaçla herhangibir ürünün güncellenmeye çalışıldığını düşünelim ve ürün adında **Delete** kelimesini kullandığımızı varsayalım. İşte sonuç...



Tataaaa!!! Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ValidationSystem2.rar (1,86 mb)

.Net RIA Servisleri - Doğrulama(Validation) (2009-05-30T12:01:00)

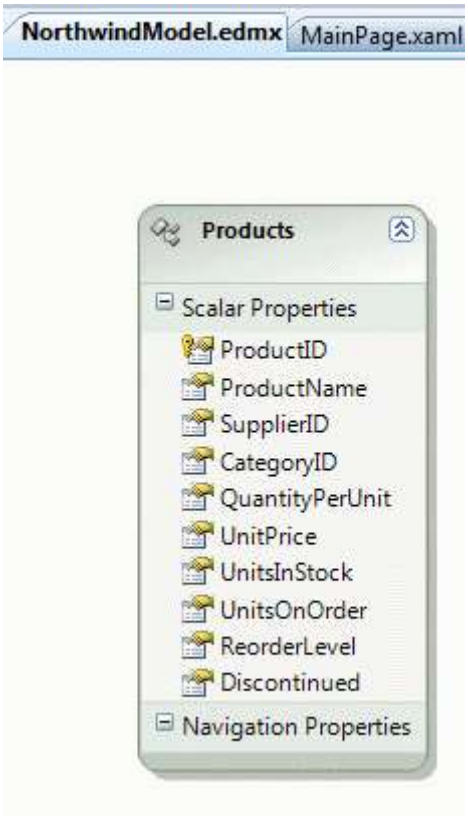
.net ria services,silverlight,

Merhaba Arkadaşlar,

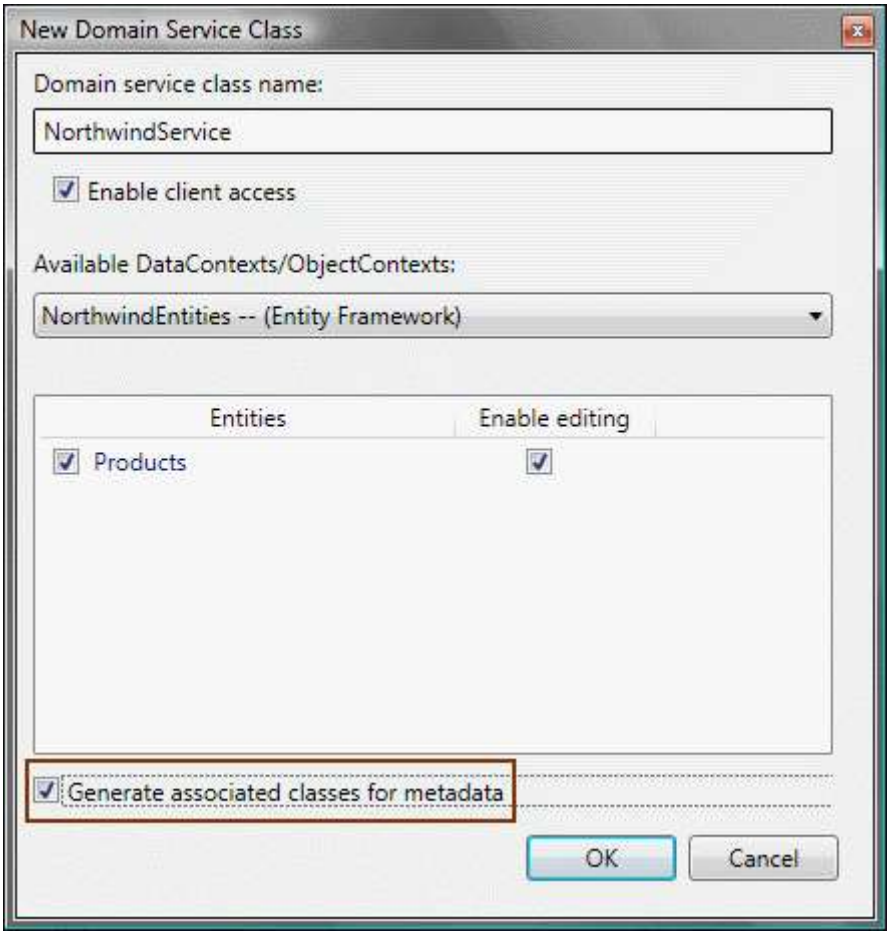
Bildiğiniz gibi bir süredir **PLINQ(Parallel Language INtegrated Query)** ile ilişkili araştırmalarıma devam etmekteyim. Nevarki dün gece uzun süredir ara verdiğim bir konuyu farkettim. **.Net RIA Servisleri**. Bunun üzerine yaz sıcaklarının kendini iyiden iyiye hissettirmeye başladığı şu günlerde serinlemek için deniz kenarında bir yerlere gitmeden önce,*(örneğin güzelim Ortaköy sahilinde denize karşı çay içmek gibi)* esinti ferahlığı verecek, hafif ve basit bir konuyu araştırmaya niyetlendim. Bunun üzerine, verinin eklenmesi veya güncellenmesi sırasında doğrulama işlemlerinin nasıl yapılabileceğini incelemeye karar verdim. özellikle **Asp.Net** veya **Windows Forms, WPF** gibi, kullanıcı ile etkileşimde olan arayüzlerin kullanıldığı projelerde, verinin çeşitli nedenler ile doğrulanması gerekebilir. Doğruluğu kanıtlanan veri, kaynağa eklenebilir,

güncelleştirilebilir. Tabiki verinin doğruluğunu kontrol etmek adına pek çok stratejiden faydalanılabilir. örneğin, Asp.Net web tabanlı uygulamalarda kullanılan doğrulama işlemleri, istemci tarafında **Javascript** savunması ile başlayıp sunucu tarafında devam eder. En büyük amaç, maliyeti yüksek olan veri işlemlerinden önce çeşitli kriterlerin sağlandığını kontrol etmek olarak düşünülebilir. Tabiki bu maliyet hesabına, güvenlik kontrolünüde eklediğimizde, doğrulamanın aslında son derece önemli bir cephe olduğu ortaya çıkmaktadır. Doğrulama kontrolleri, bir ürün fiyatının belirli aralıkta olması şeklinde düşünülebileceği gibi, bir kredi kartı numarasının **Lhun** algoritmasına uygun olup olmadığı gibi karmaşık bir denetim mekanizması olarak ta düşünülebilir. Peki **Silverlight, .Net RIA Services** ve üretilen **Entity** tipleri göz önüne alındığında söz konusu doğrulama işlemleri acaba nasıl yapılabilir?

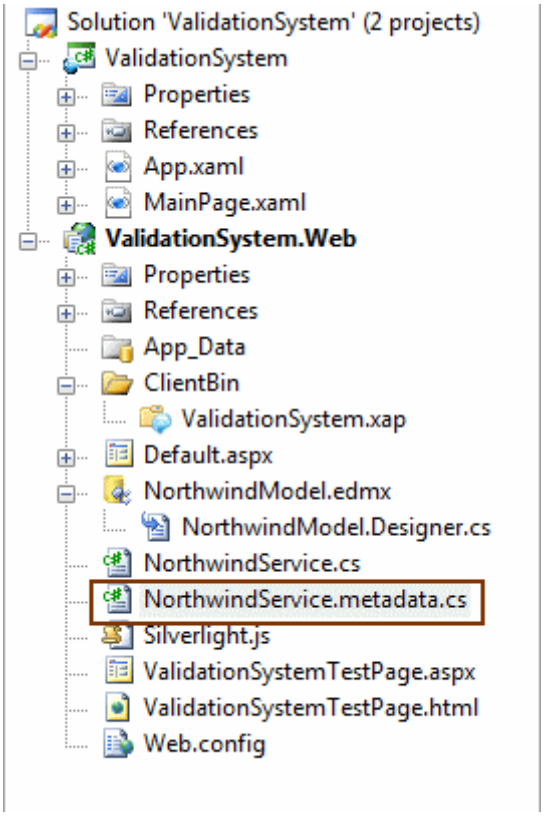
Dilerseniz konuyu basit bir örnek üzerinden ele almaya çalışalım. İşe biraz daha renk katmak adınada **Silverlight** tarafında **DataForm** veri kontrolünü kullanabiliriz. Nitekim bu kontrol özellikle doğrulama işlemleri sırasında olan hataları gayet hoş bir biçimde gösterebilmektedir. Tabi işe ilk olarak **Silverlight** projesini oluşturarak başlamak gerektiğini hepimiz biliyoruz. Söz konusu projede .Net RIA Servislerini ele alacağımızdan, **veri erişim katmanında(Data Access Layer) Ado.Net Entity Framework** veya **LINQ to SQL** modellerinden birisini kullanmayı tercih edebiliriz. Ben Ado.Net Entity Framework kullanmayı tercih ettim ve kobay tablo olarak Northwind veritabanında yer alan Products nesnesini seçtim. Buna göre oluşan EDM diagramı aşağıdaki gibidir.



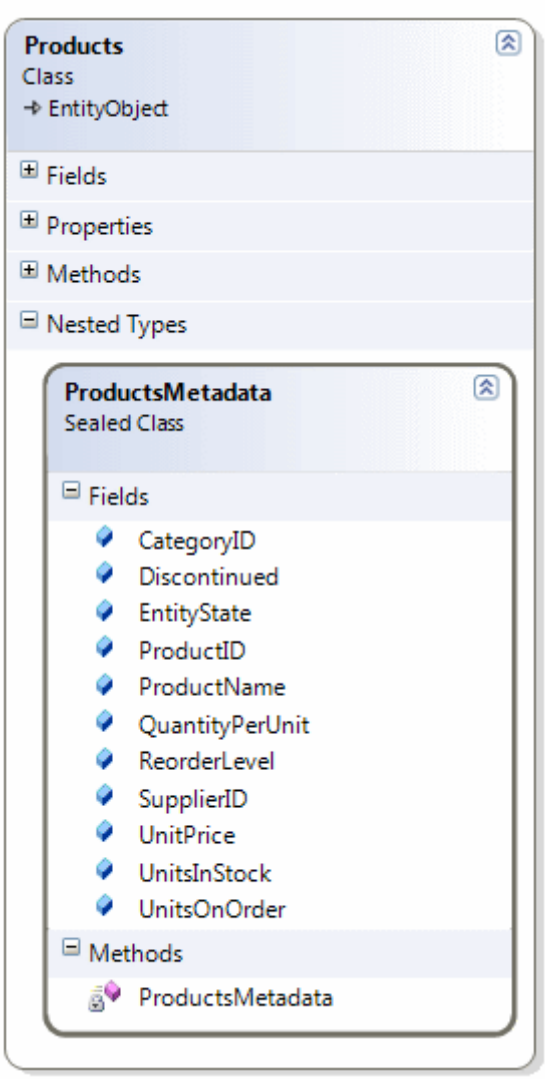
Tabiki bu işlemin ardından Web projesine, **Domain Service Class** öğesininde eklenmesi gerekmektedir. Bu şekilde .Net RIA Servisin, sunucu tarafındaki kısmı ve build işlemi sonrasındada istemci tarafındaki **Domain Context** parçası oluşturulmuş olacaktır. Ancak dikkat edilmesi gereken önemli bir nokta vardır.



Şekildende görüldüğü gibi **Domain Service Class** öğesinin eklenmesi sırasında **Generate associated classes for metadata** özelliği etkinleştirilmiştir. Bu sayede Web projesine, NorthwindService.metadata.cs isimli bir dosyanın daha eklendiği görülür.



Bu metadata dosyası içerisinde ise, **Products** isimli **Entity** sınıfı için üretilmiş partial bir tip daha olduğu görülür. Asıl önemli olan bu partial tip içerisinde, **ProductsMetadata ({EnitiyName}Metadata)** isimli **internal** erişim belirleyicisine sahip(yani sadece bulunduğu assembly içerisinde kullanılabilen) bir sınıf daha tanımlanmış olmasıdır. Bu sınıf aynı zamanda sealed bir tiptir. Bir başka deyişle kendisinden türetilme yapılamamaktadır.



Peki bu metadata sınıfını neden ürettirdik? Ne işe yaramaktadır?

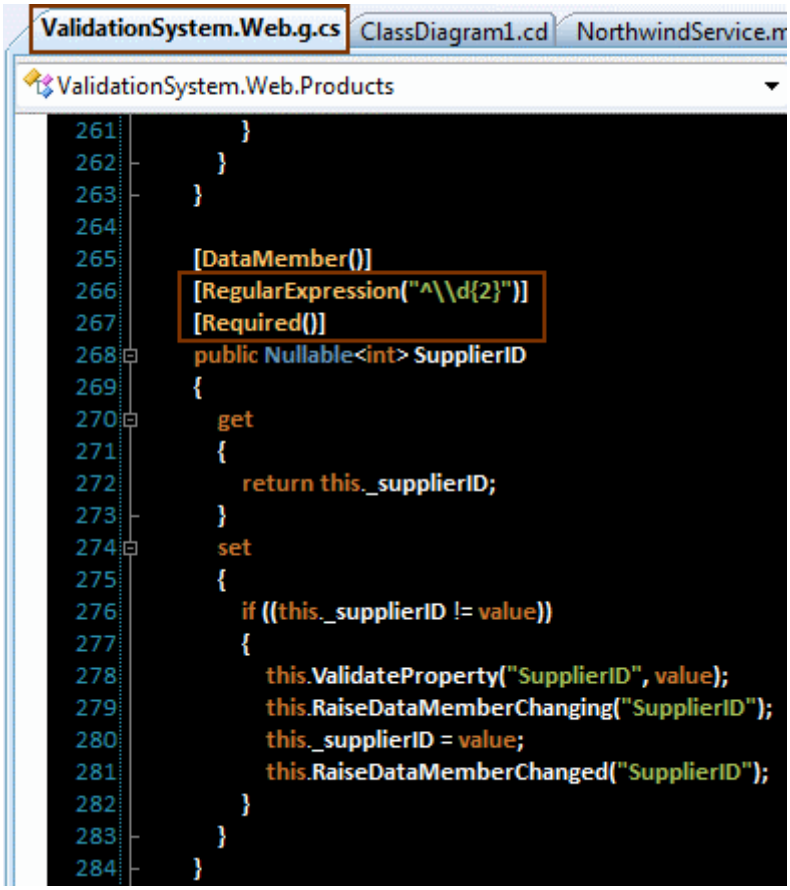
Senaryomuzda dikkat edileceği üzere **Ado.Net Entity Framework** modeli kullanılmaktadır. Bu model, veritabanından seçilen nesnelerin karşılığı olan sınıfların otomatik üretilmesinide içermektedir. Dolayısıyla **Products** tablosunun karşılığı olan **Entity** tipi otomatik olarak üretilen bir **sınıftır** aslında. Aynı durumu **LINQ to SQL** tarafı içinde geçerlidir. çok doğal olarak üretilen **Entity** tipi, modele has nitelikler ile süslenmiştir. örneğin, özelliğin identity olup olmadığı, hangi entity ile ilişkili olduğu vb... Oysaki doğrulama gibi, ek nitelikler yardımıyla gerçekleştirmek isteyebileceğimiz işlemler, **Entity'** ye değil, **.Net RIA Servis** tarafına özeldir. Bu nedenle var olan **Entity** yapısının bozulması istenmediğinden ve hatta otomatik üretimler sonucu eski haline dönmesi ihtimalide bulunduğundan, ek metadata girişlerinin güvenli bir şekilde yapılması için .Net RIA Servis tarafında ayrı bir sınıf daha üretilmektedir. Yani Entity ile ilişkili yapmak istediğimiz ek işlemleri bu metadata sınıfı içerisinde toplayabiliriz. Dolayısıyla doğrulama işlemleri ile ilgili nitelik tanımlamalarının yapılacağı en uygun yerin, üretilen metadata sınıfları olduğunu söyleyebiliriz. Bizde üretilen bu metadata sınıfı üzerinde aşağıdaki değişiklikleri yapabiliriz.

```
namespace ValidationSystem.Web
{
   using System;
   using System.Collections.Generic;
   using System.ComponentModel;
   using System.ComponentModel.DataAnnotations;
   using System.Linq;
   using System.Web.Ria;
   using System.Web.Ria.Data;
   using System.Web.DomainServices;
   using System.Data;

   [MetadataTypeAttribute(typeof(Products.ProductsMetadata))]
   public partial class Products
   {

#pragma warning disable 649

      internal sealed class ProductsMetadata
      {
         private ProductsMetadata()
         {
         }

         public int ProductID;

         [Required(ErrorMessage="Lütfen ürün adını giriniz")] // Mutlaka
girilmeli(boş geçilemez)
         public string ProductName;

         [Required] // Mutlaka gerekli
         [RegularExpression("^\\d{2}")]  // İki hanel sayısal değer
         public Nullable<int> SupplierID;

         [Required]
         public Nullable<int> CategoryID;

         [StringLength(20,MinimumLength=3)] // minimum 3 karakter maksimum 20
karakter
         [Required]
         public string QuantityPerUnit;
```

```
        [Range(0,300)]
        [Required]
        public Nullable<Decimal> UnitPrice;

        [Range(0,1000)]
        [Required]
        public Nullable<short> UnitsInStock;

        public Nullable<short> UnitsOnOrder;

        public Nullable<short> ReorderLevel;

        public bool Discontinued;

        public EntityState EntityState;
    }

#pragma warning restore 649
  }
}
```

- Required niteliği tahmin edileceği üzere, söz konusu alanın mutlaka girilmesi gerektiğini, bir başka deyişle boş geçilemeyeceğini belirtmektedir.
- Range niteliği ile sayısal değer aralığı belirtilir.
- StringLength niteliği ise, minimum ve maksimum karaketer aralığını belirtir.
- RegularExpression niteliğinde ise, basit bir RegEx ifadesi kullanılarak, verinin hangi formatta olması gerektiği belirtilir.

Dikkat edileceği üzere bu niteliklerin benzer özellikleri vardır.
örneğin **ErrorMessage** özelliklerine atanan değerler ile, hata sonrası gösterilecek mesajın içeriği belirlenebilir. Yapılan bu ilaveler sırasında
kullanılan **niteliklerin(attributes),** istemci tarafında üretilen otomatik **cs** dosyası içerisinde yer alan **Products** sınıfınada aktarıldığı gözlemlenecektir.



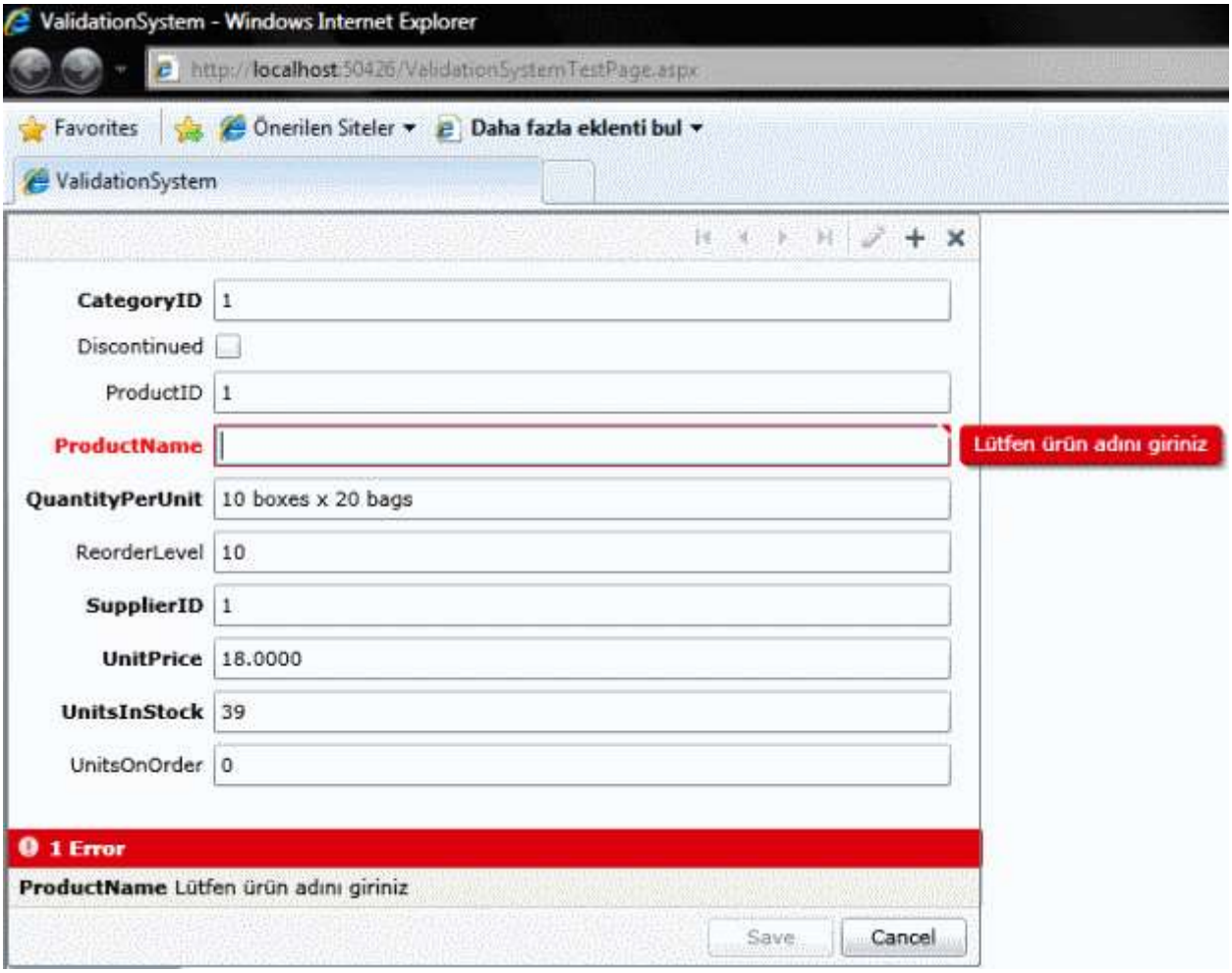
Artık testleri yapmak üzere XAML tarafındaki geliştirmelere başlayabiliriz. Baştada belirttiğimiz gibi **DataForm** isimli veri bağlı kontrolden yararlanıyor olacağız. Bu kontrol kendi içerisindeki Field' larda verinin nasıl gösterileceğini tanımlayabilmemizede olanak sağlamaktadır. Ayrıca, doğrulama işlemeleri sırasındaki hata mesajlarınıda güzel bir şekilde göstermektedir. İşte MainPage.xaml içeriğimiz;

```
<UserControl xmlns:dataControls="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.DataFor
m"  x:Class="ValidationSystem.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   Width="550" Height="455">
   <StackPanel x:Name="LayoutRoot" Background="White" Orientation="Vertical">
      <dataControls:DataForm x:Name="dfProducts" Width="550"
Height="425"></dataControls:DataForm>
      <Button x:Name="btnSaveChanges" Content="Save Changes" Width="100"
HorizontalAlignment="Left" Height="25" Click="btnSaveChanges_Click"/>
   </StackPanel>
</UserControl>
```

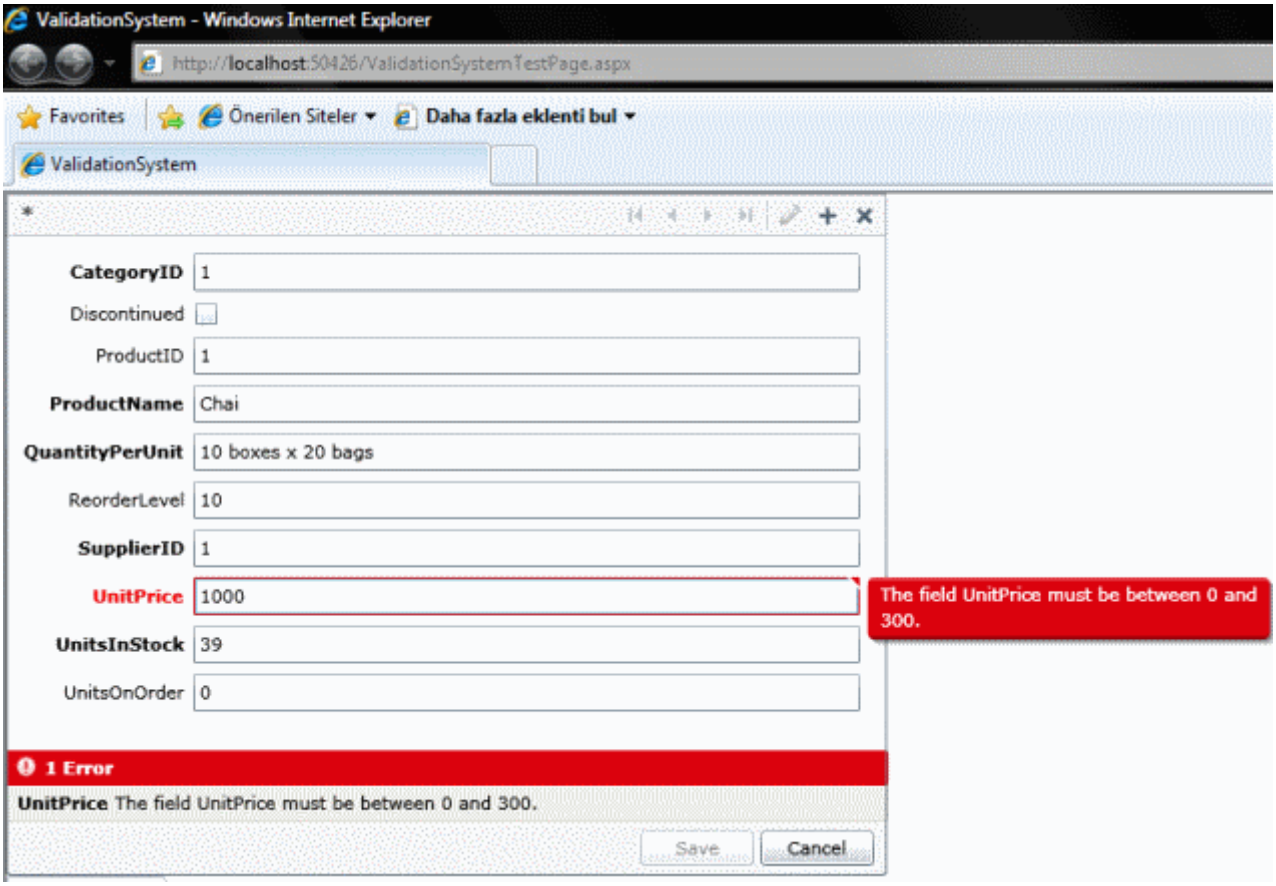
Kod tarafını ise aşağıdaki gibi geliştirmemiz yeterlidir.

```
using System.Windows;
using System.Windows.Controls;
using ValidationSystem.Web;

namespace ValidationSystem
{
   public partial class MainPage : UserControl
   {
      // DomainContext nesnesi tanımlanır
      NorthwindContext context = null;

      public MainPage()
      {
         InitializeComponent();
         // DomainContext nesnesi örneklenir
         context = new NorthwindContext();
         // Veri kaynağı olarak Products özelliğinin işaret ettiği referans belirtilir
         dfProducts.ItemsSource = context.Products;
         // ürünler yüklenir
         context.LoadProducts();
      }

      private void btnSaveChanges_Click(object sender, RoutedEventArgs e)
      {
         if(dfProducts.Mode== DataFormMode.Display)  //Edit veya Inserted moddayken
SubmitChanges metodunun verebileceği olası exception' lara karşı alınan geçici tedbirdir
            context.SubmitChanges();        // Değişiklikler sunucu tarafına gönderilir
      }
   }
}
```

Uygulamayı **Start without debugging modda** çalıştırarak testlerimize başlayalım. **DataForm** kontrolü verileri eğer aksi belirtilmediyse, tipe göre **TextBox** veya **CheckBox** gibi kontrollerde göstermektedir. **Domain Service** öğesi oluşturulurken **Enable Editing** seçeneği işaretlendiğinden, kontrolün sağ üst köşesinde ekleme, düzenleme ve silme işlemleri için gerekli düğmelerinde otomatik olarak üretildiği görülebilir. Bizim ağırlıklı olarak üzerinde duracağımız kısım doğrulam işlemleridir. örneğin bir ürünü düzenleme moduna aldığımız varsayalım. Bu durumda, doğrulam işlemlerine tabi tutulan alanlara ait **TextBlock** kontrollerinin **Bold** olarak işaretlendiği fark edilecektir. Buda kullanıcıya hangi kontrollerin doğrulama denetimi altında olduğunu işaret eden bir ipucu olarak düşünülebilir. örnek olarak seçilen ürünün adını boş geçmeye çalıştığımızı var sayalım.**ProductName** alanı ile ilişkili kontrolü terk ettiğimiz anda(başka bir kontrole geçerek), aşağıdaki şekilde görüldüğü gibi ekranın kırmızı fontlu yazılar ile dolduğunu görebiliriz.



Harika. Burada dikkat edilmesi gereken noktalardan birisi hata mesajıdır. Dikkat edileceği üzere, ilgili niteliğin ErrorMessage özelliğine atanan değer gösterilmektedir. Peki ErrorMessage belirtmediklerimizde durum nasıldır? örneğin UnitPrice değerini 1000 olarak girdiğimizi varsayalım. Range niteliğinde 0 ile 300 arasında bir değer olabileceğini belirtmiştik. Aşağıdaki durum ile karşılaşırız.



Görüldüğü gibi, built-in tasarlanmış standart bir hata mesajı üretilmektedir. Dolayısıyla doğrulama işlemlerini uyguladığımız senaryolarda **ErrorMessage** özelliğine bir değer atamamız, kullanıcıyı daha anlaşılır bir şekilde bilgilendirmek adına tercih edilmelidir. İyi güzel her şey hoş ama Asp.Net tarafından biliyoruzki, istersek standart doğrulama işlemleri dışındaki ihityaçlarda **CustomValidator** bileşeninden yararlanabilmekteyiz. Tabi bu durumda, istemci tarafı için gerekli **Javascript** içeriğini ve sunucu tarafındaki olay metodu kodlarınıda geliştirmemiz gerekmektedir.

Peki .Net RIA Servislerinin kullanıldığı senaryoda, özel doğrulama işlemleri nasıl ele alınabilir? Bu özel doğrulama işlemleri sınıf seviyesinde veya özellik seviyesinde nasıl uygulanabilir? Bir sonraki blog yazımda bu konulara değinmeye çalışıyor olacağım. Tekrardan görüşünceye dek hepinize mutlu bir yaz günü dilerim.

ValidationSystem.rar (1,85 mb)

PLINQ - ForAll [Beta 1] (2009-05-28T17:43:00)

plinq,linq,

Merhaba Arkadaşlar,

Bildiğiniz gibi bir süredir LINQ sorgularının paralel çalıştırılması ile ilişkili çalışmalarıma ve araştırmalarıma devam etmekteyim. Bu yazımdaki konumuz ise **System.Linq.ParallelEnumerable static** sınıfı içerisinde tanımlanmış olan **ForAll genişletme metodudur(extension methods).**

public static void ForAll<TSource>(this ParallelQuery<TSource> source, Action<TSource> action);

ForAll metodu yukarıdaki prototipinden de görüldüğü gibi **ParallelQuery** referanslarına uygulanabilmektedir. Bununla birlikte metod ikinci parametre olarak, **Action<TSource>** tipinden generic bir temsilci almaktadır.

public delegate void Action<in T>(T obj);

Yukarıdaki prototipe göreyse, **Action<T>** temsilcisi(delegate), generic tip olarak ForAll metoduna gelen tipi**(TSource)** kullanmaktadır. Bu generic tip tahmin edeceğiniz üzere **ParalleQuery** referansınında kaynak tipidir. Ayrıca temsilci geriye herhangibir **değer döndürmeyen(void)** metodları işaret edebilmektedir.

Sonuç olarak ForAll metodu aslında, **AsParallel** metodunun kullanılması sonucu üretilen referans üzerinden gelen her bir nesne örneği için yapılması istenen işlemleri ele almaktadır. Bu açıdan bakıldığında akla gelen soru şu olacaktır.

*Paralel sorguların çalışması sonucu üretilen çıktılar üzerinde **foreach** döngüleri yardımıyla da dolaşabiliyorken, **ForAll** metodunu neden kullanırız?*

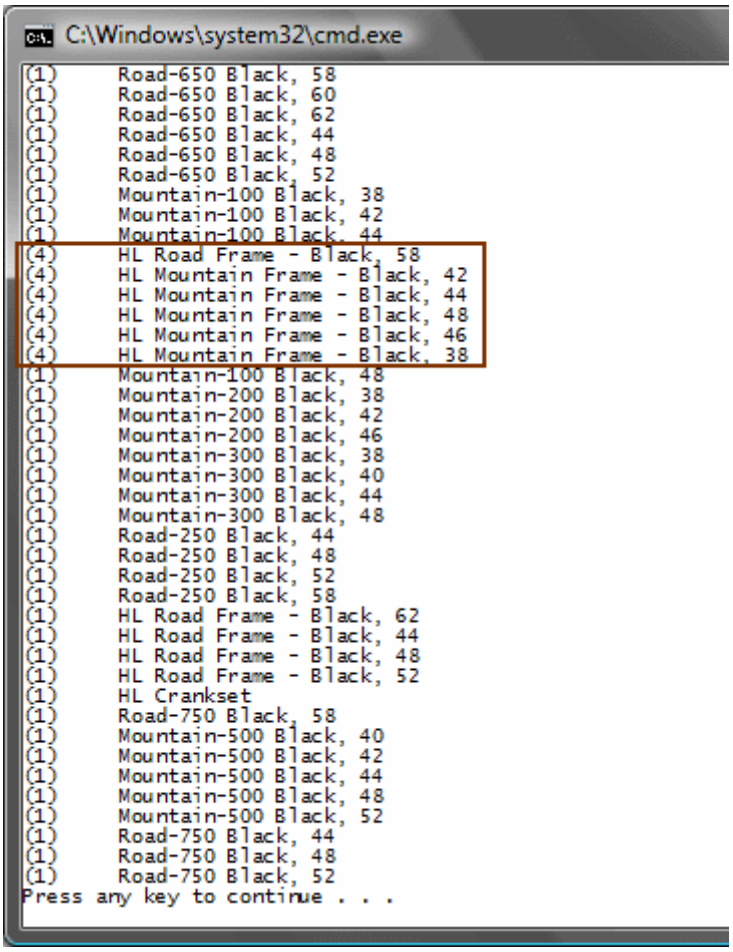
Aşağıdaki kod parçasını göz önüne alalım.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;
using System.Threading;

namespace UsingForAll
{
   class Program
   {
      static void Main(string[] args)
      {
         List<Product> productList = GetProductList();

         var result = from p in
productList.AsParallel()//.WithExecutionMode(ParallelExecutionMode.ForceParallelism)
```

```
                where p.ListPrice>=400 && p.Color=="Black"
                select p;

        result.ForAll(p=>Console.WriteLine("("+Thread.CurrentThread.ManagedThreadId.ToString()+")\t"+p.Name));
    }

    static List<Product> GetProductList()
    {
        List<Product> productList = new List<Product>();

        SqlConnection conn = new SqlConnection("data source=Manchester;database=AdventureWorks2008;integrated security=true");
        SqlCommand cmd = new SqlCommand("Select ProductId,Name,ListPrice,ProductNumber,Color,SafetyStockLevel From Production.Product", conn);
        conn.Open();
        SqlDataReader reader = cmd.ExecuteReader(CommandBehavior.CloseConnection);
        while (reader.Read())
        {
            productList.Add(new Product
            {
                ProductId = Convert.ToInt32(reader[0]),
                Name = reader[1].ToString(),
                ListPrice = Convert.ToDecimal(reader[2]),
                ProductNumber = reader[3].ToString(),
                Color = reader[4].ToString(),
                SafetyStockLevel = Convert.ToInt32(reader[5])
            }
            );
        }
        reader.Close();
        return productList;
    }
}

class Product
{
    public int ProductId { get; set; }
    public string Name { get; set; }
    public decimal ListPrice { get; set; }
    public string ProductNumber { get; set; }
    public string Color { get; set; }
    public int SafetyStockLevel { get; set; }
```

```
    }
}
```

Bu kod parçasında odaklanmamız gereken nokta **result** referansı üzerinden **ForAll** metodunun çağırılışıdır. Bu çağrı sırasında **labmda operatöründen(=>)** yaralanılmaktadır ve bulunan ürünlerin adları ile o an çalışmakta olan **Thread**' in numarası(**ManagedThreadId**) **Console** ekranına yazdırılmaktadır. Sonuç aşağıdakine benzer olacaktır.

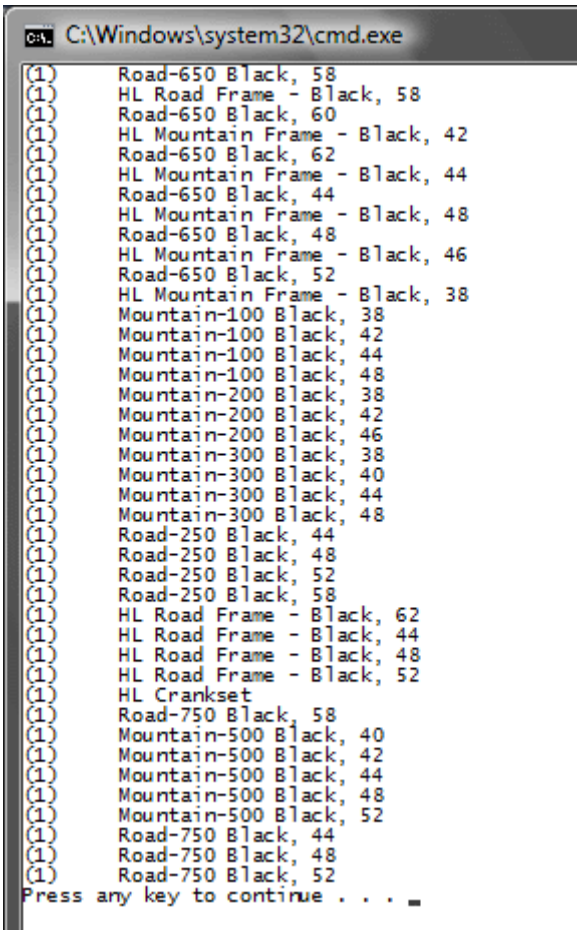


Her ne kadar **Thread** sayıları eşit olmasada **4** ve **1** nolu iki ayrı iş parçasının çalıştırıldığı görülmektedir. Şimdi aynı sorgu sonuçlarını foreach döngüsü yardımıyla elde etmeye çalıştığımızı düşünelim.

```
List<Product> productList = GetProductList();

var result = from p in
productList.AsParallel()//.WithExecutionMode(ParallelExecutionMode.ForceParallelism)
             where p.ListPrice>=400 && p.Color=="Black"
             select p;
```

```
foreach (Product p in result)
{
  Console.WriteLine("(" + Thread.CurrentThread.ManagedThreadId.ToString() +
")\t" + p.Name);
}
```

Bu kez uygulama çalıştırıldığında aşağıdaki sonuçları alırız.

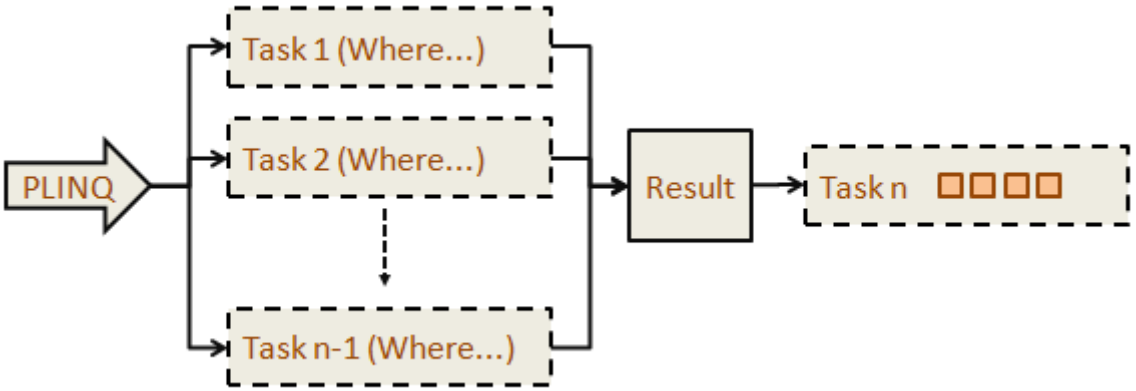


Volaaa!!! 😎 1 numaralı sadece tek bir thread görünüyor.

Bu nasıl oldu? Acaba **foreach** döngüsü kullanıldığında sorgu **AsParallel** metodu olmasına rağmen paralel çalıştırılmadı mı? Yoksa çalışma zamanı(runtime) sorgunun paralel çalıştırılmaya değer olmadığına mı kanaat getirdi(ki böyle bir meselede var)?

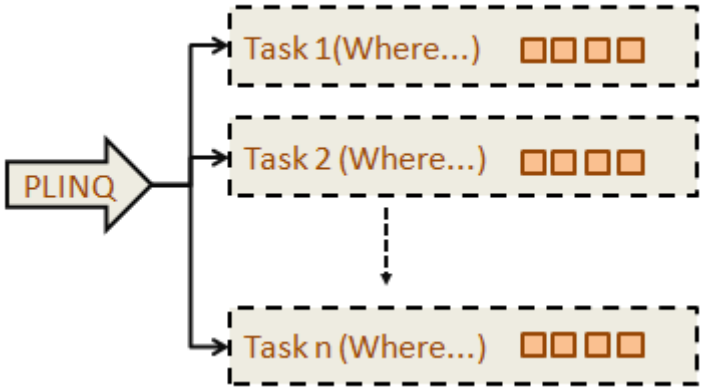
Aslında farklı çalışmanın sebebi şu. LINQ sorguları bilindiği gibi kullanıldıkları yerde çalıştırılırlar(**deferred execution ilkesi**). Bu nedenle sorgunun çalıştırılması foreach döngüsünde ilk eleman elde edilmeye çalışıldığı sırada olur. Lakin sorgu **AsParallel** metodu nedeniyle paralel çalışmasına rağmen, foreach metodu okuma işlemine başlamadan önce tüm yönetimli thread' leri tekrardan tek bir thread içerisinde birleştirir. Yani foreach döngüsünün kendisi paralel çalışma özelliğne sahip değildir. Bu nedenle paralel çalıştırılan sorgu sonuçlarını, o an üzerinde çalıştığı thread' de birleştirmeden ilerleyemez. **ForAll** metodu ise tam aksine çalışmakta olup, okuma

işlemlerininde paralel yürütülmesini sağlamaktadır. Aslında durumu basit iki resim ile canlandırmaya çalışalım. Aşağıdaki şekilde foreach çalışması sırasındaki işleyiş sembolize edilmektedir.



Buna göre sorgu paralel olarak çalışan görevlere ayrılmakta ve herbir görev içerisinde where gibi koşullar ele alınmaktadır. Ancak tüm PLINQ ifadesi tamamlandığında sonuçlar tek bir Task altında birleştirilmektedir. (Kahverengi çerçeveli kutucuklar bulunan nesne örnekleri üzerinden yapılan işlemleri sembolize etmektedir. örneğin Console.WriteLine gibi) Sonrasında ise herbir öğe için foreach döngüsü içerisinde yazılan kodlar işletilmektedir.

Aşağıdaki şekilde ise ForAll kullanımı sırasındaki senaryo ifade edilmeye çalışılmaktadır.



Yine PLINQ ifadesinin çalışması sırasında n adet Task paralel olarak başlatılır. Ancak foreach' ten farklı olarak her task' in içerisinde hem Where gibi koşulların kontrolü ele alınmakta hemde örneğimizde ki her bir sonuç için ayrı ayrı işlemler(Console.WriteLine gibi) gerçekleştirilmektedir. Yani task' ler paralel olarak işledikten sonra tek bir Task altında birleştirilmezler. Sanıyorum şekil yardımıyla sizde benim gibi, gerçekleşen iki farklı işleyişi daha net canlandırabildiniz. *(Tabi işlemcinin içerisine girip olan biteni canlı canlı görmemiz mümkün değil. Ama kim bilir, belki gelecek nesil sistemlerde çalışma*

zamanını, tıpkı bir doktorun sanal bir hastanın organları içerisinde ilerleyişi gibi, bilgisayar donanımı üzerindem gözlemleyebiliriz.)

Herşey güzel ama, hangisini ne zaman kullanmak gerekir öyleyse?

Aslında foreach döngüsünü daha çok sorgu sonuçlarının **sırasını(order)** korumak istediğimiz durumlarda değerlendirebiliriz. Bununla birlikte, sonuç listesi üzerinde ardışıl olarak işlemler yapmak istiyorsakta tercih edebiliriz.

Böylece geldik kısa bir yazımızın daha sonuna. Bir sonraki yazımızda görüşünceye dek hepinize mutlu günler dilerim.

UsingForAll.rar (22,17 kb)

Paralel Sorgularda İstisna Yönetimi(Exception Handling) [Beta 1] (2009-05-26T17:30:00)

plinq,linq,

Merhaba Arkadaşlar,

Yönetimli kod(Managed Code) tarafında istisna yönetimi oldukça önemli konulardan birisidir. Uygulamaların veya kod süreçlerinin istem dışı sonlanmasının önüne geçilmek istendiği durumlarda, basit **try...catch...finally** bloklarından yararlanabilir yada **Enterprise Library** gibi kütüphanelerin sunduğu bloklardan faydalanarak istisna yönetimini üst seviyede sağlayabiliriz.

Bu yazımda çok geniş kapsamda düşünmeyip, **PLINQ(Parallel Language INtegrated Query)** ifadelerinde oluşabilecek istisnai durumların nasıl ele alınması gerektiği üzerinde durmaya çalışacağız. Olaya hızlı bir giriş yapıp aşağıdaki örnek kod parçasına sahip olduğumuzu düşünelim.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;

namespace SequentialPLINQ
{
   class Program
   {
      static void Main(string[] args)
      {
         List<Product> productList = GetProductList();
```

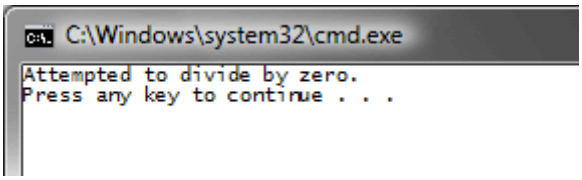
```
        productList[497].SafetyStockLevel = 0;
        productList[503].SafetyStockLevel = 0;

        var result1 = from product in productList.AsParallel()
                      orderby product.ProductId
                      where product.ListPrice>=500
                      select new
                      {
                         product.ProductId,
                         product.Name,
                         product.ListPrice,
                         product.Color,
                         SellPrice=FindSellPrice(product.ListPrice,product.SafetyStockLeve
l)    // İlk durum
                      };


        try
        {
           foreach (var r in result1)
           {
              Console.WriteLine(r.ProductId + " " + r.Name + " (" +
r.SellPrice.ToString("C2") + ")");
           }
        }
        catch (AggregateException excp) // Hata oluştuğunda PLINQ içerisinde başlatılan
tüm alt işlemler(Threads) iptal edilir
        {
           // PLINQ ifadeleri çalıştırıldığı sırada oluşan istisnalar AggreateException nesne
örneği içerisindeki InnerExceptions özelliğinin refere ettiği koleksiyonda toplanırlar.
           foreach (Exception error in excp.InnerExceptions)
           {
              Console.WriteLine(error.Message);
           }
        }
     }
       static decimal FindSellPrice(decimal listPrice,int stockLevel)
     {
        decimal result=-1;
           if (DateTime.Now.Day >= 25
              && DateTime.Now.Day <= 28)
              result=listPrice - (listPrice * (1 / stockLevel)); // SafetyStockLevel' ın 0
gelmesi halinde exception oluşacak olan yer.
           else
              result= listPrice * 1.18M;
```

```
            return result;
        }

        static List<Product> GetProductList()
        {
            List<Product> productList = new List<Product>();

            SqlConnection conn = new SqlConnection("data
source=Manchester;database=AdventureWorks2008;integrated security=true");
            SqlCommand cmd = new SqlCommand("Select
ProductId,Name,ListPrice,ProductNumber,Color,SafetyStockLevel From
Production.Product", conn);
            conn.Open();
            SqlDataReader reader =
cmd.ExecuteReader(CommandBehavior.CloseConnection);
            while (reader.Read())
            {
                productList.Add(new Product
                {
                    ProductId = Convert.ToInt32(reader[0]),
                    Name = reader[1].ToString(),
                    ListPrice = Convert.ToDecimal(reader[2]),
                    ProductNumber = reader[3].ToString(),
                    Color = reader[4].ToString(),
                    SafetyStockLevel = Convert.ToInt32(reader[5])
                }
                );
            }
            reader.Close();
            return productList;
        }
    }

    class Product
    {
        public int ProductId { get; set; }
        public string Name { get; set; }
        public decimal ListPrice { get; set; }
        public string ProductNumber { get; set; }
        public string Color { get; set; }
        public int SafetyStockLevel { get; set; }
    }
}
```

Visual Studio 2010 Professional Beta 1 ile geliştirilen bu kod parçasında, **Product** isimli bir sınıftan yararlanılmaktadır. Product tipine ait veriler, SQL sunucusu üzerindeki Product tablosundan alındıktan sonra, generic **List<Product>** koleksiyonu içerisinde tutulmakta ve sonrasında ise paralel sorgulamaya tabi tutulmaktadır. Burada ayrıca dikkat edilmesi gereken bir noktada, sorgulama sırasında**FindSellPrice** isimli metodun çağırılması ve parametre olarak, sorgunun t anındaki **Product** nesnesine ait **ListPrice** ile **StockLevel** değerlerinin gönderilmesidir. Dikkat edileceği üzere**FindSellPrice** metodu içerisinde güne göre ürünlerde indirim uygulanmasını hedef alan bir formül yer almaktadır. Formülü tamamen kafadan uydurduğumu ifirat etmek isterim. Zaten sizde bunu anlamışsınızdır. Asıl varmak istediğim nokta, StockLevel değerlerinin 497 ve 503 nolu ürünler için bilinçli olarak sıfıra set edilmiş olmasıdır. Bu nedenle bölme işlemi sırasında bir istisna oluşması kaçınılmazdır.

(Yani kendi kendimize kaşınıp kod içerisine bir bubi tuzağı koymuş durumdayız.)

önemli olan nokta, paralel sorgu motorunun bu tip bir durum ile karşılaştığında ne yapacağıdır. Nitekim söz konusu sorgulama tekniğine göre, operasyon bir kaç parça **Thread'** e bölünmete ve bu nedenle oluşacak bir istisnada(veya istisnalarda) çalışan iş parçalarına ne olacağı sorusu akla gelmektedir. İşte kodun yukarıdaki halinin çalışması sonrası ekran görüntümüz.



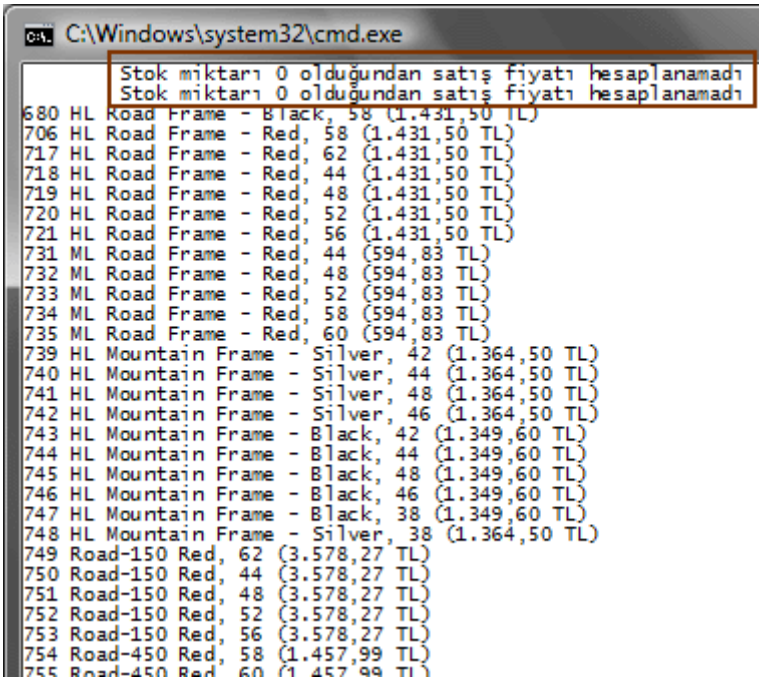
Görüldüğü gibi hiç bir Product bilgisi ekrana çıktı olarak gelmemiştir. Bu son derece doğaldır. Nitekim paralel sorgulama motoru herhangibir istisna ile karşılaştığında, çalışmakta olan tüm **Thread'** lerin iptal edilmesini sağlamaktadır. Diğer yandan istisna nesnesinin tipine dikkat edilmelidir. **PLINQ** ifadeleri içerisinde meydana gelebilecek istisnalar, **AggregateException** istisna sınıfı tarafından sarmalanmaktadır. Birden fazla istisna olabileceğinden, AggregateException sınıfı **InnerExceptions** isimli birde özelliğe sahiptir. Dolayısıyla **catch** bloğu içerisinde, sıfıra bölem hatasının yakalanması için, **InnerExceptions** koleksiyonunda dolaşılması gerekemtekdir. *(InnerExceptions özelliği ReadOnlyCollection<Exception> tipinden bir koleksiyon döndürmektedir.)*

Peki ya, istisna olan parçaların atlanması(bu örneğe göre) istenirse. Bir başka deyişle istisna almayan parçaların yinede paralel sorgulama sonucu ele alınması istenirse ne yapabiliriz?

Bu sorunun cevabı son derece basittir aslında. Exception yönetimi, **FindSellPrice** isimli metod içerisinde gerçekleştirilir.

```
static decimal FindSellPrice(decimal listPrice,int stockLevel)
{
   decimal result=-1;
   try
   {
   if (DateTime.Now.Day >= 25 && DateTime.Now.Day <= 28)
     result=listPrice - (listPrice * (1 / stockLevel)); // SafetyStockLevel' ın 0 gelmesi
halinde exception oluşacak olan yer.
   else
     result= listPrice * 1.18M;
   }
   catch(DivideByZeroException excp)
   {
    Console.WriteLine("\tStok miktarı 0 olduğundan satış fiyatı hesaplanamadı");
   }
   return result;
}
```

Bu durumda uygulama başarılı bir şekilde çalışacak ve aşağıdaki ekran görüntüsüne benzer sonuçlar alınacaktır.



Görüldüğü gibi istisnaya neden olan ürünler kontrollü bir şekilde elenmiş ve sorgunun paralel olarak yürütülmesine devam edilebilmiştir. Böylece geldik bir yazımızın daha sonuna. Bu yazımızda PLINQ sorgularında, istisna yönetiminde nelere dikkat etmemiz gerektiğine değinmeye çalıştık. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

ExceptionHandling.rar (27,33 kb)

PLINQ - Paralellik Altında Ardışık(Sequential) Çalışmak [Beta 1] (2009-05-25T23:34:00)

plinq,linq,

Merhaba Arkadaşlar,

Bir önceki blog yazımızda PLINQ ifadelerinde sıralama konusuna değinmeye çalışmıştık. Bu yazımızda ise, paralel olarak çalıştırılan LINQ sorguları içerisinde, **ardışık(Sequential)** olarak nasıl işlem yapılabileceğini incelemeye çalışacağız.

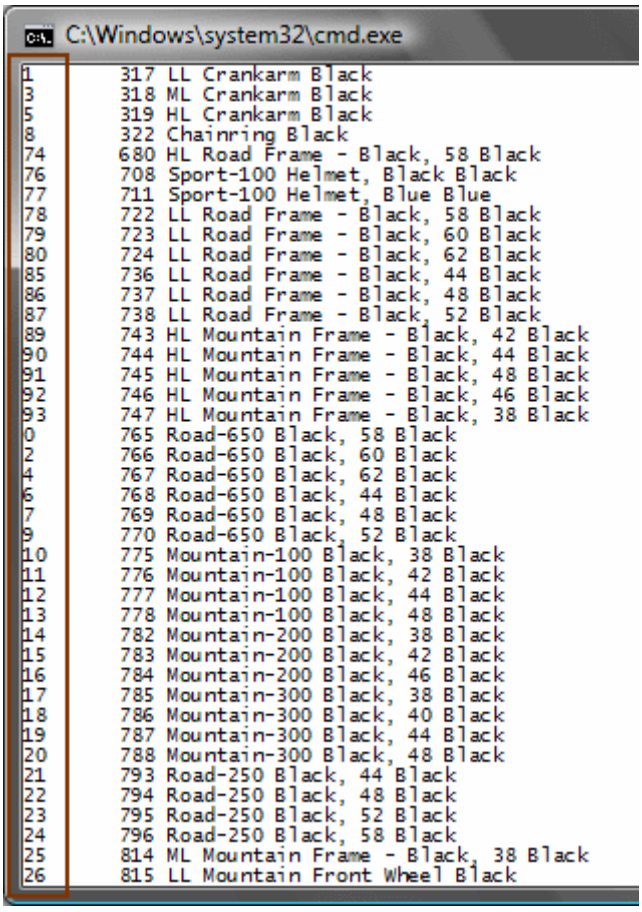
PLINQ ifadeleri, sorgu içerisindeki işlemleri paralel çalışan görevlere ayırmakta son derece başarılıdır. Ancak öyle senaryolar olabilirki, sorgunun belirli bir noktasından(noktalarından) sonra ardışık olarak işlemlerin devam etmesi istenebilir.*(Hatta sonra tekrardan paralel olarak devam edilmeside sağlanabilir)* Tabi bu şekilde anlatmaya çalışınca inanın benim kafamda karışıyor. 😕 Gelin olayı basit bir örnek ile ele almaya çalışalım. İşte **Visual Studio 2010 Professional Beta 1** üzerinde geliştirdiğim Console uygulamasına ait kodlar.

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Linq;

namespace SequentialPLINQ
{
   class Program
   {
      static void Main(string[] args)
      {
         List<Product> productList = GetProductList();

         int tid= 0;
         var result1 = from product in productList.AsParallel()
                       where product.Color.StartsWith("B")
                       orderby product.ProductId
                       select new
                       {
                          Id = tid++,
                          product.ProductId,
                          product.Name,
                          product.ListPrice,
                          product.Color
```

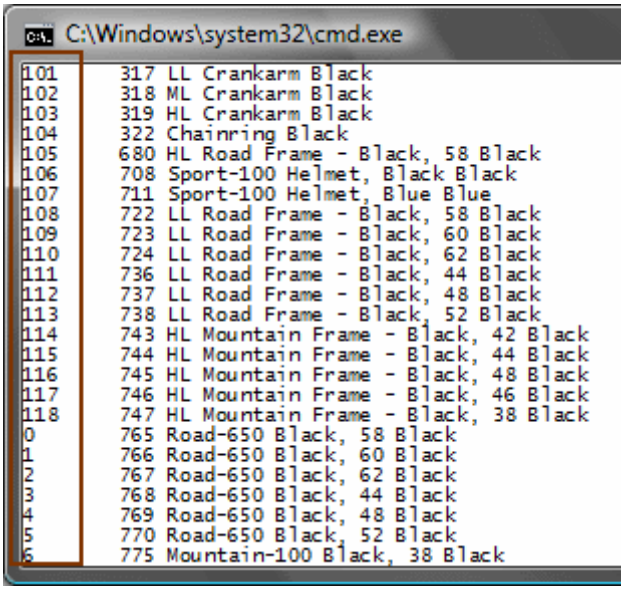
```
                };


            foreach (var r in result1)
            {
                Console.WriteLine(r.Id+" \t"+r.ProductId+" "+r.Name+" "+r.Color);
            }
        }

        static List<Product> GetProductList()
        {
            List<Product> productList = new List<Product>();

            SqlConnection conn = new SqlConnection("data
source=Manchester;database=AdventureWorks2008;integrated security=true");
            SqlCommand cmd = new SqlCommand("Select
ProductId,Name,ListPrice,ProductNumber,Color,SafetyStockLevel From
Production.Product", conn);
            conn.Open();
            SqlDataReader reader =
cmd.ExecuteReader(CommandBehavior.CloseConnection);
            while (reader.Read())
            {
                productList.Add(new Product
                {
                    ProductId=Convert.ToInt32(reader[0]),
                    Name=reader[1].ToString(),
                    ListPrice=Convert.ToDecimal(reader[2]),
                    ProductNumber=reader[3].ToString(),
                    Color=reader[4].ToString(),
                    SafetyStockLevel=Convert.ToInt32(reader[5])
                }
                );
            }
            reader.Close();
            return productList;
        }
    }

    class Product
    {
        public int ProductId { get; set; }
        public string Name { get; set; }
        public decimal ListPrice { get; set; }
        public string ProductNumber{ get; set; }
```

```
        public string Color { get; set; }
        public int SafetyStockLevel { get; set; }
    }
}
```

örnekte SQL Server 2008 üzerinde kurulu olan, **AdventureWorks2008** veritabanındaki **Production** şemasında yer alan **Product** tablosuna ait veriler kullanılmaktadır. Product tablosunun kod içerisindeki temsili için, Product isimli bir sınıf tasarlanmıştır. Sınıfa ait nesne örneklerinden oluşan generic **List<Product>** koleksiyonunun doldurulması için **GetProductList** metodundan yararlanılmaktadır. Söz konusu generic liste **PLINQ** ifadesi yardımıylada sorgulanmaktadır. Buraya kadarki kısımda zaten ilginç bir şey yok. Dikkat edeceğimiz nokta, **isimsiz tip(Anonymous Type)** içerisinde **tid** isimli sayacın arttırılmasıdır. öyleki uygulamayı çalıştırdığımızda aşağıdaki sonuçları elde ederiz.



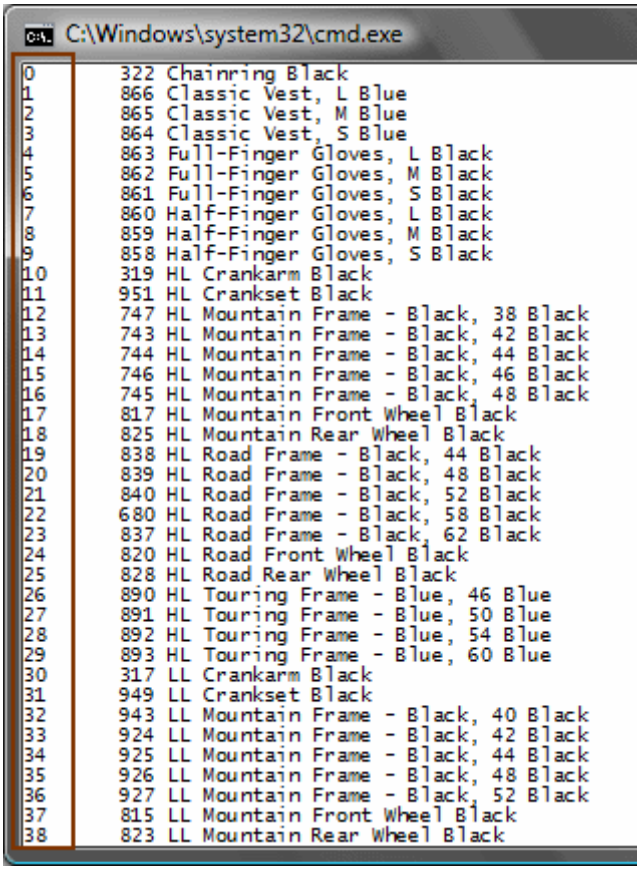
Burada dikkat edilmesi gereken nokta **tid** değerlerinin ardışık mantığa göre değil, paralel çalışmanın bir sonucu olarak farklı sıralarda üretilmesidir. Bu nedenle 1, 3, 5, 8, 74... gibi bir dizi oluşmuştur. Bu dizi kodun her çalıştırılmasında farklı şekillerde üretilebilir. örneği ikinci kez çalıştırdığımda bu kez aşağıdaki sonuçları aldım.



Görüldüğü üzere **tid** değerlerinin arttırımının, çıktıya yansıyışı farklı olmaktadır. Hatta **MSDN** kaynaklarında, arttırımı yapılan değerlerin tekrar etmesininde mümkün olabileceği berlitilmektedir. İşte yazmış olduğum bu anlamsız örnekteki gibi *(örneğin arttırım işlemlerinin ardışık bir şekilde gerçekleştirilmesinin gerektiği vb...)*, paralel olarak çalışan LINQ sorguları içerisinde, ardışık çalışması gereken bölümler var ise, **AsSequential genişletme metodunun(Extension Method)** kullanılması gerekmektedir. Buna göre yukarıdaki örnekte yer alan LINQ sorgusunu,

```
var result1 = productList
            .AsParallel()
            .Where(p => p.Color.StartsWith("B"))
            .OrderBy(p => p.Name)
            .AsSequential()
            .Select(
            p => new
        {
            Id = tid++,
            p.ProductId,
            p.Name,
            p.ListPrice,
            p.Color
        });
```

şeklinde değiştirir ve örneği tekrar çalıştırırsak aşağıdaki sonuçları elde ederiz.



Görüldüğü gibi **tid** arttırımı düzenli bir şekilde çıktıya yansıtılmıştır. (*Yinede dikkatlice baktığınızda Name özelliğine göre yapılan sıralamalarda, her iki LINQ sorgusu arasında küçük farklar olabileceğini görebilirsiniz)*

Sonuç olarak paralel olarak çalıştırılan LINQ sorguları içerisinde, ardışık yürütülmesi gereken operasyonlar var ise bu durumda **AsSequential** genişletme metodnun kullanılması gerekmektedir. Diğer taraftan elbetteki bu kullanım, söz konusu paralel çalışmanın hızını düşürecek bir etkiye neden olacaktır. Bu da gözden kaçırılmaması gereken diğer bir noktadır.

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

SequentialPLINQ.rar (25,80 kb)

PLINQ - Sıralamayı(Ordering) Korumak [Beta 1] (2009-05-24T05:30:00)

plinq,linq,

Merhaba Arkadaşlar,

Hatırlayacağınız gibi, **PLINQ(Parallel LINQ)** ile ilişkili ilk yazımda, LINQ sorgularının eş zamanlı olarak nasıl çalıştırılabileceğini incelemeye çalışmıştık. Hello World örneğimizde ağırlıklı olarak aşağıdaki sorgu üzerinde durmuştuk.

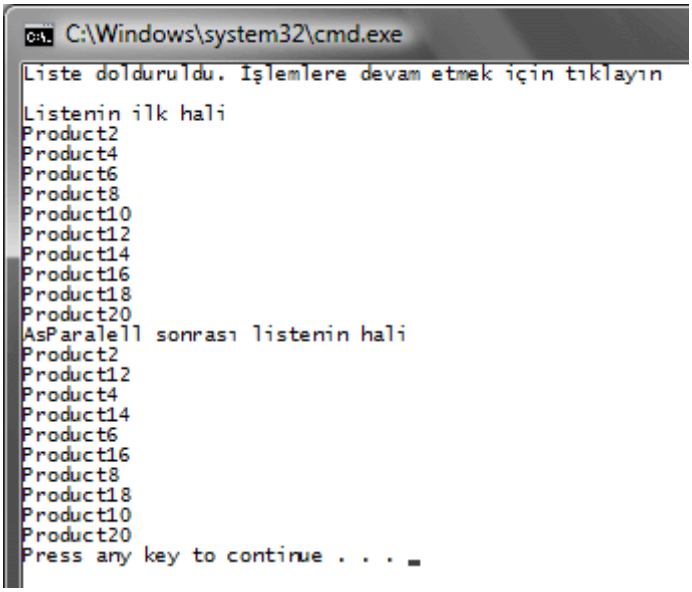
```
var result2 = from p in products.AsParallel()
              where p.ListPrice >= 10 && p.InStock==true
              orderby p.Name descending
              select p;
```

Bu sorguda yer alan orderby kelimesi aslında çok büyük bir öneme sahiptir. Gelin ne demek istediğimi size anlatmaya çalışayım. Yine **Visual Studio 2010 Professional Beta 1** ortamında geliştirilen aşağıdaki kod parçasına sahip bir Console uygulamamız olduğunu göz önüne alacağız.

```
using System;
using System.Collections.Generic;
using System.Linq;

namespace HelloWorld
{
   class Program
   {
      static void Main(string[] args)
      {
         List<Product> products = FillProducts();
         Console.WriteLine("Liste dolduruldu. İşlemlere devam etmek için tıklayın");
         Console.ReadLine();

         Console.WriteLine("Listenin ilk hali");
         foreach (Product prd in products)
         {
            if(prd.InStock==true)
               Console.WriteLine(prd.Name);
         }

         var result = from p in products.AsParallel()
                      where p.InStock == true
                      select p;

         Console.WriteLine("AsParalell sonrası listenin hali");
         foreach (Product prd in result)
         {
            Console.WriteLine(prd.Name);
         }
      }

      static List<Product> FillProducts()
      {
         List<Product> products = new List<Product>();
```

```
        for (long i = 1; i < 21; i++)
        {
            Product prd = new Product
            {
                Id = i
                ,Name = "Product" + i.ToString()
                ,ListPrice = i * 0.1M
                ,InStock = i % 2 == 0 ? true : false
            };
            products.Add(prd);
        }

        return products;
    }
}


class Product
{
    public long Id { get; set; }
    public string Name { get; set; }
    public decimal ListPrice { get; set; }
    public bool InStock { get; set; }
}
}
```

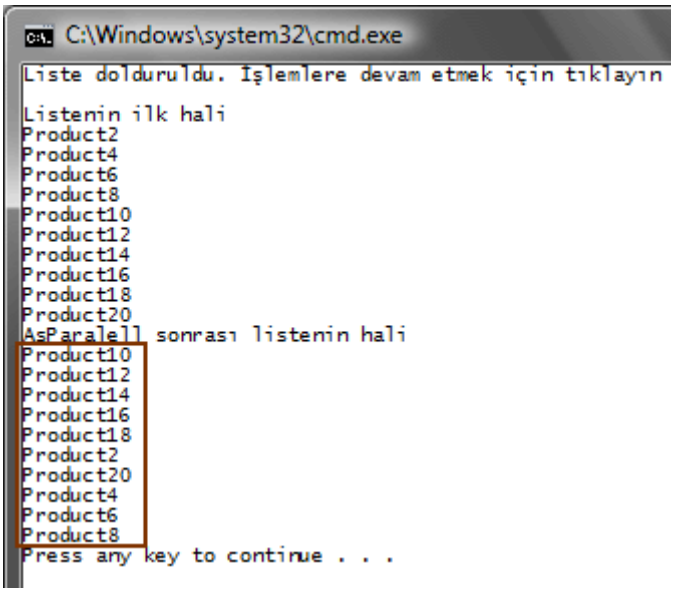
Bu seferki örneğimizde temel amacımız hız veya işlemlerin daha kısa sürede tamamlanması değildir. products isimli generic List koleksiyonu doldurulduktan sonra bir foreach döngüsü yardımıyla dolaşılmakta ve stokta olanlar(InStock==true) ekrana yazdırılmaktadır. Sonrasında ise **PLINQ** sorgumuz gelmekte ve aynı sonuçları paralel çalışan task' ler üzerinde elde etmemizi sağlamaktadır. Hemen küçük bir dip not belirtelim; LINQ sorguları aslında kullanıldıkları anda çalıştırılmaktadır**(Deferred Execution).** Yani çalışma zamanında ikinci foreach döngüsüne gelindiğinde, söz konusu PLINQ sorgusu yürütülmekte ve dolayısıyla paralel görevler devreye girmektedir. Peki herşey güzel hoş... Hoş ama niye bunun üzerinde duruyoruz? Aslında çalışma zamanındaki ekran görüntüsü herşeyi biraz olsun açıklıyor.



Dikkat edileceği üzere, listenin ilk halinde stokta olan ürünler, koleksiyonda yer aldıkları sıraya göre ekrana getirilmektedir. Ancak **AsParallel** genişletme metodunun kullanılması sonrasında elde edilen listedeki ürünler sıralı bir şekilde gelmemektedir. Bu çok doğal olarak paralel çalışmanın bir sonucudur. Ancak bazı hallerde **AsParallel** kullanımı sonrasında, kaynak listenin sıralı olarak elde edilmesi istenebilir. Bu durumda **orderby** kullanımı sorunu çözecektir. Söz gelimi yukarıdaki örnekte yer alan LINQ ifadesinde orderby aşağıdaki gibi kullanılabilir.

```
var result = from p in products.AsParallel()
             where p.InStock == true
             orderby p.Name
             select p;
```

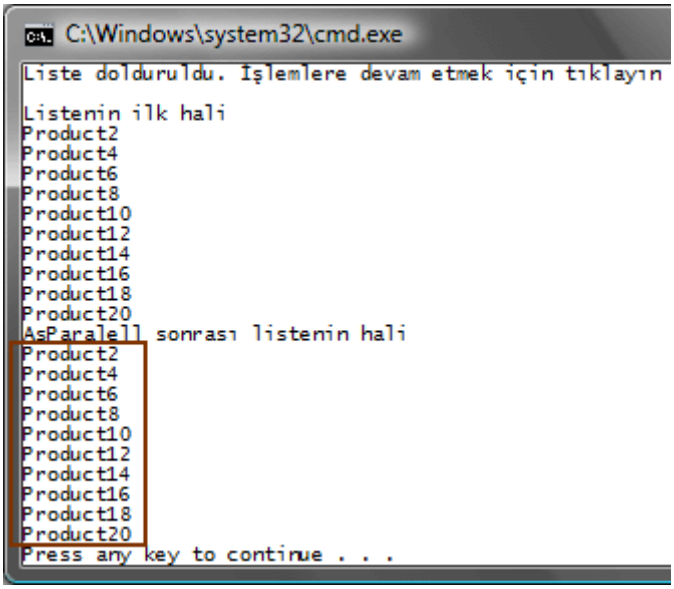
Bu durumda örneğin çalışması sonucu aşağıdaki çıktı elde edilir.



Ancak PLINQ tipinden sorgunun çalıştırılması sırasında orjinal nesne sırasının korunması da istenebilir ki bu orderby kullanımından daha farklı anlamdadır. *(orderby kullanımında listenin, koleksiyondaki orjinal sırası yerine, sıralama kriteri belirtilir.)* İşte bu noktada devreye **ParallelEnumerable** static sınıfı içerisinde tanımlanmış olan **AsOrdered** isimli**genişletme metodu(Extension Method)** girer. Yani sorguyu aşağıdaki hale getirirsek, liste elemanlarının koleksiyon içerisindeki orjinal sırasını koruyarak sonuç elde edilmesi sağlanabilir.

```
var result = from p in products.AsParallel()
             where p.InStock == true
             select p;
```

Ve sonuç...



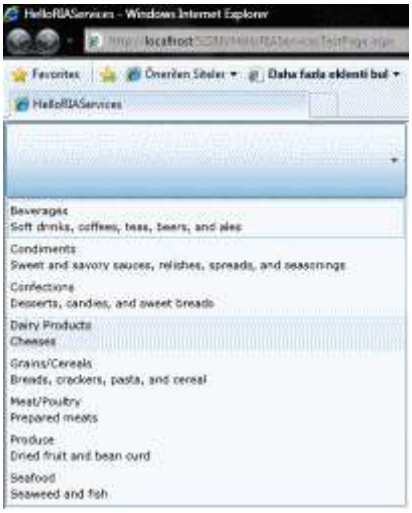
Tabi burada önemli bir nokta daha vardır. AsOrdered çok doğal olarak PLINQ sorgunun çalışma zamanında yavaş işlemesine neden olacaktır. çünkü, sorgu sonucu elde edilen liste eşitliğin sol tarafına aktarılmadan önce, AsOrdered nedeni ile orjinal sıra konumlarına yerleştirilir. Bu ek işlem, sonucun elde edilmesini yavaşlatacaktır. Bu nedenle **MSND** kaynaklarında,**AsOrdered** genişletme metodunun gerekmedikçe kullanılmaması öğütlenmektedir. Hatta, **orderby** kullanımının tercih edilmesi önerilmektedir. Yazımızın başında belirttiğimiz Orderby kullanımının neden önemli olduğunu sanırım anlamış bulunuyoruz. Tabiki zorunlu hallerde **AsOrdered** kullanılmasıda gerekebilir.

Bu yazımda sizlere önemsiz gibi görünen fakat dikkat edilmesi gereken bir konuyu aktarmaya çalıştım. Bir sonraki yazımızda görüşünceye dek mutlu günler dilerim.

Ordering.rar (22,74 kb)

.Net TV - .Net RIA Servisleri Hello World (2009-05-23T00:45:00)

.net ria services,



Merhaba arkadaşlar,

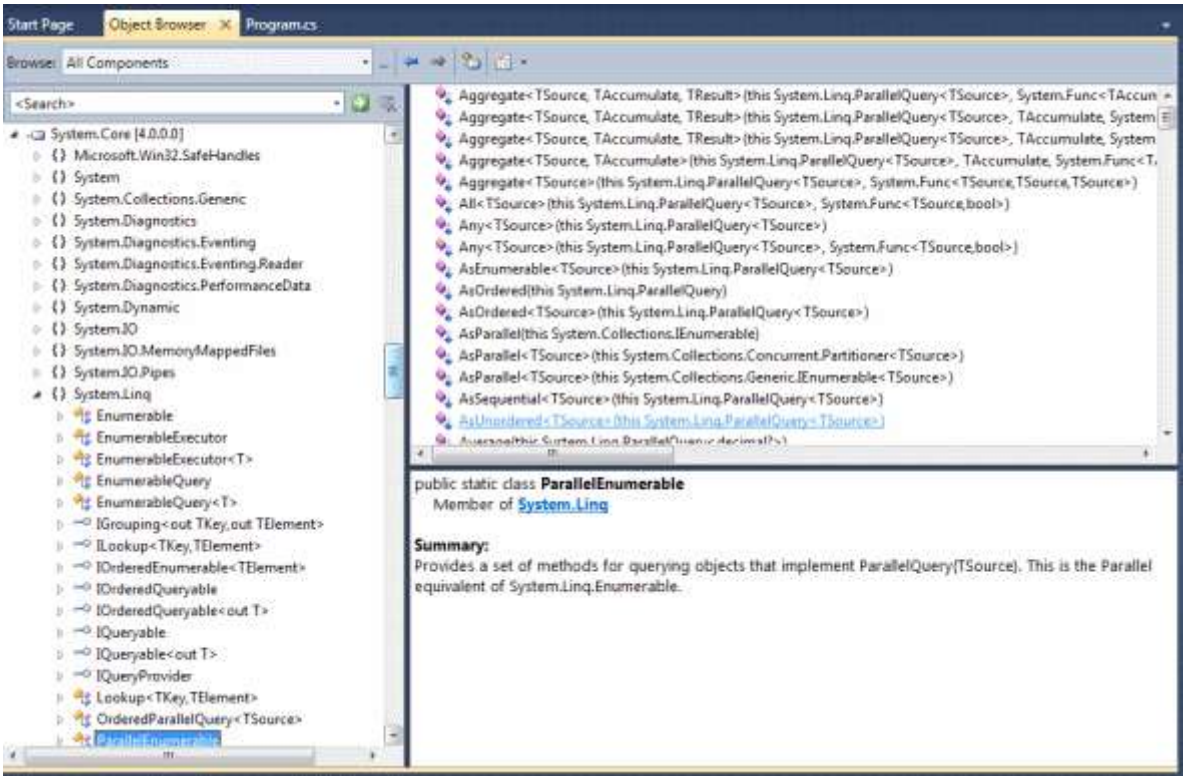
Bir süredir .Net RIA Servisleri ile ilişkili araştırmalar yapıyorum ve bildiklerimi sizinle paylaşıyorum. Yazıları desteklemesi açısından giriş seviyesinde bir görsel videoyuda az önce yayınladım. Şu adresten indirip, .Net RIA Servislerinin nasıl kullanıldığını izleyebilirsiniz. Faydalı olması dileğiyle iyi seyirler.

PLINQ (Parallel LINQ) - Hello World [Beta 1] (2009-05-22T07:11:00)

plinq,linq,

Merhaba Arkadaşlar,

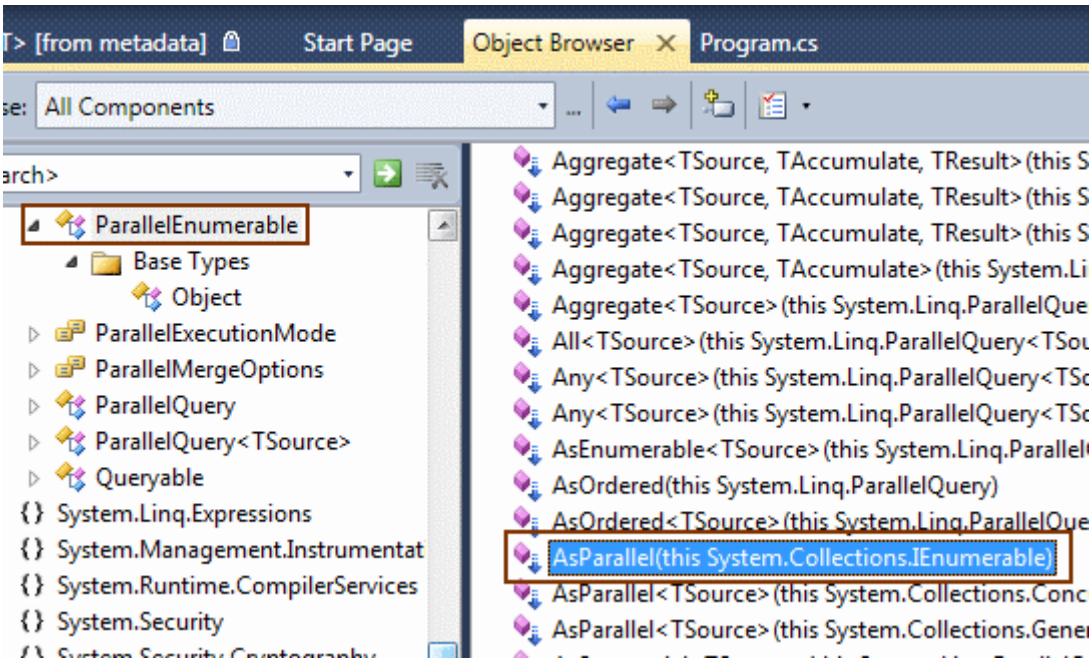
Bildiğiniz gibi son yazımı deniz kenarında bir kafede tatildeyken yazmıştım Ama tatil bitti malesef ve tekrardan Morpheus' un sözleri kulaklarımda çınladı **"Wellcome to the real world".** Yinede 1 haftalığınada olsa tatil yapabildiğime şükrediyorum. Gerçek dünyaya döndükten sonra tabiki bir süre adaptasyon sorunları ile karşılaşıyor insan doğal olaraktan. Bu adaptasyon sorunları içerisinde boğuşurken, neleri araştırabilirim diye düşünürken buluverdim kendimi. Herşeyden önce **.Net Framework 4.0** ve **Visual Studio 2010 Beta 1**sürümlerinin yayınlandığını hepimiz biliyoruz. Dolayısıyla odaklanılacak konu zaten ap açık ortadaydı. .Net Framework 4.0 içerisinde entegre olarak gelen bir yenilik hemen ilgi odağım oldu. **PLINQ(Parallel Language INtegrated Query)**. Aslında PLINQ yeni çıkmış bir eklenti değil. Zaten uzun süredir **.Net Framework 3.5** ve **Visual Studio 2008** üzerinde CTP sürümü ile testlerimizi yapabiliyorduk. Ne varki, .Net Framework 4.0 göz önüne alındığında PLINQ ile ilişkili tiplerin **System.Core.dll** assembly' ının **4.0** versiyonu içerisine doğrudan ilave edildiğini görüyoruz. Aşağıdaki Visual Studio 2010 Object Browser' dan alınan görüntüde bu durum açık bir şekilde gözlemlenebiliyor.



Tabiki öncelikli olarak **PLINQ** kavramından biraz bahsetmemizde yarar var. PLINQ aslında, **Microsfot Research** ve **CLR(Common Language Runtime)** takımları tarafından ortaklaşa geliştirilen**Parallel Extensions** isimli genişletmelerin sadece bir paçasıdır. Diğer parça ise **TBL(Task Parallel Library)** dir.(Bunu ilerleyen yazılarımda ele almaya çalışacağım) Her iki yapının kullanım amacı, **Yönetimli Kod(Managed Code)** tarafındaki eş zamanlı işleyişlerin kolay bir şekilde sağlanmasıdır. Söz konusu yapı PLINQ olunca haliyle, LINQ sorgularının kendi içerisine parçalanarak farklı **thread'** lerde çalışması ve bu parçaların paralel yürüyerek sonuçların elde edilmesi akla gelmektedir. Gerçektende PLINQ yapısının temel amacı bu şekilde özetlenebilir. Hatta PLINQ için **Eş Zamanlı Sorgu Yürütme Motorudur(Concurrency Query Execution Engine)** diyebiliriz. PLINQ temel olarak **LINQ to XML** ve **LINQ to Objects**gibi uygulama alanları üzerinde etkin bir şekilde kullanılabilmektedir.

NOT : *Unutulmaması gereken noktalardan biriside, PLINQ ifadelerinin aslında çift çekirdek ve üstü işlemcilerin yada birden fazla işlemcinin olduğu sistemlerde anlamlı olmasıdır. Nitekim, PLINQ motoru, çalışmakta olan sorgu sürecini, makinenin sahip olduğu çekirdek sayısına göre parçalara ayırır ve yürütür. Bu özellikle büyük çaplı projeler göz önüne alındığında, şirketin sahip olduğu kaç bilgisayar var ise hepsini en azından çift çekirdekli olacak şekilde yenilemek gibi bir maliyet anlamına da gelmemelidir. Nitekim bazı istemci-sunucu mimarilerinde, sunucu tarafında çalışmakta olan pek çok LINQ sorgusu, PLINQ motoru kullanılaraktan daha efektif hale getirilebilir. Bir başka deyişle, istemciler birden fazla çekirdekli işlemcilere sahip olmasalarda, mümkün mertebe LINQ ifadelerini içeren iş mantıklarının, sunucu tarafında olduğu senaryolarda PLINQ büyük avantajlar sağlayabilir(çok kısa bir süre önce çalışmakta olduğum bir projede yer alan test makinesinin, 8 işlemcili olduğunu hatırlıyorum)*

Tabi burada var olan nesneler üzerindeki LINQ sorgularının paralel olarak çalıştırılması için, **Select, Where** gibi **genişletme metodlarının(Extension Methods)** çalışma sırasında işi farklı parçalara bölebilecek versiyonlarının olması gerektiği düşünülebilir. İşte bu noktada devreye, **System.Core assembly**' ının **4.0** versiyonu içerisinde yer alan ve**System.Linq** isim alanında bulunan **ParallelEnumerable** adlı **static** sınıf girmektedir.



Bu sınıftaki en önemli genişletme **AsParallel** isimli fonksiyondur. Bu metodun görevi, **IEnumerable** türevli bir koleksiyonun paralel olarak sorgulanabilir hale getirilmesi veya hazırlanmasıdır. öyleki, metod geriye **ParallelQuery** isimli sınıfa ait bir nesne örneği döndürmektedir. **ParallelQuery** sınıfı **IEnumerable** arayüzünü uygulamaktadır ama herşeyden önemlisi paralel sorgulanabilme için gerekli ön hazırlıkları içeren operasyonlarada sahiptir.

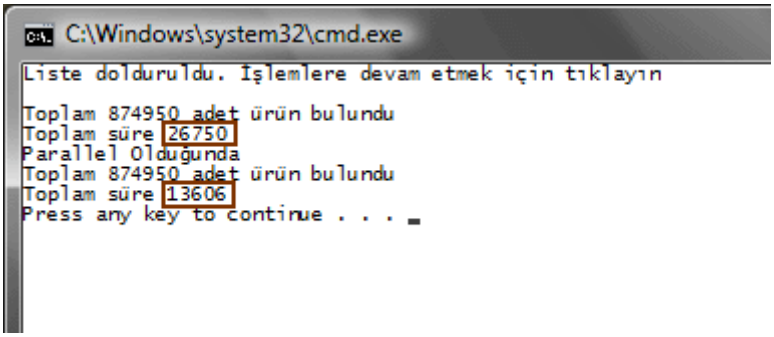
Bu teknik detaylar eminimki bir Hello World yazısında sizede sıkıcı gelmiştir. Hiç vakit kaybetmeden basit bir örnek ile ilerlemekte yarar olduğunu düşünmekteyim. Aşağıdaki kod parçası **Visual Studio 2010 Beta 1** sürümünde yazılmış basit bir **Console** uygulamasına aittir.

```
using System;
using System.Collections.Generic;
using System.Diagnostics;
using System.Linq;

namespace HelloWorld
{
   class Program
   {
      static void Main(string[] args)
```

```
        {
            List<Product> products = FillProducts();
            Console.WriteLine("Liste dolduruldu. İşlemlere devam etmek için tıklayın");
            Console.ReadLine();

            Stopwatch watch = Stopwatch.StartNew();

            var result1 = from p in products
                          where p.ListPrice >= 10 && p.InStock == true
                          orderby p.Name descending
                          select p;
            Console.WriteLine("Toplam {0} adet ürün
bulundu",result1.ToList().Count.ToString());

            Console.WriteLine("Toplam süre {0}",watch.ElapsedMilliseconds.ToString());
            Console.WriteLine("Parallel Olduğunda");

            Stopwatch watch2 = Stopwatch.StartNew();

            var result2 = from p in products.AsParallel()
                          where p.ListPrice >= 10 && p.InStock==true
                          orderby p.Name descending
                          select p;
            Console.WriteLine("Toplam {0} adet ürün bulundu",
result2.ToList().Count.ToString());

            Console.WriteLine("Toplam süre {0}", watch2.ElapsedMilliseconds.ToString());
        }

        static List<Product> FillProducts()
        {
            List<Product> products = new List<Product>();

            for (long i = 1; i < 1750000; i++)
            {
                Product prd = new Product {
                    Id = i
                    , Name = "Product" + i.ToString()
                    , ListPrice = i * 0.1M
                    , InStock=i%2==0?true:false
                };
                products.Add(prd);
            }
```

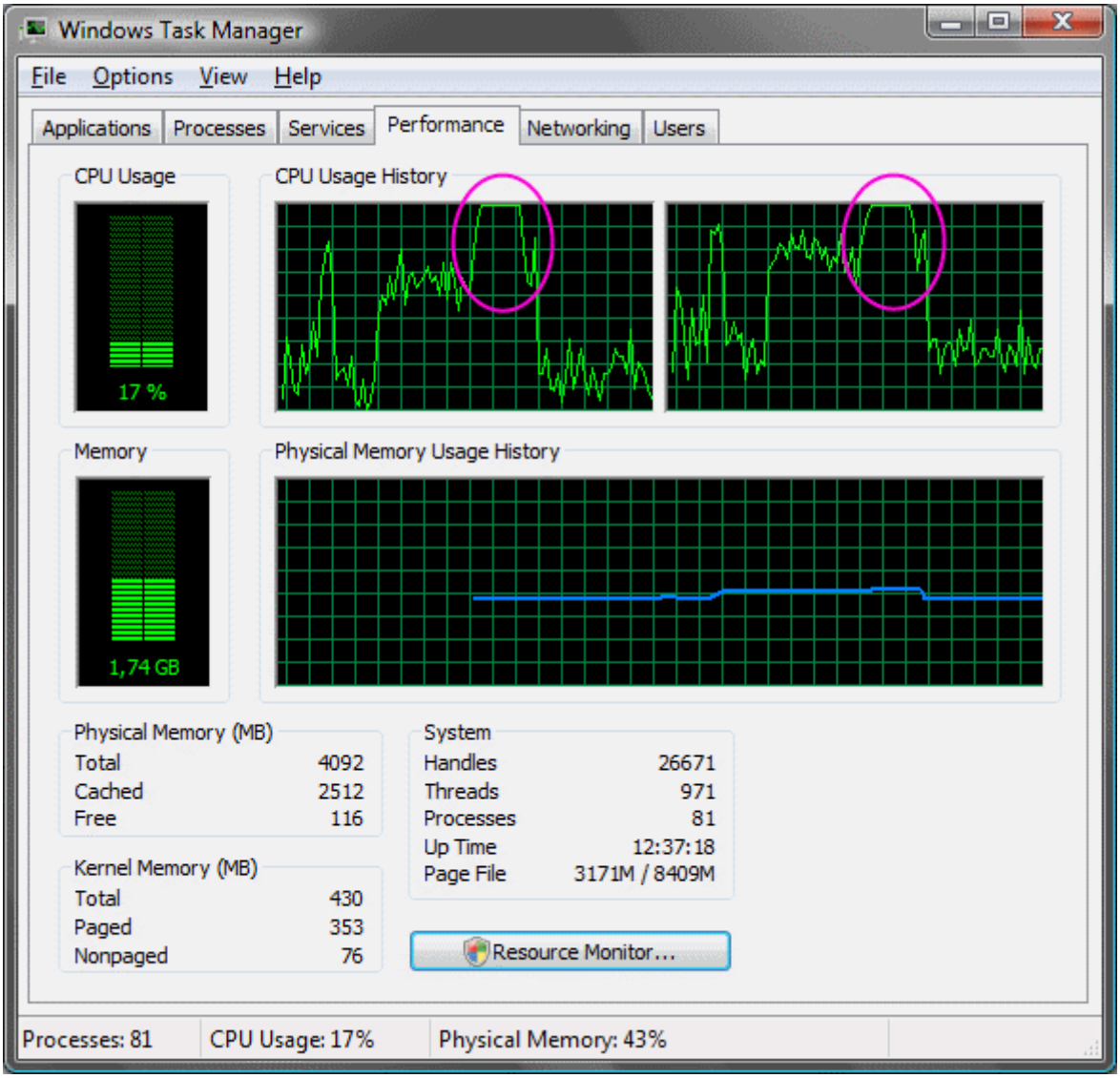
```
            return products;
        }
    }


    class Product
    {
        public long Id { get; set; }
        public string Name { get; set; }
        public decimal ListPrice { get; set; }
        public bool InStock { get; set; }
    }
}
```

Uygulama içerisinde **Products** isimli bir sınıf ve bu tipe ait nesne örneklerinden oluşan bir koleksiyon veri kaynağı olarak kullanılmaktadır. Dikkat edileceği üzere, iki adet **LINQ** sorgusu bulunmaktadır. Product tipinden olan generic **List** koleksiyonu, **FillProducts** metodu yardımıyla tamamen hayali veriler ile doldurulmuştur. Her iki sorguda **ListPrice değeri 10' un üzerinde olan ve stokta bulunan ürünleri**, **adlarına göre ters sırada** döndürmektedir. Ancak önemli olan nokta ikinci LINQ ifadesinde **AsParallel** metodunun kullanılmasıdır. Bu örnek kod parçasını çalıştırdığımda, aşağıdaki ekran görüntüsünde yer alan sonuçları aldım.



Hemen şunu belirteyim. Programı yazdığım makinede **çift çekirdekli Intel işlemci** ve **4 Gb Ram** bulunmakta. İşletim sistemi olarakta **Windows Vista** Enterprise yer alıyor. Tabi bu örnek için Intel tabanlı işlemcinin daha büyük önem taşıdığını hemen söyleyebiliriz. çalışma zamanındanda görüldüğü gibi, paralel olarak yürütülen LINQ ifadesi neredeyse **%50** daha az zamanda tamamlanmıştır. (Aslında bu kod parçasını 4 çekirdekli bir işlemcide test etmeyi çok istiyorum. Bu konuda siz değerli okurlarımın yorumlarını ve test sonuçlarını bekliyor olacağım ☺)

Uygulama çalışırken **Task Manager** aracı ile **CPU** kullanım durumuna baktığımda ise aşağıdaki sonuçlar ile karşılaştım.

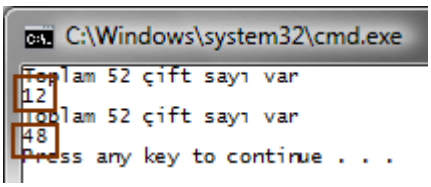


Bu ekran görüntüsünde yer alan sonuçlar tam anlamıyla durmun net analizi olmasada bir parça olsun fikir vermektedir. Yuvarlak içerisine aldığımda kısımlar, sorgunun **PLINQ** motoru tarafından ele alınmaya başladığı yerlerdeki ölçüm değerleridir. Dikkat edileceği üzere CPU çekirdeklerinin kullanım değerleri **%100'** e vurmuş durumdadır ki buda aslında, sorgunun çalıştırılması sırasında tüm işlemci gücünün kullanıldığı anlamınada gelmektedir. (Aslında Einstein' ın kuramına göre bu durum göreceli olarak iyi sayılabilir. Ama sayılmayadabilir 😕)

Yazdığım örnek kod parçasında işlemleri gerçekten yavaşlatmak adına bir sıralama işlemide kullandım. Anacak bunun yapılması zorunlu değildir. özellikle sıralama kullanılmadığında sorgu çalıştırma sürelerinin birbirlerine çok yakın olduğunu gördüm. Açıkçası, **PLINQ'** in avantajı gerçekten çok uzun sürebilecek sorgular söz konusu olduğunda ortaya çıkmata. Bu nedenle her LINQ sorgusunun PLINQ formatına dönüştürülmesininde anlamlı olmadığını(olmayacağını) söyleyebiliriz. Nitekim, bazı durumlarda herşey tersine dönebilir. örneğin aşağıdaki kod parçasını göz önüne alalım.

```
int[] values = new int[100];Random rnd = new Random();
for (int i = 1; i < values.Length; i++)
{
 values[i] = rnd.Next(1, 1000);
}

Stopwatch watch3 = Stopwatch.StartNew();
var result3 = from value in values
              where value % 2 == 0
              select value;
Console.WriteLine("Toplam {0} çift sayı var",result3.ToList().Count.ToString());
Console.WriteLine(watch3.ElapsedMilliseconds.ToString());

Stopwatch watch4 = Stopwatch.StartNew();
var result4 = from value in values.AsParallel()
              where value % 2 == 0
              select value;
Console.WriteLine("Toplam {0} çift sayı var", result4.ToList().Count.ToString());
Console.WriteLine(watch4.ElapsedMilliseconds.ToString());
```

Bu kod parçasındaki LINQ sorgularında, 100 tane raslantısal ve 1 ile 1000 arasında olan tamsayı değerlerinden oluşan bir dizi içerisinde kaç çift sayı olduğu tespit edilmektedir. İkinci sorgu, PLINQ motoru tarafından ele alınmaktadır. PLINQ' in, sorguyu paralel olan iş parçalarına bölerek çalıştırdığı düşünüldüğünde, ikinci ifadenin birincisine göre çok daha hızlı çalışması gerektiği tahmin edilebilir. Ama gerçekten böylemi olacaktır. İşte sonuçlar...



Aslında bu sonuç son derece doğaldır. **PLINQ** motoru çalışma zamanında, çekirdeklere bölünecek işler için hazırlıklar yapmalıdır, thread' leri ayarlamalıdır vb... Bu ön hazırlıklar nedeni ile zaten sorgunun sürece girmesi başlı başına bir zaman kaybı anlamına gelmektedir. Bu örnek en basit anlamda, PLINQ' in her LINQ ifadesi için ele alınmaması gerektiğinide göstermektedir.

Böylece geldik bir yazımızın daha sonuna. Tatil dönüşü sonrası üzerimdeki adaptasyon bıkkınlığını hafifleten bu yazımda sizlere, **.Net Framework 4.0** içerisinde artık standart olarak yer alan ve **Parallel Extension** mimarisinin bir parçası olan **PLINQ** konusunu anlatmaya çalıştım. Elbetteki PLINQ içerisindede çok daha fazlası var. Bunlarıda ilerleyen yazılarımda aktarmaya çalışıyor olacağım. Sizlerde bu adresten Parallel Extension ile ilişkili son bilgileri alabilirsiniz. Hatta şu saatlerde VS 2010 ile gelen yeniliklerde anlatılmakta. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HelloWorld.rar (21,81 kb)

.Net RIA Servisleri - DomainDataSource Kulanımı (2009-05-14T21:30:00)

.net ria services,silverlight,

Merhaba Arkadaşlar,

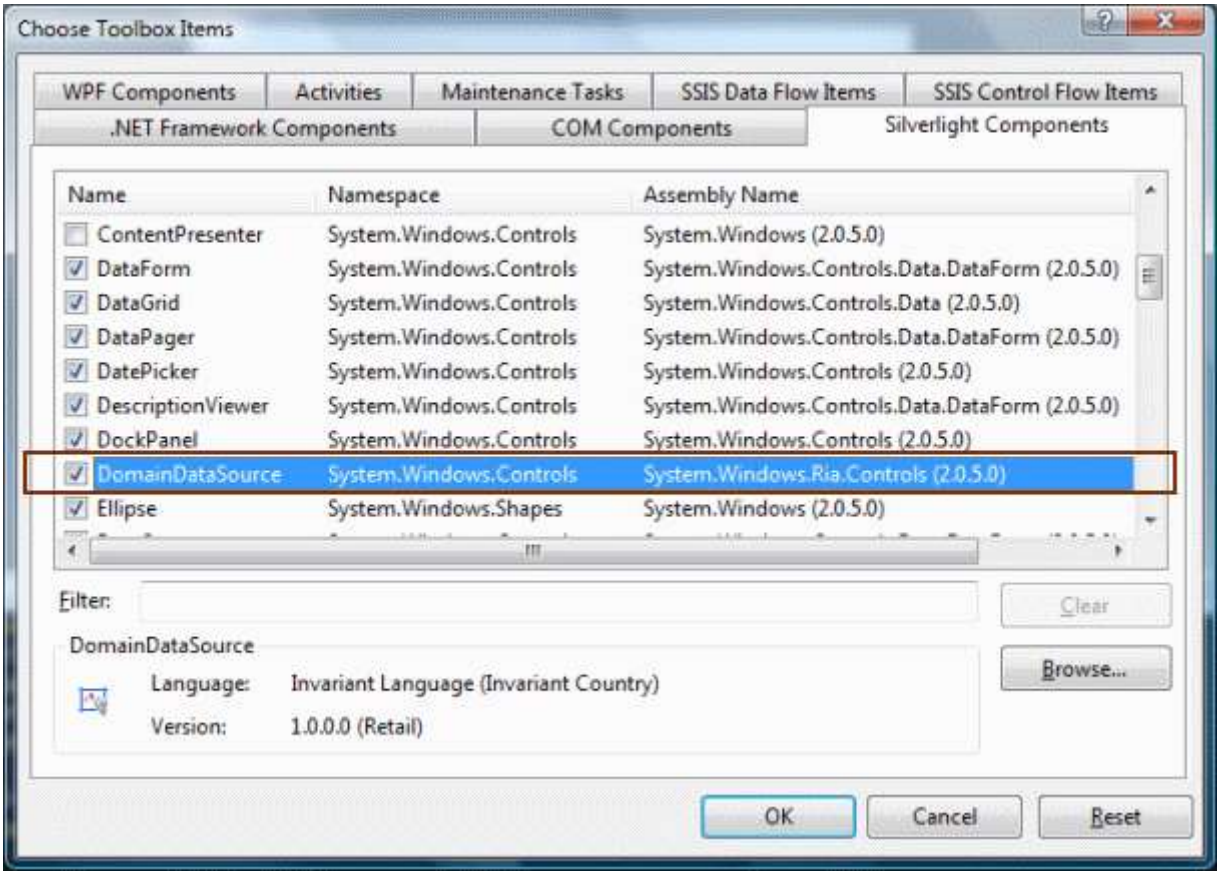
Her ne kadar şu günlerde güzel ülkemizin **Ege** kıyılarında kısa bir dinlenme molası vermiş olsamda, internetin sahil kıyılarındaki cafe' lere kadar girmiş olması, herşeyi değiştiriyor. 😎 Artık bir yaşam tarzı haline gelen Yazılımdan, onun gizemli dünyasından uzak durmak bu nedenle, şu sıralar aşağıdaki şekilde görülen yerde tatilde bile olsam çok zor.

Bu kısa yazımda sizlere yine **.Net RIA Servisleri** ile ilişkili bilgilerimi aktarmaya gayret edeceğim. Bu seferki konumuz **DomainDataSource** isimli **Silverlight** kontrolü. Kontrolün adında yer alan**DataSource** son eki aslında olayı biraz olsun açıklamakta. .Net RIA Servislerinin kullanıldığı senaryolarda, sunucu tarafında mutlaka bir veri kaynağı yer almaktadır. Ağırlık olarak**Ado.Net Entity Framework** veya **LINQ to SQL** tabanlı sağlayıcılar ile eriştiğimiz bu veri kaynaklarını, istemci tarafında değiştirmek gibi işlemlerle uğraştığımızda bir gerçektir. Kısacası, istemci tarafına çekilen verinin sadece gösterilmesi dışında, düzenlenmesi, yenilerinin eklenmesi veya var olanların silinmesi gibi operasyonlar söz konusudur. Bunlara ek olarak, **Silverlight** tabanlı istemci tarafını düşündüğümüzde, verinin kullanıcı ile etkileşimde olan kontrollerde gösterilmeside bu işin önemli kısımlarından birisidir.

Tam bu noktada aklıma Asp.Net 2.0 ile birlikte gelen **veri-bağlı kontrolleri(Data-Bound Controls)** geliyor. **SqlDataSource**, **ObjectDataSource**, **SiteMapDataSource** vb...Bu kontrollerin en büyük amacı, sayfa üzerindeki sunucu kontrollerini, veri kaynağını bağlamaktır. **SqlDataSource** gibi kontroller sayesinde bu bağlama işlemleri ile birlikte, **Insert, Update** ve **Delete** operasyonlarına hizmet edecek kod parçalarının kolay bir şekilde geliştirilmesi ve ele alınmasıda mümkün olmaktadır. Şu anda bulunduğumuz noktayı düşündüğümüzde, .Net RIA Servislerini kullanan Silverlight istemcileri içinde benzer bir kolaylığın sağlanması önemlidir. öyleyse bu kontrol nasıl kullanılır, tam olarak ne işe yarar hemen bir bakalım.

örnek Silverlight Projemizde bu kez Northwind veri kaynağına **Ado.Net Entity Framework** öğesi yardımıyla bağlanıyor olacağız. Yine bir önceki blog yazımızda olduğu gibi**Categories** tablosunu ve ek olarak **Products** tablosunu kullanabiliriz. İstemci tarafında,

Insert, Update ve Delete gibi işlemleride ele alma ihtimalimiz olduğundan(en azından sonraki blog yazılarımda),**DomainService** tipinin eklenmesi sırasında, her iki **Entity** tipi içinde **Enable Editing** özelliğinin işaretli olduğuna dikkat etmemiz gerekmektedir. Gelelim kullanacağımız **DomainDataSource**bileşenine. Bu bileşen varsayılan olarak **Silverlight** kontrol sekmesinde görünmemektedir. Dolayısıyla söz konusu kontrolün, .Net RIA Service sisteme yüklendikten sonra **Visual Studio 2008**ortamında kullanılabilmesi için **Toolbox'** a **Silverlight Components** kısmından eklenmesi gerekir.



DataSource bileşenleri genellikle veri-bağlı kontroller ile kullanılırlar. **DomainDataSource** kontrolünü bu anlamda, **DataGrid** bileşeni ile etkileştirebiliriz. **XAML** içeriğimizin ilk halini aslında aşağıdaki gibi tasarladım.

```
<UserControl xmlns:riaControls="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Ria.Controls"
   xmlns:data="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
   x:Class="DomainDS.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:ds="clr-namespace:DomainDS.Web"
   Width="500" Height="300">
   <Grid x:Name="LayoutRoot" Background="White">
      <riaControls:DomainDataSource x:Name="dsProducts" AutoLoad="True"
```

```
LoadSize="10" LoadMethodName="LoadProducts">
        <riaControls:DomainDataSource.DomainContext>
          <ds:NorthwindContext/>
        </riaControls:DomainDataSource.DomainContext>
      </riaControls:DomainDataSource>
   <data:DataGrid x:Name="dgProducts" Height="Auto"
Background="BlanchedAlmond" ItemsSource="{Binding Data,
ElementName=dsProducts}"/>
   </Grid>
</UserControl>
```

Şimdi bu XAML içeriğinde, üzerinde durulması gereken önemli noktalar olduğu aşikardır. öncelikli olarak **DataGrid** veya **DomainDataSouce** kontrollerini kullanabilmemiz için gerekli **assembly**veya isim alanları(Namespaces), ilgili bileşenleri **XAML** içerisine sürüklediğimizde, eğer gerekiyorsa projeye otomatik olarak dahil edileceklerdir. **riaControls** ön eki ile eklenen **DomainDataSource**elementi içerisinde kullanılan bazı nitelik verileri dikkate değerdir.

LoadSize niteliğine atanan değer, Silverlight uygulaması ilk yüklendiğinde çekilecek olan satır sayısını belirtmektedir. Gerçektende, uygulama bu haliyle çalıştırıldığında, **SQL Server Profiler** aracından yakalanan sorgu cümlesi aşağıdaki gibidir.

```
SELECT
[Limit1].[C1] AS [C1],
[Limit1].[ProductID] AS [ProductID],
[Limit1].[ProductName] AS [ProductName],
[Limit1].[SupplierID] AS [SupplierID],
[Limit1].[QuantityPerUnit] AS [QuantityPerUnit],
[Limit1].[UnitPrice] AS [UnitPrice],
[Limit1].[UnitsInStock] AS [UnitsInStock],
[Limit1].[UnitsOnOrder] AS [UnitsOnOrder],
[Limit1].[ReorderLevel] AS [ReorderLevel],
[Limit1].[Discontinued] AS [Discontinued],
[Limit1].[CategoryID] AS [CategoryID]
FROM ( SELECT TOP (10)
 [Extent1].[ProductID] AS [ProductID],
 [Extent1].[ProductName] AS [ProductName],
 [Extent1].[SupplierID] AS [SupplierID],
 [Extent1].[CategoryID] AS [CategoryID],
 [Extent1].[QuantityPerUnit] AS [QuantityPerUnit],
 [Extent1].[UnitPrice] AS [UnitPrice],
 [Extent1].[UnitsInStock] AS [UnitsInStock],
 [Extent1].[UnitsOnOrder] AS [UnitsOnOrder],
 [Extent1].[ReorderLevel] AS [ReorderLevel],
 [Extent1].[Discontinued] AS [Discontinued],
```

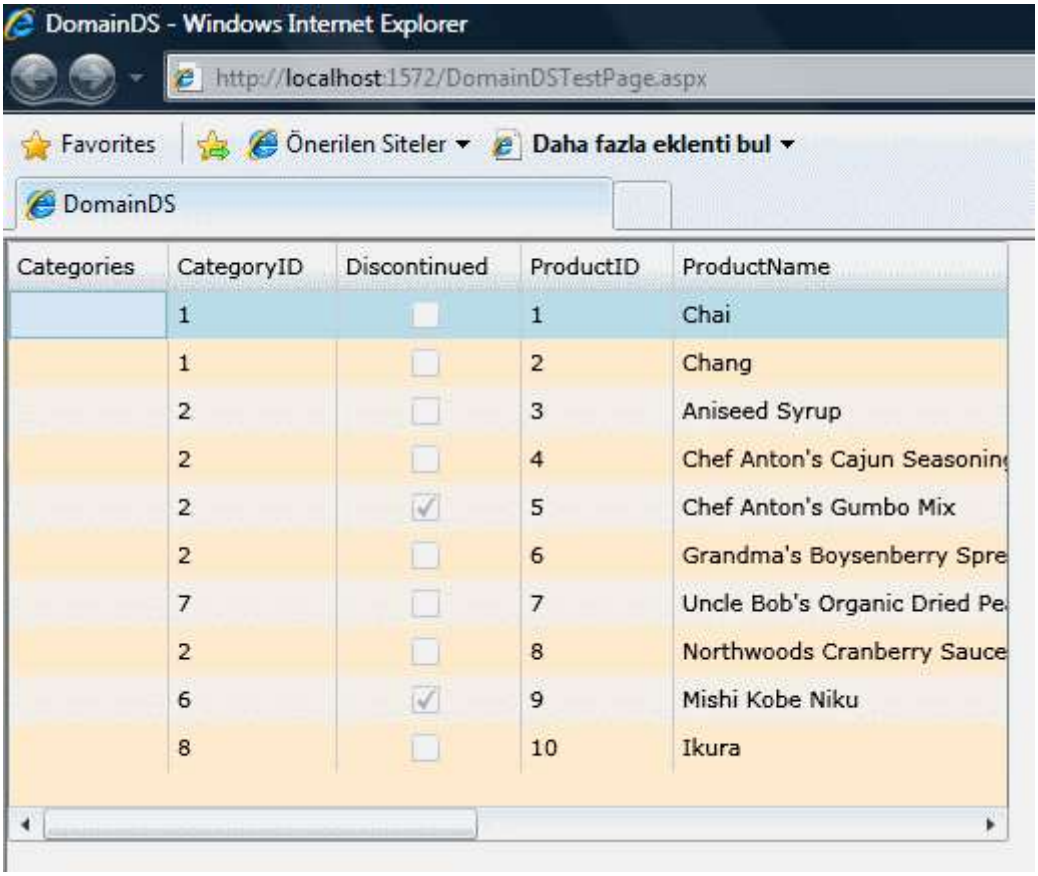
```
 1 AS [C1]
 FROM [dbo].[Products] AS [Extent1]
)  AS [Limit1]
```

Sanıyorumki **SELECT TOP (10)** ifadesi sizlerin dikkatinizden kaçmamıştır. Ancak **LoadSize** niteliği kaldırılırsa bu durumda SQL tarafında aşağıdaki sorgunun çalıştırıldığı görülecektir.

```
SELECT
1 AS [C1],
[Extent1].[ProductID] AS [ProductID],
[Extent1].[ProductName] AS [ProductName],
[Extent1].[SupplierID] AS [SupplierID],
[Extent1].[QuantityPerUnit] AS [QuantityPerUnit],
[Extent1].[UnitPrice] AS [UnitPrice],
[Extent1].[UnitsInStock] AS [UnitsInStock],
[Extent1].[UnitsOnOrder] AS [UnitsOnOrder],
[Extent1].[ReorderLevel] AS [ReorderLevel],
[Extent1].[Discontinued] AS [Discontinued],
[Extent1].[CategoryID] AS [CategoryID]
FROM [dbo].[Products] AS [Extent1]
```

Bu kez tüm Products tablosunun içeriği seçilmektedir. **LoadSize** özelliğini kullanarak, Silverlight uygulamasının ilk açılışı sırasındaki veri kümesinin yoğunluğunu kontrol altına alabilir ve performansı doğudan etkileyebiliriz. Burada dikkat çeken bir diğer önemli nitelik ise **LoadMethodName** niteliğine atanan değerdir. Bu değer, istemci tarafında kullanılan **DomainContext**tipinin içerisinde yer alan yükleme metodunun kendisidir. örneğimizde bu metod **LoadCategories** isimli fonksiyondur. Ancak fonksiyon adı text tabanlı olarak yazılmaktadır. Peki, **DomainDataSource**bileşeni, hangi **DataContext** nesne örneği içerisindeki **LoadCategories** metodunu kullanacağını nasıl bilecektir? 😕 Bu sorunun cevabı, **DomainDataSource.DomainContext** elementi içerisinde verilmektedir. Bu kısımda, **ds** ön
ekli **namespace** üzerinden **NorthwindContext** isimli **DomainContext** nesne referansının tanımlaması yapılmaktadır. Böylece, DomainDataSource bileşeninin kullanacağı DomainContext nesne örneği belirlenmiş olur.

DataGrid bileşeninin söz konusu **DomainDataSource** kontrolüne bağlanması içinse, **ItemsSource** niteliğine ilgili değerin atanması yeterlidir. (İtiraf etmeliyim ki, ItemsSource niteliğine atanan değerin yazım stilini, ne kadar dekleratif olsada halen ezbere yazamamaktayım 😐) Uygulamayı bu haliyle çalıştırdığımızda aşağıdaki ekran görüntüsü ile karşılaşmamız son derece muhtemeldir.



Burada önemli olan noktalardan birisi, tanımlamalarım tamamen dekleratif olarak yapılmış olması ve geliştiricinin herhangibir kodlama yapmamış olmasıdır. Daha önceki blog yazılarımda yer alan örnekler göz önüne aldığımızda, CRUD operasyonları için istemci tarafında bazı kodlamalar yaptığımız ortadadır. Şimdi XAML içeriğimizi biraz daha zengineştirmeye çalışalım. örneğin, sıralama kriteri ekleyebiliriz. Bunun için **SortDescriptor** elementinin kullanılması gerekmektedir. Bu element, **System.Windows.Ria.Controls.dll assembly'** ı içerisinde yer aldığında, isim alanının XAML içeriğinde bildirilmesi gerekir. Bu şekilde başlayan düzenlemelerin son hali aşağıdaki gibidir.

```
<UserControl xmlns:riaControls="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Ria.Controls"  xmlns:d
ata="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
   x:Class="DomainDS.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:ds="clr-namespace:DomainDS.Web"
   xmlns:riaData="clr-
namespace:System.Windows.Data;assembly=System.Windows.Ria.Controls"
   Width="500" Height="300">
   <Grid x:Name="LayoutRoot" Background="White">
      <riaControls:DomainDataSource x:Name="dsProducts"
```

```
AutoLoad="True" LoadSize="40" LoadMethodName="LoadProducts">
        <riaControls:DomainDataSource.DomainContext>
          <ds:NorthwindContext/>
        </riaControls:DomainDataSource.DomainContext>
        <riaControls:DomainDataSource.SortDescriptors>
          <riaData:SortDescriptor Direction="Descending"
PropertyPath="UnitPrice"/>
        </riaControls:DomainDataSource.SortDescriptors>
      </riaControls:DomainDataSource>
   <data:DataGrid x:Name="dgProducts" Height="Auto" Background="BlanchedAlmond"
ItemsSource="{Binding Data, ElementName=dsProducts}"/>
   </Grid>
</UserControl>
```

SortDescription elementi içerisinde yer alan **Direction** niteliğine atanan değer ile sıralamanın yönü belirtilmektedir. Diğer taraftan **PropertyPath** niteliğine atanan değer ilede hangi alana göre sıralama yapılacağına karar verilir. Bu ayarlamalara göre Products tablosundan ilk yüklemede **40** adet ürün bilgisi, **UnitPrice** değerlerine göre ters sırada çekilecektir. Nitekim SQL tarafında çalıştırılan sorguya bakıldığında aşağıdaki cümlenin çalıştırıldığı kolayca tespit edilebilir.

```
SELECT TOP (40)
[Project1].[C1] AS [C1],
[Project1].[ProductID] AS [ProductID],
[Project1].[ProductName] AS [ProductName],
[Project1].[SupplierID] AS [SupplierID],
[Project1].[QuantityPerUnit] AS [QuantityPerUnit],
[Project1].[UnitPrice] AS [UnitPrice],
[Project1].[UnitsInStock] AS [UnitsInStock],
[Project1].[UnitsOnOrder] AS [UnitsOnOrder],
[Project1].[ReorderLevel] AS [ReorderLevel],
[Project1].[Discontinued] AS [Discontinued],
[Project1].[CategoryID] AS [CategoryID]
FROM ( SELECT [Project1].[ProductID] AS [ProductID], [Project1].[ProductName] AS
[ProductName], [Project1].[SupplierID] AS [SupplierID], [Project1].[CategoryID] AS
[CategoryID], [Project1].[QuantityPerUnit] AS [QuantityPerUnit], [Project1].[UnitPrice]
AS [UnitPrice], [Project1].[UnitsInStock] AS [UnitsInStock], [Project1].[UnitsOnOrder]
AS [UnitsOnOrder], [Project1].[ReorderLevel] AS [ReorderLevel],
[Project1].[Discontinued] AS [Discontinued], [Project1].[C1] AS [C1], row_number()
OVER (ORDER BY [Project1].[UnitPrice] DESC) AS [row_number]
 FROM ( SELECT
  [Extent1].[ProductID] AS [ProductID],
  [Extent1].[ProductName] AS [ProductName],
  [Extent1].[SupplierID] AS [SupplierID],
  [Extent1].[CategoryID] AS [CategoryID],
```

```
  [Extent1].[QuantityPerUnit] AS [QuantityPerUnit],
  [Extent1].[UnitPrice] AS [UnitPrice],
  [Extent1].[UnitsInStock] AS [UnitsInStock],
  [Extent1].[UnitsOnOrder] AS [UnitsOnOrder],
  [Extent1].[ReorderLevel] AS [ReorderLevel],
  [Extent1].[Discontinued] AS [Discontinued],
  1 AS [C1]
  FROM [dbo].[Products] AS [Extent1]
 )  AS [Project1]
)  AS [Project1]
WHERE [Project1].[row_number] > 40
ORDER BY [Project1].[UnitPrice] DESC
```

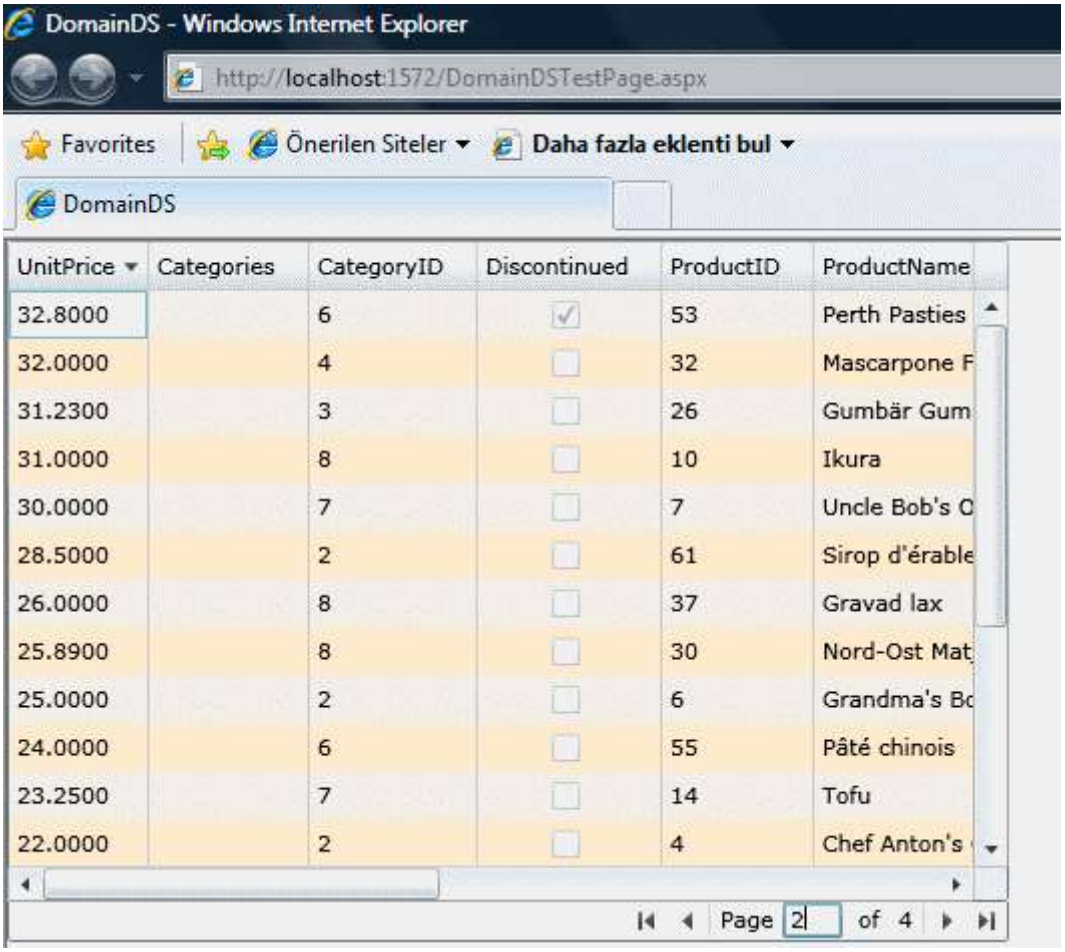
Oldukça kolay gördüğünüz gibi. **.Net RIA Servislerini** sisteme yüklediğimizde gelen dökümantasyon içerisinde bu tip bir örnek yapılmaktadır. İlerleyen kısımlarında, verinin çekilmesi işlemi sırasında kullanılabilecek **sayfalama(Paging) ve filtreleme(Filtering)** seçenekleride örneğe dahil edilmektedir. Size tavsiyem söz konusu dökümantasyonda yer alan örneği incelemeniz olacaktır.

Ben yazımı sonlandırmadan önce sayfalama kriterinide **XAML** içeriğine dahil etmeye çalışacağım. Bu amaçla, **DataPager** isimli Silverlight bileşenini XAML içerisine sürüklememiz yeterli olacaktır. MainPage.xaml içeriğinin son hali aşağıdaki gibidir.

```
<UserControl xmlns:dataControls="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data.DataFor
m"  xmlns:riaControls="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Ria.Controls"  xmlns:d
ata="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
   x:Class="DomainDS.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:ds="clr-namespace:DomainDS.Web"
   xmlns:riaData="clr-
namespace:System.Windows.Data;assembly=System.Windows.Ria.Controls"
   Width="500" Height="350">
   <StackPanel x:Name="LayoutRoot" Background="White" Orientation="Vertical">
      <riaControls:DomainDataSource x:Name="dsProducts" AutoLoad="True"
LoadSize="40" LoadMethodName="LoadProducts">
         <riaControls:DomainDataSource.DomainContext>
            <ds:NorthwindContext/>
         </riaControls:DomainDataSource.DomainContext>
         <riaControls:DomainDataSource.SortDescriptors>
            <riaData:SortDescriptor Direction="Descending" PropertyPath="UnitPrice"/>
         </riaControls:DomainDataSource.SortDescriptors>
```

```
        </riaControls:DomainDataSource>
    <data:DataGrid x:Name="dgProducts" Height="330" Background="BlanchedAlmond"
ItemsSource="{Binding Data, ElementName=dsProducts}"/>
    <dataControls:DataPager PageSize="20" Height="20" Source="{Binding
Data,ElementName=dsProducts}"></dataControls:DataPager>
    </StackPanel>
</UserControl>
```

DataPager kontrolü doğal olarak kimi(yani hangi veri kaynağını) sayfalayacağını bilmek zorundadır. Bu nedenle **Source** niteliğine **dsProducts** isimli **DomainDataSource** bileşeni atanmıştır. Diğer taraftan **PageSize** niteliğine atanan değer ile her sayfada **20** adet satırın gösterileceği belirtilmektedir. Uygulamayı bu haliyle çalıştırdığımızda aşağıdakine benzer bir ekran görüntüsü ile karşılaşmamız muhtemeldir.



Burada dikkat çeken noktalardan biriside **LoadSize** ile ilk etapta **40** satırın yüklenmesine rağmen, sayfalama içerisinde **en çok 80 (4X20)** kaydın gösterilebilecek olmasıdır. Bu aslında kayda değer ve incelenmesi gereken bir durumdur. Nitekim SQL tarafında çalıştırılan sorgu cümelelerine dikkatlice bakmak gerekmektedir. İşte eğlence başlıyor.

Sayfa ilk yüklendiğinde **TOP 40** ile **40** satırlık bir veri bloğunun yüklenmesi sağlanır. **PageSize** değeri **20** olarak berlilendiğinden 1nci sayfadan 2nci sayfaya

geçtiğimizde, SQL tarafında herhangibir sorgu çalıştırılmadığı gözlemlenir.(İyi bir gelişme) Ancak 3ncü sayfaya geçmek istediğimizde, 40 satırlık yükleme boyutunu geçtiğimiz için sunucu tarafında yeni bir SQL sorgusu çalıştırılacak ve **row_number** değeri **40'** ın üzerinde olanlar talep edilecektir. Aşağıdaki SQL cümlesinde görüldüğü gibi...

```
SELECT TOP (40)
[Project1].[C1] AS [C1],
[Project1].[ProductID] AS [ProductID],
[Project1].[ProductName] AS [ProductName],
[Project1].[SupplierID] AS [SupplierID],
[Project1].[QuantityPerUnit] AS [QuantityPerUnit],
[Project1].[UnitPrice] AS [UnitPrice],
[Project1].[UnitsInStock] AS [UnitsInStock],
[Project1].[UnitsOnOrder] AS [UnitsOnOrder],
[Project1].[ReorderLevel] AS [ReorderLevel],
[Project1].[Discontinued] AS [Discontinued],
[Project1].[CategoryID] AS [CategoryID]
FROM ( SELECT [Project1].[ProductID] AS [ProductID], [Project1].[ProductName] AS
[ProductName], [Project1].[SupplierID] AS [SupplierID], [Project1].[CategoryID] AS
[CategoryID], [Project1].[QuantityPerUnit] AS [QuantityPerUnit], [Project1].[UnitPrice]
AS [UnitPrice], [Project1].[UnitsInStock] AS [UnitsInStock], [Project1].[UnitsOnOrder]
AS [UnitsOnOrder], [Project1].[ReorderLevel] AS [ReorderLevel],
[Project1].[Discontinued] AS [Discontinued], [Project1].[C1] AS [C1], row_number()
OVER (ORDER BY [Project1].[UnitPrice] DESC) AS [row_number]
 FROM ( SELECT
  [Extent1].[ProductID] AS [ProductID],
  [Extent1].[ProductName] AS [ProductName],
  [Extent1].[SupplierID] AS [SupplierID],
  [Extent1].[CategoryID] AS [CategoryID],
  [Extent1].[QuantityPerUnit] AS [QuantityPerUnit],
  [Extent1].[UnitPrice] AS [UnitPrice],
  [Extent1].[UnitsInStock] AS [UnitsInStock],
  [Extent1].[UnitsOnOrder] AS [UnitsOnOrder],
  [Extent1].[ReorderLevel] AS [ReorderLevel],
  [Extent1].[Discontinued] AS [Discontinued],
  1 AS [C1]
  FROM [dbo].[Products] AS [Extent1]
 )  AS [Project1]
)  AS [Project1]
WHERE [Project1].[row_number] > 40
ORDER BY [Project1].[UnitPrice] DESC
```

Ne yazıkki 20 satır veri çekilmesi gerekmesine rağmen **LoadSize** özelliği nedeniyle **Top 40** kullanımı söz konusudur. (Bu açıkçası benim pek beklediğim bir durum değildi.)

Peki 4ncü sayfaya geçmek istersek ne olacaktır? Bu durumda **row_number** değeri **60'** ın (3X20 veya 3ncü sayfa X PageSize) üzerinde olan veriler çekilmeye çalışılacaktır.

```
SELECT TOP (40)
[Project1].[C1] AS [C1],
[Project1].[ProductID] AS [ProductID],
[Project1].[ProductName] AS [ProductName],
[Project1].[SupplierID] AS [SupplierID],
[Project1].[QuantityPerUnit] AS [QuantityPerUnit],
[Project1].[UnitPrice] AS [UnitPrice],
[Project1].[UnitsInStock] AS [UnitsInStock],
[Project1].[UnitsOnOrder] AS [UnitsOnOrder],
[Project1].[ReorderLevel] AS [ReorderLevel],
[Project1].[Discontinued] AS [Discontinued],
[Project1].[CategoryID] AS [CategoryID]
FROM ( SELECT [Project1].[ProductID] AS [ProductID], [Project1].[ProductName] AS
[ProductName], [Project1].[SupplierID] AS [SupplierID], [Project1].[CategoryID] AS
[CategoryID], [Project1].[QuantityPerUnit] AS [QuantityPerUnit], [Project1].[UnitPrice]
AS [UnitPrice], [Project1].[UnitsInStock] AS [UnitsInStock], [Project1].[UnitsOnOrder]
AS [UnitsOnOrder], [Project1].[ReorderLevel] AS [ReorderLevel],
[Project1].[Discontinued] AS [Discontinued], [Project1].[C1] AS [C1], row_number()
OVER (ORDER BY [Project1].[UnitPrice] DESC) AS [row_number]
 FROM ( SELECT
  [Extent1].[ProductID] AS [ProductID],
  [Extent1].[ProductName] AS [ProductName],
  [Extent1].[SupplierID] AS [SupplierID],
  [Extent1].[CategoryID] AS [CategoryID],
  [Extent1].[QuantityPerUnit] AS [QuantityPerUnit],
  [Extent1].[UnitPrice] AS [UnitPrice],
  [Extent1].[UnitsInStock] AS [UnitsInStock],
  [Extent1].[UnitsOnOrder] AS [UnitsOnOrder],
  [Extent1].[ReorderLevel] AS [ReorderLevel],
  [Extent1].[Discontinued] AS [Discontinued],
  1 AS [C1]
  FROM [dbo].[Products] AS [Extent1]
 )  AS [Project1]
)  AS [Project1]
WHERE [Project1].[row_number] > 60
ORDER BY [Project1].[UnitPrice] DESC
```

Yinede TOP 40 oluşumu söz konusudur. Ancak istediğimiz sonuç alınmıştır. Sayfalama işlemide başarılı bir şekilde gerçekleştirilmiştir.

Böylece geldik bir blog yazımızın daha sonuna. Şimdi müsadenizle biraz dinlenmeye çekileceğim. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

.Net RIA Servisleri - CRUD İşlemleri (2009-05-14T07:50:00)

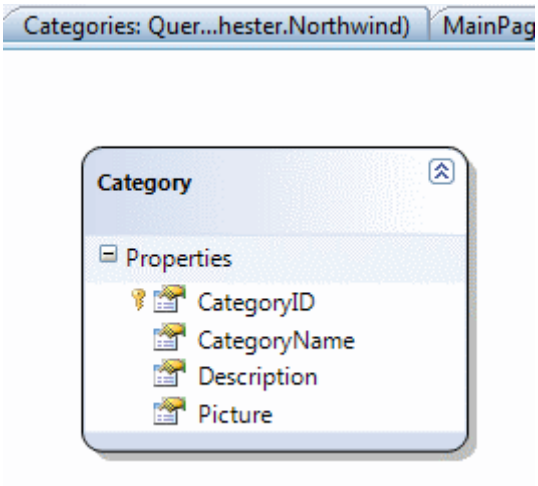
.net ria services,silverlight,

Merhaba Arkadaşlar,

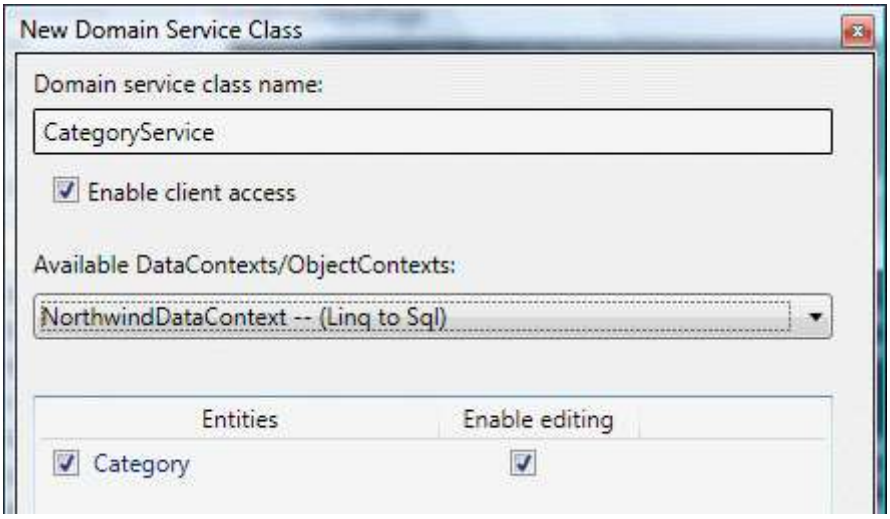
Bildiğiniz gibi bir süredir .Net RIA Servisleri ile ilişkili araştırmalarıma devam etmekteyim. Bu yazımızda, .Net RIA Servislerinde **Insert, Update** ve **Delete** işlemlerini nasıl yapabileceğimizi basit bir örnek üzerinden adım adım aktarmaya çalışacağım. Daha önceki Hello World örneğimizden farklı olarak, DAL(Data Access Layer) içerisinde **LINQ to SQL** modelini kullanıyor olacağız. İlk adımımız elbetteki bir **Silverlight Application** projesi oluşturmak olmalıdır. .Net RIA Servisini kullanacağımız için, projenin oluşturulması sırasında **Link to ASP.NET Server Project** seçeneğinin işaretli olmasına dikkat edelim. Sonrasında Web projesine bir adet **LINQ to SQL** öğesi eklemeli ve bağlanmak istediğimiz veri kaynağı üzerinden, kullanmak istediğimiz tablo veya stored procedure' leri diagram üzerine sürüklemeliyiz. Ben Insert, Update ve Delete işlemlerini çok basit bir şekilde ele almak istediğimdeN kullanabileceğim en kolay kobay tabloyu seçtim :) Northwind veritabanında yer alan Categories tablosu. Nitekim sadece CategoryName ve Description alanlarına veri eklemek bizim için yeterli olacaktır.

NOT : Ancak örneği biraz daha ileri seviyede geliştirmeye çalışmanızıda şiddetle tavsiye ederim. Söz gelimi, Categories tablosunda Image tipinden Picture isimli bir alan bulunmaktadır. Bu resim alanı binary tiptedir. Bir başka deyişle Silverlight istemcisinin, kategori resmini seçip sunucu tarafına binary formatta aktarmaya çalışmasıda iyi bir mücadele antrenmanı olarak göz önüne alınabilir. Hatta resmin istemci tarafında bir kontrol içerisinde gösterilmeside söz konusu olabilir.

LINQ to SQL diagramımızın içeriği aşağıdaki şekilde görüldüğü gibi olacaktır.



Bundan sonra yapmamız gereken, DomainService tipinin eklenmesidir. Bu seferki örneğimizde, tüm CRUD operasyonuna ihtiyacımız olacağından, Categories Entity' si için, **Enable Editing**özelliğinin etkinleştirilmiş olması şarttır.



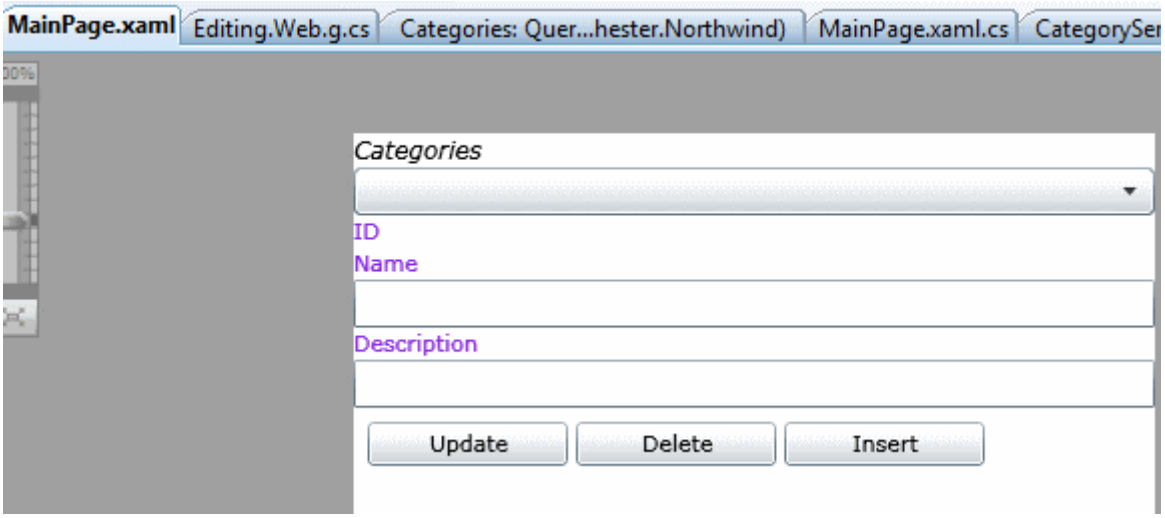
Bu işlemlerin arkasından CategoryService isimli DomainService sınıfı içerisinde aşağıdaki fonksiyonelliklerin oluşturulduğu görülür.

```
namespace Editing.Web
{
   using System.Linq;
   using System.Web.DomainServices.LinqToSql;
   using System.Web.Ria;

   [EnableClientAccess()]
   public class CategoryService : LinqToSqlDomainService<NorthwindDataContext>
   {
      public IQueryable<Category> GetCategories()
      {
         return this.Context.Categories;
      }

      public void InsertCategory(Category category)
      {
         this.Context.Categories.InsertOnSubmit(category);
      }

      public void UpdateCategory(Category currentCategory, Category originalCategory)
      {
         this.Context.Categories.Attach(currentCategory, originalCategory);
      }

      public void DeleteCategory(Category category)
      {
         this.Context.Categories.Attach(category, category);
         this.Context.Categories.DeleteOnSubmit(category);
      }
```

```
    }
}
```

GetCategories metodu ile LINQ to SQL sağlayıcısı üzerinden kategorilerin çekilmesi sağlanmaktadır. Bir kategorinin eklenmesi sırasında **InsertOnSubmit**, silinmesi işleminde **DeleteOnSubmit**ve son olarak güncellenmesinde ise **Attach** isimli LINQ to SQL tarafından hazır olarak gelen fonksiyonların kullanıldığı görülmektedir. Tabiki projenin Build edilmesi sonrasında, istemci tarafındada uygun metodları içeren **CategoryContext** isimli **DomainContext** tipi ile sunucu tarafındaki Category entity sınıfının karşılığı hazırlanmış olacaktır. Artık tek yapmamız gereken istemci tarafını tasarlamak ve kodlamaktır. Ben tasarım konusunda özürlü olduğumdan, ancak aşağıdaki **Silverlight UserControl** bileşenini oluşturabilmiş bulunuyorum.

MainPage.xaml




```
<UserControl x:Class="Editing.MainPage"
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    Width="400" Height="300">
    <StackPanel x:Name="LayoutRoot" Background="White" Orientation="Vertical">
        <TextBlock Text="Categories" FontSize="12" FontStyle="Italic"/>
        <ComboBox x:Name="cmbCategories" Height="24"
SelectionChanged="cmbCategories_SelectionChanged">
            <ComboBox.ItemTemplate>
                <DataTemplate>
                    <StackPanel Orientation="Vertical">
                        <TextBlock x:Name="id" Text="{Binding CategoryID}"
Foreground="RoyalBlue"/>
                        <TextBlock x:Name="name" Text="{Binding CategoryName}"
Foreground="SeaGreen"/>
                        <TextBlock x:Name="description" Text="{Binding Description}"
Foreground="Salmon" FontSize="9" FontStyle="Italic"/>
```

```
            </StackPanel>
          </DataTemplate>
        </ComboBox.ItemTemplate>
      </ComboBox>
      <TextBlock Text="ID"  Foreground="BlueViolet"/>
      <TextBlock x:Name="txtCategoryID"/>
      <TextBlock Text="Name" Foreground="BlueViolet"/>
      <TextBox x:Name="txtCategoryName"/>
      <TextBlock Text="Description" Foreground="BlueViolet"/>
      <TextBox x:Name="txtCategoryDescription"/>
      <StackPanel Orientation="Horizontal" Margin="5">
        <Button x:Name="btnUpdate" Click="btnUpdate_Click" Content="Update"
Width="100" Margin="2"/>
        <Button x:Name="btnDelete" Click="btnDelete_Click" Content="Delete"
Width="100" Margin="2"/>
        <Button x:Name="btnInsert" Click="btnInsert_Click" Content="Insert"
Width="100" Margin="2"/>
      </StackPanel>
  </StackPanel>
</UserControl>
```

Hemen bu sayfadanın tasarlanma amacını açıklıyayım. ComboBox kontrolümüz içerisinde, sayfanın oluşturulması sırasında yüklenen Category nesne örnekleri yer alacaktır. Bu nesne örneklerine ait CategoryID,CategoryName ve Description alanları **DataTemplate** şablonun içerisindeki TextBlock kontrollerinin **Text** özelliklerine **bağlanmıştır(Binding)**. Kullanıcı isterse, sayfanın alt kısmında yer alan TextBox kontrollerini kullanarak yeni bir Category ekleyebilir. Tek yapması gereken, Insert başlıklı Button kontrolüne basmaktır. Diğer taraftan ComboBox kontrolünde bir Category seçildiğinde, buna ait CategoryName ve Description bilgileri ile CategoryID değeri, alt tarafta yer alan kontrollere, istemci tarafındaki **DomainContext** tipinin ilgili **Categories**özelliği üzerinden getirilmektedir. Kullanıcı bu işleyişi güncelleme sırasında değerlendirebilir. Bilgiler üzerinde gerekli değişiklikleri yaptıktan sonra **Update** düğmesini kullanması yeterlidir. Benzer şekilde silme işlemi içinde sadece ve sadece Delete düğmesini ele alabilir.

NOT: Tabi bu örnekte, silinmek istenen Category bilgisinin, sunucu tarafında başka birisi tarafından silinmiş olma durumu söz konusu olabilir. Yada var olan bir kaydı güncellemek isteyen kullanıcıdan önce başka birisi güncelleştirmiş olabilir ve o anki kullanıcı eski veriye bakıyor olabilir. Sanıyorumki nereye varmak istediğimi biraz anladınız. Eş zamanlı olarak birbirlerinden habersiz bir şekilde veriyi ektiliyen istemcilerin olduğu vaka. Bu tip bir vaka .Net RIA Servislerinin kullanıldığı bir ortamda son derece olasıdır. önemli olan noktalardan birisi, SQL tarafında çalıştırılan sorgulardaki Where kriteridir. Where kriterinde varsayılan olarak nasıl bir yaklaşım sergilenmektedir? Bunu ilerleyen kısımlarda görmeye çalışacağız.

Gelelim kod tarafına...

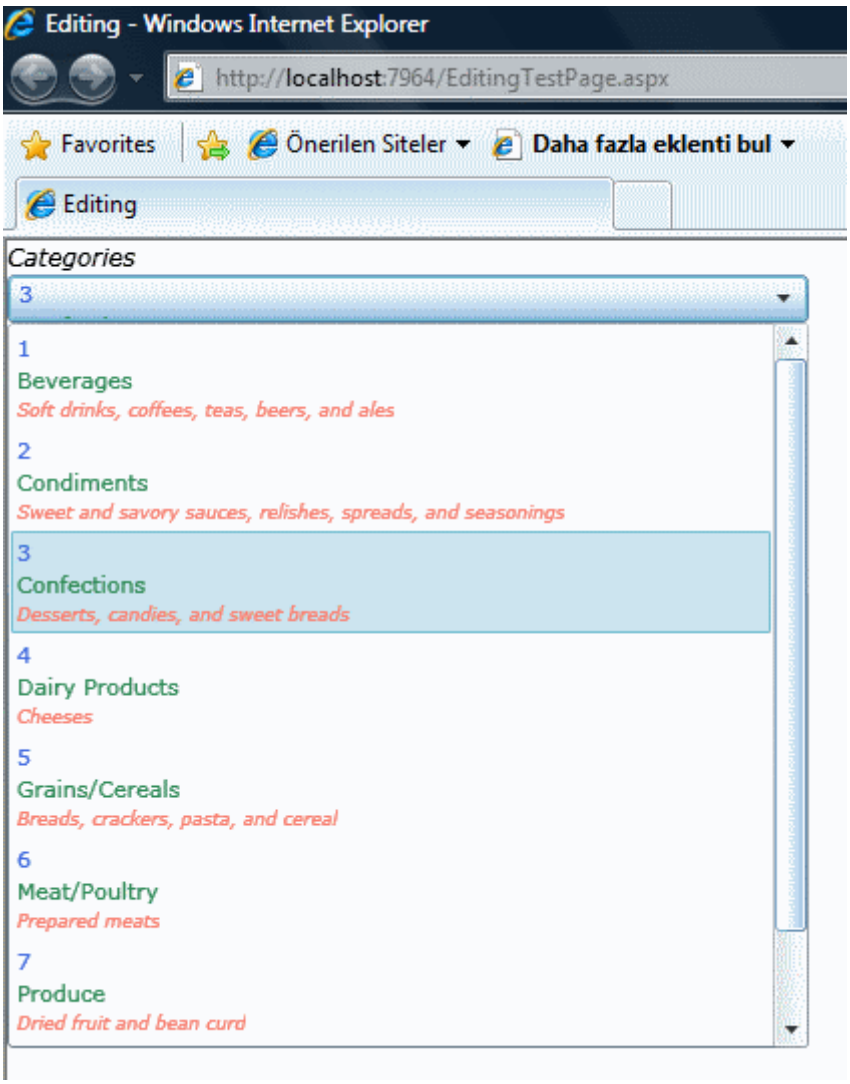
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Animation;
using System.Windows.Shapes;
using Editing.Web;

namespace Editing
{
   public partial class MainPage : UserControl
   {
      CategoryContext context = new CategoryContext();

      public MainPage()
      {
         InitializeComponent();

         cmbCategories.ItemsSource = context.Categories;
         context.LoadCategories();
      }

      private void cmbCategories_SelectionChanged(object sender,
SelectionChangedEventArgs e)
      {
         if (e.AddedItems.Count>0
            && e.AddedItems[0]!=null)
         {
            Category selectedCategory = (Category)e.AddedItems[0];

            txtCategoryID.Text = selectedCategory.CategoryID.ToString();
            txtCategoryName.Text = selectedCategory.CategoryName;
            txtCategoryDescription.Text = selectedCategory.Description;
         }
      }


      private void btnUpdate_Click(object sender, RoutedEventArgs e)
      {
         // Güncellenmek istenen category nesne örneği, DataContext referansının
```

```
Categories koleksiyonu üzerinden çekilir.
        if (!String.IsNullOrEmpty(txtCategoryID.Text))
        {
            Category category = context.Categories.Single<Category>(c =>
c.CategoryID == Convert.ToInt32(txtCategoryID.Text));

            // Güncellenmek istenen Category nesne örneğinin CategoryName ve Description
alanlarına ilgili değerler aktarılır
            category.CategoryName = txtCategoryName.Text;
            category.Description = txtCategoryDescription.Text;
        }

        // Değişiklikler sunucu tarafına gönderilir.
        context.SubmitChanges();
    }

    private void btnDelete_Click(object sender, RoutedEventArgs e)
    {
        // Silinmek istenen Category tipi ComboBox içerisinden seçildikten sonra
        // ilk olarak DataContext tipi içerisindeki koleksiyondan çıkartılır.
        context.Categories.Remove((Category)cmbCategories.SelectedItem);

        // Değişiklikler sunucu tarafına gönderilir
        context.SubmitChanges();

        // Silme işleminden sonra sunucu tarafından Categories tablosunun son içeriği alınır
        context.LoadCategories();

        // Kontrollerin içeriği temizlenir
        txtCategoryDescription.Text = "";
        txtCategoryName.Text = "";
        txtCategoryID.Text = "";
    }

    private void btnInsert_Click(object sender, RoutedEventArgs e)
    {
        // Yeni Category tipi örneklenir
        Category newCategory = new Category {
            CategoryName = txtCategoryName.Text
            , Description = txtCategoryDescription.Text };

        // örneklenen Category tipi, DataContext üzerindeki Categories koleksiyonuna
eklenir.
        context.Categories.Add(newCategory);
```

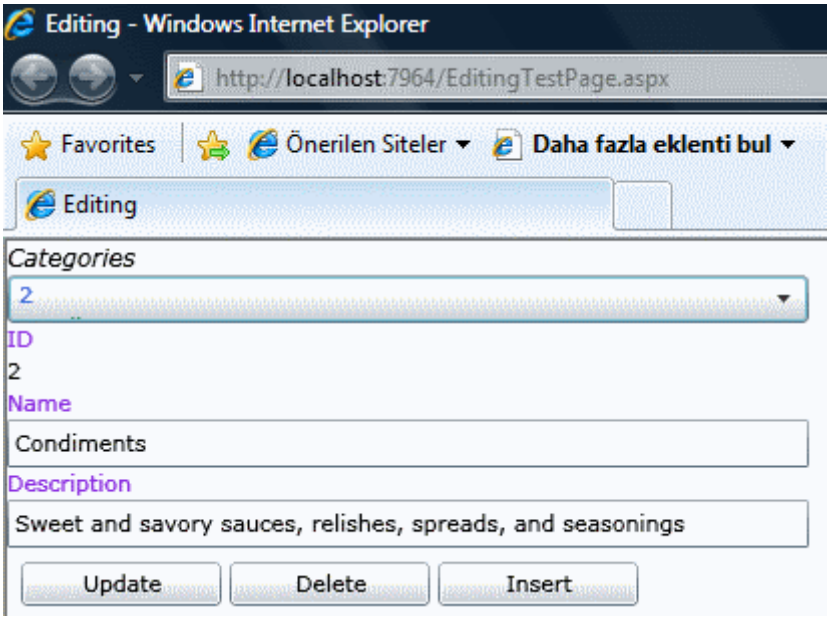
```
            // Değişiklikler onaylanır ve sunucu tarafına aktarılır
            context.SubmitChanges();
        }
    }
}
```

Artık testlerimize başlayabiliriz. Burada **Insert, Update** ve **Delete** gibi işlemler söz konusu olduğundan ve sunucu tarafında SQL kullanıldığından, arka planda çalıştırılan sorgu cümlelerini eminimki sizde en az benim ettiğim kadar merak ediyorsunuzdur. Bu nedenle **SQL Server Profiler** aracımızda bir yandan açık duruyor olacak.

İlk karşılaşacağımız ekran görüntüsü aşağıdakine benzer olacaktır.



Görüldüğü gibi ComboBox içerisinde, kategorilerin tamamı yer almaktadır. Eğer kullanıcı herhangibir öğeyi seçerse aşağıdaki ekran görüntüsünde olduğu gibi, o kategoriye ait bilgiler gelecektir.

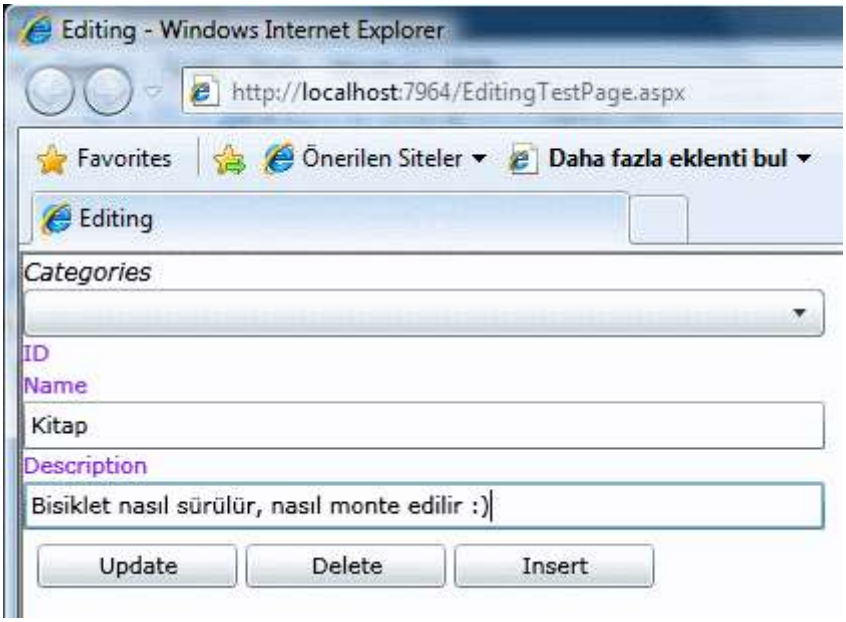


Bu sırada ekrana gelen veri içeriğini değiştirdiğimizi ve sonrasında Update tuşuna bastığımızı düşünelim. örnek olarak kategori adı ve açıklamalarının sonuna üç nokta koyduğumuzu varsayalım. Bu durumda SQL tarafında aşağıdaki sorgunun çalıştırıldığını görebiliriz.

```
exec sp_executesql N'UPDATE [dbo].[Categories]
SET [CategoryName] = @p2, [Description] = @p3
WHERE ([CategoryID] = @p0) AND ([CategoryName] = @p1)',N'@p0 int,@p1
nvarchar(10),@p2 nvarchar(13),@p3
ntext',@p0=2,@p1=N'Condiments',@p2=N'Condiments...',@p3=N'Sweet and savory
sauces, relishes, spreads, and seasonings...'
```

Hemen dikkatimi çeken bir noktayı vurgulamak istiyorum. Sadece CategoryName ve Description alanlarını güncelleştirdik. Bu nedenle SQL tarafında yürütülen Update sorgusunda yanlızca bu alanlar yer almaktadır. Picture alanı için herhangibir ifade bulunmamaktadır. Bu neden önemlidir? n tane alandan oluşan bir tablonun kullanıldığı düşünüldüğünde, sadece bir alan için güncelleştirme yapılıyorsa, tüm alanların sorguya dahil edilmesi yerine sadece ilgili olanın eklenmesi söz konusudur...Mu acaba? Bunu test etmek son derece kolaydır aslında. Sadece CategoryName alanın değerini güncelleştirdiğinizi düşünelim. Bu durumda SQL profiler ile yaklanan sorguya bakarsak eğer, Description özelliğine, txtCategoryDescription kontrolünün değişmeyen içeriğini aktarmış olsak bile, sadece CategoryName alanının sorguya dahil edildiğini görebiliriz. Bu bizim için oldukça iyi bir haber aslında.

Yeni bir kategori eklenmek istendiğindeyse,

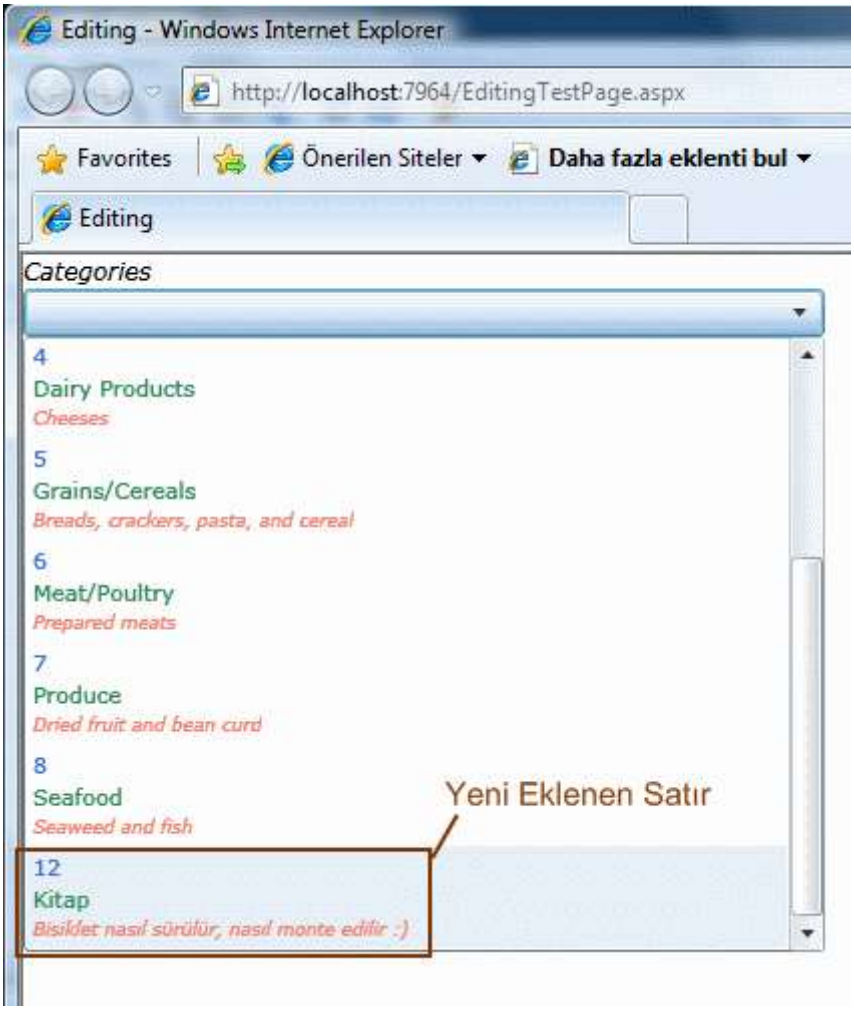


sunucu tarafında aşağıdaki SQL sorgusu çalıştırılacaktr.

exec sp_executesql N'INSERT INTO [dbo].[Categories]([CategoryName], [Description], [Picture])
VALUES (@p0, @p1, @p2)

SELECT CONVERT(Int,SCOPE_IDENTITY()) AS [value]',N'@p0 nvarchar(5),@p1 ntext,@p2 image',@p0=N'Kitap',@p1=N'Bisiklet nasıl sürülür, nasıl monte edilir :)',@p2=NULL

Sorgu cümlesinde standart bir Insert ifadesi olmasının dışında, eklenen kayıt için üretilen Identity değerinin, **SCOPE_IDENTITY()** fonksiyonundan yararlanılarak geriye döndürüldüğü gözden kaçırılmamalıdır. öyleki, yeni eklenen satıra ait bilgiler ComboBox kontrolüne otomatik olarak bağlanırken, CategoryID değerininde sunucudan alındığı rahatlıkla gözlemlenebilir.



Silme işlemi için bir kategorinin seçilmesi gerekmektedir. Seçilen kategoriye ait Category nesne örneği bulunduktan sonra ise **Remove** metodu ile **DomainContext** içerisindeki koleksiyondan çıkartılır. Sonrasında ise değişikleri sunucu göndermek için yine **SubmitChanges** metodundan yararlanılır. Sonuç itibariyle SQL sunucusuna giden sorgu cümlesi aşağıdaki gibidir.

```
exec sp_executesql N'DELETE FROM [dbo].[Categories] WHERE ([CategoryID] = @p0) AND ([CategoryName] = @p1)',N'@p0 int,@p1 nvarchar(5)',@p0=12,@p1=N'Kitap'
```

Sorguda dikkat çeken en önemli nokta Where kriterine, **Image** tipinden olan **Picture** ile **ntext** tipinden olan **Description** dışındaki tüm alanların dahil edilmesidir. Bu bir anlamda eş zamanlı çakışmaların önüne geçilmesini sağlamaktadır ki aynı durum Update için çalıştırılan SQL sorgusunda da geçerlidir. Bilmem farketmiş miydiniz? 😃

Böylece geldik bir yazımızın daha sonuna. Bu kısa yazıda, .**Net RIA Servislerinde Insert,Update** ve **Delete** işlemlerini basit bir biçimde ele almaya ve arka planda hareket eden SQL cümleciklerine bakıldığında gözümüze çarpan önemli noktaları vurgulamaya çalıştım. .Net RIA Servisleri ile ilişkiki araştırmalarıma devam ettikçe sizlerle paylaşıyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

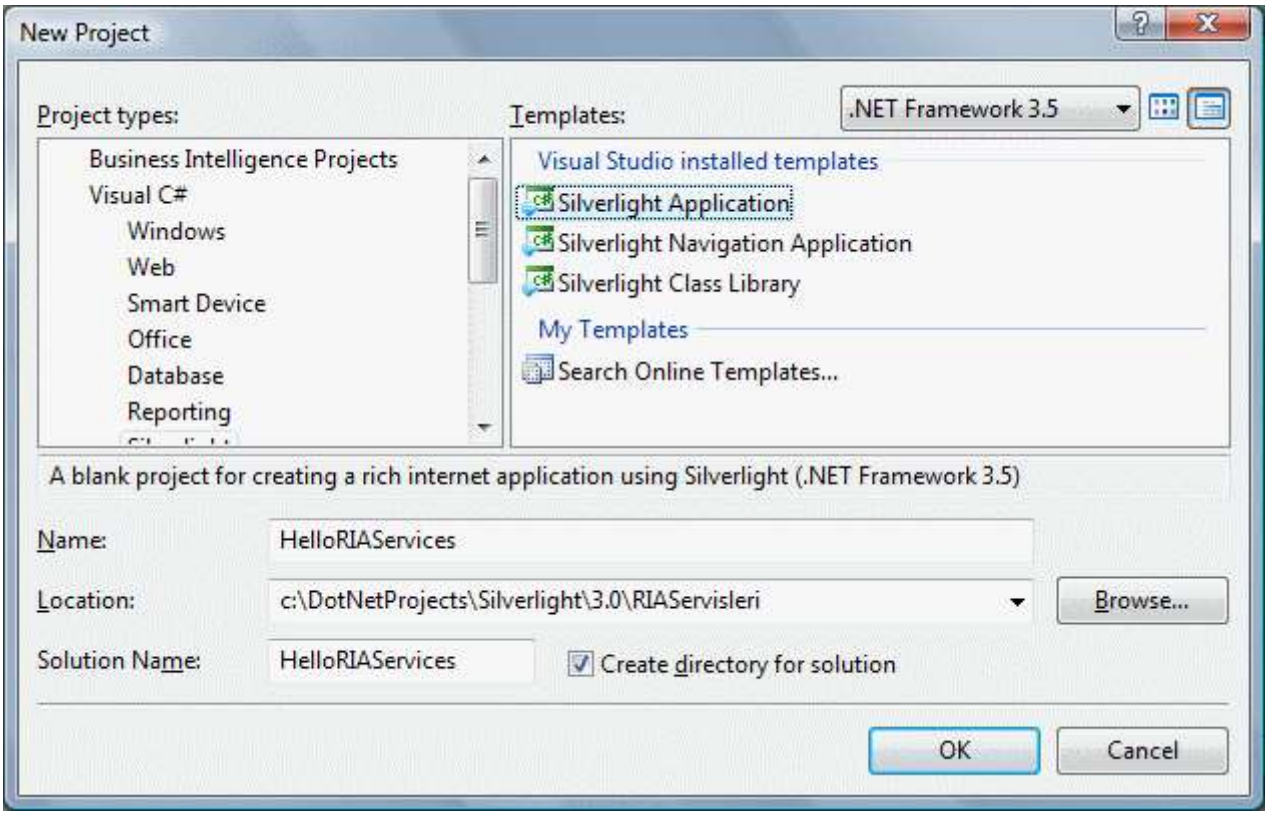
Editing.rar (1,14 mb)

.Net RIA Servisleri - Hello World (2009-05-13T22:29:00)

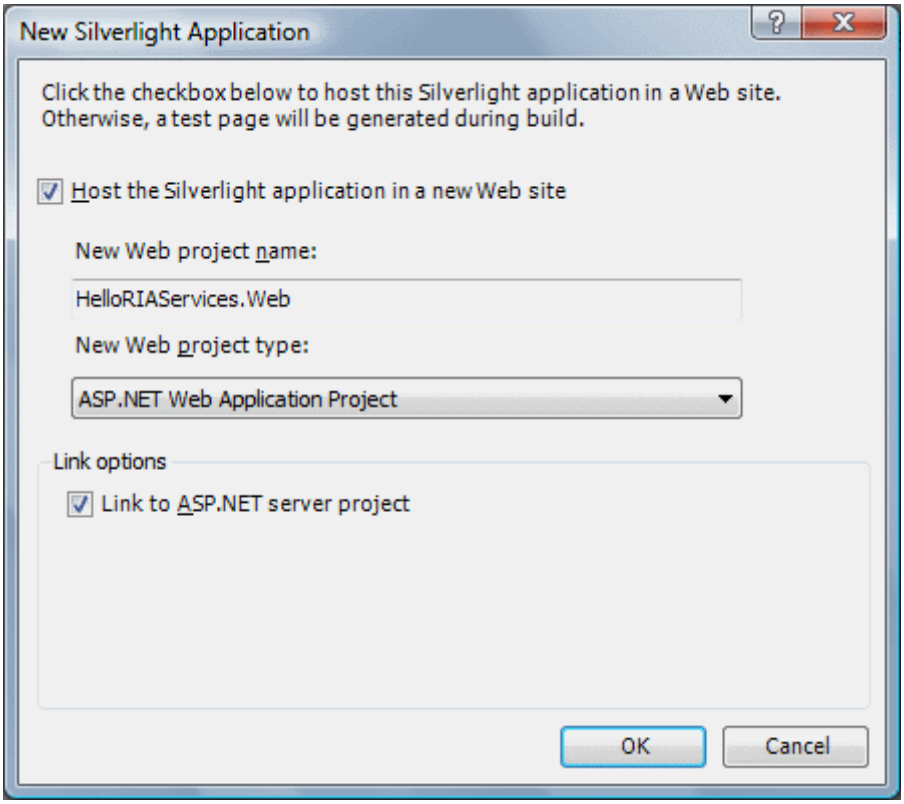
.net ria services,silverlight,

Merhaba Arkadaşlar,

Hatırlayacağınız gibi bir önceki blog yazımda, **.Net RIA Servisleri** hakkında edindiğim kısa ve özet teorik bilgileri sizinle paylaşmaya çalışmıştım. Bu yazımda ise, teoriği pratiğe dökmeye gayret edeceğim. Geliştireceğimiz örnek, .Net RIA Servisini kullanan bir **Silverlight** uygulaması olacak. Geliştirmeyi **Visual Studio 2008** üzerinde, **Silverlight 3.0** ortamını kullanarak gerçekleştireceğim. Bu nedenle aşağıdaki şekilde görüldüğü gibi, klasik bir silverlight projesi oluşturarak işe başlayabiliriz.

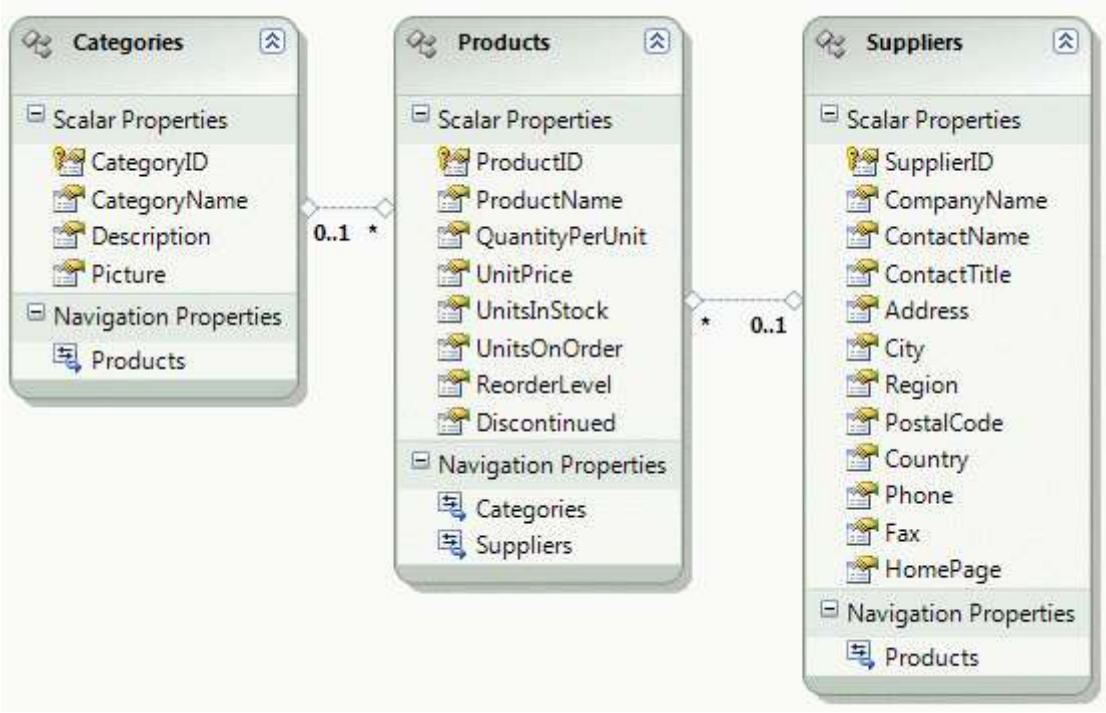


Bu işlemin ardından ekrana gelen aşağıdaki pencerede,

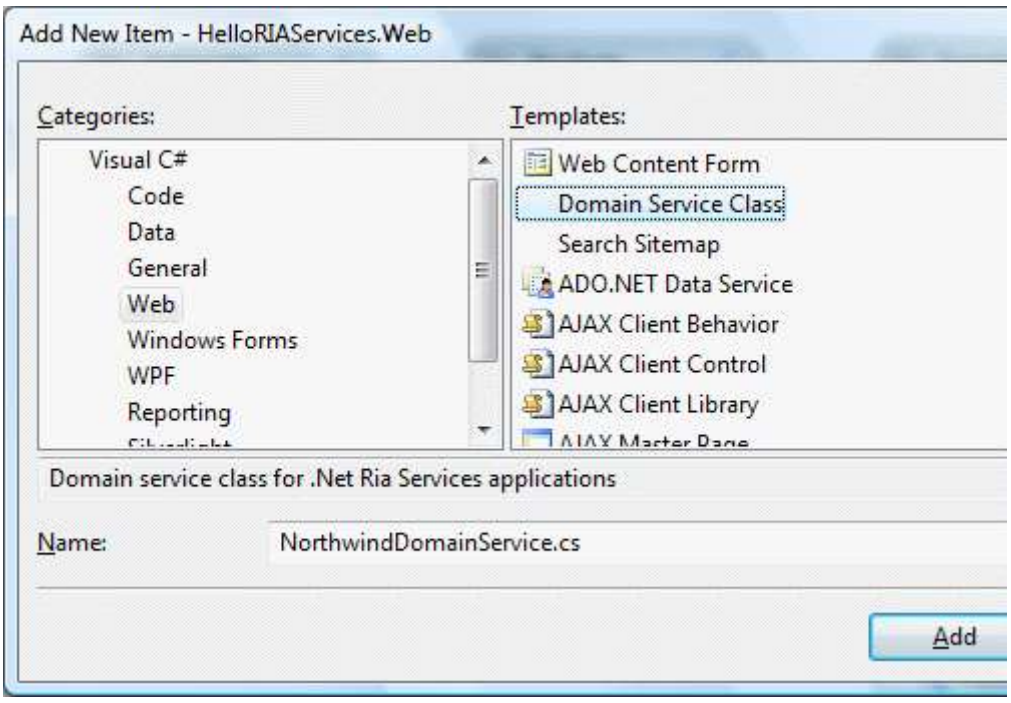


LINQ to ASP.Net Server Project seçeneğinin işaretli olması önemlidir. Böylece, **.NET RIA Servisi** için gerekli ön hazırlığın yapılması sağlanmış olur. Elbetteki bunu seçmediğimiz takdirde elimiz kolumuz bağlı değildir. Daha sonradan istenirse, Silverlight uygulamasının özelliklerinden, .Net RIA Servisi destekleyecek şekilde değişiklikler yapılabilir. Yapmış olduğumuz bu işlemlerin sonrasında, **HelloRIAServices** isimli **Silverlight** uygulaması **sunum mantığını(Presentation Logic)** içeren **istemci tarafını(client-tier)** oluştururken,**HelloRIAServices.Web** isimli web uygulaması ise, **iş mantığını(Business Logic)** içeren **orta katmanı(mid-tier)** oluşturmaktadır. Bu sebepten, veriye erişimi sağlayacak olan **LINQ to SQL(veya Ado.Net Entity Framework)** öğeleri, **Asp.Net Web** uygulaması üzerinde yer alacaktır. Aynı şekilde **DomainService** sınıfıda, Asp.Net Web uygulaması üzerinde konuşlandırılacaktır. Tahmin edileceği üzere, servis için istemci tarafından gönderilecek çağrıları ele alacak olan **içerik sınıfı ise(DataContext), Silverlight** uygulaması tarafında yer almalıdır.

Bu işlemlerin ardından, **DomainService'** in erişip istemci tarafına sunacağı veri kümesini oluşturmamız gerekmektedir. Burada, veriye erişmek amacıyla(**Data Access Layer** tarafı olarak düşünebiliriz), **Ado.Net Entity Framework** öğesini veya **LINQ to SQL** sınıflarını kullanabileceğimizi belirtmiştik. Ben örneğimizde, Ado.Net Entity Framework' ü kullanarak, **Northwind**veritabanı üzerinden aşağıdaki şemaya sahip olan tabloları kullanmayı planlıyorum. Bu noktada şunu hatırlatmakta yarar var. **Ado.Net Entity Framework** veya **LINQ to SQL** kullanımı,**.Net RIA Servisleri** açısından bakıldığında bir zorunluluk yada şart değildir. Dolayısıyla farklı veri kaynaklarını kullanabilir(örneğin **XML** tabanlı...) ve istemci tarafına bir DomainService üzerinden sunabiliriz.

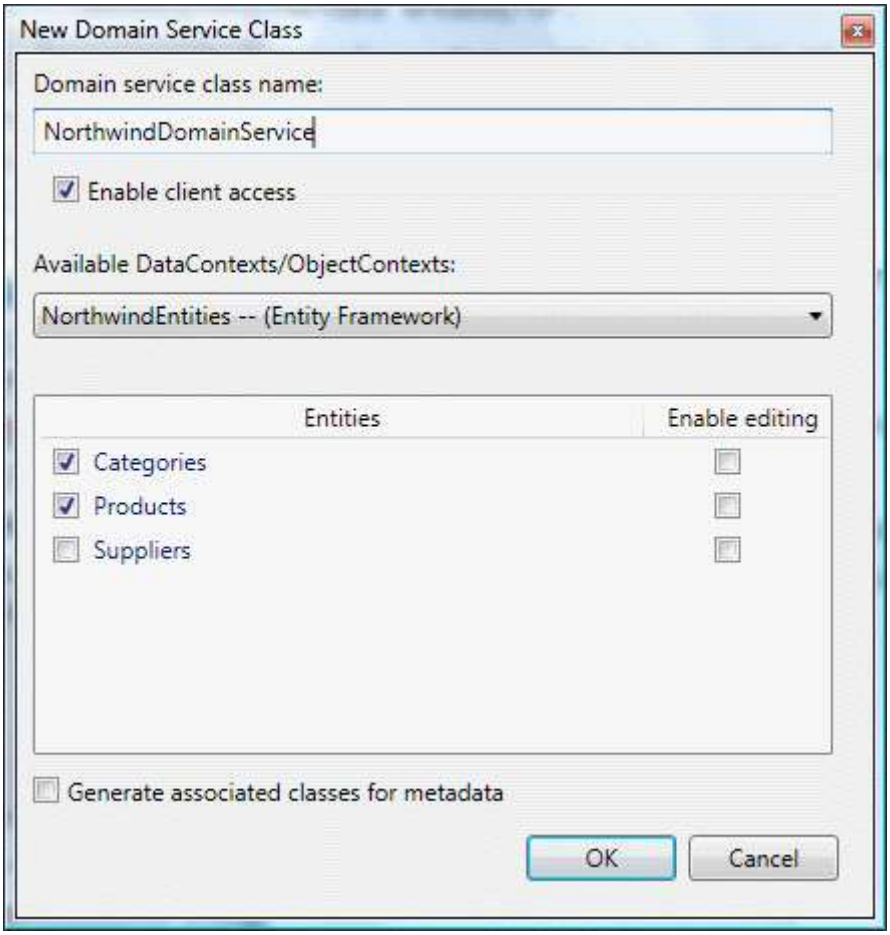


Şunu hemen belirteyim; **EDM** içeriğini **Asp.Net Web Project** üzerinde oluşturmalıyız. EDM diagramından görüldüğü üzere **Categories, Products** ve **Suppliers** tabloları için gerekli**Entity** tipleri otomatik olarak üretilmiştir. Böylece, veriye erişimi sağlayacak olan katmanı bir nevi hazırlamış bulunuyoruz. Bu işlemin ardından proje bir kere derlendikten sonra, istemciye veriyi sunacak olan **DomainService** içeriğinin hazırlanmasına başlanabilir; ki buda son derece kolaydır 😃 Tek yapmamız gereken, yine web uygulaması projesi içerisine, aşağıdaki şekildende görüldüğü gibi bir **DomainService** öğesi eklemektir.

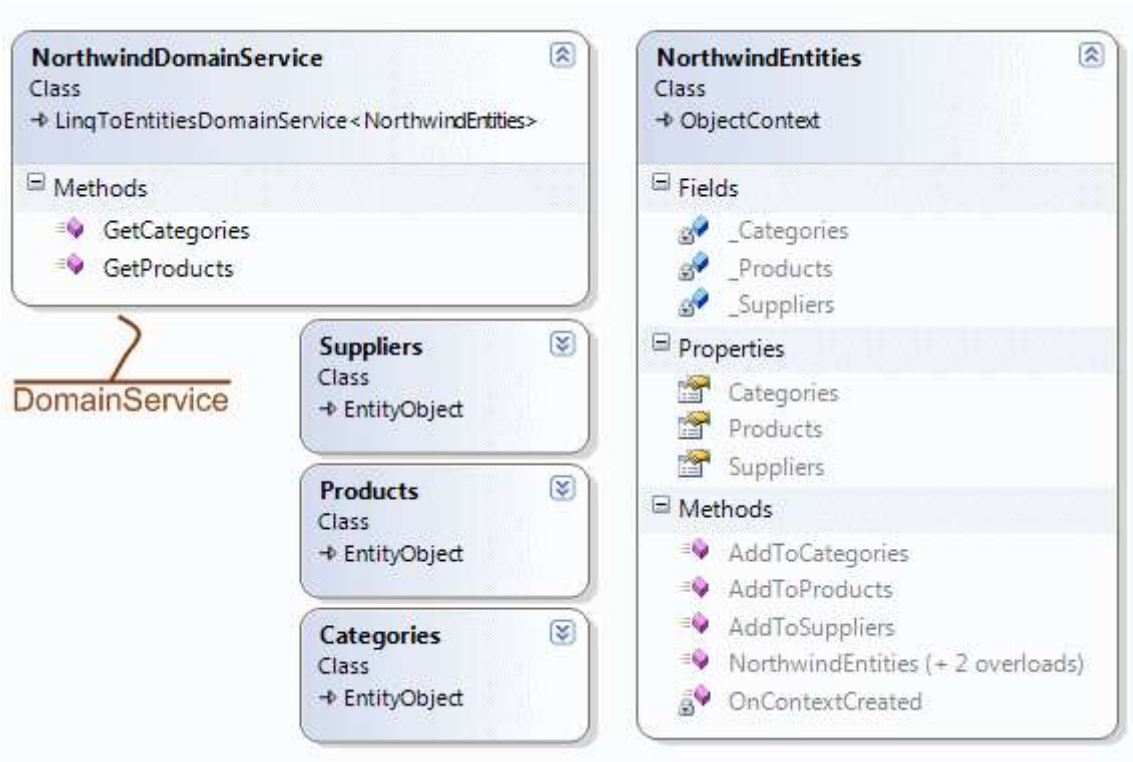


(Kullandığım sistemdeki kurulumdan kaynaklanan bir sorun olsa gerek, ikon ne yazıkki görünmüyor 😕)

Bu seçimin ardından karşımıza aşağıdaki iletişim kutusu gelecektir.



Burada **Categories** ve **Products** tipleri işaretlenmiştir. Bu tiplerin hiç birisi için **Insert, Update** veya **Delete** operasyonu hazırlanmayacaktır. Ancak bu operasyonlarında hazırlanmasını istersek,**Enable Editing** özelliklerini işaretlememiz yeterlidir. Dikkat edileceği üzere, **Available DataContexts/ObjectContexts** kısmında az önce oluşturulan **NorthwindEntities** isimli **Ado.Net Entity Framework** tipi seçilidir. Taşlar yavaş yavaş yerine oturmaktadır. Artık, **DomainService** sınıfı hazırdır ve veriyi sunmak için, **DAL** içerisinde oluşturulan Entity içeriğine bağlanmıştır. Bu noktada biraz durup, oluşturulan tipleri incelemekte yarar olacağını düşünüyorum. Web uygulaması içerisindeki **sınıf diagramını(Class Diagram)** açtığımızda aşağıdaki şekilde yer alan tiplerin oluşturulduğunu görürüz.



Products, Suppliers, Categories isimli sınıflar, **Northwind** veritabanında seçtiğimiz aynı isimli tabloların karşılıkları olan **Entity** tipleridir. **NorthwindEntities** sınıf ise, söz konusu tiplere ait koleksiyonları içerisinde özellik bazında tutmakta ve ekleme gibi temel fonksiyonellikleri içermektedir. Buraya kadarki tipler, veri erişim mantığını içeren parçalar olarak düşünülebilir.**NorthwinDomainService** sınıfı ise asıl üzerinde odaklanmamız gereken tiptir. Şimdi bu tip ile ilişkili analizlerimizi değerlendirelim.

Herşeyden önce, **LinqToEntitesDomainService<T>** isimli **generic** ve **abstract** bir sınıftan türetildiğini görüyoruz. **T** tipi olarak örneğimizde, **Ado.Net Entity Framework** tarafında ürettiğimiz, **NorthwindEntities** tipi yer almakta. Buna göre, söz konusu **DomainService** sınıfının, hangi **veri içeriğini(DataContenxt)** ve üyelerini kullanacağı belirlenmiş oluyor. Burada dikkat çekici noktalardan biriside, **DomainService** sınıfının türediği tipe ait generic kısıtlamadır.

```
namespace System.Web.DomainServices.LinqToEntities
{
   public abstract class LinqToEntitiesDomainService<T>
     : LinqToEntitiesDomainService where T : System.Data.Objects.ObjectContext
   {
      protected LinqToEntitiesDomainService();

      protected T Context { get; }
   }
}
```

Koddanda görüleceği üzere **T** tipinin **ObjectContext'** ten türeme zorunluluğu bulunmaktadır. Buda geliştiricilere bir bağımsızlık getirmektedir.
Yani, **ObjectContext** sınıfından türeteceğimiz özel tipler sayesinde farklı veri içeriklerinide **DomainService** içerisinde ele alabiliriz.

Bir diğer nokta, **DomainService** sınıfı içerisinde
sadece **GetProducts** ve **GetCategories** isimli metodların yer almasıdır. Her iki metodda **IQueryable<T>** tipinden referans döndürmektedir. Kod içeriğine baktığımızda durum biraz daha netleşmektedir.

```
namespace HelloRIAServices.Web
{
   using System.Linq;
   using System.Web.DomainServices.LinqToEntities;
   using System.Web.Ria;

   [EnableClientAccess()]
   public class NorthwindDomainService :
LinqToEntitiesDomainService<NorthwindEntities>
   {
      public IQueryable<Categories> GetCategories()
      {
         return this.Context.Categories;
      }

      public IQueryable<Products> GetProducts()
      {
         return this.Context.Products;
      }
   }
}
```

Her iki metodda basit olarak **Context** referansına gitmekte
ve **Categories** ile **Products** koleksiyonlarının içeriklerini istemci tarafına döndürmektedir. Dönüş tipleri **IQueryable** olduğundan, istemci tarafında **LINQ** ifadeleri ile sorgulanmaya devam edilmeleri pekala mümkündür. Bunlara ek olarak çok daha önemli bir nokta vardır. Metodlara istenirse parametre verilebilir ve geriye döndürülecek içerik ile ilişkili bazı kısıtlamalar yaptırılabilir. Bir başka deyişle geliştirici, metodların parametrik yapısı ile oynayabileceği gibi, dönüş içeriğini**IQueryable<T>** olmasına(**IEnumerable<T>** da olabilir) dikkat edecek şekilde değiştirebilir. Söz gelimi, belkide istemcinin bulunduğu lokasyondaki tedarikçiye göre bir Products veya Categories içeriğinin döndürülmesi sağlanabilir. Yada çok basit anlamda, içeriklerin örneğin ürün adına veya kategori adına göre sıralanarak döndürülmesi sağlanabilir. Bu nedenle kod içeriğini aşağıdaki gibi değiştirmeye karar verdim.

```
namespace HelloRIAServices.Web
{
   using System.Linq;
   using System.Web.DomainServices.LinqToEntities;
   using System.Web.Ria;

   [EnableClientAccess()]
   public class NorthwindDomainService
 : LinqToEntitiesDomainService<NorthwindEntities>
   {
      public IQueryable<Categories> GetCategories()
      {
         // Lamda operatörü ve extension method yardımıyla
         return this.Context.Categories.OrderBy(c => c.CategoryName);
      }

      public IQueryable<Products> GetProducts()
      {
         // basit bir LINQ ifadesi yardımıyla
         return (from p in this.Context.Products
                 orderby p.ProductName descending
                 select p);
      }
   }
}
```

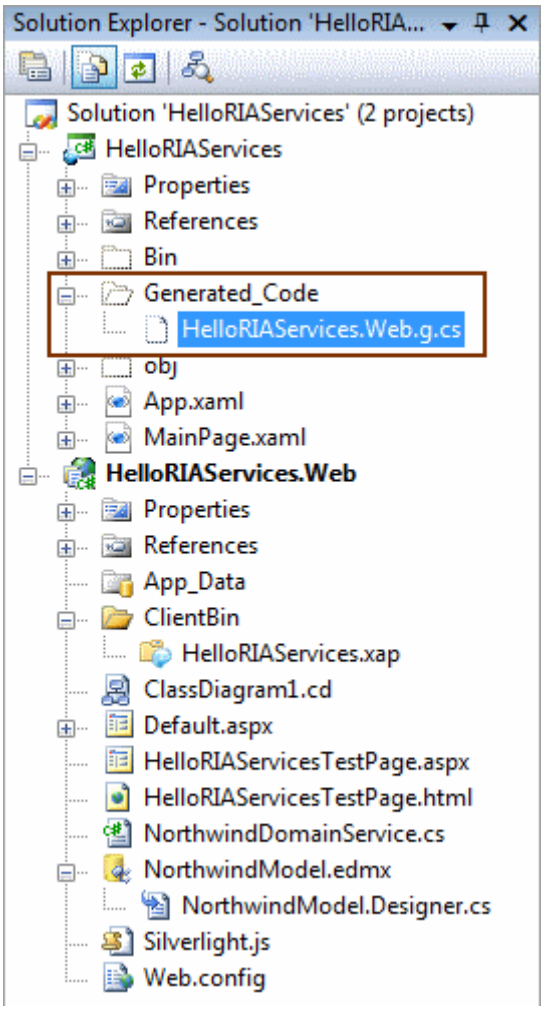
Tabiki bu değişiklikler ile sınırlı değiliz. İstersek, **DomainService** sınıfı içerisine farklı fonksiyonelliklerde ekleyebiliriz. örneğin;

```
public IQueryable<Products> GetProductsByCategory(int categoryId)
      {
         return (from p in this.Context.Products
                 where p.Categories.CategoryID == categoryId
                 orderby p.ProductName
                 select p);
      }
```

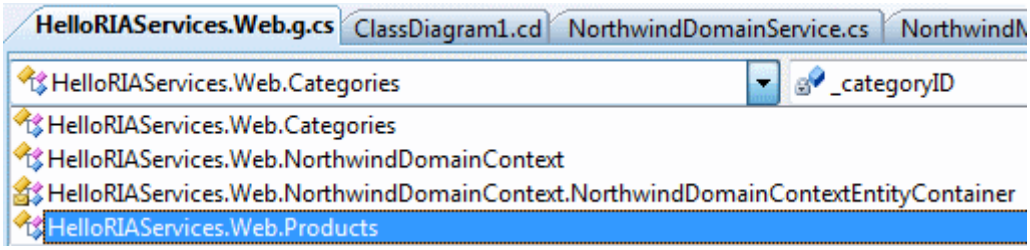
gibi.

Son olarak **DomainService** sınıfı içerisinde dikkat çeken bir noktayı daha vurgulayalım. Sınıfın kendisine **EnableClientAccess** isimli bir **nitelik(attribute)** uygulanmıştır. Bu nitelik, söz konusu sınıfın istemci katmanından görünebileceği anlamına gelmektedir.

Bu adımların ardından **Solution** tamamıyla derlenirse ve **Silverlight** uygulamasının öğelerine **Show All Files** seçeneği ile bakılırsa, aşağıdaki şekilde görülen bir dosyanın üretildiği farkedilebilir.



Buradaki kod dosyası, **DomainService** sınıfı her değiştiğinde ve bu nedenle Web projesi her derlendiğinde otomatik olarak yeniden üretilmektedir. Söz konusu kod dosyası içerisinde, servisten sunulan her bir **Entity** tipi için karşılık olan bir sınıf bulunmaktadır.



Şekildende görüleceği gibi, **Ado.Net Entity Framework** kullanarak servis üzerinden sunulan **Categories** ve **Products** tipleri için, istemci tarafında birer sınıf üretilmiştir. Ayrıca, daha önceki yazımızda da bahsettiğimiz gibi, **.NET RIA Servislerinin** önemli iki parçasından birisi olan **DomainContext** türevli bir sınıfda(**NorthwindDomainContext**)

oluşturulmuştur. Servis tarafında(**DomainService** içerisinde) yer alan **GetProducts** ve **GetCategories** metodlarına karşılık olarak, istemci tarafındaki **DomainContext** tipi içerisine **LoadProducts** ve **LoadCategories**fonksiyonları hazırlanmıştır. Yine özel olarak eklediğimiz **GetProductsByCategory** metoduna karşılık olarak, **DataContext** tarafında **LoadProductsByCategory** isimli fonksiyon üretilmiştir. Dolayısıyla, **Silverlight** uygulamasında, servis ile konuşulmasını sağlayacak olan **proxy** içeriği otomatik olarak üretilmiştir. Aslında **orta katmanda(mid-tier)** yer alan her bir**DomainService** tipi için, sunum katmanında bir **DomainContext** tipi var olacaktır. Yani, birden fazla veri kaynağına, farklı **DAL** öğeleri ile çıkan servisleri barındıran bir sunucu ile bunları ayrı ayrı kullanabilen bir istemci tasarlanması mümkündür. Artık tek yapmamız gereken, **DomainContext** tipinin ilgili fonksiyonlarını kullanarak istemci tarafını geliştirmektir. Bu amaçla**Silverlight** uygulamasının **MainPage.xaml** içeriğini aşağıdaki gibi geliştirdiğimizi düşünelim.

```
<UserControl x:Class="HelloRIAServices.MainPage"
   xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
   xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
   xmlns:data="clr-
namespace:System.Windows.Controls;assembly=System.Windows.Controls.Data"
   Width="500" Height="320">
   <StackPanel x:Name="LayoutRoot" Background="White" Orientation="Vertical">
      <ComboBox x:Name="cmbCategories" Height="50"
VerticalAlignment="Top" SelectionChanged="cmbCategories_SelectionChanged">
         <ComboBox.ItemTemplate>
            <DataTemplate>
               <StackPanel Orientation="Vertical">
                  <TextBlock x:Name="categoryId" Text="{Binding
CategoryID}" FontSize="12" FontFamily="Calibri" Foreground="Blue"/>
                  <TextBlock x:Name="categoryName" Text="{Binding
CategoryName}" FontSize="12" FontFamily="Calibri" Foreground="Black"/>
                  <TextBlock x:Name="categoryDescription" Text="{Binding
Description}" FontStyle="Italic" FontSize="9" FontFamily="Calibri"
Foreground="LimeGreen"/>
               </StackPanel>
            </DataTemplate>
         </ComboBox.ItemTemplate>
      </ComboBox>
      <data:DataGrid x:Name="grdProducts" Height="250"
Background="Lavender" BorderBrush="CadetBlue"/>
   </StackPanel>
</UserControl>
```

UserControl içerisinde bir adet **ComboBox** ve **DataGrid** bileşeni bulunmaktadır. **DataGrid** bileşeninin kullanılabilmesi için **Silverlight** uygulamasına **System.Windows.Controls.Data.dll assembly**' ının **refera**

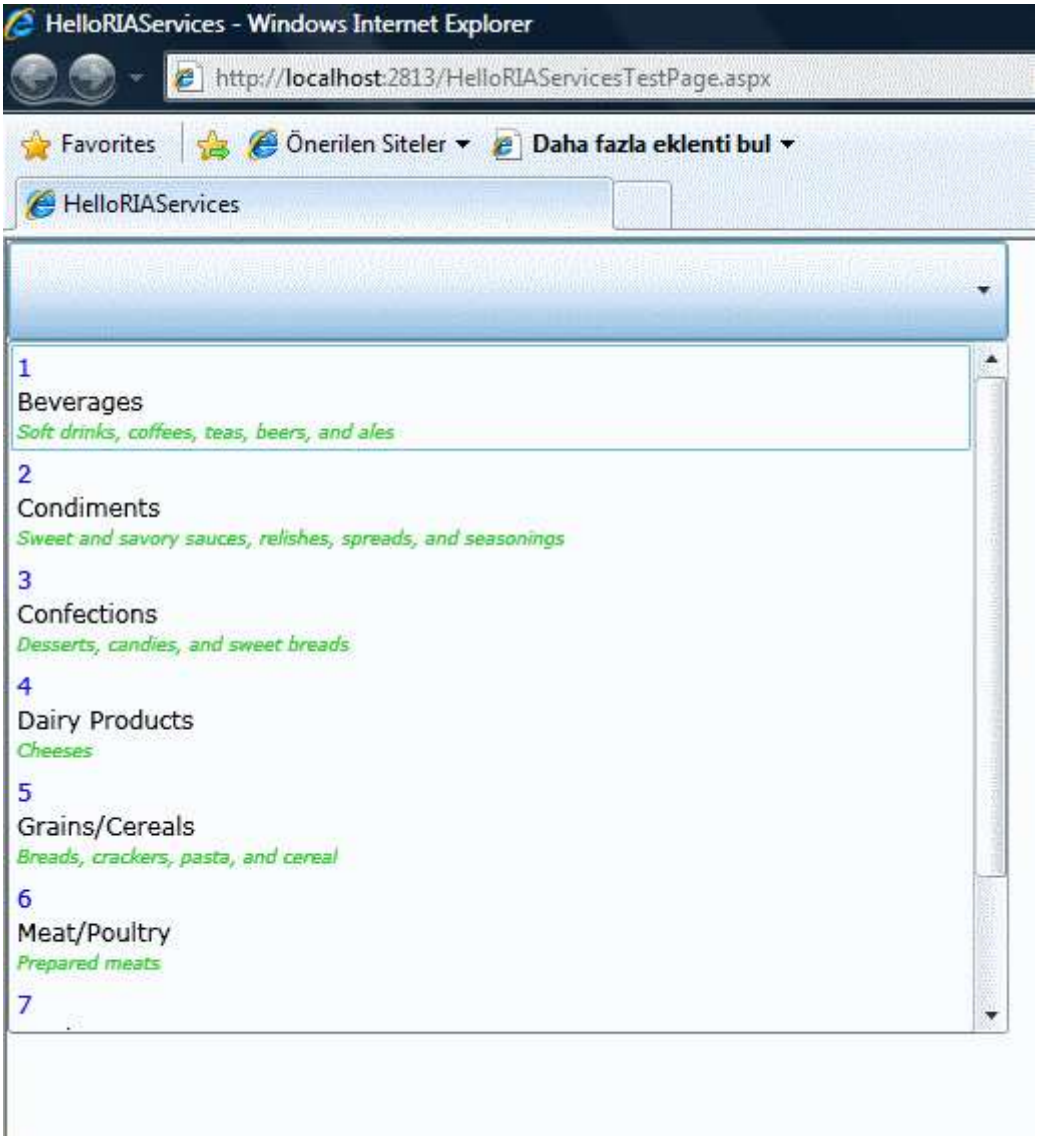
ns edilmesi gerekmektedir. Ayrıca **DataGrid** kulanımı için, **XAML** içerisinde gerekli **namespace** tanımlamasıda yapılmalıdır. Sayfanın kullanımı son derece basit olacaktır. **ComboBox** içeriği, **MainPage** yapıcı metodu içerisinde, kategoriler ile doldurulacaktır. Kullanıcı, **ComboBox** içerisinden herhangibir kategoriyi seçtiğinde ise, buna bağlı ürün listeside **DataGrid** kontrolünde gösterilecektir. **ComboBox** kontrolüne ait veri içeriğinde, bir **DataTemplate** kullanılmaktadır ve dikkat edileceği üzere **Categories** isimli **Entity** tipinin **CategoryID**,**CategoryName** ve **Description** özellik leri kullanılarak bir şablon oluşturulmuştur. **MainPage UserControl'** üne ait kod içeriği ise aşağıdaki gibidir.

```
using System.Windows.Controls;
using HelloRIAServices.Web;

namespace HelloRIAServices
{
   public partial class MainPage : UserControl
   {
      // DomainContext nesnesi
      NorthwindDomainContext context = null;

      public MainPage()
      {
         InitializeComponent();
         // DomainContext nesnesi örneklenir
         context = new NorthwindDomainContext();
         // ComboBox kontrolüne veri kaynağı olarak, EntityList tipinden olan Categories
özeliği bağlanır.
         cmbCategories.ItemsSource = context.Categories;
         // DataGrid kontrolü için veri kaynağı DomainContext nesne örneğindeki Products
özelliği ile belirlenir
         grdProducts.ItemsSource = context.Products;

         // Categories listesi LoadCategories metodu ile yüklenir.
         context.LoadCategories();
      }

      private void cmbCategories_SelectionChanged(object sender,
SelectionChangedEventArgs e)
      {
         // Seçilen öğe Categories tipinden olduğu için CategoryID özelliğine aşağıdaki kod
parçasında olduğu gibi ulaşılabilir.
         int selectedCategoryId=((Categories)e.AddedItems[0]).CategoryID;
```

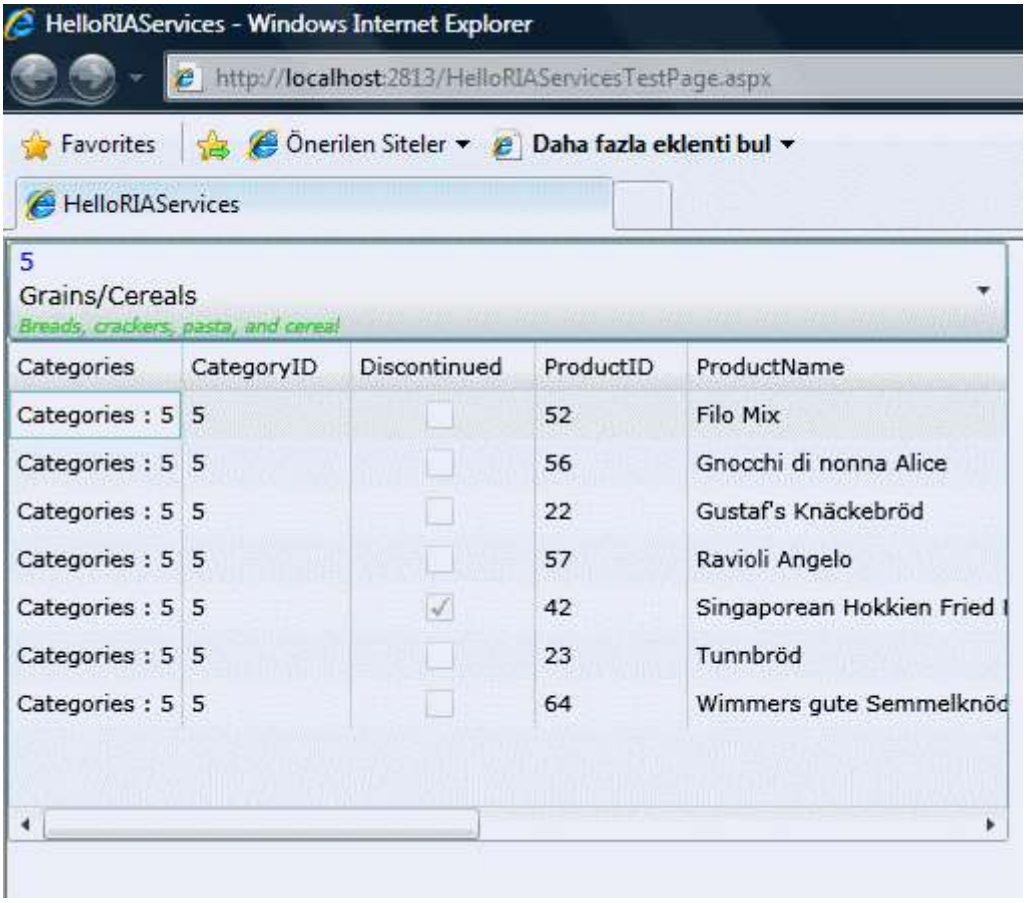
```
            // Eğer aşağıdaki temizleme işlemini uygulamassak, Grid kontrolü içerisinde veriler arka arkaya eklenerek çoğalır.
            context.Entities.GetEntityList<Products>().Clear();

            // LoadProductsByCategory metoduna, seçili kategorinin CategoryID değeri gönderilerek, bağlı olan ürün listesinin yüklenmesi sağlanır.
            context.LoadProductsByCategory(selectedCategoryId);
        }
    }
}
```

Aslında kod içeriği son derece basittir. **.Net RIA Servisleri** açısından olaya baktığımızda iki önemli nokta göze çarpmaktadır. İlk olarak veri bağlı kontrolleri, **Entity** içeriklerine bağlamak için **DomainContext** nesne örneğine ait özelliklerden yararlanılmaktadır**(Categories, Products** gibi). Diğer taraftan veriyi doldurmak için, bu isteğin sunucu tarafındaki **DomainService**tipine ulaştırılması gerektiği de ortadadır. Bu sebepten **LoadCategories** ve **LoadProductByCategory** metodlarından yararlanılmaktadır. Sonuç olarak uygulama çalışma zamanında test edildiğinde aşağıdaki örnek çıktılar ile karşılaşılacaktır.

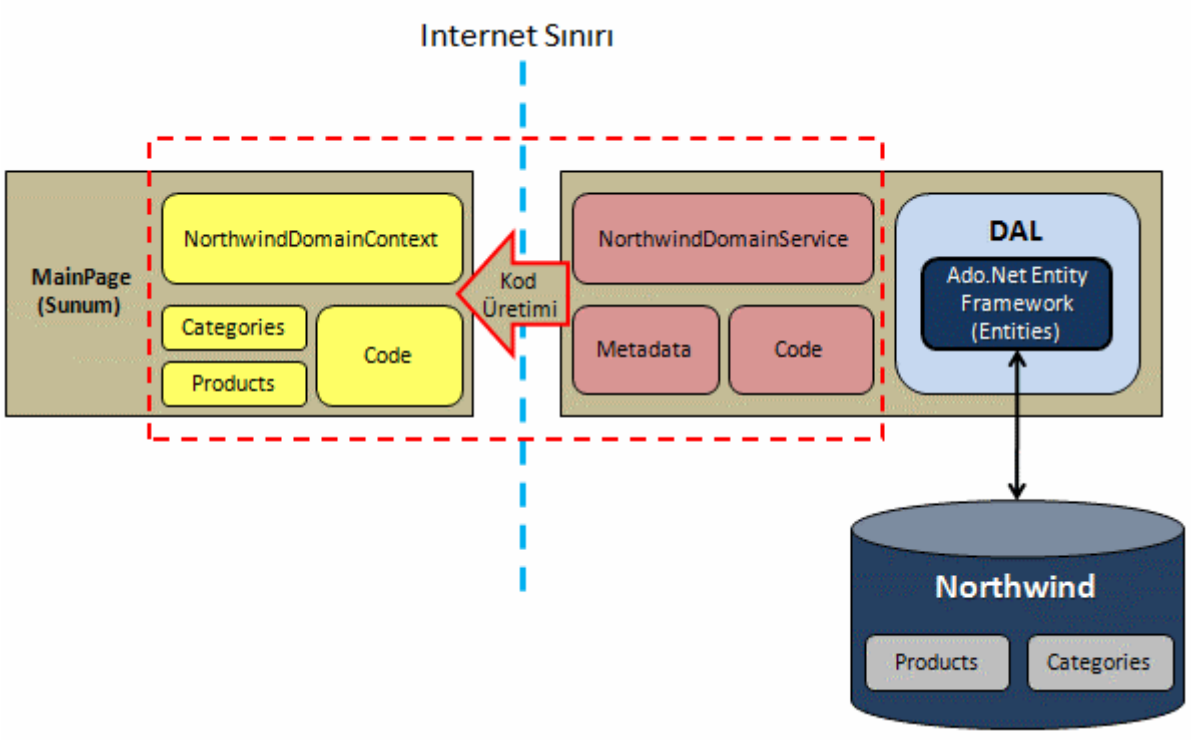
Uygulama ilk çalıştırıldığında kategoriler, ComboBox bileşeni içerisine yüklenecektir.



Herhangibir kategori seçildiğinde ise...



DataGrid kontrolü, bu kategoriye bağlı ürünler ile doldurulacaktır. İşte bu kadar. Görüldüğü gibi, **Silverlight** uygulamalarında **.Net RIA Servislerini** kullanılarak, **çok katmanlı modelin(n-tier),** basitçe **iki katmana(2-tier)** indergenmesi sağlanabilmektedir. Geliştirdiğimiz örnek göz önüne alındığında, aşağıdaki şekil durumu biraz daha açıklığa kavuşturmaktadır.



Böylece geldik bir yazımızın daha sonuna. **.Net RIA Servisleri** ile ilişkili araştırmalarıma devam ettikçe, öğrendiklerimi sizlerle paylaşmaya devam ediyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

HelloRIAServices.rar (1,60 mb)

.Net RIA Servisleri Nedir? (2009-05-08T22:41:00)

.net ria services,silverlight,

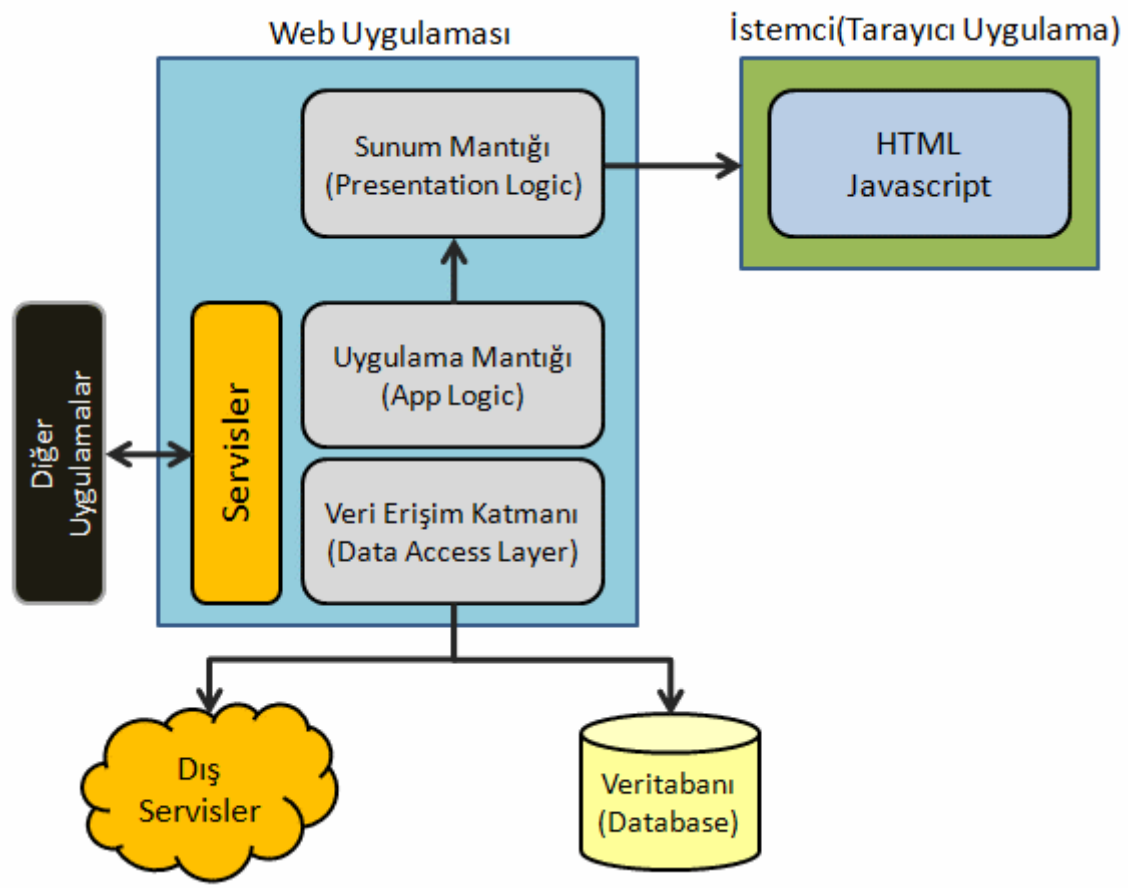
Merhaba Arkadaşlar

Son yıllarda bildiğiniz üzere **Servis Tabanlı Uygulamalar(Service Oriented Applications)** hayatımızda oldukça fazla yer kaplamaya başladı. **Microsoft** cephesinden olaya baktığımızda, en büyük sıçramanın **Windows Communication Foundation** ile **.Net Framework 3.0'** da yaşandığını söyleyebiliriz. **WCF**' in getirdiği servis bazlı uygulama geliştirme yaklaşımı, **.Net Framework 3.5** ile dahada zenginleşti. Eklenen **Web programlama modeli(Web Programming Model)** özellikleri sayesinde, **REST(Representational State Transfer)** bazlı servislerin geliştirilebilmesinin yolu açıldı. Sonrasında **Workflow Foundation** ile iç içe geçen **WCF** özellikleri sayesinde, iş akışlarının farklı domainler ile haberleşebilmesi veya servis gibi sunulabilmesi olanaklı hale geldi. Derken **.Net Framework 3.5 Service Pack 1** ile hayatımıza başka bir kavram daha girdi. **Ado.Net Data Services**. Bu model ile, **Ado.Net Entity Framework** veya**LINQ(Language INtegrated Query)** bazlı sağlayıcılar üzerinden verinin REST tabanlı olarak sunulabilmesi mümkün hale geldi.Tabi bu geçişler

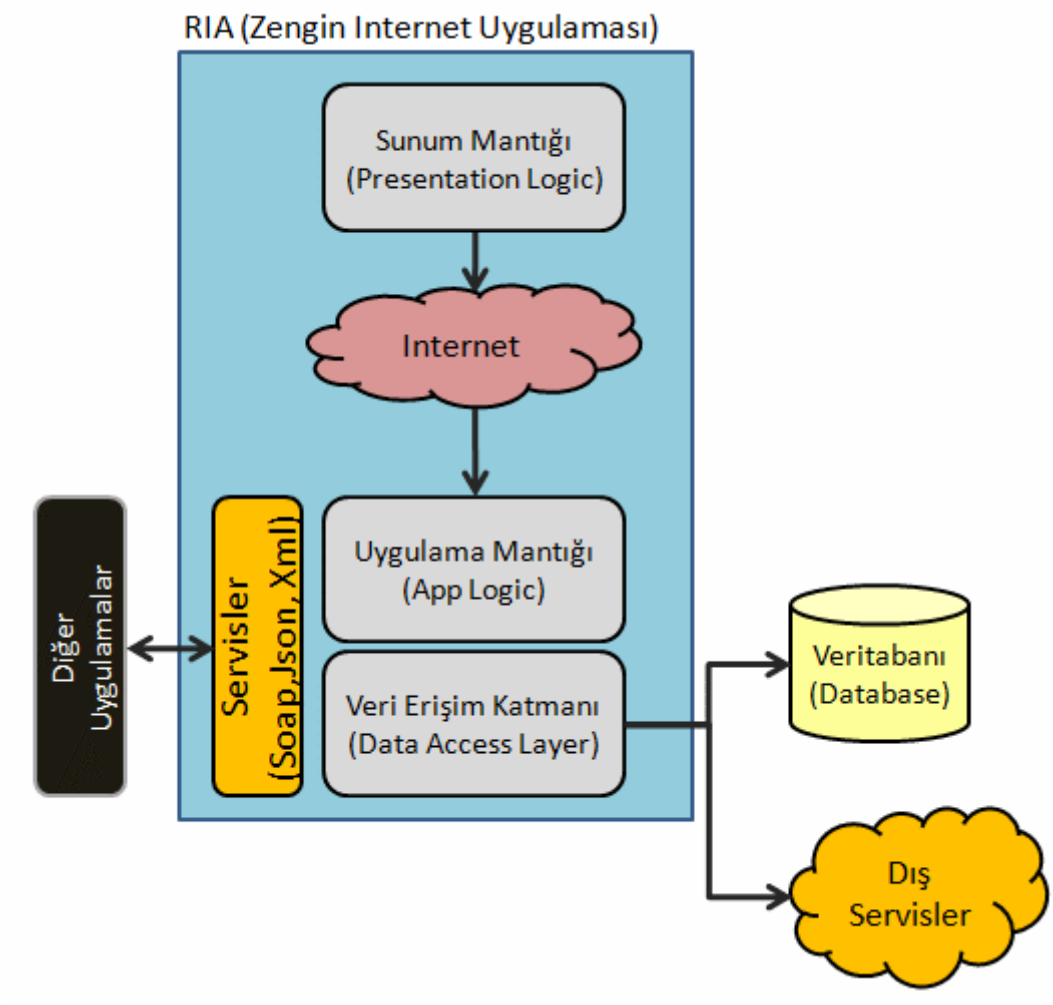
sırasında **Client Application Services** ve**Azure** gibi kavramlar ile geliştiricinin hayatını kolaylaştıran **REST Starter Kit** gibi pek çok yeni fikir ve vizyon ile karşılaştık.
Ama **Microsoft** cephesindeki yenilikler tüm hızıyla sürmeye devam etti, ediyor, edecek... Bir süredir **.Net Framework 4.0** ve bu etapta **WF 4.0&WCF 4.0** yeniliklerini incelemekteyim. Ancak arada kaçırdığım önemli bir konu var. **.Net RIA(Rich Internet Application) Services** ve **Silverlight** Dolayısıyla bu yazımda sizlere, .Net RIA Servisleri ile ilişkili öğrendiklerimi ve bilgilerimi aktarmaya çalışıyor olacağım.

En nihayetinde, **Silverlight** sayesinde istemci tarafında çok zengin içeriklere sahip olabilecek ve tarayıcı tabanlı(ve hatta Silverlight 3.0 sonrası masaüstü...) uygulamaların geliştirilmesi mümkün. Ancak Silverlight gibi bir uygulama geliştirme modelinde, istemcinin sunucu üzerinde yer alan bazı veri kaynaklarına erişmesi için, servislerin kullanılmasıda kaçınılmaz bir gerçek.*(Nedenini biraz sonra daha iyi anlatabileceğim.)*

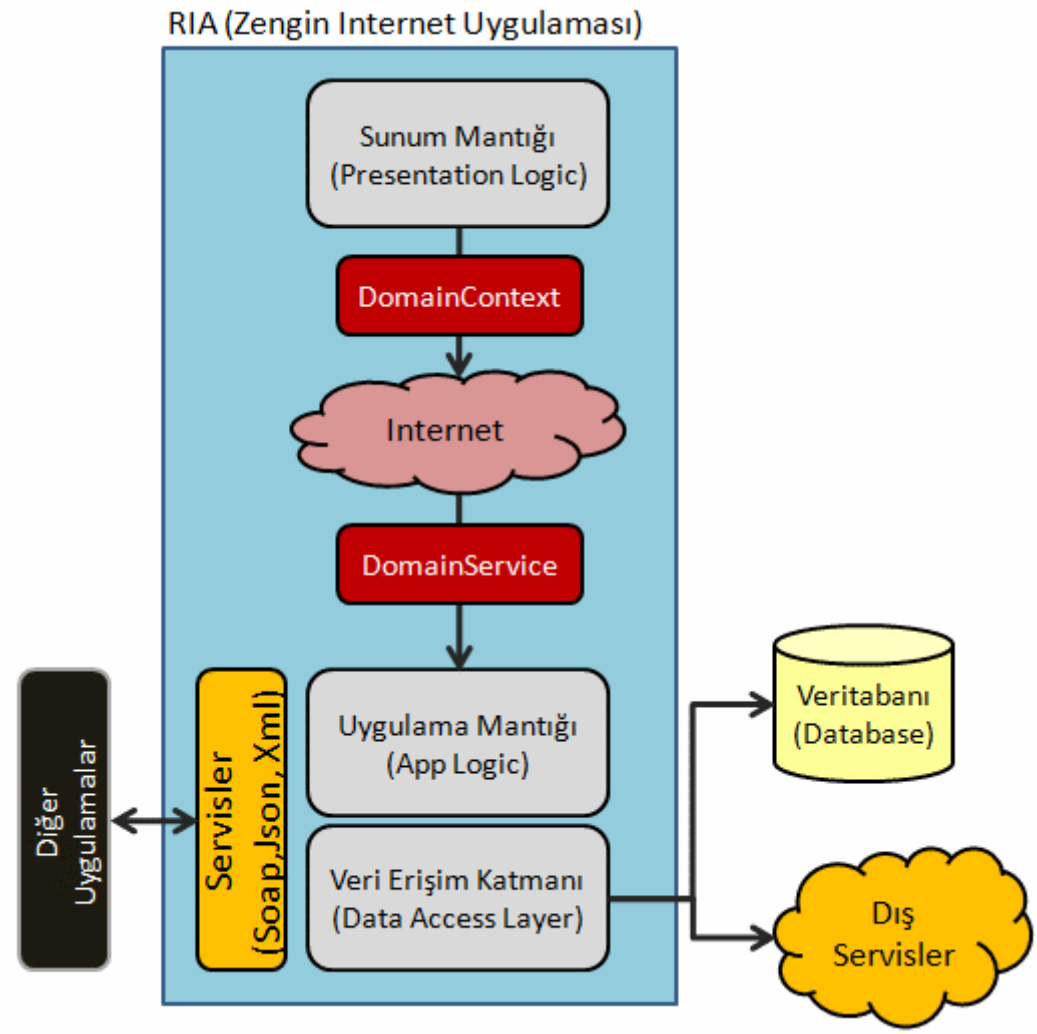
özellikle **Silverlight 3.0** ve **.Net RIA Service** çıkana kadar, geliştiricilerin sunucu verilerine erişmesi için biraz daha fazla kodlama yapması gerekmektedir. Aslında olaya sadece Silverlight değil,**Asp.Net Ajax** gibi istemciler açısından bakıldığında da, benzer kodlama süreçleri söz konusudur. Bu tip **RIA** uygulamalarını, **n-tier** tarzı mimariler ile geliştirmek istediğimizden, aslında **sunum(Presentation)** katmanının standart **Asp.Net** modelinden farklı olarak, tamamen istemci tarafına yıkıldığı oldukça önemli bir noktadır. Sanıyorum burada biraz kafaları karıştırdım. Gelin olayı standart **n-tier** modelin Asp.Net uygulamalarındaki genel kullanımı ile analiz etmeye başlayalım. Aşağıdaki şekilde bu model vurgulanmaya çalışılmaktadır.



Klasik olarak bir **Asp.Net Web** uygulamasında*(çoğunlukla Asp.Net Ajax içinde benzer durum söz konusudur),* katmanların tamamı sunucu üzerinde yer alır. **Uygulama mantığı(Application Logic-Business Layer), veriye erişim katmanı(Data Access Layer)** ve istemcinin göreceği HTML çıktının üretileceği **sunum katmanı(Presentation Layer)**. Bunlara ek olarak web uygulaması içerisinde, veri erişim katmanından dış servisler yardımıyla farklı kaynaklara gidilebilir veya uygulamanın kendisinin farklı alanlardaki programlara sunacağı bir takım hizmetler/servisler olabilir. Oldukça basit ve kullanışlı. Ancak, günümüz uygulamalarında ve özellikle son yıllarda **kullanıcı deneyimini(User Experience)** zenginleştirecek şekilde yapılan bir çok atılım vardır. *(Bu etkileşim özellikle web tabanlı mimarilerde kendini daha da ön plana çıkarmaktayken, geliştirme süreçlerinin standart masaüstü uygulamalara nazaran daha karmaşık ve zor olduğuda söylenebilir.)* Bu nedenle tarayıcı uygulamalar üzerindeki kullanıcı deneyimini zenginleştirecek **Silverlight** gibi geliştirme ortamları söz konusudur. Hal böyle olunca yukarıdaki şekilde çizdiğimiz katmanlı model biraz daha değişim göstermektedir. Aşağıdaki şekilde olduğu gibi.



Zengin internet uygulamalarında, sunum katmanı/mantığı istemci tarafına yıkılmaktadır*(Hatırlayalım, Silverlight uygulamalarının çalıştırılması için istemci tarafında minik bir framework, add-in tarzında yüklenmiş olmalıdır).* Bu da kullanıcı etkileşimini dahada üst seviyeye çıkartmak anlamına gelmektedir. Ama doğal olarak **n-tier** modelde sunum katmanı ile uygulama mantığı arasına internet ağının girmesi gerekmektedir. Buna göre **RIA**' ları basit bir istemci uygulamadan ziyade, sunucu bileşenlerinide içeren birer **Internet uygulaması** olarak düşünmek gerekmektedir. Hal böyle olunca, sunucu tarafındaki veri kaynaklarının sunum tarafında kullanılabilmesinde servisler önemli bir rol üstlenmektedir. **.Net RIA Servislerine**kadarki zaman diliminde, geliştiricilerin bu anlamda düşünmesi gereken pek çok kıstas vardır. Herşeyden önce veriyi istemci tarafına taşıyacak servisin ve metodlarının yazılması gerekir. Ayrıca istemci tarafında, bu servisin kullanılabilmesi için gerekli **proxy** üretiminin yapılması şarttır. **Silverlight** tarafında kolay olan proxy üretimi, **Asp.Net Ajax** tarafı düşünüldüğünde ek **javascript** kütüphaneleri anlamına gelmektedir. Yinede, sunucu tarafında **Ado.Net Entity Framework** veya **LINQ to SQL** gibi modelleri kullanabileceğimizden bu zahmete girmeye değmektedir. Microsoft'un söz konusu servislerin, **n-tier** içerisindeki uyarlanışını daha da kolaylaştırmak adına **.Net RIA Servislerini** geliştirdiğini söyleyebiliriz. .Net RIA Servisleri kavramsal olarak iki ana parçadan oluşur.



DataService sınıfı aslında, temel **CRUD(CreateRetrieveUpdateDelete)** işlemlerini ve özel bir takım operasyonları içerebilir. Bunlara ek olarak **doğrulama(Validation), yetkilendirme(Authorization)** gibi kısıtlarıda ele alabilir. Bu nedenle **DataService** sınıfının, veri için ele alınacak iş mantığını içerdiğini söyleyebiliriz. DataService sınıfı genel olarak arka planda, **hazır olan(built-in)** veri modellerini kullanır. Yani **Ado.Net Entity Framework** veya **LINQ to SQL** burada göz önüne alınabilir. Elbetteki diğer veri kaynaklarıda gerek servisler, gerek özel kodlamalar yardımıyla kullanılabilir.İkinci bölümde yer alan **DataContext** sınıfı ise, servislerin istemciye sunduğu verilerin, tip bazındaki karşılıklarını içermektedir. Bu nedenle istemci tarafında, verilerin yüklenmesi, üzerinde yapılan değişikliklerin tekrardan sunucu tarafına gönderilmesi için gerekli kodlamaları ve metodlarıda hazır olarak içermektedir. Tahmin edeceğiniz üzere, **.Net RIA Servislerinin Visual Studio 2008** ortamında geliştirilmesi son derece kolay ve basittir. 😀

Son olarak .Net RIA Servisleri ile ilişkili olaraktan merak edilen sorulara cevap bulabileceğiniz ve gerekli yüklemeleri edinebileceğini bir internet adresini paylaşmak isterim.

Böylece geldik bir yazımızın daha sonuna. Bu yazımda sizlere .Net RIA Servislerini, anladığım kadarıyla anlatmaya çalıştım. Bir sonraki yazımızda basit bir örnek geliştirerek Merhaba **.Net RIA Servisi** diyeceğiz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

.Net Tv - Design Patterns : Proxy (2009-05-08T06:25:00)

c#,design patterns,

Merhaba Arkadaşlar.

Kısa bir aradan sonra **.Net Tv** görsel derslerimize devam ediyorum. Bu görsel dersimizde , son derece basit ve oldukça kullanışlı tasarım desenlerinden birisi olan **Proxy** kalıbını incelemeye çalışıyoruz.

Tasarım kalıpları(Design Patterns) ile ilişkili diğer görsel derslerimide, aşağıdaki adreslerden indirebilir ve izleyebilirsiniz.

Adapter Tasarım Kalıbı

Abstract Factory Tasarım Kalıbı

Factory Method Tsaarım Kalıbı

Façade Tasarım Kalıbı

Singleton Tasarım Kalıbı

WF - XAML Bazlı Workflow Örnekleri Geliştirmek (2009-05-06T20:23:00)

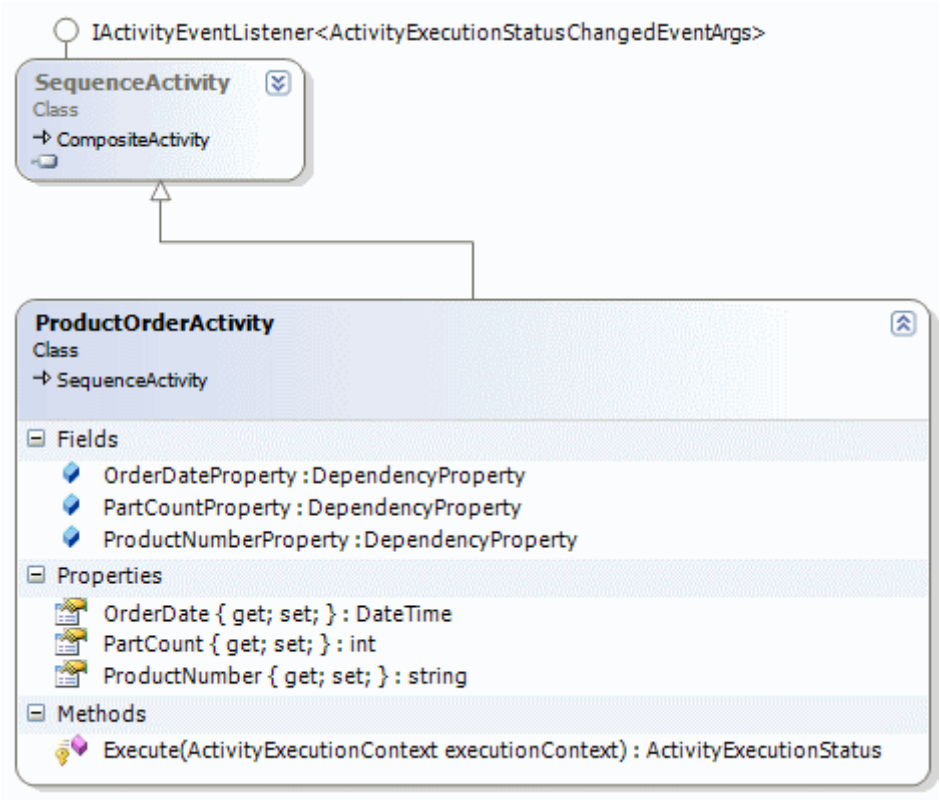
wf,xaml,

Merhaba Arkadaşlar,

Geçtiğimiz günlerde **Workflow 4.0** ile ilişkili araştırmalarıma devam ederken, özellikle dekleratif olarak tanımlanabilen WF servislerindeki önemli bir noktayı farkettim. Bu, aynı zamanda WF 4.0 ile birlikte gelen en önemli yenilikler arasındaydı. *(Hatta WF motorunun-*

*Engine- değişmesi veya **temel aktivite kütüphanesinde(Base Activity Library),** ata tip olarak **WorkflowElement** isimli yeni bir sınıfın getirilmesi kadar önemliydi)* Bir workflow örneğinin sadece **XAML** içeriğinden oluşacak şekilde koda ihtiyaç duymadan tasarlanabilmesi(design), derlenebilmesi(Compile) ve gerektiğinde çalışma zamanında basit bir notepad uygulaması ile değiştirilerek güncellenebilmesi...Burada özellikle derleme konusu son derece dikkat çekici. Nedeni mi?

Nedeni araştırmak için elbette basit bir senaryo üzerinden ilerlemem gerekiyordu. Bu yüzden dün gece biraz geç bir vakittede olsa üşenmeden kodlamaya başladım. Senaryoya göre, **.Net 3.5**açısından olaya bakıp, sadece XAML içeriğinden oluşacak bir Workflow örneğini oluşturmak ve çalıştırmak istiyordum. Bu sebeple öncelikle, örnek bir aktivite tipi geliştirmeye karar verdim. **ProductOrderActivity** isimli aktivite bileşenini, ayrı bir **Workflow Activity Library** projesi içerisinde aşağıdaki sınıf diagramında olduğu gibi tasarladım.



Kod içeriği

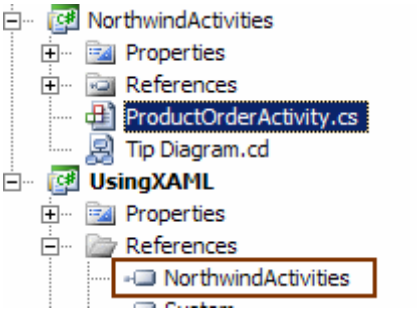
```
using System;
using System.ComponentModel;
using System.Workflow.Activities;
using System.Workflow.ComponentModel;

namespace NorthwindActivities
{
 public class ProductOrderActivity
```

```
    : SequenceActivity
{
    public static DependencyProperty ProductNumberProperty =
DependencyProperty.Register("ProductNumber", typeof(string),
typeof(ProductOrderActivity));
    public static DependencyProperty PartCountProperty =
DependencyProperty.Register("PartCount", typeof(int), typeof(ProductOrderActivity));
    public static DependencyProperty OrderDateProperty =
DependencyProperty.Register("OrderDate", typeof(DateTime),
typeof(ProductOrderActivity));

    [Description("ürün Numarası")]
    [Category("Sipariş Parametreleri")]
    [Browsable(true)]
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
    public string ProductNumber
    {
      get
      {
        return ((string)(base.GetValue(ProductOrderActivity.ProductNumberProperty)));
      }
      set
      {
        base.SetValue(ProductOrderActivity.ProductNumberProperty, value);
      }
    }

    [Description("Parça sayısı")]
    [Category("Sipariş Parametreleri")]
    [Browsable(true)]
    [DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
    public int PartCount
    {
      get
      {
        return ((int)(base.GetValue(ProductOrderActivity.PartCountProperty)));
      }
      set
      {
        base.SetValue(ProductOrderActivity.PartCountProperty, value);
      }
    }

    [Description("Sipariş Tarihi")]
    [Category("Sipariş Parametreleri")]
```

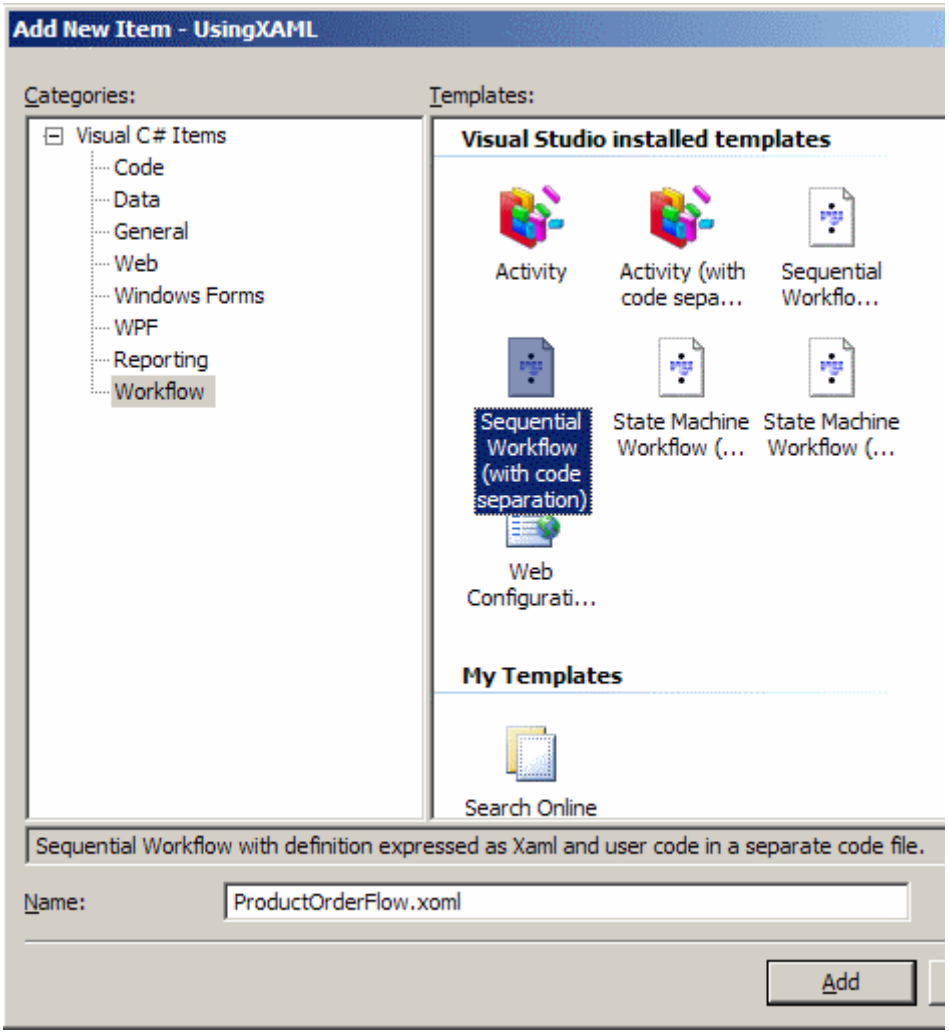
```
        [Browsable(true)]
        [DesignerSerializationVisibility(DesignerSerializationVisibility.Visible)]
        public DateTime OrderDate
        {
            get
            {
                return
Convert.ToDateTime(base.GetValue(ProductOrderActivity.OrderDateProperty));
            }
            set
            {
                base.SetValue(ProductOrderActivity.OrderDateProperty, value);
            }
        }

        protected override ActivityExecutionStatus Execute(ActivityExecutionContext
executionContext)
        {
            Console.WriteLine("{0} numaralı üründen {1} adet sipariş
işlemi...",ProductNumber,PartCount.ToString());
            Console.WriteLine("{0}  tarihine kadar sipariş
edilmelidir.",OrderDate.ToShortDateString());
            return base.Execute(executionContext);
        }
    }
}
```

Aktivite aslında basit olarak bir ürünün belirli bir tarihe kadar, istenen miktarda sipariş edilmesi adımını yürüten bir modele sahipti. Tabiki sembolik olarak. Bu sebepten **ProductNumber**,**PartCount** ve **OrderDate** isimli özellikleri bulunmaktaydı. Bu özellikler,, aktivitenin başka aktivitiler içermesi veya başka aktivitelere **bağlanması(Binding)** gibi ihtiyaçlara sahip olabileceğinden**DependencyProperty** tipi ile ilişkilendirilmiştim. özellikler, tasarım zamanında Visual Studio IDE' si tarafından değerlendirileceğinden, **Description, Category, Browsable** ve**DesignerSerializationVisibility** gibi niteliklerlede sahipti. Aktivite icra edildiğinde ise **ezilen(override) Exeute** metodu içeriği çalıştırılmaktadır. Bu kısımda işin modeline göre bir takım işlemler yapılması gerekmekte. Ben sembolik olarak sadece ekrana bazı bilgiler yazdırmayı hedefledim.

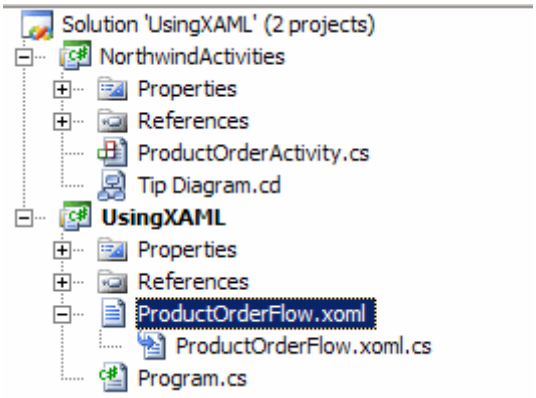
Şimdi gelelim bu aktiviteyi kullanacağımız örnek **Workflow** uygulamasına. Bu amaçla testleri kolayca yapabileceğim bir **Sequential Workflow Console Application** projesi oluşturdum. Projenin,**ProductOrderActivity** aktivitesini kullanabilmesi içinde, tanımlandığı **NorthwindActivities** kütüphanesini referans ettim.



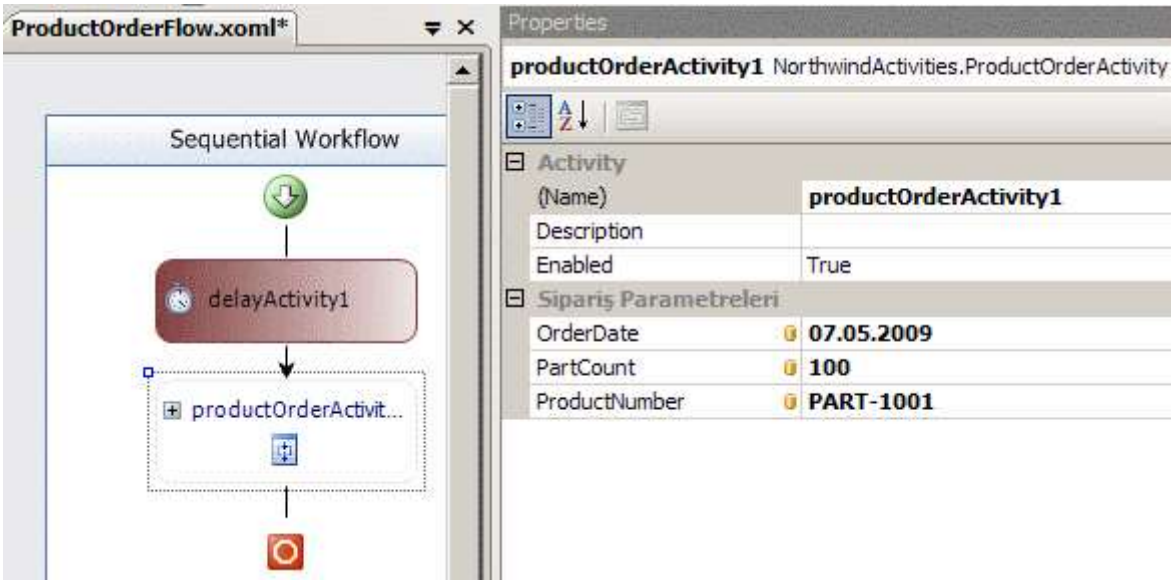
Artık ön hazırlıklar tamamlanmıştı. Sırada XAML bazlı **Sequential Activity** öğesinin eklenmesi vardı. Yanlız burada yapacağımız seçimin önemli olduğunu vurgulamak isterim. Nitekim amacımız kod içermeyen ve XAML içeriğine sahip bir Workflow örneği geliştirmek olduğundan, proje öğelerinden **Sequential Workflow (with no code)** tipini seçmemiz gerekiyor. Tabi eğer **State Machine Workflow** tipinden bir proje söz konusuysa, **State Machine Workflow(with no code)** öğesinin seçmemiz gerekiyor.



Bunun sonucunda projeye aşağıdaki şekilde görülen ProductOrderFlow isimli **XOML** uzantılı bir öğenin eklendiği görülür.



Ancak görüldüğü gibi bu oluşum sırasında **xoml** uzantılı içerik dışında **cs** uzantılı bir kod içeriğide üretilmektedir. Şunu hemen hatırlatayım. Amacımız kesin olarak kod dosyasından bağımsız bir Workflow örneği oluşturmaktır. Peki bunun için ne yapmalıyız? Aslında şu an için çözüm son derece basit. cs uzantılı dosya silinir Bende aynen böyle yaptım. Tabi şu anda Workflow içerisinde herhangibir aktivite kullanılmamakta. Ancak dikkat edilmesi gereken önemli bir nokta daha var. Bu Workflow için bir kod bloğu olmadığından, içeride kullanacağımız aktivitelerin Codebehind dosyası içerisine kod atmayacak şekilde kullanılmaları gerekmekte. Söz gelimi bir **CodeActivity** bileşenini kullanmak istediğimizde, bu bileşenin çalıştırılması sonucu devreye girecek metodun, cs kod dosyası içerisinde yer alması gerekmektedir. Oysaki şu anki teorimize göre böyle bir dosya bulunmamaktadır(olmamalıdır). Buda bizi, özel aktivite tiplerinin yazılmasına itmektedir. Ama elbetteki kod dosyasına ihtiyaç duymayan bazı aktivite bileşenleri burada ele alınabilir. örneğin **DelayActivity** aktivitesi. Bu bilgilerden yola çıkarak **ProductOrderFlow.xoml** içeriğini tasarım zamanında aşağıdaki gibi oluşturdum.



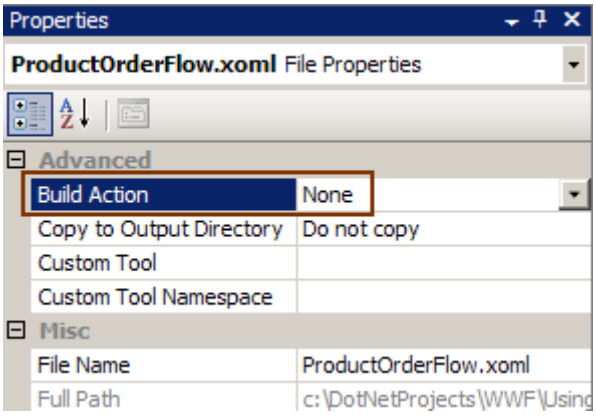
Şekildende görüldüğü gibi, **Workflow** içerisinde önce **delayActivity** bileşeni ve peşinden yazdığım **productOrderActivity** bileşeni çalıştırılmakta. **productOrderActivity1**bileşeninin OrderDate, PartCount ve ProductNumber isimli özelliklerine ise sembolik değerler aktarılmış

durumdadır. Şimdi **xoml** içeriğini **XML Editor** yardımıyla açarsak, aşağıdaki içeriğin oluşturulduğunu görürüz.

<SequentialWorkflowActivity x:Class="NorthwindActivities.ProductOrderFlow" x:Name="ProductOrderFlow" **xmlns:ns0="clr-namespace:NorthwindActivities;Assembly=NorthwindActivities, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"** xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" xmlns="http://schemas.microsoft.com/winfx/2006/xaml/workflow">
<DelayActivity TimeoutDuration="00:00:05" x:Name="delayActivity1" />
<ns0:ProductOrderActivity x:Name="productOrderActivity1" ProductNumber="PART-1001" PartCount="100" OrderDate="2009-05-07T00:00:00.0000000" />
</SequentialWorkflowActivity>

Görüldüğü gibi **Workflow** içerisinde aktivitilerin tamamı, **XAML** içeriği olarak oluşturulmuştur. Bu, zihinlerde yeni ufuklar açacak kadar önemli bir ayrıntır. çünkü, istenirse bu içerikte yer alan elementlerin yerleri değiştirilerek akışın şekline müdahele edilebilir(Koda girmeye gerek kalmadan). Yada başka elementler basit bir notepad programı yardımıyla içeriğe dahil edilip akışa yeni adımların eklenmesi sağlanabilir. Hatta bu içerik belki bir depolama ortamında saklanarak farklı görsel uygulamaların bu akışları ele alabilmesi, değiştirebilmesi sağlanabilir*(**Oslo, Quadrant** kavramına gitmeye çalıştığımı sanıyorumki anlamışsınızdır)*

Sonrasında aşırı heyecan yapmaya gerek olmadığını farkedip devam etmeye karar verdim. Bu nedenle projeyi derleyerek yoluma devam etmek istedim. Ancak oldukça ilginç bir durumla karşılaştım. Proje içerisinde birden **ProductOrderFlow.xoml.cs** isimli kod dosyası ortaya çıktı. Gecenin karanlığında sanki bir korku filminde yaşanan gerilimi hissetmiştim. Ensemden soğuk bir ter damlası ilerlerken, bu hortlağın nereden çıktığını düşünüyordum. Aslında bu son derece doğaldı. Nitekim proje derlendiğinde, **xoml** dosyası da hesaba katıldığından, cs dosyası otomatik olarak üretilmekteydi. Bu tabiki istediğim bir durum değildi. Bu nedenle **ProductOrderFlow.xoml** dosyasının **Build Action** özelliğinin değerini **None** olarak belirlemek yeterliydi. Tabiki sonrasında(öncesinde) cs dosyasını silmeyi unutmamak da gerekiyordu.

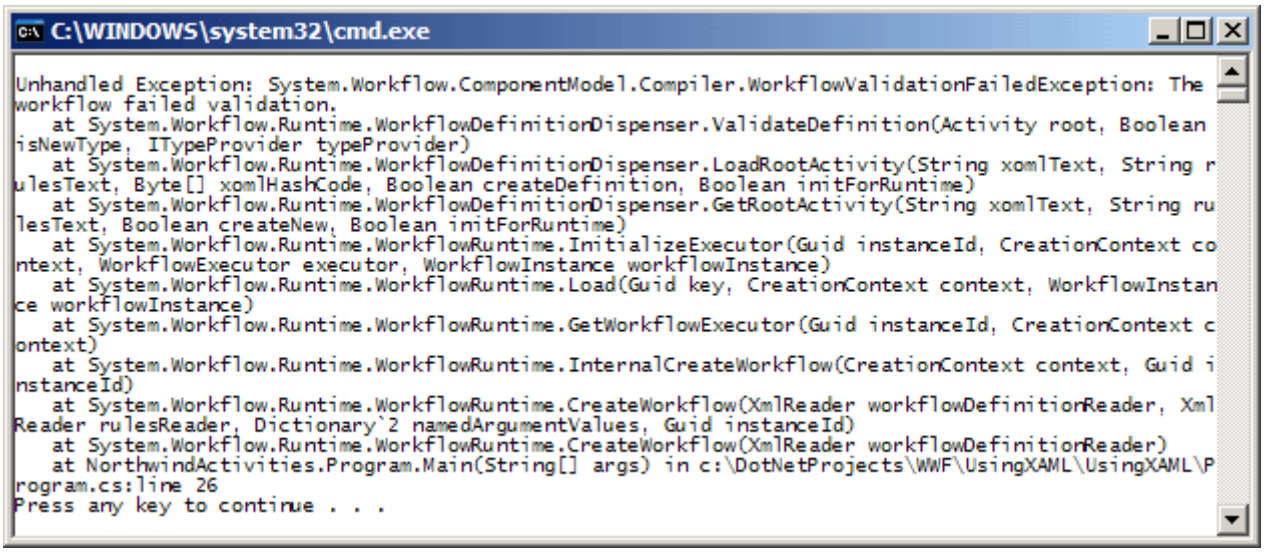


Evettt...Herşey hazır gibi. Mi acaba? Aslında unuttuğum önemli bir nokta var. Söz konusu **Workflow** nasıl çalıştırılacak? Nitekim, **build** işlemi sırasında **ProductOrderFlow.xoml**içeriğini devre dışı bıraktığımızdan, bunun çalışma zamanında bir şekilde yükleniyor olması gerekiyor. Ancak derlenmiş kod içerisinde bu Workflow' a ait bir tip tanımlamasıda yer almadığından*(çünkü Build Action=None olarak belirlendi)* çalışma zamanında **xoml** dosyasının içeriğinin ele alınması gerekmekte. Bunu sağlamak için tek yapılması gereken çalışma zamanı kodlamasını aşağıdaki gibi değiştirmek.

```
using System;
using System.Threading;
using System.Workflow.Runtime;
using System.Xml;

namespace NorthwindActivities
{
   class Program
   {
      static void Main(string[] args)
      {
         using(WorkflowRuntime workflowRuntime = new WorkflowRuntime())
         {
            AutoResetEvent waitHandle = new AutoResetEvent(false);
            workflowRuntime.WorkflowCompleted += delegate(object sender,
WorkflowCompletedEventArgs e) {
               waitHandle.Set();
               Console.WriteLine("İşlemler tamamlandı");
            };
            workflowRuntime.WorkflowTerminated += delegate(object sender,
WorkflowTerminatedEventArgs e)
            {
               Console.WriteLine(e.Exception.Message);
               waitHandle.Set();
            };

            XmlReader reader = XmlReader.Create("..\\..\\ProductOrderFlow.xoml");
            WorkflowInstance instance = workflowRuntime.CreateWorkflow(
               reader
               );
            instance.Start();

            waitHandle.WaitOne();
         }
      }
```

```
    }
}
```

Koddanda görüldüğü gibi yapılması gereken, **xoml** içeriğini **XmlReader** nesnesi yardımıyla ortama almak ve **WorkflowInstance** örneğinin oluşturulması sırasında**CreateWorkflow** metoduna parametre olarak vermektir. Uygulamayı bu haliyle çalıştırdığımda aşağıdaki sonucu elde ettim.



Sanıyorumki ben dahil herkes, uygulamanın çalışmasını bekliyordu. Ancak yukarıda görüldüğü gibi bir istisna(Exception) aldım. Karabasan devam ediyordu sanki. çözümü bulmam biraz zamanımı aldı. Aslında problem, xoml içeriğinde yer alan

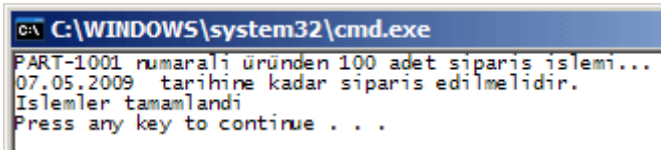
x:Class="NorthwindActivities.ProductOrderFlow"

bildirimiydi. çalışma zamanının kızması son derece doğaldı. Hak vermem gerekiyordu. Hata mesajındanda anlaşılacağı üzere bir doğrulama(Validation) sorunu vardı. Bunun kaynadığında ise **ProductOrderFlow** tipi yer almakta. Derken tepemde bir ampül yanıverdi. **cs** dosyasını çıkarmış ve **xoml** içeriğini uygulamaya dahil etmemiştim. Dolayısıyla söz konusu tip zaten yoktu ve çalışma zamanı, akışı doğrulamaya çalışırken tam bu noktada çatlıyordu. Neden böyle olmuştu peki? Tabiki **Visual Studio** ortamında sadece **XAML**içeriğinden oluşan bir akış geliştirme desteği bulunmamaktaydı ve ben kod parçalı oluşturulan akışın üzerinde değişiklikler yapıyordum. Yani varsayılan modele karşı gelmiştim. Haliyle xoml içeriğini aşağıdaki gibi değiştirmem gerekti.

```
<SequentialWorkflowActivity x:Name="ProductOrderFlow" xmlns:ns0="clr-namespace:NorthwindActivities;Assembly=NorthwindActivities, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/workflow">
 <DelayActivity TimeoutDuration="00:00:05" x:Name="delayActivity1" />
```

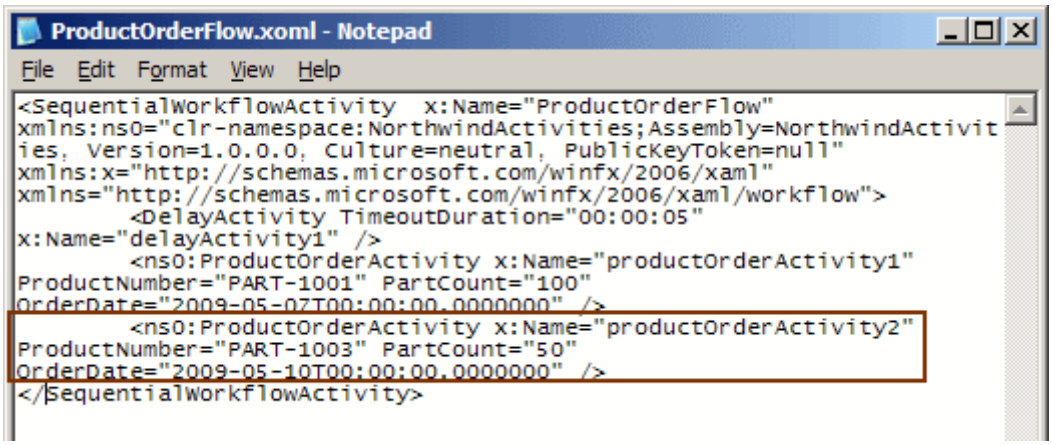
<ns0:ProductOrderActivity x:Name="productOrderActivity1" ProductNumber="PART-1001" PartCount="100" OrderDate="2009-05-07T00:00:00.0000000" />
</SequentialWorkflowActivity>

Artık tekrar testi yapabilirdim. Programı çalıştırdığımda aşağıdaki sonuçla karşılaştım.

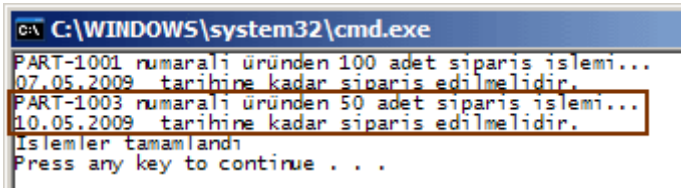


Nihayet

Artık xoml içeriği ile biraz oynayabilirdim. Bu amaçla xoml dosyasını notepad ile açtım ve aşağıdaki hale getirdim.



Görüldüğü gibi **productOrderActivity2** isimli yeni bir bileşeni akış içerisine dahil edip özelliklerine sembolik değerler atadım. Bundan sonra program kodunu **derlemeden** çalıştırdığımdaysa aşağıdaki ekran görüntüsü ile karşılaştım.



Görüldüğü gibi yeni eklenen bileşende başarılı bir şekilde çalıştırıldı. Sonuç olarak; bir workflow örneğinin koddan bağımsız olacak şekilde tasarlanabilmesi, **XAML**içeriğinin basit bir editor yardımıyla değiştirilip akışın güncellenebilmesi sağlanabilmektedir. Burada önemli olan noktalardan birisi, söz konusu **xoml** dosyalarının, çalışma zamanında değerlendirilip yürütülmeleridir.

Şimdi şöyle bir senaryoyu göz önüne alalım. Buradaki gibi tamamen XAML bazlı akışların bir depoda saklandığını düşünelim. örneğin **SQL** sunucusu üzerinde veya bir x veri depolama sisteminde. Sonrasında ise, bu akışları kullanan(içeren) süreçler ve programların

görsel bir araç yardımıyla tasarlanabildiğini ve değiştirilebildiğini göz önüne alalım. Hatta bu akışların istenirse **export** edilip farklı uygulama alanlarına **import** edilebildiklerini farz edelim...Derken zaten **Microsoft'** un gitmek istediği noktada yer alan bir kaç temel ihtiyaçtan bir kısmını özetlemiş oluyoruz.

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

UsingXAML.rar (43,63 kb)

REST Starter Kit Nedir? (2009-05-05T21:41:00)

wcf,rest,

Merhaba Arkadaşlar,

Bildiğiniz üzere bir süredir WCF REST Starter Kit ile ilişkili yazılarımı ve görsel derslerimi sizlerle paylaşmaktayım. Ancak önemli bir noktayı kaçırdığımı düşünüyorum. Nedir bu **WCF REST Starter Kit? Bizlere ne gibi avantajlar getirmektedir?**

WCF REST Starter Kit temel olarak, WCF servislerinin **REST(REpresentational State Transfer)** bazlı olaraktan geliştirilmesi için gerekli özellik ve şablonları**(Visual Studio Templates)** içermekte olan bir yardımcı araç kitidir. Bu kit **.Net Framework 3.5 ve SP1** ile birlikte, WCF tarafına kazandırılan Web programlama modelini kullanır ve geliştiricinin, REST bazlı WCF servislerinin yazılması ve tüketilmesi sırasında gerekli olan bir çok kıstasın(**aspect** olarakta düşünebiliriz) kolayca ele alabilmesini hedefler. WCF Rest Starter Kit, konsept olarak WCF servislerini ve REST modelini hedef almakla birlikte özellikle açık kaynak kodlu olması açısındanda önemli bir eklenti olarak düşünülmelidir.

NOT : *Aslında Xml Web Servisleri içinde zamanında Web Service Enhancements adıyla yayımlanmış bir yardımcı kit bulunmaktadır.* ***WSE'*** *nin en büyük amacı,* ***Xml Web Servislerinde*** *çok fazla kodlamayı gerektiren bazı standart kıstasların****(Transaction Flow, Security, MTOM bazlı dosya aktarımı vb...)*** *nitelik veya konfigurasyon bazında daha kolay bir şekilde uygulanabilmesidir.*

Kit aslında iki ana bölümden oluşmaktadır.

1- Sunucu Tarafı Yetenekleri; REST tabanlı WCF servislerinin host edildiği sunucu tarafı ile ilgili özelliklerdir. örneğin,

- dekleratif olarak ön bellekleme özelliklerini tanımlayabilmek(Declarative Caching),
- hata yönetimi(Error Handling),
- güvenlik(Security),
- yardım desteği(Help Page Support),
- büyük boyutlu dosyaların sunucu kaynaklarını fazla yormada taşınması(push style

streaming)...

Bunlara ek olarak **Visual Studio 2008** ortamına eklenen pek çok şablonda yer almaktadır.

- Koleksiyon bazlı servisler**(Rest Collection Wcf Service),**
- tek kaynaklı servisler**(Rest Singleton Wcf Service),**
- Atom protokolü bazlı servisler**(Atom Publishing Protocol WCF Service),**
- Atom bazlı içerik yayınlama servisi**(Atom Feed WCF Service),**
- ve basit olarak sadece HTTP bazlı taleplere cevap verecek olan WCF Servisleri**(HTTP Plain XML WCF Service)**

2- İstemci Tarafı Yetenekleri; İstemcilerin, REST tabanlı WCF servislerini kolayca kullanabilmeleri için gerekli özelliklerdir. REST Starter Kit'in ikinci versiyonunda, istemci tarafından REST mesajlarının gönderilmesi ve cevapların işlenmesini kolaylaştıracak şekilde **HttpClient** isimli yeni bir sınıf geliştirilmiştir. Ayrıca **Visual Studio IDE'** sine **Paste Xml As Type** isimli bir diğer özellik katılarak(add-in), XML içeriklerinin(XSD şemasıda kullanılabilir) serileştirilebilir tip haline dönüştürülmesi de son derece kolaylaştırılmaktadır. Bu sayede servis tarafından yayımlanan bir XML içeriğin copy-paste tekniğini kullanarak ve Paste Xml As Type seçeneğinden yararlanarak, managed tarafta ele alınabilecek bir tip haline getirebiliriz.

Bu iki ayrım haricinde, Starter Kit ile birlikte gelip göze çarpan bir kaç noktayıda, aşağıdaki gibi özetleyebiliriz.

çıktı formatları için destek(Representation Format) : Günümüz web uygulamalarında, istemci tarafının ele aldığı en popular içerik formatları, **XML(Xtensible Markup Language)** ve **JSON(JavaScript Object Notation)** dır. REST bazlı WCF servisleri zaten bu tipte yayınlama yapma yeteneğine sahiptir. Starter Kit' in burada kattığı diğer bir yetenek ise, istemciden gelen **HTTP-GET** talebine bakarak geriye XML veya JSON formatında içerik gönderilmesini kolaylaştırmaktır.

Dekleratif ön bellekleme(Declerative Caching) : Deklerafit kelimesinin burada kattığı anlam aslında, söz konusu özelliğin bir aspect olarak ele alınabilmesidir. Buna gore **nitelik(attribute)** ve**konfigurasyon**(web.config örneğin) bazında önbellekleme ile ilişkili bildirimler kolayca yapılabilir. Starter Kit burada işlemleri kolaylaştırmak için **WebCache** niteliğini sunmaktadır.

Yardım Desteği(Help Support) : REST bazlı WCF servislerinin istemci tarafından nasıl kullanılabileceğinin bilinmesi önemlidir. Burada servise giden taleplerin **URI** formatında olduğu düşünüldüğünde, servisin sunduğu operasyonların URI taleplerinin ne olacağı, operasyonun özet olarak ne yaptığı gibi bilgileri istemciye sunmak için REST starter kit **/help** sayfa desteğini getirmektedir. Burada, operasyonlar açısından önem arz eden konulardan biriside **WebHelp** niteliği yardımıyla özet bilgilerin belirtilmesi ve yardım sayfasında gösterilmelerinin sağlanmasıdır.

Hata Desteği(Error Handling) : Servis tarafında bir hata oluştuğunda bunun istemci tarafında string veya geliştirici tarafından yazılmış bir .Net tipi olarak XML veya JSON formatında döndürülebilmesi, starter kit ile dahada kolaylaştırılmaktadır.
Burada **WebProtocolException** tipinden yararlanılmaktadır.
Tabi **JSON** veya **XML** tipinden dönüş
olacağına **WebGet**veya **WebInvoke** niteliklerindeki, **ResponseFormat** özelliğine atanan değer ile karar verilir.

Service Host: REST Starter kit, daha az konfigurasyon ile çalışma zamanının pek çok değerinin ayarlanabilmesini sağlamaktadır. Bu nedenle REST Starter Kit, **WebServiceHost2** isimli bir tip içermektedir. Bu tipi, **Visual Studio 2008** altındaki REST şablonlarını oluşturduğumuzda, svc öğelerine ait markup dosyaları içerisinde görebiliriz.

Güvenlik (Security) : REST bazlı WCF servislerini, starter kit ile geliştirirken, **Asp.Net** güvenlik özelliklerden yararlanılabilir. Böylece, örneğin **form tabanlı doğrulama(Form Based Authentication)** veya **rol bazlı yetkilendirme(Role Based Authorization)** yapılabilir. Bu işlemler için Asp.Net' in standart **Membership API**' sinden yararlanılabilir yada özel provider' lar kullanılabilir.

HttpClient : İstemci tarafına getirilen bu tip ile, REST bazlı WCF servislerini **HTTP Put, Get, Delete, Post** metodlarına göre kullanmak dahada kolaylaşmaktadır. Ayrıca asenkron programlama**(async programming)** modeline de destek verilmektedir. üstelik, **olay bazlı asenkron programlama(Event Based Asnyc Programming)** modelide kullanılabilmektedir. Ayrıca **Atom**protokolüne gore yayın yapan bir servisin istemci tarafından ele alınması için **AtomPubClient** sınıfıda HttpClient tipini kullanmak üzere, REST Starter Kit' e getirilmiş genişletmelerden birisidir.

Tabiki daha pek çok detay bulunmaktadır. Ancak sanıyorumki bu kısa bilgiler sizlerde WCF Rest Starter Kit hakkında bir fikir oluşturmuştur.

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

C# 4.0 - Seçilebilen, İsimlendirilebilen Parametreler(Named and Optional Parameters), ref' i Görmezden Gelmek(Ommit Ref) ve PIA için Yenilikler (2009-05-04T22:46:00)

c# 4.0,

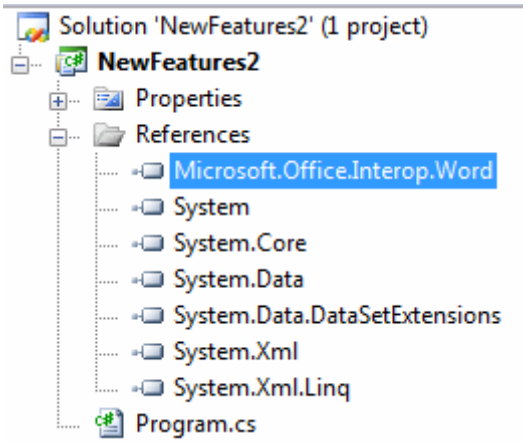
Merhaba Arkadaşlar,

Bir önceki blog yazımızda **C# 4.0** ile birlikte gelen önemli yeniliklerden birisi olan **dynamic** kavramına değinmeye çalışmıştık. Elbetteki C# 4.0 ile birlikte gelen başka yeniliklerde var. Bu yeniliklerde, diğerleri gibi belirli ihtiyaçlardan ortaya çıkmıştır.

öncelikli olarak bu ihtiyaçları ortaya koymaya çalışıyor olacağız. Bu nedenle **PDC 2008'de** dağıtılan **Visual Studio 2010(PreBeta)** sürümü ile yazdığım aşağıdaki kod parçasını bir süreliğine göz önüne alalım.

```
using System;
using System.Reflection;
using Word=Microsoft.Office.Interop.Word;

namespace NewFeatures2
{
   class Program
   {
      static void Main(string[] args)
      {
         Word.Application wrdApp = new
Microsoft.Office.Interop.Word.Application();
         wrdApp.Visible = true;
         object fileNamePath = @"C:\Yeni Ozellikler.docx";
         object missingValue = Missing.Value;

         wrdApp
            .Documents
            .Open(ref fileNamePath, ref missingValue, ref missingValue, ref
missingValue, ref missingValue, ref missingValue, ref missingValue, ref missingValue,
ref missingValue, ref missingValue, ref missingValue, ref missingValue, ref
missingValue, ref missingValue, ref missingValue, ref missingValue);
         Console.WriteLine("Kapatmak için bir tuşa basınız");
         Console.ReadLine();
      }
   }
}
```

İlk olarak şunu belirtmek isterim; bu basit Console uygulamasında **Microsoft.Office.Interop.Word** assembly' ına ait bir referans yer almaktadır.



Program, sistemde yüklü olan **Yeni Ozellikler.docx** isimli bir **Word** dosyasını açmak için gerekli kodları içermektedir. Bu Word dosyasının açılması içinse,**Microsoft.Office.Interop.Word** assembly' ından yaralanılmaktadır. Bu assembly aslında Word ile konuşabilmemizi sağlayan COM API'sini sarmalayan(Wrap) bir **yönetimli(Managed)**kütüpanedir. Kod içerisinden, Word dökümanına erişebilmek için **Application** tipinden bir nesne oluşturulmaktadır. Uygulamanın görünürlüğü Visible özelliği ile set edildikten sonra ise, **Open** metodundan yararlanarak ilgili Word belgesinin açılması sağlanmaktadır. Ancak bu kod parçasında geliştirici açısından bazı zorluklar olduğu rahatlıkla gözlemlenebilir.

1- Open metoduna ait **16** adet parametrenin tamamının girilmesi zorunludur. Aşağıdaki şekilde durumun sıkıcılığı gözler önündedir.

__NOT :__ Burada __Juval Lowy__' nin __IDesign__ şirketinde kullandığı ve pek çok şirket tarafından benimsenen C# kodlama standartları aklıma geliyor. Buradaki belirttiği bir maddeyi hatırlıyorum. __"Metodların argüman sayılarının 5 i geçmesinden kaçının. Eğer öyleyse struct tipini kullanın".__ Burada bir __COM API__' sinin __Wrap__ edilmiş kütüphanesi içerisindeki bir fonksiyonun parametre yapısının değiştirilemeyeceği fikri belleğimizi tamamıyla kaplayabilir. Tabi __C# 3.0__ sonrasında bir fırsat olabileceği de akıllara gelebilir. __"Bir Extension metod yardımıyla Open metodu yerine bir alternatif geliştirebilir miyim? En azından parametre sayısını düşürmemizi kolaylaştıracak..."__ Bunu denemenizi öneririm.

2- Parametreler **COM** nesnesine iletildiğinden, dışarıdan yapılacak olan atamalarda **object** tipinin kullanılması gerekmektedir.

3- ref anahtar kelimesinin kullanılması zorunludur.

Gerçekte, **Open** metodu içerisinde işimize yarayan ve bizim için anlam ifade eden tek bir parametre yer almaktadır. O da açılmak istenen dosyanın adıdır. Diğer parametrelerinin hiçbirini kullanmadığımız halde yazmak zorunda olduğumuzu görüyoruz. Keşke sadece gerekli olanları yazsabilseydik; o zaman bu iş daha kolay olmazmıydı? ☹ Nitekim buradaki Open metodu haricinde, çok daha fazla sayıda argüman ile çalışabilen COM fonksiyonellikleri söz konusu olabilir. Böyle bir durumda tam olarak tüm parametreleri yazma zorunluluğu bir kenara dursun, bunların bütünün ne işe yaradığınında bilinmesi gerekir.

Sanırım bu cümlelerden zaten nereye varmak istediğimi anlatabilmişimdir. .Net in gelecek nesillerinin en büyük hedeflerinden birisi dinamik dillere ait nesneler ile konuşabilmek ve bunu mümkün olduğunca kolaylaştırmaktır. Bu noktada COM API' leri gibi nesnelerinde kullanımı söz konusudur. Aynen yukarıda geliştirdiğimiz örnekte olduğu gibi. Bu nedenle **C# 4.0** içerisinde seçimlik parametre kullanımına izin veren geliştirmeler yapılmıştır**(Optional Parameters)** Buna göre yukarıdaki kod parçasını C# 4.0 stilinde aşağıdaki gibi geliştirebiliriz.

Optional Parameters ile

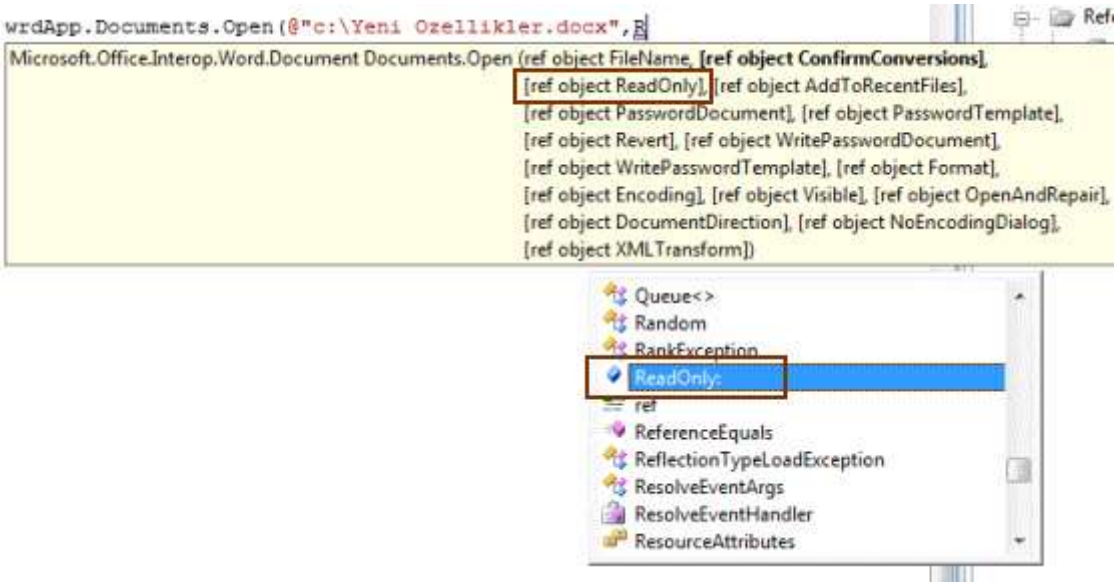
```
using System;
using System.Reflection;
using Word=Microsoft.Office.Interop.Word;

namespace NewFeatures2
{
   class Program
   {
      static void Main(string[] args)
      {
         Word.Application wrdApp = new
Microsoft.Office.Interop.Word.Application();
         wrdApp.Visible = true;
         wrdApp.Documents.Open(@"C:\Yeni Ozellikler.docx");
         Console.WriteLine("Kapatmak için bir tuşa basınız");
         Console.ReadLine();
      }
   }
}
```

Bu kod parçası çalıştığında da aynı sonucu alırız. Yine Word belgesi açılacak ve içeriği görüntülenecektir. Hem kodun okunurluğu kolaylaşmıştır, hem de kısalmıştır. Diğer taraftan parametre değerini aktarırken **ref** kullanılmadığına dikkat etmemiz gerekiyor**.(Ommit ref özelliği)** üstelik **object** tipinden değişken ataması yerine doğrudan dosya adresininin içeriğini gönderebildiğimizede dikkat edelim.

Tabi ihtiyaçlar bitmek bilmiyor. Burada görüldüğü gibi gereksiz olan parametrelerin hiç biri bildirilmemiştir. Ayrıca **ref** anahtar kelimeside herhangibir şekilde kullanılmamıştır. Ancak arada başka bir parametrenin daha kullanılması gerekirse... Söz gelimi 3ncü parametre dosyanın yanlız okunabilir**(ReadOnly)** modda açılıp açılmayacağını belirtir. **Optional Parameter**tekniğini kullanırsak ikinci parametreyi atlamamız mümkün olmayacaktır. Acaba böyle bir vakada kodu yine istemediğimiz şekliyle aşağıdaki gibi geliştirmemiz mi gerekir?

```
using System;
using System.Reflection;
using Word=Microsoft.Office.Interop.Word;

namespace NewFeatures2
{
   class Program
   {
      static void Main(string[] args)
      {
         Word.Application wrdApp = new
Microsoft.Office.Interop.Word.Application();
         wrdApp.Visible = true;
         object fileNamePath = @"C:\Yeni Ozellikler.docx";
         object missingValue = Missing.Value;
         object onlyRead = true;

         wrdApp
            .Documents
            .Open(ref fileNamePath, ref missingValue, ref onlyRead, ref missingValue, ref
missingValue, ref missingValue, ref missingValue, ref missingValue, ref missingValue, ref
missingValue, ref missingValue, ref missingValue, ref missingValue, ref missingValue, ref
missingValue, ref missingValue, ref missingValue);

         Console.WriteLine("Kapatmak için bir tuşa basınız");
         Console.ReadLine();
      }
   }
}
```

Oysaki C# 4.0 bu gibi durumlar için **isimlendirilmiş parametre(Named Parameters)** kullanımını olanaklı kılmaktadır. Aşağıdaki şekilde görüldüğü gibi, **intellisense'** de bize yardımcı olmaktadır.

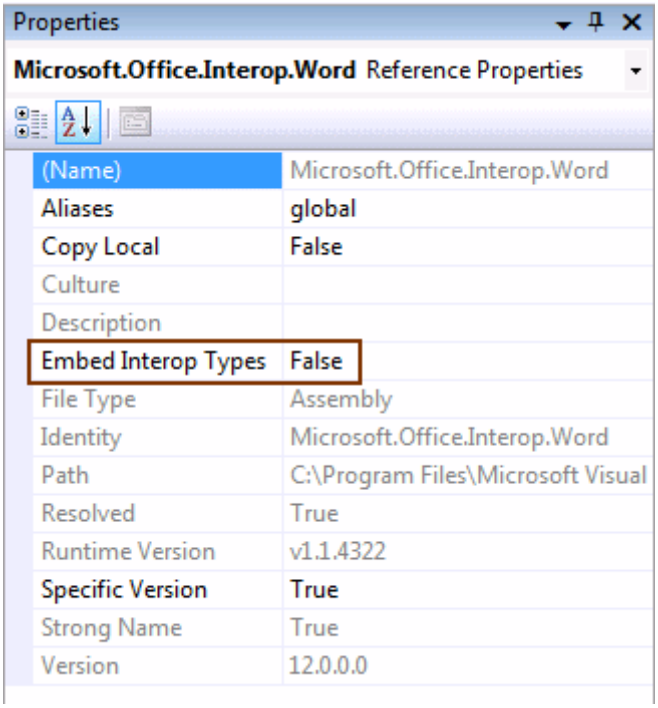


Dolayısıyla yukarıdaki kod parçasını aşağıdaki gibi geliştirebiliriz.

Named Parameters kullanımı ile

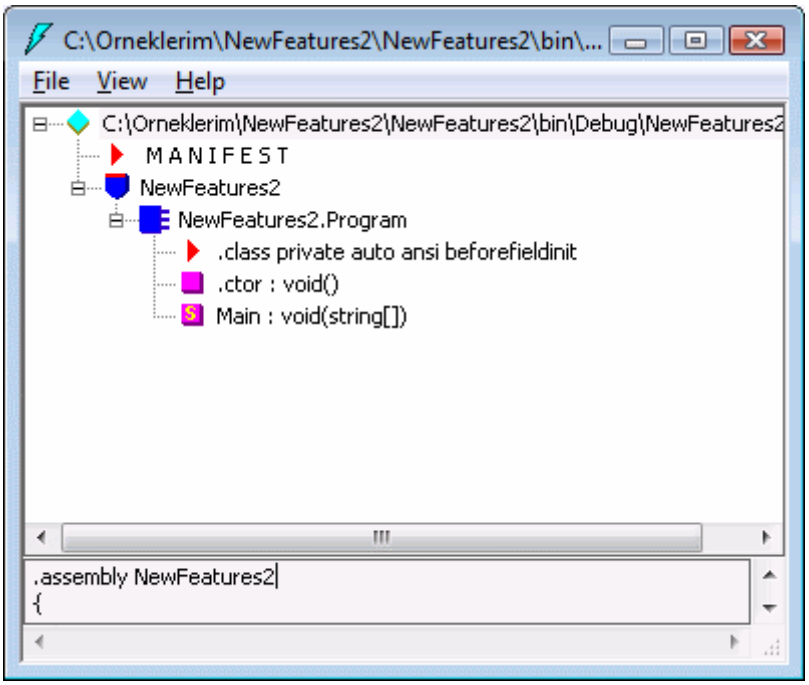
```
using System;
using System.Reflection;
using Word=Microsoft.Office.Interop.Word;

namespace NewFeatures2
{
   class Program
   {
      static void Main(string[] args)
      {
         Word.Application wrdApp = new
Microsoft.Office.Interop.Word.Application();
         wrdApp.Visible = true;

         wrdApp.Documents.Open(@"c:\Yeni Ozellikler.docx", ReadOnly: true);

         Console.WriteLine("Kapatmak için bir tuşa basınız");
         Console.ReadLine();
      }
   }
}
```

Son olarak **Platform Interop Assembly(PIA)** ile ilgili gelen yeniliklerden birisine değinmek istiyorum. Normal şartlarda **Visual Studio 2010** öncesinde bir COM API' sini uygulamaya referans ettiğimizde, sarmalanan kütüphanenin özelliklerinde aşağıdaki şekilde görülen **Embed Interop Types** isimli bir kriter olmadığı bilinmektedir.

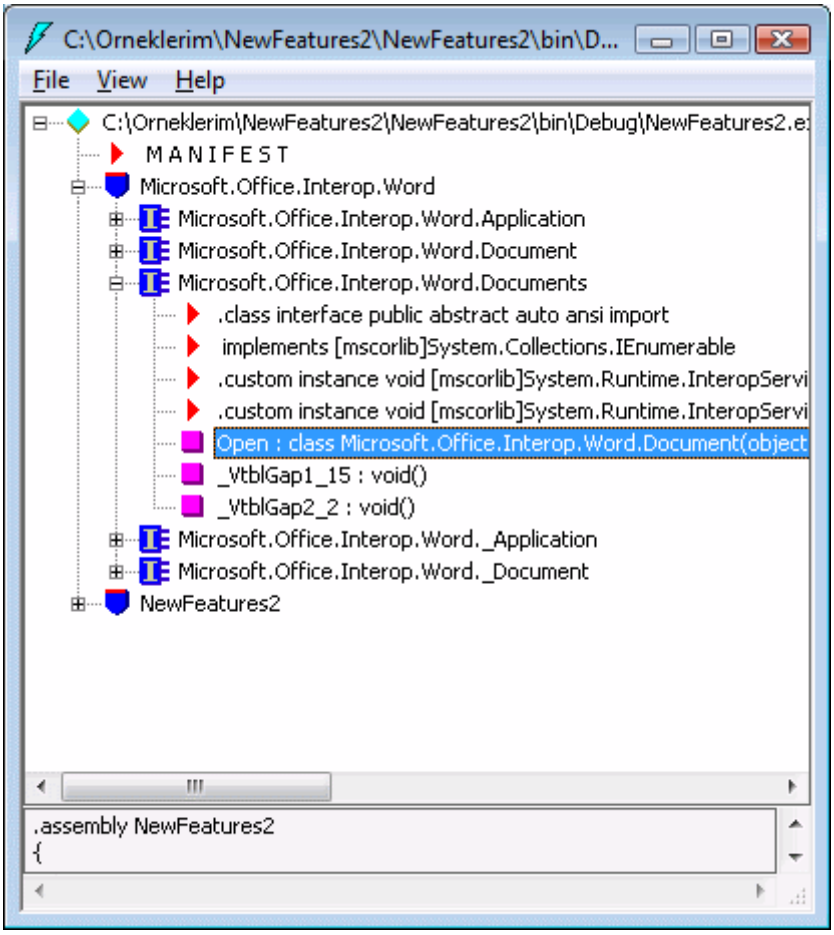


Oysaki Visual Studio 2010 ile birlikte bu özellikte gelmektedir. Bu tabiki sadece **C# 4.0** diline bağlanacak bir yetenek olarak düşünülmemelidir.

Peki ne işe yarar? Eğer yukarıda geliştirdiğimiz **C# 4.0** örneğinin **ildasm(Intermediate Language DisAsseMbler)** çıktısına bakacak olursak aşağıdaki durum ile karşılaşırız.



Göze çarpan özel bir nokta yer almamaktadır.
Ancak **Microsoft.Office.Interop.Word** assembly' ının özelliklerinde yer alan **Embed Interop Types** seçeneğini true olarak değiştirir ve söz konusu uygulamanın **IL** çıktısına tekrardan bakarsak aşağıdak sonuçlarla karşılaşırız.



Görüldüğü gibi API içerisinde yer alan tipler, **.Net** programı içerisine birer tip olarak gömülmüştür. Aslında bu yenilik, **PIA**' ların, geliştirilen asıl uygulama içerisine tip bazında gömülerekten taşınabilmelerini kolaylaştırıcı bir özellik olarak görülebilir. Bu konudaki araştırmalarıma devam ediyorum. Yeni bilgiler kazandıkça sizlerle paylaşmaya devam ediyor olacağım.

Böylece geldik bir yazımızın daha sonuna. Bu yazımızda sizlere C# 4.0 ile birlikte gelen bir kaç yeniliği aktarmaya çalıştım. Sonuç olarak bu yeniliklerin özellikle dynamic tiplerin kullanımı kolaylaştırmak üzere getirildiğini söyleyebiliriz.

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

C# 4.0 - Dynamic Olmak (2009-05-01T00:02:00)

c# 4.0,

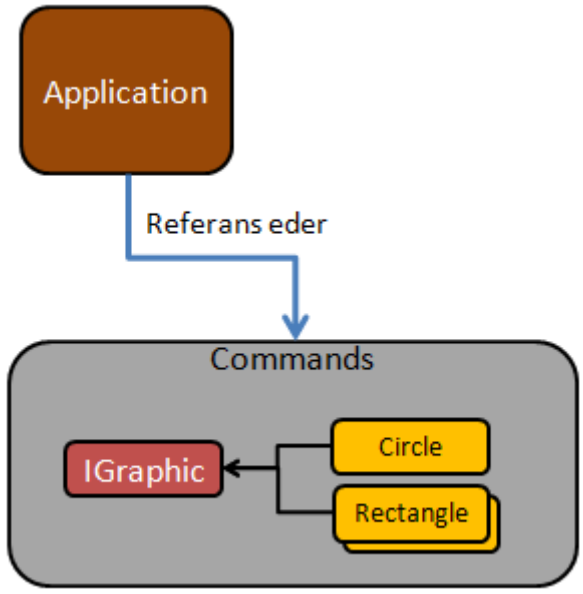
Merhaba Arkadaşlar,

Uzun bir süredir (son bir senelik zaman dilimi içerisinde) **C# 4.0** ile birlike gelen yeniliklerden haberdarız. Şöyle bir kaç sene öncesini hatırlıyorum da...😎 **Visual Studio 2005, Whidbey** kod adı ile yayınlanmış ve **C# 2.0** ile birlikte gelen pek çok yenilik

olmuştu. Ancak bunlar içerisinde belkide en önemli olanı, **CLR(Common Language Runtime)** çekirdiğinde değiştirilme yapılmasını da zorunlu kılan **generic** mimari kavramıydı. Tabiki generic dışında gelen, **yield** anahtar kelimesi, **isimsiz metodlar(anonymous methods), static sınıflar** ve diğerleride önemli gelişmelerdi. Zaman ilerledi ve **C# 3.0** ile birlikte bu kez hayatımıza, generic modelinden daha fazla etki yapan **LINQ(Language INtegrated Query)** girdi. Bir geliştirici olarak her zaman için yeniliklere açık olmamız ve yakalayabildiğimiz ölçüde takip etmemiz gerektiğini düşünüyorum. Bu bir geliştirici için neredeyse bir yaşam tarzı. Dolayısıyla artık **C# 4.0** üzerinde konuşmanın zamanı geldide geçiyor.

C# 4.0 ile birlikte gelen yeniliklerin daha çok **dinamik çalışma zamanını(Dynamic Language Runtime-DLR)** kullanan diller üzerinde odaklanmış durumda olduğunu söyleyebiliriz. Peki bu ne anlama geliyor? **DLR** tarafını ilgilendiren dillere ait nesneler ile daha kolay konuşulması olarak küçük bir sebep belirtebiliriz. Bu nedenle **C# 4.0** ile birlikte gelen önemli yeniliklerden birisi olan **dynamic**anahtar kelimesi sayesinde, **Python, Ruby** veya **Javascript** ile üretilen nesnelerin **C# 4.0** tarafında **late-binding** ile ele alınması mümkün. Hatta var olan **.Net** nesnelerinin **reflection** kullanılmadan ele alınması veya **COM** objelerine ait üyelerin çağırılmasında bu anahtar kelimeyi kullanabiliyoruz. Aslında C#' ın 2.0, 3.0 versiyonunda gelen yenilikler nasıl ki belirli ihtiyaçlar nedeni ile ortaya çıkmışsa, C# 4.0 ile gelen yenilikleride bu anlamda düşünmemiz ve araştırmamız gerekiyor.

Bu yazımda sizlerle dynamic kelimesi ile ilgili olan araştırmalarım sonucu elde ettiğim bilgileri paylaşıyor olacağım. İşe ilk olarak aşağıdaki şekilde görülen yapıya sahip olduğumuzu düşünerek başlayacağız.



Şimdi bu yapıyı kısaca açıklayalım. **Commands** isimli **sınıf kütüphanesi(Class Library) IGraphic arayüzünü(Interface)** uygulayan **Circle** ve **Rectangle** isimli sınıflara sahiptir.

IGraphic arayüzü

```
namespace Commands
{
   public interface IGraphic
   {
      void Draw();
   }
}
```

Circle sınıfı

```
using System;

namespace Commands
{
   public class Circle
      :IGraphic
   {
      #region IGraphic Members

      public void Draw()
      {
         Console.WriteLine("Circle...");
      }

      #endregion
   }
}
```

Rectangle Sınıfı

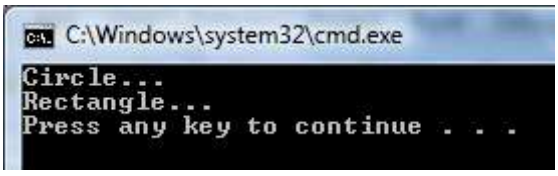
```
using System;

namespace Commands
{
   public class Rectangle
      :IGraphic
   {
      #region IGraphic Members

      public void Draw()
      {
         Console.WriteLine("Rectangle...");
      }
```

```
      #endregion
   }
}
```

Console Application tipinden olan uygulamamız, **Commands** isimli sınıf kütüphanesini referans etmekte olup başlangıçta aşağıdaki kod içeriğine sahiptir.

```
using Commands;

namespace CSharp4Features
{
   class Program
   {
      static void Draw<T>(T graphObject)
         where T : IGraphic
      {
         graphObject.Draw();
      }
      static void Main(string[] args)
      {
         #region Başlangıçtaki durumumuz

         Draw<Circle>(new Circle());
         Draw<Rectangle>(new Rectangle());

         #endregion
      }
   }
}
```

Uygulamayı çalıştırdığımızda aşağıdaki sonucu alırız.

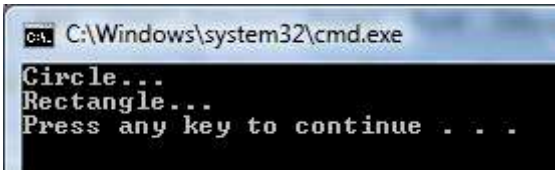


Bu kod parçasında dikkat edilmesi gereken önemli noktalardan birisi, **generic Draw<T>** metodudur. Burada yer alan generic T tipine **IGraphic** arayüzünden türeme koşulu getirilmiştir. Bu sebepten dolayı, **IGraphic** arayüzünü uygulayan tüm tiplere ait **Draw** metounu çağırmamızı sağlayan tek bir metod geliştirmiş oluyoruz. Dolayısıyla tek yapılması gereken, **Draw<T>**metodunun kullanıldığı yerde, doğru tipe ait*(bu örnek için Rectangle veya Circle)* nesne örneğini parametre olarak aktarmaktır.

Şimdi burada, bizi dynamic kelimesine götürecek bir veya bir kaç sebep arayacağız. İşte bir kaç blog içerisinde yakaladığım ortak soru geliyor...Ya **Console** uygulaması, **IGraphic**arayüzüne erişemiyor olsaydı. Bunu ayarlamak son derece kolay. Tek yapmamız gereken IGraphic arayüzünün **public** olan erişim belirleyicisini kaldırmak.*(Bir başka deyişle internal'a çekmek)* Bu durumda **Draw<T>** metodumuz için **derleme zamanı(Compile Time)** hatası alınacaktır. Peki ne yapılabilir? **Reflection** tekniklerinden yararlanarak ilgili tipin **Draw** metodunun çağırılması sağlanabilir. Yani kodu aşağıdaki hale getirebiliriz.

```
using Commands;
using System.Reflection;

namespace CSharp4Features
{
   class Program
   {
      static void Draw<T>(T graphObject)
      {
         MethodInfo methodInfo = typeof(T).GetMethod("Draw");
         if (methodInfo == null)
         {
            System.Console.WriteLine("Method bulunamadı");
         }
         methodInfo.Invoke(graphObject, new object[0]);
      }

      static void Main(string[] args)
      {
         Draw<Circle>(new Circle());
         Draw<Rectangle>(new Rectangle());
      }
   }
}
```

İlk olarak **typeof** metodu ile **T** tipi elde edilmekte ve **Draw** isimli metod istenerek **MethodInfo** tipinden bir referansa aktarılmaktadır. Bilindiği üzere **reflection** mimarisinde, çalışma zamanında tipler ve üyelerine ait bilgiler elde edilmekte ve istenirse üyelerin yürütülmesi*(örneğin metodların çağırılması)* sağlanabilmektedir. Bu nedenle ilk olarak **T** tipinin çalışma zamanı referansı üzerinden **Draw** metodu elde edilmeye çalışılır. Sonrasında ise eğer **MethodInfo** referansı **null** değilse **Invoke** fonksiyonuna gerekli parametreler gönderilerek **Draw** metodunun icra edilmesi sağlanır. *(Tabiki çalışma zamanında gelen T nesne örneğine ait olan Draw metodunun)* Uygulamayı bu haliyle çalıştırdığımızda yine aynı sonuçları alırız.

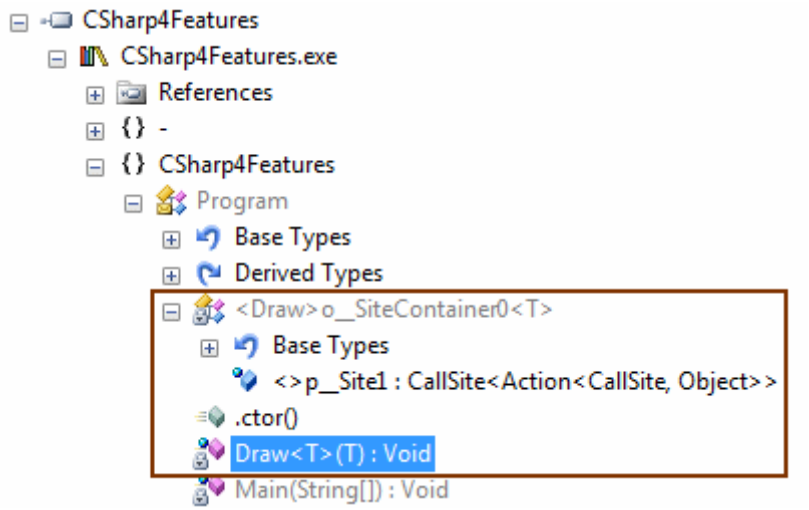


Ancak tabiki metodun bu yeni halinde tip güvenliğinden**(type-safety)** bahsetmemiz mümkün değildir. T için herhangibir tip kullanılabilir.

Peki dynamic anahtar kelimesi burada nasıl bir yaklaşım sunmaktadır. İşte aynı metodun **C# 4.0** için dynamic anahtar kelimesi ile yazılmış hali.

```
using Commands;
using System.Reflection;

namespace CSharp4Features
{
   class Program
   {
      static void Draw<T>(T graphObject)
      {
         dynamic obj = graphObject;
         obj.Draw();
      }

      static void Main(string[] args)
      {
         Draw<Circle>(new Circle());
         Draw<Rectangle>(new Rectangle());
      }
   }
}
```

Bu seferde aynı çıktıyı alırız. Tabi burada dikkat edilmesi gereken bir kaç nokta vardır ve kodun kısalmış olması bunlardan birisi değildir 🙂 öncelikli olarak **Draw** metodu, söz konusu **Circle** veya **Rectangle** nesne örneklerine çalışma zamanında bağlanmaktadır. Bu zaten bizim reflection tekniği ile yapmakta olduğumuz bir işlemdir. Diğer yandan **.Net Reflector** aracı yardımıyla üretilen uygulama koduna bakıldığında söz konusu metod için aşağıdaki **IL** çıktısının oluşturulduğunu görebiliriz.




```
private static void Draw<T>(T graphObject)
{
   object obj = graphObject;
   if (<Draw>o__SiteContainer0<T>.<>p__Site1 == null)
   {
      <Draw>o__SiteContainer0<T>.<>p__Site1 = CallSite<Action<CallSite,
object>>.Create(new CSharpInvokeMemberBinder(CSharpCallFlags.None, "Draw",
typeof(Program), null, new CSharpArgumentInfo[] { new
CSharpArgumentInfo(CSharpArgumentInfoFlags.None, null) }));
   }
   <Draw>o__SiteContainer0<T>.<>p__Site1.Target(<Draw>o__SiteContainer0<T>.<>p_
_Site1, obj);
}
```

Görüldüğü gibi **Draw<T>** metodu içerisinde **<Draw>o_SiteContainer0<T>** isimli generic bir tipin kullanıldığını ve bununda **IL(IntermediateLanguage)** tarafına eklendiğini görmekteyiz. Bir başka deyişle derleme işleminden sonra yine reflection kullanılan kod parçaları içeriye dahil edilerek, **Circle** veya **Rectangle** tiplerinden olan nesnelerin **Draw** metodunun çağırılması sağlanmış oldu.

NOT : *Burada dikkat edilmesi gereken önemli bir noktada şudur. Eğer Draw<T> metoduna, Circle ve Rectangle dışında bir tip atarsak(özellikle Draw metodu olmayan) bu durumda* ***RuntimeBinderException*** *tipinden bir istisna alırız.*

Tabiki bu kısım ve detaylarını daha iyi kavramak için belki biraz daha zamana ihtiyacımız olacak. Ancak bu anahtar kelimenin tek kullanım şeklinin, reflection ile elde edilen tiplere ait üyelerin çağırılmasını kolaylaştırmak olmadığınıda belirtmek isterim. öyleki, **dynamic** kelimesi ile **COM** objelerinin ve bu sayede **unmanaged API'** lerin dinamik olarak ele alınması mümkün olabilir. Hatta, **JSON** formatına sahip bir nesnenin dynamic kelimesi ile kolayca ele alınabileceğini söyleyebiliriz. Bu noktada **Office API'** sine ait nesnelerin dynamic kelimesi ile son derece etkili ve kolay kullanılabildiğini de belirtmek isterim. üstelik işin içerisine yine **C# 4.0** ile gelen opsiyonel ve

isimlendirilmiş parametreler**(Optional and Named Parameters)** adlı yeniliklerinde girdiğini söyleyebilirim. Bunu bir sonraki blog yazımda ele almaya çalışacağım.

özet olarak artık C# programlama dilinin, dinamik olarak türlendirilmiş tiplere ait nesnelerle daha kolay konuşabildiğini söyleyebiliriz.

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

NOT: örnekler 2008 PDC'de yayımlanmış olan Visual Studio 2010 PreBeta sürümü üzerinden geliştirilmiştir.

REST Bazlı WCF Servislerinde AdapterStream Kullanımı (2009-04-30T01:03:00)

wcf,rest,

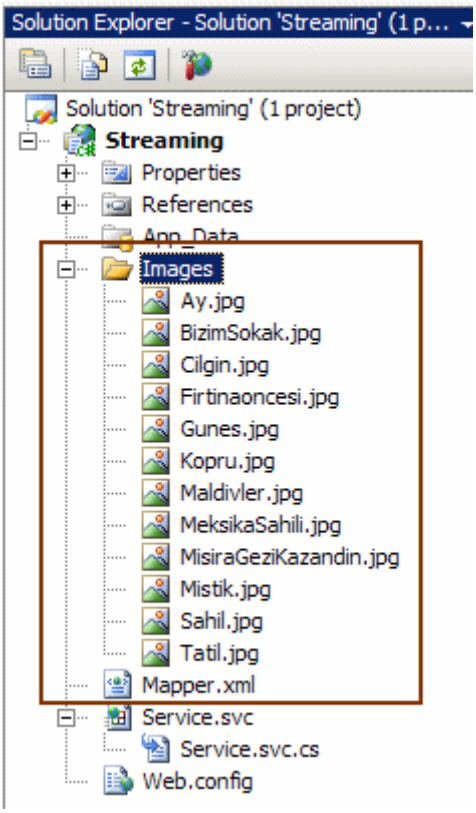
Merhaba Arkadaşlar,

REST bazlı WCF servislerinde zaman zaman istemcilere içerik boyutları yüksek olan çıktılar veriyor olabiliriz. Bunlara örnek olarak **resim** veya **metin** dosyaları verilebilir. Aslında **Stream** veya**TextWriter** bazlı içerikler dersek çok daha doğru olacaktır.*(Neden TextWriter olarak belirttiğimi yazının sonunda öğrenebileceğiz.)* özellikle istemci/sunucu bazlı uygulamalar göz önüne alındığında, büyük boyutlu içeriklerin karşı tarafa aktarılması sırasında karşılaşılabilecek pek çok performans kaybı söz konusudur. Sunucu tarafından bakıldığında, istemcinin talep ettiği içeriğin **Stream** olarak elde edilmesi sırasında bellek ve işlemci bazında yüklenmeler olabilir. Buda sunucunun performansının olumsuz yönde etkiliyebilir. Nitekim kaynakların israfı söz konusudur. Tabi istemci tarafı açısından bakıldığında da, gelen Stream içeriğinin işlenmesi esnasında bazı sıkıntılar ile karşılaşılabilir.

Biz bu yazımızda sunucu tarafındaki içeriğin Stream olarak elde edilmesi sırasında performans kazanımı için ne yapabileceğimize bakacağız. Neyse ki çok fazla uğraşmamıza gerek yok. Nitekim **WCF Rest Starter Kit** ile birlikte gelen **AdapterStream** sınıfı tam bu iş için geliştirilmiş bir tip. üstelik kit ile birlikte gelen örnek solution içerisinde kaynak kodunu görmenizde mümkün. Bu sınıf yardımıyla bir Stream' in hazırlanması sırasında, içeriğin tamamıyla değil, parça parça aktarılması sağlanabiliyor. Buda bir anlamda sunucunun bellek ve işlemci kaynaklarının daha az yorulması anlamına gelmekte.

Aslında hiç vakit kaybetmeden bu konu ile ilişkili geliştirdiğim örneği sizinle paylaşmak istiyorum. öncesinde nacizane senaryomdan biraz bahsedeyim. Servis tarafında yer alan basit bir operasyon, talep ile kendisine gelen kelimeye bakarak, istemci için bir duvar kağıdı resminin üretilmesini sağlamakta. Burada gelen kelimeleri çok basit bir düşünce ile bazı resim dosyaları ile eşleştirmekteyim. Eşleştirme için basit bir **XML** dökümanı kullanıyorum. Böylece servis tarafına yeni resimler ve bu resimlere eş düşecek kelimelerin koda müdahale etmeye gerek kalmadan eklenmesi mümkün olabilir. Kabaca aşağıdaki gibi bir durumdan bahsediyorum aslında.

Servis projesinin **klasör** yapısı,



ve **Mapper.xml** içeriği,

```
<?xml version="1.0" encoding="utf-8" ?>
<Mapper>
  <Map Keyword="ay" Image="images/Ay.jpg"/>
  <Map Keyword="bizimsokak" Image="images/BizimSokak.jpg"/>
  <Map Keyword="cilgin" Image="images/cilgin.jpg"/>
  <Map Keyword="firtinaoncesi" Image="images/firtinaoncesi.jpg"/>
  <Map Keyword="gunes" Image="images/gunes.jpg"/>
  <Map Keyword="kopru" Image="images/kopru.jpg"/>
  <Map Keyword="maldivler" Image="images/maldivler.jpg"/>
  <Map Keyword="meksikasahili" Image="images/meksikasahili.jpg"/>
  <Map Keyword="mistik" Image="images/mistik.jpg"/>
  <Map Keyword="sahil" Image="images/sahil.jpg"/>
  <Map Keyword="tatil" Image="images/tatil.jpg"/>
</Mapper>
```

Görüldüğü gibi **images** klasörü altındaki her bir resim ve eş düşen kelime, Xml içeriğinde tanımlanmış durumda. Tabi bu benim minik hayal gücümün bir ürünü. Buradaki sistem dahada etkili geliştirilebilir. Söz gelimi kullanıcının girdiği kelimeye göre, sunucu tarafında çalışacak akıllı bir robot, resim kataloğundan, kelime bire bir uymasa bile en yakın olanı bulup istemciye gönderebilir. Bu kısmı siz değerli okurlarıma bırakayım 🙂

Gelelim projenin kod yapısına. Burada **WCF Rest Starter Kit** kullandığımız için herhangibir REST şablonuna ait projelerden birisini oluşturmak yeterli. önemli olan noktalardan birisi servis tarafındaki çalışma zamanı için **WebServiceFactory2** isimli fabrikanın(Factory Class) kullanılmasıdır. O yüzden Servis' e ait markup içeriğinin aşağıdaki gibi olmasına özen göstermekte yarar vardır.

<%@ ServiceHost Language="C#" Debug="true" Service="Streaming.Service" Factory="Streaming.**AppServiceHostFactory**"%>

```
using System;
using System.ServiceModel;
using System.ServiceModel.Activation;
using Microsoft.ServiceModel.Web;
using Microsoft.ServiceModel.Web.SpecializedServices;

namespace Streaming
{
   class AppServiceHostFactory : ServiceHostFactory
   {
      protected override ServiceHost CreateServiceHost(Type serviceType, Uri[]
baseAddresses)
      {
         return new WebServiceHost2(serviceType, true, baseAddresses);
      }
   }
}
```

Servise ait kod içeriğini ise senaryoya göre aşağıdaki gibi geliştirdim.

```
using System;
using System.Drawing;
using System.Drawing.Imaging;
using System.IO;
using System.Linq;
using System.Net;
using System.Runtime.Serialization;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Web;
using System.Xml.Linq;
using Microsoft.ServiceModel.Web;

[assembly: ContractNamespace("", ClrNamespace = "Streaming")]
```

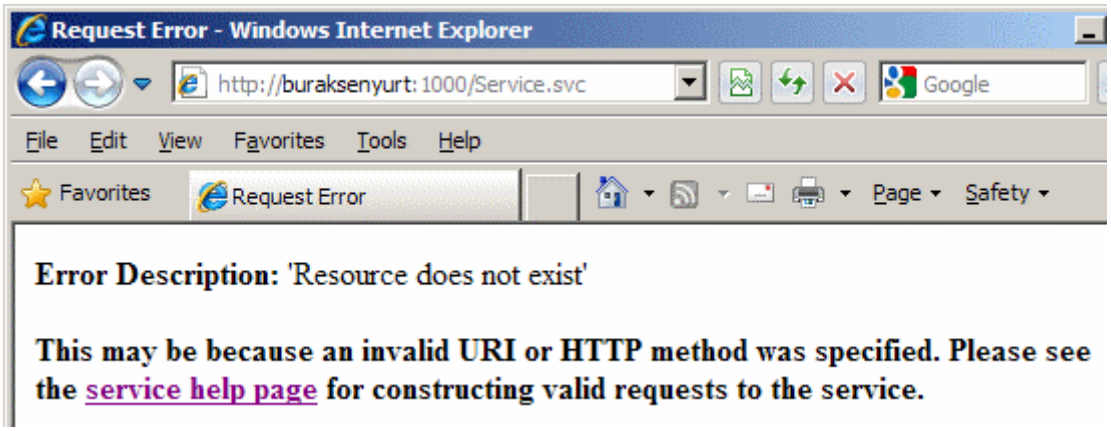
```
namespace Streaming
{
   [ServiceBehavior(IncludeExceptionDetailInFaults = true, InstanceContextMode =
InstanceContextMode.Single, ConcurrencyMode = ConcurrencyMode.Single)]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   [ServiceContract]
   public class Service
   {
      [WebHelp(Comment="Şansına göre bir duvar kağıdı döndürür")]
      [WebGet(UriTemplate="yourimage?keyword={keyword}")]
      [OperationContract]
      public Stream GetImage(string keyword)
      {
         // özellikle ilk talep sırasında keyword değeri gelmeyeceği için istemci tarafına
BadRequest tipinden Http hatası döndürülür
         // İstenirse varsayılan bir resimde ürettirilebilir
         if (String.IsNullOrEmpty(keyword))
            throw new WebProtocolException(HttpStatusCode.BadRequest);

         // gelen kelimeye eş düşen resim adresi Xml dökümanı içerisinden LINQ sorgusu
ile çekilir
         XDocument doc = new XDocument();
         doc =
XDocument.Load(System.Web.HttpContext.Current.Server.MapPath("~/Mapper.xml"));
         // TODO: Xml içeriğinde Keyword niteliğinde olmayan bir bilgi gelirse aşağıdaki
sorgu patlar. Buna tedbir alınması ve uygun Http hatasının döndürülmesi önerilir.
         string imagePath = (from img in
doc.Document.Elements("Mapper").Elements("Map")
                             where img.Attribute("Keyword").Value == keyword.ToLower()
                             select img.Attribute("Image").Value).First();

         // eğer kelimeye denk düşen resim yoksa NotFound tipinden Http hatası döndürülür
         if (String.IsNullOrEmpty(imagePath))
            throw new WebProtocolException(HttpStatusCode.NotFound);

         // Image tipi resim adresinden üretilir.
         Image image =
Image.FromFile(System.Web.HttpContext.Current.Server.MapPath(imagePath));
         // çıktının içerik tipi jpeg formatı olarak belirlenir.
         WebOperationContext.Current.OutgoingResponse.ContentType =
"image/jpeg";
         // AdapterStream yapıcısı kullanılarak stream istemci tarafına hazırlanıp gönderilir.
         return new AdapterStream(str => image.Save(str, ImageFormat.Jpeg));
      }
```

```
    }
}
```

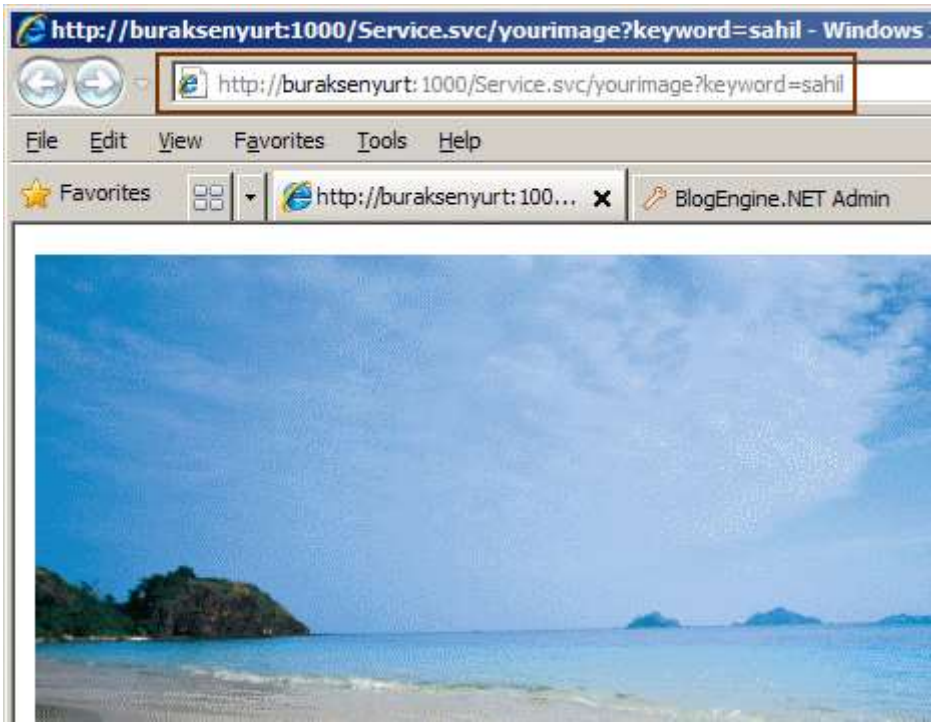
Kodun belkide en basit ama en önemli kısmı **AdapterStream** tipinin üretildiği satırdır. Bu satıra kadarki kısımda, gelen talebe göre **Xml** dökümanı içerisinden, denk düşen resim adresinin elde edilmesi ve buna göre **Image** nesnesinin örneklenmesi işlemleri yapılır. Sonrasında ise çıktının tipi**(image/jpeg)** olarak belirlenir ve son adıma gelinir. AdapterStream sınıfının yapıcı metodu parametre olarak **Action<Stream>** veya **Action<TextWriter>** tipinden bir **temsilci(delegate)** almaktadır. Bu temsilci, geriye değer döndürmeyen(void) ve parametre olarak bir Stream veya TextWriter referansı alan metodları işaret edecek şekilde tanımlanmıştır. Elimizde **=>(lambda)** operatörü gibi bir yardımcıda olduğundan nesne örneklenmesi sırasında temsilcinin tanımlanması, işaret ettiği metodun gövdesinin yazılması aynı ifade içerisinde mümkün olmaktadır.

Peki ya sonuçlar?

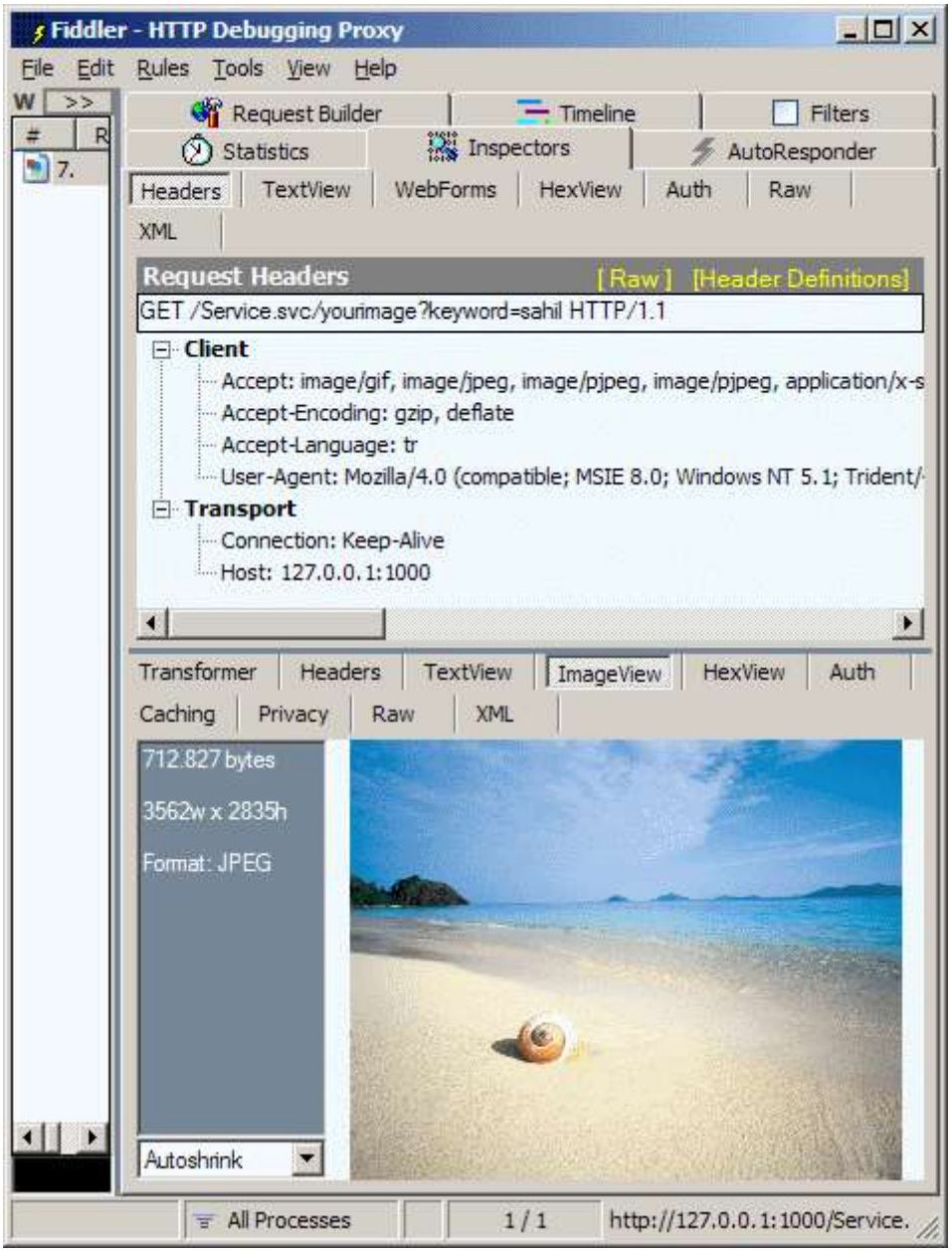
Servisi tarayıcıdan ilk seferde parametre kullanmadan talep ettiğimde aşağıdaki ekran görüntüsü ile karşılaştım.



Bu son derece doğaldı. Nitekim bu talep sonrasında servis operasyonuna herhangibir **keyword** değeri gelmemektedir. Ancak tarayıcıdan, **http://buraksenyurt:1000/Service.svc/yourimage?keyword=sahil** gibi bir talepte bulunduğumda aşağıdaki sonucu elde ettim. 😎 Sanırım şansıma deniz kıyısında şöyle güzel bir tatil çıktı.



Hemen arka planda çalışan **Fiddler** aracına baktığımdaysa, gelen talebe karşılık üretilen cevabın resim içerikli olarak üretildiğini gördüm.



Görüldüğü gibi **AdapterStream** kullanımı son derece kolay. özellikle REST bazlı WCF servislerini tüketen web uygulamalarında göz önüne alınabilir. Umarım size faydalı bir bilgi daha aktarabilmişimdir. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Streaming.rar (5,62 mb) (Dosya içerisinde bir TODO var. Bu kısmı gözden kaçırmayın)

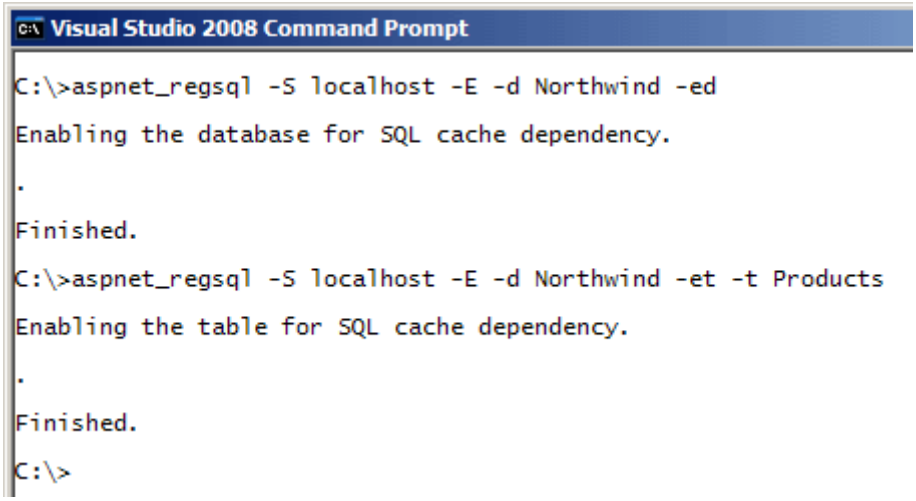
WCF Rest Servislerinde SqlCacheDependency Kullanımı (2009-04-30T00:40:00)

wcf,rest,

Merhaba Arkdaşlar,

WCF Rest Servislerinde önbellekleme(Caching) başlıklı yazımda, Rest Starter Kit ile birlikte gelen **WebCache** niteliğini(attribute) kullanarak ön bellekleme profillerinin nasıl hazırlanabileceğini ve kullanılabileceğini incelemeye çalışmıştım. önbellekleme ile ilgili olaraktan özellikle **SQL** tarafını ilgilendiren konulardan biriside **SqlCacheDependency** kullanımıdır. Daha öncedende bahsettiğim gibi, Asp.Net Caching mekanizmasının bir kaç farklı uygulama biçimi vardır. Bunlardan birisi olan **SqlCacheDependency** mekanizmasına göre, ön bellekte tutulacak olan veri içeriğinin bir tabloda meydana gelecek değişikliklere bağımlı hale getirilmesi sağlanabilir. Bu oldukça etkili bir model olmasına rağmen sadece SQL Server destekli olması çoğu çevrede pek kullanışlı olarak varsayılmamaktadır. Ama yinede SQL sunucuları üzerinde çalışılan sistemlerde bu mekanizmadan nasıl yararlanılabileceğini bilmek önemlidir. Tahmin edeceğiniz gibi yazımın bu seferki ana konusu, REST tabanlı bir WCF servisinde, **WebCache** niteliğini kullanırken **SqlCacheDependency** modelini nasıl ele alacağımızı öğrenmektir. Ama öncesinde Sql sunucusu üzerinde Cache Dependency için bazı ön hazırlıklar yapılmalıdır. öncelikli olarak veritabanının, sonrasında ise önbellek değişiklikleri için izlenecek olan tablonun**aspnet_regsql** aracı yardımıyla sisteme uygun hale getirilmeleri gerekmektedir.

Aslında sakin kafayla düşündüğümüzde, bir tabloda meydana gelecek değişiklerin izlenmesi için ilk olarak bir **trigger** yazacağımız ortadadır. Veri eklenmesi, güncellenmesi veya silinmesi sırasında devreye girecek olan bu trigger, değişiklikleri ayrı bir tabloya loglayarak başka sistemlerin(örneğin **Asp.Net Cache** Modüllerinin) kullanımına açabilir. Değişikliklerin trigger içerisinden aktarımı sırasında dahili sql ifadeleri kullanılabileceği gibi, bu işi üstlenen bir **Stored Procedure**' de söz konusu olabilir. aspnet_regsql aracının yaptığıda aslında tam olarak bu düşünce yapısına benzerdir. Lafı fazla uzatmıyayım **Visual Studio 2008 Command Prompt** üzerinden **aspnet_regsql** aracını aşağıdaki ekran görüntüsünde olduğu gibi çalıştırabiliriz.



İlk olarak Northwind veritabanının **Cache Dependency** için hazırlandığını görüyoruz. Burda **-S** ile sunucu adını(server name), **-E** ile işlemi windows' u açan kullanıcı yetkisi ile yapacağımızı belirtirken **-d** parametresinden sonra veritabanı(database) adını işaret ediyoruz. **-ed**' nin anlamı ise **enable database**

İkinci olarak Products tablosu için gerekli Cache Dependecy hazırlıklarını yapan komutu çalıştırmaktayız. Burada ise bir öncekinden farklı olark **-et** ile **enable table** diyor(bu sayede tablo için cache dependency' yi açacağımızı belirtiyoruz) ve **-t** ilede tablo adını(table name) ifade ediyoruz.

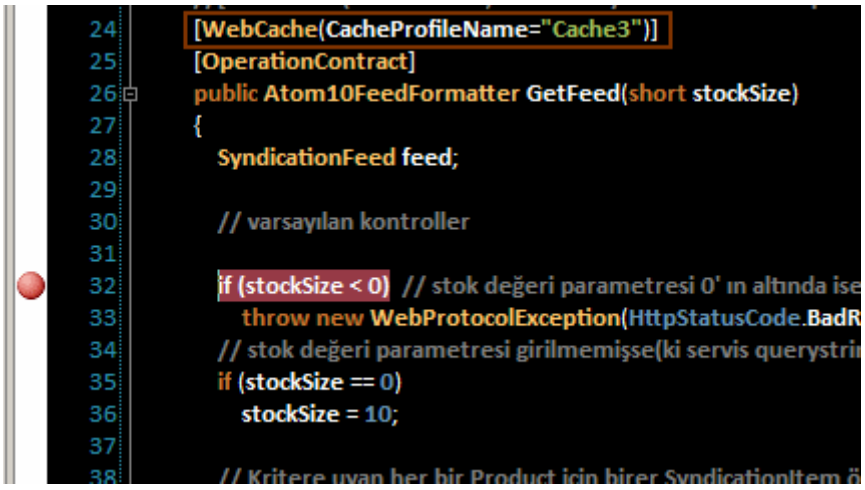
Bu işlemlerin ardından Northwind veritabanını biraz incelediğimizde Products tablosuna **Products_AspNet_SqlCacheNotification_Trigger** isimli bir **trigger** eklendiğini ve içerisinde**AspNet_SqlCacheUpdateChangeIdStoredProcedure** isimli bir sp çağırılarak **Products** parametresi gönderildiğini görebiliriz. Sp' nin yaptığı ise**AspNet_SqlCacheTablesForChangeNotification** tablosuna gerekli bilgileri eklemek. Tabiki veritabanı içerisinde izlenecek tablolar için gerekli trigger' ların oluşturulmasını sağlayan sp gibi farklı amaçlarla kullanılan nesnelerde oluştuğu gözlemlenebilir. Gelelim bizi ilgilendiren kısma. Daha önceki yazımızda geliştirdiğimiz Atom formatından içerik yayınlaması yapan servisin Caching mekanizması için Products tablosuna bağımlı olacak şekilde bir profile bildirimi yapmamız gerekiyor. Ki bu tek başına yeterli olmayacaktır. Aslında web.config dosyası içerisindeki değişiklikler üzerinden konuşursak çok daha doğru olacak. Bu amaçla web.config dosyası içeriğini aşağıdaki gibi düzenlediğimizi düşünelim.

web.config içeriği;

```
<?xml version="1.0"?>
<configuration>
  <connectionStrings>
    <add name="NorthwindConStr" connectionString="data
source=localhost; database=Northwind;integrated security=SSPI"/>
  </connectionStrings>
 <system.web>
    <caching>
      <outputCacheSettings>
        <outputCacheProfiles>
          <clear/>
          <add name="Cache1" duration="60" enabled="true" location ="Server"
varyByParam="size"/>
          <add name="Cache2" duration="20" enabled="true" location ="Client"
varyByParam="none"/>
          <add name="Cache3" sqlDependency="Northwind:Products"
varyByParam="size" enabled="true" location="Any"/>
        </outputCacheProfiles>
      </outputCacheSettings>
      <sqlCacheDependency enabled="true">
        <databases>
          <add name="Northwind" connectionStringName="NorthwindConStr"
pollTime="10000"/>
        </databases>
```

```
    </sqlCacheDependency>
  </caching>
 <compilation debug="true"/>
     </system.web>
<system.serviceModel>
 <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>
</system.serviceModel>
</configuration>
```

İlk olarak **Northwind** veritabanı için bir **connectionString** tanımlaması yapıldığını görmekteyiz. Bu tanımlama, ilerleyen kısımlarda yer alan **sqlCacheDependency** elementi içerisinde kullanılmaktadır. Yani **Cache Dependency** sisteminin hangi **SQL** bağlantısı üzerinden yürütüleceğini belirtmiş oluyoruz. Diğer taraftan **sqlCacheDependency** elementi altında **pollTime**isimli bir nitelik tanımlaması yapıldığını görmekteyiz. Buna göre **her 10 saniyede 1**, **connectionStringName** ile belirtilen bağlantıda yer alan ilgili tabloda bir değişiklik olup olmadığı kontrol edilecektir. **Polling** kelimesinin buradaki mantığıda zaten budur. (Git bak, duruma göre bir şeyler yap) Bu ayarlamalar aslında standart olarak **Asp.Net** tarafında geliştirilen web uygulamalarındakiler ile aynıdır. Ekstra olan, eklenen **Cache3** isimli önbellekleme profil içeriğidir. Bu içerikte **sqlDependency** özelliğine atanan değer ile **Northwind** isimli cache dependency için **Products**tablosuna bağlı bir önbellekleme sistemi kullanılacağı belirtilmektedir. Ayrıca önbellekleme yapısı **size** parametresine bağımlı hale getirilmiştir. Bundan sonra tek yapılması gereken WebCache niteliğini aşağıdaki gibi tanımlaktır.



Nasıl test edeceğim?

Tabiki sistemi nasıl test edeceğimizi konuşmamızda yarar var. Ben hemen kendi testimi nasıl yaptığımı anlatayım ki muhtemelen sizlerde benzer bir testi yapacaksınız. İlk etapta yukarıdaki şekilden de görüldüğü gibi **GetFeed** metodu içerisine bir **breakpoint** koydum. Uygulamayı **debug** modda çalıştırdığımda ilk etapta kod içerisine düşüldü ve sonuçlar ekrana geldi. Daha sonra örnek olarak stok miktarı 10 un altında olan ürünleri Atom formatında çekmek için **http://localhost:1000/ProductFeedService.svc/?size=10** talebini

gönderdim. Doğal olarak parametre bazlı bir önbellekleme profili kullandığımdan ve daha önceden bu talebi göndermediğimden kod içerisine tekrardan düştüm. Ancak sonraki taleplerde herhangibir şekilde koda düşmediğimi tespit ettim.*(Hatta bu sırada bloğun üst tarafındaki resimdekine benzer bir kahve almak için mutfağa gittim, kahvemi hazırladım. Derken bir arkadaşımı gördüm ve onunla bir süre sohbet ettim. Sonrada işten çıktım ve eve gittim. Aradan saatler geçtikten sonra aklıma geldi örnek ve tekrardan aynı talebi gönderdim* 😎 *)* Bu içeriğin ön bellekten getirildiğinin zaten bir kanıtıydı. Sonrasında ise stok miktarı 10 un altında olan satırlardan birisinde bir değişiklik yaptım. örnek olarak **Chef Anton's Gumbo Mix** isimli ürünün adını **Chef Anton's Gumbo Mix...** olarak değiştirdim. Bu durumda tarayıcıdan yine **http://localhost:1000/ProductFeedService.svc/?size=10** talebini gönderdiğimde tekrardan kod içerisine düştüğümü gözlemledim. Bir başka deyişle **Products** tablosunda değişiklik olduğu için **Polling** süresine göre caching mekanizması devreye girdi ve verinin tekrardan getirilmesine karar verdi. İşte bu kadar...

Böylece geldik bir yazımızın daha sonuna tekrardan görüşünceye dek hepinize mutlu günler dilerim.

CachingDependency.rar (109,00 kb)

nedirtv?com - Ankara Seminerleri (2009-04-28T16:57:00)

seminer,

nedirtv?com, 3. yıldönümü etkinliklerine devam ediyor. **3 Mayıs 2009 Pazar** günü **Ankara Bilkent üniversitesi**'nde değerli Ankaralılarla beraber olacağız. **INETA Türkiye** ve **Bilkent ACM Student Chapter** desteğiyle gerçekleştireceğimiz bu tüm günlük etkinliğe tüm Ankaralı yazılımcıları bekliyoruz.

Not: Seminerlere katılım ücretsizdir.

Seminer Programı:
09:30 **Açılış – nedirtv?com Tanıtımı**
09:45 **ASP.NET MVC (Uğur Umutluoğlu)**
11:30 **What is SharePoint? (Burak Batur)**

13:00 **Ara**

14:00 **WCF 4.0 & WF 4.0 (Burak Selim Şenyurt)**
15:30 **WPF ve Multitouch Development (Daron Yöndem)**
17:00 **Bitiş**

Zaman:
3 Mayıs 2009 Pazar

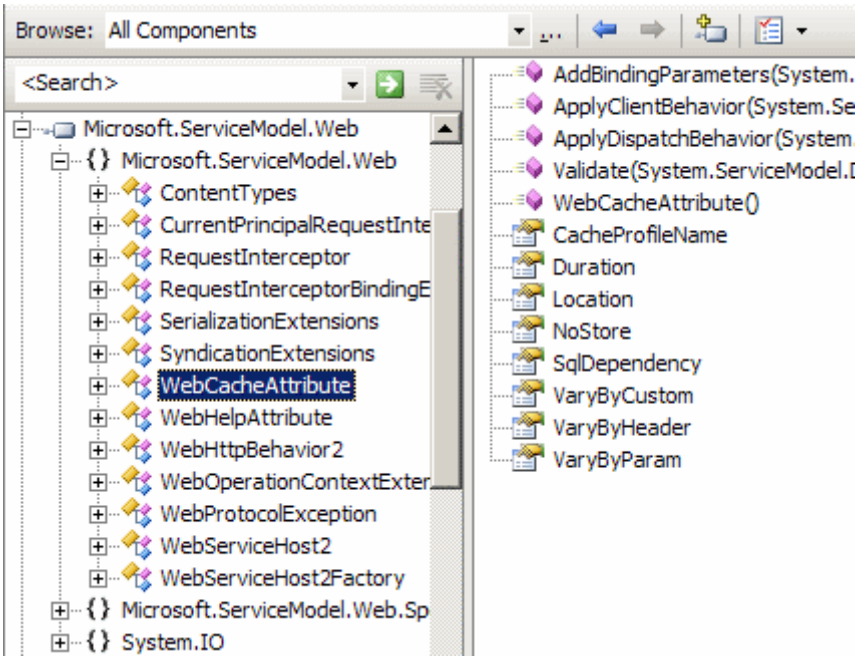
Yer:
Merkez Kampüs Rektörlük Binası Mithat çoruh Amfi Salonu, Bilkent üniversitesi - ANKARA

WCF Rest Servislerinde Önbellekleme(Caching) (2009-04-27T21:28:00)

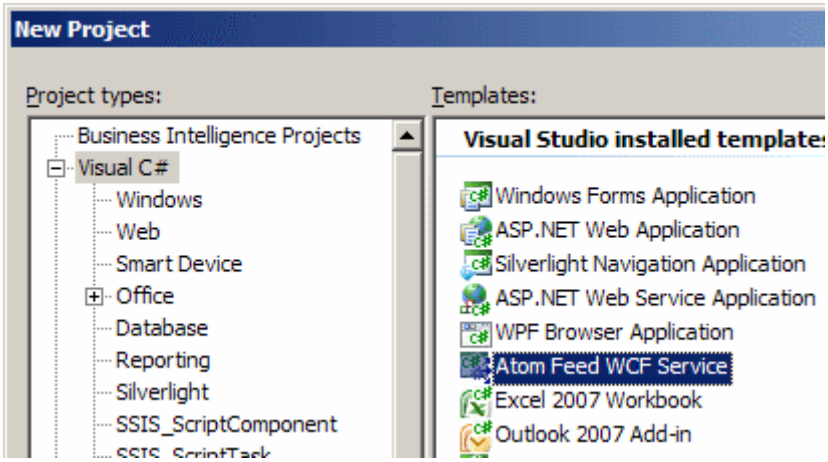
wcf,rest,

Merhaba Arkadaşlar,

REST(REpresentational State Transfer) modelini uygulayan **WCF** servislerinin geliştirilmesinde, **WCF Rest Starter Kit** ile birlikte gelen kolaylıklardan biriside, **önbellekleme(Caching)** işlemlerinin**dekleratif(Declarative)** olarak yapılabilmesidir. Burada dekleratiflikten kastımız, önbellekleme bildirimlerinin çalışma zamanına **nitelik(Attribute)** yoluyla bildirilmesidir. Web programlama modeline göre geliştirilen servislerin çeşidi ne olursa olsun, performans kriterleri söz konusu olduğunda önemli olan noktalardan biriside verilerin istemciye gönderilmeden önce gerekiyorsa belirli süreler boyunca veya bazı koşullar sağlanıncaya kadar sunucu ön belleğinde*(hatta bazen istemci tarafında tarayıcı uygulama için ayrılan özel bölgelerde)* tutulmasıdır. özellikle **Asp.Net** web uygulamalarını göz önüne alırsak, ön bellekleme ile ilişkili olarak kullanılan modüllerin pek çok farklı yapıyı desteklediğini görürüz. Söz gelimi, absolute caching(Kesin bir süre kadar önbellekte tutulması), sliding expiration caching(belirli süre içerisinde talep geldikçe önbellekte tutulma süresinin ileriye ötelenmesi), file dependency caching(önbellekteki verinin yenilenme koşulunun dosyadaki değişimlere bağlanması),sql dependency caching(ön bellekteki verinin yenilenme koşulunun bir tablodaki değişimlere bağlanması) vb... Hal böyle olunca, genellikle Asp.Net destekli sunucularda host edilen WCF servislerindende bu hazır **Caching** modüllerinden yararlanılması kaçınılmazdır. Startet Kit ise, **Microsoft.ServiceModel.Web** assembly' ı içerisinde sunduğu **WebCache** niteliği ile bu özelliğin REST bazlı WCF servislerinede uygulanabilmesini, olanaklı kılmaktadır.



Bu yazımızda söz konusu ön bellekleme sisteminin Atom formatında içerik yayınlaması**(Atom Feed Syndication)** yapan bir WCF Rest servisinde nasıl ele alınabileceğini incelemeye çalışacağız. İşe ilk olarak Atom Feed WCF Service şablonunda bir proje açarak başlayabiliriz.



Atom Feed WCF Service proje tipi, tahmin edeceğiniz üzere **WCF Rest Starter Kit** ile birlikte **Visual Studio 2008** ortamına yüklenen hazır şablonlardan birisidir. Şablon içeriği zaten hazır bir kod yapısına sahiptir ve gerekli **TODO** işaretlemeleri ile geliştiriciye neler yapması gerektiğini varsayılan ölçülerde bildirmektedir. Ancak bundan daha da önemlisi şablonun kullanım amacıdır. Bu şablon, **ATOM** formatında içerik yayınlaması yapan bir **WCF** servis operasyonunun **REST** bazlı olacak şekilde sunulabilmesini sağlamaktadır. Bu sebeten hazır olarak sunulan**GetFeed** isimli operasyonun dönüş tipi **Atom10FeedFormatter** nesne örneğindendir. Servis kodlarını tamamlamadan önce **web.config** dosyası içerisinde önbellekleme için gerekli profil özelliklerini aşağıdaki şekilde görüldüğü gibi belirleyebiliriz.

```
<?xml version="1.0"?>
<configuration>
 <system.web>
  <caching>
   <outputCacheSettings>
    <outputCacheProfiles>
     <clear/>
     <add name="Cache1" duration="60" enabled="true"
        location ="Server" varyByParam="none"/>
     <add name="Cache2" duration="20" enabled="true"
        location ="Client" varyByParam="none"/>
    </outputCacheProfiles>
   </outputCacheSettings>
  </caching>
  <compilation debug="true"/>
    </system.web>
 <system.serviceModel>
  <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>
 </system.serviceModel>
</configuration>
```

Buradaki ayarlamalara göre iki farklı önbellekleme profili tanımlanmaktadır. **Cache1** isimli profile göre, önbellekleme süresi **60** saniyedir ve sunucu taraflı bir önbellekleme yapılacağı **location**niteliğine atanan **Server** değeri ile belirtilmektedir. Diğer taraftan **Cache2** isimli profilde ise sadece süre ve lokasyon bilgileri farklıdır. Konfigurasyon dosyasında önem arz eden konulardan biriside**aspNetCompatibilityEnabled** niteliğine atanan **true** değeridir. Asp.Net ortamının önbellekleme modülünden yararlanılacağı için bu değerin true olması önemlidir.

Artık bizim yapmamız gereken tek şey, servis tarafında kullanacağımız **WebGet** niteliğine hangi **Cache** profilini kullanacağını bildirmektir. Buna göre başlangıçta **FeedService** adıyla oluşturulmuş olan ama örneğimizde **ProductFeedService** ismiyle tanımlanan servisimize ait kod içeriğini aşağıdaki gibi değiştirebiliriz.

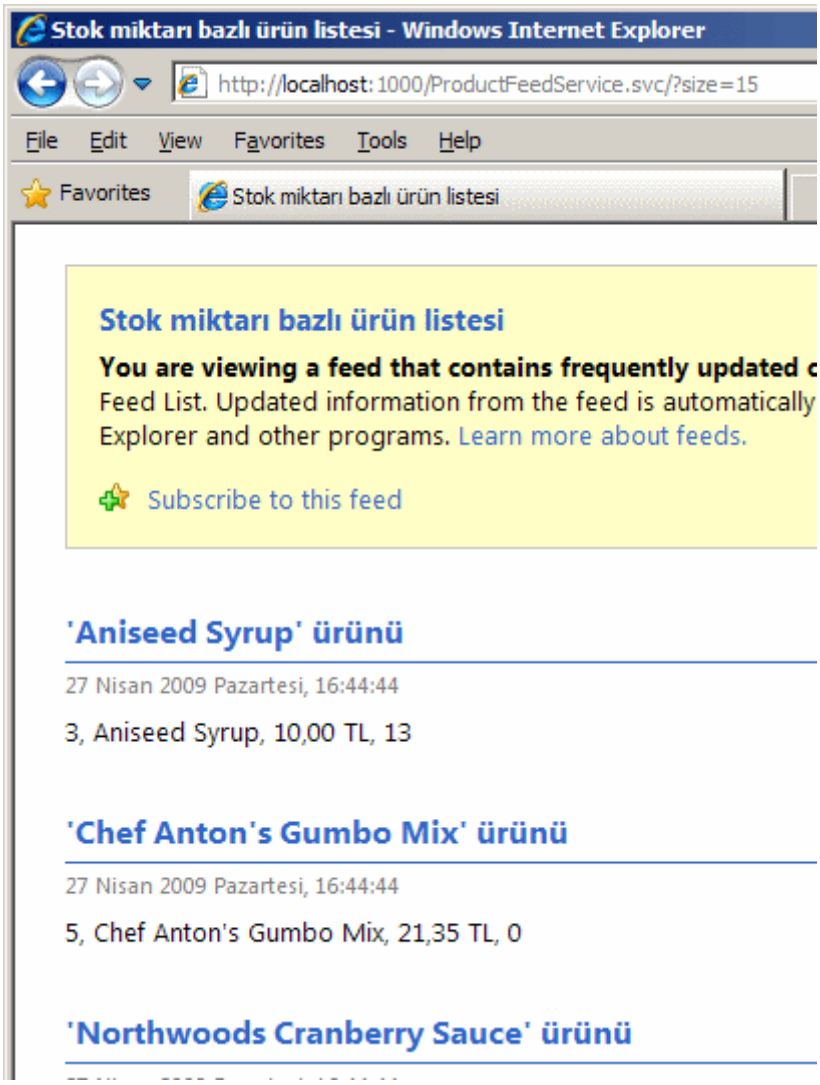
```
using System;
using System.Collections.Generic;
using System.Data.SqlClient;
using System.Globalization;
using System.Net;
using System.ServiceModel;
using System.ServiceModel.Activation;
using System.ServiceModel.Syndication;
using System.ServiceModel.Web;
using Microsoft.ServiceModel.Web;

namespace Caching
{
   [ServiceBehavior(IncludeExceptionDetailInFaults = true),
```

```
AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed), ServiceContract]
   public partial class ProductFeedService
   {
      [WebHelp(Comment = "Stok bilgisine göre tüm ürün bilgileri Atom formatında
getirilir.")]
      [WebGet(UriTemplate = "?size={stockSize}")]
      [WebCache(CacheProfileName="Cache1")] // web.config dosyasında, Cache1
isimli önbellekleme profilininin takip eden operasyon için kullanılacağı belirtilir.
      [OperationContract]
      public Atom10FeedFormatter GetFeed(short stockSize)
      {
         SyndicationFeed feed;

         // varsayılan kontroller

         if (stockSize < 0)  // stok değeri parametresi 0' ın altında ise
            throw new WebProtocolException(HttpStatusCode.BadRequest, "Stok miktarı
eksi değer olamaz.", null);
         // stok değeri parametresi girilmemişse(ki servis querystring kullanılmadan talep
edilirse bu çalışır)
         if (stockSize == 0)
            stockSize = 10;

         // Kritere uyan her bir Product için birer SyndicationItem örneği oluşturulup,
SyndicationItem tipinden generic List koleksiyonuna eklenir.
         List<SyndicationItem> items = new List<SyndicationItem>();
         foreach(Product product in GetProducts(stockSize))
         {
            items.Add(new SyndicationItem()
            {
               // Syndication içeriğinde olması gereken standart bazı özelliklerin değerleri set
edilir.

               Id = String.Format(CultureInfo.InvariantCulture,
"http://northwind.com/ProductID{0}", product.ProductId),
               Title = new TextSyndicationContent(String.Format("'{0}' ürünü",
product.Name)),
               LastUpdatedTime = DateTime.Now,
               Authors = {new SyndicationPerson() {Name = ""} // Yazar bilgisi
kullanılmadığı için boş bırakıldı
               },

               Content = new TextSyndicationContent(String.Format("{0}, {1}, {2},
{3}",product.ProductId.ToString(),product.Name,product.UnitPrice.ToString("C2"),produc
```

```
t.UnitsInStock.ToString())),
                PublishDate=DateTime.Now,
                 Summary=new TextSyndicationContent(String.Format("{0} isimli üründen
stokta {1} adet
bulunmaktadır",product.Name,product.UnitsInStock.ToString())),
            });
        }

        // Feed hazırlanır ve Items özelliğine, List<SyndicationItem> tipinden olan ve
yukarıda hazırlanan itemsi isimli koleksiyon atanır.
        feed = new SyndicationFeed()
        {
            Id = "http://northwind.com/ProductsWithStock",
            Title = new TextSyndicationContent("Stok miktarı bazlı ürün listesi"),
            Items = items
        };
        feed.AddSelfLink(WebOperationContext.Current.IncomingRequest.GetRequestUri
());
        // Operasyonun çıktı formatının Atom olacağı ContentType özelliğine atanan değer
ile belirlenir
        WebOperationContext.Current.OutgoingResponse.ContentType =
ContentTypes.Atom;
        // syndication içeriği operasyondan geriye döndürülür.
        return feed.GetAtom10Formatter();
    }

    // Yardımcı metod. UnitsInStock değeri stockSize ile gelen değerin altında olan
ürünleri Product tipinden bir koleksiyon olarak geriye döndürmektedir.
    private List<Product> GetProducts(short stockSize)
    {
        List<Product> products = new List<Product>();
        using (SqlConnection conn = new SqlConnection("data
source=.;database=Northwind;integrated security=SSPI"))
        {
            SqlCommand cmd = new SqlCommand("Select
ProductID,ProductName,UnitPrice,UnitsInStock From Products Where
UnitsInStock<@UnitsInStock", conn);
            cmd.Parameters.AddWithValue("@UnitsInStock", stockSize);
            conn.Open();
            SqlDataReader reader =
cmd.ExecuteReader(System.Data.CommandBehavior.CloseConnection);
            while (reader.Read())
            {
                products.Add(
                    new Product
```

```
                {
                    ProductId=Convert.ToInt32(reader["ProductID"]),
                     Name=reader["ProductName"].ToString(),
                      UnitPrice=Convert.ToDecimal(reader["UnitPrice"]),
                       UnitsInStock=Convert.ToInt16(reader["UnitsInStock"])
                }
                );
            }
            reader.Close();
        }
        return products;
    }
  }
}
```

GetFeed metodu Products tablosundan, stok miktarı parametre olarak gelen stockSize değerinden düşük olan ürün bilgilerini **Atom** formatından bir içerik olarak istemciye göndermektedir. Burada verinin çekilmesi sırasında yardımcı bir metod olarak **GetProducts** fonksiyonu kullanılmaktadır. Elbetteki yazımızın konusu itibariyle önem arz eden tek satır WebGet niteliğinin kullanıldığı yerdir.
Niteliğin **CacheProfileName** özelliğine atanan değer ile, **GetFeed** metodunun üreteceği çıktının hangi kriterlere göre nasıl ön bellekleneceği belirtilmektedir. Servis bu haliyle talep edildiğinde ve örneğin stok miktarı 15 birimin altında olan ürün listesi istendiğinde, aşağıdaki ekran görüntüsüne benzer bir çıktı ile karşılaşılabilir.



Tabi bu görüntü yazımız için yeterli değildir. Nitekim amacımız ön bellekleme sistemini test edebilmek. Burada izlenebilecek bir kaç yol var. En kolayı geliştiricinin **GetFeed** metodu başına **breakpoint** koyarak **debug** modunda ilerlemesidir. Bu durumda, servise gelen ilk talep ile birlikte **GetFeed** metodunun üreteceği içerik **60** saniyeliğine**(duration=60)** sunucunun ön belleğinde tutulamaya başlancaktır. Bu sırada tarayıcı üzerinden aynı sayfa tekrardan talep edilirse metodun içerisinde düşülmediği ve az önce üretilen çıktının geldiği gözlemlenecektir. Tabi burada çok dikkat edilmesi gereken bir nokta vardır. Servis bu haliyle talep edildiğinde, stockSize değeri 0 olduğundan if koşuluna göre 10 birimin altında olanlar getirilmektedir. Şayet bundan sonra tarayıcıdan **http://localhost:1000/Service.svc/?size=100** gibi bir talep girilirse yine stok miktarı 10 birimin altında olanlar getirilecektir. Neden? Tabiki ilk talep önbellekte tutulduğu için Elbette, 60 saniyelik ön bellekleme süresi beklendikten sonra bu talep gönderilirse, stok miktarı 100' ün altında olan ürünlerin elde edilebildiği ve doğal olarak **debug** moddayken ,GetFeed metodu içerisine düşülebildiği gözlemlenecektir. Bu tam olarak istenen bir şey olmayabilidir aslında. Belkide size parametresine göre ayrı ayrı ön bellekleme yapılabilmesi daha doğru olabilir ki buda son derece basittir. Tek yapılması gereken ön bellekleme profilini aşağıdaki gibi güncelleştirmektir.

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <caching>
      <outputCacheSettings>
        <outputCacheProfiles>
          <clear/>
          <add name="Cache1" duration="60" enabled="true"
              location ="Server" varyByParam="size"/>
          <add name="Cache2" duration="20" enabled="true"
              location ="Client" varyByParam="none"/>
        </outputCacheProfiles>
      </outputCacheSettings>
    </caching>
    <compilation debug="true"/>
      </system.web>
  <system.serviceModel>
    <serviceHostingEnvironment aspNetCompatibilityEnabled="true"/>
  </system.serviceModel>
</configuration>
```

Görüldüğü gibi tek yaptığımız **varyByParam** niteliğine size değerini vermektir. önbellekleme ile ilişkili detayları **web.config** dosyası içerisinde taşımanın en büyük avantajlarından biriside, koda girmeden değiştirilebilme olanağı sağlamasıdır. örneğin ürünün **IIS** altına atılmasından sonra, **size** parametresine göre önbellekleme yapılacağına karar verildiyse eğer, kodu tekrardan açmadan **web.config** dosyasına müdahale edilerek bu güncelleme gerçekleştirilebilir. Diğer yandan istenirse web.config dosyası kullanılmadan, WebGet niteliğinin özellikleri içerisindede önbellekleme bilgileri tanımlanabilir.

```
[WebHelp(Comment = "Stok bilgisine göre tüm ürün bilgileri Atom formatında getirilir.")]
[WebGet(UriTemplate = "?size={stockSize}")]
[WebCache(Duration=60, Location=System.Web.UI.OutputCacheLocation.Any,
VaryByParam="size")]
[OperationContract]
public Atom10FeedFormatter GetFeed(short stockSize)
{
```

şeklinde...

Evettt..Böylece geldik bir blog yazımızın daha sonuna. Bu kısa yazımızda **WCF REST Starter Kit'** i kullanarak bir **Atom Feed WCF Service'** in nasıl geliştirilebileceğini gördük. Ama dahada önemlisi, **WebGet** niteliği yardımıyla ön bellekleme işlemlerinin nasıl yapılabileceğine bakmaya çalıştık. Görüşmek dileğiyle...

örnek Dosya; Caching.rar (107,66 kb)

Rest Tabanlı WCF Servislerinde İstemci Tarafını Asenkron Geliştirmek (2009-04-24T18:26:00)

wcf,rest,async,

Merhaba Arkadaşlar,

Bir önceki yazımızda REST bazlı WCF servisleri için, **WCF Rest Stater Kit** yardımıyla istemci uygulamaların nasıl geliştirilebileceğini incelemeye çalışmıştık. İstemci açısından önemli olan konulardan biriside, uzun sürebilecek **request/response** operasyonları sırasında uygulamasını kullanmaya devam edebiliyor olmasıdır. Tahmin edeceğiniz üzere istemci tarafında bir request' in asenkron olarak gönderilip, işlenmesi konusunu değerlendiriyor olacağız. Aslında asenkron erişimden kastımız, istemcinin talebi gönderdikten sonra cevabın anında gelmesini beklemeden çalışmasına devam edebilmesidir. Servis tabanlı uygulamalar söz konusu olduğunda, asenkron işlemleri iki lokasyonda tasarlayabiliriz.

1. Asenkron işlemler **servis** tarafında uygulanır. Geliştiricinin asenkron modeli kendisinin uygulamasını gerektirebilir.
2. Asenkron işlemler **istemci** tarafında uygulanır. Hazır olan **temsilci(delegate)** tabanlı **(Polling, WaitHandle, Callback)** veya **olay tabanlı(Event Based)** modeller kullanılır.

Benim bu yazıda ele alacağım istemci tarafındaki asenkron işlemlerdir. Bir önceki yazımızda geliştirdiğimiz Windows uygulamasında bu amaçla yeni düzenlemeler yapılacaktır. Hazırlıklı olmamız gereken konuların başında, **delegate** kavramı ve asenkron **Callback** modeli gelmektedir. Ama sonrasıda buna event based asenkron modeli ve lambda operatörünüde katıyor olacağız. Arada ise bize bonus bir konu çıkacak. Tedbir almassak kaçınılmaz olan **Illegal Cross Thread Operations**

İlk olarak Form arkası kodlarımızı aşağıdaki gibi geliştirdiğimizi düşünelim.

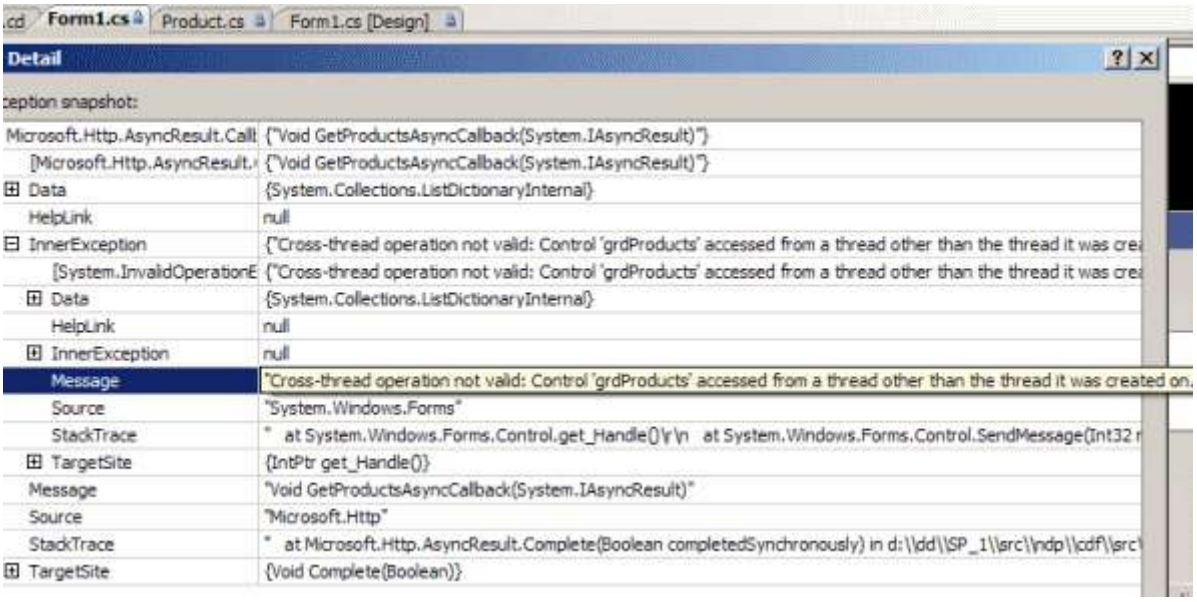
```
using System;
using System.Linq;
using System.Net;
using System.Windows.Forms;
using System.Xml.Serialization;
using Microsoft.Http;
using NorthwindV2;

namespace NorthwindClient
{
   public partial class Form1 : Form
   {
      // İstemci talepleri için kullanacağımız sınıf HttpClient
```

```
        HttpClient client = null;
        // Servis adresi (sondaki / işaretini unutmadım bu sefer)
        string serviceUri="http://localhost:1000/Service.svc/";

        public Form1()
        {
            InitializeComponent();
            // HttpClient nesnemizi örnekliyoruz
            client = new HttpClient();
        }

        private void btnGetProductsAsync_Click(object sender, EventArgs e)
        {
            // İletişim için 10 saniyelik timeout süresini belirliyoruz
            client.TransportSettings.ConnectionTimeout = TimeSpan.FromSeconds(10);
            // BeginSend metodu yardımıyla asenkron çağrıyı başlatıyoruz. İlk parametreye
göre http://localhost:1000/Service.svc/ adresine GET metodu ile talepte bulunuyoruz.
            // İkinci parametre bu işlem bittiğinde devreye girecek asenkron metodu işaret eden
meşhur AsyncCallback temsilcimiz.
            // HttpClient nesnemiz sınıf seviyesinde tanımlandığı için 3ncü parametreyi
göndermemize gerek yok.
            client.BeginSend(new HttpRequestMessage("GET", serviceUri), new
AsyncCallback(GetProductsAsyncCallback), null);
        }

        // BeginSend metodu ile başlatılan asenkron işlemler tamamlandığında devreye
girecek olan callback metodudur.
        private void GetProductsAsyncCallback(IAsyncResult iar)
        {
            // EndSend metodu ile tamamlanan operasyon sonucu cevap alınır.
            using (HttpResponseMessage response = client.EndSend(iar))
            {
                // eğer HTTP 200 kodu döndüyse exception fırlatılmadan devam edilebilir
                response.EnsureStatusIs(HttpStatusCode.OK);
                // ItemInfoList tipi ReadAsXmlSerializable metodu ile çekilir
                ItemInfoList products =
response.Content.ReadAsXmlSerializable<ItemInfoList>();

                grdProducts.DataSource = (from p in products.ItemInfo
                                          select p.Item).ToList();
            }
        }
    }
}
```

Buradaki kod parçasında, standart **Asynchronous Callback** modelinin bir uyarlaması yer almaktadır. Asenkron desenler temsilcilerin kullanıldığı senaryolarda ele alınabilirler. Genel olarak 3 farklı asenkron tekniği vardır.

- **Polling** modeline göre asenkron olarak başlatılan işlemin bitip bitmediği sürekli olarak kontrol edilir.
- **WaitHandle** modeli aslında kendi içerisinde **WaitOne, WaitAll, WaitAny** gibi 3 farklı tekniğe ayrılmaktadır. Ancak tekniklerin özünde, asenkron başlatılan bir işin, belirli bir noktadan sonra dönüş değerlerine ihtiyaç duyuluyorsa, thread' in ilgili noktada duraksatılması vardır. öyleki, bazı asenkron işlemler sonucunda gelen veriler daha çekilemeden, program içerisinde onların kullanılacağı yerlere geçişler yapılabilir. Bunu kullanıcıda yapabilir, programın kod akışıda buna müsait olabilir. Dolayısıyla ortada dönen veriler yoksa istenmeyen sonuçlar alınabilir.
- **Callback** modeli ise en sık kullanılan tekniklerden birisidir. Bu modele göre asenkron başlatılan işleyiş tamamlandığında, **.Net Framework'** ün **Built-In Delegate** tiplerinden olan**AsyncCallback'** in işaret ettiği(geriye dönüş değeri olmayan yani **void** ve **IAsyncResult** arayüzü tipinden parametre alan) metod çalıştırılır.

Ancak bir asenkron model WinForms yada WPF gibi görsel bir arabirimde uygulanıldığında çok dikkatli olunmalıdır. Yukarıda yazdığımız program kodunu denediğimizde bu acı gerçekle aşağıdaki ekran görüntüsünde olduğu gibi karşılaşmamız kaçınılmazdır.



Aslında sebep son derece basittir. Normal şartlarda **Form** üzerindenki tüm bileşenlerin sahibi olan bir ana iş parçamız vardır**(Main Thread).** Biliyoruzki bir .Net uygulaması belleğe açıldığında mutlaka bir ana Thread' e sahiptir.(Hatta **Process** içerisinde çalışan **Module(Modules)** ve bunlarında içerisindede en az bir ana thread olmak üzere birden fazla thread' de olabilir) Diğer taraftan yazdığımız asenkron modelde açılan farklı bir **thread'** de bu kontrollerden birisine(**DataGridView** bileşenimiz) erişmek

istemektedir. Bu durumda ana thread buna kızar(çünkü bencildir ve kontrollerini kimse ile paylaşmak istemez) ve çalışma zamanına yukarıda gördüğümüz istisna fırlatılır. Bunu çözmek için kolaya kaçabiliriz. Ancak en etkili çözümlerden birisi Method Invoker kullanmaktır. Bu nedenle yukarıdaki kod parçasını aşağıdaki gibi değiştirmemiz gerekmektedir.

```
using System;
using System.Linq;
using System.Net;
using System.Windows.Forms;
using System.Xml.Serialization;
using Microsoft.Http;
using NorthwindV2;

namespace NorthwindClient
{
   public partial class Form1 : Form
   {
      // İstemci talepleri için kullanacağımız sınıf HttpClient
      HttpClient client = null;
      // Servis adresi (sondaki / işaretini unutmadım bu sefer)
      string serviceUri="http://localhost:1000/Service.svc/";

      public Form1()
      {
         InitializeComponent();
         // HttpClient nesnemizi örnekliyoruz
         client = new HttpClient();
      }

      private void btnGetProductsAsync_Click(object sender, EventArgs e)
      {
         // İletişim için 10 saniyelik timeout süresini belirliyoruz
         client.TransportSettings.ConnectionTimeout = TimeSpan.FromSeconds(10);
         // BeginSend metodu yardımıyla asenkron çağrıyı başlatıyoruz. İlk parametreye
göre http://localhost:1000/Service.svc/ adresine GET metodu ile talepte bulunuyoruz.
         // İkinci parametre bu işlem bittiğinde devreye girecek asenkron metodu işaret eden
meşhur AsyncCallback temsilcimiz.
         // HttpClient nesnemiz sınıf seviyesinde tanımlandığı için 3ncü parametreyi
göndermemize gerek yok.
         client.BeginSend(new HttpRequestMessage("GET", serviceUri), new
AsyncCallback(GetProductsAsyncCallback), null);
      }
```

```
        // BeginSend metodu ile başlatılan asenkron işlemler tamamlandığında devreye
girecek olan callback metodudur.
        private void GetProductsAsyncCallback(IAsyncResult iar)
        {
            // EndSend metodu ile tamamlanan operasyon sonucu cevap alınır.
            using (HttpResponseMessage response = client.EndSend(iar))
            {
                // eğer HTTP 200 kodu döndüyse exception fırlatılmadan devam edilebilir
                response.EnsureStatusIs(HttpStatusCode.OK);
                // ItemInfoList tipi ReadAsXmlSerializable metodu ile çekilir
                ItemInfoList products =
response.Content.ReadAsXmlSerializable<ItemInfoList>();

                //grdProducts.DataSource = (from p in products.ItemInfo
                //                          select p.Item).ToList();
                LoadGrid(products);
            }
        }

        #region Method Invoker ile Illegal Cross Thread' in önüne geçmek

        private delegate void LoadGridHandler(ItemInfoList list);
        private void LoadGrid(ItemInfoList list)
        {
            if (grdProducts.InvokeRequired)
                grdProducts.Invoke(new LoadGridHandler(LoadGrid), list);
            else
                grdProducts.DataSource = (from p in list.ItemInfo
                                          select p.Item).ToList();
        }

        #endregion
    }
}
```

Bu durumda program kodumuz sorunsuz bir şekilde çalışacak ve içeriğin asenkron olarak çekilmesi sağlanabilecektir. Tabi şunuda düşünmek gerekir. **.Net Framework 2.0** relase olduğunda**Xml Web Servislerinin** kullanılması ile ilişkili istemci tarafına gelen yeniliklerden birisi, asenkron modeli olay bazlı uygulayabiliyor olmamızdı (Bu konuyu yine çok çok uzun zaman önce .Net 2.0 - Web Servislerinde Yeni Asenkron Erişim Modeli isimli görsel derste ele almıştım). Yani temsilciler ve method invoker' lar ile uğraşmak yerine basit olay(event) yüklemeleri ile işlemler çözülebilmektedir. Modelin özünde **Completed** kelimesi ile biten bir **olay(Event)**, **Async** kelimesi ile biten ve asenkron işlemi başlatmamızı sağlayan, **AsyncCancel** kelimesi ile biten ve asenkron başlatılan işlemin iptal edilmesinde kullanılan birer metod bulunmaktadır.

Aynı modeli **Rest bazlı WCF** servis istemcilerinde de uygulayabiliriz. Nitekim **HttpClient** sınıfının bu amaçla tasarlanmış **SendCompleted** isimli olayı, **SendAsync** ve **SendAsyncCancel** isimli metodları bulunmaktadır. Bu noktada kodu ilk etapta

```
private void btnGetProductsEvent_Click(object sender, EventArgs e)
{
    client.SendCompleted += new
EventHandler<SendCompletedEventArgs>(client_SendCompleted);
}

void client_SendCompleted(object sender, SendCompletedEventArgs e)
{
    throw new NotImplementedException();
}
```

şeklinde tasarlamaya başlayabiliriz. Oysaki C# 3.0 ile birlikte gelen **lambda operatörünü(=>)** kullanarak, olayın yüklenmesi, olay sonucu çalıştırılacak olan metod bloğunuz yazılması ve içerisine gerekli parametrelerin aktarılması işini aşağıdaki kod parçasında olduğu gibide gerçekleştirebiliriz.

```
private void btnGetProductsEvent_Click(object sender, EventArgs e)
{
   client.TransportSettings.ConnectionTimeout = TimeSpan.FromSeconds(10);
   client.SendCompleted+=(sndr,arg)=>{
      // İşlem iptal edilmediyse
      if (arg.Cancelled)
         MessageBox.Show("İşlem iptal edildi");
      // İşlem sonucunda bir istisna oluşmamışsa
      else if (arg.Error != null)
         MessageBox.Show(arg.Error.Message);
      else
      {
         ItemInfoList
products=arg.Response.Content.ReadAsXmlSerializable<ItemInfoList>();
         grdProducts.DataSource = (from p in products.ItemInfo
                                   select p.Item).ToList();
      }
   };
   client.SendAsync(new HttpRequestMessage("GET", serviceUri));
}
```

Kullanılan bu son teknikte herhangibir şekilde **Illegal Cross Thread Operation** sorunsalınında yaşanmadığı gözlemlenebilir. Bu olay bazlı asenkron mimarinin

bir avantajıdır. Yani **Method Invoker** kullanmamıza gerek kalmadan asenkron olarak üretilen sonuçlar **DataGridView** kontrolü içerisine alınabilmektedir.

Böylece geldik bir yazımızın daha sonuna. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

örneğin son hali : NorthwindV2.rar (592,24 kb)

Rest Tabanlı WCF Servisleri için İstemci Yazmak (2009-04-23T21:12:00)

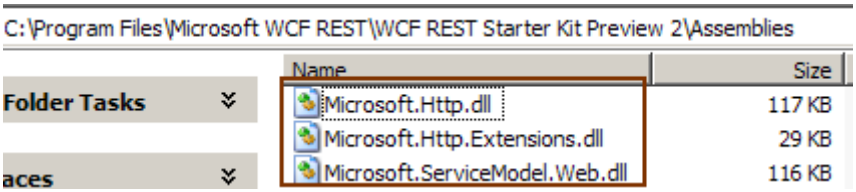
wcf,rest,

Merhaba Arkadaşlar,

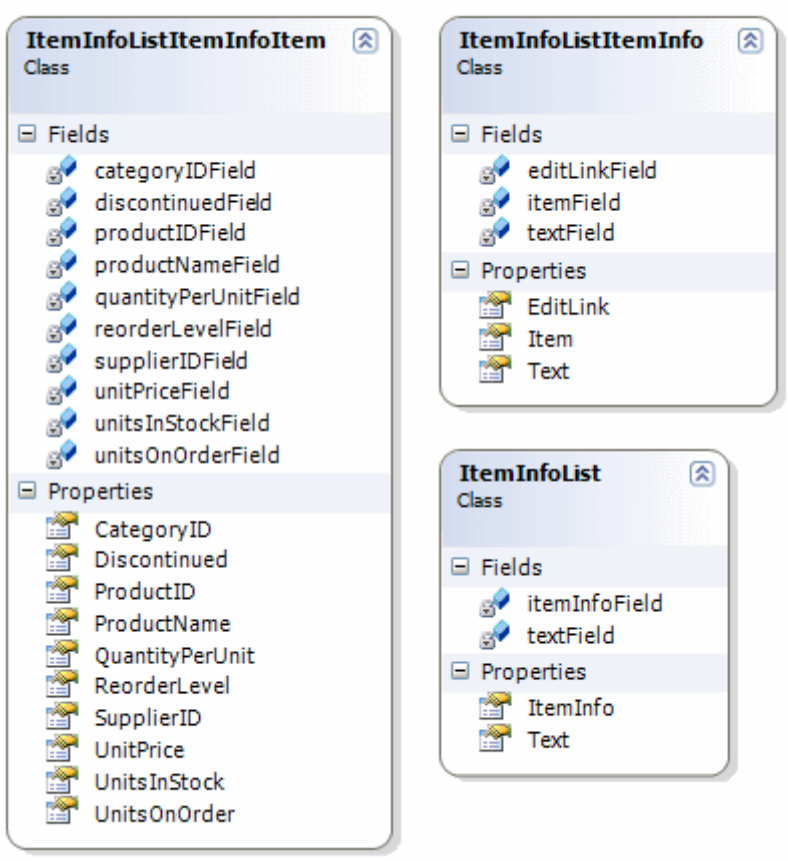
Bir önceki blog yazımızda, koleksiyon bazlı **WCF** servislerinin **REST** modeline göre geliştirilmesini incelemeye çalışmış ve **REST Starter Kit**' in sağladığı kolaylıklara değinmiştik. Belkide yazının en zor kısımlarından biriside **CUD(CreateUpdateDelete)** işlemlerinin test edilmesiydi. Nitekim burada istemciden gönderilecek **Request** paketlerinin **HTTP**protokolünün uygun olan **POST, PUT, DELETE** metodlarından birisine göre hazırlanıp iletilmesi gerekmekteydi. Bu nedenle, **Fiddler** aracını kullanarak talepleri oluşturmuş ve testleri gerçekleştirmiştik. Aslında, sadece veri çekilmesi işleminde**(HTTP GET)** işimiz nispeten çok daha kolay olmaktadır. Basit bir tarayıcı uygulama bu iş için yeterlidir. Peki ya istemci, bir geliştirici tarafından yazılacak ve söz konusu REST bazlı koleksiyon servisini tüketecek bir uygulama olacaksa... 🤒

Bir geliştirici olarak olayı son derece basit bir şekilde düşünebiliriz. Nitekim **Fiddler** veya **Internet Explorer** gibi bir tarayıcının yaptıkları, içeride gereki **HTTP** paketinin hazırlanması ve karşı tarafa gönderilmesidir. çok çok eskiden galaksinin uzak bir diyarında(a long time ago in a galaxy far far away), Web servislerinin kullanılmasında **SOAP** paketlerinin manuel olarak nasıl hazırlanıp gönderilebileceğini ve servisten dönen cevapların nasıl ele alınabileceğini Web Servislerinde SOAP ile Request ve Response isimli görsel videoda incelemiştim(2006 yılında). O örnektede **SOAP zarflarının(SOAP Envelope) .Net** tipleri yardımıyla manuel olarak hazırlanıp gönderilmesi söz konusuydu. E tabi aradan yıllar geçer, WCF gibi çok güçlü bir**SOA(Service Oriented Architecture)** çözümü ortaya çıkar. Bildiğiniz gibi **.Net Framework 3.5** ile birlikte WCF' in kazandığı web programlama modeli sayesinde de, REST bazlı geliştirmelerin yapılabilmesi olanaklı hale gelmiştir. Ayrıca, **WCF Rest Starter Kit** ile işlemlerin biraz daha kolaylaştırılması mümkündür. İstemci tarafını geliştirirkende bu kit ile birlikte gelen yardımcı tipler ve belkide en önemlisi **genişletme metodlarından(Extension Methods)** yararlanılmaktadır. öyleyse yeni bir maceraya yelken açalım ve bir önceki yazımızda geliştirdiğimiz **WCF Rest Collection Service** projesi içerisinde yayımlanan hizmeti, basit bir WinForms uygulamasından tüketmeye çalışalım.

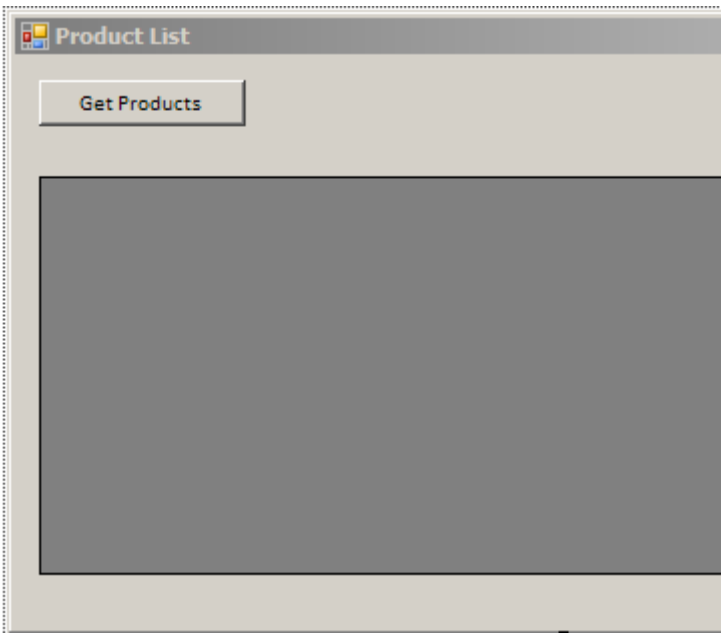
Bu amaçla ilk olarak bir Windows projesi açarak yola koyuluyoruz. Sonrasında ise projemiz için gerekli olan bazı referansları eklememiz gerekiyor.



Burada görülen **Microsoft.Http.dll**, **Microsoft.Http.Extension.dll** ve **Microsoft.ServiceModel.Web.dll assmebly'** lar, **WCF Rest Stater Kit** ile birlikte gelen ve istemci tarafından REST bazlı WCF servislerinin tüketilmesinde kullanılan pek çok yardımcı tipi ve üyeyi içermektedir. Bu muhakkakki geliştrici olarak bizleri sevindiren bir gelişmedir. Sonraki adımda ise servis tarafından yayımlanan Product tipi ve buna ait örnekleri içeren koleksiyon bazlı listenin istemci tarafında bir şekilde temsil edilmesi gerekmektedir. Nitekim, istemciden gidecek talep sonrası(örneğin tüm ürün listesinin istenmesi) servisten gelecek cevap içeriği XML tabanlı olacaktır ve kod tarafında kolay bir şekilde yönetilebilmesi arzu edilir. İşte bu noktada da WCF Rest Starter Kit kurulumu sonrası**Visual Studio 2008'** e eklenen **Paste XML as Types** menü seçeneği dikkati çekmektedir. Aslında yapacağımız tek şey, istemci tarafında boş bir **namespace** oluşturmak(adını NorthwindV2 olarak verebiliriz), servisi bir kere kullanıp tüm ürün listesini istedikten sonra üretilen **XML** içeriğini tamamıyla kopyalamak ve **Paste XML as Types** menü seçeneği ile yapıştırmaktır. Bunun sonucunda istemci uygulama tarafında aşağıdaki sınıf diagramında görülen tipler otomatik olarak oluşturulacaktır.



Görüldüğü gibi istemci tarafında, servisten gelen koleksiyon bazlı içeriği **yönetimli kod(Managed Code)** tarafında temsil edebilmemiz için gerekli tüm tipler oluşturulmuştur. Ama Oz büyücüsünün yardımları sadece buraya kadardır. Artık developer olarak bizim direksiyonun başına geçmemiz gerekiyor. Peki ama neden? öncelikli olarak amacımızı belirtelim. İlk etapta tüm ürün listesini istemci uygulama tarafına çekebilmek istiyoruz. Bu amaçla aşağıdaki ekran görütüsüne sahip basit bir **WinForm** ' umuz olduğunu düşünelim.

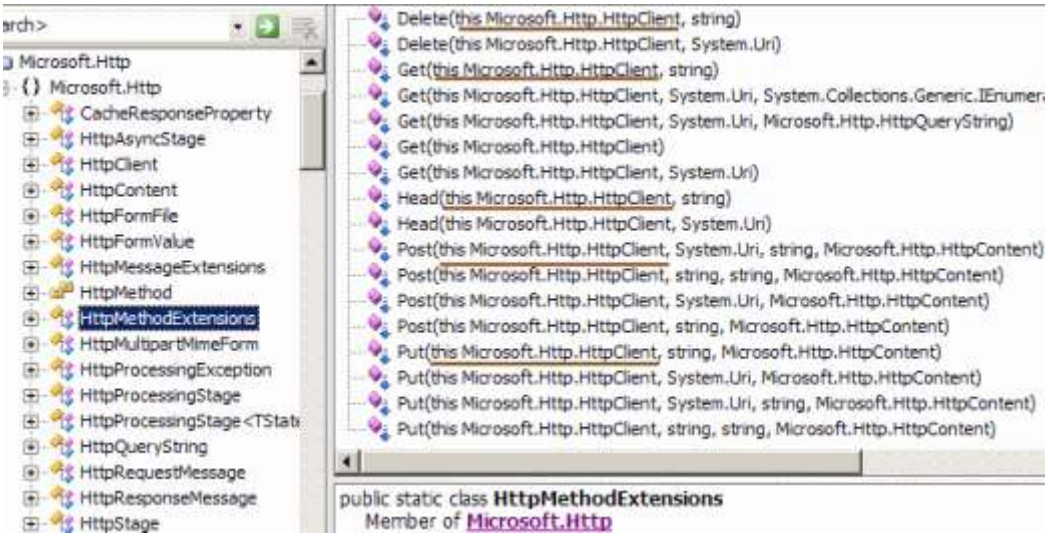


Form üzerindeki **Button** kullanıldığında, servis tarafından talep edilen ürün listesini **DataGridView** kontrolünde göstermeyi ilk hedefimiz olarak seçebiliriz. Burada önemli olan noktalardan birisi**talebin(Request)** oluşturulması, servise **GET** metoduna göre gönderilmesi ve gelen **cevap(Response)** içerisinde yer alan **XML** içeriğinin yönetimli kod tarafında **ItemInfoListItemInfoItem** tipine kadar indirgenebilmesidir. öyleyse kod içeriğini aşağıdaki gibi oluşturalım.

```
using System;
using System.Linq;
using System.Net;
using System.Windows.Forms;
using System.Xml.Serialization;
using Microsoft.Http;
using NorthwindV2;

namespace NorthwindClient
{
   public partial class Form1 : Form
   {
      // İstemci talepleri için kullanacağımız sınıf HttpClient
      HttpClient client = null;
      // Servis adresi (sondaki / işaretini unutmadım bu sefer)
      string serviceUri="http://localhost:1000/Service.svc/";

      public Form1()
      {
         InitializeComponent();
         // HttpClient nesnemizi örnekliyoruz
         client = new HttpClient();
      }

      private void btnGetProducts_Click(object sender, EventArgs e)
      {
         // IDisposable interface' ini implemente eden HttpResponseMessage nesne örneğinin içeriğinin HttpClient tipinin Get metodu yardımıyla elde ediyoruz. Get metodu parametre olarak servis adresini almakta.
         using (HttpResponseMessage response = client.Get(serviceUri))
         {
            // Eğer servis tarafından Http 200(yani OK) cevabı gelirse işlemler devam edecektir. Aksi durumda istisna mesajı fırlatılacaktır.
            response.EnsureStatusIs(HttpStatusCode.OK);

            // cevap olarak gelen paket içeriği XML formatlıdır. Bu içeriğin ItemInfoList tipine cast edilmesi ve yönetimli kod ile ele alınabilmesi için ReadAsXmlSerializable<T> generic metodu kullanılır.
```

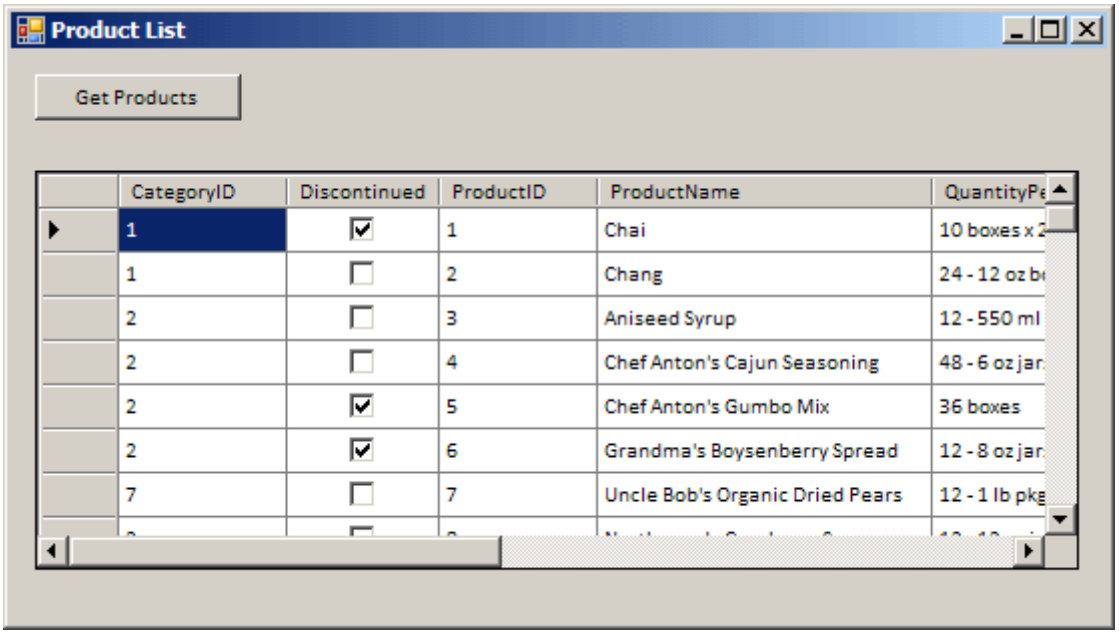
```
            ItemInfoList
products=response.Content.ReadAsXmlSerializable<ItemInfoList>();

            // ilk olarak tüm ürünleri listeleriz. ToList metoduna generic parametre olarak,
servis tarafındaki Product tipinin istemcide otomatik üretilen karşılığı olan
ItemInfoListItemInfoItem sınıfı verilmiştir, buna dikkat edelim.
            grdProducts.DataSource = (from p in products.ItemInfo
                                      select p.Item).ToList<ItemInfoListItemInfoItem>();
        }
    }
  }
}
```

Burada dikkat edilmesi gereken noktalardan biriside **HttpClient** tipine ait nesne örneği yardımıyla **Get** metodunun kullanılışıdır. Parametrenin servis adresini gösteriyor olması aslında, **HTTP Get** metoduna göre **http://localhost:1000/Service.svc/** adresine bir talep gönderiliyor olması anlamına gelmektedir. Eğer istemci tarafından **Post, Put** veya **Delete** talepleri gönderilmek isteniyorsa yine**HttpClient** nesne örneği üzerinden aynı isimli metodlar kullanılabilir. Bu metodlar aslında aşağıdaki şekildende görüldüğü üzere,



HttpMethodExtension sınıfı içerisinde yazılmış genişletme metodlarıdır. (LINQ mimarisinde çok önemli bir yere sahip olan genişletme metodları ile ilişkili daha detaylı bilgileri C# 3.0 : Derinlemesine Extension Methods isimli makalemden öğrenebilirsiniz).

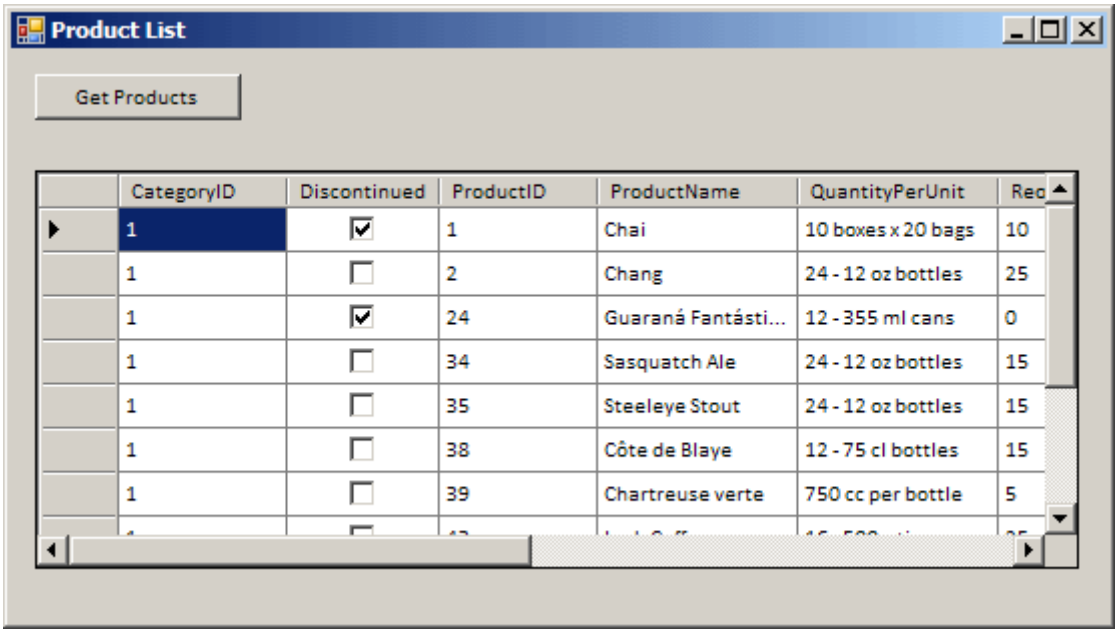
Form uygulamamızı ilk haliyle çalıştırıp testlerimize başlayabilir. Tabi burada küçük bir ayrıntıyı gözden kaçırmamak gerekiyor. Servis örneğinin çalışıyor olmasına dikkat etmeliyiz. örnekte geliştirdiğim **NorthwindV2** servis uygulaması, **Asp.Net Development Server** üzerinden **Host** edildiğinden manuel olarak çalıştırılıyor olması gerekebilir. İşte ilk sonuçlar;



Görüldüğü üzere tüm Product bilgileri istemci tarafına gelmiştir. Tabiki koleksiyon içeriğini istemci tarafına indirdikten sonra sorgularıda istediğimiz şekilde değiştirebiliriz. örneğin;

```
grdProducts.DataSource = (from p in products.ItemInfo
                          where p.Item.CategoryID == 1
                          select p.Item).ToList<ItemInfoListItemInfoItem>();
```

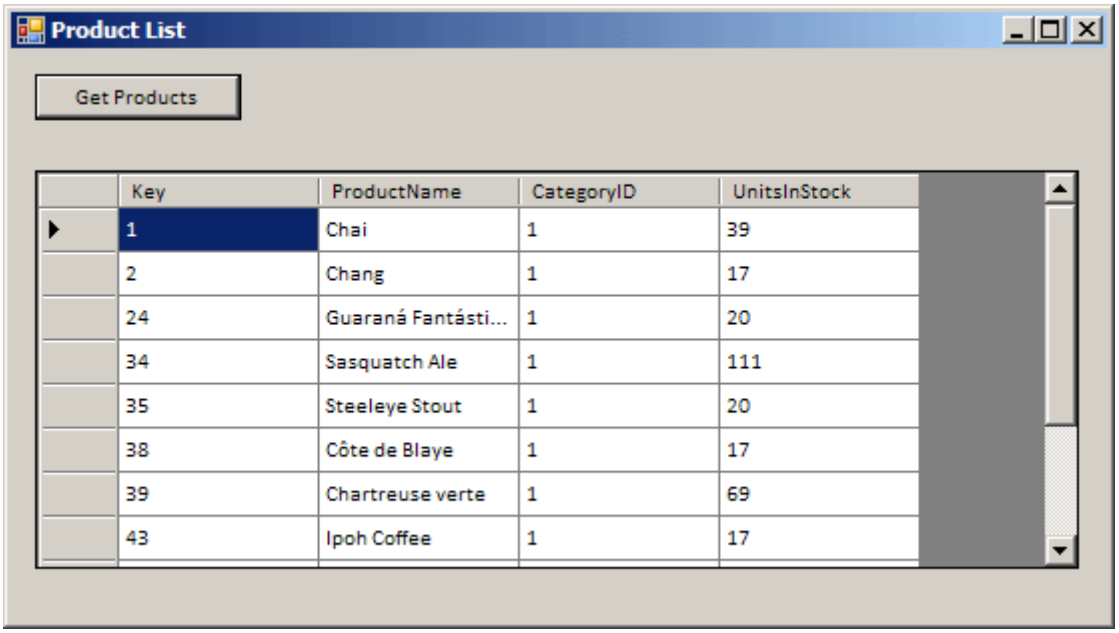
kodunu denediğimizde kategorisi 1 olan ürünlerin getirilmesi sağlanacaktır.



Hatta istersek **anoymous type(isimsiz tip)** kullanımıda söz konusu olabilir. Söz gelimi

```
grdProducts.DataSource = (from p in products.ItemInfo
                          where p.Item.CategoryID == 1
                          select new
                          {
                             Key = p.Item.ProductID,
                             p.Item.ProductName,
                             p.Item.CategoryID,
                             p.Item.UnitsInStock
                          }).ToList();
```

kodu ile sadece **ProductID**, **ProductName**, **CategoryID** ve **UnitsInStock** alanlarını içeren bir isimsiz tip topluluğunu **DataGridView** içerisinde gösterilmesi sağlanabilir.



Gayet kolay gördüğünüz gibi. Artık hedefimiz **Post, Put** ve **Delete** metodlarını istemci tarafından gönderip ele alabilmek. Yazıyı sonlandırmadan önce aslında bu modelin ne gibi bir farkı olduğuna bakmakta yarar var. Dikkat edileceği üzere istemci tarafı için ürettiğimiz herhangibir **Proxy** tipi bulunmamaktadır. (Hiç **Add Service Reference** dediğimi duydunuz mu?)

Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

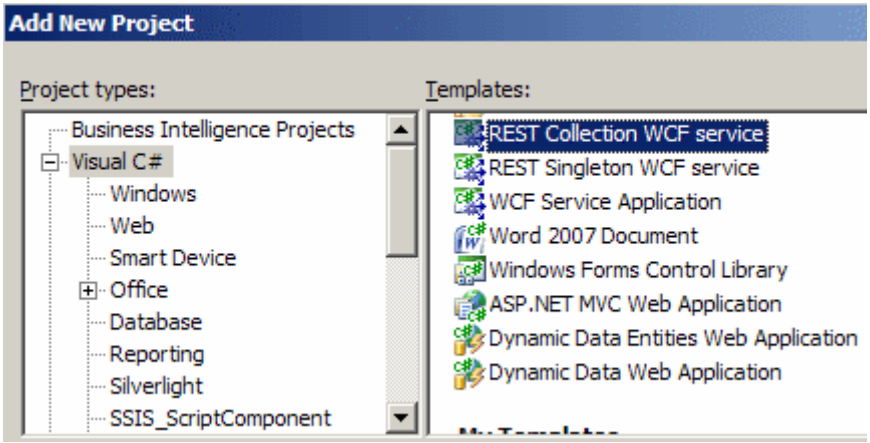
Koleksiyon Bazlı WCF Rest Servisleri (2009-04-22T17:07:00)

wcf,rest,

Merhaba Arkadaşlar,

Bildiğiniz üzere bir süredir **WCF** servislerinin **REST** modeline göre geliştirilmesi ile ilgili bilgilerimi ve öğrendiklerimi sizlerle paylaşmaktayım. Bu nedenle dün gece yaşadığım bir macerayıda aktararak başıma gelenleri sizlerle paylaşmak istiyorum. Bir süre önce WCF Rest Starter Kit' i incelemeye başlamış ve bu konuyla ilgili olaraktan iki görsel video yayınlamıştım.(C#Nedir? bağlantısı , NedirTv? bağlantısı) . Bu görsel derste, veri kaynağı olarak içerdikleri koleksiyonları yanlız okuma(Read-only) amaçlı ele alan REST bazlı WCF servislerinin,**WCF Rest Start Kit Preview 1** sürümü ile nasıl geliştirilebileceğini incelemeye çalışmıştım. Tabi aradan uzun zaman geçti ve **WCF Rest Start Kit Preview 2** sürümü yayınlandı. Ayrıca görsel derste insert, update ve delete işlemlerini ele almamıştım. Bende hazır fırsat varken, bu tip WCF servislerinde **Insert, Update, Delete** işlemlerini nasıl yapabiliriz konusunu araştırmaya başladım. Starter Kit ile birlikte gelen Lab' lar içerisinde (3ncü alıştırmada) bu konu oldukça kolay anlaşılır bir şekilde ele alınmaktadır. Benim size aktaracaklarım daha çok başıma nelerin geldiği.

İlk olarak **REST Collection Service** kavramını biraz açmamızda yarar var. Starter Kit ile birlikte **Visual Studio 2008** ortamına bir proje şablonu olarak gelen bu yapı, **REST** modeline göre veri kümelerinin, servis tarafında koleksiyon bazlı olaraktan ele alınmasını otomatikleştirmektedir. Veriler istemci tarafına **XML** formatı dışında **JSON(JavaScript Object Notation)**standartlarına görede yayımlanabilir. Ayrıca daha önceki yazılarımızda değindiğimiz **WebGet** niteliği ile **UriTemplate'** ler oluşturulmasına gerek yoktur. çünkü buda hazır olarak gelmektedir. Bunlara ek olarak, **HTTP POST, GET, DELETE** ve **PUT** metodlarına cevap verecek şekilde bir çalışma zamanı alt yapısına sahiptir ki bu sayede Select dışında Insert,Update,Delete gibi işlemleride yapabiliriz. Tabiki request' leri doğru bir şekilde gönderebildiğimiz takdirde. Şablonu **Visual Studio 2008** ortamında kullanmak son derece kolaydır.



Aslında Lab içerisindeki adımlarda ilerlerken ilk dikkat çeken noktalardan birisi, **Service** tipinin **CollectionServiceBase<TItem> abstract** sınıfından türemesi ve **ICollectionService<TItem>arayüzünü(Interface)** uygulamasıydı. **TItem** tipi koleksiyon içerisinde kullanılacak veri tipini işaret etmektedir. CollectionServiceBase abstract sınıfı içerisinde, Servis tipi tarafından uygulanması gereken bazı abstract metodlar yer almaktadır**(OnGetItems, OnGetItem, OnAddItem, OnDeleteItem, OnUpdateItem)**. Diğer taraftan arayüzün içerisindede az önce belirttiğimiz

metodların **Json** ve **Xml** formatları için olan tanımlamaları yer almaktadır. Aslında bir servis geliştirilirken bilindiği üzere **Servis Sözleşmesi(Service Contract)** ve onu uygulayan asıl servis sınıfı ele alınmaktadır. **REST Collection Service** şablonunda, sözleşme görevini üstlenen arayüz **ICollectionService<TItem>** dır. Burada tanımlanan operasyonların sahip olduğu**WebHelp, WebGet, WebInvoke** gibi niteliklerin içeriklerinde istenirse oynamalar yapılabilir. Ancak şablon, bu niteliklere varsayılan değerlerini koyarak, hazır bir uygulanış biçiminide sunmaktadır.

Antrenmanı Lab üzerinden adım adım yaparken, asıl veri kaynağı olarak neyi kullanacağımı düşünüyordum. Her zamanki gibi kolaya kaçıp tembellik yaptığımdan,**Northwind** veritabanında yer alan **Products** tablosunu ve CUD işlemleri içinde bir kaç **SP** ile **Enterprise Library** kullanmaya karar vermiştim. Genellikle profesyonel çaptaki projelerde basit CRUD işlemlerini Stored Procedure' ler içerisine almak çok sık yapılan bir şey değildir. Ancak amacım sadece bir veri kümesi kullanmak ve **CUD(CreateUpdateDelete)** işlemlerini **REST** modeli üzerinden test etmek olduğu için bu durumu şimdilik görmezden geldim.

Not : *Tabiki istenirse farklı veri kaynaklarıda koleksiyon bazlı olacak şekilde REST modeline göre servisleştirilebilir. örneğin XML tabanlı bir veri kaynağı ele alınabilir yada program alanında tutulan bir koleksiyon. Nitekim servisin kullandığı* ***Dictionary<string,TItem>*** *koleksiyonu içerisinde tutulan nesneler aslında veriyi sembolize eden birer entity olarak düşünülmelidir.*

Neyse çok fazla dağıtmadan konuyu devam edeyim. **Stored Procedure**' leri temel **CUD** işlemlerini gerçekleştirmek üzere aşağıdaki gibi tasarladım.

Ekleme işlemi

```
CREATE PROCEDURE InsertProduct
        @ProductName nvarchar(40)
       ,@SupplierID int
       ,@CategoryID int
       ,@QuantityPerUnit nvarchar(20)
       ,@UnitPrice money
       ,@UnitsInStock smallint
       ,@UnitsOnOrder smallint
       ,@ReorderLevel smallint
       ,@Discontinued bit
AS
INSERT INTO [Northwind].[dbo].[Products]
       (ProductName
       ,SupplierID
       ,CategoryID
       ,QuantityPerUnit
       ,UnitPrice
```

```
        ,UnitsInStock
        ,UnitsOnOrder
        ,ReorderLevel
        ,Discontinued
        )
    VALUES
        (
        @ProductName
        ,@SupplierID
        ,@CategoryID
        ,@QuantityPerUnit
        ,@UnitPrice
        ,@UnitsInStock
        ,@UnitsOnOrder
        ,@ReorderLevel
        ,@Discontinued
       )
Select SCOPE_IDENTITY()
```

Güncelleme işlemi

```
CREATE PROCEDURE UpdateProduct
   @ProductName nvarchar(40)
        ,@SupplierID int
        ,@CategoryID int
        ,@QuantityPerUnit nvarchar(20)
        ,@UnitPrice money
        ,@UnitsInStock smallint
        ,@UnitsOnOrder smallint
        ,@ReorderLevel smallint
        ,@Discontinued bit
        ,@ProductID int
AS

Update [Northwind].[dbo].[Products]
Set
        ProductName=@ProductName
        ,SupplierID=@SupplierID
        ,CategoryID=@CategoryID
        ,QuantityPerUnit=@QuantityPerUnit
        ,UnitPrice=@UnitPrice
        ,UnitsInStock=@UnitsInStock
        ,UnitsOnOrder=@UnitsOnOrder
        ,ReorderLevel=@ReorderLevel
        ,Discontinued=@Discontinued
```

```
Where
   ProductID=@ProductID
```

Silme işlemi

```
CREATE PROCEDURE DeleteProduct
(
 @ProductID int
)
AS
Delete From Products Where ProductID=@ProductID
RETURN
```

Daha sonrada Servis sınıfını aşağıdaki gibi yeniledim. Yeniledim diyorum, nitekim proje şablonu zaten içerisinde hazır olarak bir uyarlama gerçekleştirmekte ve **SampleItem** isimli bir tipi koleksiyon içerisinde kullanmaktadır. Ayrıca **OnGetItems, OnGetItem, OnAddItem, OnUpdateItem** ve **OnDeleteItem** metodları içinde hazır kodlamalar yer almaktadır. (Bu tip şablonların**Visual Studio 2010** içerisinde dahada otomatikleştirilmesi söz konusu olabilir.)

```
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;
using System.ServiceModel;
using Microsoft.ServiceModel.Web;
using System.ServiceModel.Activation;
using System.Net;
using Microsoft.ServiceModel.Web.SpecializedServices;
using Microsoft.Practices.EnterpriseLibrary.Data;
using System.Data;

[assembly: ContractNamespace("", ClrNamespace = "NorthwindV2")]

namespace NorthwindV2
{
   [ServiceBehavior(IncludeExceptionDetailInFaults = true, InstanceContextMode =
InstanceContextMode.Single, ConcurrencyMode = ConcurrencyMode.Single)]
   [AspNetCompatibilityRequirements(RequirementsMode =
AspNetCompatibilityRequirementsMode.Allowed)]
   public class Service
      : CollectionServiceBase<Product>, ICollectionService<Product>
   {
      // products isimli Dictionary koleksiyonunda Product nesne örnekleri value, ProductID
değerleri ise key olarak tutulmakta.
      Dictionary<string, Product> products = new Dictionary<string, Product>();
```

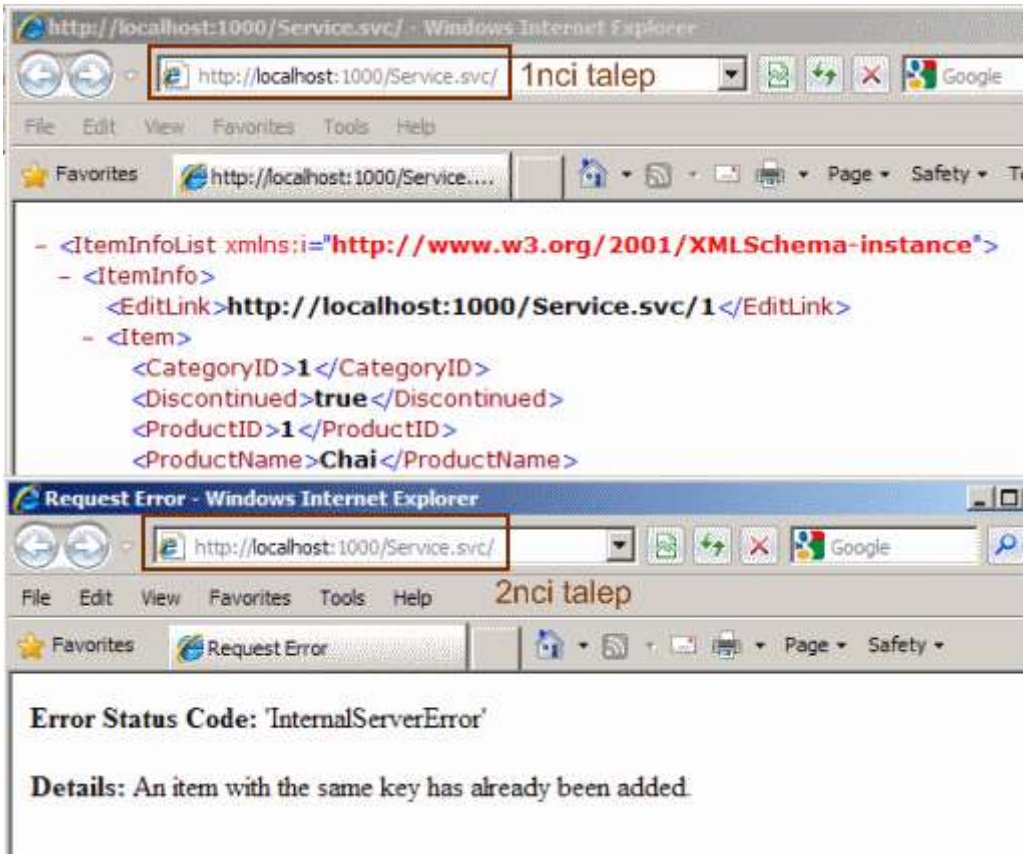
```
        // Enterprise Library' den yararlanarak Database nesnesi üretiliyor.
        Database db = DatabaseFactory.CreateDatabase("NorthConStr");

        // Bu metod ile istemci tarafından gelecek talep sonrasında tüm ürünlerin istenilen formatta gösterilmesi sağlanmaktadır.
        // CategoryID değeri null olmayan satırlar çekilir ve IDataReader üzerinden Product nesnesi şeklinde oluşturulan örnekler, products koleksiyonuna eklenerek geriye döndürülür.
        // http://localhost:1000/Service.svc talebi sonrası bu metod devreye girer.
        protected override IEnumerable<KeyValuePair<string, Product>> OnGetItems()
        {
            IDataReader reader = db.ExecuteReader(CommandType.Text, "Select * From Products where CategoryID is not null");

            while (reader.Read())
            {
                products.Add(reader["ProductID"].ToString(),
                    new Product
                    {
                        ProductID=Convert.ToInt32(reader["ProductID"]),
                        CategoryID=Convert.ToInt32(reader["CategoryID"]),
                         Discontinued=Convert.ToBoolean(reader["Discontinued"]),
                          ProductName=reader["ProductName"].ToString(),
                           QuantityPerUnit=reader["QuantityPerUnit"].ToString(),
                            ReorderLevel=Convert.ToInt16(reader["ReorderLevel"]),
                             SupplierID=Convert.ToInt32(reader["SupplierID"]),
                              UnitPrice=Convert.ToDecimal(reader["UnitPrice"]),
                               UnitsInStock=Convert.ToInt16(reader["UnitsInStock"]),
                                UnitsOnOrder=Convert.ToInt16(reader["UnitsOnOrder"])

                    }
                    );
            }
            return this.products;
        }

        // Id değeri üzerinden bir Product' ın elde edilip geriye döndürülmesi için kullanılır. özellikle Update metodunda dahili olaraktan kullanılmaktadır.
        // Yani http://localhost:1000/Service.svc/4 gibi bir talep sonrası bu metod devreye girmektedir.
        protected override Product OnGetItem(string id)
        {
            int productId;
            // int tipinden olmayan bir ProductId değeri ise
            if (!Int32.TryParse(id, out productId))
```

```
        {
            // Exception fırlatılır
            throw new WebProtocolException(HttpStatusCode.BadRequest);
        }
        return this.products[id];
    }

    // Yeni bir Product tipinin eklenmesi için kullanılan bu metoddan geriye, yeni
oluşturulan satırın ProductID değeri döndürülür(eğer işlemler başarılı ise)
    protected override Product OnAddItem(Product initialValue, out string id)
    {
        try
        {
            id = db.ExecuteScalar("InsertProduct", initialValue.ProductName,
initialValue.SupplierID, initialValue.CategoryID, initialValue.QuantityPerUnit,
initialValue.UnitPrice, initialValue.UnitsInStock, initialValue.UnitsOnOrder,
initialValue.ReorderLevel, initialValue.Discontinued).ToString();
            initialValue.ProductID = Convert.ToInt32(id);
            this.products.Add(id, initialValue);
        }
        catch(Exception excp)
        {
            throw new
WebException(excp.Message.ToUpper(), WebExceptionStatus.RequestCanceled);
        }
        return initialValue;
    }

    // Bir Product' ın güncelleştirilmesi işlemi sırasında kullanılan metoddur. Metod
başarılı bir şekilde güncelleştirme işlemini yaparsa geriye Product tipinin son hali
döndürülür.
    protected override Product OnUpdateItem(string id, Product newValue)
    {
        int result = 0;
        try
        {
            result = db.ExecuteNonQuery("UpdateProduct", newValue.ProductName,
newValue.SupplierID, newValue.CategoryID, newValue.QuantityPerUnit,
newValue.UnitPrice, newValue.UnitsInStock, newValue.UnitsOnOrder,
newValue.ReorderLevel, newValue.Discontinued, Convert.ToInt32(id));
        }
        catch (Exception excp)
        {
            throw new
WebException(excp.Message, WebExceptionStatus.RequestCanceled);
```

```
        }

        if (oldValue == null) // Eğer veri kaynağında güncelleştirilecek bir Product nesnesi yoksa istisna mesajı verilir.
        {
            throw new WebProtocolException(HttpStatusCode.NotFound);
        }

        int result=db.ExecuteNonQuery("UpdateProduct", newValue.ProductName, newValue.SupplierID, newValue.CategoryID, newValue.QuantityPerUnit, newValue.UnitPrice, newValue.UnitsInStock, newValue.UnitsOnOrder, newValue.ReorderLevel, newValue.Discontinued, newValue.ProductID);

        if (result == 1) // ProductID değerleri Auto Identity tipinden olduklarında güncelleştirilen kayıt sayısı 1 ise
        {
            // products koleksiyonundaki değer güncellenir
            this.products[id] = newValue;
            // güncellenen değerlere sahip Product tipi geriye döndürülür.
            return newValue;
        }
        else
            return null; // Aksi durumda null döndürülür
    }

    // Bir Product' ın silinmesi için kullanılan metoddur. Silme işleminin başarılı olması halinde true değeri döndürülür.
    protected override bool OnDeleteItem(string id)
    {
        // İlk olarak silinmek istenen id değerine sahip Product tipi çekilir
        Product item = OnGetItem(id);
        // Eğer ilgili Product null ise false değer döndürülür
        if (item == null)
            return false;
        // Eğer var ise Products tablosundan silme işlemi yapılır
        int result=db.ExecuteNonQuery("DeleteProduct", id);
        // Eğer 1 satır silinebildiyse,
        if (result == 1)
        {
            // products koleksiyonundan da çıkartma işlemi yapılır.
            this.products.Remove(id);
            return true;
        }
        else
            return false;
```

```
        }
    }

    // Koleksiyon içerisinde kullanılan Product sınıfı.
    public class Product
    {
        public int ProductID { get; set; }
        public string ProductName { get; set; }
        public int SupplierID { get; set; }
        public int CategoryID { get; set; }
        public string QuantityPerUnit { get; set; }
        public decimal UnitPrice { get; set; }
        public short UnitsInStock { get; set; }
        public short UnitsOnOrder { get; set; }
        public short ReorderLevel { get; set; }
        public bool Discontinued { get; set; }
    }
}
```

Artık herşey test için hazırdı. İlk etapta **1000** numaralı port üzerinden hizmet verecek olan servisi F5 ile çalıştırdım. **Lab** içerisinde söz konusu **REST** servisinin testi sırasında **Post, Put, Delete** talepleri için bir istemci uygulama yerine Fiddler aracı kullanılmaktaydı. Yani yeni bir ürün eklemek, silmek veya güncellemek istediğimde istemci tarafından gönderilecek olan **HTTP**paketini **Fiddler** aracı yardımıyla hazırlayıp gönderilmesi öneriliyordu. Bu sadece test ve içerik analizi için bir öneriydi. Ancak daha o adımlara geçmeden önce ilk hata mesajım ile karşılaştım. İlk etapta sorun yok gibi görünüyordu. Ancak aynı servisi ikinci bir tarayıcı penceresinden talep ettiğimde aşağıdaki görüntü ile karşılaştım.

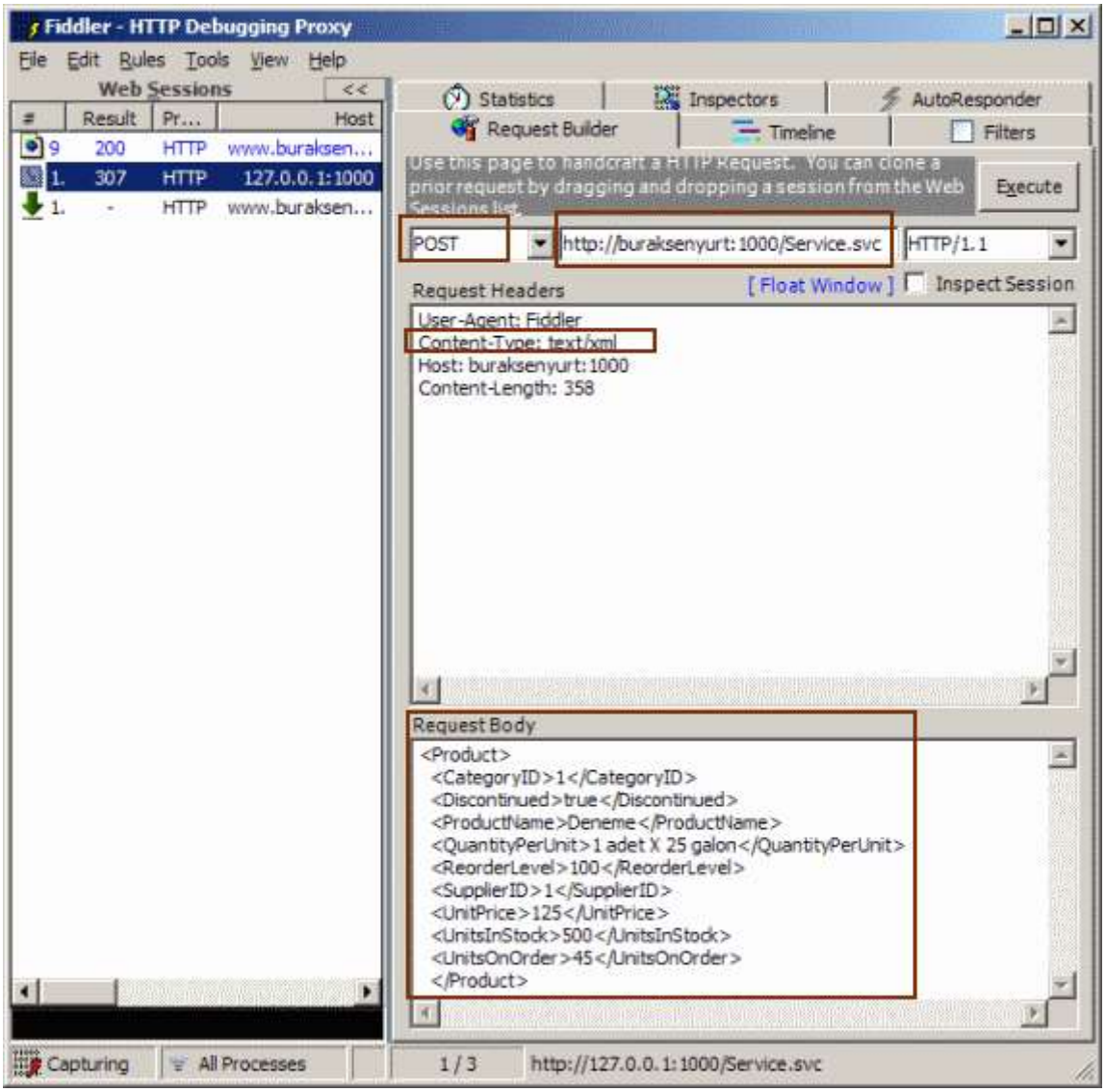


Hata nerededir diye araştırırken aslında her talepte **OnGetItems** metodunun çağırıldığını ve bu sebeple koleksiyona veri ekleme işleminden önce aslında temizlenmesi gerektiğini farkettim. Dolayısıyla kodu aşağıdaki halde yenilemek sorunu çözdü.

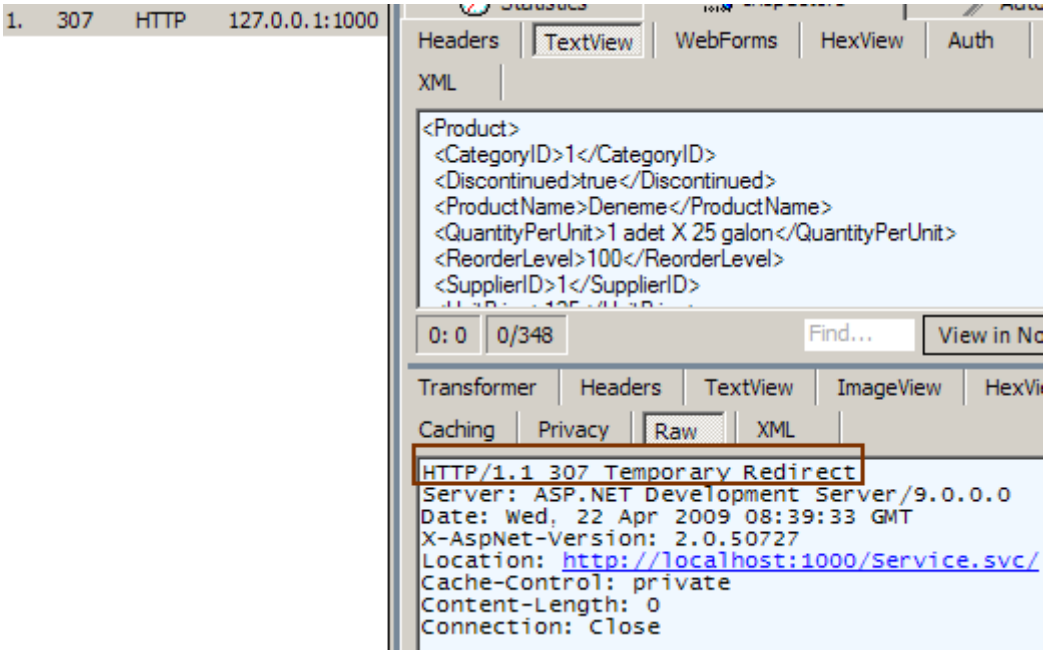
```
protected override IEnumerable<KeyValuePair<string, Product>> OnGetItems()
{
  products.Clear(); // Temizlemediğimizde ikinci bir request için hata mesajı alınır.
  IDataReader reader = db.ExecuteReader(CommandType.Text, "Select * From Products
where CategoryID is not null");
```

*Not : Tabi burada servisin çok sık değişmeyen bir koleksiyonu yayınlaması durumunda, her talep için veritabanından bir **Select** sorgusu ile veri çekmesi yerine, performansı arttırmak için belki **önbellekleme(Caching)** sistemi kullanılabilir. Nitekim servis, sonuç itibariyle **Asp.Net Host** ortamında sunulmaktadır. Bu nedenle **Cache** mimarisini ele alabilir. Hatta**SqlCacheDependency** kullanılarak Cache içeriğinin gerçekten tabloda değişiklik olduğu durumlarda ele alınmasıda sağlanabilir. Bu durumu ilerleyen yazılarımızda ele almayı planlıyorum.*

Son sorunu çözdükten sonra hemen yeni bir satır Product eklemeye karar verdim. Aynen Lab' da belirtildiği gibi, paketi manuel olarak Fiddler aracı ile hazırlayıp servise gönderim.

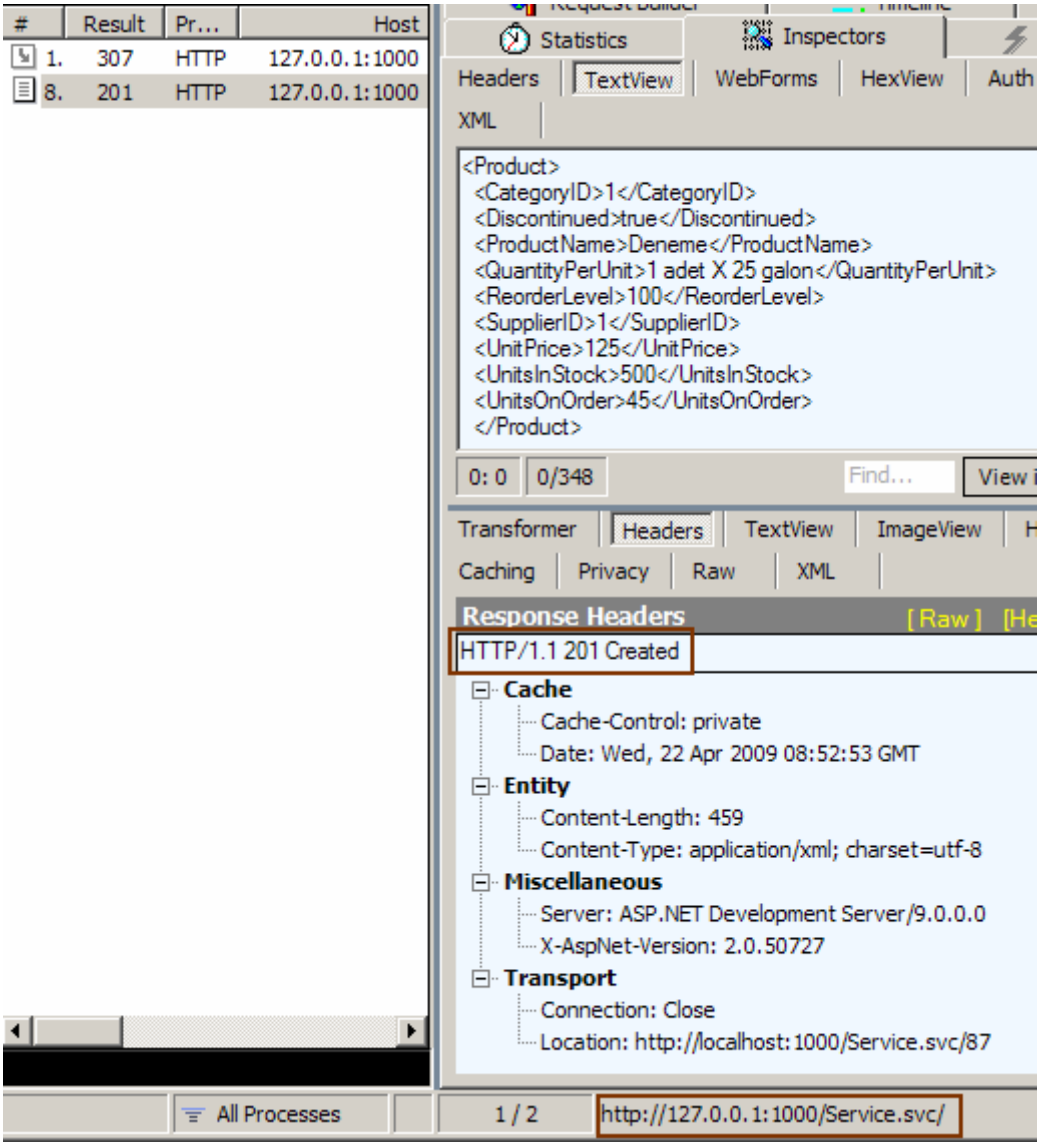


Burada talep metodunun **POST** olarak seçildiğine, **Content** tipinin **text/xml** olarak belirtildiğine dikkat etmek lazım. Diğer tarafında **RequestBody** kısmında manuel olarak yazdığımız **XML**içeriğinde **ProductID** değeri yazmadığımı da belirtelim.
Nitekim, **ProductID** otomatik artan ve insert sorgusuna dahil edilmeyen bir alandır. Ancak ne varki **Execute** işleminden sonra servis tarafından **307** kodlu bir cevap gelmiştir**(Temporary Redirect).** ☹ Oysaki **201** cevabının gelmesi gerekirdi.

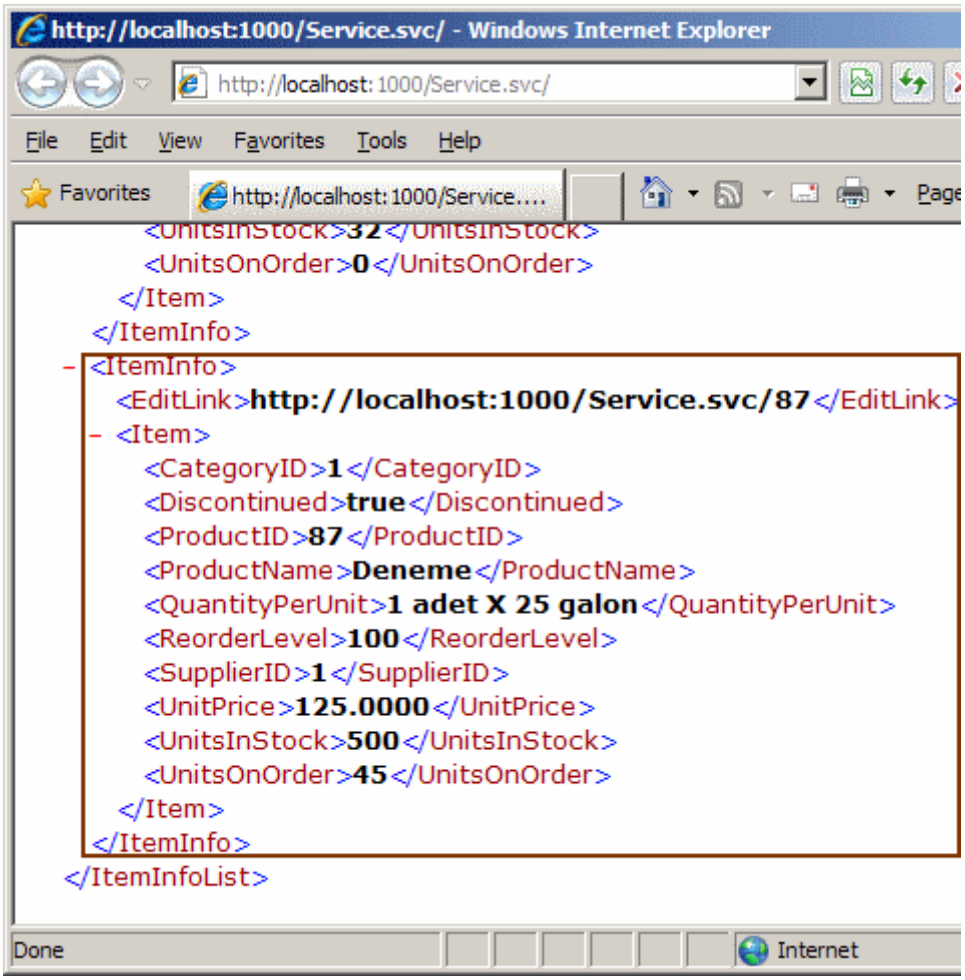


Bu hatayla uzun bir süre cebelleştikten sonra, sorunun adres kısmını yanlış yazmamdan kaynaklandığını tespit ettim. Yani adresin **http://buraksenyurt:1000/Service.svc** adresinin**http://buraksenyurt:1000/Service.svc/** olarak yazılması gerekiyormuş. Tamamen benim hatam...Adresi bu şekilde düzelttikten sonra insert işleminin gerçekleştirildiğini ve hem koleksiyonda hemde Products tablosunda yeni Product tipi için gerekli eklemelerin yapıldığını görebildim.

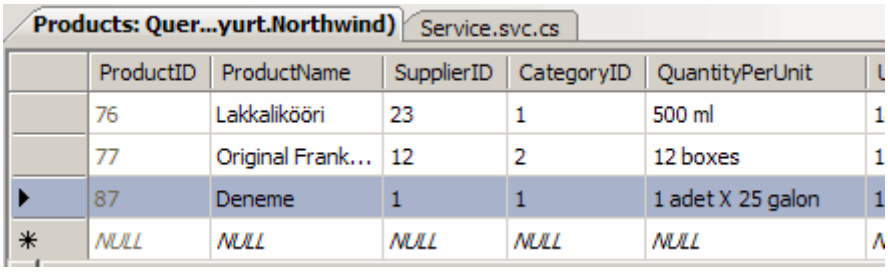
Fiddler görüntüsü



Tarayıcı görüntüsü

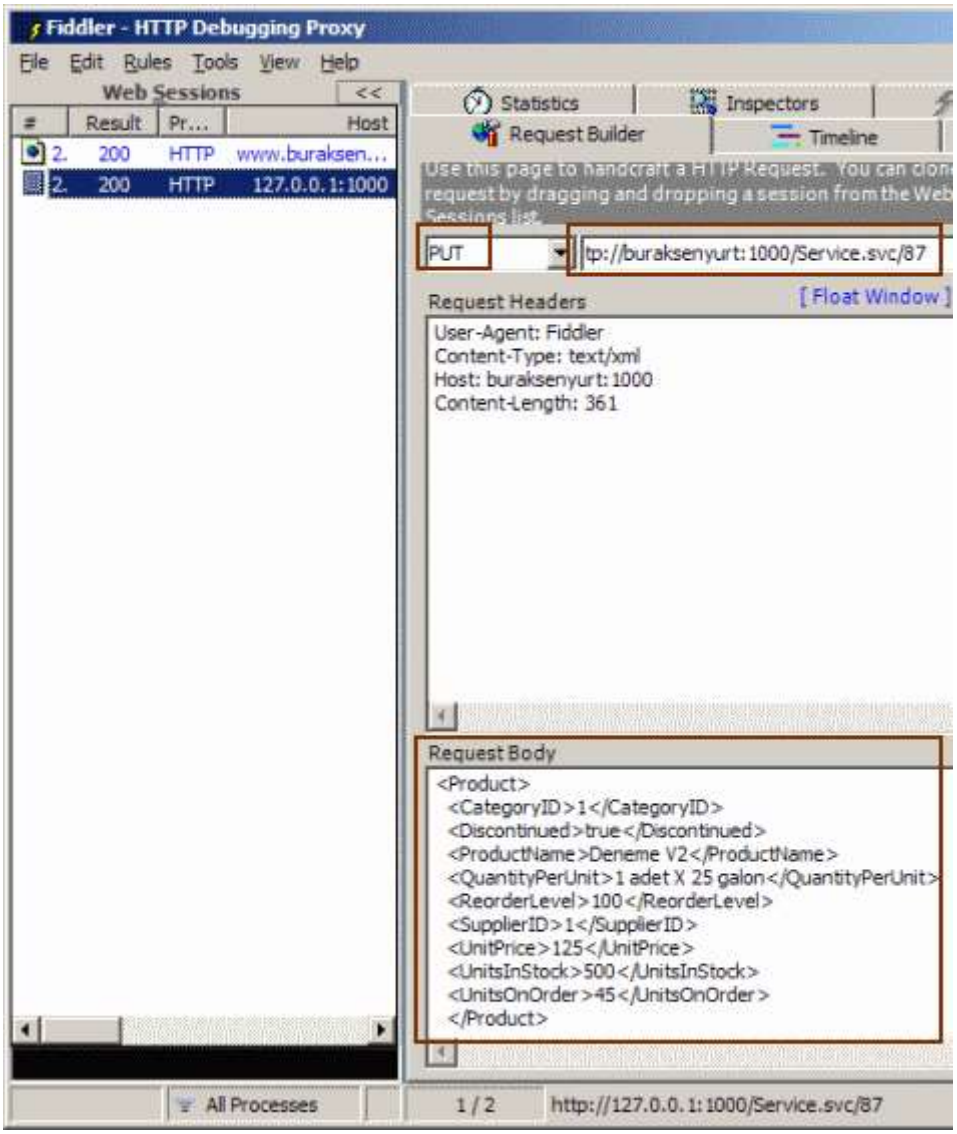


SQL Tarafı

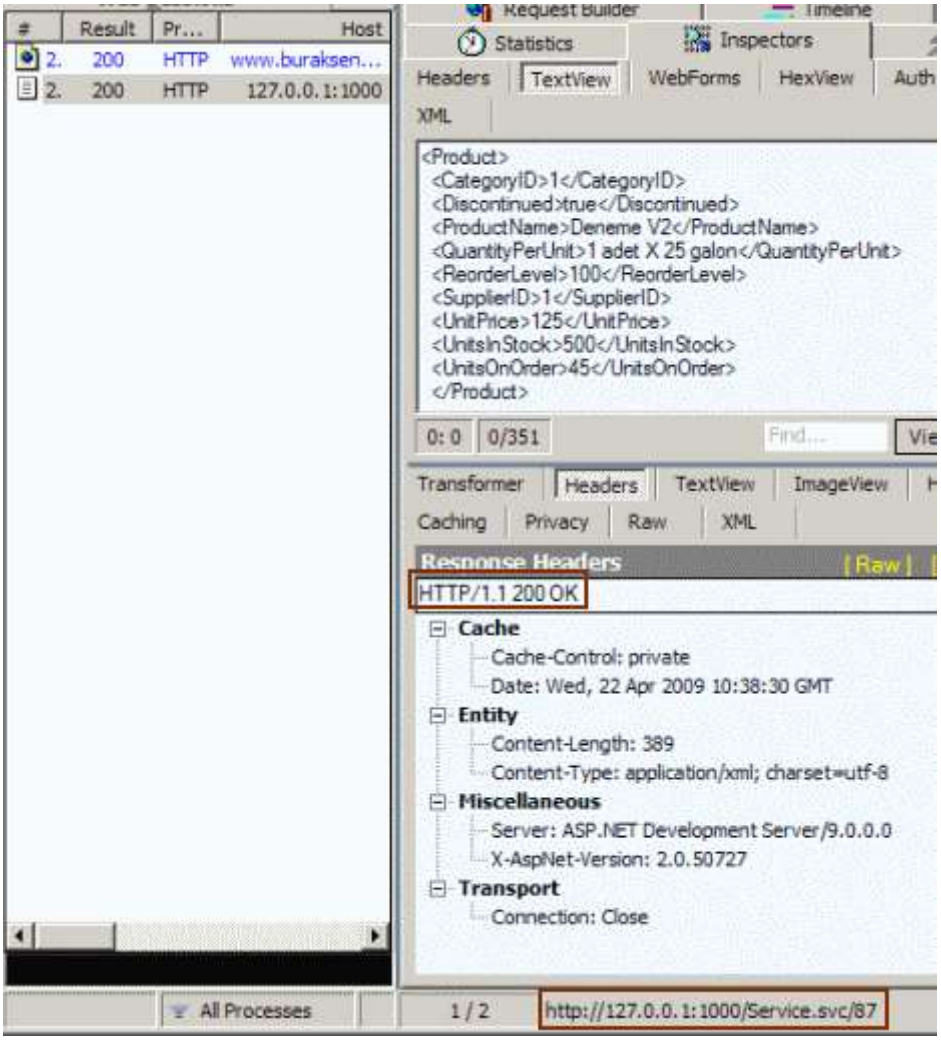


ProductID	ProductName	SupplierID	CategoryID	QuantityPerUnit
76	Lakkalikööri	23	1	500 ml
77	Original Frank...	12	2	12 boxes
87	Deneme	1	1	1 adet X 25 galon
NULL	NULL	NULL	NULL	NULL

Artık güncelleme ve silme işlemlerini tespit edebilirdim. Güncelleştirme işlemi sırasında dikkat etmem gereken noktalardan ilki, **HTTP** protokolünün **Put** metodunu kullanmam gerektiğiydi. Ancak ilk denemede yine patlayınca aslında güncelleştirmek istediğim satırın **ProductID** değerini **Request Body** içerisindeki **XML** kısmında değil, **URL** kısmında belirtmem gerektiğini farkettim. 😐 Buna göre **Fiddler** aracı yardımıyla aşağıdaki talebi gönderdiğimde,

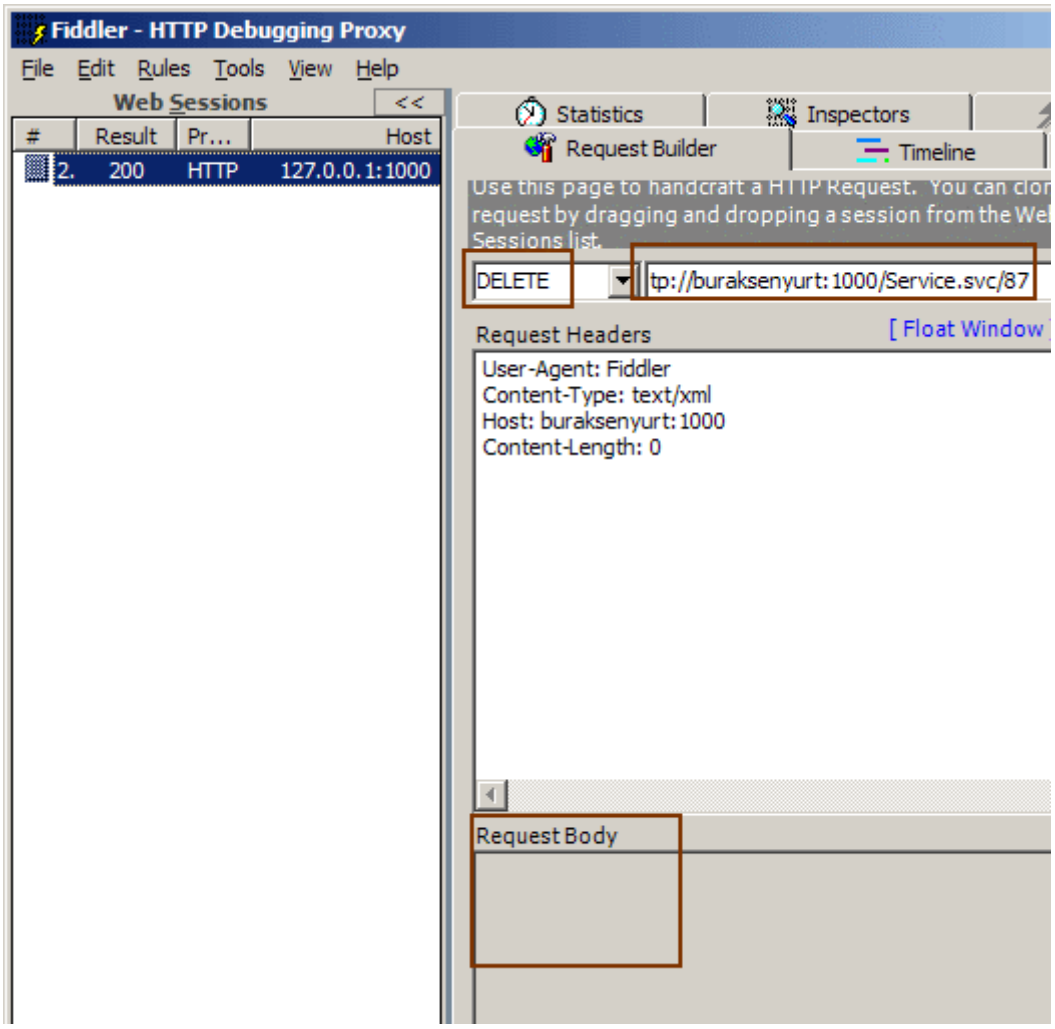


servis tarafından **200-Ok** cevabını alabildiğimi gördüm.



İşte mutluluğun resmi.

Resmi tamamlamak için son olarak **delete** işlemini test etmem gerekiyordu. Bu sefer **HTTP** protokolünü kullanarak göndereceğim talepte, **Delete** metodunu seçmem gerektiğini biliyordum. Ayrıca silmek istediğim **Product** satırının **ProductID** değerinide bir önceki **Update** işlemine göre **URL** satırından göndermem gerektiğinin farkındaydım. Hatta **Request**' in **body** kısmında herhangibir bilgi olmaması gerektiğinide tahmin edebilmiştim. Dolayısıyla tek seferde çalıştırabileceğim düşüncesindeydim.



İşte bu kadar. Bir maceramızın daha sonuna geldik. **REST** bazlı **WCF** servislerinin kullanımı ile ilişkili çalışmalarıma ve araştırmalarıma devam ederken bunları sizlerlede paylaşıyor olacağım. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

örnek Uygulama ; NorthwindV2.rar (322,50 kb)

Sp ler için script dosyası : script.sql (4,65 kb)

WCF Rest Modelinde UriTemplate Kullanımı (2009-04-21T04:19:00)

wcf,rest,

Merhabalar,

SOAP(Simple Object Access Protocol) bazlı **WCF** servislerininin **REST(REpresentational State Transfer)** modeline taşınmasını ele aldığımız bir önceki yazımızda varsayılan **URL**şablonu kullanılmıştır. Varsayılan URL şablonu, **WebGet** niteliğinde herhangibir başka desen belirtilmediğinde devreye girmektedir. Kabaca aşağıdaki dizime benzer bir yapıdadır.

http://servisAdresi/servisAdi.svc/**OperasyonAdi?parametre1=parametreDegeri&parametre2=parametreDegeri**

Bu standart şablona göre, **WebGet niteliği(Attribute)** ile imzalanmış olan metod, **operasyonAdi** kısmına gelmekte, sonrasında ise eğer varsa metod parametreleri ve değerleri yer almaktadır. Parametre adları metodda kullanılanlar ile aynı olmalıdır. Söz gelimi ürünlerin belirli bir kategori altında olanlarını getirmek istediğimizde, **REST** bazlı bir servise gönderilecek talepler için aşağıdakine benzer bir adresleme kullanılır.

http://servisAdresi/**servisAdi.svc/GetProducts?CategoryID=1**

Oysaki bunun yerine,

http://servisAdresi/servisAdi.svc/**Products/1**

veya

http://servisAdresi/servisAdi.svc/**Products(1)**

ve hatta

http://servisAdresi/**Products/Kitap**

gibi adreslemelerde bulunmak daha anlaşılırdır.

Nitekim günümüz web teknolojilerinde, **URL** üzerinde daha anlaşılır bilgilerin yazılması tercih edilmektedir(örneğin arama motorları bu kriterlere çok fazla dikkat eder) **ASP.Net** tarafında**URL Rewriting** teknikleri ile ele alınan, **MVC(ModelViewController**, ki **Uğur Umutluoğlu-MVP** hocamızın çok güzel bir makelesi vardır, buradan okuyabilirsiniz.) deseninde önemli bir yere sahip olan, **ADO.Net Data Service'** lerde varsayılan olarak kullanılan URL şablonları, istenirse **WebGet** niteliğinin **UriTemplate** özelliği ile **REST** modeline taşınmış WCF servislerinde de uygulanabilir. Bu yazımızda bu konsepti ele almaya çalışıyor olacağız. Ne kadar basit olduğunu biraz sonra sizlerde göreceksiniz.

Her zamanki gibi elimizde REST modelin taşınmış hazır bir WCF Servis uygulaması olduğunu düşünelim. Uygulamada yer alan basit bir operasyon, **Northwind** veritabanına bağlanarak**Products** tablosundan, kullanıcıdan gelen **categoryId** değerine sahip olanları, **Product** sınıfı tipinden örnekleri taşıyan **generic** bir **List<Product>** koleksiyonunda geriye döndürmektedir. özellikle veri çekilmesi için basit bir **Stored Procedure** kullanılmakta ve kod tarafında bu işlem **Enterprise Library 4.1** sürümü ile gerçekleştirilmektedir.

Sp içeriği

```
CREATE PROCEDURE dbo.GetProductsByCategory
(
 @CategoryID int
)

AS
 Select
  ProductID
  ,ProductName
  ,SupplierID
  ,CategoryID
  ,QuantityPerUnit
  ,UnitPrice
  ,UnitsInStock
  ,UnitsOnOrder
  ,ReorderLevel
  ,Discontinued
  From Products Where CategoryID=@CategoryID

 RETURN
```

Söz konusu uygulamanın **servis sözleşmesi(Service Contract)** ve uygulayıcı içerikleri ise aşağıdaki gibidir.

Product isimli veri tipimiz

```
using System;

namespace Northwind
{
   public class Product
   {
      public int ProductID { get; set; }
      public string ProductName { get; set; }
      public int SupplierID { get; set; }
      public int CategoryID { get; set; }
      public string QuantityPerUnit { get; set; }
      public decimal UnitPrice { get; set; }
      public short UnitsInStock { get; set; }
      public short UnitsOnOrder { get; set; }
      public short ReorderLevel { get; set; }
      public bool Discontinued { get; set; }
   }
}
```

Servis sözleşmesi

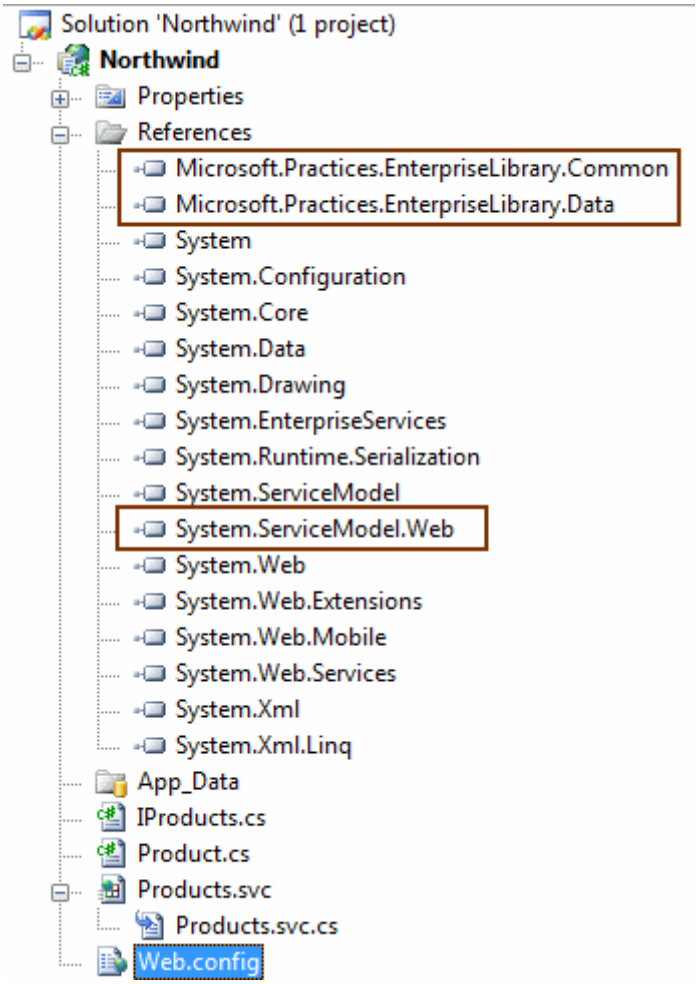
```
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Collections.Generic;

namespace Northwind
{
   [ServiceContract]
   public interface IProducts
   {
      [OperationContract]
        [WebGet]
      List<Product> GetProducts(string categoryId);
   }
}
```

Uygulayıcı tip

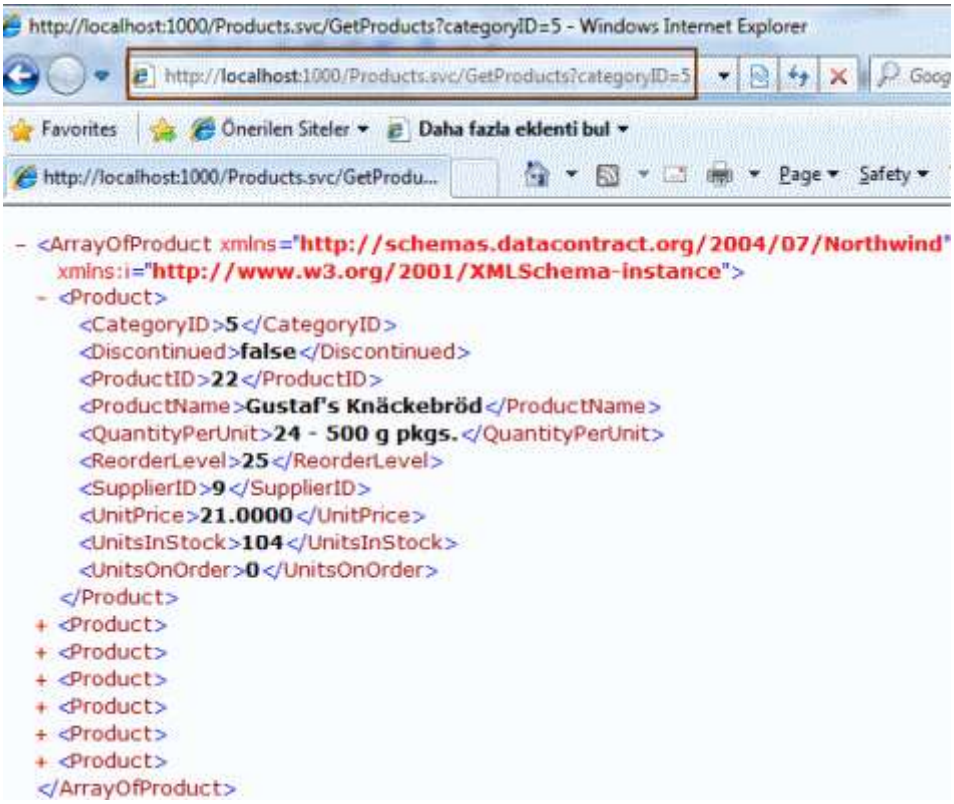
```
using System;
using Microsoft.Practices.EnterpriseLibrary.Data;
using System.Data;
using System.Collections.Generic;

namespace Northwind
{
   public class Products
      : IProducts
   {
      #region IProducts Members

      public List<Product> GetProducts(string categoryId)
      {
         List<Product> products = new List<Product>();
         Database db = DatabaseFactory.CreateDatabase("NorthConStr");
         IDataReader
reader=db.ExecuteReader("GetProductsByCategory",Convert.ToInt32(categoryId));
         while (reader.Read())
         {
            products.Add(
               new Product
               {
                   CategoryID=Convert.ToInt32(reader["CategoryID"]),
                    ProductID=Convert.ToInt32(reader["ProductID"]),
                     ProductName=reader["ProductName"].ToString(),
```

```
                    QuantityPerUnit=reader["QuantityPerUnit"].ToString(),
                     ReorderLevel=Convert.ToInt16(reader["ReorderLevel"]),
                     SupplierID=Convert.ToInt32(reader["SupplierID"]),
                      UnitPrice=Convert.ToDecimal(reader["UnitPrice"]),
                       UnitsInStock=Convert.ToInt16(reader["UnitsInStock"]),
                        UnitsOnOrder=Convert.ToInt16(reader["UnitsOnOrder"]),
                         Discontinued=Convert.ToBoolean(reader["Discontinued"])
                }
                );
            }
            return products;
        }

        #endregion
    }
}
```

Bu arada unutmamamız gereken noktalardan biriside, **Enterprise Library** için **Microsoft.Practices.EnterpriseLibrary.Common**, **Microsoft.Practices.EnterpriseLibrary.Data** ile **WebServiceHostFactory** tipi için(Servisin çalışma zamanında REST modeline göre ele alınması için gereken ve **Markup** kısmındaki ServiceHost direktifi içerisinde **Factory** niteliği ile belirtilen tiptir) **System.ServiceModel.Web assembly'** larının referans edilmesidir.



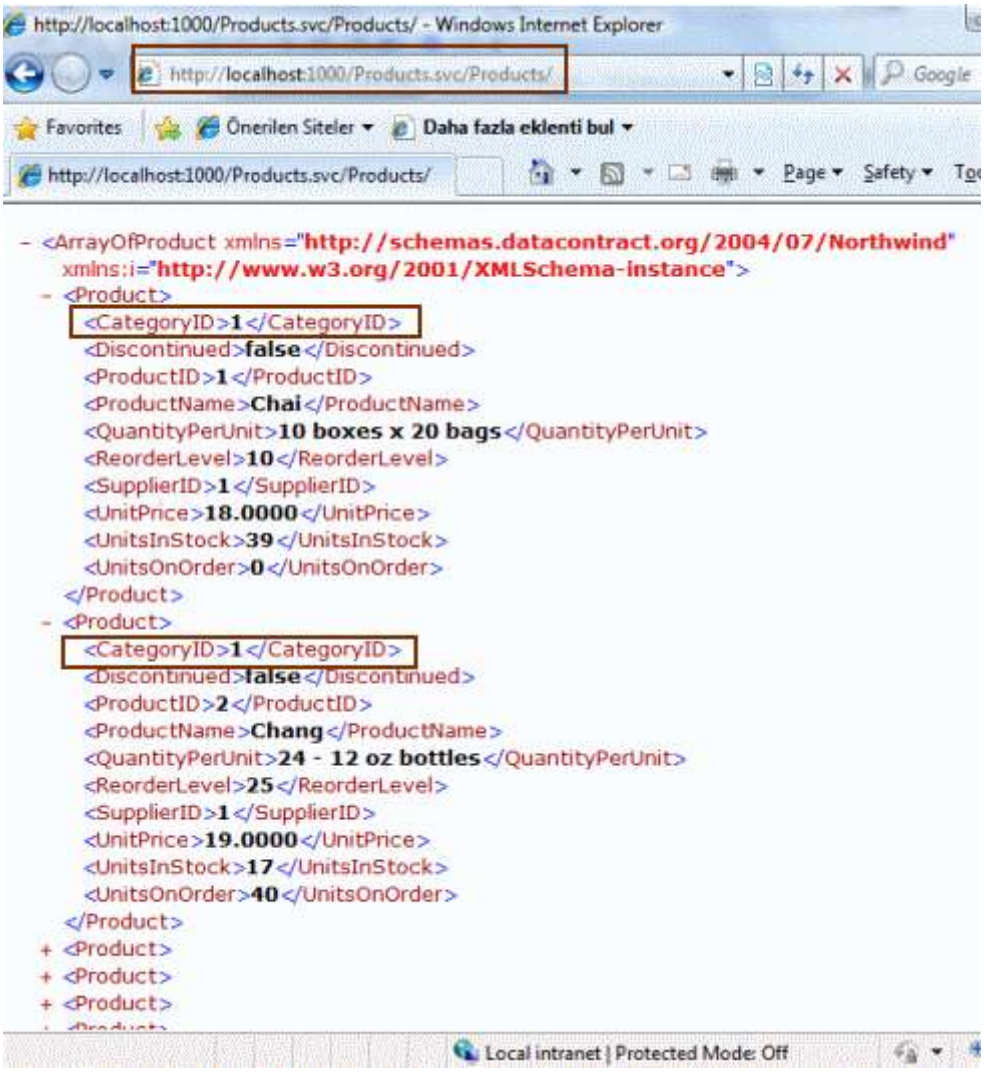
Şimdiii.....Bu haliyle Product.svc talep edildiğinde her hangibir kategoriye ait ürünleri elde edebilmek için aşağıdaki ekran görüntüsünde yer alan URL diziminin kullanılması gerekmektedir.



Burada URL için varsayılan davranışı görmekteyiz. Sırada ilk hamlemiz var. Bu hamlemizde **WebGet** niteliğini aşağıdaki gibi değiştiriyoruz.

```
[WebGet(UriTemplate = "Products/{categoryId=1}")]
List<Product> GetProducts(string categoryId);
```

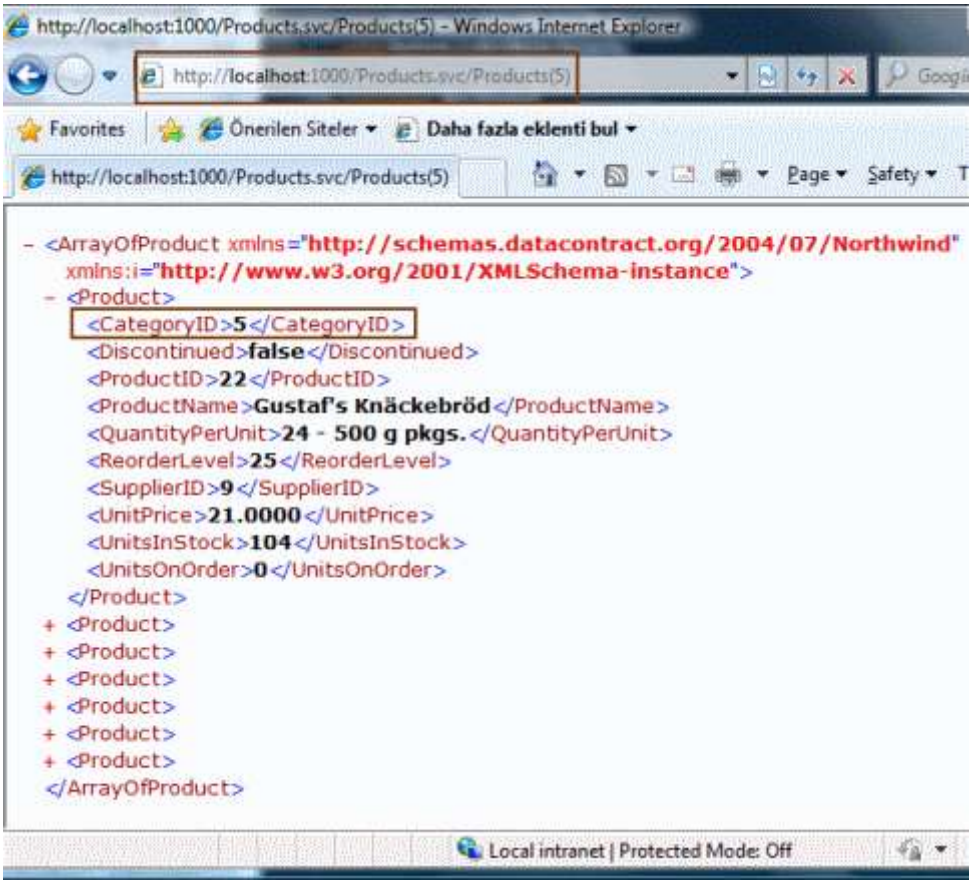
Buna göre categoryId için varsayılan olarak 1 değerini belirtmiş oluyoruz. Yani, kullanıcı talebinde eğer categoryId değeri girilmesse, varsayılan olarak 1 olanları getirecektir.



Elbette artık **Products/2** veya **Products/3** gibi kullanımlarda mümkün olacaktır. Hatta **WebGet** niteliğindeki **UriTemplate** kısmını,

[WebGet(UriTemplate = "Products/({categoryId=1})")]
List<Product> GetProducts(string categoryId);

şeklinde değiştirirsek, aşağıdaki ekran görüntüsünde yer alan sonuçları elde edebiliriz.



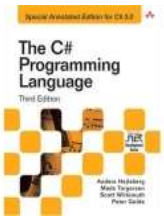
Ne kadar basit öyle değil mi? Hatta istersek **IIS 7.0** üzerindeki ayarları kullanarak(bir sonrak yazımda ele almaya çalışacağım) **svc** uzantılı kısımlardan kurtulabilir ve URL kısmını dahada özelleştirebiliriz.

Böylece geldik bir blog yazımın daha sonuna. Bu yazımızda WebGet niteliğinin UriTemplate özelliğini kullanarak, istemciden servis tarafına gelecek olan URL bilgilerinin nasıl özelleştirilebileceğini, WCF Resf modeli için değerlendirmeye çalıştık. Tekrardan görüşünceye dek hepinize mutlu günler dilerim...

Northwind.rar (234,38 kb)

Her C# Programcısının Yanı Başında Olması Gerekenler (2009-04-19T07:00:00)

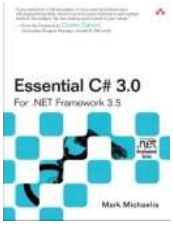
c#,



The C# Programming Language

Son zamanlarda okuduğum en başarılı C# kitaplarından birisi. çok başarılı bir kitap, nitekim yazarlarından biriside **Anders Hejslberg**. C# ve .Net platformunun babası diyebileceğimiz Anders Hejlsberg' in kaleminden bu dili tanımak bir başka zevk. Yıllardır C# ile programlama yapmama ve pek çok konusunu biliyor olmama rağmen, bildiklerimi tekrar etme, ölçme ve

farklı çok farklı bir bakış açısı ile değerlendirme fırsatı buldum. Gerçekten de bazı durumlarda, profesyonel bakış açısına sahip insanları dinlemek veya okumak, bizlere, yazılım sevdalılarına çok fazla katkı sağlayabiliyor.



Essential C# 3.0 For .NET Framework 3.5

Bu kitabın C# 2.0 için olan versiyonunu okumuş ve Essential demesine rağmen, örneğin Thread konusunda girdiği ayrıntı ve detayları görünce, Essential kavramının çok daha ötesinde olduğunu anlamıştım. C# 3.0 versiyonuda aynı derinliğe sahip. Gerçekten dilin her ayrıntısını en ince detayına kadar inceleyen ve sade bir dille aktarmayı başarabilen bir kitap. üstelik C# dilini iyi bildiğini düşünen birisi bu kitabı okuduktan sonra, "Hımmm...Ya şu konuları okuduğum iyi oldu...Böyle bir şeyde varmış...Bu sebepten öyle yazılıyormuş" gibi kendi kendine söylemlerde bulunabilir.

More Effective C# , 50 Specific Ways to Improve Your C#

İşte bir başka güzel kaynak. Bu kitapta C# programlama dilini kullanırken daha verimli kodlama yapmamızı sağlayan yollardan bahsedilmekte. Bir önceki kitap kapağı yeşil-siyah renkteydi(ki şu anda bu ayrıntının herhangibir anlamı yok). Kitap **Microsot MVP'** lerinden ve aynı zamanda **Regional Director'** lerinden olan **Bill Wagner** tarafından kaleme alınmış. çok ilginç konular içermekte. üstelik sadece 336 sayfalık bir kitap. Bir başka deyişle öğlen aralarında 15er dakikanızı bile ayırmanız en azından fikir sahibi olmanız açısından yeterli. Tabiki mutlaka öğrenilenlerin uygulanması gerekiyor. Biliyor olsakta...çünkü pratik yapmak mükemmelleştirir.

Tabiki bir süredir **C# 4.0** programlama dili ve beraberinde gelen yeniliklerden bahsedilmekte(dynamic tip kullanımı, opsiyonel ve isimlendirilmiş parametreler, arttırılmış COM verimliliği ve iyileştirmeleri vb...). Dolayısıyla yakın zamanda bu konu ile ilişkili kitaplarıda göreceğiz. Ancak versiyon yenilensede, ek özellikler ilave edilsede, programlama dilinin temellerinde çok büyük değişiklikler olmamaktadır. özellikle dilin **OOP** kriterlerini sağladığı noktaları iyi bilmek, yeni gelen özellikleride kolayca anlayabilmemizde önemli bir etkendir. O nedenle burada bahsettiğim kitapların mutlaka faydası vardır, olacaktır.

Soap Bazlı WCF Servislerini REST Modeline Taşımak (2009-04-18T05:19:00)

wcf,rest,

.Net Framework 3.5 ile birlikte, **WCF(Windows Communication Foundation)** tarafına kazandırılan önemli yeteneklerden biriside **Web programlama** modelidir. Bu modelin getirileri arasında, WCF servislerinin **REST**(Representational State Transfer) tekniğine göre yazılıp, kullanılabilmesi de vardır. özellikle **SOAP(Simple Object Access Procotol)** bazlı WCF Servisleri ile **REST** modeline göre tasarlanmış servisler arasındaki en

büyük fark, **HTTP** metodunun çeşididir. SOAP bazlı modelde istemciler proxy' ler aracılığıyla HTTP protokolünün **POST** metoduna göre isteklerini gönderirler. REST modeline göre tasarlanmış bir servise ise HTTP protokolünün **GET** metoduna göre talepte bulunulmaktadır. URL bazlı **Querystring** parametreleri ele alınabilir, **URL Rewriting/URL Routing** tarzında taleplerin istenmesi sağlanabilir(İlerleyen bir yazımda nasıl ele alındığını göstereceğim). Bu yazımızda SOAP bazlı olarak tasarlanmış bir WCF servisinin, basitçe REST modeline nasıl taşınabileceğini incelemeye çalışacağız. öncelikli olarak elimizde aşağıdaki kod parçalarında yer alan sözleşme**(Service Contract)** ve tiplere sahip bir **WCF Service Application'** ımız olduğunu düşünelim.

Servis sözleşmemiz;

```
using System.ServiceModel;

namespace NorthwindServices
{
   [ServiceContract]
   public interface IProductService
   {
      [OperationContract]
      Product GetProduct(int id);
   }
}
```

Uygulayıcı tip;

```
namespace NorthwindServices
{
   public class ProductService
      : IProductService
   {
      #region IProductService Members

      public Product GetProduct(int id)
      {
         return new Product
         {
            Id=id,
            Name="Bisiklet(Hemde Kırmızı)",
            ListPrice=10.45
         };
      }
```

```
        #endregion
    }
}
```

Veri sözleşmeli tipi;

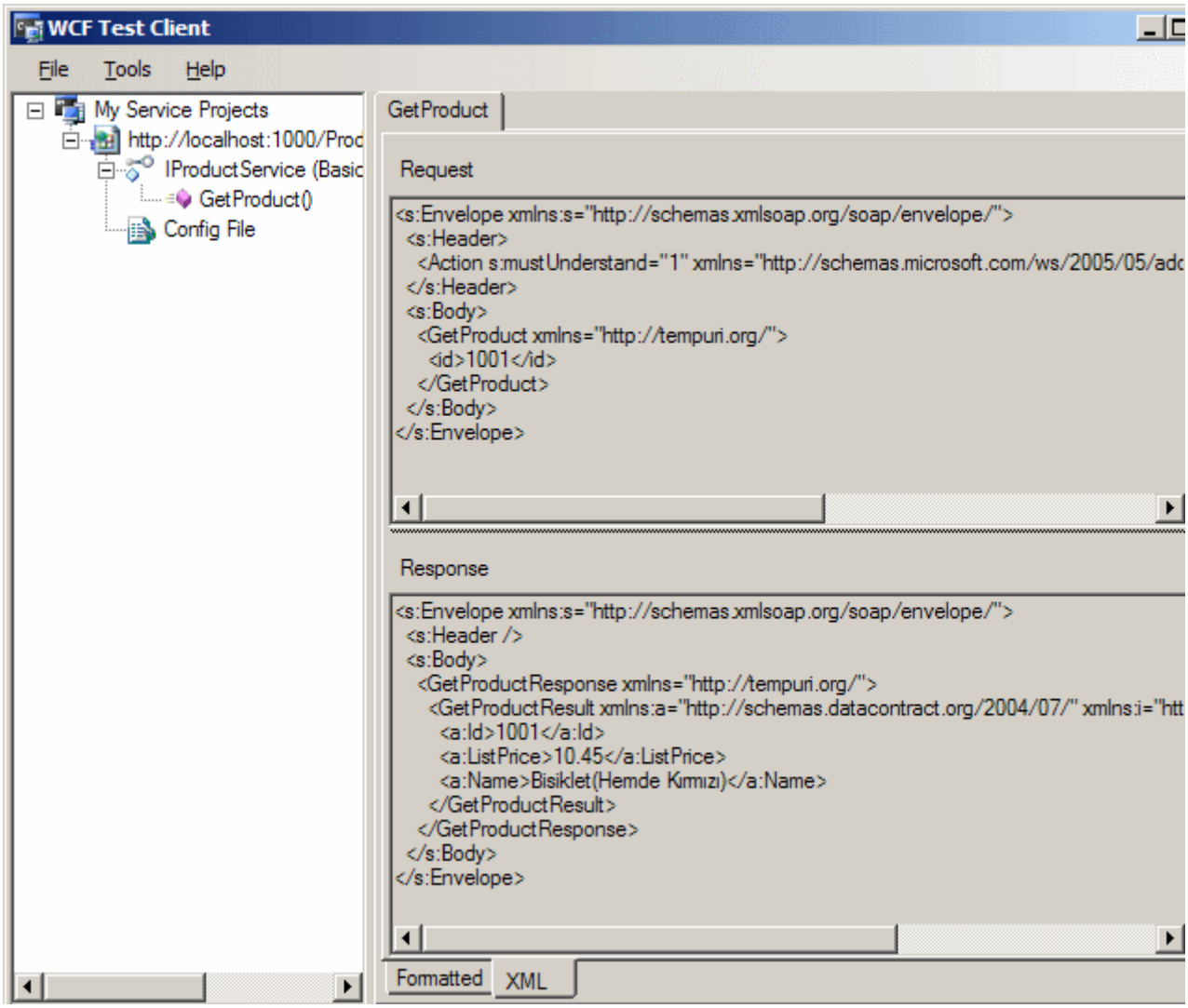
```
using System.Runtime.Serialization;

[DataContract]
public class Product
{
    [DataMember]
    public int Id { get; set; }
    [DataMember]
    public string Name { get; set; }
    [DataMember]
    public double ListPrice { get; set; }
}
```

Söz konusu servis SOAP bazlı tasarlandığı için web.config dosyasındaki system.serviceModel içeriğide aşağıdaki gibidir.

```
ScriptResourceHandler, System.Web.Extensions, Version=3.5.0.0, Culture=neutral, PublicKeyToken=31B
    </handlers>
  </system.webServer>

 <system.serviceModel>
  <services>
   <service behaviorConfiguration="NorthwindServices.ProductServiceBehavior"
    name="NorthwindServices.ProductService">
    <endpoint address="" binding="basicHttpBinding" contract="NorthwindServices.IProductService">
    </endpoint>
    <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
   </service>
  </services>
  <behaviors>
   <serviceBehaviors>
    <behavior name="NorthwindServices.ProductServiceBehavior">
     <serviceMetadata httpGetEnabled="true" />
     <serviceDebug includeExceptionDetailInFaults="false" />
    </behavior>
   </serviceBehaviors>
  </behaviors>
 </system.serviceModel>
</configuration>
```

Bu servis test edilmek istenirse eğer, yine 3.5 sürümü ile birlikte gelen **WcfTestClient** aracı kullanılabilir. Tabi, servisin **IIS** veya **Asp.Net Development**

Server üzerinden host ediliyor olması gerekmektedir. Aşağıdaki ekran görüntüsünde servisin ilgili GetProduct metodunun test edilişi görülmektedir.



Görüldüğü üzere istemci ve servis aralarında, **SOAP zarflarını(SOAP Envelope)** kullanarak haberleşmektedir. Eğer **Fiddler** gibi bir aracı kullanırsanız bu durumda **HTTP Post**metoduna göre bir talepte bulunulduğunu görebilirsiniz. Diğer tarafan istemcinin servisi kullanabilmesi için proxy tipinede ihtiyacı vardır. Ancak REST modeli ele alındığında ve HTTP Get ile talepte bulunulduğunda arada buna gerek yoktur ki buda her iki model arasındaki ikinci önemli fark olarak görülebilir. Şu anda yapmak istediğimiz söz konusu servisi **REST** modeline taşımaktadır.

İlk olarak projeye **System.ServiceMode.Web** assembly' ının referans edilmesi gerekmektedir. İkinci adımda ise **HTTP Get** metodu ile erişilmesi istenen operasyonların **WebGet niteliği(attribute)** ile imzalanması yeterli olacaktır.

```
using System.ServiceModel;
using System.ServiceModel.Web;
```

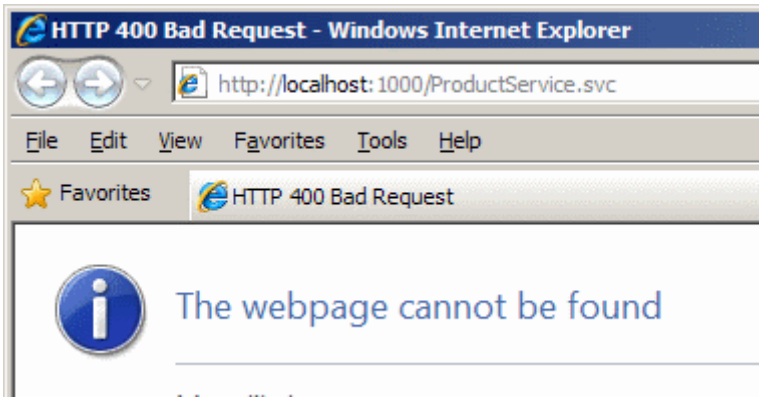
```
namespace NorthwindServices
{
   [ServiceContract]
   public interface IProductService
   {
      [OperationContract]
      [WebGet]
      Product GetProduct(int id);
   }
}
```

Bu işlem ile GetProduct operasyonunun **HTTP Get** metoduna göre çağırlabileceği bildirilmektedir. Peki kime?

Bir nitelik söz konusu ise eğer, bunu çalışma zamanında ele alan bir yapının olması gerekmektedir. Bu senaryoda, WCF çalışma zamanı ortamının bu niteliği göz önüne alarak, operasyona gelecek olan çağrılarda HTTP Get metodunu değerlendiriyor olması gerekir. Var olan **ServiceHost** fabrikası bu işlemi yapmamaktadır. Bu sebepten servise ait Markup kodları içerisinde aşağıdaki değişikliğin yapılması üçüncü adımdır.

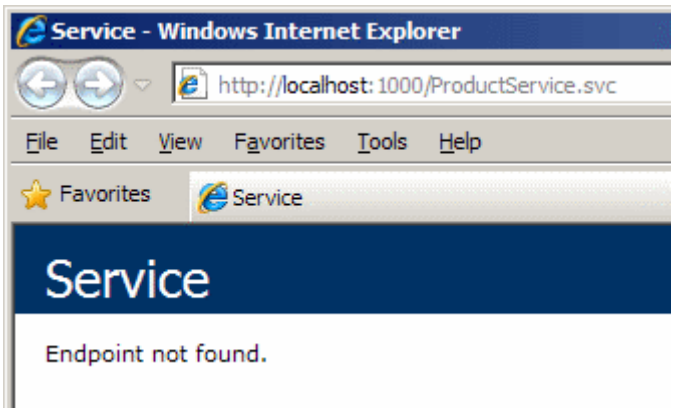
```
<%@ ServiceHost Language="C#" Debug="true"
Service="NorthwindServices.ProductService"
CodeBehind="ProductService.svc.cs" Factory="System.ServiceModel.Activation.WebServiceHostFactory" %>
```

Buradaki en önemli değişiklik **WebServiceHostFactory** fabrika tipinin kullnılıyor olmasıdır. Dolayısıyla artık ProductService.svc isimli hizmete gelecek olan HTTP Get talepleri karşılanabilecektir. Buna göre üçüncü adımdan sonra servis bir tarayıcı uygulama yardımıyla test edilebilir. Ancak bu durumda aşağıdaki görüntü ile karşılaşılması muhtemeldir.



Bu son derece doğaldır. Nitekim halen **web.config** dosyası içerisinde service ile ilişkili ayarlar SOAP modeline göre durmaktadır.(**BasicHttpBinding** kullandığımız hatırlayalım) İki alternatif vardır. web.config içeriğinde yer alan servis ilişkili bildirimler tamamen kaldırılabilir. Yada **WebHttpBinding** bağlayıcı tipinin kullanılması tercih

edilebilir. Ben web.config içerisindeki**system.ServiceModel** içeriğini tamamen kaldırmayı tercih ettim. Buda bizim 4ncü adımımız oldu. Peki bu adımdan sonra servisi tekrar denersek...



Sonuç yine hüsran.

Aslında problem servisten nasıl istekte bulunacağımızı bilmiyor oluşum.
Nitekim **WebGet** niteliği ile imzalanmış olan GetProduct operasyonuna HTTP Get modeline göre parametre ile birlikte talepte bulunmamız gerekiyor. Aynen aşağıdaki şekilde olduğu gibi.

Dikkat edileceği üzere **/GetProduct?id=1001** ile gönderilen talep, başarılı bir şekilde servis tarafından ele alınmış ve geriye, Product nesne örneği için oluşturulan **XML** içeriği döndürülmüştür.

REST modeli öylesine tutuldu ki **WCF 4.0** içerisinde zaten gömülü olarak daha fazla eklenti ile birlikte gelecek. *(Ado.Net Data Service' leri bu modelin güzel bir açılımı olarak görebiliriz aslında)* Bu eklentileri bir süre önce **CodePlex**' te yayımlanan WCF Rest Starter Kit yardımıyla **Visual Studio 2008** ortamı üzerinde test etmemiz ve geliştirmemizde mümkün. Hatta geçtiğimiz Mart ayında bu geliştirme kitinin 2nci versiyonuda yayınladı. REST modeli ile ilişkili olaraktan yeni yazılar eklemeye devam

ediyor olacağım. Hatta Rest modeli ile ilişkili iki görsel dersimede aşağıdaki linklerden ulaşabilirsiniz.

Atom Feed Service C#Nedir? | NedirTv?

Read-Only Collection Service C#Nedir? | NedirTv?

NorthwindServices.rar (16,34 kb)

Artık dinlenmeye çekilebilir miyim acaba ? 😎

WCF Servisleri için Unit Test (2009-04-17T16:28:00)

wcf,unit test,

Merhaba Arkadaşlar,

Yazdığımız programların belirli kriterlere göre test edilmesi proje süreçleri içerisinde önem arz eden konulardan birisidir. Bilindiği üzere pek çok test çeşidi vardır. Bunların bir kısmı standart haline gelmiş tekniklerden oluşmaktadır. örneğin web uygulamalarının beliri bir düzenle çalıştırılaran Request' ler ile test edilmesi. Stres testlerine tabi tutularaktan çok sayıda request sonrası web uygulamasının çalışmasının analiz edilmesi veya en basit anlamda bir programın çalışmasının ana parçalarından olan metodlarının, beklenen sonuçları verip vermediğinin araştırılması vb...

Nesne yönelimli programlama(Object Oriented Programming) dilleri ile geliştirilen uygulamalarda, özellikle metod bazında yapılan **birim testlerini(Unit Test)** **WCF** servislerine de uygulayabiliriz. Bu kısa yazımızda söz konusu işlemin **Visual Studio** ile birlikte gelen **Test Tool'** u yardımıyla nasıl gerçekleştirebileceğimizi incelemeye çalışacağız. Konu ile ilişkili olarak internette eğer araştırma yaparsanız, test amacıyla **NUnit** gibi araçlardan da faydalanabileceğimizi görebilirsiniz. Nitekim bazı durumlarda test yapan tarafta **Visual Studio** gibi bir araç olmayabilir ve NUnitaracı ile söz konusu analizler kolayca yapılabilir.

Şimdi elimizde kobay olarak kullanacağımız basit bir **WCF** servisi olduğunu düşünelim. Bu konu ile uğraşırken yaptığım araştırmalarda pek çok sitede, dört işlemin ele alındığına şahid oldum. Geleneği bozmadan devam edelim :) Servisimizi **Visual Studio 2008 Professional** ortamında geliştirebiliriz. Sözleşme(Contract) ve uygulayıcı tip içeriklerimiz aşağıdaki kodlarda görüldüğü gibidir.

IAlgebraService arayüzü(Interface)

```
using System.ServiceModel;
```

```
[ServiceContract]
public interface IAlgebraService
{
   [OperationContract]
   double Toplama(double x, double y);

   [OperationContract]
   double Cikarma(double x, double y);

   [OperationContract]
   double Carpma(double x, double y);

   [OperationContract]
   double Bolme(double x, double y);
}
```

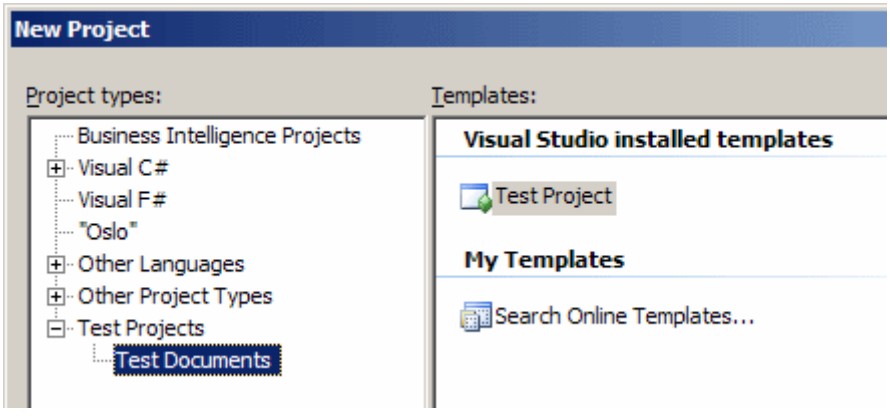
Uygulayıcı sınıf (AlgebraService.cs);

```
public class AlgebraService
   : IAlgebraService
{
   #region IAlgebraService Members

   public double Toplama(double x, double y)
   {
      return x + y;
   }

   public double Cikarma(double x, double y)
   {
      return x - y;
   }

   public double Carpma(double x, double y)
   {
      return x * y;
   }

   public double Bolme(double x, double y)
   {
      return x / y;
   }

   #endregion
}
```

Operasyonlarımızı AlgebraService.svc isimli servis üzerinden host ettiğimizi düşünecek olursak aşağıdaki **web.config** ayarlarını test amacıyla kullanabiliriz.

```
<system.serviceModel>
  <services>
    <service behaviorConfiguration="AlgebraServiceBehavior" name="AlgebraService">
      <endpoint address="" binding="basicHttpBinding" contract="IAlgebraService">
        <identity>
          <dns value="localhost" />
        </identity>
      </endpoint>
      <endpoint address="mex" binding="mexHttpBinding" contract="IMetadataExchange" />
    </service>
  </services>
  <behaviors>
    <serviceBehaviors>
      <behavior name="AlgebraServiceBehavior">
        <serviceMetadata httpGetEnabled="true" />
        <serviceDebug includeExceptionDetailInFaults="false" />
      </behavior>
    </serviceBehaviors>
  </behaviors>
</system.serviceModel>
</configuration>
```

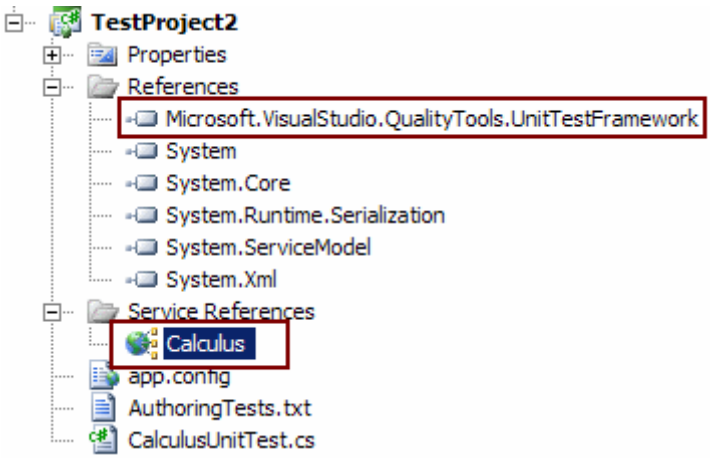
Görüldüğü gibi çok basit olarak **BasicHttpBinding** bağlayıcı tipini kullanan dolayısıyla **HTTP** bazlı hizmet veren bir servisimiz var. Böyle bir servis geliştirildiğinde developer' ların test etmek amacıyla ilk yapacağı iş servisin tarayıcı üzerinden erişilebilirliğidir. Ki bu sayede geliştirici kendini biraz iyi hisseder :) Ancak bizim amacımız servis metodlarının birim olarak test edilmesidir. Bu amaçla Solution içerisinde yeni bir **Test Project** açmak ilk adımımız olacaktır. (Test Project şablon olarak Unit Test için gerekli nitelikleri içeren assembly' ın referansını otomatik olarak içerir. Dolayısıyla işimiz kolaylaştıran bir şablondur.)



Test projesi içerisinde önemli olan noktalardan birisi, test edilmek istenen metodları kapsülleyen tipin bir servis sınıfı olmasıdır. Bir başka deyişle, test projesi söz konusu metodları deneyecekse eğer, AlgebraService isimli servise ulaşabiliyor ve metodlarını

çağırabiliyor olmalıdır. Bu açıdan bakıldığında yapılan testin gerçekten birim testi olduğu açıktır.

İlerlemek için servisin, test projesine referans edilmesi yeterlidir.



Şekildende görüleceği üzere amacıma ancak ikinci test projesinde ulaşabilmiş durumdayım :) Dikkat edilmesi gereken bir hususda proje referanslarında**Microsoft.VisualStudio.QualityTools.UnitTestFramework assembly**' ının var olmasıdır. Bu assembly içerisinde yer alan **nitelikleri(attributes)** birim testini yapacak sınıfı yazarken kullanıyor olacağız. Tahmin edileceği üzere bir **Test Project** oluşturulduğunda Wizard yardımıyla hızlı ve kolay bir şekilde ilerlenebilir. Ben örneğimizde Wizard kullanmadan ilerlemeyi denedim ve aşağıdaki test sınıfını oluşturdum.

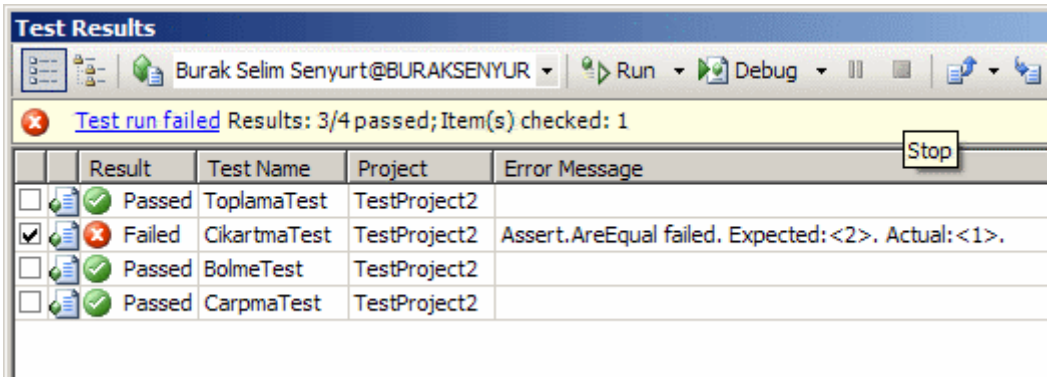
```
using Microsoft.VisualStudio.TestTools.UnitTesting;
using TestProject2.Calculus;

namespace TestProject2
{
   [TestClass]
   public class CalculusUnitTest
   {
      private AlgebraServiceClient client = null;

      [TestInitialize]
      public void Baslat()
      {
         client = new AlgebraServiceClient();
      }

      [TestCleanup]
      public void Bitir()
      {
```

```
            if(client.State== System.ServiceModel.CommunicationState.Opened)
                client.Close();
        }

        [TestMethod]
        [Description("Toplama testi")]
        public void ToplamaTest()
        {
            Assert.AreEqual(3, client.Toplama(2, 1));
        }

        [TestMethod]
        public void CikartmaTest()
        {
            Assert.AreEqual(2, client.Cikarma(3, 2));
        }

        [TestMethod]
        public void BolmeTest()
        {
            Assert.AreEqual(5, client.Bolme(25, 5));
        }

        [TestMethod]
        public void CarpmaTest()
        {
            Assert.AreEqual(8, client.Carpma(2, 4));
        }
    }
}
```

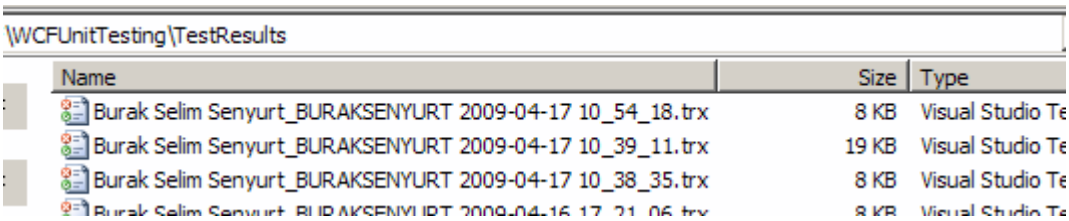
İşte yazımızın en önemli kısmı bu sınıftır. Dikkat edileceği üzere sınıfımız **TestClass** isimli bir nitelikle imzalanmıştır. Diğer taraftan **TestInitialize, TestCleanup, TestMethod** isimli nitelikler yardımıyla imzalanmış olan metodlarda vardır. Nitelik kullanılması, çalışma zamanında yer alan bir ortamın, bir takım hazırlıklar yapacağı anlamına gelmektedir. özetle bu nitelikler Visual Studio ortamında yer alan Test araçları tarafından çalışma zamanında değerlendirilmektedir. **TestInitialize** niteliği ile imzalanan metodlarda, test başlamadan önce yapılması gereken ön hazırlıklar yer almaktadır. Söz gelimi birim testlerine tabi olacak metodları içeren servis nesnesinin örneklenmesi gibi. **TestCleanup** niteliği ilede, testlerin bitmesi ile işletilmesi gereken kodları içeren metod imzalanmaktadır. örneğin birim testleri için kullanılan servis örneğinin kapatılması burada yapılacak işlemlerden birisi olarak düşünülebilir.

TestMethod niteliği ile imzalanmış olan metod içerikleri ise, **VS** ortamındaki test aracı tarafından ele alınacak ve testleri yapılacak içeriklere sahiptir. Metod içeriklerine dikkat

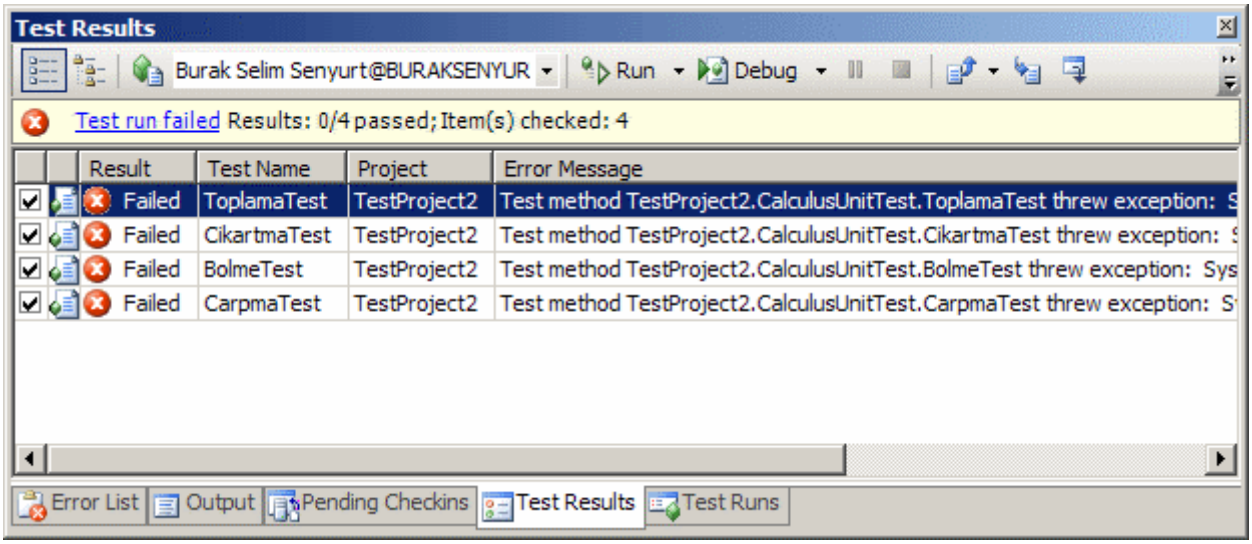
edilirse **Assert**tipine ait **AreEqual** fonksiyonlarının kullanıldığı görülmektedir. İlk parametre ile test sonucu beklenilen değer yazılırken, ikinci parametre ilede test edilecek fonksiyon çağrısı gerçekleştirilir. Cikarma metodunu test ettiğimiz yerde 2 sonucu beklenirken biz Cikarma metoduna (ki Cikartma olarak yazsam dilimize daha uygun olurmuş, hata yapmışım) 3 ile 2 parametrelerini göndermekteyiz. Yani aslında sonucun 1 olması gerekiyor. Bunu bilinçli olarak eklediğimi ve hata sonucu test aracının nasıl davrandığını görmek istediğimi belirteyim. Artık projemiz hazır. **Build** ettikten sonra çalıştırırsak, Visual Studio ortamında aşağıdaki görüntü ile karşılaşma ihtimalimiz yüksektir.



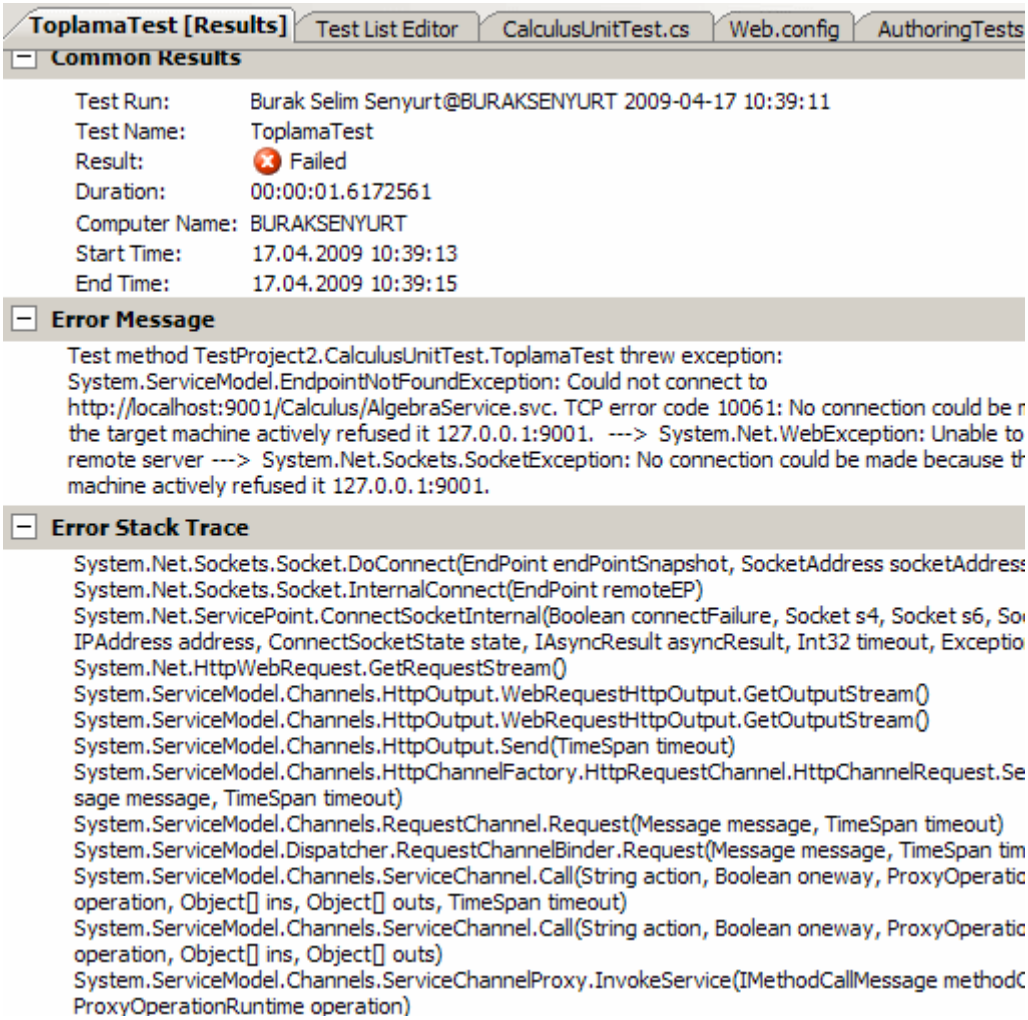
Görüldüğü gibi CikartmaTest isimli metod haricindeki tüm testlerde beklenen sonuçlara ulaşılmıştır. Yapılan testlere ait sonuçlar aynı zamanda **trx** uzantılı dosyalarda klasör bazlı olaraktan saklanmaktadır. Aşağıdaki ekran görüntüsünde olduğu gibi.



Not : *Eğer servisi host eden uygulama (IIS, Windows Servisi veya diğer host seçeneklerinde yer alan uygulama çeşitleri olabilir) çalışmıyorsa, çok doğal olarak test gerçekleşmeyecek ve aşağıdaki ekran görüntüsü ile karşılaşılacaktır.*

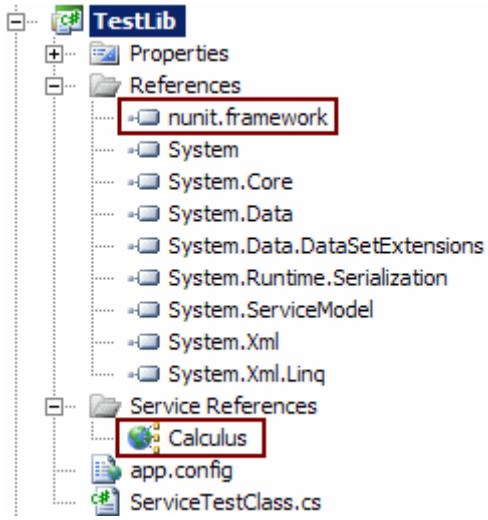


Burada **.Net Remoting'** den bu yana bildiğimiz meşhur hata mesajını alırız. **"No connection could be made because the target machine actively refused it"** 😎 Hatta hata mesajlarından birisine çift tıklarsak aşağıdaki detaylı bilgi penceresinede ulaşma şansımız bulunmaktadır.



Peki Visual Studio yerine **NUnit** aracını kullanarak birim testi gerçekleştirmek istersek...

İlk olarak **NUnit** programının sistemimizde kurulu olduğunu göz önüne alıyoruz. Sonrasında ise aynen bir önceki projemizde olduğu gibi servis referansını eklememiz gerekiyor. Bunlara ek olarak, **NUnit** test programının çalışma ortamına bilgi vermek amacıyla kullanacağımız nitelikleri için, projeye **nunit framwork'** ununde (blog girişini yaptığım tarihlerde varsayılan olarak **C:\Program Files\NUnit 2.5\bin\net-2.0\framework\nunit.framework.dll** klasöründe yer almaktadır) referans edilmesi gerekir.

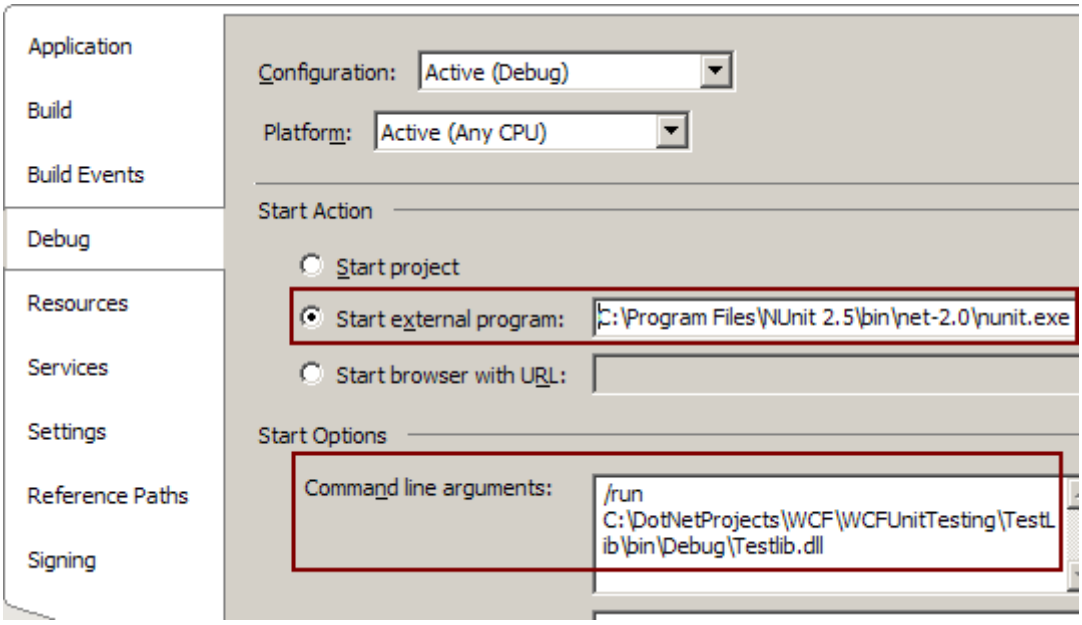


ServiceTestClass isimli sınıfın içeriği ise aşağıdaki gibidir.

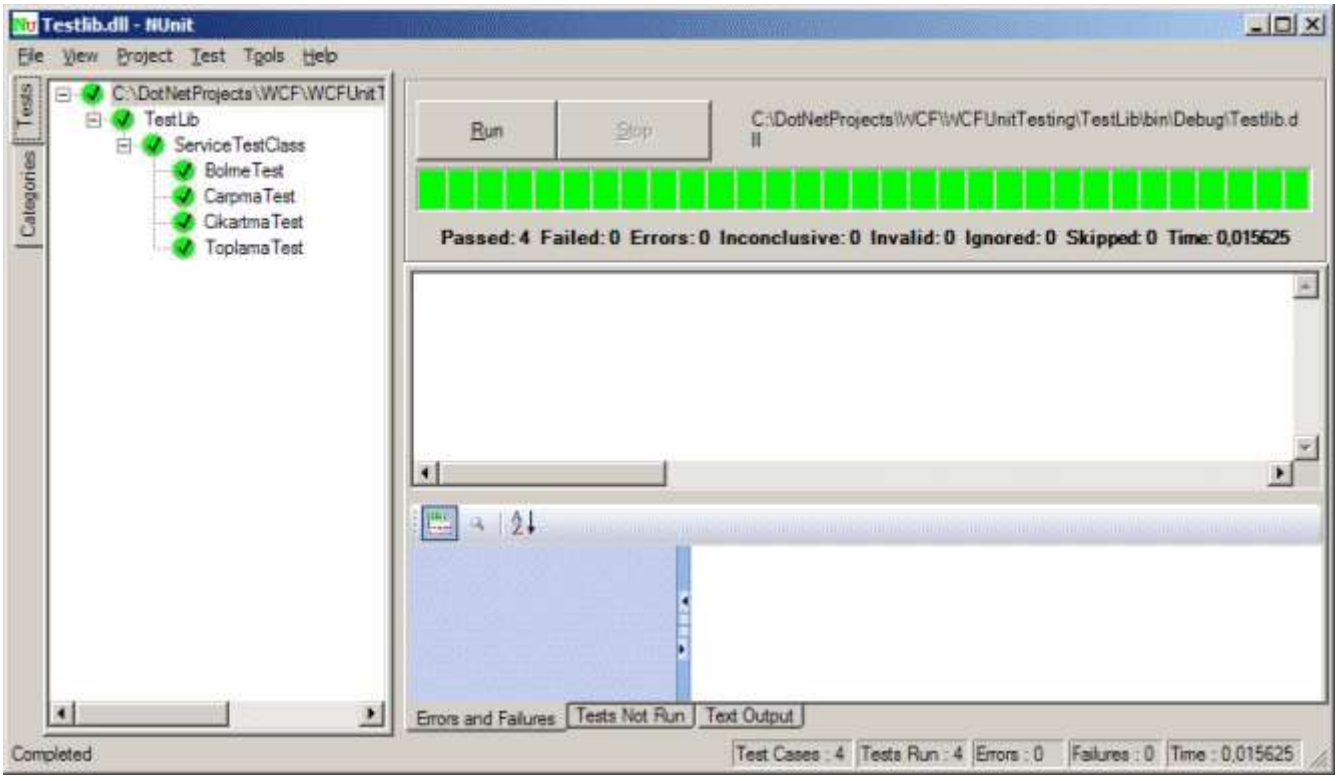
```
using TestLib.Calculus;
using NUnit.Framework;

namespace TestLib
{
   [TestFixture]
   public class ServiceTestClass
   {
      private AlgebraServiceClient client=null;

      [TestFixtureSetUp]
      public void Ornekle()
      {
         client = new AlgebraServiceClient();
         client.Open();
      }

      [TestFixtureTearDown]
      public void Kapat()
      {
         if(client.State== System.ServiceModel.CommunicationState.Opened)
```

```
            client.Close();
        }

        [Test]
        public void ToplamaTest()
        {
            Assert.AreEqual(5, client.Toplama(2, 3));
        }

        [Test]
        public void CikartmaTest()
        {
            Assert.AreEqual(7, client.Cikarma(10, 3));
        }

        [Test]
        public void BolmeTest()
        {
            Assert.AreEqual(3, client.Bolme(9, 3));
        }

        [Test]
        public void CarpmaTest()
        {
            Assert.AreEqual(6, client.Carpma(2, 3));
        }
    }
}
```

İlk projemizdekine benzer olaraktan buradada çalışma zamanını ilgilendiren bir takım nitelikler bulunmaktadır. Teste tabi olacak metodlarımız için **Test** niteliği, testin başlamasından önce yapılması istenen ön hazırlıklar için **TestFixtureSetUp**, test sonrasında yapılması istenen işlemler içinse **TestFixtureTearDown** nitelikleri ele alınır. Yine beklenen değerleri tespit etmek amacıyla **Assert** tipinin **AreEqual** metodundan yararlanılmaktadır. Projenin çalıştırılması halinde genellikle **NUnit** aracının otomatik olarak başlatıldığına şahit oldum. Bu nedenle projenin özelliklerinde aşağıdaki ayarlamayı yapmamız yeterli olacaktır.



Görüldüğü gibi bu dll çalıştırıldığında **NUnit tool'** unuda başlatıyoruz. **Command line arguments** kısmında ise **/run [dll adı]** parametrelerini vererekten, **NUnit** aracının**[dll adı]** isimli **assembly'** daki testleri başlatmasını belirtiyoruz. Artık tek yapmamız gereken servise ait host uygulamanın çalıştığından emin olmak. Sonrasında **TestLib.dll** isimli assembly çalışıtırılırsa aşağıdaki ekran görüntüsünde olduğu gibi NUnit aracının çalıştığını ve testlerin yapıldığını görebiliriz.



İşte bu kadar. Hemen son bir noktayı daha aklıma gelmişken belirteyim. **Visual Studio Test Tool'** unu kullandığımız senaryoda istenirse **debug** işlemleride yapılabilir.

Umuyorumki yararlı bir yazı olmuştur. Tekrardan görüşmek dileğiyle...

WCFUnitTesting.rar (725,67 kb)

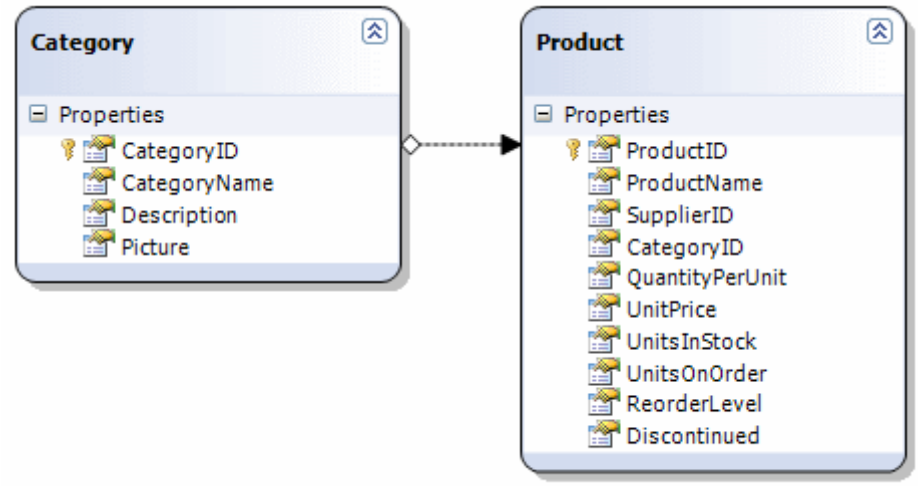
Ado.Net Entity Framework' de Lazy ve Eager Loading (2009-04-16T15:41:00)

ado.net entity framework,linq to sql,

Merhaba Arkadaşlar,

Bildiğiniz üzere uzun bir süre önce Microsoft, **LINQ to SQL** yerine **Ado.Net Entity Framework** ile ilerleme kararı aldı. Bu konu ile ilişkili olaraktan okuduğum hemem hemen bütün kitaplarda**Ado.Net**' in geleceğinde önemli bir yere sahip olan **Ado.Net Entity Framework** alt yapısının geliştiriciler tarafından asla ihmal edilmemesi gerektiğide sıkça vurgulanmakta. Peki günlüğüme konu olan mesele nedir?

Aslında günlüğe yazmadan önce odaklandığım nokta, aralarında **master-detail** ilişki bulunan tablo verilerinin, **LINQ to SQL** tarafında nasıl yüklendiğinin incelenmesiydi. Dolayısıyla önce **LINQ to SQL** tarafındaki duruma bir göz atmakya yarar var. İşe öncelikle basit bir **Console** uygulamasında kobay nesnelerimizden **Northwind** veritabanını kullanarak başlayabiliriz. Burada**Visual Studio 2008** üzerinde ve **.Net Framework 3.5** tabanlı bir geliştirme yaptığımızı belirtelim. Söz konusu uygulamamızda kullanacağımız **LINQ to SQL** diagramının içeriği aşağıdaki gibi tasarlanabilir.

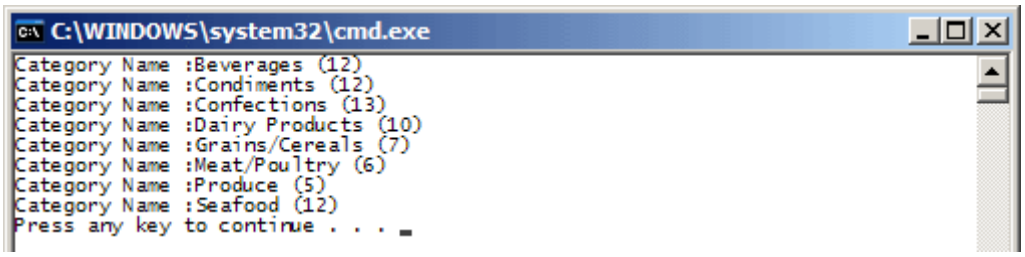


örnekte **Category** ve **Product** tablolarına ait tipleri göz önüne almaktayız. Buna göre **"Kategoriler ve bu kategorilerdeki toplam ürün sayılarını öğrenmek"** gibi basit bir sonuç kümesi elde etmek istediğimizde aşağıdakine benzer program kodlarını ele alacağımız muhtemeldir.

```
using System;
using System.Collections.Generic;
```

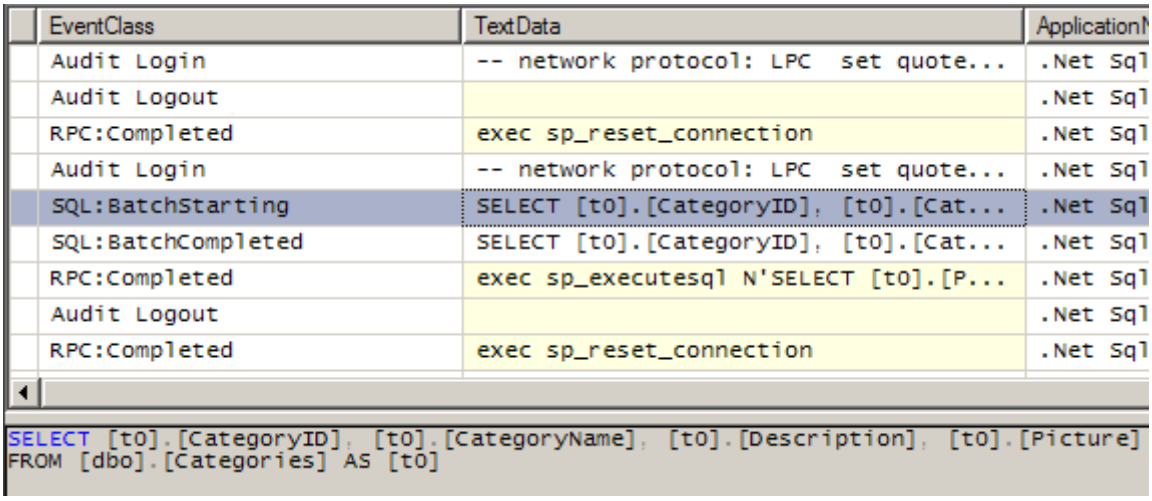
```
using System.Linq;
using System.Text;

namespace BlogSample
{
   class Program
   {
      static void Main(string[] args)
      {
         using (NorthwindDataContext northContext = new NorthwindDataContext())
         {
            // northContext.Log = Console.Out; // İsterseniz SQL sorgularını Console
çıktısından da takip edebilirsiniz.

            foreach (Category category in northContext.Categories)
            {
               Console.WriteLine("Category Name :{0} ({1})",
category.CategoryName, category.Products.Count);
            }
         }
      }
   }
}
```

Programın bu haliyele çıktısı aşağıdaki gibidir.



Aslında istediğimizi elde etmiş görünüyoruz.

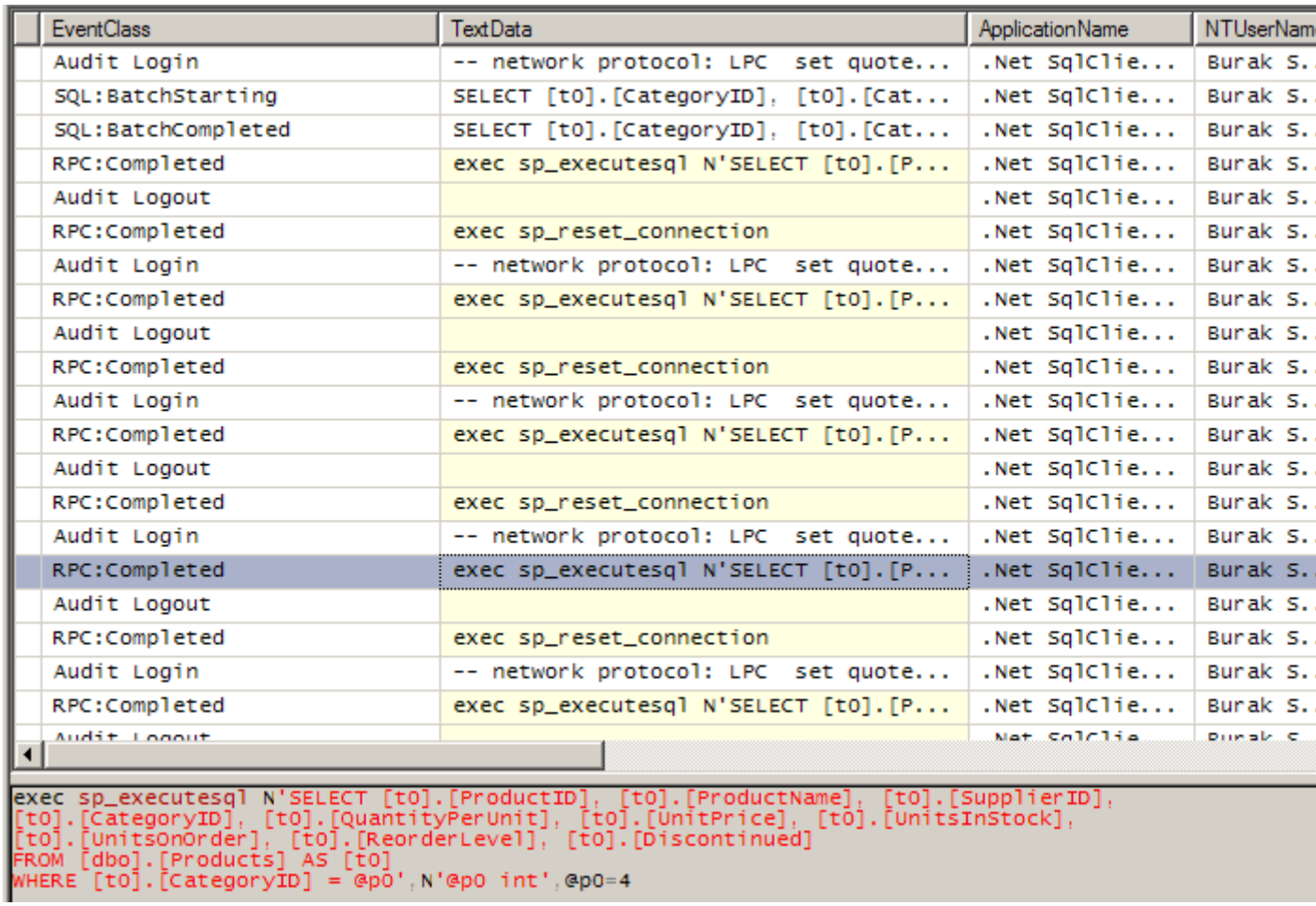
Gayet doğal olarak **foreach** döngüsü içerisinde bir kategoriye bağlı ürün sayıları bulunurken **Count** özelliğinden yararlanılmaktadır. Ancak bu durumda **Lazy Loading** adı verilen durum oluşmakta ve **SQL** tarafına bakıldığında oldukça fazla sorgunun çalıştığı görülmektedir. öyleki yukarıdaki kod çıktısı **SQL Profiler** yardımıyla incelendiğinde ilk sırada aşağıdaki sorgunun çalıştığı görülür.



EventClass	TextData	Application
Audit Login	-- network protocol: LPC set quote...	.Net Sql
Audit Logout		.Net Sql
RPC:Completed	exec sp_reset_connection	.Net Sql
Audit Login	-- network protocol: LPC set quote...	.Net Sql
SQL:BatchStarting	SELECT [t0].[CategoryID], [t0].[Cat...	.Net Sql
SQL:BatchCompleted	SELECT [t0].[CategoryID], [t0].[Cat...	.Net Sql
RPC:Completed	exec sp_executesql N'SELECT [t0].[P...	.Net Sql
Audit Logout		.Net Sql
RPC:Completed	exec sp_reset_connection	.Net Sql

```
SELECT [t0].[CategoryID], [t0].[CategoryName], [t0].[Description], [t0].[Picture]
FROM [dbo].[Categories] AS [t0]
```

Görüldüğü üzere ilk olarak tüm kategoriler **Select** sorgusu ile çekilmektedir. Ancak iş bundan sonra biraz daha dikkate değer bir hal alır. Nitekim her kategoriye ait ürün sayıları**Products** özelliği üzerinden elde edilmek istenmektedir. Bu durumda SQL tarafında her bir kategori satırı için birer sorgu cümlesi daha çalıştırılır.



EventClass	TextData	ApplicationName	NTUserNam
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S.
SQL:BatchStarting	SELECT [t0].[CategoryID], [t0].[Cat...	.Net SqlClie...	Burak S.
SQL:BatchCompleted	SELECT [t0].[CategoryID], [t0].[Cat...	.Net SqlClie...	Burak S.
RPC:Completed	exec sp_executesql N'SELECT [t0].[P...	.Net SqlClie...	Burak S.
Audit Logout		.Net SqlClie...	Burak S.
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	Burak S.
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S.
RPC:Completed	exec sp_executesql N'SELECT [t0].[P...	.Net SqlClie...	Burak S.
Audit Logout		.Net SqlClie...	Burak S.
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	Burak S.
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S.
RPC:Completed	exec sp_executesql N'SELECT [t0].[P...	.Net SqlClie...	Burak S.
Audit Logout		.Net SqlClie...	Burak S.
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	Burak S.
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S.
RPC:Completed	exec sp_executesql N'SELECT [t0].[P...	.Net SqlClie...	Burak S.
Audit Logout		.Net SqlClie...	Burak S.
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	Burak S.
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S.
RPC:Completed	exec sp_executesql N'SELECT [t0].[P...	.Net SqlClie...	Burak S.

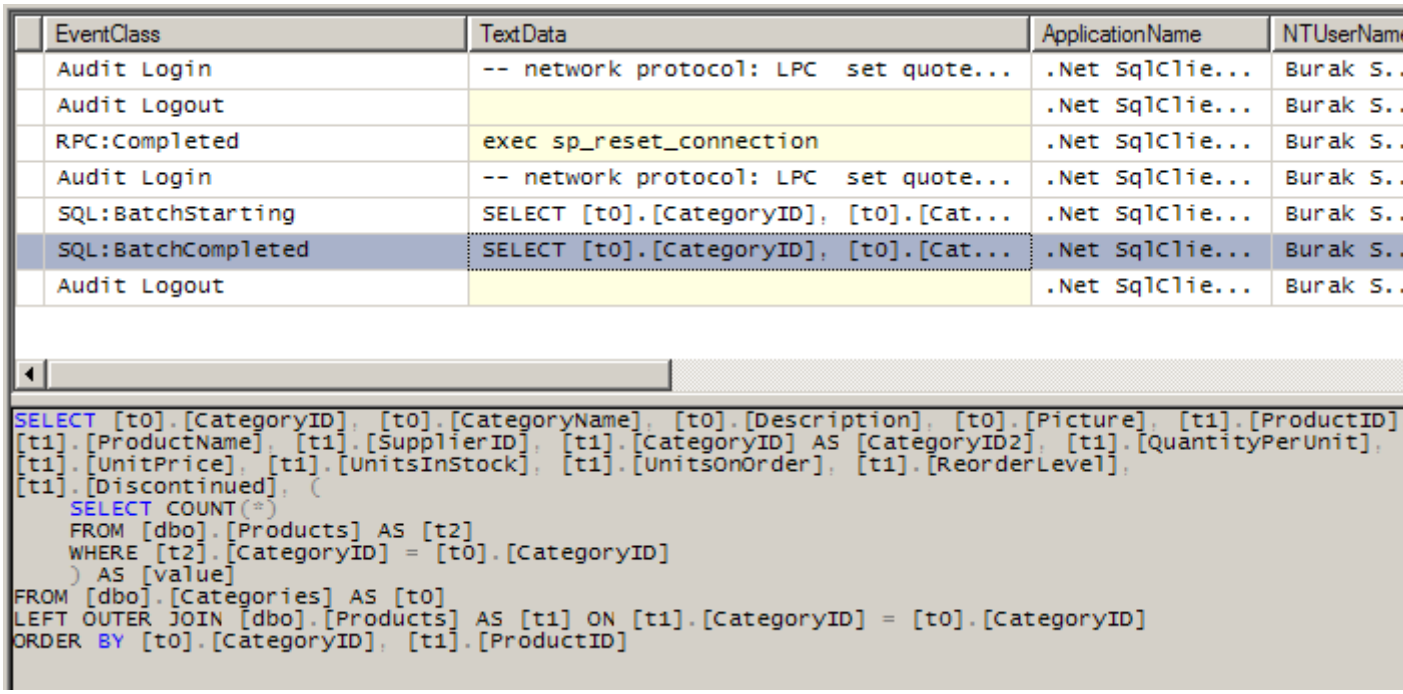
```
exec sp_executesql N'SELECT [t0].[ProductID], [t0].[ProductName], [t0].[SupplierID],
[t0].[CategoryID], [t0].[QuantityPerUnit], [t0].[UnitPrice], [t0].[UnitsInStock],
[t0].[UnitsOnOrder], [t0].[ReorderLevel], [t0].[Discontinued]
FROM [dbo].[Products] AS [t0]
WHERE [t0].[CategoryID] = @p0',N'@p0 int',@p0=4
```

Bu bir kaç satırlık veri için önemli gözükmese de, büyük boyutlu veriler ile çalışıldığı durumlarda önemli performans kayıplarına neden olabilir. Bu nedenle istenirse **Eager**

Loading isimli bir teknikten de yararlanılabilir. Tek yapılması gereken kod tarafına aşağıdaki değişiklikleri eklemektir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Linq;

namespace BlogSample
{
   class Program
   {
      static void Main(string[] args)
      {
         using (NorthwindDataContext northContext = new NorthwindDataContext())
         {
            // northContext.Log = Console.Out; // İsterseniz SQL sorgularını Console çıktısından da takip edebilirsiniz.
            DataLoadOptions loadOption = new DataLoadOptions();
            loadOption.LoadWith<Category>(c => c.Products);
            northContext.LoadOptions = loadOption;

            foreach (Category category in northContext.Categories)
            {
               Console.WriteLine("Category Name :{0} ({1})", category.CategoryName, category.Products.Count);
            }
         }
      }
   }
}
```

Bu durumda da kod aynı sonuçları verir ve SQL tarafındaki çıktıya bakıldığına tek bir sorgunun çalıştırıldığı gözlemlenebilir.



EventClass	TextData	ApplicationName	NTUserNam
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S..
Audit Logout		.Net SqlClie...	Burak S..
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	Burak S..
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S..
SQL:BatchStarting	SELECT [t0].[CategoryID], [t0].[Cat...	.Net SqlClie...	Burak S..
SQL:BatchCompleted	SELECT [t0].[CategoryID], [t0].[Cat...	.Net SqlClie...	Burak S..
Audit Logout		.Net SqlClie...	Burak S..

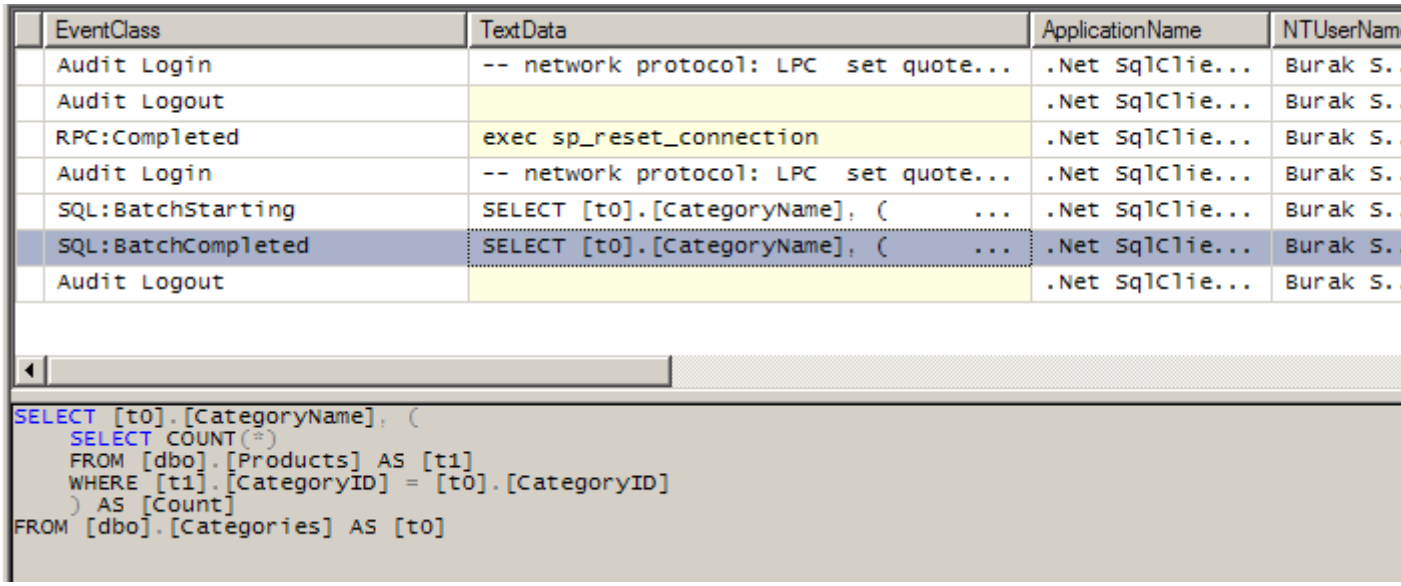
```
SELECT [t0].[CategoryID], [t0].[CategoryName], [t0].[Description], [t0].[Picture], [t1].[ProductID],
[t1].[ProductName], [t1].[SupplierID], [t1].[CategoryID] AS [CategoryID2], [t1].[QuantityPerUnit],
[t1].[UnitPrice], [t1].[UnitsInStock], [t1].[UnitsOnOrder], [t1].[ReorderLevel],
[t1].[Discontinued], (
    SELECT COUNT(*)
    FROM [dbo].[Products] AS [t2]
    WHERE [t2].[CategoryID] = [t0].[CategoryID]
    ) AS [value]
FROM [dbo].[Categories] AS [t0]
LEFT OUTER JOIN [dbo].[Products] AS [t1] ON [t1].[CategoryID] = [t0].[CategoryID]
ORDER BY [t0].[CategoryID], [t1].[ProductID]
```

Ancak en iyi performans için bu teknikler yerine **Anonymous Type** kullanılması çok daha doğrudur. öyleki sorguda istenen sadece kategori adları ve o kategoriye bağlı ürünlerin toplam sayılarıdır. Oysa ki sorgu cümlelerine bakıldığında o anda gerekli olmayan tablo alanlarının da hesaba katıldığı görülmektedir. Dolayısıyla sorgunun sadece, **CategoryName** ve buna bağlı **Product** satırlarının toplam sayısını bulacak şekilde iyileştirilebilmesi gerekmektedir. Aslında burada **LINQ** sorgularının **defered execution** adı verilen **"gerektiği yerde sorguyu gönder"** sistemide önemlidir. Kısaca LINQ sorgusunun yazıldığı satırda değilde, ilk kullanıldığı yerde SQL cümlesinin gönderilmesinden bahsediyoruz. Lafı fazla uzatmadan kod tarafındaki değişikliklerimize bakalım.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Data.Linq;

namespace BlogSample
{
   class Program
   {
      static void Main(string[] args)
      {
         using (NorthwindDataContext northContext = new NorthwindDataContext())
         {
            var resultSet = from c in northContext.Categories
                            select new
                            {
```

```
                        c.CategoryName,
                        c.Products.Count
                    };
            foreach (var result in resultSet)
            {
                Console.WriteLine("Category Name :{0} ({1})",
result.CategoryName,result.Count.ToString());
            }
        }
    }
}
```

Bu durumda **SQL** tarafına giden sorgunun aşağıdaki gibi olduğu görülebilir.



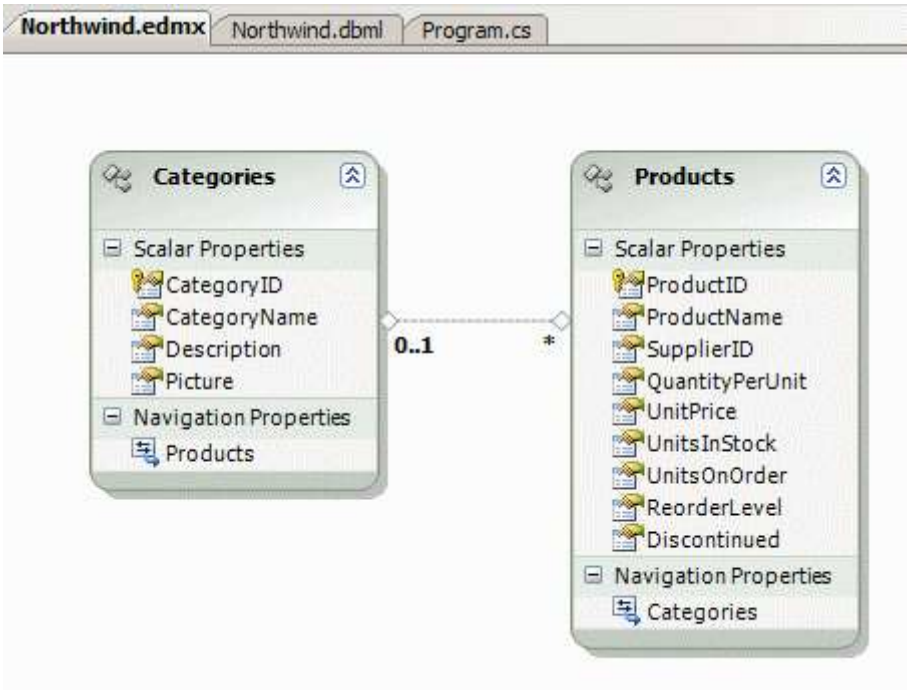
EventClass	TextData	ApplicationName	NTUserNam
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S.
Audit Logout		.Net SqlClie...	Burak S.
RPC:Completed	exec sp_reset_connection	.Net SqlClie...	Burak S.
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S.
SQL:BatchStarting	SELECT [t0].[CategoryName], (...	.Net SqlClie...	Burak S.
SQL:BatchCompleted	SELECT [t0].[CategoryName], (...	.Net SqlClie...	Burak S.
Audit Logout		.Net SqlClie...	Burak S.

```
SELECT [t0].[CategoryName], (
    SELECT COUNT(*)
    FROM [dbo].[Products] AS [t1]
    WHERE [t1].[CategoryID] = [t0].[CategoryID]
    ) AS [Count]
FROM [dbo].[Categories] AS [t0]
```

Dikkat edileceği üzere sadece istediğimiz alanlar sorguya dahil edilmiştir.

Derken rüyadan uyanırım ve birden aklıma artık **Ado.Net Entity Framework** konulu bir yazı yazacağım gelir.

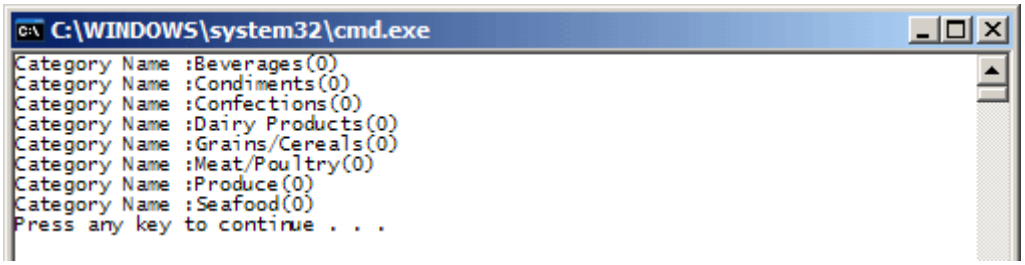
Bakalım o tararfa **Lazy Loading, Eager Loading** durumları nasıl ele alınabilir. Bu kez projede kullanacağımız tiplerin **EDM(Entity Data Model)** diagramındaki görüntüsü aşağıdaki gibidir. Aynen yukarıda örneklerde olduğu gibi **Category** ve **Product** tablolarını ele alıyor olacağız.



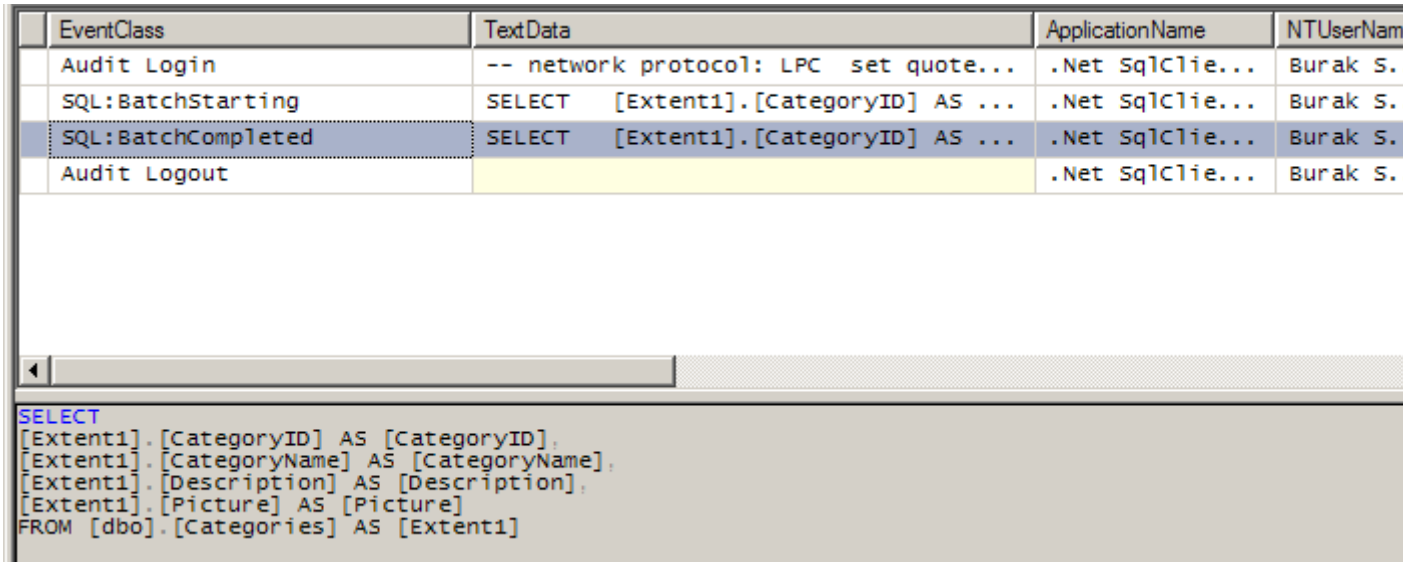
Şimdi ilk kod parçamızı geliştirelim.

```
using (NorthwindEntities entity = new NorthwindEntities())
        {
           foreach (var category in entity.Categories)
           {
              Console.WriteLine("Category Name
:{0}({1})",category.CategoryName,category.Products.Count);
           }
        }
```

SQL Profiler'a geçmeden önce uygulama çalıştırıldığında aşağıdaki sonuç ile karşılaşılır.



Dikkatli gözlerden, ürün sayılarının **0** olarak geldiği kaçmayacaktır. **SQL Profiler** aracı ile gönderilen sorguya bakıldığındaysa sadece kategorilerin çekildiği ancak ürün sayılarının bulunması ile ilişkili bir şey yapılmadığı gözlemlenebilir.



EventClass	TextData	ApplicationName	NTUserNam
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S.
SQL:BatchStarting	SELECT [Extent1].[CategoryID] AS ...	.Net SqlClie...	Burak S.
SQL:BatchCompleted	SELECT [Extent1].[CategoryID] AS ...	.Net SqlClie...	Burak S.
Audit Logout		.Net SqlClie...	Burak S.

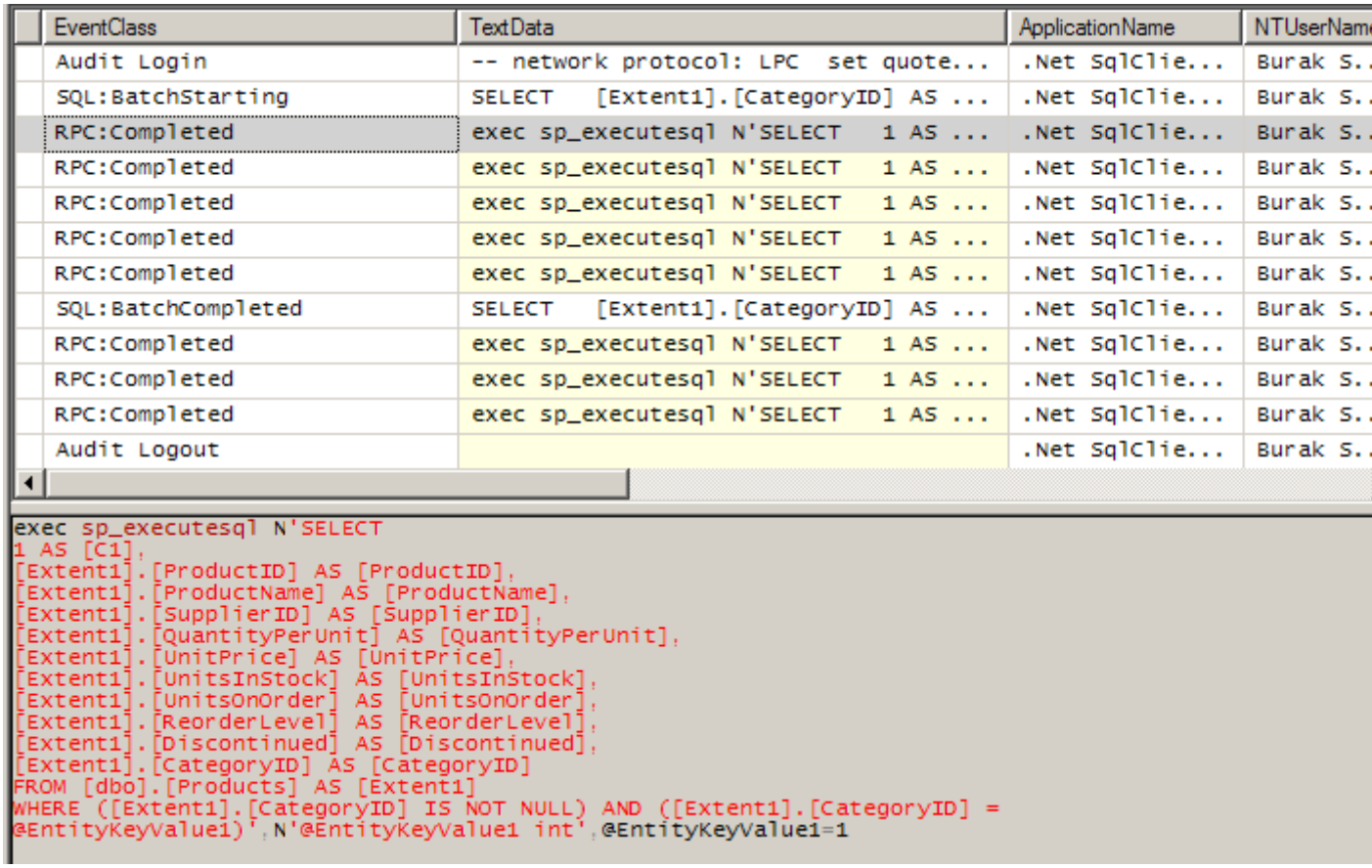
```
SELECT
[Extent1].[CategoryID] AS [CategoryID],
[Extent1].[CategoryName] AS [CategoryName],
[Extent1].[Description] AS [Description],
[Extent1].[Picture] AS [Picture]
FROM [dbo].[Categories] AS [Extent1]
```

Aslında bu son derece doğaldır nitekim **Ado.Net Entity Framework** modelinde **Lazy Loading'** in bilinçli ve açık bir şekilde yapılması istenmektedir. Dolayısıyla geliştiricinin gerçekten**Lazy Loading** yapmak istediğini kod tarafında belirtmesi gerekir. Peki bu nasıl yapılır? Aşağıdaki basit kod parçasında olduğu gibi...

```
using (NorthwindEntities entity = new NorthwindEntities())
        {
           foreach (var category in entity.Categories)
           {
              if(!category.Products.IsLoaded)
                 category.Products.Load();
              Console.WriteLine("Category Name
:{0}({1})",category.CategoryName,category.Products.Count);
           }
        }
```

SQL Profiler' a bakıldığında aşağıdaki çıktı ile karşılaşılır.

İlk olarak kateogoriler çekilmiş sonrasında ilk 5 kategori için sırasıyla ürünlere ait sorgular çalıştırılmıştır. Ardından ise Category tablosu için yine bir Select çalıştırılmakta ve kalan 3 kategorinin her biri için tekrardan ürün sorguları yürütülmektedir.(Bu yazıda anlattıklarımı bire bir uygulamanızı, **SQL Profiler** aracaı yardımıylada irdelemeye çalışmanızı şiddetle öneririm.)



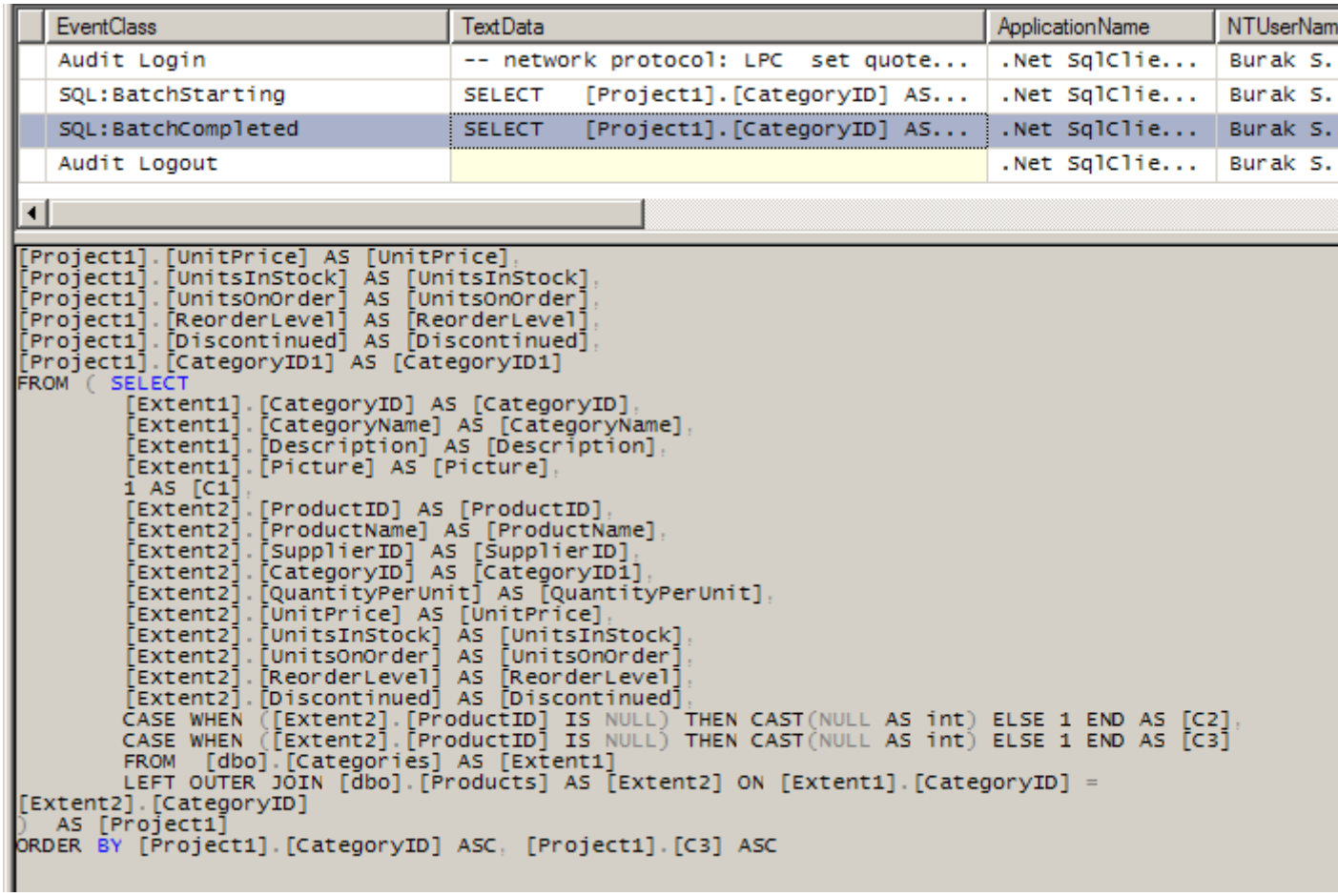
EventClass	TextData	ApplicationName	NTUserNam
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S..
SQL:BatchStarting	SELECT [Extent1].[CategoryID] AS ...	.Net SqlClie...	Burak S..
RPC:Completed	exec sp_executesql N'SELECT 1 AS ...	.Net SqlClie...	Burak S..
RPC:Completed	exec sp_executesql N'SELECT 1 AS ...	.Net SqlClie...	Burak S..
RPC:Completed	exec sp_executesql N'SELECT 1 AS ...	.Net SqlClie...	Burak S..
RPC:Completed	exec sp_executesql N'SELECT 1 AS ...	.Net SqlClie...	Burak S..
RPC:Completed	exec sp_executesql N'SELECT 1 AS ...	.Net SqlClie...	Burak S..
SQL:BatchCompleted	SELECT [Extent1].[CategoryID] AS ...	.Net SqlClie...	Burak S..
RPC:Completed	exec sp_executesql N'SELECT 1 AS ...	.Net SqlClie...	Burak S..
RPC:Completed	exec sp_executesql N'SELECT 1 AS ...	.Net SqlClie...	Burak S..
RPC:Completed	exec sp_executesql N'SELECT 1 AS ...	.Net SqlClie...	Burak S..
Audit Logout		.Net SqlClie...	Burak S..

```
exec sp_executesql N'SELECT
1 AS [C1],
[Extent1].[ProductID] AS [ProductID],
[Extent1].[ProductName] AS [ProductName],
[Extent1].[SupplierID] AS [SupplierID],
[Extent1].[QuantityPerUnit] AS [QuantityPerUnit],
[Extent1].[UnitPrice] AS [UnitPrice],
[Extent1].[UnitsInStock] AS [UnitsInStock],
[Extent1].[UnitsOnOrder] AS [UnitsOnOrder],
[Extent1].[ReorderLevel] AS [ReorderLevel],
[Extent1].[Discontinued] AS [Discontinued],
[Extent1].[CategoryID] AS [CategoryID]
FROM [dbo].[Products] AS [Extent1]
WHERE ([Extent1].[CategoryID] IS NOT NULL) AND ([Extent1].[CategoryID] =
@EntityKeyValue1)',N'@EntityKeyValue1 int',@EntityKeyValue1=1
```

Buradaki kod parçasında her bir kategori için buna bağlı ürünlerinde Products özelliği ile işaret edilen referansa yüklenmesi istendiği **Load** metodu yardımıyla belirtilmektedir. Bu kod ile**Lazy Loading** gerçekleştirilmiş olmaktadır. Peki ya **Eager Loading**? Eager Loading için aşağıdaki kod parçasını kullanmak yeterli olacaktır.

```
using (NorthwindEntities entity = new NorthwindEntities())
        {
           foreach (var category in entity.Categories.Include("Products"))
           {
              Console.WriteLine("Category Name
:{0}({1})",category.CategoryName,category.Products.Count);
           }
        }
```

Aslında tek yaptığımız **Categories** özelliği üzerinden **Include** metodunu çağırmak ve **Products** değerini vermek olmuştur. Buna göre SQL tarafında aşağıdaki sorgunun çalıştırıldığı görülür.



EventClass	TextData	ApplicationName	NTUserNam
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S.
SQL:BatchStarting	SELECT [Project1].[CategoryID] AS...	.Net SqlClie...	Burak S.
SQL:BatchCompleted	SELECT [Project1].[CategoryID] AS...	.Net SqlClie...	Burak S.
Audit Logout		.Net SqlClie...	Burak S.

```
[Project1].[UnitPrice] AS [UnitPrice],
[Project1].[UnitsInStock] AS [UnitsInStock],
[Project1].[UnitsOnOrder] AS [UnitsOnOrder],
[Project1].[ReorderLevel] AS [ReorderLevel],
[Project1].[Discontinued] AS [Discontinued],
[Project1].[CategoryID1] AS [CategoryID1]
FROM ( SELECT
        [Extent1].[CategoryID] AS [CategoryID],
        [Extent1].[CategoryName] AS [CategoryName],
        [Extent1].[Description] AS [Description],
        [Extent1].[Picture] AS [Picture],
        1 AS [C1],
        [Extent2].[ProductID] AS [ProductID],
        [Extent2].[ProductName] AS [ProductName],
        [Extent2].[SupplierID] AS [SupplierID],
        [Extent2].[CategoryID] AS [CategoryID1],
        [Extent2].[QuantityPerUnit] AS [QuantityPerUnit],
        [Extent2].[UnitPrice] AS [UnitPrice],
        [Extent2].[UnitsInStock] AS [UnitsInStock],
        [Extent2].[UnitsOnOrder] AS [UnitsOnOrder],
        [Extent2].[ReorderLevel] AS [ReorderLevel],
        [Extent2].[Discontinued] AS [Discontinued],
        CASE WHEN ([Extent2].[ProductID] IS NULL) THEN CAST(NULL AS int) ELSE 1 END AS [C2],
        CASE WHEN ([Extent2].[ProductID] IS NULL) THEN CAST(NULL AS int) ELSE 1 END AS [C3]
        FROM  [dbo].[Categories] AS [Extent1]
        LEFT OUTER JOIN [dbo].[Products] AS [Extent2] ON [Extent1].[CategoryID] =
[Extent2].[CategoryID]
)  AS [Project1]
ORDER BY [Project1].[CategoryID] ASC, [Project1].[C3] ASC
```

Elbette yine istediğimizi alamadık. Nitekim sadece kategori adı ve ürün sayılarını elde etmek gibi bir amacımız vardı. **LINQ to SQL** tarafında yapmış olduğumuz iyileştirmeyi**Ado.Net Entity Framework** tarafında da **Include** metodunu hesaba katarak ve yine **Anonymous Type** kullanarak gerçekleştirebiliriz.

```
using (NorthwindEntities entity = new NorthwindEntities())
        {
           var resultSet = from c in entity.Categories.Include("Products")
                           select new
                           {
                              c.CategoryName,
                              c.Products.Count
                           };
           foreach (var result in resultSet)
           {
              Console.WriteLine("Category Name :{0}({1})", result.CategoryName,
result.Count);
           }
        }
```

Sonuç olarak **SQL** tarafına aşağıdaki sorgu cümlesi gönderilecektir.

EventClass	TextData	ApplicationName	NTUserNam
Audit Login	-- network protocol: LPC set quote...	.Net SqlClie...	Burak S.
SQL:BatchStarting	SELECT 1 AS [C1], [Project1].[C...	.Net SqlClie...	Burak S.
SQL:BatchCompleted	SELECT 1 AS [C1], [Project1].[C...	.Net SqlClie...	Burak S.
Audit Logout		.Net SqlClie...	Burak S.

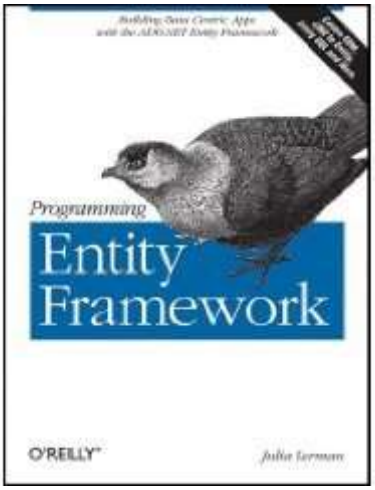
```
SELECT
1 AS [C1],
[Project1].[CategoryName] AS [CategoryName],
[Project1].[C1] AS [C2]
FROM ( SELECT
        [Extent1].[CategoryName] AS [CategoryName],
        (SELECT
                COUNT(cast(1 as bit)) AS [A1]
                FROM [dbo].[Products] AS [Extent2]
                WHERE [Extent1].[CategoryID] = [Extent2].[CategoryID]) AS [C1]
        FROM [dbo].[Categories] AS [Extent1]
)  AS [Project1]
```

Görüdüğü üzere sadece CategoryName ve sub query ile birlikte ürün sayıları hesaba katılmaktadır.

Böylece geldik bir günlük yazımızın daha sonuna. Bu yazımızda **LINQ to SQL** ve **Ado.Net Entity Framework** tarafında **Lazy** ve **Eager Loading** kavramlarını değerlendirmeye çalıştık. Umarım yararlı olmuştur.

Görüşmek dileğiyle...

Programming Entity Framework (2009-04-16T10:20:00)



çok yakın bir zamanda **O'Reilly** yayınlarından çıkmış olan 828 sayfalık bu kitap, **Ado.Net Entity Framework** ile ilişkili dolu dolu bilgiler içeriyor. **Amazon**' dan tedarik edebileceğiniz bu kitap özellikle yeni dönemde baş ucumuzda durması gereken kaynaklardan birisi.

Julie Lerman tarafından yazılmış(blog adresi) olan kitaba daha ilk bölümden itibaren bağlanmamak elde değil. öncelikle **EDM(Entity Data Model)** kavramı üzerinde güzel bir giriş yapıp, sorunlara ve nedenlere cevaplar veriliyor. Sonrasında ise **Microsoft**' un **Ado.Net** tarafında söz konusu EDM modelini **Entity Framework** ile nasıl uyguladığına değinilmeye başlanıyor ve siz ısındıktan sonrada teknik detaylara geçilerek profesyonel seviyede ilerleniyor.

Ben kütüphanemin baş köşesine koydum. Tabi teknoloji tekrardan yeni değişiklikleri beraberinde getirinceye dek.

NedirTv? Nisan Ayı Webinerleri (2009-04-16T09:33:00)

webiner,

Nisan ayı içerisinde NedirTv? webinerleri devam ediyor. Bu ayki program oldukça yoğun. **Silverlight 3.0 Beta**, **ASP.NET MVC** ve **SQL Data Services** gibi… Ben bu aylık kendimi nadasa çekmiş durumyadım. Ancak Mayıs ayı içerisinde olağanüstü durumlar olmassa WF 4.0, WCF 4.0, Dublin gibi konuları ele alanan webinerlerimle tekrar karşınızda olacağım umuyorum ki.

Detaylı programa ve webiner bilgilerine **bu linkten** ulaşabilir, **bu linkten** de Facebook'taki olayı takviminize kaydedebilirsiniz.

WF 4.0 Makaleleri ve ilk Screencast' im [Pre Beta] (2009-04-16T06:48:00)

wcf 4.0,wf 4.0,

2008 yılının ikinci yarısından beri, WF 4.0 ve WCF 4.0 ile ilişkili araştırmalar ve çalışmalar yapmaktayım. Artık çalışmaları sizlerlede paylaşmanın zamanı geldi. Her ne ne kadar konuları PDC 2008' de yayımlanan VPC imajı üzerinden ve dolayısıla PreBeta sürümünden yapsamda, gelecek olan değişikliker gerçekten heyecan verici.

İşte ilk paylaşımlarım...

Faydalı olmasını dilerim.

Makaleler;

Windows Workflow Foundation 4.0 - İlk İzlenimler

WF 4.0 - WCF Servislerini Kullanmak

Screencasts;

WF 4.0 - Aktivite Geliştirmek(Pre Beta)

INETA Next Hit Gerçekleşti (2009-04-16T06:35:00)

ineta seminerleri,

INETA Türkiye tarafından düzenlenen ve 11, 12 Nisan 2009 tarihlerinde Yıldız Teknik üniversitesi, Barbaros Oditoryumunda tamamlanan etkinlikte, Windows Workflow Foundation 4.0 ve Windows Communication Foundation konulu bir seminer verdim.

Sunumda PDC 2008' de yayımlanan VPC imajları üzerinden çok kısada olsa bir demo yapma şansımızda oldu. Seminerdeki sunumuma WCF4WF4-Sunum.pptx (1,43 mb) linkinden ulaşabilirsiniz.

Katılan ve beni sabırla dinleyen tüm meslektaşlarıma ayağınıza sağlık diyorum.

WF için SQL Persistence Hizmetinin Kullanımı (2009-04-13T07:44:00)

wf,

Workflow örneklerine ait aktivitelerde dikkat edilmesi gereken noktalardan biriside uzun süreli çalışmalara neden olacak akışlarında söz konusu olmasıdır. Bu noktada SQL tabanlı olarak, Workflow örneklerinin kalıcı olarak saklanması yaygın olarak kullanılan tekniklerden birisidir. Bu konuyu .Net Framework 3.5 sürümü üzerinde detaylı olarak inceledim ve konuyla ilgili olaraktanşu makaleyi yazdım.

Bu arada WF 4.0 içerisinde özellikle Persistence işlemleri için getirilen bir aktivite tipi olduğunuda belirtmek isterim. Diğer taraftan Dublin kod adlı Windows Application Server ile birlikte IIS üzerine gelen eklentiler ile, WF Servislerinin, persistence, tracing, monitoring,throttling gibi önemli kıstaslarını yönetmemiz daha kolay ve güçlü bir şekilde olacak. (Bu konularla ilişkili olaraktan ilerleyen zamanlarda bir kaç görsel ders yayınlıyor olacağım.

Nihayet, en sonunda, çok şükür, Blog' um yayında... (2009-04-12T20:22:00)

Merhabalar,

Uzun süredir kişisel sitemden makale, görsel video paylaşımı yapmaktayım. 2004 yılında C#Nedir? ile başladığım makale serüvenimde senkronize bir şekilde kişisel sitemide kullandım. Görsel iletişimin gücüne inandığım için .Net TV ile başlayan Screencast' ler, NedirTV? de devam eden video editörlüğü derken, pek çok meslektaşımın sorusunu yıllarca düşündüm durdum. "Niye bir blog yok?" .

Ama artık var. Genellikle uzun olan(özellikle MVP olduğum dönemden beri, ortalama 15sayfanın üstünde tutmaya çalışıyorum) makalelerimi yine kişisel sitem ve C#Nedir? aracılığıyla takip edebilir buradaki blog'tanda eklenen makale veya görsel dersler(ki ağırlık bundan sonra bu yönde olacak) hakkında bilgi alabilirsiniz. Bunların haricinde, okuduğum ve sizlere tavsiye edeceğim kitaplar, seminer veya webiner gibi etkinlik duyuruları, kısa tip ve trick' ler. Yine makale olarak sayılamayacak, yani az sayıda sayfadan oluşan teknik yazılarım bu blog aracılığıylada ulaşabilirsiniz.

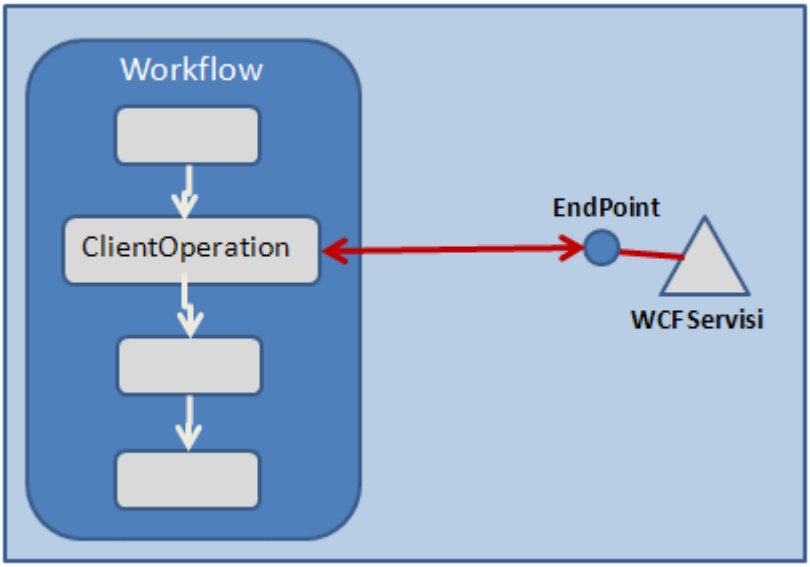
Elbetteki emektar sitemide (Asp.Net 2.0' a taşınmış ama 3.5 hazırlıklarıda devam eden ve belkide 4.0 ile yayınlanacak olan) unutmamanızı rica ederim. Oradaki çalışmalarımda elbetteki devam ediyor olacak.

WF 4.0 - WCF Servislerini Kullanmak (2009-04-01T04:57:00)

wf 4.0,wcf 4.0,

Bir önceki yazımızda **Windows Workflow Foundation 4.0 (WF 4.0)** ile birlikte gelmesi muhtemel*(yüksek bir olasılıkla çok az değişikle gelecekler)* kavramları incelemeye çalışmıştık ve pek çok yeni aktivite tipinin alt yapıya dahil edilmiş olduğunu gördük. **WF** örnekleri bilindiği üzere çoğu zaman servisler ile haberleşmek durumundadır. Bu özellikle gerçek hayat senaryolarında çok sık karşılaşına ve ihtiyaç duyulan bir durumdur. Nitekim WF içerisinde yer alan akışların dış ortamlara olan bir bağımlılığı söz

konusu olabilir. Bir Bankacılık sisteminde yer alan akışlarda, servisler yardımıyla ulaşılabilen bazı operasyonlar bu bağımlılığa örnek gösterilebilir örneğin. **WF** alt yapısı bu anlamda**WCF(Windows Communication Foundation)** servisleri ile haberleşilebilmesini kolaylaştırmak amacıyla **.Net Framework 3.5** ile birlikte yeni aktivite bileşenlerine sahip olmuştur. **SendActivity** ve**ReceiveActivity** isimli bu tipler temel olarak servislere ait operayonların çağırılması veya **WF** içerisinde servis bazlı operasyonların dış dünyaya sunulmasında etkin olarak kullanılmaktadır. Ancak WF 4.0 ile birlikte servisler ile olan iletişimde daha yetenekli aktivite tipleri yer almaktadır. Özellikle görsel açıdan geliştiriciye kolaylıklar sağlayan ama asıl etkisini **XAML** bazlı servis tanımlamalarının yapılabilmesinde gösteren aktiviteler söz konusudur. Zaten **WF** ve **WCF 4.0** içerisinde XAML tabanlı deklerafit tanımlamaların son derece etkin bir şekilde kullanıldığı bir gerçektir. **WF 4.0** açısında bakıldığında bir **WCF** operasyonu ile sağlanan **istek/cevap(Request/Response)** odaklı iletişim temel olarak aşağıdaki şekilde görüldüğü gibidir.



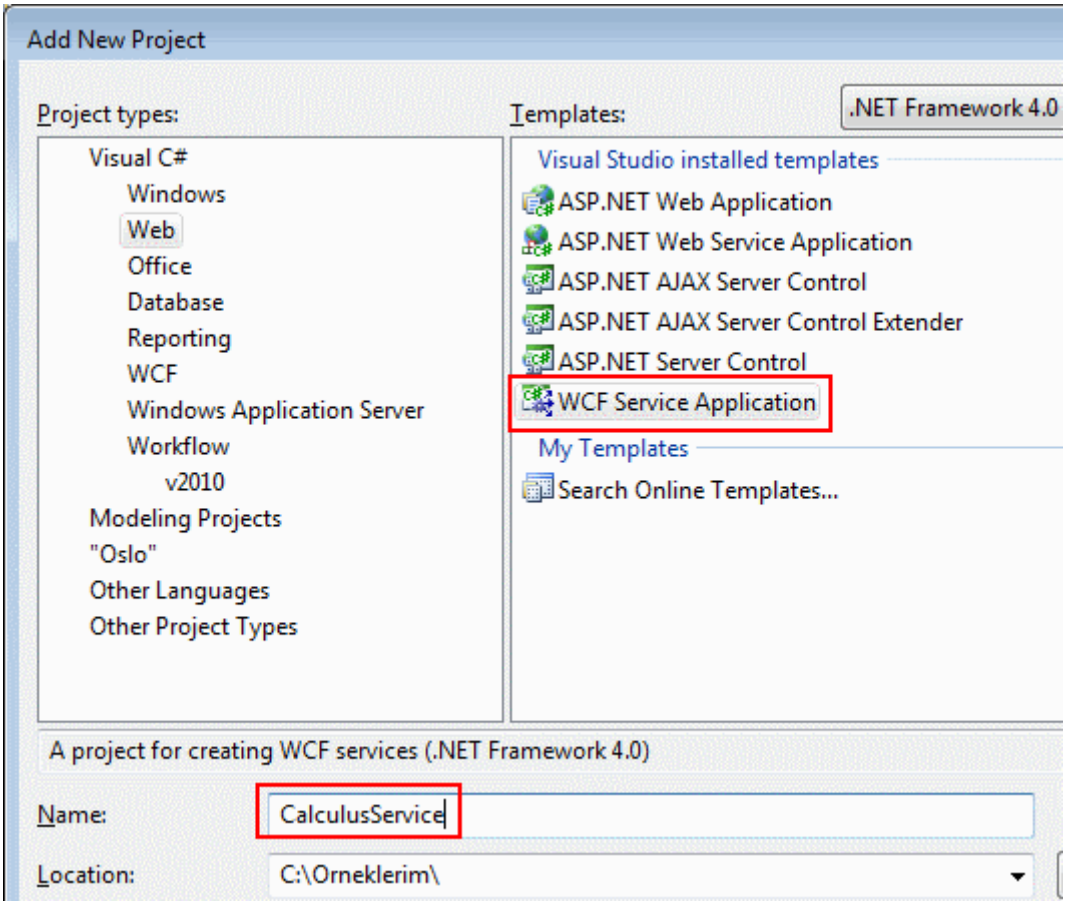
Buna göre **WF** içerisinde herhangibir noktada servis operasyonlarını çağırmak için **ClientOperation** isimli aktivite kullanılmaktadır. Bu operasyonun yer aldığı **WCF** servisi bir veya daha fazla**EndPoint** üzerinden istemcilere hizmet verebilir. **WF** örnekleri kendi içerisinde, bu **EndPoint**' in belirttiği **Address**, **Binding** ve **Contract** tipine göre uygun bir mesajlaşma trafiği başlatabilir. Buna göre servis operasyonları çağırabilir ve sonuçlarını ele alabilir. **ClientOperation**, **Workflow Foundation 4.0** ile birlikte gelen yeni aktivite tiplerinden birisidir. Peki bu yeni tipin **.Net Framework 3.5**sürümü ile gelen ve aynı amaçla kullanılan **SendActivity** tipine göre farklılıkları, özellikle avantakları neler olabilir? İşte bu yazımızda kısaca bu sorulara cevap bulmaya çalışacak ve aynı zamanda bizleri bekleyen yenilikleri göreceğiz. Bu nedenle bir örnek üzerinden adım adım ilerleyerek devam etememizde yarar olacağı kanısındayım.

Senaryomuza göre basit ve herzamanki gibi gerçek hayatta kullanılmayacak bir **Sequential Workflow** uygulaması geliştireceğiz. Akışımız yine geliştirici tarafından tasarlanmış özel

bir aktivite tipini kullanarak istemciden iki sayısal değer alacaktır. Bu değerler bir servise gönderilerek işlenecek ve sonuçlar yine akış içerisine yönlendirilerek diğer bir özel aktivite tipi yardımıyla ekrana yazdırılacaktır. İlk olarak servis uygulamasının tasarlanmasında yarar vardır.

Not : *Yazımızda geliştirmekte olduğumuz örnek henüz **Relase** olmamış **.Net Framework 4.0** sürümü üzerinde geliştirilmekte ve bu amaçla **Visual Studio 2010**' un **PDC 2008**' de yayımlanan **Virtual PC** versiyonu kullanılmaktadır. Dolayısıyla yazılan ve işlenen kavramlarda veya **Visual Studio 2010** sürümünde köklü değişiklikler olabilir, olması muhtemeldir.*

CalculusService isimli **WCF** uygulamamız **WCF Service Application** tipinde geliştirilmektedir.



Serviste kullanılan sözleşme içeriği ise aşağıdaki kod parçasında görüldüğü gibidir.

```
using System.ServiceModel;

namespace CalculusService
{
   [ServiceContract]
   public interface IMatService
```

```
    {
        [OperationContract]
        double Sum(double x, double y);
    }
}
```

Servis sözleşmesinde**(Service Contract)**, çok basit olarak **double** tipinden iki sayısal değerin toplamını alarak geriye sonucunu döndüren **Sum** isimli bir operasyon yer almaktadır. **IMatService**isimli servis arayüz tipini**(Interface)** uygulayan sınıfa ait kod içeriği ise aşağıdaki gibidir.

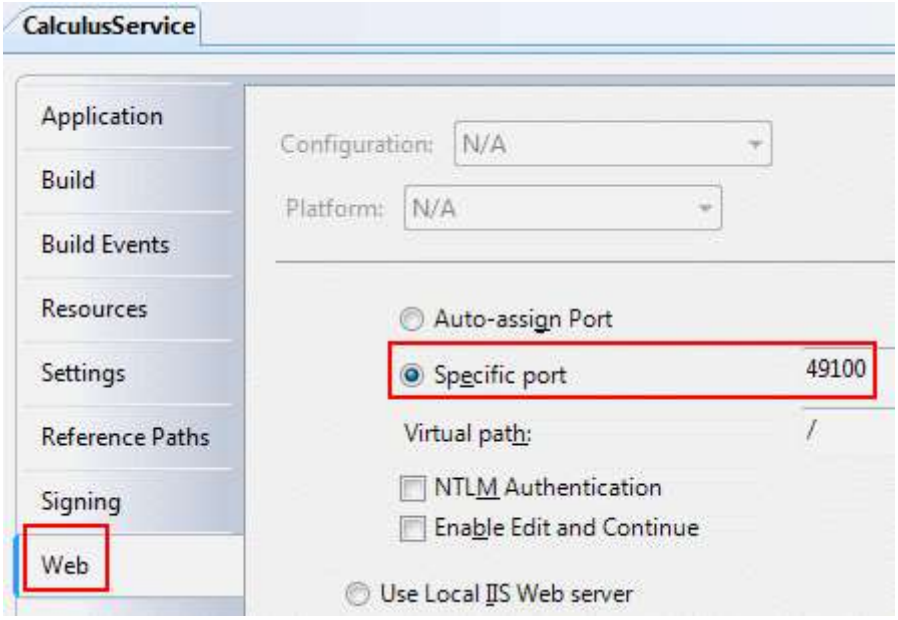
```
namespace CalculusService
{
    public class MatService
        : IMatService
    {
        public double Sum(double x, double y)
        {
            return x + y;
        }
    }
}
```

Servis üzerinde tanımlanmış olan operasyonlar basit **HTTP** protokolüne göre sunulmaktadır. Bu nedenle bağlayıcı tip olarak **BasicHttpBinding** kullanılmaktadır. Servis için birde **HTTP**üzerinden **metadata** bilgisinin verilebilmesi amacıyla **MexHttpBinding** bazlı bir **EndPoint** daha söz konusudur.*(Gerçi biraz sonra görebileceğimiz gibi, **WF** örneğinin servise ait bir **Metadata**indirmesine gerek kalmayacaktır :))* Örnek **WCF** servis uygulamamıza **EndPoint** ayarları aşağıdaki **Web.config** dosyasının içeriğinde olduğu gibi tanımlanabilir.

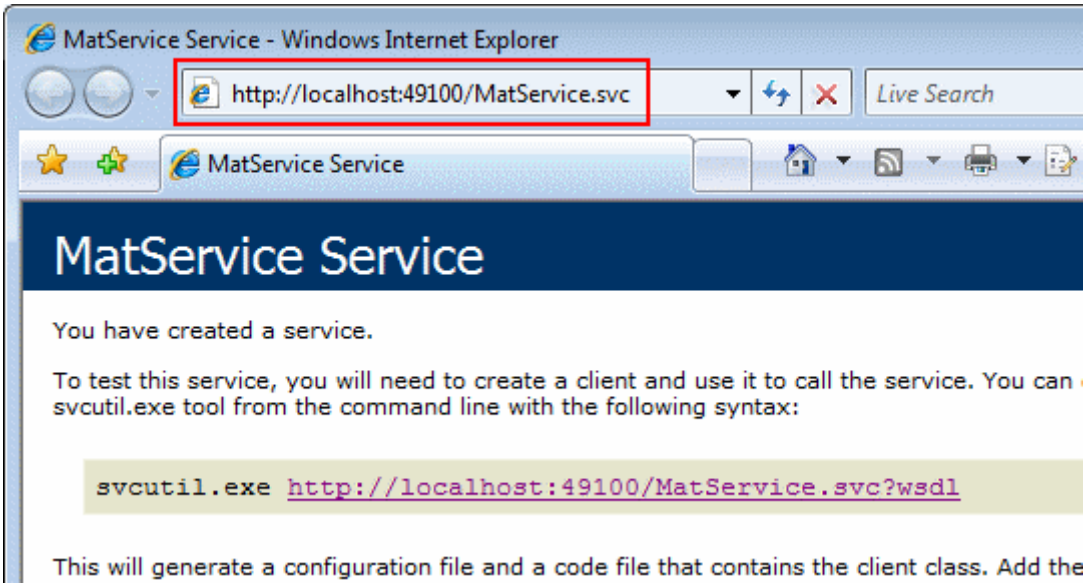
```
<?xml version="1.0"?>
<configuration>
    <appSettings/>
    <connectionStrings/>
    <system.web>
        <compilation debug="true"></compilation>
        <authentication mode="Windows"/>
    </system.web>
    <system.serviceModel>
        <services>
            <service behaviorConfiguration="CalculusService.MatServiceBehavior"
name="CalculusService.MatService">
                <endpoint address="" binding="basicHttpBinding"
bindingConfiguration="" contract="CalculusService.IMatService">
```

```
            <identity>
               <dns value="localhost" />
            </identity>
         </endpoint>
         <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange" />
      </service>
   </services>
   <behaviors>
      <serviceBehaviors>
         <behavior name="CalculusService.MatServiceBehavior">
            <serviceMetadata httpGetEnabled="true" />
            <serviceDebug includeExceptionDetailInFaults="true" />
         </behavior>
      </serviceBehaviors>
   </behaviors>
</system.serviceModel>
</configuration>
```

WCF servis uygulamamız test amacıyla geliştirildiğinden **IIS(Internet Information Services)** üzerine atılmamıştır. Bu nedenle **Solution** içerisinde yer alan istemci uygulamalar ile haberleşmesi sırasındaki geliştirme sürecini kolaylaştırmak adına, sabit bir **HTTP** portu kullanması sağlanabilir. Sabit **port** ayarlaması için **WCF** projesinin özelliklerinden***(Properties)*** aşağıdaki ekran görüntüsünde olduğu gibi **Specific Port** özelliğine bir değer atamak yeterli olacaktır. *(Elbetteki bu **WCF** uygulamasını, **IIS** altına atmak kolay bir şekilde özellikler penceresinde yer alan **Use Local IIS Web server** seçeneği ile mümkün olabilir. Yada bu amaçla **Publish** işlemlerinden yararlanılabilir.)*



Bu aşamada ilerlemeden önce servisin **HTTP** üzerinden çağırılabildiğinden emin olmak gerekir. Bu amaçla, **MatService.svc** dosyasının bir tarayıcı içerisinde açılması yeterlidir. Eğer aşağıdaki ekran görüntüsünde yer alan sonuçlar elde edilebiliyorsa **WCF** servisinin çağırılabilir olduğu sonucuna varılabilir.



Gelelim **WF** tarafına. Her zamanki gibi basit
bir **Sequential Workflow Console Application** üzerinden ilerliyor olacağız. Servise gönderilecek parametreleri ekrandan almak ve yazdırmak içinse bir önceki yazı dizimizdekilere benzer iki basit aktivite tipi kullanıyor olacağız. **Console** ekranından bilgi okumak için kullanılan **Read** aktivitesine ait kod içeriği aşağıdaki gibi geliştirilebilir.

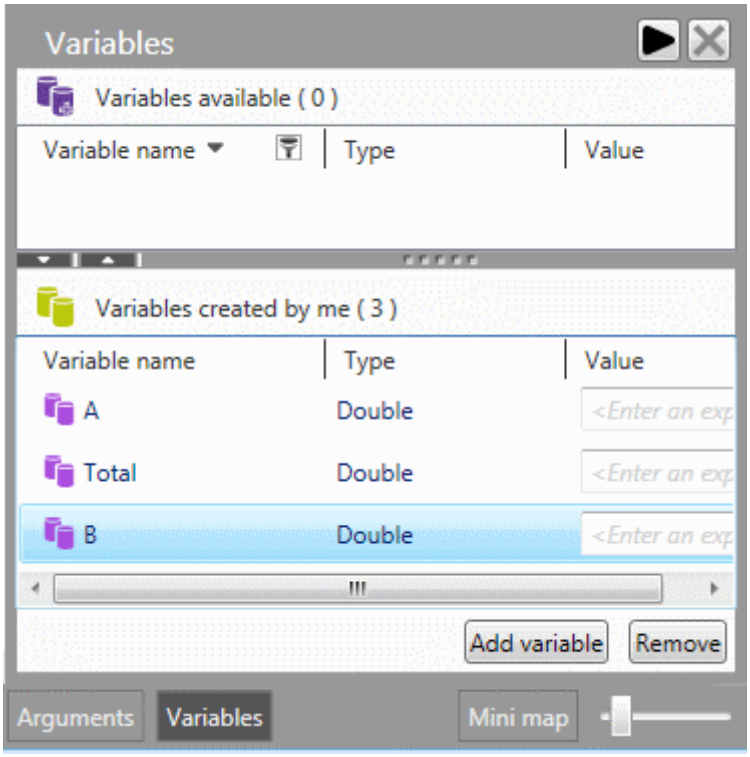
```
using System;
using System.WorkflowModel;

namespace CalculusWF
{
   public class Read
      :WorkflowElement
   {
      public OutArgument<double> Value1 { get; set; }
      public OutArgument<double> Value2 { get; set; }

      protected override void Execute(ActivityExecutionContext context)
      {
         Console.WriteLine("Value 1 ?");
         Value1.Set(context, Convert.ToDouble(Console.ReadLine()));
         Console.WriteLine("Value 2 ?");
         Value2.Set(context, Convert.ToDouble(Console.ReadLine()));
      }
```

```
    }
}
```

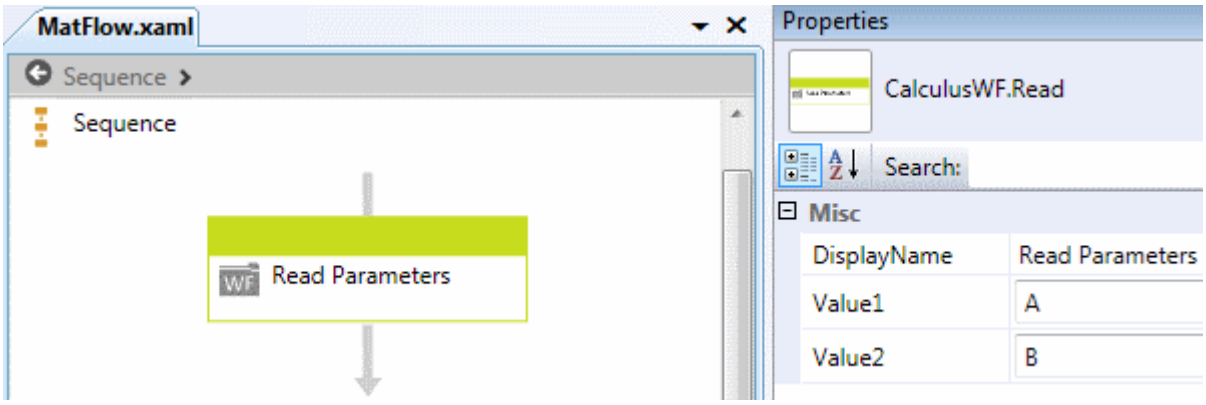
Read aktivitesi görüldüğü üzere iki adet **OutArgument<T>** tipinden özellik kullanmakta ve **override** ettiği **Execute** metodu içerisinde ekrandan aldığı değerleri dış ortama sunmaktadır. Yine dikkat edilmesi gereken noktalardan birisi, aktivitenin **WF Base Library** içerisindeki yeni ata sınıf olan **WorkflowElement** tipinden türemiş olmasıdır. **Write** aktiviteside **Read** aktivitesi gibi**WorkflowElement** türevlidir ve **WF** içeriğinden(**ActivityExecutionContext** yardımıyla) gelen sonuç değerini ekrana yazdırmak için kullanılmaktadır. **Write** aktivitesine ait kod içeriği aşağıda görüldüğü gibidir.

```
using System;
using System.WorkflowModel;

namespace CalculusWF
{
   class Write
      :WorkflowElement
   {
      public InArgument<double> Result { get; set; }

      protected override void Execute(ActivityExecutionContext context)
      {
         double r=Result.Get(context);
         Console.WriteLine("İşlem sonucu {0} dır",r.ToString());
      }
   }
}
```

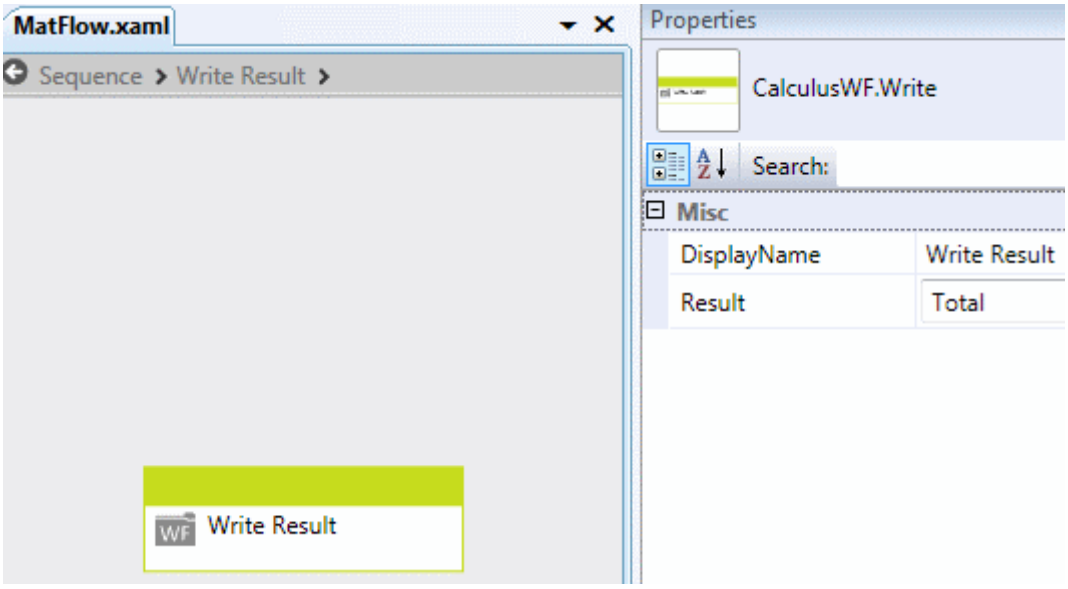
Workflow içerisinde toplama işlemi için kullanılacak değerler ile işlem sonucuna tüm aktivite boyunca ulaşılması istendiğinden aşağıdaki ekran görüntüsünde yer aldığı gibi üç adet **Variable** tanımlaması yapılmaktadır. Bu tanımlamalar **Matflow.xaml** isimli **Sequential Activity** tipinin tamamı için geçerlidir.



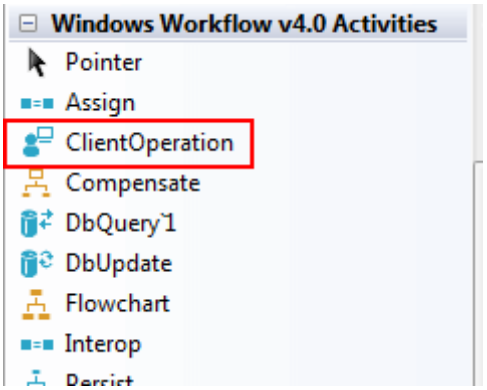
A, **B** ve **Total** isimli değişkenler **double** tipinden tanımlanmıştır. Bu değerler **Read**, **Write** aktiviteleri ile **ClientOperation** tarafından kullanılabilecektir. Bu işlemin ardından artık aktivite dizisinin oluşturulmasına başlanabilir. İlk olarak Read aktivitesi sürüklenir. Söz konusu aktivitenin özellikleri aşağıdaki gibi ayarlanabilir.



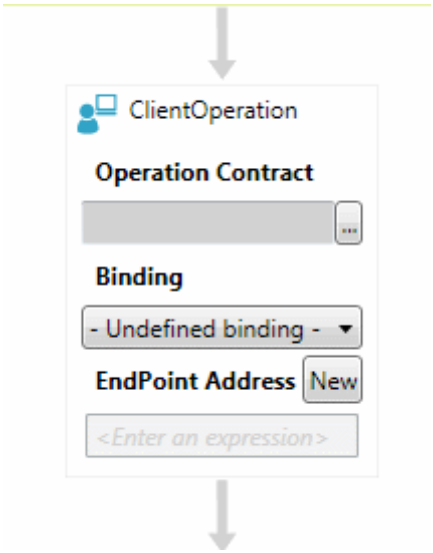
Dikkat edileceği üzere **Read** aktivitesi içerisinde tanımlanmış olan **Value1** ve **Value2** isimli özelliklere, **Sequence** aktivitesi içerisinde tanımlanan **A** ve **B** değişkenleri atanmıştır. Bir başka deyişle **Read** aktivitesi ile **komut satırından** okunup set edilen **Value1** ve **Value2** değerleri diğer aktiviteler tarafından kolayca ele alınabileceklerdir. Nitekim **A** ve **B** isimli **global**değişkenleri taşınmaktadırlar. **Read** ve **Write** aktiviteleri arasına **ClientOperation** aktivitesini eklemeden önce, **Write** aktivitesi içinde aşağıda görülen özellik ayarlamalarını yapmamız yeterli olacaktır.



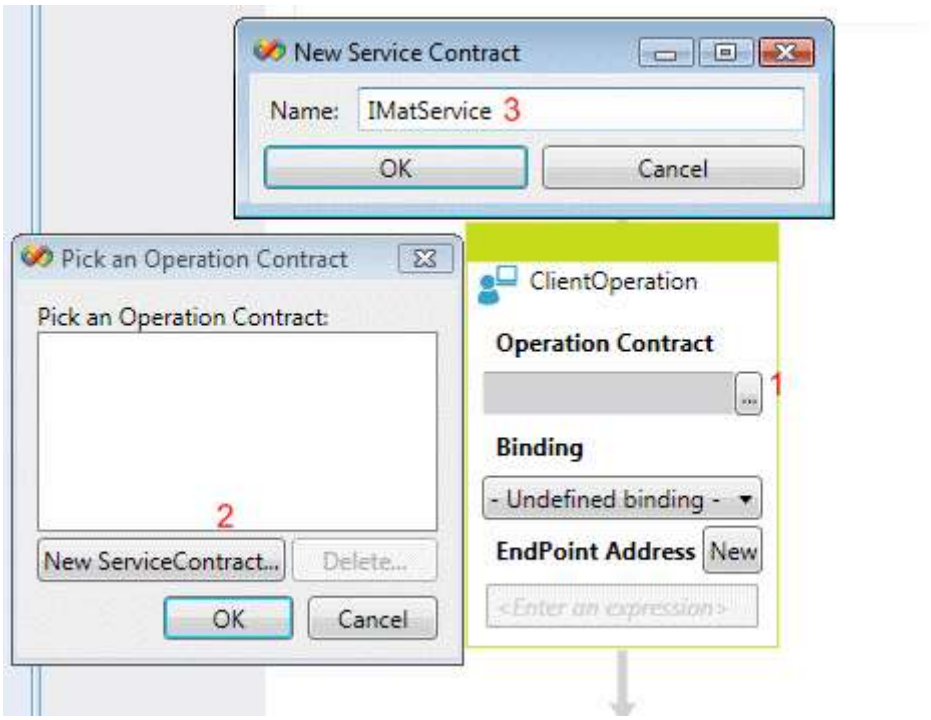
Dikkat edileceği üzere **Write** aktivitesi içerisinde tanımlanmış olan **Result** isimli özelliğe **global** değişkenlerden **Total** atanmıştır. Artık **Write** aktivitesi ile okunarak global seviyedeki **A** ve**B** değişkenlerine atanan değerleri, kullanılmak üzere ele alacak ve toplam sonucunu **Total** isimli değişkene verecek olan **ClientOperation** aktivitesini geliştirmeye başlayabiliriz.



ClientOperation aktivitesini **Read** ve **Write** aktiviteleri arasına sürükleyip bıraktığımızda ilk etapta aşağıdaki ekran görüntüsü ile karşılaşırız. Dikkat edileceği üzere**Operation Contract**, **Binding** ve **EndPoint Address** isimli 3 önemli özellik göze çarpmaktadır.

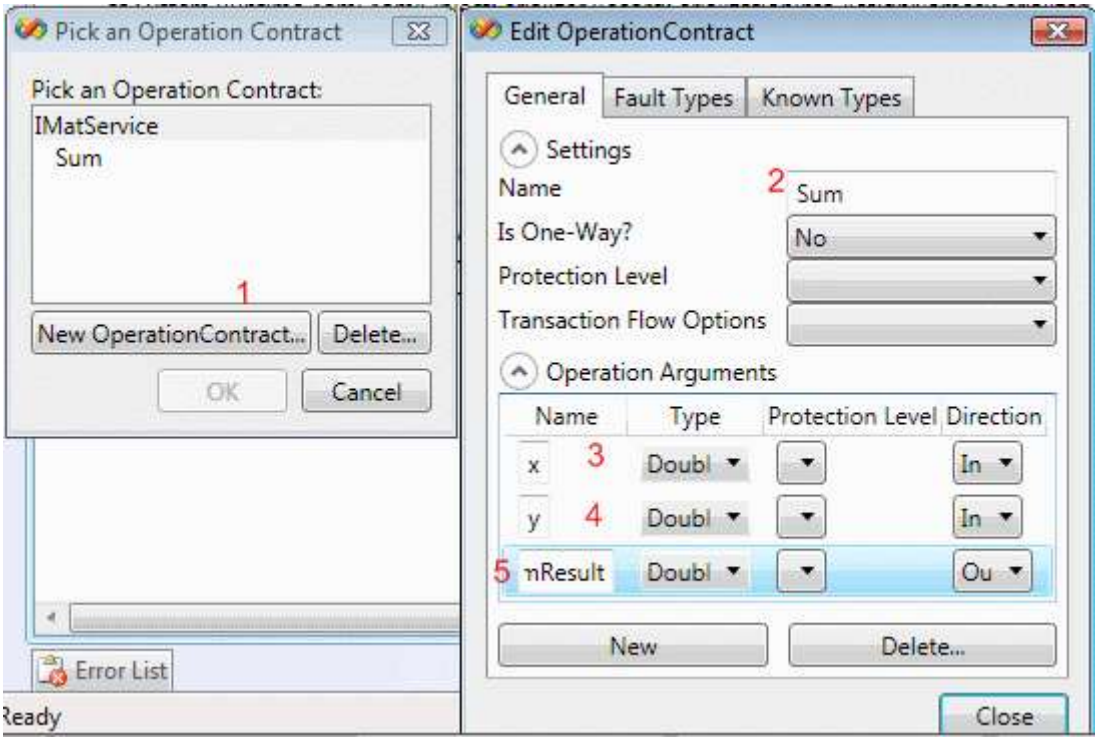


Tahmin edileceği üzere bu özelliklerin değerleri ile istemci için gerekli bir **EndPoint** bilgisi oluşturulabilir. Bir başka deyişle bir **EndPoint** tanımını oluşturan **adresleme(Address), bağlayıcı tip(Binding Type)** ve **sözleşme(Contract)** bilgilerinin tamamı bu aktivite tipi içerisinde belirlenmektedir. İlk olarak aşağıdaki ekran görüntüsünde yer alan adımlar takip edilerekten**servis sözleşmesinin(Service Contract)** adı girilir.

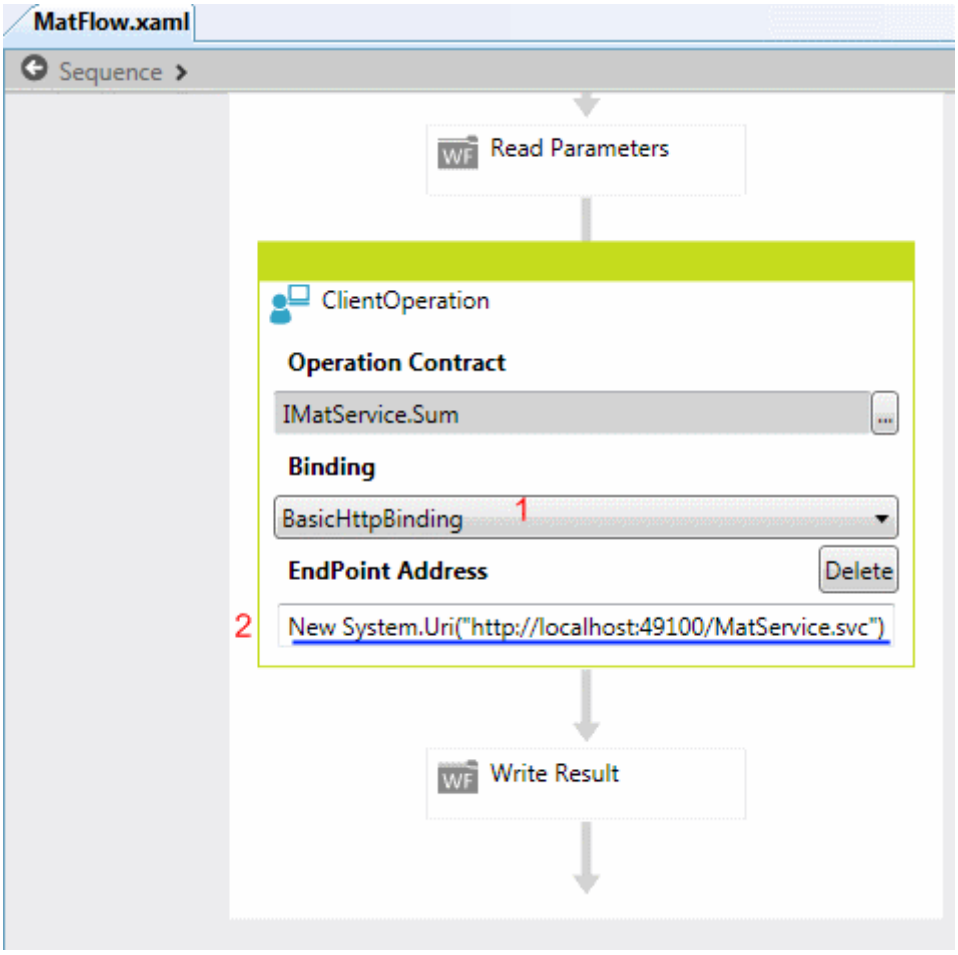


Visual Studio 2010 ürün olarak sunulduğunda servis sözleşmesi gibi kısımların elle değil otomatik olarak girilebilecek şekilde ayarlanabileceğini düşünmekteyim. Şimdilik servis tarafındaki sözleşme arayüzü tipinin adını elle(büyük küçük harf duyarlılığına da dikkat ederekten) yazmamız gerekmektedir. Servis sözleşmesinin belirlenmesi tek başına yeterli değildir. **ClientOperation** aktivitesinin bu servis operasyonu üzerinde hangi operasyonu çağıracağınında belirlenmesi gerekmektedir. Burada **New ServiceContract** düğmesi

yardımıyla sözleşme tanımlandığında, başlığın **New OperationContract** olarak değiştiği gözlemlenir. Tahmin edileceği gibi **New OperationContract** düğmesi ve takip eden adımlar ile**Sum** operasyonuna ait tanımlamalar görsel olarak yapılabilmektedir. Aynen aşağıdaki ekran görüntüsünde yer aldığı gibi.

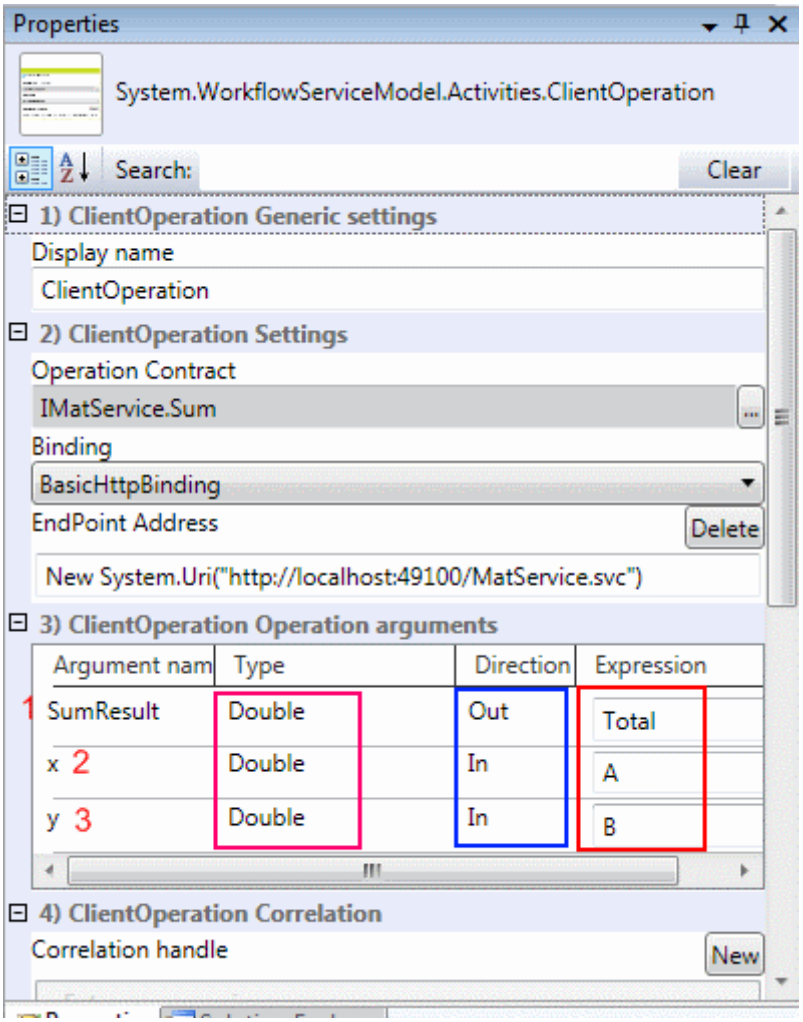


Burada çok detaylı operasyon ayarlamaları yapılabilmektedir. Güvenlik ile ilişkili işlemler**(Protection Level)**, **Transaction** aktarma opsiyonları, operasyonun **tek yönlü(One-Way)** olup olmadığı, operasyondan dönebilecek **Fault** tipleri vb... Geliştirdiğimiz örnekte sadece operasyon parametrelerinin girilmesi yeterlidir. **x** ve **y** isimli argümanlar, **Sum** operasyonuna ait girdi parametreleri olduklarından **Direction** özellikleri **In** olarak belirlenmiştir. Diğer taraftan **SumResult** isimli argüman, operasyonun dönüş değerini işaret etmekte olduğundan, **Direction** özelliğine **Out** değeri verilmiştir. Şu anda çağırılacak servis operasyonu ile ilişkili tanımlamalar yapılmıştır. Artık bu servis ile hangi bağlayıcı tip ve adres ile iletişime geçileceğine dair özelliklerin belirlenmesi gerekmektedir. Bu amaçla aşağıdaki ekran görüntüsünde yer alan atamaların yapılması yeterlidir.



Unutulmaması gereken noktalardan biriside **WCF** servisleri ile haberleşecek olan istemci uygulamaların, servis tarafında belirtilen **EndPoint** içeriğine uygun **EndPoint** bildirimlerine sahip olması zorunluluğudur. Geliştirilen örnekte servis tarafında **BasicHttpBinding** tipi kullanıldığından istemci tarafındaki **EndPoint** içinde aynı tipte bir bağlayıcının kullanılması gerekmektedir. Benzer olarak servis tarafındaki operasyon **HTTP** bazlı olaraktan **localhost** isimli makineden ve **49100** nolu **port** üzerinden sunulmaktadır. Bu nedenle istemci tarafından bu kritere uygun bir**adres** için talepte bulunulmalıdır. Dikkat çekici noktalardan biriside adres kısmında **C#** notasyonuna pek uymayan bir şekilde büyük harfle başlayan bir **New** anahtar kelimesi olmasıdır. (**WF 4.0** ile ilişkili yenilikleri öğrendiğim **Microsoft LAB** dökümanlarında bu durumun final sürümünde değişeceğine dair bir bilgi yer almaktadır. Tabi final sürümünde bizleri neler bekliyor neler...) Artık istemci tarafındaki **EndPoint** için gerekli **ABC(AddressBindingContract)** bilgileri tanımlanmış durumdadır.

Yapılması gereken önemli işlemlerden biriside **WF** içerisindeki değişkenleri **ClientOperation** ile servise aktarmak ve servisten dönen değeride tekrardan **WF** içerisine yönlendirmektir. Bunun için**ClientOperation** elementinin özelliklerinde aşağıdaki ekran görüntüsünde yer alan ayarlamaların yapılması yeterli olacaktır.



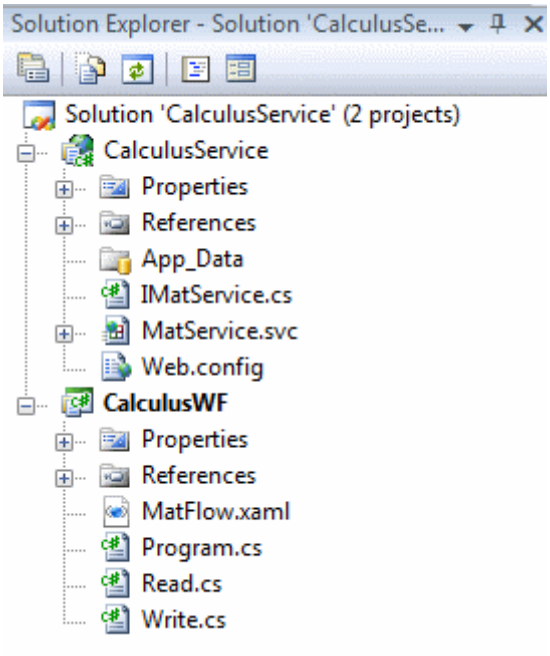
Görüldüğü üzere **SumResult** isimli servis operasyon argümanı **Total** isimli **WF** değişkenine, **x** isimli servis operasyon argümanı **A** isimli **WF** değişkenine ve son olarak **y** isimli servis operasyonu argümanı **B** isimli **WF** değişkenine set edilmiştir. İşte bu kadar. Görüldüğü üzere görsel olarak pek çok ayarlama yapılmıştır. Sonuç olarak **MatFlow.xaml** adlı **Workflow** örneğimiz için aşağıdaki **XAML(eXtensible Application Markup Language)** içeriği üretilmektedir.

<p:Activity x:Class="CalculusWF.MatFlow" xmlns:c="clr-namespace:CalculusWF;assembly=CalculusWF" xmlns:p="http://schemas.microsoft.com/netfx/2009/xaml/workflowmodel" xmlns:p1="http://schemas.microsoft.com/netfx/2008/xaml/schema" xmlns:p2="http://schemas.microsoft.com/netfx/2009/xaml/servicemodel" xmlns:s="clr-namespace:System;assembly=System" xmlns:swd="clr-namespace:System.WorkflowModel.Debugger;assembly=System.WorkflowModel" xmlns:swdx="clr-namespace:System.WorkflowModel.Design.Xaml;assembly=System.WorkflowModel.Design" xmlns:sx="clr-namespace:System.Xml;assembly=System.Runtime.Serialization" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

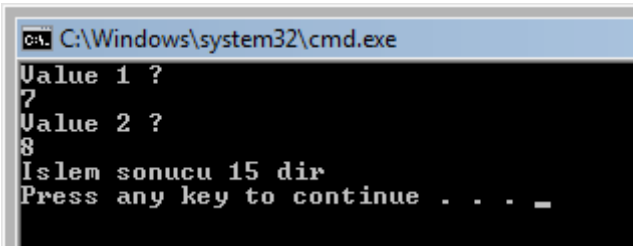
```
   <p:Sequence swd:XamlDebuggerXmlReader.FileName=
"C:\Orneklerim\CalculusService\CalculusWF\MatFlow.xaml">
      <p:Sequence.Variables>
         <p:Variable x:TypeArguments="p1:Double" Name="A" />
         <p:Variable x:TypeArguments="p1:Double" Name="Total" />
         <p:Variable x:TypeArguments="p1:Double" Name="B" />
      </p:Sequence.Variables>
      <c:Read DisplayName="Read Parameters" Value1="[A]" Value2="[B]" />
      <p2:ClientOperation EndpointAddress="[New
s:Uri(&quot;http://localhost:49100/MatService.svc&quot;)]"
OperationName="Sum">
         <p2:ClientOperation.Endpoint>
            <p2:Endpoint Name="ClientOperationEndpoint">
               <p2:Endpoint.Binding>
                  <p2:BasicHttpBinding TextEncoding="utf-8">
                     <p2:BasicHttpBinding.ReaderQuotas>
                        <sx:XmlDictionaryReaderQuotas />
                     </p2:BasicHttpBinding.ReaderQuotas>
                     <p2:BasicHttpBinding.Security>
                        <p2:BasicHttpSecurity Mode="None">
                           <p2:BasicHttpSecurity.Message>
                              <p2:BasicHttpMessageSecurity AlgorithmSuite="Default"
ClientCredentialType="UserName" />
                           </p2:BasicHttpSecurity.Message>
                           <p2:BasicHttpSecurity.Transport>
                              <p2:HttpTransportSecurity />
                           </p2:BasicHttpSecurity.Transport>
                     </p2:BasicHttpSecurity>
                  </p2:BasicHttpBinding.Security>
               </p2:BasicHttpBinding>
            </p2:Endpoint.Binding>
            <p2:Endpoint.ContractProjection>
               <p2:SoapContractProjection>
                  <p2:SoapContractProjection.Contract>
                     <p2:ServiceContract Name="IMatService">
                        <p2:OperationContract Name="Sum">
                           <p2:OperationArgument Name="x" Type="p1:Double" />
                           <p2:OperationArgument Name="y" Type="p1:Double" />
                           <p2:OperationArgument Direction="Out" Name="SumResult"
Type="p1:Double" />
                        </p2:OperationContract>
                     </p2:ServiceContract>
                  </p2:SoapContractProjection.Contract>
               </p2:SoapContractProjection>
            </p2:Endpoint.ContractProjection>
```

```
      </p2:Endpoint>
   </p2:ClientOperation.Endpoint>
      <p2:ClientOperation.OperationArguments>
         <p:OutArgument x:Key="SumResult"
x:TypeArguments="p1:Double">[Total]</p:OutArgument>
         <p:InArgument x:Key="x" x:TypeArguments="p1:Double">
[A]</p:InArgument>
         <p:InArgument x:Key="y" x:TypeArguments="p1:Double">
[B]</p:InArgument>
      </p2:ClientOperation.OperationArguments>
   </p2:ClientOperation>
   <c:Write DisplayName="Write Result" Result="[Total]" />
  </p:Sequence>
</p:Activity>
```

Burada durup aslında biraz soluklanmak belki bir yudum kahve içmek ve sonrasında **XAML** içeriğine yoğunlaşarak düşünmek gerekmektedir. Dikkat edileceği üzere **WF**' in tüm içeriği,**Write**, **Read**, **ClientOperation** aktiviteleri, **global WF** değişkenleri bu **XAML** içeriğinde bildirilmektedir. Tabiki bu **XAML** içeriği çalışma zamanı tarafında değerlendirilmektedir.**Dekleratif(Declerative)** bir yaklaşımdan ziyade **ClientOperation** aktivitesinin **XAML** içerisine nasıl gömüldüğüne dikkat etmemizde yarar vardır. Öyleki **EndPoint** ve buna ait**AddressBindingContract** bilgilerinin tamamı **XAML** elementleri içerisinde oluşturulmuştur. Buna göre **XAML** içeriği basit bir editor yardımıyla değiştirildiği takdirde **Workflow** örneğinin yeni ortam şartlarına göre adapte edilmesi kolayca sağlanabilir. Söz gelimi, servis adresinin değişmesi veya operasyon adında yada parametrik yapısında oluşabilecek değişiklikleri koda girmeden düzenleyebilir ve **WF**' in güncellenmesini sağlayabiliriz. Bunun sağlanmasının en büyük etkenlerinden birisi **Workflow** bazlı **WCF** servislerinin kod yazmadan **XAML** tabanlı geliştirilip kullanılabiliyor olmasıdır. *(Bu alt yapıyı ve **Workflow** bazlı bir **WCF** servisinin **XAML** ile dekleratif olarak nasıl tanımlanabileceğini ilerleyen makalelerimizde veya görsel derslerimizde incelemeye çalışacağız.)* Artık örneğimizi test etmeye başlayabiliriz. Bundan önce **Solution**' ımızın son halinin aşağıdaki ekran görüntüsüne benzer olacağını düşünebiliriz.



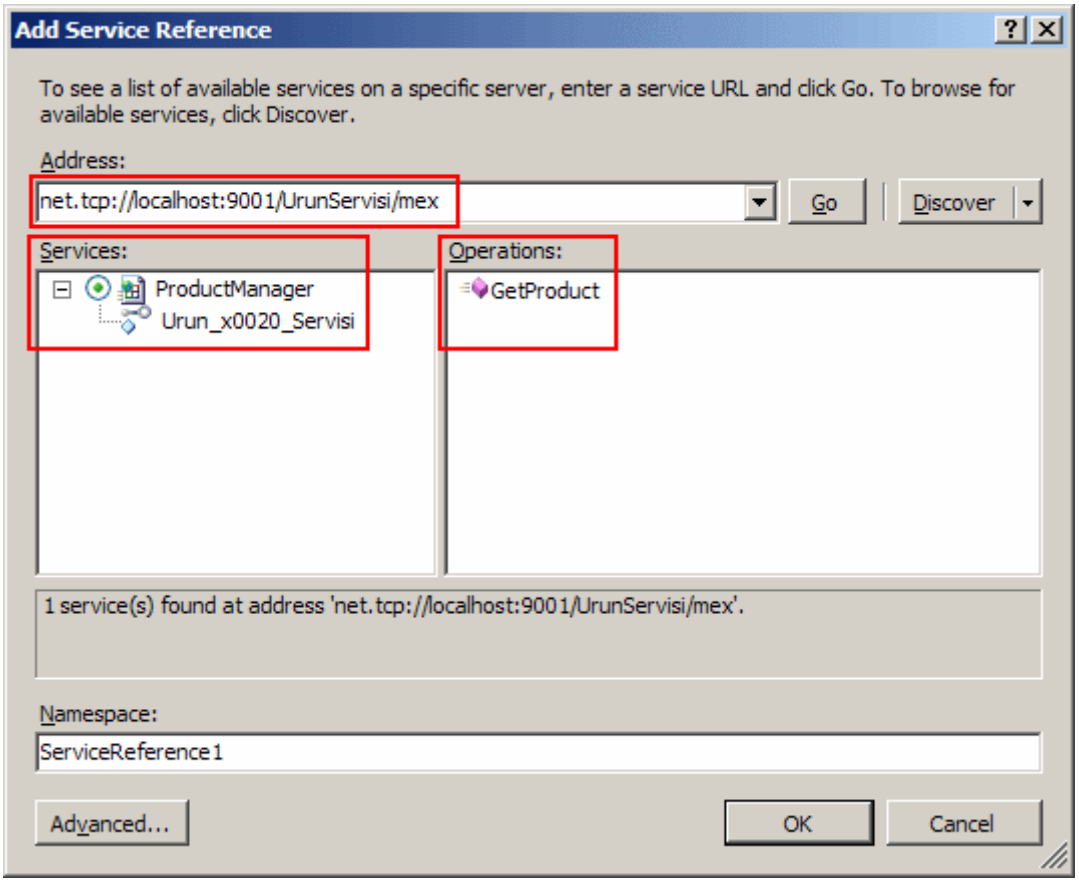
Programın çalışmasının sonucu aşağıdakine benzer bir ekran çıktısı oluşacaktır.



Dikkat edilmesi ve unutulmaması gereken noktalardan biriside programın çalışması için servisinde çalışıyor olması gerekliliğidir. Nitekim servisin ayakta olmaması halinde istemcilerin taleplerine karşılık **çalışma zamanı istisnaları(Runtime Exception)** alması söz konusudur.

Peki **ClientOperation** kullandığımızda **.Net 3.5** sürümü ile Workflow Foundation alt yapısına kazandırılan **SendActivity** tipine göre en büyük farklılık(farklılıklar) nedir?

NOT : *Aşağıdaki ekran görüntüsü* ***17.4.2008*** *tarihinde yayınlanan* ***WCF ile WF Entegrasyonu -1*** *makalesine aittir. Bu makaleden hatırlanacağı üzere* ***SendActivity*** *tarafından kullanılan* ***WCF*** *servisinin* ***Host*** *uygulama üzerinde ela alınabilmesi için* ***Add Service Reference(veya svcutil ile komut satırından)*** *ile* ***proxy*** *üretiminin yapılması gerekmektedir.*



Herşeyden önce servis kullanan bir istemci geliştirilirken, **Add Service Reference** seçeneği ile **proxy** tiplerinin eklenmesi gerekmektedir. Oysaki geliştirdiğimiz örnekte böyle bir işleme başvurulmamıştır. Bunun yerine **ClientOperation** elementi için **XAML** tabanlı tanımlamalar yapılmıştır. Dolayısıyla yeni çalışma zamanı motorunun bu içeriğe bakaraktan servis ile iletişime geçtiğini ve fiziki bir proxy' ye ihtiyaç duymadığını söyleyebiliriz. Bir başka deyişle örneğin servis tarafında oluşabilecek bazı değişikliklerin **WF** tarafına bildirilmesi için bir referans güncellemesi yapılmasına gerek kalmamaktadır. Buda önemli bir avantajdır.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde yeni **WF 4.0** ve **WCF 4.0** kabiliyetlerinin örnek bir sonucunu değerlendirmeye çalıştık. Bu amaçla yeni gelen aktivite tiplerinden**ClientOperation** tipini ele aldık ve bir servis operasyonunun referansını eklemeden, dekleratif olarak tanımlanıp **XAML** ile oluşturulmasını ve kullanılmasını gördük. Tabi bu konuda konuşulabilecek farklı vakalarda vardır. En önemli sorunlardan biriside servis operasyonlarının uzun süreli**(Long Running Workflows)** olabileceğidir. Bu durumda **WF**' in **kalıcı olaraksaklanması(Persistence)** gibi durumlar söz konusudur ki bunu **WF 4.0** üzerinde kurmak ve **yönetmek(Management)** son derece kolaydır. Özellikle yönetim ve izleme safhasında devreye giren **Dublin**kod adlı **Windows Application Server**' ın yeteneklerini en kısa sürede sizinle paylaşmaya çalışıyor olacağım. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

WF 4.0 - İlk İzlenimler [Pre Beta] (2009-03-26T04:55:00)

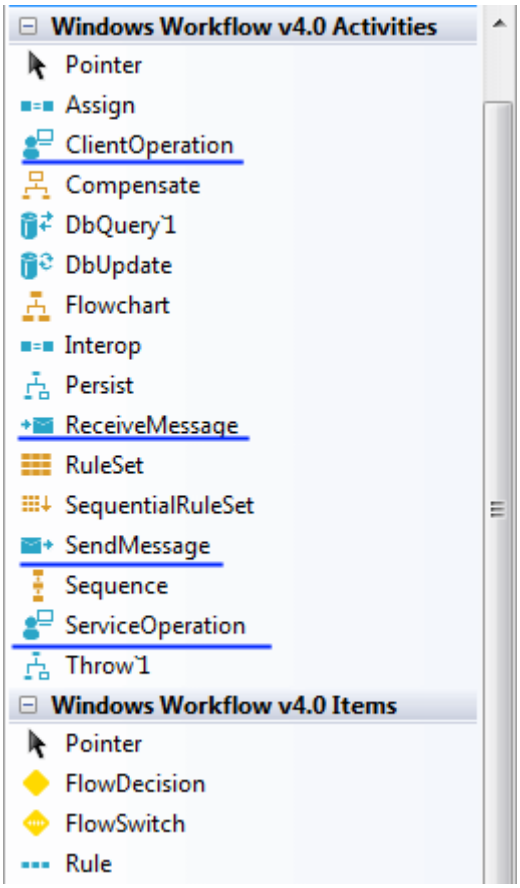
wf 4.0,

Bundan sadece bir kaç sene önce **.Net Framework 3.0** versiyonu ile birlikte iş akışlarının(Workflows) kod içerisinde modellenerek farklı uygulamalarda kullanılabilmesini sağlamak amacıyla**Windows Workflow Foundation(WWF)** alt yapısı duyurulmuştu. Paralelinde ise, **Servis Odaklı Mimarilere(Service Oriented Architecture)** yeni bir yaklaşım, **Windows Communication Foundation(WCF)** ile birlikte getirilmişti. **Workflow Foundation** her ne kadar iş akışlarının*(çoğu zaman kod akışlarının)* kendi içinde modellenmesini sağlasa da, zaman içerisinde dış ortamlar ile olan haberleşmesinde **WCF** ile birlikte hareket etmeye başlamıştır. Bu nedenle **.Net Framework 3.5** ile birlikte her iki alt yapınında birbirleriyle daha kolay haberleşebilmesi için bazı eklentiler yapılmıştır. Bu genişletmelerden en önemlileri **Workflow Activity Library** içerisine eklenen **SendActivity** ve **ReceiveActivity** aktivite tipleridir. Böylece bir **Workflow** uygulamasının servisler yardımıyla dış dünya ile **tek yönlü(One-Way)** mesajlaşabilmesi yada kendi içerisinden dış dünyaya servis bazlı operasyonlar sunabilmesi mümkün hale gelmiştir. Ama belkide en önemli genişletme **WorkflowServiceHost** sınıfıdır. Bu sınıf sayesinde, WF uygulamalarının servis olarak host edilebilmesi, bir başka deyişle **IIS/WAS(Internet Information Services/Windows Process Activation Service)** içerisinde sunulması mümkün kılınmaktadır. Sonuç olarak artık günümüzde, **Workflow Tabanlı Servisler(Workflow Based Services)**kavramı hayatımızın bir parçası haline gelmeye başlamıştır. Yinede özellikle geliştirici açısından bakıldığında halen daha eksiklikler bulunmaktadır. özellikle **WF** ile **WCF**entegrasyonunda karşılaşılan bu zorlukların üstesinden gelebilmek için **.Net Framework 4.0** içerisinde önemli yenilikler bulunmaktadır. İşte bu yazımızda halen son sürümü ile yayınlanmamış olsada **WF 4.0** ile gelmesi muhtemel yeniliklere değinilmeye çalışılacaktır.

***NOT :** Yazımızda yer alan örnekler ve teknik terimlerin çoğu **PDC 2008**' de yayımlanan **Virtual PC** imajı üzerinde gerçekleştirilmektedir. Mayıs ayında beta sürümünün yayımlanması planlanan **Visual Studio 2010**' un ilk görüntüsü ele alınmaktadır. Bu nedenle son sürümler yayımlandıktan sonra makalede yer alan kavramların bir kısmının değiştiği görülebilir ve muhtemeldir.*

WF uygulamaları **3.5** sürümünde **XAML(eXtensible Application Markup Language)** yardımıyla yapılabilsede yinede geliştirici açısından tam anlamıyla oturmuş bir yapı değildi. örneğin debug edilmelerinde sorundu. Ancak **4.0** sürümünde sadece **XAML** bazlı **Workflow** örneklerinin geliştirilmesi çok daha kolay bir hale getirilmektedir. Aslında buradaki en büyük amaçlardan biriside, **Workflow tabanlı WCF servislerinin**, **XAML** bazlı olaraktan **dekleratif(Declerative)** tanımlanabilmesinin sağlanmasıdır. Dekleratif şekilde yapılan tanımlamalar, kodlamaya girmeden çalışma zamanında bazı değişikliklerin kolayca yapılabilmesini sağlamaktadır. Dolayısıyla **4.0** sürümünde **WF** tarafında ve **WCF** tarafında **XAML** yapısını çok daha yaygın bir şekilde görüyor ve kullanıyor olacağız. **XAML** tabanlı bu içerikler herhangibir

depolama alanında*(Muhtemelen **Oslo** kod adlı yapının değerlendireceği **saklama-Repository** alanlarında)* saklanabileceği gibi, çalışma zamanına devredilerek yürütülebileceklerde. Yani bir WF uygulamasının*(hatta bir WCF servisinin ve çok doğal olarak bir Workflow servisinin)* bulunduğu ortamdan **Export** edilerek başka bir platforma taşınması ve orada **Import** edilerek yürütülmeye başlanması mümkün olabilecek. Diğer taraftan depolanarak saklanan bu **Workflow Servislerinin** veya diğer **WCF** servislerinin kolayca **yönetilebilmesi(Management)** içinde **Dublin(Windows Application Server)** kod adlı bir çalışma yürütülmektedir. Bu sunucu ve **IIS**' e gelen eklentiler sayesinde, servislerin kolay bir şekilde **Import/ Export** edilmesi, izlenebilmesi**(Tracking)**, durumlarının denetlenmesi**(Monitoring)** gibi Admin seviyesindeki işlemler kolaylıkla gerçekleştirilebilecektir. Tabiki bu konular ile ilişkili detayları önümüzdeki zamanlarda işlemeye çalışıyor olacağız. **XAML** tabanlı **WF** örnekleri,**4.0** ile gelen yeniliklerden sadece bir tanesi. Bunun dışında aşağıdaki şekildende görebileceğiniz gibi pek çok yeni aktivite tipi ile karşı karşıyayız.

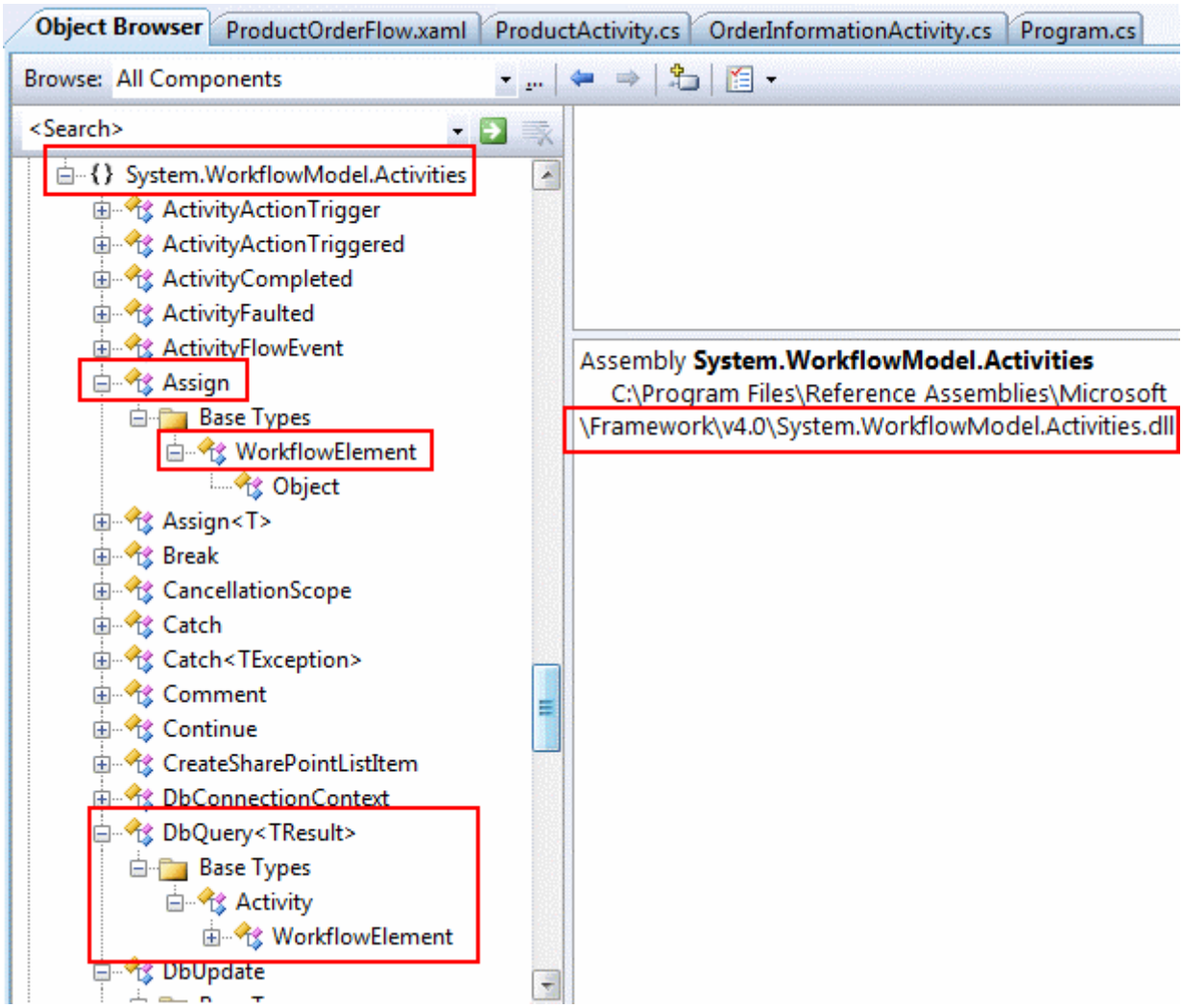


Görüldüğü üzere pek çok farklı aktivite tipi yer almakta. Bu şekilde altı mavi çizgi ile işaretlenmiş olan aktiviteler*(veya Workflow elementleri)* **WCF** odaklı bileşenlerdir. örneğin**ClientOperation** bileşeni ile,
bir **WCF** operasyonunu **SendMessage/ReceiveMessage** yapısına uygun olacak şekilde çağırmak mümkün olmaktadır. **ServiceOperation** bileşeni ile **WF**içerisinden dışarıyı bir **WCF** operasyonu(Operation) yine **SendMessage/ReceiveMessage** yapısına uygun olaraktan sunulabilmektedir. **SendMessage** aynen **3.5**' teki **SendActivity** aktivitesine benzer olacak şekilde **tek yönlü mesaj(One Way Message)** gönderilmesinde kullanılır.

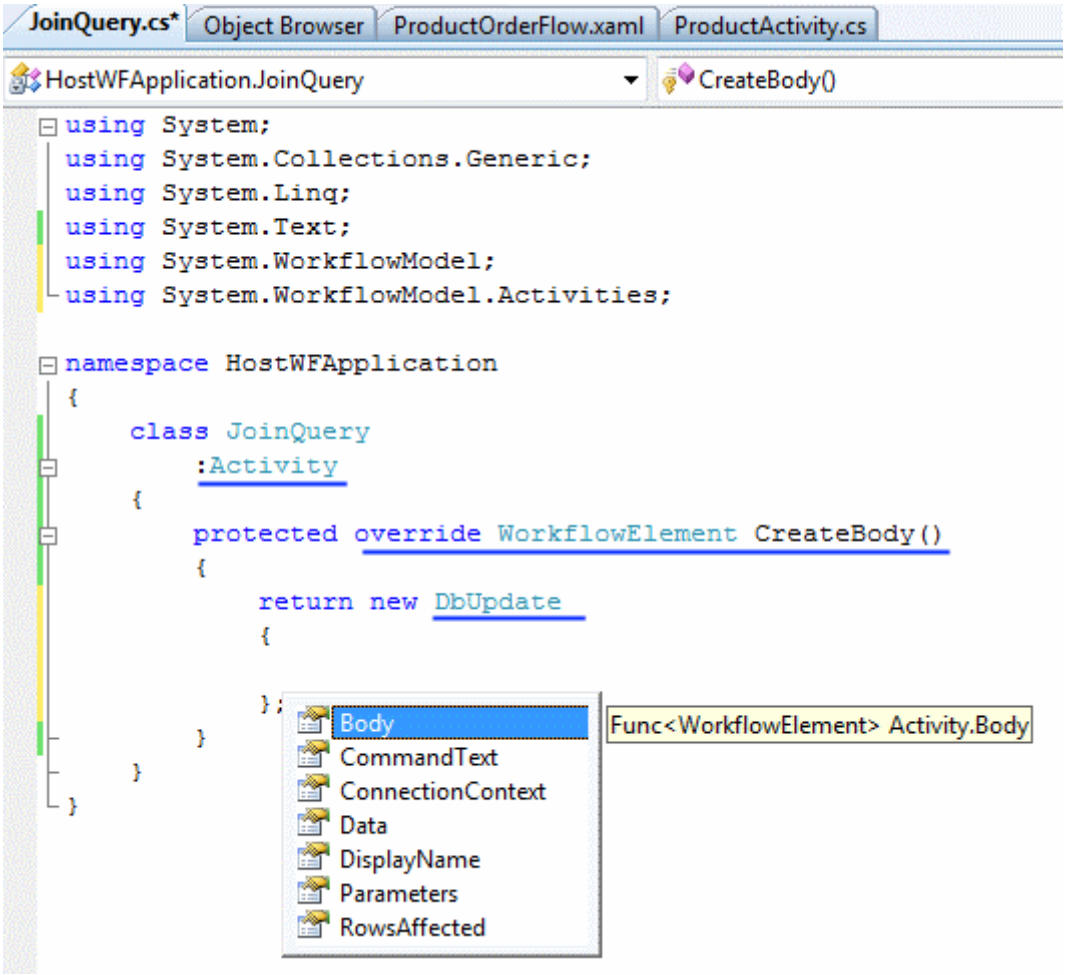
Tahmin edileceği üzere **ReceiveMessage** bileşenide tek yönlü olaraktan **WCF** mesajlarının alınmasında kullanılmaktadır.

WCF tabanlı örnek bu bileşenler dışında dikkat çekici noktalardan biriside **FlowChart** aktivite tipidir. Bu bileşen yardımıyla akış diagramı mantığında basit karar yapıları ve anahtar adımlar ile süreçlerin kolayca tasarlanması mümkündür. Geliştiriciler açısından çok yaygın olarak kullanılabilecek bir akış tipidir. Bu akış tipi **Sequential** ve **StateMachine** aktivite tiplerinin bazı yanlarını kendi içerisinde barındırmaktadır. Buna ek olarak örneğin **Assign** bileşeni ile **workflow** seviyesindeki bir değişkene değer atanması sağlanabilir. İlgi çekici diğer aktiviteler ise **DbQuery'1**, **DbUpdate** ve **Persist** bileşenleridir. Aslında **DbQuery'1** bileşeni yardımıyla **SQL** sorgularının çalıştırılması ve **DbUpdate**ile veri kaynağına doğru güncelleştirmelerin yapılması mümkündür. Bu belki çok ekstra bir özellikmiş gibi gelmeyebilir ama önemli olan nokta söz konusu aktivitelere ait ayarlamaların dekleratif olarak*(yani XAML bazlı)* yapılabilmesidir. **Persist** bileşeni ise, **Workflow**' un herhangibir noktasında **persistence** hizmetinin devreye alınarak akışın uzun süreliğine korunabilmesini sağlamaktadır ki **Long Running Workflow Service** tipleri için önemli bir özelliktir. Bunu 3.5 versiyonunda kod yardımıyla yaptığımız düşünülürse bir aktivite bileşeninin olması geliştirime safhasını kolaylaştırmakta ve yönetimi dahada güçlendirmektedir. *(Bu aktivite tiplerini ve nasıl kullanıldıklarını ilerleyen görsel derslerimizde ele almaya çalışıyor olacağım)*

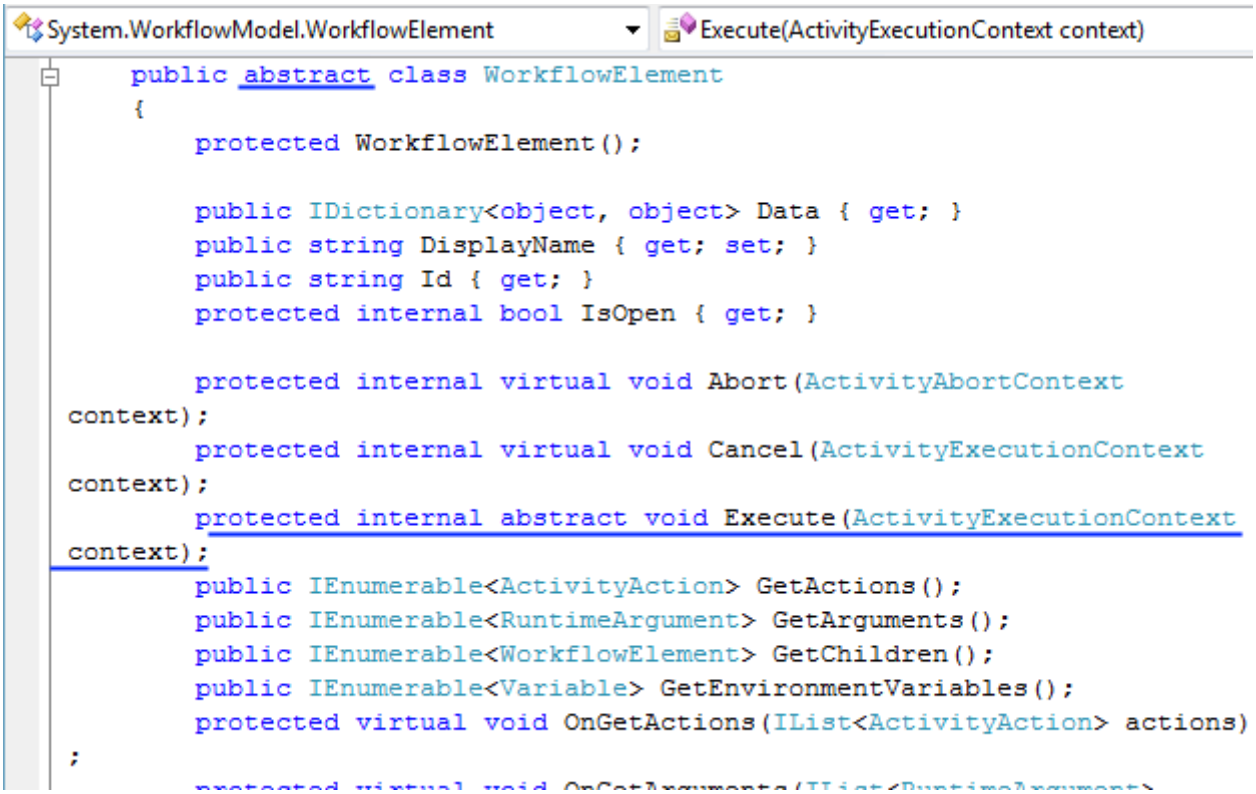
Gelelim aktivite kütüphanesine. Artık aktivitelerimiz **WorkflowElement** isimli yeni bir tipten türetilmektedir. Bir başka deyişle **Base Activity Library**' de yer **WF** elementlerin ata sınıfı**WorkflowElement** tipi olmaktadır. Yine **Visual Studio 2010 Object Browser**' dan alınan ekran görüntüsünde bu durum aşağıdaki şekildende görüldüğü gibi tespit edilebilir.



Dikkat edileceği üzere **DbQuery** aktivitesi, **Activity** tipinden türemiş olmasına rağmen, **Activity** tipinin kendisi **WorkflowElement** tipinden türediği için dolaylı olarak bir **WorkflowElement**' tir.**Assign** bileşeni ise doğrudan **WorkflowElement** tipinden türemektedir. Burada aslında önemli bir noktayı daha vurgulamak gerekiyor. Eğer var olan bir aktivite türünden genişletme yapılarak yeni bir bileşen üretilecekse, **Activity** tipinden türetilme yapılması ve bu yeni sınıf içerisinde, geriye **WorkflowElement** referansı döndüren **CreateBody** metodunun ezilmesi(override) önerilmektedir. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.

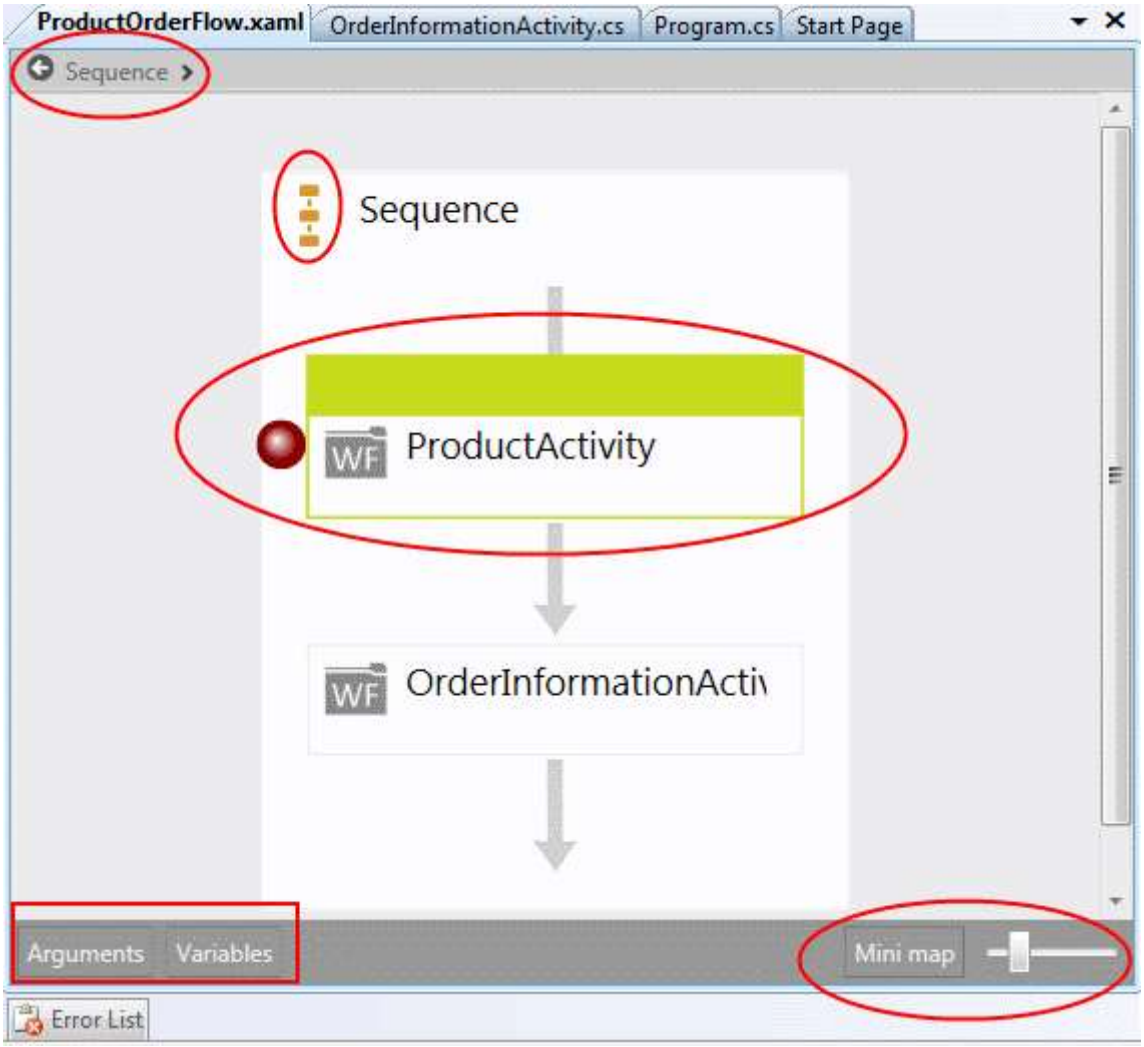


Diğer taraftan eğer sıfırdan bir aktivite tipi tasarlanacaksa **WorkflowElement** tipinden türetip **Execute** metodunun **override** edilmesi gerekmektedir. **WorkflowElement** aslında **abstract** bir sınıf ve**Execute** metodu **abstract** olarak tanımlanmıştır. Bu nedenle **implemente** eden tip içerisinde **Execute** metodunun mutlaka ezilmesi şarttır.***(C# Object Oriented** kurallarını hatırlayalım)*



Execute metodu parametre olarak **ActivityExecutionContext** tipinden bir referans değeri almaktadır. Bu sayede söz konusu aktivitenin çalışma zamanında içinde bulunduğu **Activite** ile konuşabilmesi mümkün olmaktadır. Aktivite tasarlanması ile ilişkili önemli noktalardan biriside dış ortama değişken aktarımı veya tam tersidir. Bu noktada **InArgument<T>** ve**OutArgument<T>** tiplerinden yararlanılarak, aktivite içerisine veri girişi veya aktiviteden dış ortama veri çıkışı sağlanabilmektedir. Biraz sonra geliştireceğimiz örneğimizde iki özel aktivite tipi yazarken bu **generic** sınıflardan yararlanılacaktır.

WF 4.0 akışlarının kolay bir şekilde tasarlanabilmesi için **Visual Studio 2010** içerisinde **WPF(Windows Presentation Foundation)** tabanlı bir arayüz sunulmaktadır. Bu arayüz sayesinde akışların daha zengin bir görsellikle hazırlanması mümkündür. Söz gelimi **zoom** özelliği ile büyük akışların ekran içerisinde daha verimli şekilde görülebilmesi sağlanmaktadır. Zannediyorum aşağıdaki ekran görüntüsünde bu durumun kafamızda biraz daha netleşmesi için yeterlidir.



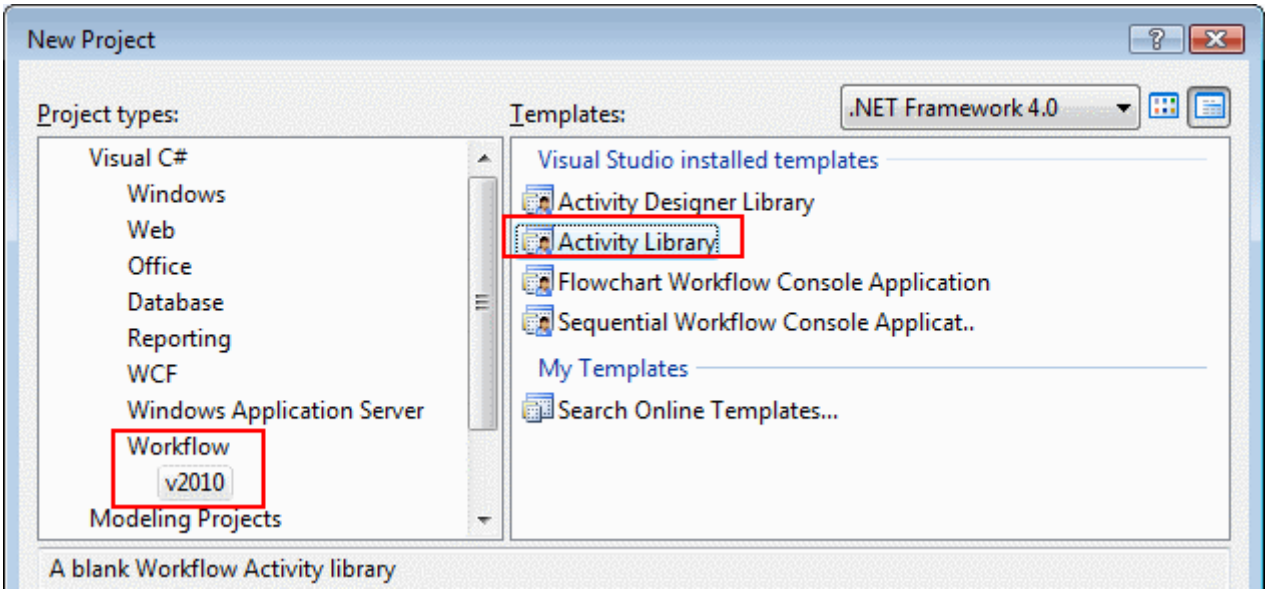
Dikkat çekici noktalardan bir diğeride örnekte yer alan ProductActivity bileşeni yanında bir **Breakpoint** yer almasıdır. Dolayısıyla **debug** işlemleri için **Workflow** elementlerinin kendisinin**designer** içerisinde doğrudan işaretlenebildiğini söyleyebiliriz. Diğer dikkat çekici kısımda sol alt tarafta yer alan **Arguments** ve **Variables** bölümleridir. Bugün yazımızda **Variables** kısmını kullanarak Sequence aktivitesinin tamamını ilgilendiren değişkenlerin nasıl tanımlanabileceğini ve kullanılabileceğinide görmüş olacağız. önemli noktalarda biriside Arguments veya Variables gibi bölümler ile Workflow içerisinde çeşitli tanımlamaların koda girmeye gerek duymadan kolay bir şekilde görsel olarak yapılabilmesidir. *(Designer tarafının kullanımını daha kolay kavramak için yayınlanacak görsel derslerimizi takip etmenizi öneririm.)*

***NOT : Variables** ve **Arguments**;*

***WF 4.0**' da **Variables** kavramı verinin depolanmasını ifade eder. Değişkenler(Variables) **Workflow**' un(yada **Activity** bileşeninin) çalışma zamanı örneği boyunca veri depolamak amacıyla kullanılırlar. Tanımlanırken adları ve tipleri belirtilir. Yaşamları, **Workflow** veya **Activity** örneğinin bellekte kaldığı süre boyunca geçerlidir. Yani **Workflow** sonlandığında veya **Activity** bileşeni çalışmasını tamamladığında referansları bellekten kaldırılmaktadır.*

Arguments *kavramı ise verinin aktivite içerisine veya aktivite içerisinden dışarıya aktarılması anlamında kullanılmaktadır. Her argümanın bir yönü****(Direction)*** *vardır.****(Input, Output, Input/Output)*** *Argümanlar aktivite içerisinde tanımlanırken* ***InArgument<T>, OutArgument<T>*** *veya* ***InOutArgument<T>*** *tipinden tanımlanırlar.* ***T*** *ile kullanılacak verinin tipi belirtlir.*

Dilerseniz basit bir örnek geliştirerek yenilikleri daha yakından tanımaya çalışalım. öncelikle **Visual Studio 2010** ortamında **.Net Framework 4.0** şablonunda bir **Activity Library** geliştirerel işe başlayacağız. Bu kütüphane örneğimizde kullanacağımız özel aktivite tiplerini barındırıyor olacak.



Bu işlemin ardından projemize bir adet sınıf ekleyerek devam edebiliriz. Sınıfımız içeriği ise aşağıdaki gibidir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.WorkflowModel;

namespace SampleActiviyLibrary
{
   public class ProductActivity
      :WorkflowElement
   {
      public OutArgument<int> ProductId { get; set; }
      public OutArgument<int> Count { get; set; }

      protected override void Execute(ActivityExecutionContext context)
```

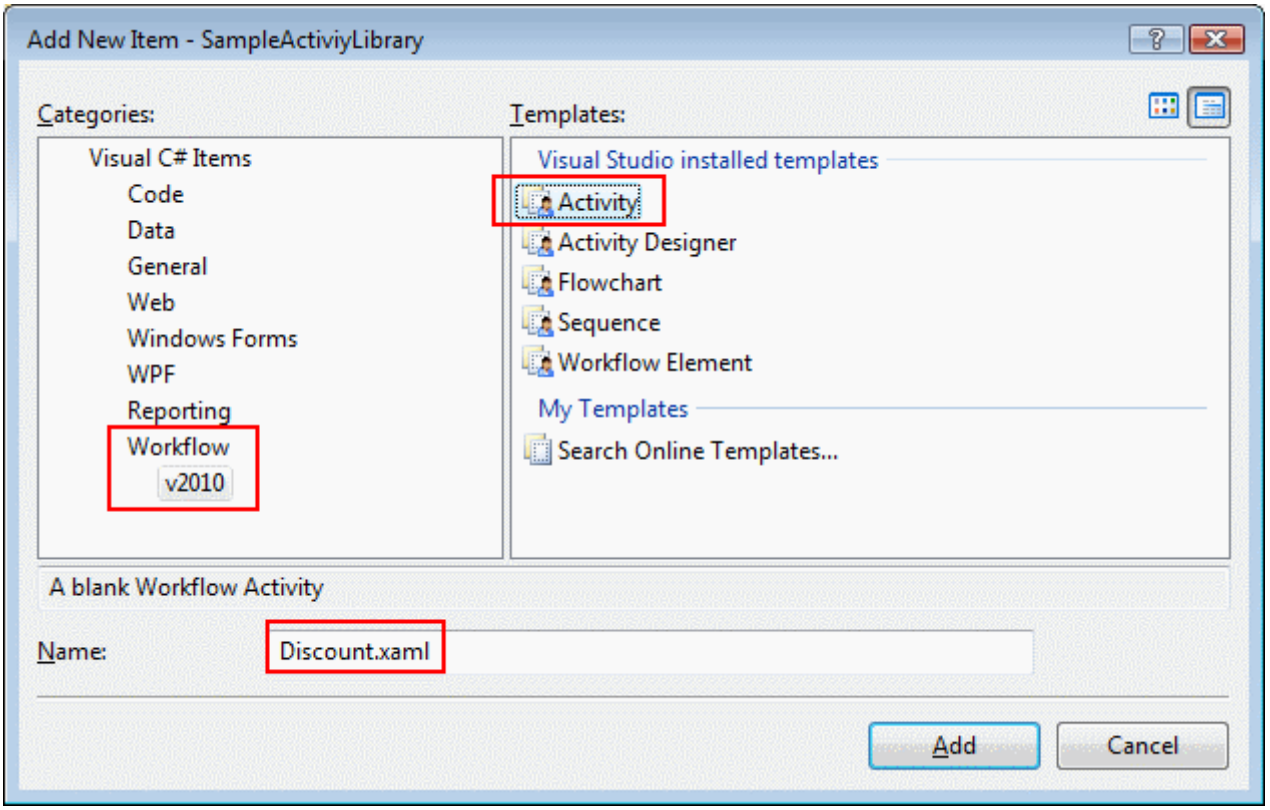
```
      {
         Console.WriteLine("ProductId ?");
         ProductId.Set(context, Convert.ToInt32(Console.ReadLine()));
         Console.WriteLine("Requested count ?");
         Count.Set(context, Convert.ToInt32(Console.ReadLine()));
      }
   }
}
```

Burada dikkat edeceğiniz üzere kod yardımıyla özel bir **aktivite tipi(Custom Activity)** geliştirilmektedir. **ProductActivity** sınıfı **WorkflowElement** tipinden türemektedir ve içerisinde**OutArgument<int>** tipinden **ProductId** ve **Count** isimli **özellikler(Properties)** yer almaktadır. **Override** edilen **Execute** metodu içerisinde, **OutArgument<T>** tipinin **Set** metodundan yararlanılarak, **Console**penceresinden okunan değerlerin **ProductId** ve **Count** özelliklerine aktarılması sağlanır. **OutArgument<T>** kullanılması nedeniyle, özelliklerin değerleri **ProductActivity** aktivitesinin kullanıldığı **Workflow** ortamına aktarılabilmektedir. Yani bu aktivite içerisinde **Console** penceresinden alınan bir takım değerler*(ki bu sadece örnek olarak verilmiştir, gerçek hayat senaryolarında bu değerlerin farklı kaynaklardan gelmesi muhtemeldir.)*, aktivitenin kullanıldığı **Workflow** içeriğine **ProductId** ve **Count** isimleriyle taşınabilir. Gelelim ikinci özel aktivitemize.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.WorkflowModel;

namespace SampleActiviyLibrary
{
   public class OrderInformationActivity
      :WorkflowElement
   {
      public InArgument<int> OrderedProductId { get; set; }
      public InArgument<int> OrderedProductCount { get; set; }

      protected override void Execute(ActivityExecutionContext context)
      {
         int orderedId=OrderedProductId.Get<int>(context);
         int orderedCount = OrderedProductCount.Get<int>(context);

         // TODO: Mail gönderme işlemleri.
         Console.WriteLine("{0} numaralı üründen {1} adet siparis
```

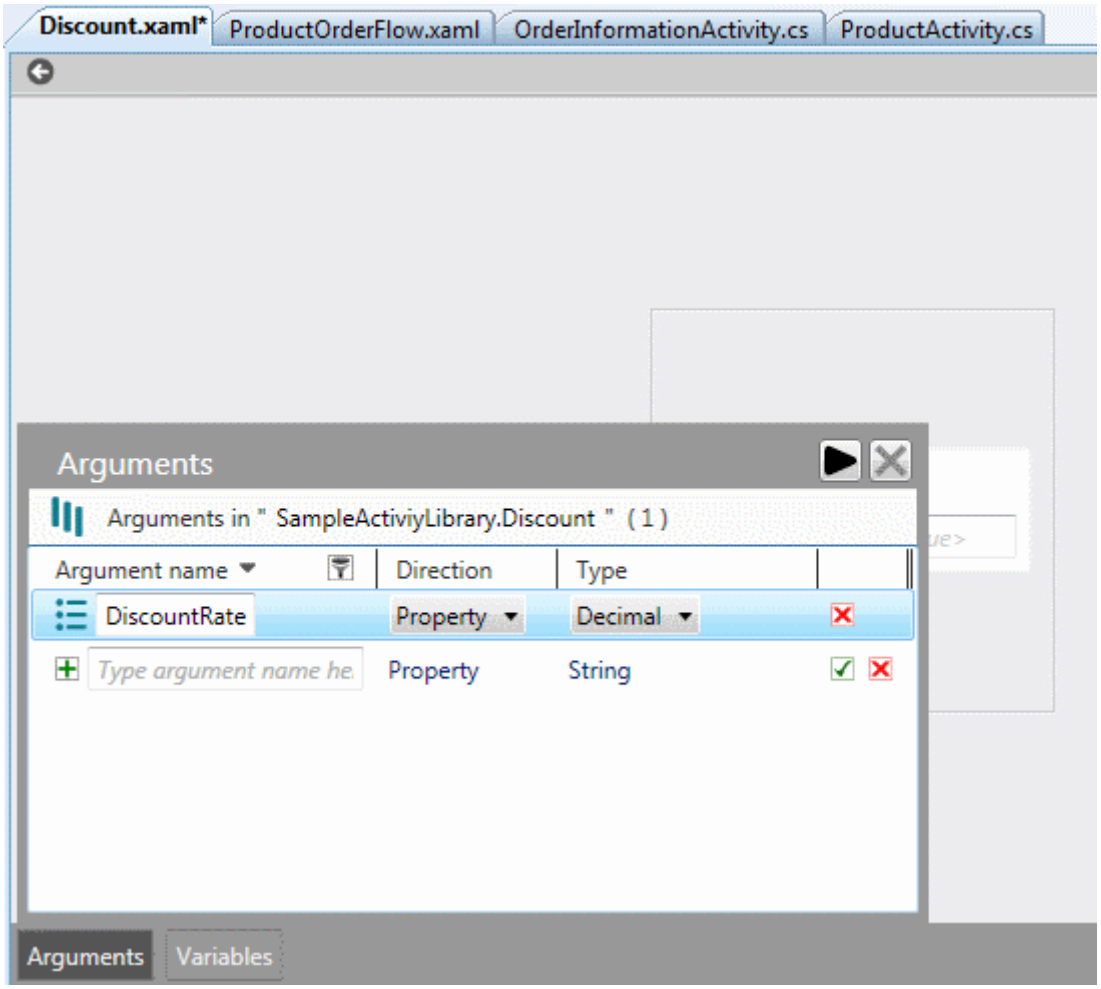
```
edilmistir",orderedId,orderedCount);
        }
    }
}
```

OrderInformationActivity bileşenide **ProductActivity** sınıfı gibi **WorkflowElement** tipinden türemektedir. Ancak bu kez **InArgument<int>** tipinden **OrderedProductId** ve **OrderedProductCount** isimli özellikler kullanılmaktadır. Yani, **Workflow** içerisinden bu aktiviteye, InArgument<T> tipinden tanımlanan özellikler üzerinden veri taşınabilir. Girilen değerlerin **Execute** metodu içerisinde elde edilişi sırasında **InArgument<T>** tipinin generic **Get<T>** metodu kullanılır. Dikkat edileceği üzere metodlar parametre olarak **ActivityExecutionContext** tipinden referans almaktadır. Yani, aktivitenin kullanıldığı **Workflow** ortamına ait bir takım bilgiler*(örneğin **InstanceId** gibi)* ***Execute*** metodu içerisinde taşınabilmektedir. Bu aktivite ilede, sembolik olarak **Console** penceresine sipariş edilen ürün numarası ve miktarı bilgileri yazdırılmakta ve belkide mail gönderme işlemleri gerçekleştirilmektedir. **Workflow** örneğimizi tasarlamadan önce arada **XAML** bazlı bir aktivite tipinin nasıl geliştirileceğine de değinmek isterim. Bu amaçla **Visual Studio 2010** ortamında, projeye **Add New Item** ile bir **Activity** öğesi aşağıdaki şekilde görüldüğü gibi eklenir.



Dikkat edileceği üzere aktivitenin uzantısı **XAML**' dır. Bununla birlikte açılan pencereden görüleceği gibi **Activity** tipi görsel olarak tasarlanabilir. Buradaki aktivitede aslında birde

özellik tanımlanmaktadır. **DiscountRate** isimli bu özellik **Argument** penceresi kullanılaraktan aşağıdaki gibi oluşturulmuştur! Süper.



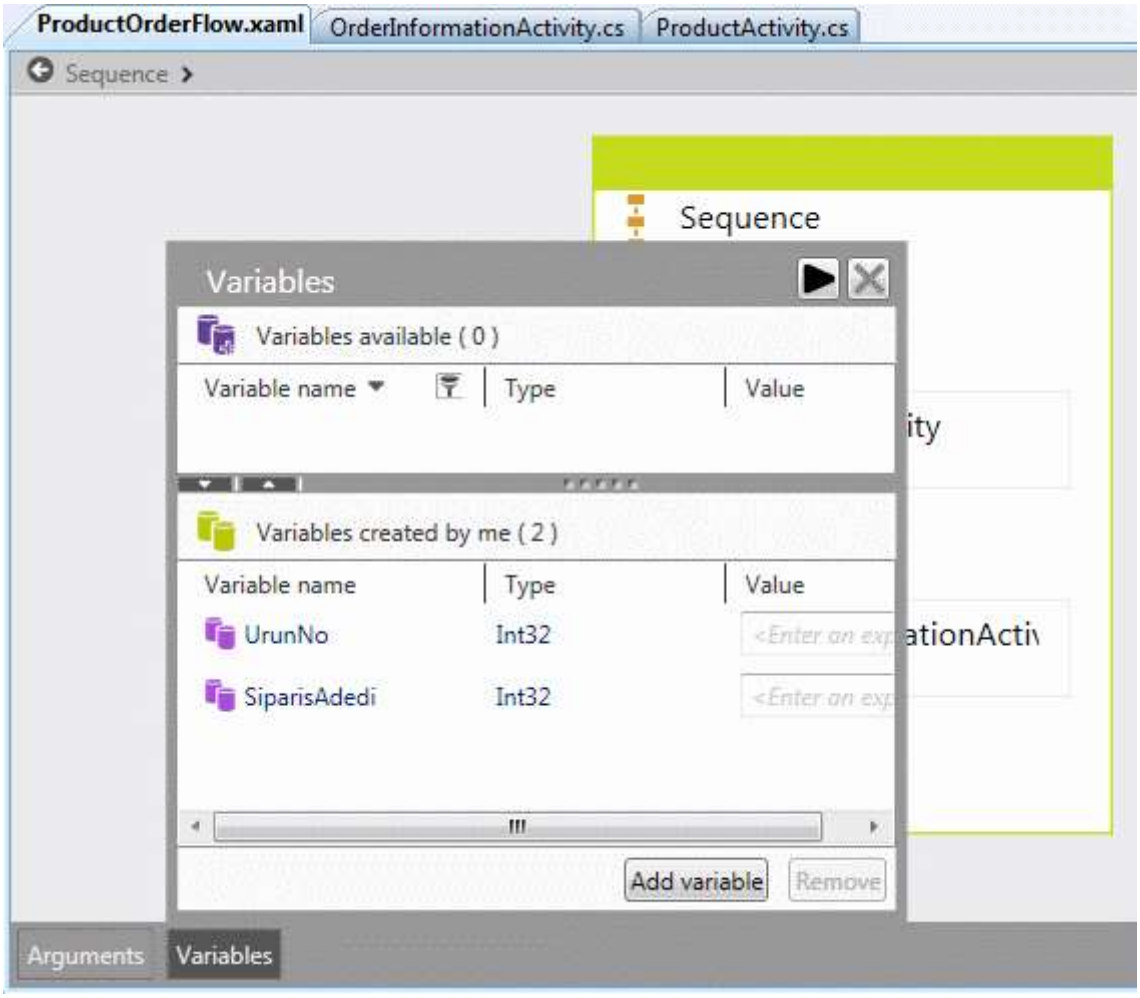
Görüldüğü gibi herhangibir şekilde kod yazılmamıştır. Ardından tasarım ortamında basit bir **Assign** bileşeni sürüklenerek, **Discount** aktivitemizde aşağıda görüldüğü gibi kullanılması sağlanmaktadır.

Assign aktivitesinin yaptığı tek şey **DiscountRate** isimli özelliğe sabit bir değerin atanmasının sağlanmasıdır. Ancak burada önemli olan noktalar **Discount** bileşeninin tamamen görsel olarak tasarlanması ve sonucun **XAML** olarak aşağıdaki gibi üretilmesidir.

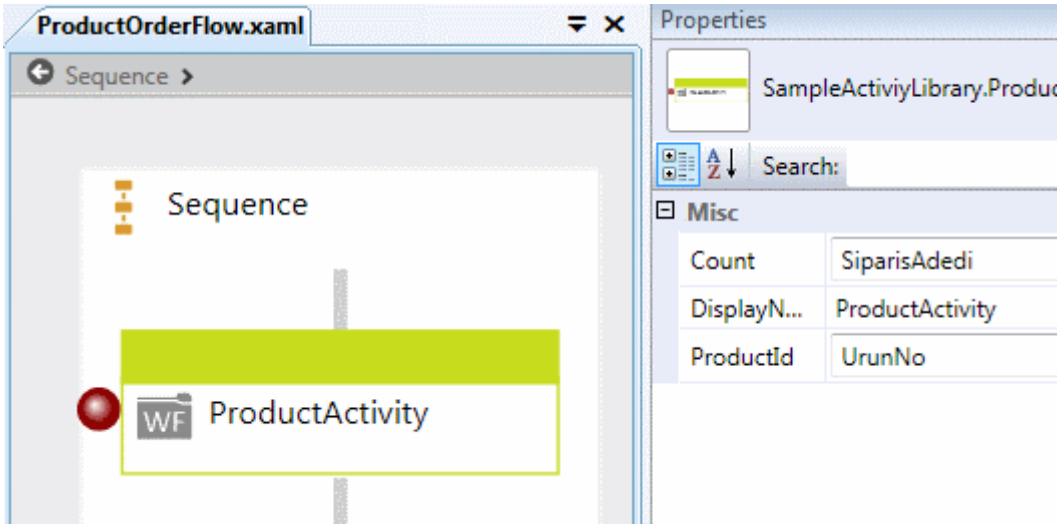
```
<p1:Activity x:Class="SampleActiviyLibrary.Discount" xmlns:p="http://schemas.microsoft.com/netfx/2008/xaml/schema"
xmlns:p1="http://schemas.microsoft.com/netfx/2009/xaml/workflowmodel" xmlns:s="clr-namespace:System;assembly=mscorlib" xmlns:swdx="clr-namespace:System.WorkflowModel.Design.Xaml;assembly=System.WorkflowModel.Design" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
  <p:SchemaType.Members>
    <p:SchemaProperty Name="DiscountRate" Type="s:Decimal" />
  </p:SchemaType.Members>
  <p1:Assign>
    <p1:Assign.To>
      <p1:OutArgument x:TypeArguments="s:Decimal">[DiscountRate]</p1:OutArgument>
    </p1:Assign.To>
    <p1:Assign.Value>
      <p1:InArgument x:TypeArguments="p:Double">[1.12R]</p1:InArgument>
    </p1:Assign.Value>
  </p1:Assign>
</p1:Activity>
```

Görüldüğü gibi, tasarım zamanında geliştirdiğimiz bu aktivite tipi **sadece XAML(Only XAML)** içerikli olacak şekilde üretilmiştir. İşte **dekleratif** tanımla dediğimizde tam olarak budur. üretilen bu **XAML** içeriği herhangibir ortamda depolanabilir ve dahada önemlisi **WF çalışma zamanı** tarafından yürütülerek başka **Workflow**' lar tarafından ele alınıp kullanılabilir.

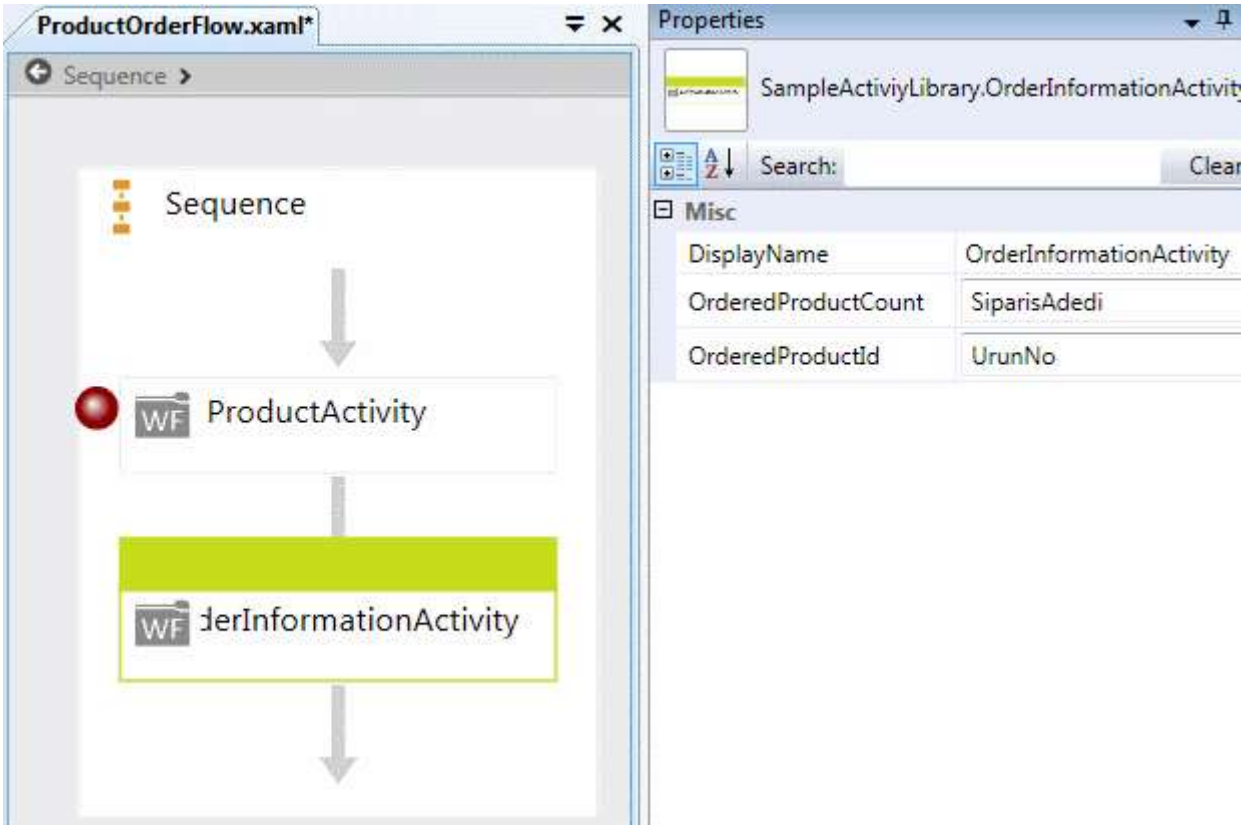
Bu kısa bilgilerden sonra tekrar örneğimize dönebiliriz. Artık geliştirdiğimiz **Workflow** kütüphanesini basit bir **Sequential Workflow Console Application** projesinde referans ederek kullanacağız. **Console** uygulamasına ekleyeceğimiz **ProductOrderFlow.xaml** isimli aktivite içerisinde bir **Sequence** bileşeni bulunmaktadır. Bu bileşen içinde ise ürün numarası ve sipariş adedi bilgileri için iki değişken tanımlanmaktadır. Bu sefer **Variables** kısmını kullanaraktan aşağıdaki ekran görüntüsünde olduğu gibi gerekli değişkenleri kolayca belirleyebiliriz.



UrunNo ve **SiparisAdedi** isimli **Int32** tipinden olan değişkenler **Sequence** aktivitesi içerisinde tanımlandıklarından, alt aktiviteler tarafındanda erişilip kullanılabilmektedirler. Şimdi Sequence aktivitesi içerisine
önce **ProductActivity** sonrada **OrderInformationActivity** bileşenlerini ekleyeceğiz. **ProductActivity** bileşeni hatırlayacağınız gibi **Console** penceresinden okuduğu değerleri**ProductId** ve **Count** isimli **output** özelliklerine aktarmaktaydı. Dolayısıyla bu değerleri **Sequence** içerisinde tanımlanan **UrunNo** ve **SiparisAdedi** isimli değişkenler ile eşleştirmemiz mümkündür. Bunun için tek yapılması gereken, **ProductActivity** bileşeninin özelliklerinden ilgili atamaların aşağıdaki ekran görüntüsünde olduğu gibi yapılmasıdır.



ProductActivity bileşeninin hemen arkasına **OrderInformationActivity** bileşenini ekleyerek devam edebiliriz. Bu bileşende hatırlayacağınız gibi kullanıldığı ortamdan **girdi(Input)**değerleri almak üzere iki özelliğe sahiptir. Bu nedenle söz konusu aktivitenin özelliklerinde aşağıdaki ekran görüntüsünde yer alan ayarları yapmamız yeterlidir.



Böylece bir önceki aktivite ile, **Sequence** aktivitesindeki **SiparisAdedi** ve **UrunNo** isimli değişkenlere taşınan değerler, **OrderInformationActivity** içerisinden elde edilebilirler. İşte bu kadar. Tabi şimdilik :) Tüm bu işlemlerin
arından **ProductOrderFlow.xaml** aktivitesine ait **XAML** içeriğine bakıldığında, aşağıdaki kod parçasında yer alan çıktının elde edildiği görülebilir.

```
<p:Activity
x:Class="HostWFApplication.ProductOrderFlow" xmlns:p="http://schemas.microsoft.
com/netfx/2009/xaml/workflowmodel" xmlns:p1="http://schemas.microsoft.com/netfx/
2008/xaml/schema"xmlns:s="clr-namespace:SampleActiviyLibrary;assembly=
SampleActiviyLibrary" xmlns:swd="clr-
namespace:System.WorkflowModel.Debugger;assembly= System.WorkflowModel"
xmlns:swdx="clr-namespace:System.WorkflowModel.Design.Xaml;assembly=
System.WorkflowModel.Design"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
   <p:Sequence swd:XamlDebuggerXmlReader.FileName="C:\Orneklerim\SampleActiviy
Library \HostWFApplication\ProductOrderFlow.xaml">
      <p:Sequence.Variables>
         <p:Variable x:TypeArguments="p1:Int32" Name="UrunNo" />
         <p:Variable x:TypeArguments="p1:Int32" Name="SiparisAdedi" />
      </p:Sequence.Variables>
      <s:ProductActivity Count="[SiparisAdedi]" ProductId="[UrunNo]" />
      <s:OrderInformationActivity OrderedProductCount="[SiparisAdedi]"
OrderedProductId="[UrunNo]" />
   </p:Sequence>
</p:Activity>
```

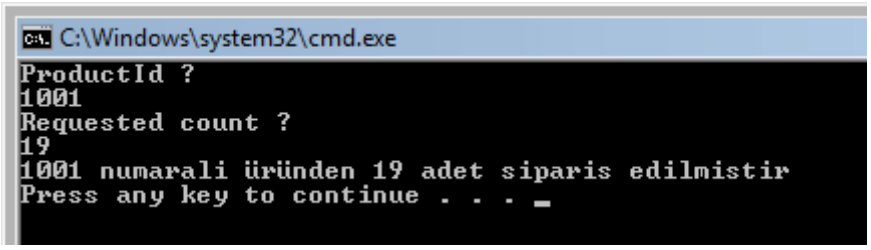
Harika değil mi? Tüm akış içeriği **XAML** bazlı olacak şekilde tanımlanmış durumda. Burada durup bu **XAML** içeriğini yorumlayacak programları düşünmek gerekiyor. Daha karmaşık akışlara ait bu içerikler çeşitli araçlar yardımıyla yönetilebilir ve akışların değiştirilerek yeni halleriyle devreye alınması sağlanabilir.

Peki akışı devreye sokacak olan çalışma zamanı ortamının hazırlayacısı nerededir? Sonuç itibariyle yazılan **Workflow** örneklerinin mutlaka bir **Host** uygulama içerisinde ele alınıyor olması şarttır. örneğimiz **Visual Studio 2010** içerisine gömülmüş hazır bir **Console** şablonu olduğundan, .**Net 3.0 ve .Net 3.5**' teki **WF** projelerinde olduğu gibi tüm gerekli kodlar otomatik olarak üretilmektedir. Geliştirdiğimiz örnekte bu kodlar Program.cs dosyasının bir parçası olarak aşağıdaki gibi üretilir.

```
namespace HostWFApplication
{
   using System;
   using System.Linq;
   using System.Threading;
   using System.WorkflowModel;
   using System.WorkflowModel.Activities;

   class Program
   {
      static void Main(string[] args)
      {
```

```
        Sequence s;
        AutoResetEvent syncEvent = new AutoResetEvent(false);

        WorkflowInstance myInstance = WorkflowInstance.Create(new
ProductOrderFlow());
        myInstance.Completed += delegate(object sender, WorkflowCompletedEventArgs
e)
            {
               syncEvent.Set();
            };
        myInstance.Resume();

        syncEvent.WaitOne();

      }
   }
}
```

Görüldüğü gibi ilk olarak **ProductOrderFlow** tipinden bir **WorkflowInstance** referansı üretilmektedir. **Workflow** tamamlandığında **Completed** olay metodu devreye girer. Her zamanki gibi**WorkflowCompletedEventArgs** parametresinden yararlanarak örneğin **WF** tarafından üretilen **Output** değerleri ele alınabilir. *(Bu arada s isimli Sequence tipinden bir değişken yer aldığı görülmektedir. Hiç kullanılmayan bu değişkenin final sürümünde zaten ortadan kaldırılacağı söyleniyor. Burada kazayla kaldığını sanıyorız ;))* Uygulamamızı bu son haliyle çalıştırdığımızda aşağıdaki ekran görüntüsü ile karşılaşırız.



Dikkat edileceği üzere kullanıcıdan ürün numarası ve sipariş adedi istenmiş sonrasında ise buna uygun bir işlem yürütülmüş ve tasarlanmış olan akış başarılı bir şekilde tamamlanmıştır.

Buraya kadar anlattıklarımız ile aslında **Workflow Foundation 4.0** ile gelen yeniliklerin sadece bir kısmını inceleme şansını bulduk. Durumu değerlendirdiğimizde aşağıdaki maddeler ile kısa bir özet geçebiliriz;

- **WF 4.0** içerisinde **XAML** kullanımı daha etkili ve yaygın hale getirilmiş, bu sayede **Workflow Based Service**' lerin dekleratif olarak geliştirilmesinin yolu tam olarak açılmıştır. Bu konuyu bir sonraki makalemizde*(veya görsel dersimizde)* incelemeye çalışıyor olacağım.

- **WF 4.0** içerisinde **WCF 4.0** ile daha iyi anlaşılmasını sağlayacak yeni aktiviteler eklenmiştir.
- Sayısız pek çok yeni aktivite dışında **FlowChart** tipide göz ardı edilmemelidir. Bu tip özellikle geliştiricilerin bildiği akış şemaları mantığına uygun olacak şekilde süreç tasarlanmasını olanaklı kılmaktadır.
- **Activity Base Library** içerisinde yer alan ata aktivite tipi artık **WorkflowElement** olarak tasarlanmıştır. Buna göre sıfırdan tasarlanacak aktivitelerin **WorkflowElement abstract** sınıfından türemesi, var olanları genişletecek olanların ise **Activity** sınıfından türeyerek **CreatBody** metodunu ezmesi önerilir.
- Geliştirici tanımlı bir aktivite, istenirse kod yerine tasarım aracı yardımıyla da üretilebilir. Bu durumda bir **XAML** içeriği oluşturulmaktadır.
- **Visual Studio 2010** tarafında **WPF** tabanlı yeni tasarım aracı sayesinde, **WF** geliştirilmesi son derece kolay ve zevkli bir hal almaktadır. özellikle aktivitelerin **arguman(Argument)** veya**değişkenlerinin(Variables)** görsel olarak belirlenebilmesi önemli noktalardan birisidir.
- **XAML** tabanlı **WF** yapıları, **Oslo** gibi modellerin sağladığı **depolama alanlarında(Repositories)** saklanabilir, başka uygulamalar tarafından*(**Quadrant** gibi)* yönetilip değiştirilebilir, diğer **WF çalışma zamanı motorlarına** aktarılabilmeleri**(Import)** için bulundukları ortamlardan **çıktı(Export)** olarak verilebilirler.

Elbette bahsettiğimiz konuların tamamında değişiklikler olabilir, olması muhtemeldir. Kesinlik ancak son sürümler ile ortaya çıkacaktır. Böylece geldik bir makalemizin daha sonuna. Bu makalemizde ilk bakışta **WF 4.0** ile gelen yeniliklerin bir kısmına değinmeye çalıştık ve konuyu pekiştirmek amacıyla bir örnek geliştirdik. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

SQL Persistence Hizmeti (2009-03-06T04:41:00)

wf,sql,

Workflow Foundation yardımıyla kod akışlarının modellenebilmesi ve herhangibir .Net uygulaması içerisinden host edilerek çalıştırılabilmesi mümkündür. Bu kavram işin içerisine servisler girdiğinde çok daha genişlemektedir. Nitekim servisler yardımıyla **Workflow** örneklerinin **host** edildikleri uygulama dışındaki ortamlar ile haberleşebilmeleri mümkün olmaktadır. Hatta servislerin kendi içlerinde Workflow aktivitilerini kullanabilmeleri ve böylece belirli kod akışlarını yürütebilmeleride mümkündür. Bu cümleden sonra durup düşünüldüğünde **Workflow Foundation** kavramının akışları, platform bağımsız ortamlara taşıyabileceği sonucuda ortaya çıkmaktadır. Diğer taraftan Workflow uygulamaları sadece dış ortamlar ile haberleşmek için servislerden yararlanmazlar. İlaveten, **Workflow çalışma zamanını(Runtime)** ilgilendiren ve özellikle aktivitilerin dayanıklı olarak saklanmasını**(Durable Persistence)**, çalışma hayatlarının izlenmesini**(Tracking)**, adımlar

arası geçişlerin özel olarak planlanmasını**(Scheduling)** sağlayan ve **.Net Framework**içerisinde önceden tanımlanmış servislerde söz konusudur.

***NOT : Workflow** servisleri, çalışma zamanına eklenerek workflow örneklerine yeni kabiliyetler kazandırılmasını sağlamak amacıyla kullanılırlar.*
*örneğin **PersistenceServices**ve **TrackingServices**, SQL veritabanlarını varsayılan olarak kullanan çalışma zamanı servisleridir. **PersistenceServices** ile workflow' ların kalıcı olarak saklanabilmesi sağlanabilir.**TrackingServices** yardımıylada çalışma zamanı workflow örneklerinin izlenebilmesi mümkündür. Bir başka örnek olarak workflow çalışma zamanı davranışlarını değiştirmemizi sağlayan **Manual Scheduler** servisi göz önüne alınabilir. Bu servislerin bir kısmının kullanılabilmesi için, çalışma zamanına bilinçli olarak eklenmeleri gerekmektedir. Bir başka deyişle etkinleştirilmeleri gerekir.*

İşte bu yazımızdaki konumuz, uzun süreli çalışma ihtimali olan bir Workflow' un belirli koşullarda kalıcı olarak fiziki bir ortamda saklanmasının, **SQL Persistence Service** yardımıyla nasıl gerçekleştirilebileceğidir. **SQL Persistence Service** sayesinde, bir Workflow' un faaliyetsiz kalması**(Idle)** halinde bellek yerine, tablo bazlı bir ortamda saklanabilmesi ve bu durum sona erdiğinde söz konusu depolama alanından tekrar ayağa kaldırılarak çalışmaya devam etmesi mümkün olmaktadır. Konuyu daha kolay kavrayabilmek adına ilk önce **Workflow çalışmazamanının** kendi ve yönettiği **WF** örnekleri ile ilişkili yaşam döngüsünü incelemekte yarar vardır. Bu yaşam döngüsünün kolayca ele alınabilmesi için **WorkflowRuntime** sınıfı içerisine çeşitli olaylar eklenmiştir.

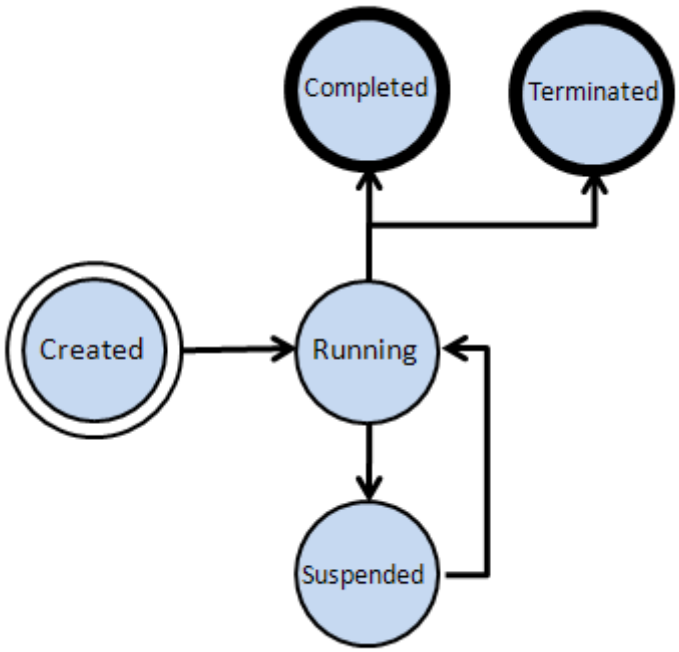
Bilindiği üzere **WorkflowRuntime** sınıfı çalışma zamanında WF örneklerinin yönetiminden sorumludur. Bu yönetim işlemi sırasında **WorkflowRuntime** sınıfı üzerinden ele alınabilecek **14** farklı olay metodu vardır. Söz konusu olayların bir kısmı sadece çalışma zamanını ilgilendirirken, bir kısmıda **WF** örneklerinin yaşam döngülerine**(LifeCycle)**adanmıştır. Buna göre **ServicesExceptionNotHandled**, **Started** ve **Stopped** olayları Workflow çalışma zamanı olayları olarak düşünülebilir.

Workflow çalışma Zamanı Olayları	
ServicesExceptionNotHandled	Workflow çalışma zamanı servislerinden herhangibirinde kontr
Started	**Workflow** çalışma zamanı motoru, üzerine eklenmiş servisler i servislerin başarılı bir şekilde başlatıldıklarına dair bir bilgilend
Stopped	**Started** olayına benzer olaraktan, WF çalışma zamanı motorun durdurulması sonrasında tetiklenir. Servisler başarılı bir şekilde

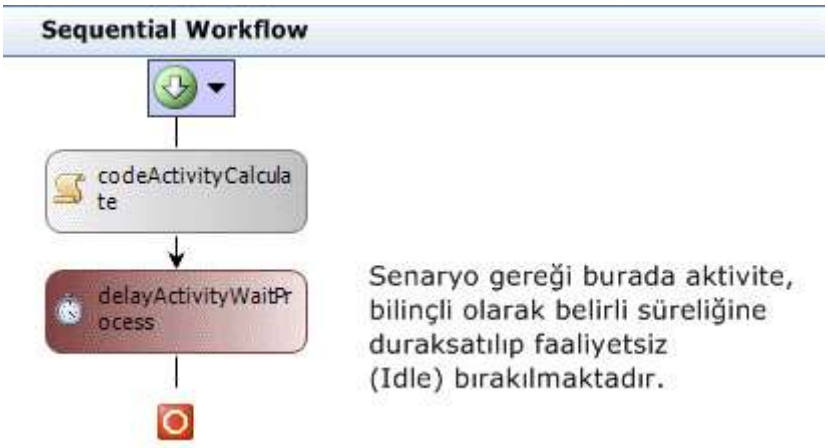
Aşağıdaki tabloda açıklamaları verilmiş olan olaylar ise, WF çalışma zamanının yönettiği **Workflow** örneklerinin **durumlarının(State)** değiştiği hallerde tetiklenmektedir.

Workflow örneğine Adanmış Olaylar	
WorkflowAborted	Workflow örneği devre dışı bırakıldığında tetiklenir. özellikle **Persisten** olarak saklanması ve tekrar kaldığı yerden **ayağa kaldırılması(Resume)**
WorkflowCompleted	Bir Workflow örneği tamamlanıp bellekten henüz kaldırılmadan önce de Workflow örneğinin **output** parametrelerini döndürmek mümkündür.
WorkflowCreated	Workflow örneği oluşturulduğunda ancak **Start** metodu ile çalıştırılmad
WorkflowIdled	Bir workflow örneği dışarıdan beklediği bir etki veya **Delay** aktivitesi ne özellikle Idle olma durumunda Persistence hizmetlerinin kullanımı önem
WorkflowLoaded	**Persistence** servisi kullanıldığı durumlarda, Workflow örneğinin herhan olaydır.
WorkflowPersisted	**Persistence** servisin kullanılması halinde bir workflow örneğinin saklan varsayılan olarak **SQL**veritabanını kullandığından, söz konusu kaydetm
WorkflowResumed	Bir erteleme nedeni ile **Suspended** moda geçen bir örneğin tekrar ayağa çalıştırılmadan önce devreye giren olaydır.
WorkflowStarted	**Workflow** örneği yürütülmeye başlatıldığında tetiklenir. Bu başlangıç **k**
WorkflowSuspended	Workflow örneği, **Suspend** metoduna yapılan çağrı veya **Suspend** aktiv
WorkflowTerminated	Workflow örneği yok edildikten ama bellekten atılmadan az önce çalışa yardımıyla, **Terminate aktivitesine** gelinmesi nedeniyle veya ele **alınm** Eğer persistence servis kullanılıyorsa, Terminate edilen workflow örneğ servisi göz önüne alındığında bu, tablolar üzerinde gerekli silme işlemle
WorkflowUnloaded	**Persistence** servisleri kullanıldığında, Workflow örneği depolama alanı

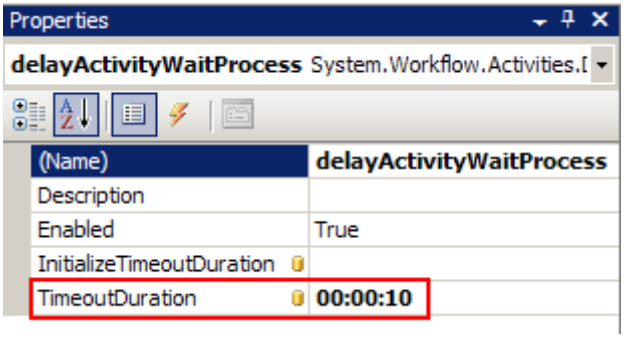
Aslında **WF çalışma Zamanı Motorunun(WF Runtime Engine)** kendisinin bir **State Machine** olduğu rahatlıkla düşünülebilir. Nitekim, yönetmekte olduğu örneklerin durumları arasındaki geçişleri kontrol altına almakta ve bununla ilişkili olayları yönetmektedir. Temel olarak workflow örnekleri **Created**, **Running**, **Suspended**, **Completed** ve **Terminated** olmak üzere 5 farklı duruma sahip olabilir. Bu durum aşağıdaki diyagram ile özetlenebilir.



İlk etapta, **Persistence** servisinin devrede olmadığı durumlarda standart olarak çalışan olay metodlarını ele alacağımız bir örnek geliştirerek devam edebiliriz. Bu amaçla örnek bir**Sequential Workflow Console Application** oluşturarak başladığımızı düşünebiliriz. Söz konusu örnek içerisinde **Costflow** isimli bir **Sequential Activity** kullanılmakta olup adımları aşağıdaki şekilde görüldüğü gibidir.



Söz konusu aktivite aynı zamanda dışarıdan parametre alıp, bir sonuç üretmektedir. Aktivitenin **faaliyetsiz(Idle)** moda geçtiğini görmek için sembolik olarak **Delay** aktivitesinden yararlanılmaktadır. Söz konusu aktivititede sadece duraksama süresi özelliği 10 saniye olarak set edilmiştir.



Costflow aktivitesine ait kod içeriği aşağıdaki gibi tasarlanabilir.

```
using System;
using System.Workflow.Activities;

namespace WFCostFactory
{
   public enum WorkType
   {
      Consumer,
      Corporate
   }
   public sealed partial class Costflow
         : SequentialWorkflowActivity
   {
      #region Workflow özellikleri(Properties)

      // Dış ortamdan gelen parametreler
      public int TotalDays { get; set; }
      public decimal CostValue { get; set; }
      public WorkType WorkT { get; set; } // Dış ortama sonuç olarak döndürülen
parametre

      #endregion

      public Costflow()
      {
         InitializeComponent();
      }

      // CodeActivity tarafından çalıştırılan örnek fonksiyonellik
      private void Calculate(object sender, EventArgs e)
      {
         Console.WriteLine("Calculate Metodu. Maliyet hesaplama işlemleri yapılır");
         switch (WorkT)
         {
```

```
            case WorkType.Consumer:
               CostValue = TotalDays * 1.10M;
               break;
            case WorkType.Corporate:
               CostValue = TotalDays * 1.15M;
               break;
            default:
               CostValue = 1;
               break;
         }
      }
   }
}
```

Söz konusu aktiviteyi host eden **Console** uygulamasına ait kod içeriği ise aşağıdaki gibidir. Burada dikkat edilmesi gereken nokta **WorkflowRuntime** örneğine ait tüm olayların yüklenmiş olmasıdır. Bu olayların çoğu örneğimizde devreye girmeyecektir. Ancak hangi durumlarda devreye gireceği yukarıdaki tablolarda belirtilmiştir.

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Workflow.Runtime;

namespace WFCostFactory
{
   class Program
   {
      static WorkflowRuntime wfRuntime = null;
      static AutoResetEvent wHandle = null;

      static void Main(string[] args)
      {
         // Workflow Runtime nesnesi örneklenir
         using(wfRuntime = new WorkflowRuntime())
         {
            wHandle = new AutoResetEvent(false);

            #region Event Tanımlamaları

            wfRuntime.Started += new
EventHandler<WorkflowRuntimeEventArgs>(wfRuntime_Started);
            wfRuntime.ServicesExceptionNotHandled += new
EventHandler<ServicesExceptionNotHandledEventArgs>(wfRuntime_ServicesException
NotHandled);
```

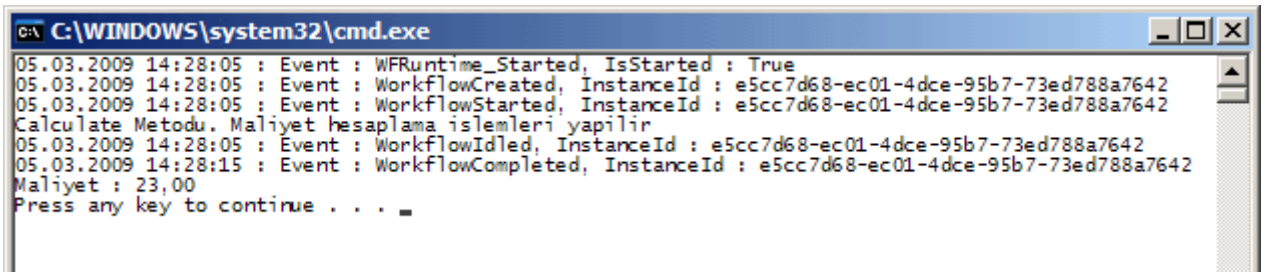
```
            wfRuntime.Stopped += new
EventHandler<WorkflowRuntimeEventArgs>(wfRuntime_Stopped);
            wfRuntime.WorkflowAborted += new
EventHandler<WorkflowEventArgs>(wfRuntime_WorkflowAborted);
            wfRuntime.WorkflowCreated += new
EventHandler<WorkflowEventArgs>(wfRuntime_WorkflowCreated);
            wfRuntime.WorkflowIdled += new
EventHandler<WorkflowEventArgs>(wfRuntime_WorkflowIdled);
            wfRuntime.WorkflowLoaded += new
EventHandler<WorkflowEventArgs>(wfRuntime_WorkflowLoaded);
            wfRuntime.WorkflowPersisted += new
EventHandler<WorkflowEventArgs>(wfRuntime_WorkflowPersisted);
            wfRuntime.WorkflowResumed += new
EventHandler<WorkflowEventArgs>(wfRuntime_WorkflowResumed);
            wfRuntime.WorkflowStarted += new
EventHandler<WorkflowEventArgs>(wfRuntime_WorkflowStarted);
            wfRuntime.WorkflowSuspended += new
EventHandler<WorkflowSuspendedEventArgs>(wfRuntime_WorkflowSuspended);
            wfRuntime.WorkflowTerminated += new
EventHandler<WorkflowTerminatedEventArgs>(wfRuntime_WorkflowTerminated);
            wfRuntime.WorkflowUnloaded += new
EventHandler<WorkflowEventArgs>(wfRuntime_WorkflowUnloaded);
            wfRuntime.WorkflowCompleted+=new
EventHandler<WorkflowCompletedEventArgs>(wfRuntime_WorkflowCompleted);

            #endregion

            // Workflow nesne örneği oluşturulur
            // TotalDays ve WorkT özellikleri için ilk değerler set edilir
            WorkflowInstance instance = wfRuntime.CreateWorkflow(
               typeof(WFCostFactory.Costflow)
               , new Dictionary<string,object>
                  {
                     {"TotalDays",20}
                     ,{"WorkT",WorkType.Corporate}
                  }
               );
            // Workflow örneği başlatılır
            instance.Start();
            // İşlemler tamamlana kadar bekle
            wHandle.WaitOne();
         }
      }

      static void wfRuntime_WorkflowUnloaded(object sender, WorkflowEventArgs e)
```

```
    {
        Console.WriteLine("{0} : Event : {1}, InstanceId : {2}", DateTime.Now,
"WorkflowUnloaded", e.WorkflowInstance.InstanceId.ToString());
    }

    static void wfRuntime_WorkflowTerminated(object sender,
WorkflowTerminatedEventArgs e)
    {
        Console.WriteLine("{0} : Event : {1}, InstanceId : {2} Exception Message : {3}",
DateTime.Now,
"WorkflowTerminated",          e.WorkflowInstance.InstanceId.ToString(),
e.Exception.Message);
        wHandle.Set();
    }

    static void wfRuntime_WorkflowSuspended(object sender,
WorkflowSuspendedEventArgs e)
    {
        Console.WriteLine("{0} : Event : {1}, InstanceId : {2}", DateTime.Now,
"WorkflowSuspended", e.WorkflowInstance.InstanceId.ToString());
    }

    static void wfRuntime_WorkflowStarted(object sender, WorkflowEventArgs e)
    {
        Console.WriteLine("{0} : Event : {1}, InstanceId : {2}", DateTime.Now,
"WorkflowStarted", e.WorkflowInstance.InstanceId.ToString());
    }

    static void wfRuntime_WorkflowResumed(object sender, WorkflowEventArgs e)
    {
        Console.WriteLine("{0} : Event : {1}, InstanceId : {2}", DateTime.Now,
"WorkflowResumed", e.WorkflowInstance.InstanceId.ToString());
    }

    static void wfRuntime_WorkflowPersisted(object sender, WorkflowEventArgs e)
    {
        Console.WriteLine("{0} : Event : {1}, InstanceId : {2}", DateTime.Now,
"WorkflowPersisted", e.WorkflowInstance.InstanceId.ToString());
    }

    static void wfRuntime_WorkflowLoaded(object sender, WorkflowEventArgs e)
    {
        Console.WriteLine("{0} : Event : {1}, InstanceId : {2}", DateTime.Now,
"WorkflowLoaded", e.WorkflowInstance.InstanceId.ToString());
    }
```

```
        static void wfRuntime_WorkflowIdled(object sender, WorkflowEventArgs e)
        {
            Console.WriteLine("{0} : Event : {1}, InstanceId : {2}", DateTime.Now,
"WorkflowIdled", e.WorkflowInstance.InstanceId.ToString());
        }

        static void wfRuntime_WorkflowCreated(object sender, WorkflowEventArgs e)
        {
            Console.WriteLine("{0} : Event : {1}, InstanceId : {2}", DateTime.Now,
"WorkflowCreated", e.WorkflowInstance.InstanceId.ToString());
        }

        static void wfRuntime_WorkflowCompleted(object sender,
WorkflowCompletedEventArgs e)
        {
            Console.WriteLine("{0} : Event : {1}, InstanceId : {2}", DateTime.Now,
"WorkflowCompleted", e.WorkflowInstance.InstanceId.ToString());
            Console.WriteLine("Maliyet : {0}", e.OutputParameters["CostValue"].ToString());
            wHandle.Set();
        }

        static void wfRuntime_WorkflowAborted(object sender, WorkflowEventArgs e)
        {
            Console.WriteLine("{0} : Event : {1}, InstanceId : {2}", DateTime.Now,
"WorkflowAborted", e.WorkflowInstance.InstanceId);
        }

        static void wfRuntime_Stopped(object sender, WorkflowRuntimeEventArgs e)
        {
            Console.WriteLine("{0} : Event : {1}, IsStarted : {2}", DateTime.Now,
"WFRuntime_Stopped", e.IsStarted.ToString());
        }

        static void wfRuntime_ServicesExceptionNotHandled(object sender,
ServicesExceptionNotHandledEventArgs e)
        {
            Console.WriteLine("{0} : InstanceId : {1} Event : {2}, Exception Message : {3}",
DateTime.Now,
e.WorkflowInstanceId.ToString(),"WF        Runtime_ServicesExceptionNotHandled",e.Ex
ception.Message);
        }

        static void wfRuntime_Started(object sender, WorkflowRuntimeEventArgs e)
        {
```

```
            Console.WriteLine("{0} : Event : {1}, IsStarted : {2}", DateTime.Now,
"WFRuntime_Started", e.IsStarted.ToString());
        }
    }
}
```

örneği ilk etapta bu haliye çalıştırdığımızda aşağıdaki ekran çıktısı ile karşılaşırız.



İlk analizimi yapabiliriz artık. Dikkat edileceği üzere ilk olarak **Workflow** çalışma zamanına ait **Started** olayı tetiklenmiş ve **IsStarted** değeri **true** olarak set edilmiştir. Hatırlanacağı üzere Workflow motorunun kullandığı veya başlattığı tüm servislerin **IsStarted** özelliğine etkisi vardır ve bu özelliğin değeri **true** kalmadığı sürece çalışma zamanı başlatılamayacaktır. Bu işlemin arkasından **Workflow** nesnesi örneklendiği için **WorkflowCreated** ve **WorkflowStared** olayları sırasıyla çalışmaktadır. Süreç devam etmekteyken**Delay** aktivitesi devreye girmiştir. İşte bu noktada Workflow örneği faaliyetsiz kalarak, çalışma zamanı moturu tarafından bellekte tutulmaya devam edilmektedir. Bu anda **WorkflowIdled** olayı tetiklenmiştir.

NOT : *özellikle olay metodlarının bazıları içerisindeden **GUID** tipinden **InstanceId** değerlerinin elde edilebiliyor olması önemlidir ki bu sayede hangi **WF** örneğinin faaliyetsiz kaldığı kolayca anlaşılabilmektedir. Tahmin edeceğiniz üzere bu Id değeri **persistence** ortamları içinde benzersiliği sağlamak açısından önemlidir.*

Faaliyetsiz kalma durumu **Delay** aktitivitesinde belirtilen süre sonlanıncaya kadar devam eder. Süre sonunda ise **Workflow** örneği çalışmasına kaldığı yerden tekrar başlayacaktır. İşte bizim en büyük amacımız bu faaliyetsiz kalma anında söz konusu **WF** örneğini **SQL Persistence Service** yardımıyla fiziki bir ortama kaydetmektir.

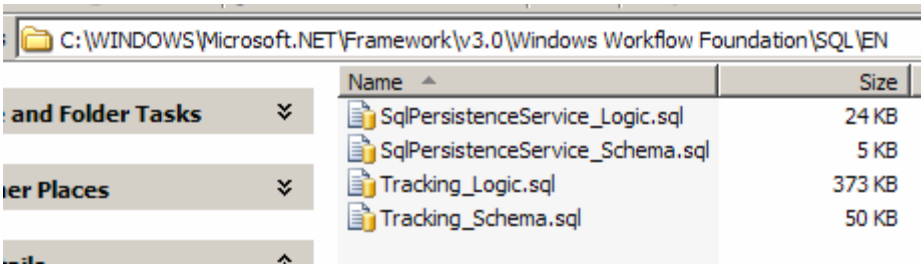
Varsayılan olarak **WF** örneklerinin durumu bellekte saklanır. Bir başka deyişle, **workflow** herhangibir sebeple faaliyetsiz duruma geçtiğinde söz konusu örnek bellekte asılı olarak kalır ve beklemeye başlar. Ancak gerçek hayat senaryolarında çalışmakta olan **Workflow** örneklerinin uzun süre asılı kalmasıda söz konusu olabilir. Bu, özellikle faaliyetine devam etmesi için onu bekleten operasyona bağlıdır. Dolayısıyla bu tip vakalarda Workflow örneklerinin kalıcı olarak saklanmaları tercih edilebilir. Kalıcılıkta esas olan **Workflow** örneğinin o anki durumu ve değerleri ile fiziki bir depolama alanına atılmasıdır. Bu noktada çoğunlukla **SQL** gibi veritabanı kaynaklarının kullanılması tercih edilir. Diğer taraftan elbetteki **Persistence** servisleri özelleştirilebilir ve farklı veri

kaynaklarına kaydetme işlemleri gerçekleştirilebilir.

Workflow Foundation, çalışma zamanındaki **WF** örneklerinin bellekten kaldırılıp çalışmasının durdurulması sonrasında, kalıcı olarak saklanabilmeleri için önceden geliştirilmiş bir **Persistence** servisi sunmaktadır. Hangi tip Persistence kullanılırsa kullanılsın, Workflow çalışma zamanı, kalıcı olarak saklanan Workflow örneğine(örneklerine) gelen mesajları takip de eder. Bu sayede gerektiği anda, üzerinde yer alan **Persistence** servisi devreye alarak, ilgili WF örneğinin tekrardan belleğe yüklenmesini sağlayabilir. Workflow çalışma zamanı, **Persistence** servisini belirli durumlar gerçekleştirildiğinde çağırmaktadır. Bu durumlar;

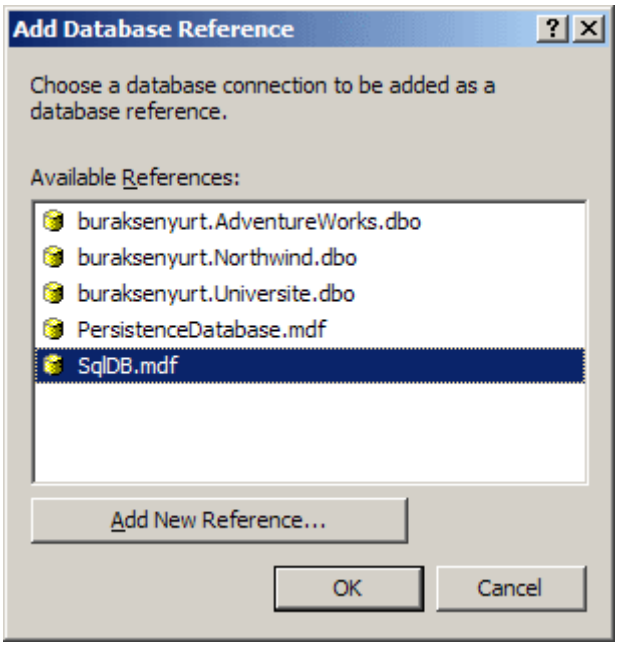
- WF örneği belirli bir nedenden askıya alındığında yani faaliyetsiz hale geçtiğinde**(Idle)**.
- WF örneği yok edilmeden**(Terminate)** önce.
- WF örneği tamamlanmadan**(Complete)** önce.
- Workflow örneği üzerinde **Unload**, **TryUnload** metodları çağırıldığında.
- **PersistOnCloseAttribute** niteliği ile imzalanmış olan bir aktivitenin tamamlanması sonrasında. özellikle **transaction** kullanan aktivitiler bu niteliğe sahiptir. Diğer taraftan içerisinde transaction kullanılacak olan aktivitilerinde bu nitleği uygulaması gerekir.

WorkflowPersistenceService abstract sınıfından türetme yapılarak istenirse özel persistence sınıflarıda yazılabilmektedir. Biz örneğimizde **SqlWorkflowPersistenceService** tipinden yararlanarak WF örneklerini **SQL** veritabanı üzerinde saklamaya çalışacağız. Şimdi bu durumu incelememiz gerekiyor. Ancak depolama alanı için **SQL** tarafında gerekli hazırlıkların yapılması gerekmektedir. Bu noktada **.Net Framework** ile birlikte hazır olarak gelen **WF SQL betikleri(Scripts)** kullanılabilir. Söz konusu SQL betikleri varsayılan olarak örneğin **Windows XP**işletim sisteminin kurulu olduğu bir makinede **C:\WINDOWS\Microsoft.NET\Framework\v3.0\Windows Workflow Foundation\SQL\EN** klasöründe yer almaktadır.

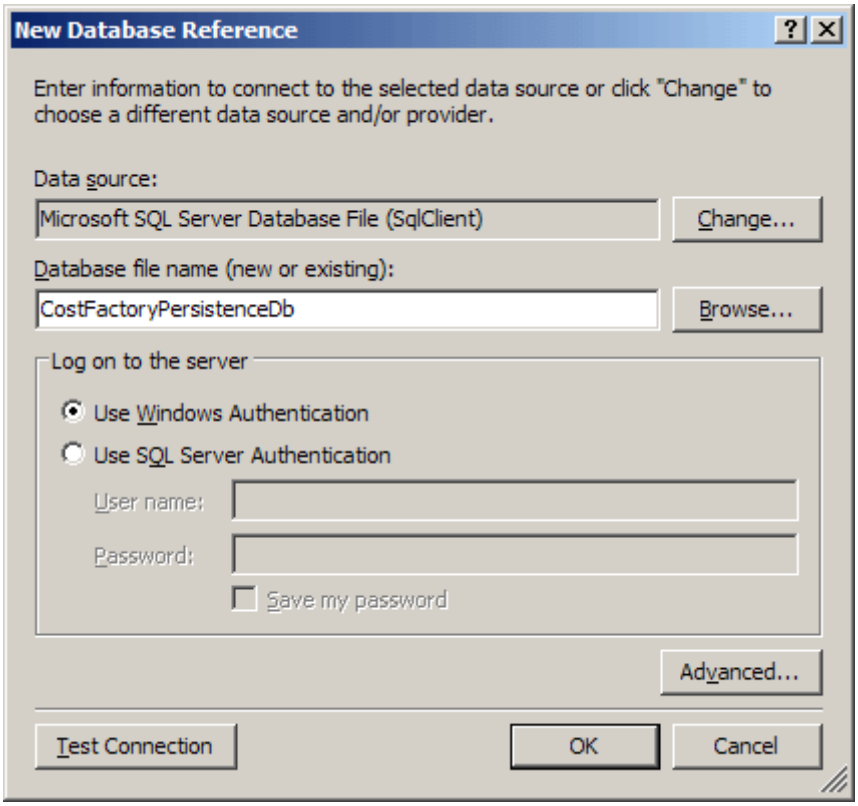


Burada yer alan **SqlPersistenceService_Logic** ile **SqlPersistenceService_Schema** betikleri, **Persistence** depolama alanı için gerekli **tablo(Table), saklı yordam(Stored Procedure)** gibi veritabanı nesnelerini oluşturmakla görevlidir. Söz konusu betikler, bir SQL sunucusu üzerinde çalıştırılıp kullanılabileceği gibi, istenirse ilgili Workflow çalışma zamanını **Host** eden uygulamanın erişebileceği dosya bazlı bir veritabanı üzerindende

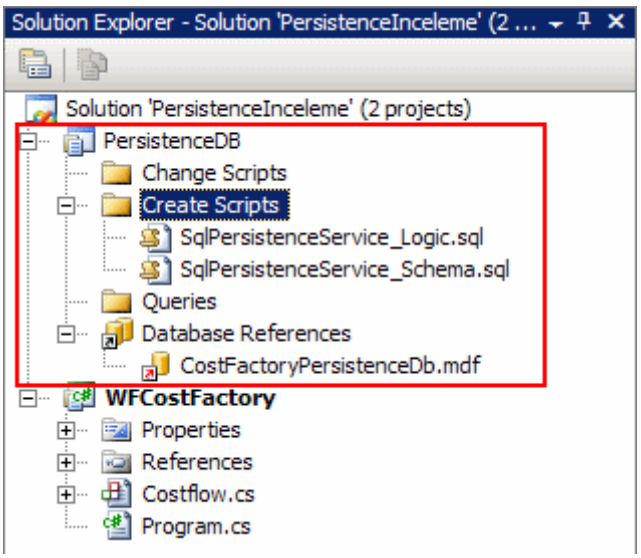
çalıştırılabilir. Biz örneğimizde ikinci seçeneği kullanacağız. Yani, söz konusu depolama alanı için **SQL Express Edition** temelli bir veritabanı dosyasını ele alacağız. Bu amaçla ilk olarak **Solution**' ımıza bir **Database Project** ekleyerek devam edebiliriz. Proje eklenmesi sırasında bize aşağıdaki ekran görüntüsünde olduğu gibi kullanmak istediğimiz veritabanı sorulacaktır.



Elimizde böyle bir veritabanı olmadığını göz önüne alaraktan **Add New Reference** seçeneğine tıklayalım. Sürekli kullandığımız standart bağlantı ekleme iletişim kutusu ile karşılacağız. Burada önemli olan veri kaynağı olarak **Microsoft SQL Server Database File (SqlClient)** tipinin seçilmesidir. Sonrasında ise veritabanımıza bir isim vererek devam edebiliriz.



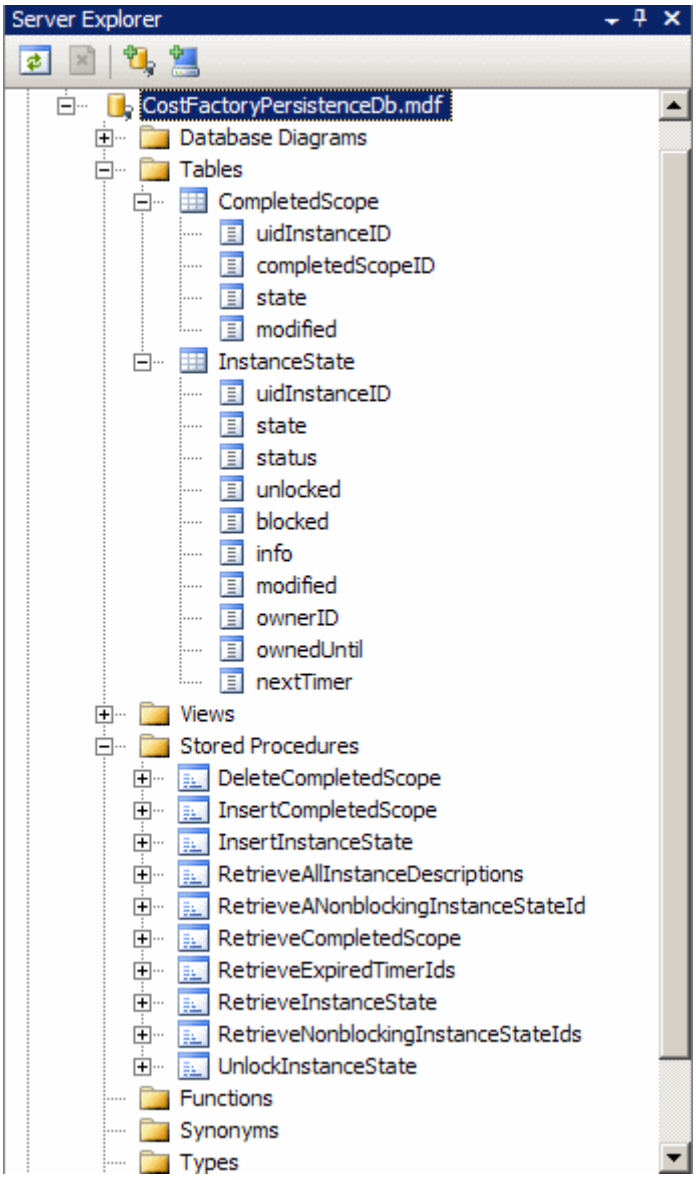
Ok düğmesine bastığımızda söz konusu veritabanı yoksa eğer, oluşturmak isteyip istemediğimize dair bir soru sorulacaktır. Bu oluşturma işlemi sırasında unutulmaması gereken noktalardan biriside **SQL Express** servisinin çalışıyor olması zorunluluğudur. Eğer servis çalışmıyorsa tahmin edileceği üzere söz konusu veritabanı oluşturulamayacaktır.**Database** projesi oluşturulduktan sonra yukarıda değindiğimiz **SQL betiklerini Create Scripts** klasörü altına ekleyerek devam edebiliriz. Bu işlemler sonrasında proje içeriği aşağıdakine benzer olacaktır.



(Buradaki gibi bir veritabanı projesinin oluşturulması aslında şart değildir. Bu sadece söz konusu veritabanının yönetimin kolaylaştırılmasını sağlayan ve belirli bir düzeni tesis eden

bir opsiyon olarak görülmelidir. Genel olarak gerçek hayat uygulamalarında depolama alanı olarak sunucu bazlı veritabanları tercih edilir. Sizlere tavsiyem aynı örneği SQL sunucusu üzerinde gerçekleştirmeye çalışmanızdır.)

Sıradaki işlem, söz konusu **SQL** betiklerinin çalıştırılmasıdır. Bu betikler çalıştırıldıktan sonra **CostFactoryPersistenceDb** isimli veritabanının içeriği aşağıdaki şekilde görüldüğü gibi oluşturulacaktır.



Burada temel olarak **Workflow** örneklerinin **saklanması(Insert)**, **kilitlenmesi(lock)**, **elde edilmesi(retrieve)** veya **silinmesi(delete)** ile ilişkili gerekli **Stored Procedure**' ler ve **tablolar** yer almaktadır. Artık depolama alanıda tanımlandığına göre, **Workflow** uygulamamız için gerekli kod değişikliklerini yapabiliriz. Bu amaçla host uygulama üzerinde **SqlWorkflowPersistenceService**' in oluşturulması ve çalışma zamanına eklenmesi gerekmektedir. İşte örnek kodlarımız;

```
using System;
using System.Collections.Generic;
using System.Threading;
using System.Workflow.Runtime;
using System.Workflow.Runtime.Hosting;

namespace WFCostFactory
{
   class Program
   {
      static WorkflowRuntime wfRuntime = null;
      static AutoResetEvent wHandle = null;

      static void Main(string[] args)
      {
         // Workflow Runtime nesnesi örneklenir
         using(wfRuntime = new WorkflowRuntime())
         {
            // Varsayılan ayarları ile persistence servisi örneklenir
            SqlWorkflowPersistenceService persistenceService = new
SqlWorkflowPersistenceService
            (
               @"Data Source=.\SQLEXPRESS;AttachDbFilename=C:\Documents and
Settings\Burak Selim Senyurt\My Documents\CostFactoryPersistenceDb.mdf;Integrated
Security=True;Connect Timeout=30;User Instance=True"
            );
            // Oluşturulan servis çalışma zamanına bildirilir.
            wfRuntime.AddService(persistenceService);
            //Diğer kod satırları...
```

İlk olarak **System.Workflow.Runtime.Hosting** isim alanında(namespace) yer alan **SqlWorkflowPersistence** hizmetine ait bir nesne örneği oluşturulur. Nesne örneklenirken **yapıcımetod(Constructor)** içerisinde **persistence** için kullanılacak veri depolama alanın bağlantı bilgisi verilmektedir. Bu en basit yapıcı metodu versiyonudur. Diğer versiyonlarını kullanarak farklı başlangıç ayarlamaları yapılabilir. Söz gelimi Workflow örneklerinin **Idle** moda geçtiklerinde bellekten kaldırılıp kaldırılmayacakları, birden fazla Workflow çalışma zamanı moturunun **aynı** WF örneklerini kullanmaları halinde, birbirlerini kesmemeleri için **kilit sürelerinin(Lock Time)** ne olacağı gibi kriterlerde yapıcı metod parametreleri ile belirlenebilir.

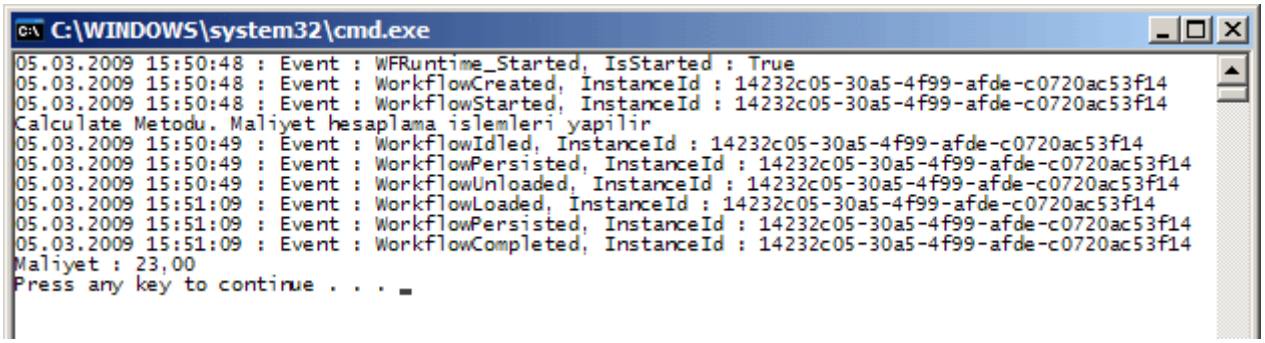
NOT : *Servis tanımlaması ile ilgili ayarlar istenirse* ***konfigurasyon*** *dosyasında da yapılabilir. Bunun için host uygulamaya ait konfigurasyon dosyasında örneğin aşağıdaki tanımlamaların yapılması yeterlidir.*

flow.cs | Program.cs | App.config | InstanceState...ISTENCEDB.MDF)

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <configSections>
    <section
      name="WorkflowRuntime"
      type="System.Workflow.Runtime.Configuration.WorkflowRuntimeSection,
System.Workflow.Runtime, Version=3.0.00000.0, Culture=neutral, PublicKeyToken
=31bf3856ad364e35" />
  </configSections>

  <WorkflowRuntime>
    <Services>
      <add
        type="System.Workflow.Runtime.Hosting.SqlWorkflowPersistenceService,
System.Workflow.Runtime, Version=3.0.00000.0, Culture=neutral, PublicKeyToken
=31bf3856ad364e35"
          ConnectionString="Data Source=.\SQLEXPRESS;AttachDbFilename=C:\
Documents and Settings\Burak Selim Senyurt\My Documents\
CostFactoryPersistenceDb.mdf;Integrated Security=True;Connect Timeout=30;
User Instance=True"/>
    </Services>
  </WorkflowRuntime>
</configuration>
```

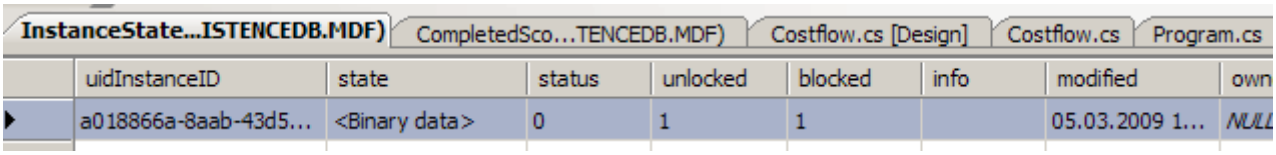
*Tabi host uygulama içerisinde **WorkflowRuntime** nesne örneği oluşturulurken **wfRuntime=new WorkflowRuntime("WorkflowRuntime");** şeklinde bir kullanım söz konusudur. Burada parametre olarak **app.config** dosyasındaki **section** adı verilmektedir. Bu ad benzersizdir. Yani farklı bir isim olamaz. Diğer taraftan yapıcı metodun bu versiyonunun çalıştırılabilmesi için(örneğin geliştirdiğimiz Console uygulamasında) mutlaka **System.Configuration assembly**' ının projeye referans edilmesi gerekmektedir.*

Artık uygulamamızı test etmeye başlayabiliriz. Konuyu kolay takip edebilmek amacıyla geliştirdiğiniz örneği **Debug** ederken adım adım ilerlemenizi öneririm. öncelikle programın çalışması sonrasındaki ekran görüntüsüne bakalım.



Dikkat edileceği üzere **Workflow** örneği faaliyetsiz hale geçtikten sonra(WorkflowIdled olayının tetiklenmesi sonrası) sırasıyla **WorkflowPersisted**, **WorkflowUnloaded** olayları çalışmıştır. Bir başka deyişle faaliyetsiz kalan Workflow örneği veritabanındaki ilgili

tablolara yazılmıştır. Diğer taraftan faaliyetsiz kalma süresi dolduğunda ve Workflow akışı tekrar devam etmek istediğinde
sırasıyla **WorkflowLoaded** ve **WorkflowPersisted** olayları tetiklenmiştir. Yani WF örneği tablodan tekrar yüklenerek yürütülmeye devam etmiş ve son olarakta tamamlanmıştır. özellikle Workflow faaliyetsiz hale geldiğinde **InstanceState** isimli tabloda aşağıdaki ekran görüntüsüne benzer olacak şekilde bir satır açıldığı ve **Delay**süresi sona erdikten sonra ise WF örneğinin tekrar ayağa kaldırılmasıyla birlikte söz konusu satırın silindiği görülür.



Tahmin edileceği üzere bu satır saklanan WF örneğine ait
bilgileri **serileştirerek** tutmaktadır. Bu nedenle özellikle WF içerisinde kullanılan tiplerin, eğer **SQL Persistence** hizmeti kullanılıyorsa serileştirilebilir olmalarına dikkat etmek gerekmektedir. Bu durumu analiz etmek için **Costflow** aktivitesine **Customer** isimli tipten bir özellik eklenmiştir.

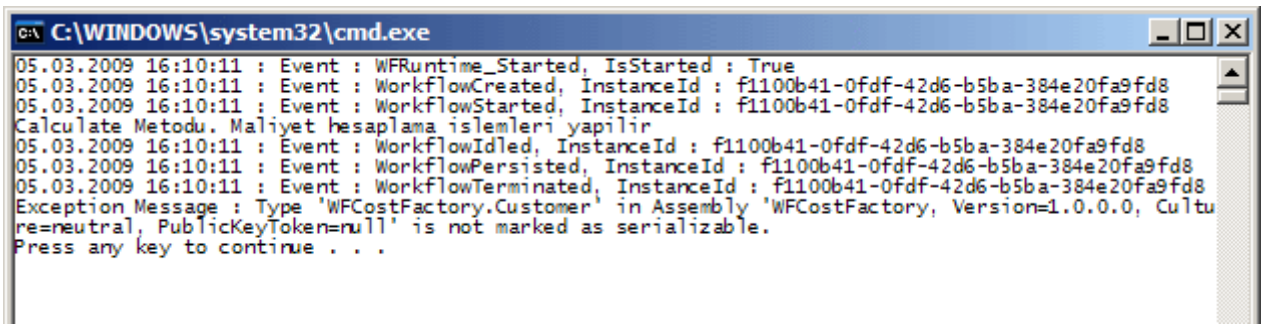
```
using System;
using System.Workflow.Activities;

namespace WFCostFactory
{
   public enum WorkType
   {
      Consumer,
      Corporate
   }
   public class Customer
   {
      public string Name { get; set; }
      public int Id { get; set; }
   }
   public sealed partial class Costflow : SequentialWorkflowActivity
   {
      #region Workflow özellikleri(Properties)

      // Dış ortamdan gelen parametreler
      public int TotalDays { get; set; }
      public decimal CostValue { get; set; }
      public WorkType WorkT { get; set; } // Dış ortama sonuç olarak döndürülen
parametre
      public Customer Owner { get; set; }
```

```
    #endregion

    public Costflow()
    {
       InitializeComponent();
    }
    // Diğer kod satırları
```

Burada hemen bir noktayı vurgulamak isterim. Söz konusu örnek bu haliyle çalıştırıldığında serileştirme ile ilişkili herhangibir hata mesajının alınmadığı görülecektir. Bunun nedeni **Customer** tipine ait nesne örneğinin WF içerisinde kullanılmamış olmasıdır. Bu nedenle Workflow nesnesi host uygulamada örneklenirken aşağıdaki kod değişikliğini yapmamız çalışma zamanında serileştirme hatasını almamız için gerekli ve yeterlidir.

```
WorkflowInstance instance = wfRuntime.CreateWorkflow(
    typeof(WFCostFactory.Costflow)
    , new Dictionary<string,object>
    {
       {"TotalDays",20}
       ,{"WorkT",WorkType.Corporate}
       ,{"Owner",new Customer{ Id=1000, Name="Burak Selim Şenyurt"}}
    }
);
```

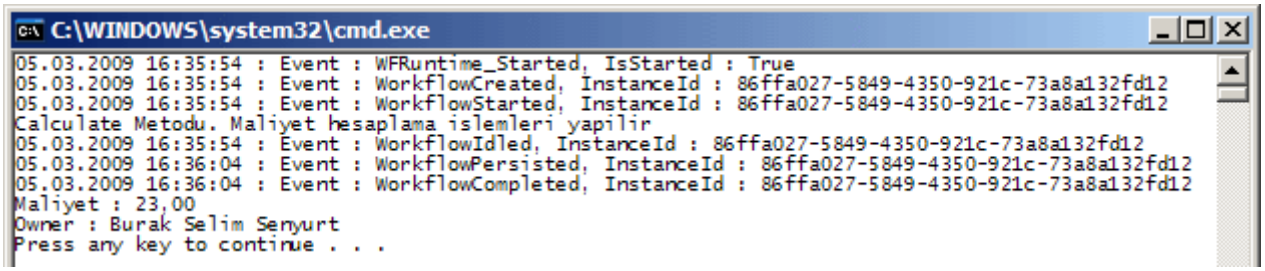
örnek bu haliyle çalıştırıldığında aşağıdaki görüntü ile karşılaşılır.



Dikkat edileceği üzere, **WorkflowPersisted** olay metodunun hemen arkasından **WorkflowTerminated** olayı tetiklenmiş ve oluşan **istisna(Exception)** mesajı ekrana yazdırılmıştır. Bir başka deyişle **Workflow** örneği tabloya yazdırılamamış ve istisna fırlatarak sonlanmıştır. İşte bu durumun sebebi **Owner** isimli **Customer** tipinin **binary** formatta serileştirilebilir **tanımlanmamasıdır**. Bu nedenle Customer tipinin **Serializable niteliği(Attribute)** ile aşağıdaki kod parçasında görüldüğü gibi işaretlenmesi gerekir.

```
[Serializable]
public class Customer
{
   public string Name { get; set; }
   public int Id { get; set; }
}
```

Uygulama tekrar denendiğinde sorunsuz olarak çalıştığı hatta **WF** örneği oluşturulurken parametre olarak verilen **Customer** nesnesinin, **WorkflowCompleted** olayında (tabloda saklanıp tekrar elde edilmesi ile birlikte) tedarik edilebildiği görülebilir.



Görüldüğü üzere **Workflow** örneklerinin, çalışma zamanında belirli koşulların sağlanması şartıyla bir depolama alanında saklanması ve sonradan tekrardan ayağa kaldırılıp yürütülmesi **SQL Persistence Service** yardımıyla son derece kolay bir şekilde gerçekleştirilebilmektedir. Elbetteki gerçek hayat koşullarında **SQL** dışı kaynakların kullanılmasıda istenebilir. Bu gibi vakalarda söz konusu hizmet için özel bir geliştirme yapılması gerekmektedir. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

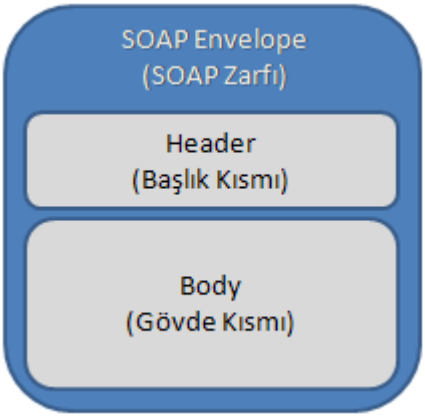
PersistenceInceleme.rar (33,50 kb)

WCF - Mesaj Sözleşmeleri(Message Contracts) (2009-02-09T04:37:00)

wcf,

Servis tabanlı uygulamalarda en önemli noktalardan biriside aradaki bilgi transferlerinin nasıl ve ne şekilde gerçekleştirildiğidir. Gerçek şuki, bu bilgi transferinin oluşma şekli çoğu zaman geliştiricinin gözünden kaçan yada çok fazla ilgilenmediği bir konu olmaktadır. Nitekim çoğu servis geliştirme aracı buradaki söz konusu içeriğin hazırlanmasını , gönderilmesini veya alınmasını otomatikleştirmektedir. özellikle **Windows Communication Foundation** tarafında, bilginin istemci ve servis arasındaki dolaşımında **bağlayıcı tiplerin(Binding Type)**seçilmesi ile zaten arka tarafta ne şekilde bir haberleşme olacağı ve paketlerin nasıl hazırlanacağı belirlenmiş olur. Aslında servis ve istemci tarafında mesaj bazlı bir iletişim olduğu son derece açıktır. Farklı platformlar üzerinde koşan servislerin haberleşmeleri yada farklı tipteki istemci uygulamaların servisleri kullanabilmeleri gerektiğinde ise, aradaki haberleşmenin bir standart üzerinde ve esnek olması beklenir. Bu nedenle özellikle **SOAP** bazlı web servisleri göz önüne

alındığında mesajın tipi ve içeriğide bellidir. İşte burada **SOAP(Simpe Object Access Protocol)** tarzı mesajlardan söz edilebilir. Tipik olarak **SOAP** mesajları bir zarf olarak temsil edilmekte**(SOAP Envelope)** ve **Header**, **Body** isimli iki parçadan oluşmaktadır. Aşağıdaki şekilde bu içerik temsil edilmeye çalışılmıştır.



Peki bu mesajların makalemize konu olmasının sebebi nedir? Bilindiği üzere **WCF** mimarisinde çeşitli tipte **sözleşmeler(Contracts)** söz konusudur. örneğin servislerin ne iş yaptığının, nasıl fonksiyonellikler sunduğunun ifade edilmesinde **Sevis Sözleşmeleri(Service Contracts)** kullanılmaktadır. Benzer şekilde istemci tarafına aktarılacak **serileştirilebilir(Serializable)** tipler söz konusu ise **Veri Sözleşmeleri(Data Contracts)** tanımlanır. Yine istemci tarafına aktarılacak istisna mesajlarının çeşitli durumlar için özelleştirilmesi düşünüldüğünde**Hata Sözleşmeleri(Fault Contracts)** kullanılır. Ancak bu sözleşme çeşitleri dışında birde **Mesaj Sözleşmeleri(Message Contracts)** bulunmaktadır. İşte bu yazımızın konusuda budur.

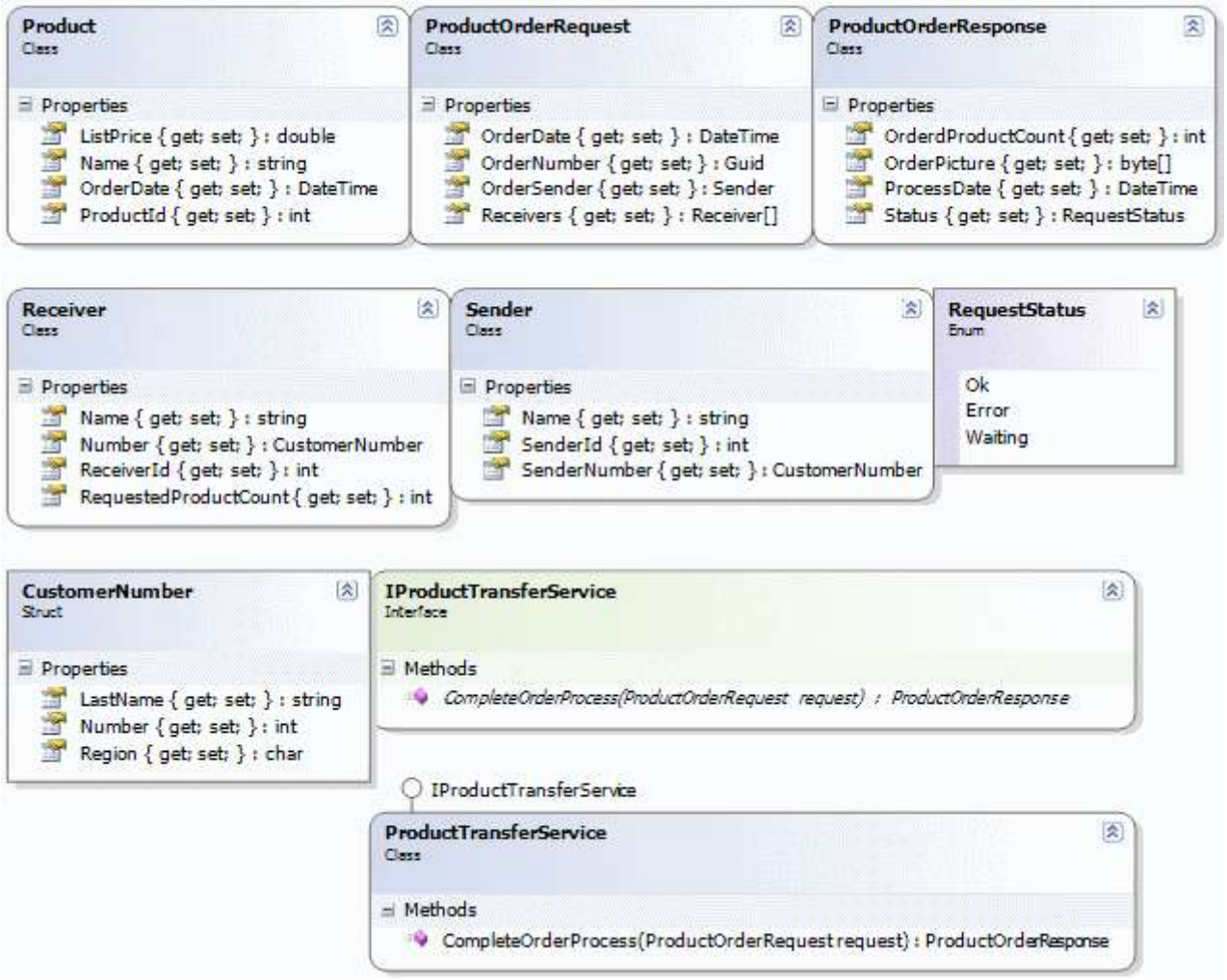
Yazımıza servis odaklı uygulamalarda mesajların yerini konumlandırmaya çalışarak başladık. özellikle **SOAP** tabanlı bu mesajlar gerektiğinde özel olarak tasarlanabilirler. **WCF** tarafında bunu gerçekleştirebilmek için **Mesaj Sözleşmelerinden** yararlanılır. Mesaj sözleşmelerinin ne zaman kullanılacağına karar verilmesi genellikle zordur. Farklı platformlar için destek verebilme imkanı**(Interoperability)** ve mesaj kontrolü çoğunlukla karar vermeyi kolaylaştırmaktadır. Gerçektende servis tarafından istemciye gönderilecek veya alınacak mesajların farklı platformlara destek verebilecek şekilde tasarlanması gerektiği durumlarda özel **Mesaj Sözleşmeleri** göz önüne alınabilir. Diğer taraftan **Mesaj Sözleşmeleri** ile taşınacak bilginin değişik parçalarının **SOAP** paketinin **Header** veya **Body** kısmına ayrıştırılması ve bu sayede de, gerekli olmayan parçaların mesaj ile birlikte taşınmaması sağlanabilmektedir. Bu tam anlamıyla aradaki mesajlaşmanın kontrol altına alınması anlamına gelmektedir. Hatta, istemci ve servislerin belirli olduğu vakalarda, arada özel bir mesaj formatına göre veri içeriğinin taşınmasıda mümkün olabilir. Diğer taraftan göz ardı edilmemesi gereken bir noktada, mesaj seviyesinde güvenliktir. Mesaj Sözleşmeleri kullanılırken bir tipin **SOAP** zarfının içerisindeki yayılımı belirlenebildiği gibi*(hangi kısımları Header' da olacak vb...)* verinin **şifrelenmeside(Encryption)** özelleştirilebilir. Böylece vakaya göre bir mesaj deseninin oluşturulması ve kullanılması mümkün olabilmektedir.

NOT : *çoğu durumda **Mesaj Sözleşmeleri** yerine **Veri Sözleşmelerininde** aynı işi yapıyor olduğu görülür. Ancak genel kanıya göre, eğer bir tip **n** sayıda mesaj içerisinde kullanılacaksa(yani **reusable type** olarak düşünülebilirse) **Veri Sözleşmesi** olarak tanımlanması önerilmektedir.*
*Ancak **tip(type)** sadece **istek/cevap(Request/Respone)** modeline göre bir kereliğine kullanılıyorsa, **Mesaj Sözleşmesi** olacak şekilde tanımlanır.*

Mesaj sözleşmelerinin uygulanması son derece kolaydır. Ancak dikkat edilmesi gereken noktalar vardır. Herşeyden
önce **MessageContract**, **MessageHeader**, **MessageBodyMember**,**MessageHeaderArray** gibi **niteliklerinden(attributes)** yararlanılarak **Mesaj Sözleşmesi** tanımlanabilmektedir. Bununla birlikte servis operasyonlarında **Mesaj Sözleşmelerinin** kullanılması söz konusu ise metod yapısında uyulması gereken kurallar vardır. Buna göre metod desenleri aşağıdaki örnekler olduğu gibi olmalıdır. Bu tablodaki örnek kullanımlarda yer alan**ProductOrderResponse** ve **ProductOrderRequest** isimli tipler örnek Mesaj Sözleşmesi sınıflarıdır.

Geçerli Mesaj Sözleşme Kullanımları	
[OperationContract] **ProductOrderResponse** CompleteOrderProcess(**ProductOrderRequest**request);	*Op*
[OperationContract] **ProductOrderResponse** CompleteOrderProcess();	*Op* *dö*
[OperationContract] **void** CompleteOrderPrococes2(**ProductOrderRequest** request);	*Op* *dö*
Geçersiz Mesaj Sözleşme Kullanımları	
[OperationContract] **int** CompleteOrderProcess(**ProductOrderRequest** request);	*Pa* *ha*
[OperationContract] void CompleteOrderProcess(**ProductOrderRequest** request1,**ProductOrderRequest** request2);	*Bi*

Bu kısa teorik bilgileri devam ettireceğiz ancak dilerseniz basit bir örnek üzerinden ilerleyerek devam edelim. öncelikli olarak bir **WCF Sınıf Kütüphanesi** projesi oluşturduğumuzu düşünelim. Bu projemizde yer alacak olan tiplerin sınıf diygramındaki görüntüsü aşağıdaki gibi tasarlanabilir.



Sınıf diyagramı(Class Diagram) gözümüzü korkutmasın. Senaryomuz aslında sadece **Mesaj Sözleşmelerinin** nasıl kullanılacağını göstermeye yönelik olduğundan çok anlamlı olmayan operasyonlar içermekte. Bu yüzden örnek olarak bir sipariş sürecine özel mesajları tasarladığımız bir durum söz konusu. Kullanılan tiplerin içerikleri sırasıyla aşağıdaki gibidir;

Product Sınıfı. *(**Veri Sözleşmesi-Data Contract** olarak tanımlanmıştır)*

```
using System;
using System.Runtime.Serialization;

namespace ProductTransferLib
{
   [DataContract(Namespace =
"http://Northwind/ProductTransferService/Product")]
   public class Product
   {
      [DataMember(Order=0)]
      public int ProductId { get; set; }
```

```
        [DataMember(Order=1)]
        public string Name { get; set; }
        [DataMember(Order=2)]
        public double ListPrice { get; set; }
        [DataMember(Order=3)]
        public DateTime OrderDate { get; set; }
    }
}
```

CustomerNumber yapısı-struct.***(Veri Sözleşmesi-Data Contract** olarak tanımlanmıştır)*

```
using System.Runtime.Serialization;

namespace ProductTransferLib
{
    [DataContract(Namespace =
"http://Northwind/ProductTransferService/CustomerNumber")]
    public struct CustomerNumber
    {
        [DataMember]
        public char Region { get; set; }
        [DataMember]
        public int Number { get; set; }
        [DataMember]
        public string LastName { get; set; }
    }
}
```

Receiver Sınıfı***(Veri Sözleşmesi** olarak tanımlanmıştır)*

```
using System.Runtime.Serialization;

namespace ProductTransferLib
{
    [DataContract(Namespace="http://Northwind/ProductTransferService/Receiver")]
    public class Receiver
    {
        [DataMember]
        public int ReceiverId { get; set; }
        [DataMember]
        public string Name { get; set; }
        [DataMember]
        public CustomerNumber Number { get; set; }
        [DataMember]
        public int RequestedProductCount { get; set; }
```

```
    }
}
```

Sender Sınıfı.*(**Veri Sözleşmesi** olarak tanımlanmıştır)*

```
using System.Runtime.Serialization;

namespace ProductTransferLib
{
    [DataContract(Namespace = "http://Northwind/ProductTransferService/Sender")]
    public class Sender
    {
        [DataMember]
        public int SenderId { get; set; }
        [DataMember]
        public string Name { get; set; }
        [DataMember]
        public CustomerNumber SenderNumber { get; set; }
    }
}
```

RequestStatus Enum sabiti.*(**Veri Sözleşmesi** olarak tanımlanmıştır. Enum sabiti söz konusu olduğu için değerler DataMember yerine EnumMember isimli nitelik ile işaretlenmiştir.)*

```
using System.Runtime.Serialization;

namespace ProductTransferLib
{
    [DataContract(Namespace =
"http://Northwind/ProductTransferService/RequestStatus")]
    public enum RequestStatus
    {
        [EnumMember]
        Ok,
        [EnumMember]
        Error,
        [EnumMember]
        Waiting
    }
}
```

Buraya kadar tanımladığımız tipler içerisinde sınıf, yapı ve enum sabiti tipleri söz konusudur. Bu tipler birer **Veri Sözleşmesi** olarak tanımlanmıştır ve **Mesaj Sözleşmeleri**içerisinde ele alınmaktadır. Yazımızın konusu

olan **Mesaj Sözleşmelerinden** iki adet tanımlanmalıdır. Bu tanımlamalardan birisi **istek(Request)** diğer ise **cevap(Response)** içeriklerinin yapısını işaret etmektedir. Bir başka deyişle, istemciden servise gelecek veya geriye döndürülecek olan **SOAP** zarflarının içerikleri kod yardımıyla belirlenmektedir. İstemci tarafından gelecek olan taleplere ait Mesaj Sözleşmesi aşağıdaki kod parçasında olduğu gibi tanımlanmıştır.

```
using System;
using System.ServiceModel;

namespace ProductTransferLib
{
   [MessageContract]
   public class ProductOrderRequest
   {
      #region Header Kısmına yazılacak özellikler

      [MessageHeader]
      public Guid OrderNumber { get; set; }
      [MessageHeader]
      public DateTime OrderDate { get; set; }
      [MessageHeader]
      public Product OrderedProduct { get; set; }

      #endregion

      #region Body kısmına yazılacak özellikler

      [MessageBodyMember(ProtectionLevel=System.Net.Security.ProtectionLevel.None)] // ProtectionLevel için varsayılan değre None' dur.
      public Sender OrderSender { get; set; }
      [MessageBodyMember]
      public Receiver[] Receivers { get; set; }

      #endregion
   }
}
```

ProductOrderRequest isimli sınıf bir **Mesaj Sözleşmesi** olacak şekilde tanımlanmıştır. Bu nedenle **MessageContract** niteliği ile imzalanmıştır. Bu nitelik sadece **sınıf(Class)** veya**yapılara(Structs)** uygulanabilir. Yazımızın başında mesajın **Header** ve **Body** kısımlarından bahsetmiştik. **Header** kısmında taşınacak olan **alan(Field)** veya**özellikleri(Property)** belirtmek için **MessageHeader** niteliği kullanılmaktadır. örnektende görüldüğü gibi, **Header** kısmında **Guid**, **DateTime** gibi bilinen tipler dışında **Product**isimli geliştirici tanımlı bir sınıfada yer verilmiştir. Söz konusu tipler mesaj içerisine alınırken serileştirilmektedir. Bu nedenle **Product** sınıfı ve

diğer geliştirici tanımlı tipler birer**Veri Sözleşmesi** olarak tanımlanmıştır. **Body** kısmında yer alacak özellik veya alanlar ise **MessageBodyMember** niteliği ile tanımlanırlar. Yine **Body** kısmındada, **Sender** ve**Receiver** isimli geliştirici tanımlı **Veri Sözleşmelerine** yer verilmektedir. özellikle **Receiver** tipinden bir **Array** kullanıldığınada dikkat edilmelidir.

***NOT : Header** veya **Body** kısımlarında **Array**' ler kullanılıyorsa **MessageHeader** ve **MessageBodyMember** nitelikleri bu dizilerin elemanlarını bir elementin **alt elementleri(Child Element)**olacak şekilde konumlandırır. örneğin;*

```
<diziAdi>
   <diziElemanTipiAdi>içeriği</diziElemanTipiAdi>
   <diziElemanTipiAdi>içeriği</diziElemanTipiAdi>
</diziAdi>
```

*Ancak istenirse her bir dizi elemanının ayrı birer boğum olarak ele alınması sağlanabilir. Bunun için **MessageHeaderArray** niteliği kullanılır.*

```
<diziAdi>içeriği</diziAdi>
<diziAdi>içeriği</diziAdi>
```

*Yanlız bu nitelik sadece dizilere uygulanabilir. Bir başka deyişle **koleksiyonlara** uygulanamamaktadır.*

*Eğer **SOAP** içeriğinde **byte** tipinden bir diziye yer verilmişse **MessageHeader** veya **MessageBodyMember** niteliklerinin kullanılması halinde bunlar doğrudan **Base64** tipine dönüştürülürler. Ancak, eğer **MessageHeaderArray** niteliği kullanılıyorsa, ele alınan serileştirme tipine göre**(DataContractSerializer, XmlSerializer gibi)** bir aktarım gerçekleştirilir.*

MessageHeader ve **MessageBodyMember** niteliklerinde yer alan **ProtectionLevel** özelliği kullanılarak **dijital olarak imzalama(Sign)** veya **şifreleme(Encryption)** sağlanabilir. **ProtectionLevel**özelliği **System.Net.Security.ProtectionLevel enum** sabiti tipinden bir değer alabilir. Bu değerler **None**, **EncryptAndSign**, **Sign** olabilir. Varsayılan değeri **None**' dur. **Sign** seçilirse dijital imzalama söz konusudur. **EncryptAndSign** seçilirsede **şifreleme** ve **dijital imzalama** söz konusudur. Elbette **None** dışındaki değerlerin işe yaraması için WCF çalışma ortamına yönelik olaraktan gerekli **Binding** ve **Behavior** ayarlamalarının yapılması gerekir. Aksi durumda çalışma zamanında doğrulama işlemi sırasında bir istisnası alınır. **ProtectionLevel**, **Header** kısmında her bir eleman için ayrı ayrı uygulanmaktadır. **Body** kısmı söz konusu olduğunda ise kaç eleman olursa olsun hepsi için aynı **ProtectionLevel** seviyesi söz konusudur. Buna göre**MessageBodyMember** niteliği içinde seviyesi yüksek olan **ProtectionLevel** değeri, diğerleri içinde uygulanır. Söz gelimi

3 farklı **MessageBodyMember** için sırasıyla **None**,**EncryptAndSign**, **Sign** değerleri belirlenmişse, tüm mesaj gövdesi için **EncrptyAndSign** seçeneği göz önüne alınmaktadır.

***NOT : SOAP** ile ilişkili web servisi standartlarında **1.1** versiyonu için **Actor** ve **1.2** için **Role** adı verilen bir özellik yer almaktadır. Bu özelliğin değerini **WCF** tarafında ele almak için**MessageHeader** niteliğinin **Actor** özelliği kullanılır. Bunun dışında **MustUnderstand** ve **Relay** özelliklerindende yararlanılarak, **SOAP** standarlarına göre bazı niteliklerin mesajlaşma süreçlerine kazandırılması da sağlanabilir.*

İstemciye gönderilecek cevap mesajının içeriği ise ProductOrderResponse isimli Mesaj Sözleşmesi ile tanımlanmaktadır.

```
using System.ServiceModel;
using System;

namespace ProductTransferLib
{
   [MessageContract]
   public class ProductOrderResponse
   {
      [MessageBodyMember]
      public RequestStatus Status { get; set; }

      [MessageBodyMember]
      public DateTime ProcessDate{ get; set; }

      [MessageBodyMember]
      public byte[] OrderPicture { get; set; } // Burada byte[] tipinden bir dizi söz konusu olduğu için SOAP body' si içerisinde Base64 tipinden bir kodlama(encoding) söz konusu olacaktır

      [MessageHeader]
      public int OrderdProductCount { get; set; }
   }
}
```

ProductOrderResponse tipi **Mesaj Sözleşmesi** olarak tanımlanırken amaç istemci tarafına gönderilecek olan **SOAP** mesajının **Header** ve **Body** kısımlarında neler olacağına karar verilmesidir. Dikkat edileceği üzere **Body** kısmında **RequestStatus enum** sabiti, **DateTime** ve **byte[]** dizisi tipinden bir içerik yer almaktadır. Diğer taraftan **Header** kısmında ise örnek olarak **int**veri tipinden bir değer döndürülmektedir.

NOT : *Bir tipin hem* ***MessageContract*** *hemde* ***DataContract*** *olacak şekilde tanımlanması da mümkündür. Böyle bir vakada,* ***WCF*** *çalışma zamanında servis operasyonları uygulanırken, söz konusu tip için* ***Mesaj Sözleşmesi*** *kriterleri göz önüne alınmaktadır.*

Artık istemci ve servis arasında dolaşacak olan **SOAP** mesajlarına ait içerikler tanımlanmıştır. Dolayısıyla bu mesajlaşma modelini kullanacak bir **Servis Sözleşmesi** ve uygulayıcı sınıfı tasarlanabilir. Dikkat edileceği üzere **Servis Sözleşmesinde** yer alan **CompleteOrderProcess** isimli operasyonun dönüş tipi ve parametresi birer **Mesaj Sözleşmesidir**.

```
using System.ServiceModel;

namespace ProductTransferLib
{
   [ServiceContract(
                        Name="ProductTransferService"
                        ,Namespace="http://Northwind/ProductTransferService")]
   public interface IProductTransferService
   {
      [OperationContract]
      ProductOrderResponse CompleteOrderProcess(ProductOrderRequest request);
   }
}
```

Operasyonun uygulanışı içinse aşağıdaki gibi bir kod örneği geliştirilebilir.

```
using System;
using System.IO;

namespace ProductTransferLib
{
public class ProductTransferService
   :IProductTransferService
{
   #region IProductTransferService Members

   public ProductOrderResponse CompleteOrderProcess(ProductOrderRequest request)
   {
      DateTime requestDate = request.OrderDate;
      Guid requestOrderNumber = request.OrderNumber;
      Sender requestSender = request.OrderSender;
      Receiver[] requestReceivers = request.Receivers;

      int orderedProductCount = 0;
      foreach (Receiver receiver in requestReceivers)
```

```
        {
            orderedProductCount += receiver.RequestedProductCount;
        }

        // Not : XP_HDD.gif resminin byte içeriğinin dizi boyutu istemci tarafına
gönderilebilecek varsayılan dizi limini aşabilir. Bu nedenle istemci tarafındaki
konfigurasyon ayarlarında maxArrayLength değerinin bilinçli olarak arttırılması
gerekebilir.
        return new ProductOrderResponse
                    {
                        ProcessDate=DateTime.Now,
                        Status= RequestStatus.Ok,
                        OrderPicture=File.ReadAllBytes(System.Environment.CurrentDirec
tory + "\\XP_HDD.gif"),
                        OrderdProductCount=orderedProductCount
                    };
        }

        #endregion
    }
}
```

Burada **request** değişkeninden yararlanılarak istemci tarafından gelen **SOAP** paketindeki mesaj içeriği ele alınmakta ve kullanılmaktadır. Sembolik olarak paket içerisinde gelen **Receivers**dizisindeki her bir **Receiver** nesne örneğinin sipariş sayısının toplamı tespit edilmektedir. Ayrıca örnek **byte[]** içeriği döndürülmesi için küçük bir resim dosyasından(XP_HDD.gif) yararlanılmaktadır. İşlemin tarihi, durumu, sipariş ile ilişkili resim ve toplam sipariş sayısı bilgileri kullanılaraktanda bir cevap mesajı oluşturulmakta ve istemci tarafına gönderilmektedir.

***NOT : SOAP** mesajlarının içerikleri aslında **XML** tabanlıdır. Bu içeriği yönetirken **Mesaj Sözleşmeleri**, nesne tabanlı bir modeli ele alabilmemizi sağlamaktadır. Bir başka deyişle, kod tarafında **XML** yapısı ile uğraşmak yerine, nesne tabanlı bir modeli kullanarak mesaj içeriğini kolayca oluşturabilmemiz olanaklı hale gelmektedir ki bu geliştirme süreci için önemli bir avantajdır.*

Bu işlemlerin tamamlanmasının ardından servis kütüphanesini **Host** edecek basit bir uygulama geliştirilebilir. Biz örneğimizde her zaman olduğu gibi, sunucu ve istemci tarafları için birer **Console**uygulaması geliştiriyor olacağız. Sunucu uygulama kodları ve konfigurasyon içeriği aşağıdaki kod parçalarında olduğu gibi tanımlanabilirler.

Sunucu uygulama kodları;

```
using System;
using System.ServiceModel;
```

```
using ProductTransferLib;

namespace ServerApp
{
   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(ProductTransferService));
         host.Open();
         Console.WriteLine("Servis dinlemede.\nKapatmak için bir tuşa basınız.");
         Console.ReadLine();
         host.Close();
      }
   }
}
```

Sunucu tarafı konfigurasyon içeriği;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <behaviors>
         <serviceBehaviors>
            <behavior name="ProductTransferServiceBehavior">
               <serviceDebug includeExceptionDetailInFaults="true" />
               <serviceMetadata />
            </behavior>
         </serviceBehaviors>
      </behaviors>
      <services>
         <service behaviorConfiguration="ProductTransferServiceBehavior"
name="ProductTransferLib.ProductTransferService">
            <endpoint address="" binding="basicHttpBinding"
bindingConfiguration="" name="ProductTransferServiceHttpEndPoint"
contract="ProductTransferLib.IProductTransferService" />
            <endpoint address="Mex" binding="mexHttpBinding"
bindingConfiguration="" name="ProductTransferServiceMexEndPoint"
contract="IMetadataExchange" />
            <host>
               <baseAddresses>
                  <add
baseAddress="http://buraksenyurt:1000/ProductTransferService" />
               </baseAddresses>
            </host>
```

```
        </service>
      </services>
   </system.serviceModel>
</configuration>
```

Sunucu uygulama basit olarak **HTTP** tabanlı bir sunum yapmakta ve **BasicHttpBinding** bağlayıcı tipini ele almaktadır. Bununla birlikte istemci tarafının, servise ait **Metadata** bilgisini çekebilmesi için**IMetadataExchange** arayüzünü kullanan bir **MexHttpBinding EndPoint**' ide kullanılmaktadır.

İstemci uygulamamızı kullanırken yine **Add Service Reference** seçeneği ile aynı **solution** içerisinde yer alan örnek servise ait referans üretimini gerçekleştirebiliriz. İstemci tarafına ait konfigurasyon içeriği aşağıdaki gibidir*(Bu içerik **Add Service Reference** seçeneğinin kullanılması sonucunda otomatik olarak üretilmektedir.)*

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <bindings>
         <basicHttpBinding>
            <binding name="ProductTransferServiceHttpEndPoint"
closeTimeout="00:01:00" openTimeout="00:01:00" receiveTimeout="00:10:00"
sendTimeout="00:01:00" allowCookies="false" bypassProxyOnLocal="false"
hostNameComparisonMode="StrongWildcard" maxBufferSize="65536"
maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
messageEncoding="Text" textEncoding="utf-8" transferMode="Buffered"
useDefaultWebProxy="true">
               <readerQuotas maxDepth="32" maxStringContentLength="8192"
maxArrayLength="163840" maxBytesPerRead="4096"
maxNameTableCharCount="16384"/>
               <security mode="None">
                  <transport clientCredentialType="None" proxyCredentialType="None"
realm="" />
                  <message clientCredentialType="UserName" algorithmSuite="Default" />
               </security>
            </binding>
         </basicHttpBinding>
      </bindings>
      <client>
         <endpoint address="http://buraksenyurt:1000/ProductTransferService"
binding="basicHttpBinding" bindingConfiguration="ProductTransferServiceHttpE
ndPoint"
contract="ProductTransferServiceReference.ProductTransferService"
name="ProductTransferServiceHttpEndPoint" />
      </client>
```

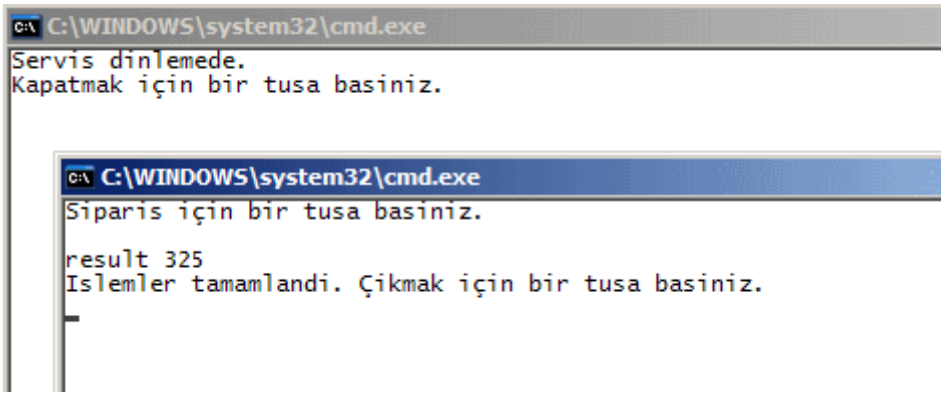
```
   </system.serviceModel>
</configuration>
```

İstemci tarafındaki örnek kod içeriği ise aşağıdaki gibidir.

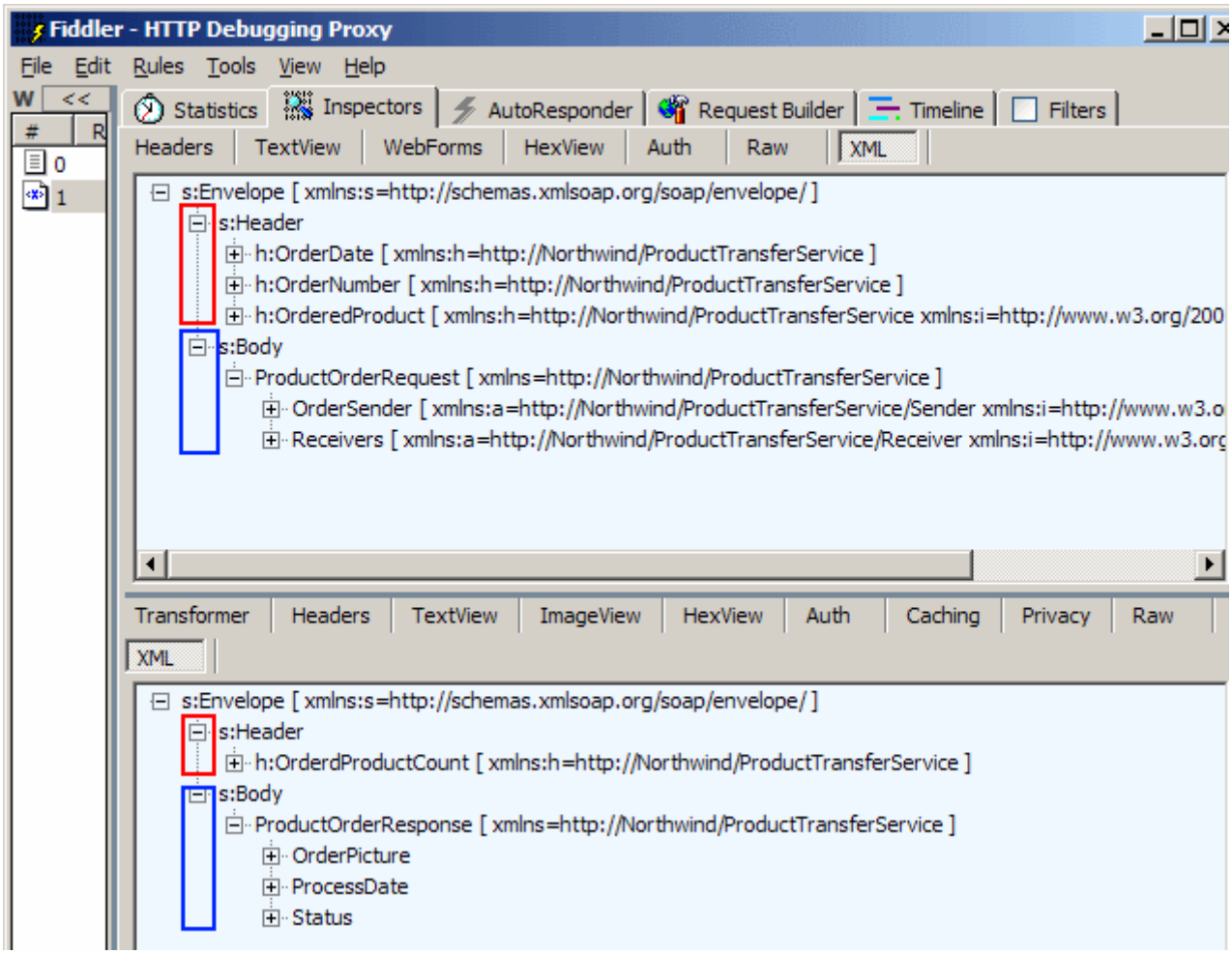
```
using System;
using System.IO;
using ClientApp.ProductTransferServiceReference;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         ProductTransferServiceClient client = new
ProductTransferServiceClient("ProductTransferServiceHttpEndPoint");

         Sender sndr = new Sender
                        {
                           Name="Burak Selim",
                           SenderId=10001,
                           SenderNumber=new CustomerNumber{ Number=1, Region='A',
LastName="SENYURT"}
                        };

         Receiver[] receivers = {
            new Receiver{ Name="Bil", Number=new CustomerNumber{
LastName="Geyts", Region='B', Number=1}, ReceiverId=10002,
RequestedProductCount=100},
            new Receiver{ Name="Deyv", Number=new CustomerNumber{
LastName="Masteyn", Region='C', Number=2}, ReceiverId=10003,
RequestedProductCount=150},
            new Receiver{ Name="Co", Number=new CustomerNumber{
LastName="Satriyani", Region='C', Number=3}, ReceiverId=10055,
RequestedProductCount=75}
         };

         RequestStatus requestStatus;
         DateTime processDate;
         byte[] orderPicture;

         Console.WriteLine("Sipariş için bir tuşa basınız.");
         Console.ReadLine();
```

```
        int result=client.CompleteOrderProcess(
                                                DateTime.Now,
                                                Guid.NewGuid(),
                                                new Product{ ProductId=1, Name="Her
Yönüyle WCF", ListPrice=10, OrderDate=DateTime.Now},
                                                sndr,
                                                receivers,
                                                out orderPicture,
                                                out processDate,
                                                out requestStatus);

        Console.WriteLine("result {0}",result.ToString());
        File.WriteAllBytes(System.Environment.CurrentDirectory +
"\\ResponsePicture.gif", orderPicture);

        Console.WriteLine("İşlemler tamamlandı. çıkmak için bir tuşa basınız.");
        Console.ReadLine();
      }
   }
}
```

İstemci uygulamada servise ait proxy nesnesi örneklendikten sonra **CompleteOrderProcess** metodunun ihtiyacı olan parametreler hazırlanmaktadır. **CompleteOrderProcess** metodu aslında**ProcessOrderResponse Mesaj Sözleşmesi** tipinden bir parametre almaktadır. Ne varki istemci tarafında metodun uygulanış şekli biraz farklıdır. Herşeyden önce, servise gönderilecek**SOAP** paketi içerisinde yer alacak **Header** ve **Body** elementlerinin her biri, istemci tarafında ayrı birer **metod parametresi** şekline ele alınmaktadır. Metodun çağırılması sonucu istemciye dönecek olan **SOAP** mesajındaki **Header** kısmında yer alan **int** değer aslında istemci tarafında, **CompleteOrderProcess**' in dönüş değeridir. Yine istemciye döndürülen ve **Body** kısmında yer alan **orderPicture**,**processDate** ve **requestStatus** değişkenleri ise, **CompleteOrderProcess** metodunun **out** tipinden parametreleri olarak ele alınmaktadır. **orderPicture** değişkeni bir**byte[]** dizisi olarak mesaj içeriğinden toparlanmakta ve fiziki olarak istemci tarafındaki bir dosyaya yazdırılmaktadır. Bu tahmin edileceği üzere servis tarafından gönderilen resimdir. Projemizde hem sunucu hemde istemci uygulamamızı çalıştırdığımızda aşağıdaki ekran görüntüsünde yer alan sonuçları elde ederiz.



Aslında burada şaşırtıcı bir sonuç yoktur. Nesne tabanlı olacak şekilde istemci ve servis arasındaki tipler kolay bir şekilde kullanılmıştır. Ancak bizim için önemli olan arka planda hareket eden **SOAP** mesajlarının içeriklerinin ne hale geldiğidir. Bu amaçla **Fiddler** isimli **HTTP Debugging** aracından yararlanırsak, örneğin çalıştırılması sonrasında ağ trafiğinde, aşağıdaki ekran görüntüsünde yer alan mesajlaşmanın oluştuğunu görürüz.



Dikkat edileceği üzere, Mesaj Sözleşmelerinde **Header** ve **Body** kısımlarında hangi bilgilerin yer almasını istiyorsak buna göre bir ağaç yapısı oluşmuştur. **Request** kısmına ait olan **SOAP** zarfının **XML** içeriği tam olarak aşağıdaki gibidir.

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
  <s:Header>
    <h:OrderDate xmlns:h="http://Northwind/ProductTransferService">2009-02-
08T01:53:03+02:00</h:OrderDate>
    <h:OrderNumber xmlns:h="http://Northwind/ProductTransferService">9476df3d-
0f74-4643-9fcf-b7344d5da37d</h:OrderNumber>
    <h:OrderedProduct xmlns:h="http://Northwind/ProductTransferService"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
      <ProductId
xmlns="http://Northwind/ProductTransferService/Product">1</ProductId>
      <Name xmlns="http://Northwind/ProductTransferService/Product">Her Yönüyle
WCF</Name>
      <ListPrice
xmlns="http://Northwind/ProductTransferService/Product">10</ListPrice>
      <OrderDate xmlns="http://Northwind/ProductTransferService/Product">2009-02-
08T01:53:03+02:00</OrderDate>
    </h:OrderedProduct>
  </s:Header>
  <s:Body>
    <ProductOrderRequest xmlns="http://Northwind/ProductTransferService">
      <OrderSender xmlns:a="http://Northwind/ProductTransferService/Sender"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
        <a:Name>Burak Selim</a:Name>
        <a:SenderId>10001</a:SenderId>
        <a:SenderNumber
xmlns:b="http://Northwind/ProductTransferService/CustomerNumber">
          <b:LastName>SENYURT</b:LastName>
          <b:Number>1</b:Number>
          <b:Region>65</b:Region>
        </a:SenderNumber>
      </OrderSender>
      <Receivers xmlns:a="http://Northwind/ProductTransferService/Receiver"
xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
        <a:Receiver>
          <a:Name>Bil</a:Name>
          <a:Number
xmlns:b="http://Northwind/ProductTransferService/CustomerNumber">
            <b:LastName>Geyts</b:LastName>
            <b:Number>1</b:Number>
            <b:Region>66</b:Region>
          </a:Number>
          <a:ReceiverId>10002</a:ReceiverId>
          <a:RequestedProductCount>100</a:RequestedProductCount>
        </a:Receiver>
        <a:Receiver>
```

```
            <a:Name>Deyv</a:Name>
            <a:Number
xmlns:b="http://Northwind/ProductTransferService/CustomerNumber">
               <b:LastName>Masteyn</b:LastName>
               <b:Number>2</b:Number>
               <b:Region>67</b:Region>
            </a:Number>
            <a:ReceiverId>10003</a:ReceiverId>
            <a:RequestedProductCount>150</a:RequestedProductCount>
         </a:Receiver>
         <a:Receiver>
            <a:Name>Co</a:Name>
            <a:Number
xmlns:b="http://Northwind/ProductTransferService/CustomerNumber">
               <b:LastName>Satriyani</b:LastName>
               <b:Number>3</b:Number>
               <b:Region>67</b:Region>
            </a:Number>
            <a:ReceiverId>10055</a:ReceiverId>
            <a:RequestedProductCount>75</a:RequestedProductCount>
         </a:Receiver>
      </Receivers>
   </ProductOrderRequest>
</s:Body>
</s:Envelope>
```

Bu **XML** içeriği incelendiğinde tam olarak **Mesaj Sözleşmesinde** belirttiğimiz kriterlere uyulduğu görülmektedir. Söz gelimi **Header** kısmında **OrderDate**, **OrderNumber** ve **OrderProduct**elementleri yer almaktayken, **Body** kısmında **ProductOrderRequest** elementi tarafından sarmalanmış olan, **OrderSender** ve **Receivers** elementleri bulunmaktadır. **Receivers** aslında **Receiver[]dizisinin** kullanılması nedeni ile kendi içerisinde birden fazla **Receiver** alt elementi içermektedir. Burada geliştirici tanımlı tiplerin*(Product,Receiver,Sender gibi)* **Veri Sözleşmesi** olarak tanımlanmaları nedeniyle **XML** elemetleri içerisine aktarılmış olmalarıda gözden kaçırılmamalıdır. Yine istemciye **dönen mesajın(Response)** tam içeriğine bakıldığında aşağıdakine benzer bir **SOAP** çıktısı ile karşılaşılmaktadır.

```
<s:Envelope xmlns:s="http://schemas.xmlsoap.org/soap/envelope/">
   <s:Header>
      <h:OrderdProductCount
xmlns:h="http://Northwind/ProductTransferService">325</h:OrderdProductCount>
   </s:Header>
   <s:Body>
      <ProductOrderResponse
```

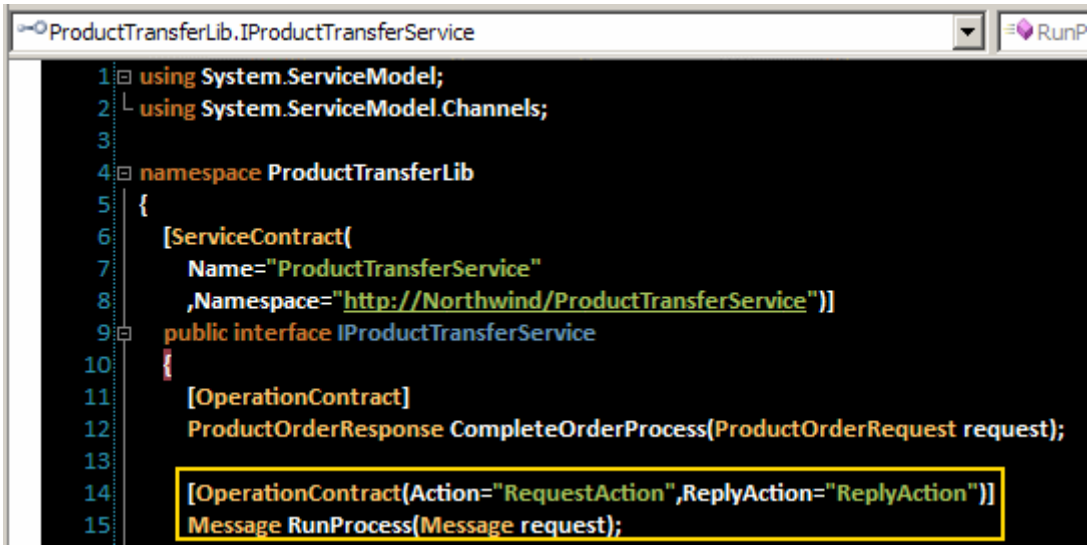
xmlns="http://Northwind/ProductTransferService">
<OrderPicture>R0lGODlhIAAgAPcAAAAAAB8hNR4hODMpHCQgLiIkNyYqRysvUC4wQy
0wTi0xUzU3SzE1VjE1WzU5Xjs
+VDw/WDU6Yzk+ZTk/bDg5dTFILDl0FzpGRz5BXD5CYz1CbD5EdVYzLXI/SF9OMF1
WP05tK0JoN0pxNVF5JVR1M2lfPn
FHKn5GP3VpL31vN31wL3d0OEZIVkZUSUFFY0FGbURIZEZKaklLYExPaEJIek1SaExTd1VWalNXd1prWFp1Rl96bWRcS
2JNYGtrRnhuRGdlaGNkdmh5eTc3mDxDxT5P2Dxdxj5d3T9g1z9r40VMg0pRiVZbhFRbl1hkilljl0Nxr0h2t1tlqFVzqGFmjW
ZpgWtthGlti2Npl25xiGd6l25xkG1ym3ByhnN0inl2h3N2kXF2m3R4lnR5mXp8lWNqq2pzp2xzsXN5q3l+pXp/qnR7tEFW0E
Jt0EN85l+fGkmLJ0mSIVOKJFWJNVeXJFqpH1y2GFqiJGOWKWK3Hm2oIV6KVFnFGGn
NF3POGnrSGWHJIHbNInvQIHiDiH
qAonyCuH6Qr0qK3kOF7EeP8EqU8Vua6lWb8miW1XGYzWWk8XSq73et8H+x8YM2N5oxJ7MsG4pfPIN4K5J7LpR0OKFV
KaB2HbB+FYNtQpp8eKRNR7JyRtAqE+UsEfg+D/Q2ENRNE9dyH8F0NuxHD/pJDfxWDO
1+HP1lDv12EbeDC7OaFKmUIqm
XNruDK72nEKyMfoHSHYTRJMeKCMiRCN+GH9WWBc2xDNqjA8GnIv2GE/uaF/CGIf2W
IOCrBOS1BfqlHPq0HfysIfu3I+vG
A/fXBPrcFvzeKfzeOP3gOIKDj4yDlZCCkYaKrYGHs4CGu4SJtYWLvYqPtIiNuo6TvJWXp
pGVu5+ivoiOwIySwpGWxJSZxJ
ecyJmdxJmeyJ2hw52iy4i28JK88aGlxqGmzKSpzqmsyaar0Kmt0a6xzK2x07a4zbG11bS417a52Lq907m92ZzC8b7B3K3J
7aXH8MTG3MjL4c7R5dHT5gAAAAAAAAAAACH5BAEAAP8ALAAAAAAgACAAAAj/AP8JH
EiwoMGDCNvNsCcvHjx37tixW
5cuHTpy5MSB++atW7UM2BDaszduGxkyXrp0AWMFTBYrYqpY+8bxmxpu0xDSs5eGTIIX
DjRIcCBBgoYAWxCE4ditG45
q1Q7mo0ePEJkFCAoM8GCCA4ECBWAsCAOuo1OPB/nJgzdP3z5auY4Ra6UK1ad3+aaKa1r
NBreoBvM5hLf21K9jv+iOCu
VlkLhv1TxWe1ponsF58DK707bKWDFfrFKRopRDzpwdY8II2QMHS2WD9B5C9BRMri9Vojvo
oGOnT58/f/pYoFLonMF0E
CeCClZsLm5RN0bw/g2cTwUsa74ZRAdP4jhTxIr9/2KlalSlG3Tq8KHeh88DJ2uobU+QThw0YLI
yWdJUitSk0nX4VosgfcT
BhAsNoGHQQ9es8wwLNzjhhA01nNADCXTk0YcftgQCSAhmwGeGQe6MQ006UjhyT

zmPJGLEEl
yAcIeGgXTIBwtrPLHGG
QXls04Y6URziDn4lMMIIkcM8cUNLcjxhh54iHABE2tg8QSPBOmDDhbsmPFIPfVEwog
bR1CQyS2Y
8LAACx9ckoIZa5SBA
5YDtUMOF+JAAQk+5jyiSBJEBMEJL8Mko0wzyiRTghlnYNFEGe0Q5A453VATRTlFP
oLkEEDowgsyh
jKTDDIxnHHGE2WU
oQ5BF4lDyCL14P8jphtsUNCJLsIkY+gyybzChKlnpGoNQdpoNEUjhmjxBBRtEPHCJro
Umswyia5wJTep
SiHfQN885gQLz4
BBhRMwUHDDLZ/q2oyuLEgjTqplSFEGQeBsRA013dCTDzloUBGEK8KAqgyvu1hRD
bxSSPEEQZA11c01
3HTjDj/6qNPFD
7DM0kssKDzwaLwJN9HEQO9QY8A13fxVzTTTFFJIN95EcwUONdCwBKogS6GEAiM
PhFxEE1Fk0UXhFB0
OTdqEow022i
zN9IgDMbCBEgUYoEADEUQwAQ00NKCAAgd8DfbUSpQ9NQ4EZUCDElZnPcEGG3
DttQNifz01EzgoYfMVB
ME9sLYBAhj
QgAZxr+2113XH7cMKeivhBUHZxBADBA9U/gAGGLjgAgYZYO555pprHgMNCJVu+
umop6766gcFBAA7
</OrderPicture>
<ProcessDate>2009-02-08T01:53:03.46875+02:00</ProcessDate>
<Status>Ok</Status>
</ProductOrderResponse>
</s:Body>
</s:Envelope>

Gözden kaçmayacak olan nokta **OrderPicture** elementinin içeriğidir :) Tahmin edileceği üzere bu elementin içeriği, servis tarafındaki resmimizin **byte[]** dizisi haline geldikten sonra, **SOAP** mesajı içeriğine **Base64** kodlamasına göre serileştirilmiş halidir. Bu içerik, istemci tarafında ters serileştirilme işleminden sonra yine **byte[]** dizisi olacak şekilde ele alınabilmektedir. Bunların haricinde**Header** kısmında **OrderProductCount** elementinin, **Body** kısmında ise **ProductOrderResponse** elementi ile sarmalanmış olan **OrderPicture**, **ProcessDate** ve **Status** alt elementlerinin olduğu görülmektedir.

Mesaj Sözleşmelerinde ele alınan bir diğer durumda türlendirilmemiş versiyonların kullanılmasıdır**(Untyped Message Contracts)**.
Burada **System.ServiceModel.Channels** isim alanında yer alan **Message** sınıfı ele alınmaktadır. **SOAP 1.1** ve **SOAP 1.2** uyumlu mesajları işaret edebilen bu sınıf yardımıyla, istemciden gelen talepler ele alınabilir ve cevaplar oluşturularak**Message** tipinden örnekler üzerinden karşı tarafa gönderilebilir. Son olarak bu durumu değerlendirip makalemizi

tamamlayalım. Bu amaçla **Servis Sözleşmemize** aşağıdaki ekran görüntüsünde yer alan yeni bir operasyon ilave ettiğimizi düşünelim.



RunProcess isimli operasyon parametre ve dönüş değeri olarak **Message** tipini kullanmaktadır. Söz konusu operasyon metodunun **ProductTransferService** içerisindeki uyarlaması ise aşağıdaki gibi yapılabilir.

```
// Untyped Message alıp veren örnek servis operasyonu metodu.
public Message RunProcess(Message request)
{
   // Servise gelen Untyped Message ' ın Body kısmında yer alan Product dizi içeriğini elde etmek için GetBody metodunun generic versiyonundan yararlanılır.
   Product[] products = request.GetBody<Product[]>();

   // İstemciye döndürelecek Untyped Message' ın Body kısmında yer alacak örnek Product içeriği için dizi oluşturulur.
   Product[] resultSet=new Product[products.Length];

   // Gelen mesajın Body kısmından elde edilen dizi üzerinde örnek işlemler yapılır.
   // örnekte ListPrice bilgisi 1 birim arttırılmıştır.
   for (int i = 0; i < products.Length; i++)
   {
      products[i].ListPrice += 1;
      resultSet[i] = products[i];
   }

   // Operasyondan döndürelecek olan Untyped Message oluşturulur.
   // İlk parametre SOAP versiyonunu belirtir. örneğin "SOAP 1.1".
   // İkinci parametre servis operasyonunda ReplyAction özelliğine atanan değerdir.
   // üçüncü parametre ise Body kısmında yer alacak olan nesne örneğidir.
   Message response = Message.CreateMessage(request.Version,
```

```
"ReplyAction",resultSet);

   // Untyped Message geriye döndürülür.
   return response;
}
```

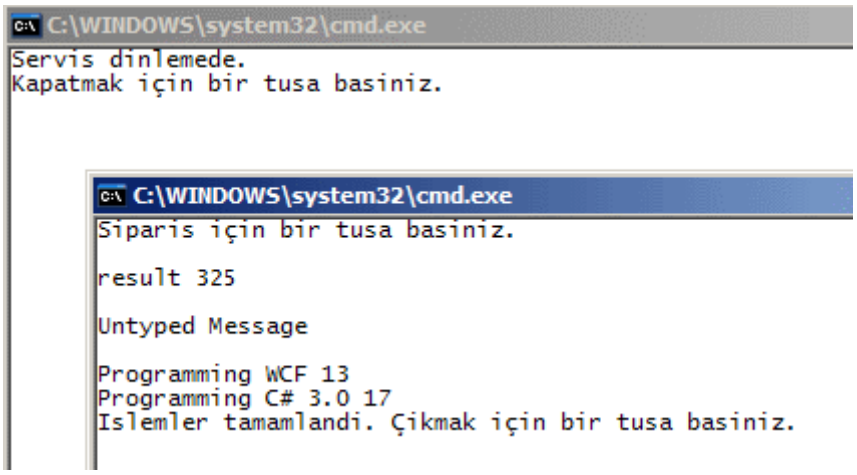
RunProcess isimli servis operasyonu, istemciden gelen mesajın **Body** kısmında yer alan **Product** nesne verilerini ele almakta ve örnek olarak **ListPrice** değerlerini 1 birim arttırarak geriye döndürmektedir. Metoda gelen türlendirilmemiş mesajın gövdesindeki veri içeriğini ele alabilmek için **GetBody<T>** metodundan yararlanılır. Tahmin edileceği üzere metodun kullandığı generic tip üzerinden bir **XML** ters serileştirme işlemi söz konusudur. Nitekim istemciden gelen mesaj **XML** tipindedir ve kod içerisinde nesnel olarak kullanılması gerekmektedir. Bunlara ek olaraktan, servisin istemciye göndereceği türlendirilmemiş mesajın üretimi
için, **Message** sınıfının **static CreateMessage** fonksiyonundan yararlanılır. Metodun **aşırı yüklenmiş(overload)** 11 farklı versiyonu bulunmaktadır. örneğimizde kullandığımız halinde, ilk parametre ile **SOAP** versiyonu, ikinci parametre ile **SOAP Action** adı ve son olarak üçüncü parametre ilede **Body** kısmına gelecek olan nesne örneği belirtilmiştir. Eklenen bu yeni fonksiyonellik nedeniyle istemci tarafında yer alan servis referansınında güncellenmesi gerekmektedir. Bu güncelleme işleminin ardından **RunProcess** isimli operasyon istemci tarafında örnek olarak aşağıdaki kod parçasında görüldüğü gibi kullanılabilir.

```
Console.WriteLine("\nUntyped Message\n");

// Güncel kanal implementasyonundan yararlanarak OperationContextScope nesnesi
örneklenir.
// OperationContextScope nesnesinden yararlanarak gelen ve giden mesajların içerikleri
yönetilebilir, Header, Body gibi kısımlarına müdahale edilebilir.
using (new OperationContextScope(client.InnerChannel))
{
   // Untyped mesaj içerisinde gönderilecek olan Product nesneleri için bir dizi hazırlanır.
   Product[] products =
      {
         new Product{ Name="Programming WCF", ListPrice=12,
OrderDate=DateTime.Now, ProductId=19},
         new Product{ Name="Programming C# 3.0", ListPrice=16,
OrderDate=DateTime.Now, ProductId=21}
      };

   // İstemciden servise gönderilecek olan Untyped Message hazırlanır.
   // İlk parametre mesaj versiyonudur. (SOAP 1.1 gibi).
   // İkinci parametre servis sözleşmesinde RunProcess operasyonunda belirtilen Action
özelliğinin değeridir.
   // üçüncü parametre ise mesaj içeriğinde gönderilecek olan serileştirilebilir nesne
```

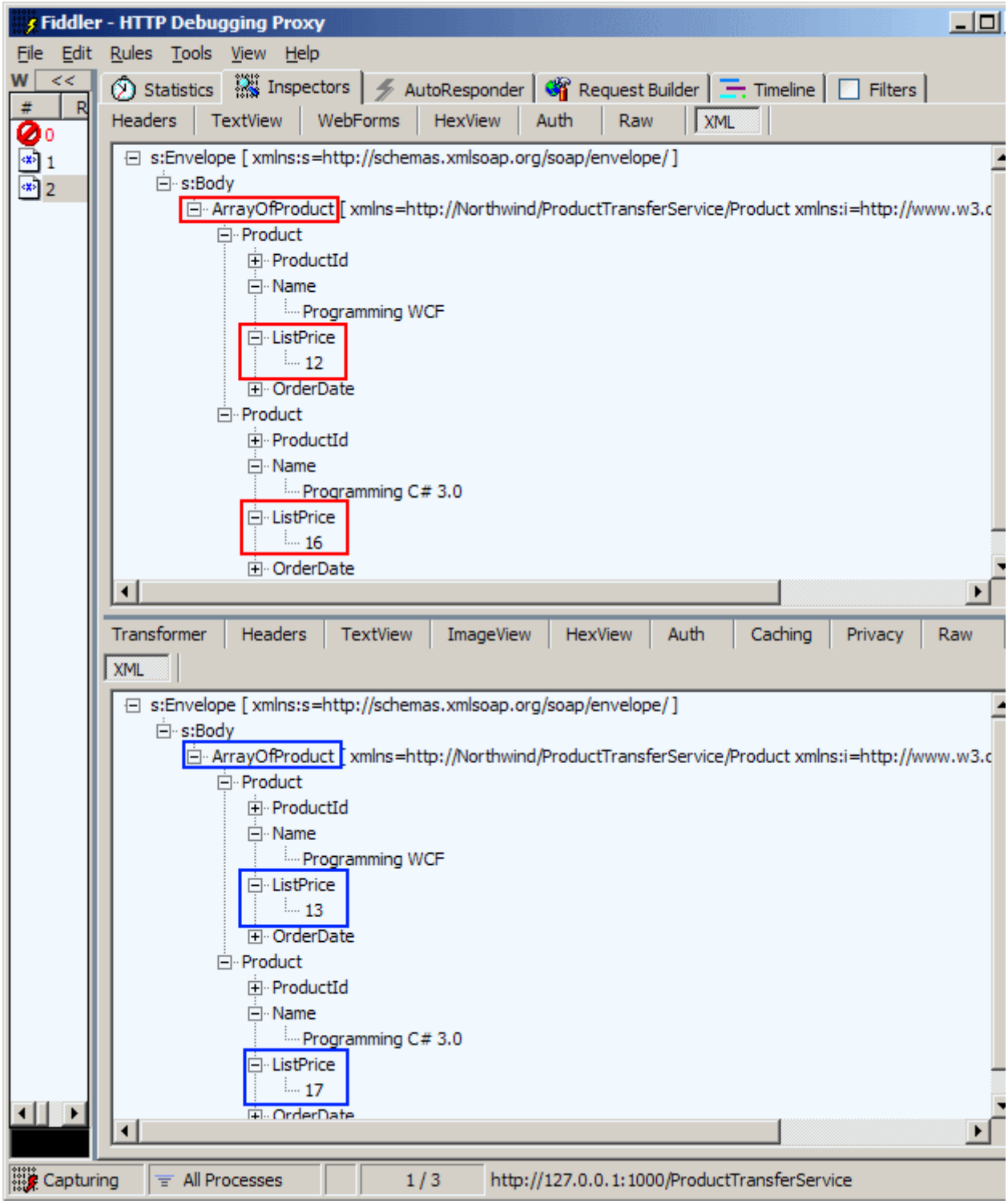
örneğidir. Bu örnekte Product tipinden bir dizi kullanılmaktadır.

```
    Message request =
Message.CreateMessage(OperationContext.Current.OutgoingMessageHeaders.Messa
geVersion, "RequestAction", products);

    // Operasyon çağrısı yapılır ve parametre olarak hazırlanan Untyped Message örneği
gönderilir.
    // çağrı sonucu yine bir Untyped Message örneğidir.
    Message reply = client.RunProcess(request);

    // Servisten gelen Untyped Message içerisindeki Body kısmında tutulan Product
topluluğunu dizi olarak ele almak için GetBody metodunun generic versiyonu
kullanılır.     Bunun sonucu olarak elde edilen sonuç Product tipinden bir dizi olacaktır.
    Product[] response = reply.GetBody<Product[]>();

    foreach (Product product in response)
    {
        Console.WriteLine(product.Name+" "+product.ListPrice);
    }
}
```

İstemci tarafında **RunProcess** metodu çağırılmadan önce gönderilecek mesajın oluşturulması için yine **CreateMessage static** metodundan yararlanılmaktadır. Yine ilk parametre olarak**SOAP** versiyonu, ikinci parametre olarak **SOAP Action** değeri ve üçüncü parametre olarakta **Body** kısmına serileştirilecek nesne örneği belirtilmiştir. Servis tarafından gelen mesaja ait Body bilgisinin okunması içinde **GetBody<T>** metodundan yararlanılmaktadır. İstemci tarafında dikkat edilmesi gereken noktalardan biriside tüm bu işlemleri içerisine alan **Using** bloğunda**OperationContextScope** nesnesinden yararlanılması ve o anki **kanal(Channel)** bilgisinin kullanılmasıdır. örnek uygulamamız bu haliyle test edildiğinde çalışma zamanı görüntüsü aşağıdakine benzer olacaktır.

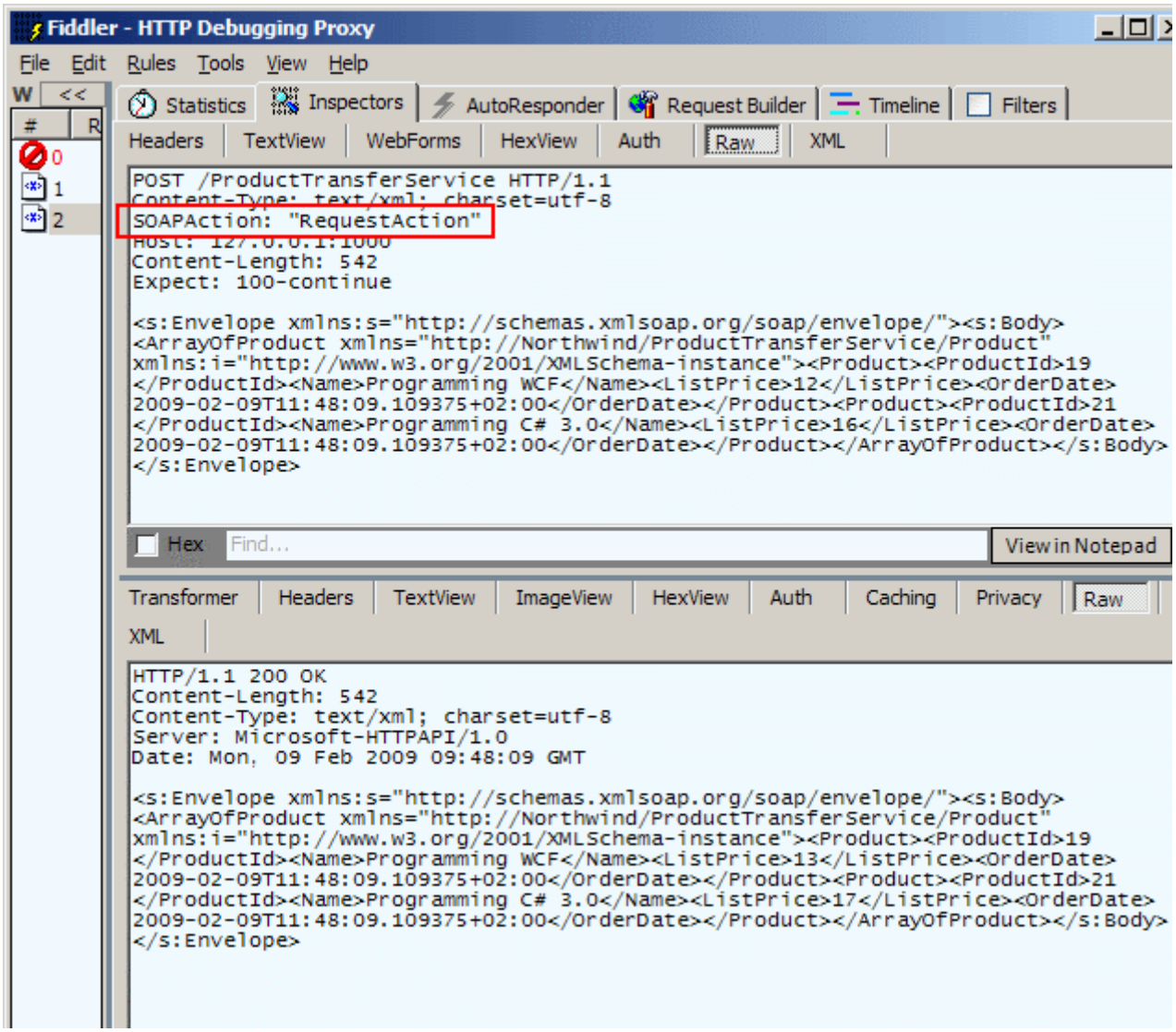


Ancak elbetteki arka planda yer alan mesaj içeriğine **Fiddler** aracı yardımıyla bakıldığında **Body** kısmında hareket eden **Product** verilerinin içeriği açık bir şekilde görülebilmektedir.

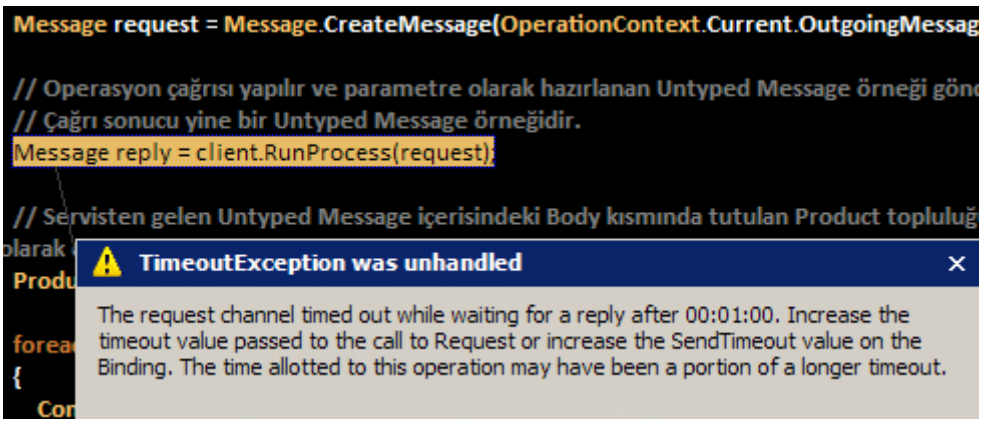


Dikkat edileceği üzere **Request** mesajında gönderilen **Product** nesnelerine ait **ListPrice** değerleri, **Response** mesajı içerisinde **1** birim arttırılmıştır. Eğer mesajların **RAW** içeriklerine bakılırsa **SOAP Action** bilgisininde set edilmiş olduğu görülebilir. *(Size tavsiyem **GetBody** metodlarına olan*

*çağrılarda **BreakPoint** kullanarak **request** ve **response** değişkenlerinin çalışma zamanı içeriklerini **QuickWatch** ile izlemenizdir.)*



***NOT :** Eğer istemciden talep gönderildikten sonra varsayılan olarak **1** dakikalık zaman dilimi içerisinde servis tarafından cevap gelmezse aşağıdaki ekran görüntüsünde yer alan**TimeoutException** istisnası ile karşılaşılır.*



*Bu sorun **SendTimeout** değeri arttırılarak çözümlenebilir. Bu sorun, uzun süren operasyonların söz konusu olduğu durumda dikkate alınması gereken istisnaların başında gelmektedir.*

Buraya kadar yaptıklarımıza baktığımızda, istemci ve sunucu arasındaki **Mesaj** içeriklerinin yönetiminin **Mesaj Sözleşmeleri** yardımıyla ele alınabildiği sonucu ortaya çıkmaktadır. Buna göre istenirse, istemci ve sunucu arasında taşınacak bir veri tipinin belirli parçalarının **SOAP** zarfı içerisindek **Header** veya **Body** bölümleri arasında ayrıştırılması mümkün olabilmektedir. Hatta, istemci ve servis arasında özel mesaj desenlerinin oluşturulması da söz konusu ve olasıdır. Elbette bu işi tamamlayıcı en önemli nokta**şifreleme(Encryption)** işlemlerininde hesaba katılmasıdır. Buda yapıldığı takdirde mesajın daha güvenilir bir şekilde ele alınması ve korunması mümkün hale gelmektedir. Diğer taraftan istemci ve servis arasındaki mesajların türlendirilmemiş olmaları halindede ele alınabildikleri ve içeriklerinin yönetilebildikleride ortadadır. **Mesaj Sözleşmeleri** ile ilişkili olarak daha detayı bilgi almak için, MSDN' de yayınlanan içeriği takip etmenizi öneririm. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

UsingMessageContracts.rar (141,52 kb)

Ado.Net Data Services Ders Notları - Security (2009-02-02T06:16:00)

ado.net data services,

Değerli Okurlarım Merhabalar,

Yazılım dünyasının en önemli zorluklarından biriside uygulamanın kapsamına göre güvenliğin etkili bir şekilde nasıl sağlanacağı ile ilişkilidir. Burada hassas bilgilerin korunması, kullanıcıların tanınması ve yetkilendirilmesi, kodun erişim ilkelerinin belirlenmesi, verinin şifrelenmesi gibi pek çok faktör söz konusudur. Genel anlamda günvelik farklı şekillerde göz önüne alınabilir.

- *Kimi zaman uygulama içerisinde kullanılan parametrik dış ortam değişkenlerini korumak gerekir. örneğin uygulamanın kullandığı değişkenlerin konfigurasyon(**app.config, web.config** gibi) dosyasında şifrelenerek saklanması önemlidir ki bu pek çok uygulama standardınında ilkeleri arasında yer almaktadır.*
- *Kimi zaman uygulamanın içerisindeki kodların ne tür işlemler yapabileceğinin belirlenmesi**(Kode Erişim Güvenliği-Code Access Security)** önemlidir. örneğin uygulamanın, kurulduğu sistem üzerinde dosya yazma yetkisi olmamasının sağlanması, yada sadece dosya okuma yapmasına izin verilmesi veya uygulama içerisinden ağ ortamına bağlantıya izin verilmemesi gibi.*
- *Kimi zaman uygulamayı açan kişilerin doğrulanması ve yapabileceklerinin sınırlandırılması gerekir**(Authentication/Authorization)**. örneğin uygulamayı açma*

yetkisi olan bir kullanıcının sahip olduğu role göre her menü seçeneğini kullanamaması gibi.

Vakalar ve gereklilikler çoğaltılabilir. Tek bir makine üzerinde kendi başına çalışan uygulamalar için güvenliğin sağlanması nispeten daha kolaydır. Ancak**istemci/sunucu(Client/Server)** bazlı mimariye geçildiğinde güvenliği sağlamak her zamankinden dahada zor bir hal almaktadır. Bunun en büyük nedenlerinden birisi farklı ortamlar arasında verinin, çeşitli protokollere göre mesajlar üzerinden transfer edilmesi gerekliliği ve bu nedenle iletişiminde güvenli hale getirilmesinin zorluğudur. öyleki, mesajların şifrelenmesi, iletişim kanalının güvenli hale getirilmesi, aradaki mesajların yakalanması ve değiştirilmesi ihtimaline karşılık gerekli tedbirlerin alınması gibi kıstaslar söz konusudur. Yine durum istemci ve sunucu tarafındaki uygulamaların belirli olmaları halinde biraz daha kolay bir şekilde ele alınabilir. Oysaki sunucu tarafında bir servis uygulamasının bulunması ve buna herhangibir istemcinin bağlanabilecek olması gibi durumlarda ulusal bir takım güvenlik ilkelerine uygun olacak şekilde iletişimi sağlamak ve mesajlaşmak gerekmektedir.

Web servisleri göz önüne alındığında bu tip güvenlik konularını kolay bir şekilde tesis etmek adına **WSE(Web Service Enhancements)** alt yapısından yararlanılmaktadır. **.Net Remoting** tabanlı dağıtık çözümlerde sorumluluk neredeyse geliştiricinin kendisine aittir. Ancak **Windows Communication Foundation** uygulamaları göz önüne alındığında güvenlik, daha kolay ve etkili bir şekilde **iletişim(Transport)** veya **mesaj(Message)** seviyesinde ele alınabilmektedir ki bu kriterler şu anda konumuz dışındadır :)

Ado.Net Data Service' lerde temel olarak birer **WCF** servisidir. Bununla birlikte söz konusu servisler bilindiği üzere istemcilere **RESTful** modele göre hizmet vermektedir. Yani **HTTP** protokolünün**GET,HEAD,DELETE,PUT** gibi metodlarını ele alıp **ATOM,XML,JSON** gibi standartları kullanmaktadır. Basit bir servis olarak göz önüne alındığında güvenlik konusunda henüz yeteri kadar gelişmiş olmadığı düşünülebilir; mi acaba? İşte bu makalemizde daha çok bu soruya cevap bulmaya çalışacağız.

Her şeyden önce en önemli nokta **Ado.Net Data Service**' lerin veri kaynaklarını istemciye **RESTful** modeline göre sunmasıdır. Bu açıdan bakıldığında geliştiricilerin daha çok üzerinde duracağı nokta verinin erişilebilirliğinin güvenli hale getirilmesidir. Dolayısıyla **Ado.Net Data Service**' leri kullanan istemcilerin, servis tarafında bir şekilde **doğrulanması(Authenticate)** ve sonrasında durumlarına bakılarak **yetkilendirilmeleri(Authorization)** güvenliğin sağlanması adına önemlidir. Oysaki **Ado.Net Data Service** örnekleri, aslında herhangibir uygulama üzerinde host edilebilecek şekilde kullanılabilir. **WCF** kadar geniş bir konsepti yoktur. Bu nedenle kural basittir; doğrulama işlemlerinin sorumluluğu aslında **Ado.Net Data Service**örneğini host eden uygulamaya aittir. Bu anlamda, servisin bir **WCF** projesinde veya bir **ASP.NET** uygulamasında barındırılması doğrulama ve yetkilendirme işlemlerinin daha kolay ele alınabilmesi açısından önemlidir. **Ado.Net Data Service**' lerinde güvenlik 4 farklı alanda değerlendirilmektedir.

Kriter	Güvenlik Alanı	
Host uygulamaya ait doğrulama modeli	Authentication (Doğrulama)	Servisi kullanacak olan istemcilerin doğrulanması için **Ho** hosting işleminde **built-in Membership API**' si kullanıla
Servis Operasyonları **(Service Operations)**	Authorization (Yetkilendirme)	Metod bazlı operasyonlar yazılarak veriye olan erişim kıs edilmesine izin verilmesi**(Single Result)** sağlanabilir.
Veri Kesmeleri **(Data Interceptors)**		İstemcinin talep ettiği verinin elde edilmesinden**(Read)** v yapılması sağlanabilir. örneğin kullanıcının yetkisine göre
Entity Görünürlüğü **(Entity Visibility)**		Servisin dış ortama sunduğu **Entity** örneklerinin işlenme işlemlerine izin verilmesi gibi.

Buradaki kriterlerin uygulanması ile bir **Ado.Net Data Service** ve içeriğine olan erişim yetkilendirilebilir. Aslında teknik detayları, geliştireceğimiz örneğin aralarına serpiştirerek devam edebiliriz. İlk olarak bize bir test servisi gerekmektedir. Konuyu kolay işlemek adına **Ado.Net Entity Framework** öğesini kullanaraktan **Northwind** veritabanındaki tüm tabloları ele aldığımızı düşünelim. Sonrasında ise basit bir **Ado.Net Data Service** öğesi geliştireceğiz. Peki ama bu öğeyi nerede barındıracağız? İşte burada **kullanıcı doğrulamasını(Authentication)** kolayca tesis edebileceğimiz bir **Asp.Net** uygulamasını göz önüne alabiliriz. Elbetteki **Asp.Net Web Site Administration Tool'** unu kullanaraktan bir kaç test kullanıcısı ve rolü oluşturmaktada yarar olacaktır. örneklerimizde kullanacağımız kullanıcı bilgileri aşağıdaki gibidir.*(örnekte **SQL Express Edition** kullanılmış ve bu nedenle **ASPNETDB.MDF**dosyası web sitesinin olduğu **App_Data** klasörü altında oluşturulmuştur.)*

Kullanıcı	Şifre	Rol
dealer1	dealer1.	Dealer
dealer2	dealer2.	
dealer3	dealer3.	
region1	region1.	Region
region2	region2.	

önemli unsurlardan biriside siteye olan ve amacımız gereği özellikle **Ado.Net Data Service** öğelerine olan erişimi kısıtlamaktır. Yani siteye **isimsiz kullanıcı(Anonymous User)** girişi kesin olarak engellenmelidir. Aşağıdaki ekran görüntüsüde, web.config

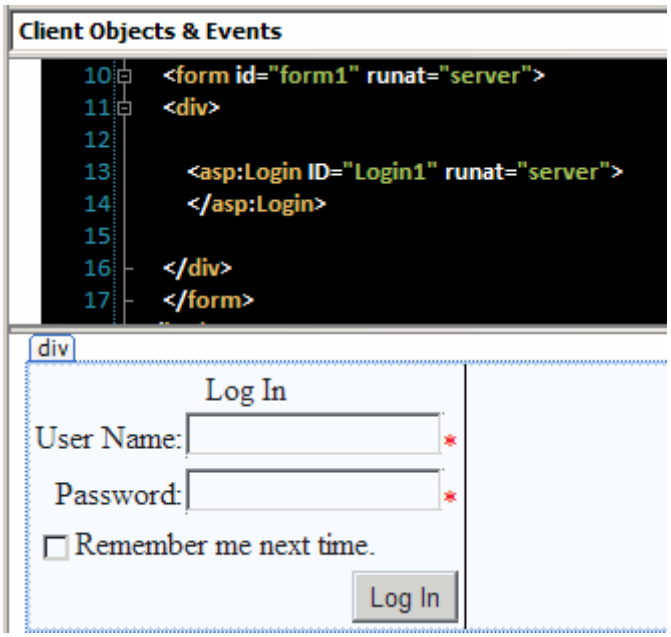
dosyasında söz konusu engelleme için yapılmış olan değişikliler açık bir şekilde görülebilir.

```
28  <system.web>
29    ...
35    <authorization>
36    <deny users="?" />
37    </authorization>
38    <roleManager enabled="true" />
39    <compilation debug="true">...</compilation>
55    ...
60    <authentication mode="Forms" />
61    ...
```

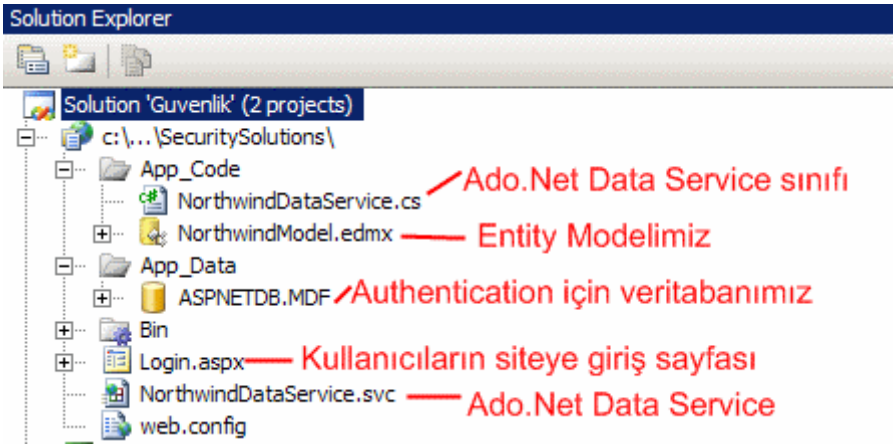
Dikkat edileceği üzere **Form** tabanlı doğrulama kullanılmaktadır ve isimsiz kullanıcıların sisteme girmeleri yasaklanmıştır.

NOT : *Elbetteki* ***Form Tabanlı Doğrulama(Form Based Authentication)*** *şart değildir. özellikle* ***Intranet*** *sistemlerde* ***Windows tabanlı doğrulama(Windows Based Authentication)****' da ele alınabilir. Hatta istenirse* ***Passport*** *Tabanlı Doğrulama da etkinleştirilebilir. Hangisi kullanılırsa kullanılsın, servisi host eden web sisteminin doğrulama yetenekleri, Ado.Net Data Service çalışma zamanı tarafından ele alınabilmektedir.*

Form tabanlı doğrulama söz konusu olduğundan varsayılan olarak aksi belirtilmedikçe*(Asp.Net Güvenlik konularını hatırlayalım)* **Login.aspx** isimli bir giriş sayfasına ihtiyacımız olacaktır. Bu sayfa üzerinde yine **Asp.Net** bileşenlerinden olan **Login** kontrolü kullanılabilir.



Sonuç olarak servis tarafındaki projenin durumu aşağıdaki şekilde olduğu gibidir.



Dikkat edileceği üzere servisi kullanacak olan kişilerin doğrulanması işlemini host uygulamanın kendisine vermiş bulunmaktayız. Buna göre kullanıcılar herhangibir şekilde servis dosyasını talep ettiklerinde, eğer bir bilete sahip değillerse, otomatik olarak **Login.aspx** sayfasına yönlendirileceklerdir. **ASPNETDB.MDF** veritabanı dosyasında yer alan ve etkin olan bir kullanıcı ile sisteme girildiğnde ise, kodlama tarafında karar vereceğimiz yetkilendirmeler devreye girecektir. Yani veriye olan erişim istenirse kullanıcıya göre kısıtlandırılabilecektir. Şimdi bu durumları analiz etmeye çalışalım. Konuyu son derece basit bir şekilde ele alacağımızdan **NorthwindDataService.cs** kod içeriğini aşağıdaki gibi yazmamız şimdilik yeterli olacaktır.

```
using System;
using System.Data.Services;
using System.Linq;
using System.Linq.Expressions;
using System.ServiceModel.Web;
using System.Web;
using NorthwindModel;
using System.Collections.Generic;

public class NorthwindDataService
   : DataService<NorthwindEntities>
{
   public static void InitializeService(IDataServiceConfiguration config)
   {
      // Rol tabanlı veri çekişi işlemleri örnek olarak gösterilebilir;
HttpContext.Current.User.IsInRole();

      config.SetEntitySetAccessRule("*", EntitySetRights.All);
      config.SetEntitySetAccessRule("Employees", EntitySetRights.ReadMultiple);
      config.SetEntitySetAccessRule("Orders", EntitySetRights.WriteAppend);
      config.SetEntitySetAccessRule("Customers", EntitySetRights.None);

      config.SetServiceOperationAccessRule("CustomerCities",
```

```
ServiceOperationRights.All);
      config.SetServiceOperationAccessRule("MySuppliers",
ServiceOperationRights.All);

      // Eğer alttaki satır açılırsa FilterForProducts metodu devre dışı sayılır ve Products
entity' sine hiç bir şekilde erişilemez.
      //config.SetEntitySetAccessRule("Products", EntitySetRights.None);

   }

   [QueryInterceptor("Products")]
   public Expression<Func<Products, bool>> FilterForProducts()
   {
      string name = HttpContext.Current.User.Identity.Name;

      if (name == "dealer1")
         return p => p.Suppliers.SupplierID == 1 || p.Suppliers.SupplierID == 2;
      else if (name == "dealer2")
         return p => p.Suppliers.SupplierID == 3;
      else if (name == "dealer3")
         return p => p.Suppliers.SupplierID == 1 || p.Suppliers.SupplierID == 2 ||
p.Suppliers.SupplierID == 7;
      else
         return p => p.Suppliers.SupplierID != null;
   }

   [ChangeInterceptor("Products")]
   public void ProductChange(Products p, UpdateOperations operation)
   {
      switch (operation)
      {
         case UpdateOperations.Add:
            break;
         case UpdateOperations.Change:
            if(!HttpContext.Current.User.IsInRole("Region"))
               throw new DataServiceException(405,"UnitPrice için bu değişikliğe izin
verilmedi");
            break;
         case UpdateOperations.Delete:
            break;
         case UpdateOperations.None:
            break;
         default:
            break;
      }
```

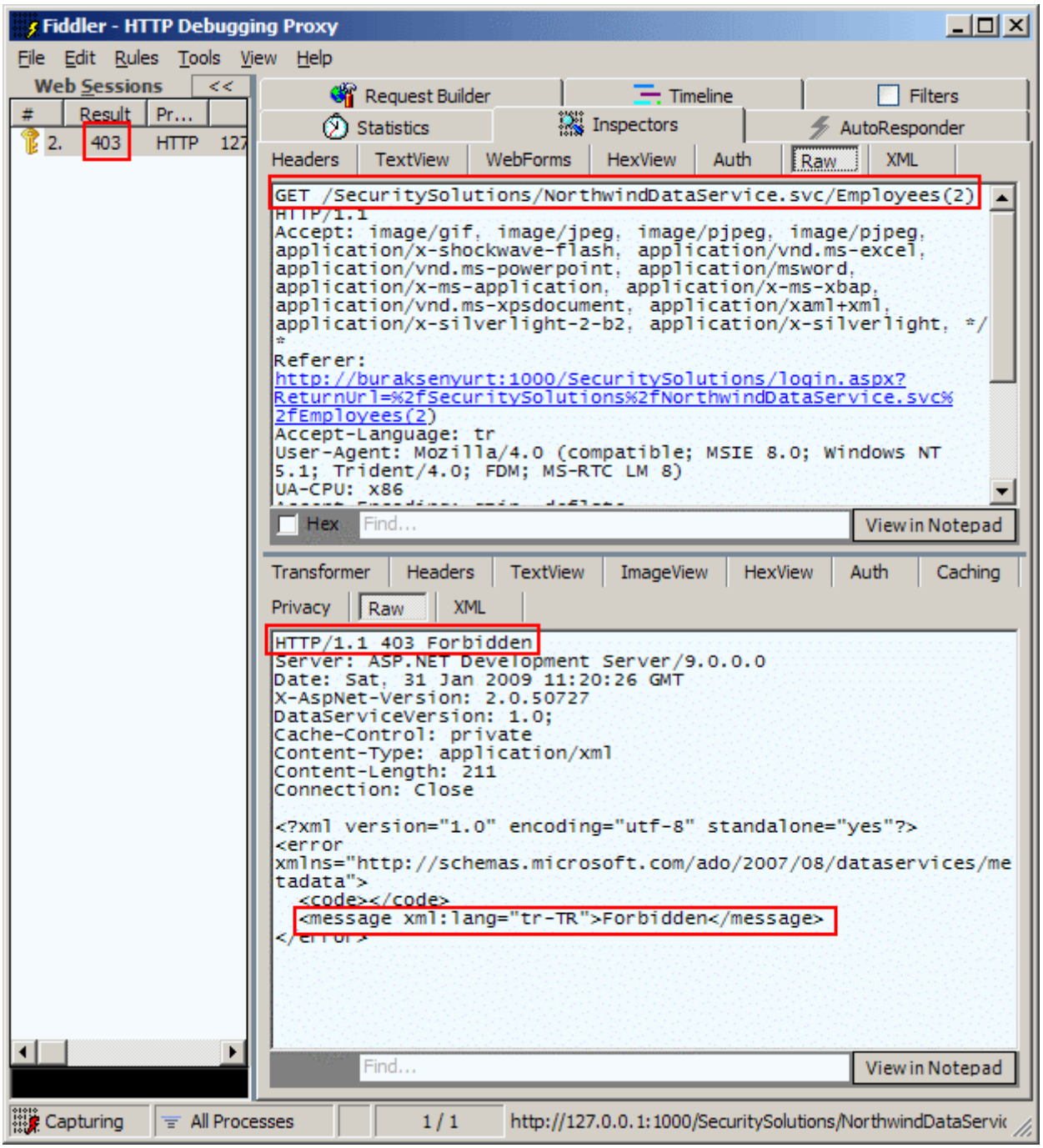
```
    }

    #region Service Operations

    [WebGet]
    public IQueryable<string> CustomerCities()
    {
        return (from c in this.CurrentDataSource.Customers
                select c.City).Distinct();
    }

    // filter, orderby gibi operatörler ve key bazlı erişim gibi sorgulara izin verilmez. Sadece
entity bazlı erişim söz konusudur
    [WebGet]
    public IEnumerable<Suppliers> MySuppliers()
    {
        return from c in this.CurrentDataSource.Suppliers
               orderby c.CompanyName
               select c;
    }

    #endregion
}
```

İlk olarak uygulamamızı test etmeye çalıştığımızda **Login.aspx** sayfasına yönlendirildiğimizi göreceğiz. Bir başka deyişle herhangibir isimiz kullanıcı, servise doğrudan erişilmek istendiğinde **http://localhost:1000/SecuritySolutions/login.aspx?ReturnUrl=%2fSecuritySolutions%2fNorthwindDataService.svc** adresine gönderilecektir ve dolayısıyla doğrulama sürecine girilmiş olacaktır. Eğer geçerli bir kullanıcı bilgisi ile giriş yapabilirsek bu durumda standart olarak servise erişebildiğimizi ve izin verilen sorgulamaları yapabildiğimizi göreceğiz. Şimdi kodu biraz analiz etmeye çalışalım. **InitializeService** metodu içerisine bakıldığında **Entity** seviyesinde bazı yetkilendirmeler yapıldığı görülmektedir.

```
config.SetEntitySetAccessRule("*", EntitySetRights.All);
config.SetEntitySetAccessRule("Employees", EntitySetRights.ReadMultiple);
config.SetEntitySetAccessRule("Orders", EntitySetRights.WriteAppend);
config.SetEntitySetAccessRule("Customers", EntitySetRights.None);

config.SetServiceOperationAccessRule("CustomerCities", ServiceOperationRights.All);
config.SetServiceOperationAccessRule("MySuppliers", ServiceOperationRights.All);
```

Buna göre tüm Entity kümelerine erişim hakkı izni, ilk satırdaki kod ile verilmiştir. Bu yetkilendirme, ilk satırdaki ***** ve **EntitySetRights.All** ile sağlanmaktadır. Ne varki ikinci

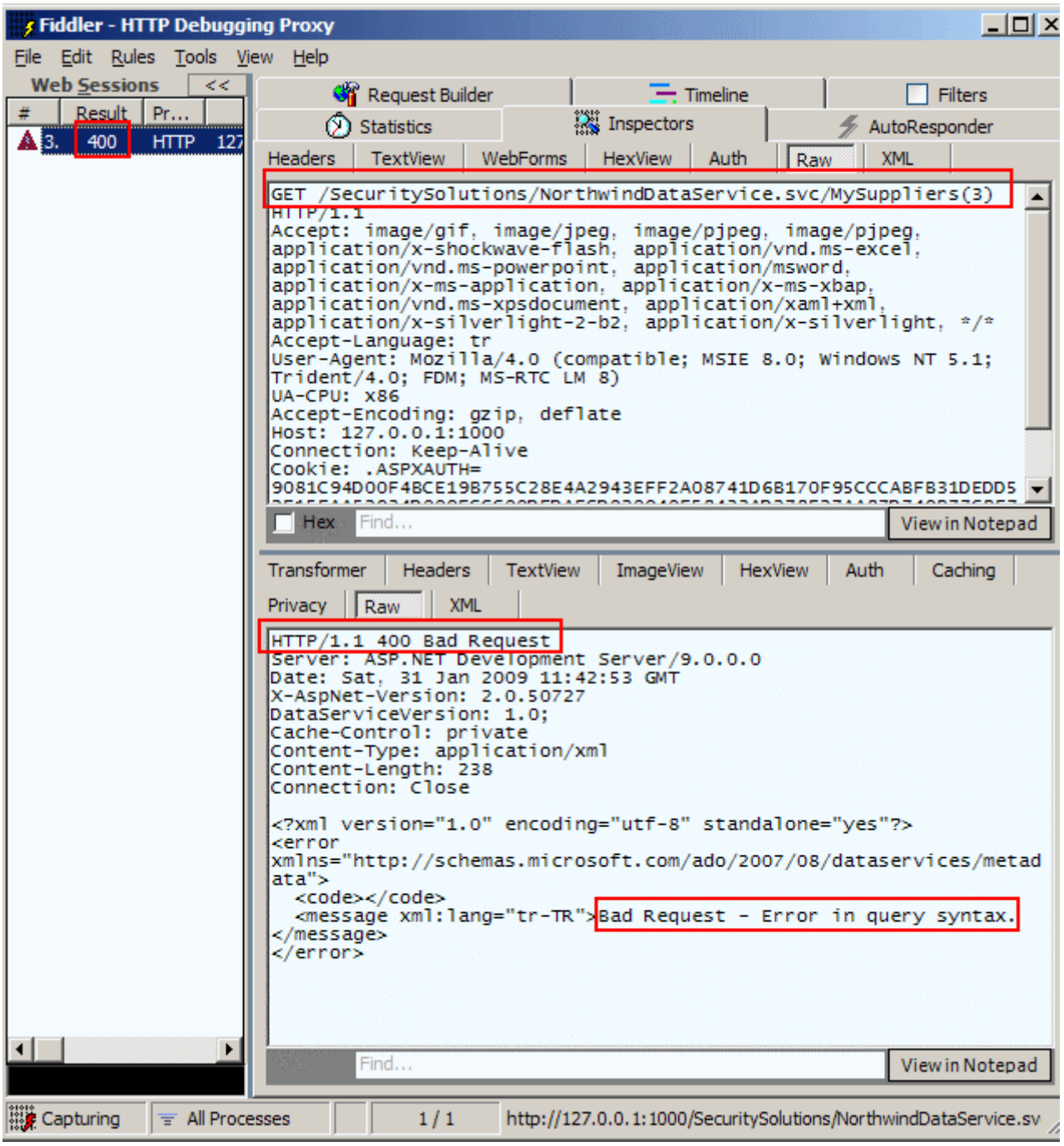
satırda **Employees**Entity içeriği için, **ReadMultiple** kısıtlaması yapılmıştır. Buna göre **Employees** kümesi üzerinde örneğin anahtar bazlı sorgulara izin verilmeyecektir. Yani **Employees(2)** gibi bir talepte bulunulursa **HTTP 403 Forbidden** hatası alınır. Gerçektende durum **Fiddler** aracı yardımıyla izlendiğinde aşağıdaki ekran görüntüsünde yer alan sonuçlar ile karşılaşılır.



Dikkat edileceği üzere istemci tarafına döndürülen **XML** içeriğinde **Forbidden** mesajı yazılmaktadır. Aynı zamanda hata kodu **403**' tür. Bu durum istemci uygulama tarafından değerlendirilmelidir. Şu anda ilk yetkilendirmemizi Entity seviyesinde yapmış bulunuyoruz. Görüldüğü üzere, tüm Entity' lere erişim hakkı verilmiş olmasına rağmen **Employees**üzerinde bir kısıtlama uygulanmıştır. üçüncü satırdaki kısıtlamaya

göre **Orders** kümesi için sadece yeni öğe eklenmesine izin verilmektedir. Bu nedenle **Orders entity** içeriği yine tarayıcı üzerinden sorgulanmak istendiğinde **HTTP 403 Forbidden** hatası alınacaktır. Ancak istemci bir uygulama tarafından, Orders isimli veri kümesine yeni öğelerin eklenmesi işlemine izin verilecektir. Son olarak **Customers Entity**' sine herhangibir şekilde erişim izni kesinlikle verilmemektedir.
Nitekim **EntitySeyRights** enum sabiti için **None** değeri verilmiştir.

Lakin burada **CustomerCities** isimli bir operasyon için izin verildiği gözlemlenmektedir. **CustomerCities** isimli operasyon **string** tabanlı **IQueryable** tipinden bir referans döndürmektedir. İçerideki sorgu cümlesine bakıldığında **Customers Entity**' si içerisindeki **City** adlarının **Distinct** fonksiyonu ile benzersiz olacak şekilde döndürüldüğü görülmektedir. Dolayısıyla **Customers**veri kümesini dış ortama tamamen kapatıp, kendisine ait şehir adlarını istemciye sunan bir operasyon tanımlaması söz konusudur ki buda bir güvenlik tedbiri olarak düşünülebilir. Yine devam eden satırda **MySuppliers** isimli bir servis operasyonuna tüm haklar ile erişim izni verilmiştir. Bu servis operasyonuna bakıldığında ise **IEnumerable<Suppliers>** tipinden bir referans döndürdüğü görülmektedir. Operasyon içerisindeki **LINQ** sorgusunda özel bir ifade yoktur. Ancak metodun dönüş tipinin **IEnumerable** olmasının bir anlamı vardır. Buna göre**Suppliers** tablosu sorgulanırken **filter**, **orderby** gibi operatörler kullanılamaz. Ayrıca anahtar bazlı erişimlere de (örneğin **Suppliers(2)** gibi) izin verilmez. Sadece sonuç kümesinin ham hali istemciye sunulur. Buda sonuç itibariyle bir kısıtlamadır. Nitekim çalışma zamanında örneğin, **SupplierId** değeri **3** olan **Supplier** bilgisini almak istediğimizde **HTTP 400 Bad Request** hatasını aldığımız görebiliriz.



Bu **IQueryable** ve **IEnumerable** arasındaki farkı biraz daha net bir şekilde görmüş bulunuyoruz. Yanlız dikkat edilmesi gereken bir nokta vardır. Servis operasyonunun döndürdüğü bir**Entity** içeriği var iken**(örneğin IEnumerable<Suppliers>) InitializeService** metodu içerisinde söz konusu tip için **EnititySetRights.None** değerinin kullanılması sonrasında servis çalışmayacaktır. örneğimizde **Customers** için yapılan kısıtlamaya dikkat edildiğinde **MyCustomers** isimli operasyonun geriye **string** bazlı bir sonuç kümesi döndürdüğü görülmektedir. Bu servisin çalışmasına engel olmamaktadır.

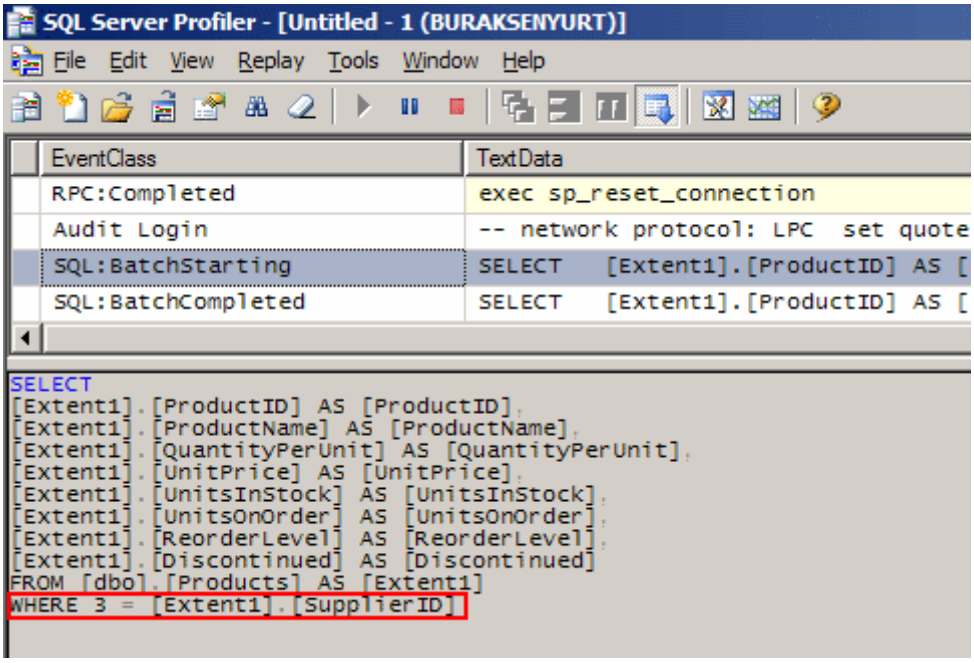
özellikle **Entity** tipleri ve servis operasyonları için yapılan erişim kısıtlamalarında devreye giren **EntitySetRights** enum sabitinin alabileceği değerler aşağıdaki tabloda belirtildiği gibidir.

	EntitySetRights D
None	**Entity** kümesine erişilmesi yasaklanmıştır. **Metadata** içerisinde görünn
ReadSingle	**Entity** üzerinde **anahtar bazlı(Key Based)** aramalara izin verilir. (**Cus**
ReadMultiple	**Entity** içeriğinin sorgulanmasına izin verilir, ancak anahtar bazlı erişim
WriteAppend	Yeni **Entity** örnekleri eklenebilir.
WriteMerge	Var olan **Entity** içeriği güncellenirken birleştirilme işlemi uygulanır.
WriteReplace	Var olan **Entity** içeriği yenisi ile değiştirilerek güncellenir.
WriteDelete	**Silme** işlemine izin verilir.
AllRead	**ReadSingle** ve **ReadMultiple** değerlerinin birleşimidir.
AllWrite	**WriteInsert**, **WriteUpdate** ve **WriteDelete** değerlerinin birleşimidir.
All	**Tam okuma** ve **yazma** erişimine izin verilir.

Servis operasyonlarında erişim kısıtlamalarını belirleyen **ServiceOperationRights** enum sabitinin alabileceği değerler ise aşağıdaki tabloda görüldüğü gibidir.

	ServiceOperationRigh
None	Servis operasyonuna erişim izni yoktur.
ReadSingle	Tek bir veri öğesinin okunmasına izin verilir.
ReadMultiple	Servis operasyonu kullanılarak birden fazla veri öğesinin okunmasına iz
AllRead	Tekil veya çoğul veri öğelerinin okunmasına izin verilir.
All	Servis operasyonu için tüm haklar sağlanır.

Kodumuzda ilerlediğimizde, **FilterForProducts** ve **ProductChange** isimli iki metod ile karşılaşmaktayız. Bu metodlar **veri kesme(Data Interceptor)** fonksiyonellikleridir. Dikkat edileceği üzere**FilterByProducts** metodu içerisinde o anki kullanıcının adına bakılaraktan örnek bir içeriğin döndürülmesi sağlanmaktadır. Söz gelimi **dealer2** isimli kullanıcı ile sisteme girildiğinde ve **Products** içeriği talep edildiğinde koddaki kesme metodunun içeriğine göre **SupplierID** değeri **3** olan listenin elde edildiği görülür. özellikle durumu **SQL Server Profiler** aracı yardımıyla incelendiğinde gerçektende kesme metodunun devreye girdiği aşikardır.



Burada belkide en önemli nokta kesme işlemi sırasında, kullanıcı bilgisinin **HttpContext.Current.User.Identity.Name** ifadesi ile alınmasıdır. Bu tahmin edileceği üzere servisi kullanmak için**Login** olan kullanıcının adıdır.

örneğimizde senaryoyu çok basit bir şekilde ele almak istediğimizden kullanıcı adlarının **dealer1, dealer2, dealer3** olması halleri ele alınmıştır. Oysaki gerçek hayat senaryolarında daha faydalı bir kesme işlemi yapılabilir. Bununla ilişkili olaraktan sizlere bir alıştırma senaryosu örneği vermek isterim. Söz gelimi, kullanıcının hangi ürünlere bakacağı bilgisi, kullanıcının dahil olduğu bölgeye bağlı olabilir. Bu durumda **ASPNETDB.MDF** veritabanında yer alan kullanıcı bilgisi ile, kullanıcıların dahil olduğu bölgeleri tutan başka bir eşleştirme tablosu bu senaryo için çok yararlı olabilir. Böylece kesme metodu içerisinde, eşleştirme tablosundan yararlanılarak, **giren kullanıcının sadece dahil olduğu bölgeye ait ürünleri görmesi** sağlanabilir. Bunu kendi başınıza denemenizi ve yapmaya çalışmanızı öneririm.

NOT: *örneğimizdeki veri kesme metodları****(Data Interceptors)*** *içerisinde servisin host edildiği uygulama ortamının içeriğinin kullanıldığı görülmektedir. Burada* ***Web*** *ortamında olunmasının büyük bir avantaj sağladığı çok açıktır. Nitekim, o anki* ***HTTP*** *içeriğine* ***HttpContext*** *özelliği üzerinden ulaşılabilmektedir. Bu sebepten sisteme giriş yapmış olan kullanıcıyı tespit etmek son derece kolaydır. Ayrıca* ***Ado.Net Data Service****' lerin host edildiği web ortamlarında,* ***Application****,* ***Session****,* ***Caching*** *gibi yapılarında ele alınması mümkün hale gelmektedir.*

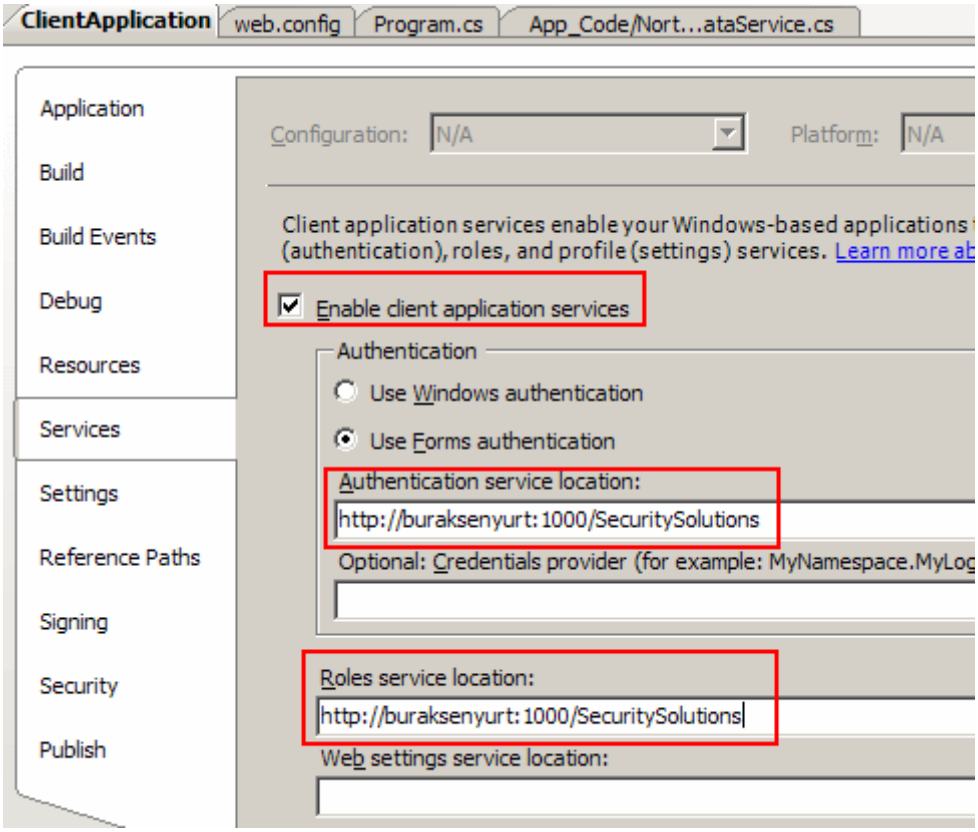
Gelelim **ProductChange** metoduna. Bu metod içerisinde giriş yapan kullanıcın **Region** rolünde olması halinde değişiklik yapabilmesine müsade edilmektedir. Eğer giren kullanıcı **Region**rolünde değilse istemci tarafına **HTTP Statu Code 405** mesajı gönderilmektedir. Aslında kesme operasyonlarını daha net bir şekilde ele alabilmek için bir istemci uygulama yazılmasında yarar vardır. Veri değiştirme işlemleri sırasında devreye

giren bu metodda önemli olan noktalardan biriside **UpdateOperations enum** sabitinin kullanılmasıdır. Bu sabitin değerine göre kullanıcının nasıl bir operasyon gerçekleştirmek istediği kolayca tespit edilebilir. Metodun ilk parametresi kesme operasyonunun kime uygulanacağını işaret etmektedir. Buna göre söz konusu kesme operasyonları **Products** tipine uygulanabilir. Diğer önemli bir noktada istemci tarafına **HTTP 405** mesajının **DataServiceException** tipinden bir nesne örneği fırlatılarak gönderiliyor olmasıdır. çok doğal olarak bu istisna tipinin istemci uygulama tarafından ele alınıyor olması gerekmektedir. *(Yani istemci tarafında try...catch...finally blokları kullanılarak istisna yönetimi yapılmalıdır.)*

Söz konusu sistemde istemci uygulamanın, servisten talepte bulunurken belirli bir kullanıcı bilgisini göndermesi de şarttır. Aslında bu noktada **Client Application Services**' lerden faydalanılabilinir.*(Bu konu ile ilişkili olarak daha önceki makalemi takip etmenizi öneririm)* özellikle **.Net** tabanlı istemcilerde İstemci Uygulama Servisinin kullanılmasını öneririm. Tabi bunun için servis tarafındaki konfigurasyon dosyasında bir takım değişikliklerin yapılması gerekmektedir. Bu sebeple host uygulamanın **web.config** dosyasında aşağıda yer alan eklemeleri yapabiliriz.

```
...
<appSettings/>

<system.web.extensions>
   <scripting>
      <webServices>
         <authenticationService enabled="true"/>
         <roleService enabled="true"/>
      </webServices>
   </scripting>
</system.web.extensions>

<connectionStrings>
...
```

İstemci uygulama tarafında ise **proje özelliklerinden(Properties)**, **Services** kısmına geçmemiz ve geliştirdiğimiz web uygulamasının adresini işaret etmemiz yeterlidir.

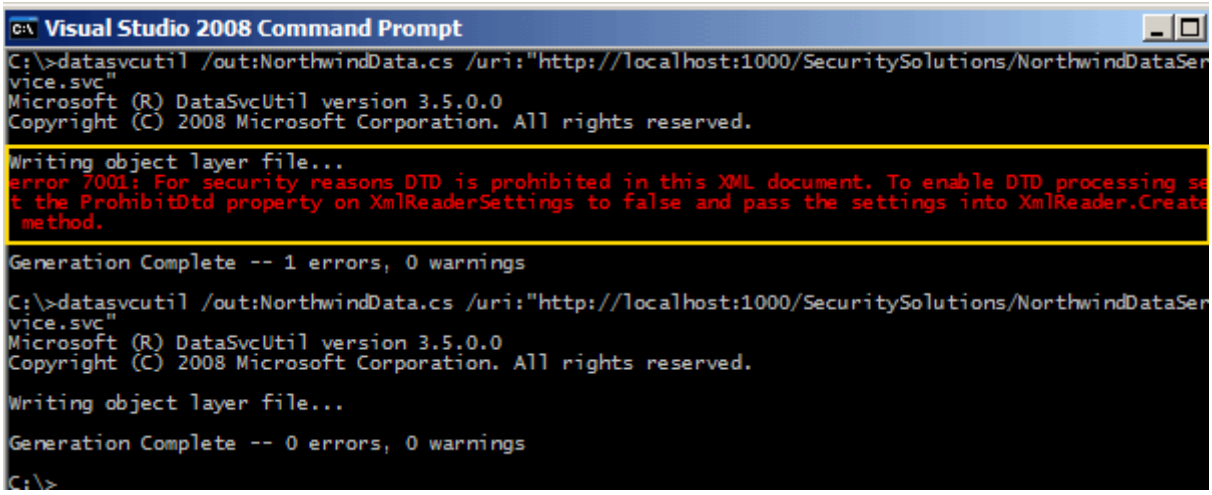


Bölyece istemci uygulamanın **doğrulama(authentication)** ve **rol(role)** yönetimi için geliştirilen web uygulamasının üyelik sistemini(**Membership API)** kullanılacağı belirtilmiş olur. örnek içerisinde **Membership** sınıfını kullanarak doğrulama yapacağımızdan **System.Web.dll assembly**' ının servis referansı ile birlikte istemci uygulamaya ekleniyor olmasıda gerekmektedir.

*Servis tarafında **isimsiz kullanıcıların(Anonymous Users)** sisteme girişini kapattığımız için **Add Service Reference** kısmından servise ait **WSDL** içeriğini elde edemediğimizi görürüz. Nitekim **Visual Studio** ortamında söz konusu servis talep edildiğinde otomatikman web uygulamasının **authentication** kuralı devreye girmekte ve bizi **Login.aspx** sayfasına yönlendirmeye çalışmaktadır. Hal böyle oluncada servise ulaşılamamakta ve referansı elde edilememektedir.*

*örnekte servisin host edildiği web uygulamasındaki **deny user="?"** kısmı, istemci uygulamayla aynı solution içerisindeki servis referansı eklendikten sonra etkin hale getirilmiştir. Bu elbetteki istenen çözüm değildir ve ayrıca daha sonrada servisin güncelleştirilmesi sırasında problemlere neden olmaktadır.*

*Diğer taraftan **datasvcutil** aracı yardımıylada istemci tarafı için gerekli tipler üretilmek istendiğinde benzer sonuçlar ile karşılaşılacaktır. Aşağıdaki ekran görüntüsünde ilk denemede **anonymous** kullanıcıların geri çevrildiği senaryo sonrası alınan uyarı mesajı görülmektedir. İkinci deneme yapılmadan önce ise **web.config**dosyasındaki **deny users="?"** kısmı **allow users="?"** olarak değiştirilmiş ve gerekli tiplerin üretildiği görülmüştür. Elbetteki buda istenen bir aktarım şekli değildir.*



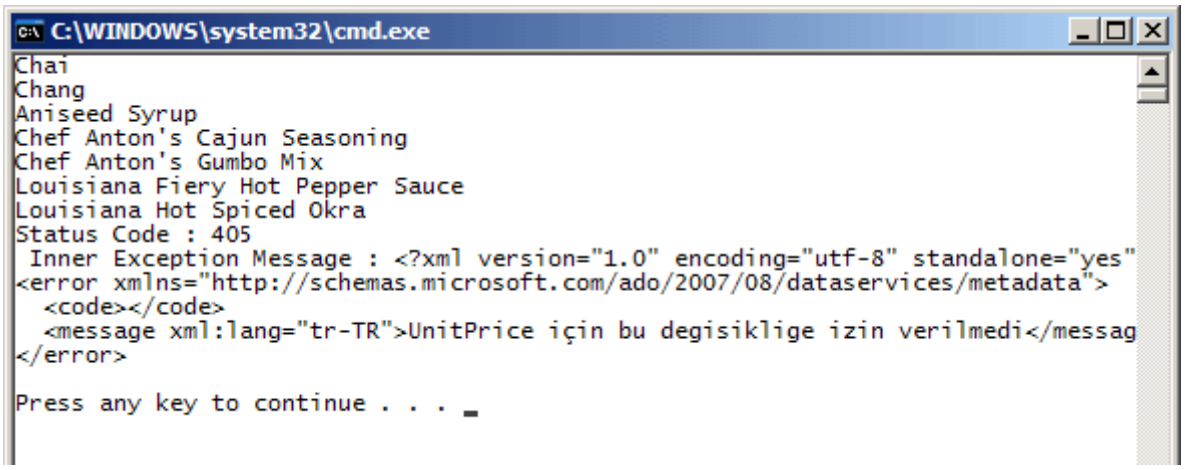
Bu noktada belkide servisin kullanılabilmesi için, istemci tarafınca gerek duyulan proxy tiplerinin önceden üretilip, kullanacak olan uygulamalara dağıtılması yöntemi tercih edilebilir. Açıkçası bu, servisi kullanacak olan istemcilerin belli olduğu durumlarda düşünülebilecek bir senaryodur ki pek çok büyük çaplı şirket içi projede göz önüne alınabilir.

Bu arada istemci tarafındaki Console uygulamasının içeriğini aşağıdaki gibi örnekleyebiliriz.

```
using System;
using System.Data.Services.Client;
using System.Linq;
using System.Net;
using ClientApplication.NorthwindServiceReference;
using System.Web.ClientServices;
using System.Threading;
using System.Web.Security;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         NorthwindEntities entities = new NorthwindEntities(
            new
Uri("http://buraksenyurt:1000/SecuritySolutions/NorthwindDataService.svc")
            );

         entities.SendingRequest+=delegate(object sender,SendingRequestEventArgs e)
         {
            ClientFormsIdentity identity = Thread.CurrentPrincipal.Identity as
```

```
ClientFormsIdentity;
            HttpWebRequest webRequest = e.Request as HttpWebRequest;
            if (identity != null)
               webRequest.CookieContainer = identity.AuthenticationCookies;
         };

         try
         {
            if (Membership.ValidateUser("dealer1", "dealer1."))
            {
               // QueryInterceptor için istemci kodu
               var tumUrunler = from urun in entities.Products
                                select urun;

               foreach (var urun in tumUrunler)
               {
                  Console.WriteLine(urun.ProductName);
               }

               // Http durum kodları(Http Status Code) için link-
> http://en.wikipedia.org/wiki/List_of_HTTP_status_codes

               // ChangeInterceptor için istemci kodu
               var u = (from urun in entities.Products
                  where urun.ProductID == 1
                     select urun).First<Products>();
               u.UnitPrice +=1;

               entities.UpdateObject(u);
               entities.SaveChanges();
            }
         }
         catch (DataServiceRequestException excp)
         {
            string excpMessage = String.Format("Status Code : {0}\n Inner Exception
Message : {1} ",
                  ((DataServiceClientException)excp.InnerException).StatusCode.ToStrin
g(), excp.InnerException.Message
               );
            Console.WriteLine(excpMessage);
         }
      }
   }
}
```

Burada belkide en can alıcı nokta doğrulama için istemci tarafından kullanıcı ve şifre bilgilerinin nasıl gönderildiğidir. Dikkat edilecek olursa servise talep gönderilmeden önce ilgili doğrulama bilgileri yollanmaktadır. **Membership** sınıfının **ValidateUser** metodu yardımıyla kullanıcı doğrulandıktan sonra ise **entity** talebinde bulunulmakta ve bir güncelleştirme işlemi gerçekleştirilmektedir. Uygulamayı çalıştırdığımızda aşağıdaki ekran görüntüsü ile karşılaşırız.



ürün bilgileri alınırken servis tarafındaki **FilterForProducts** isimli veri kesme metodu devreye girmiş ve **delaer1** için **SupplierID** değerleri **1** veya **2** olanlar getirilmiştir. Yine dikkat edilecek olursa**SaveChanges** metodundan sonra servis tarafındaki **ProductChange** kesme metodunun devreye girmesi sonucu istemciye hata mesajı döndürülmüş ve bir **istisna(exception)** oluşmuştur. Bu son derece doğaldır nitekim **dealer1** kullanıcısı **Region** rolünde değildir. Ancak örneğin **region1** kullanıcısı ile giriş yaparsak bu durumda **SupplierID** değeri **null** olmayan ürünleri çekebildiğimizi ve aynı zamanda bunlardan ilkinin **UnitPrice** değerlerinide değiştirebildiğimizi görürüz. Hatta **SQL Server Profiler** aracı ile arka plandaki sorgu durumunu izlersek aşağıdakine benzer bir sorgunun işletildiğini kolayca izleyebiliriz.

```
exec sp_executesql N'update [dbo].[Products]
set [ProductName] = @0, [QuantityPerUnit] = @1, [UnitPrice] = @2, [UnitsInStock] =
@3, [UnitsOnOrder] = @4, [ReorderLevel] = @5, [Discontinued] = @6
where ([ProductID] = @7)
',N'@0 nvarchar(4),@1 nvarchar(18),@2 decimal(19,4),@3 smallint,@4 smallint,@5
smallint,@6 bit,@7 int',@0=N'Chai',@1=N'10 boxes x 20
bags',@2=22.0000,@3=39,@4=0,@5=10,@6=0,@7=1
```

örneklerdende görüldüğü üzere **Ado.Net Data Service**' lerde güvenliği sağlarken **doğrulama(Authentication)** ve **yetkilendirme(Authorization)** adına yapılabilecek belirli işlemler söz konusudur. Bu işlemler için bazı kuralların uygulanması gerekmektedir. Söz gelimi servis operasyonları göz önüne alındığında, yazılacak olan metodlarda dikkat edilmesi gereken kurallar şunlardır.

- *Metodun **public** erişim belirleyicisine sahip olması gerekmektedir.*

- *Metodun dönüş tipi **IQueryable<T>** veya **IEnumerable<T>** olabilir. Buradaki **T**, **Entity** tipidir. Eğer operasyonun döndürdüğü sonuç kümesi üzerinde **sıralama**, **sayfalama**,**filtreleme** gibi işlemler yapılacaksa **IQueryable<T>** tipinin döndürülmesi gerekir.*
- ***HTTP Get** metoduna uygun çağrılar için **WebGet, HTTP Post,Delete,Put** gibi talepler içinse **WebInvoke** niteliği**(Attribute)** kullanılmalıdır.*

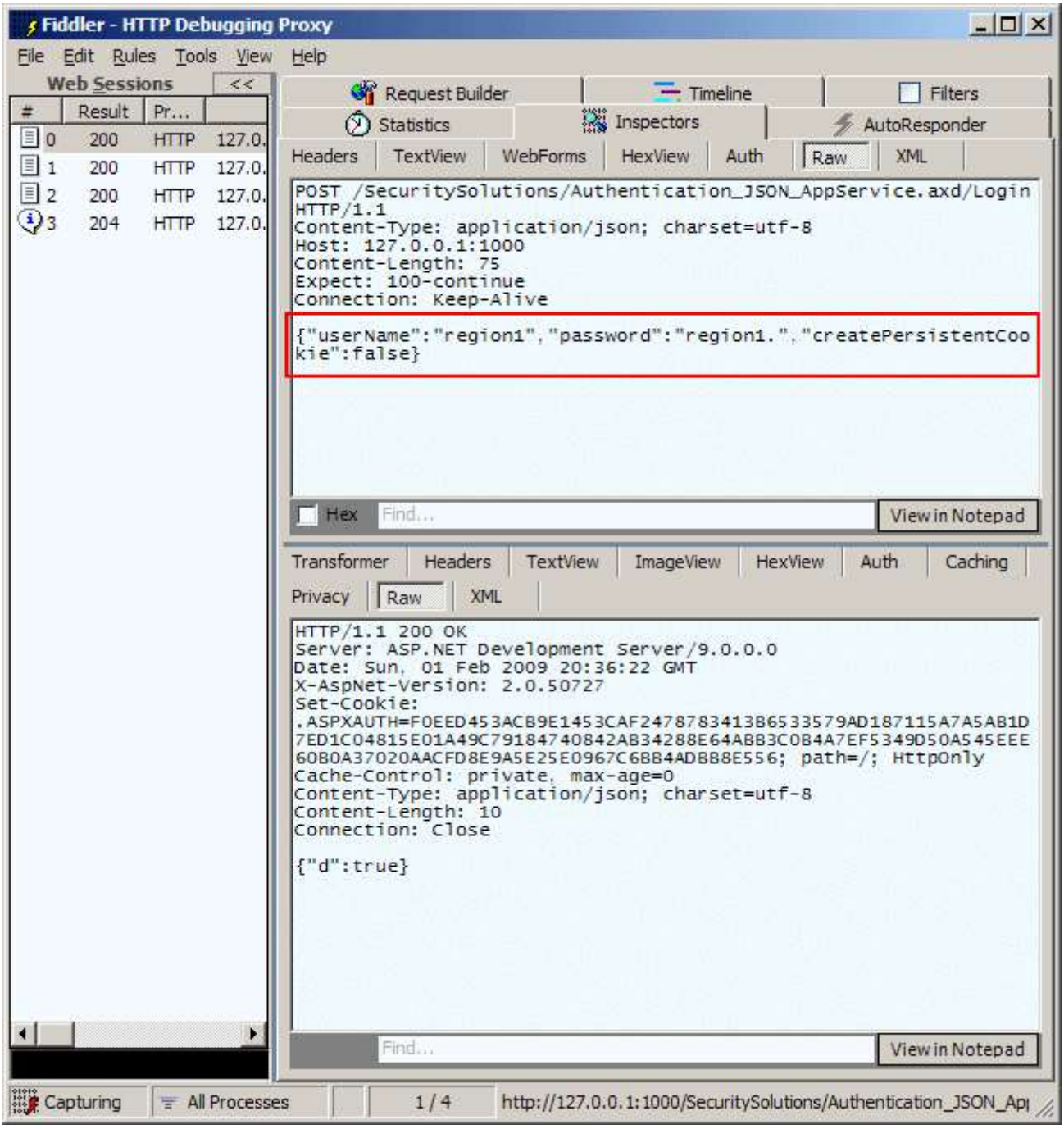
Benzer şekilde okuma işlemleri sırasındaki kesme fonksiyonelliklerininde uygulaması gereken bazı kurallar vardır. Buna göre;

- *Metodun **public** erişim belirleyicisine sahip olması gerekir.*
- *Metoda **[QueryInterceptor("EntityName")]** niteliğinin uygulanması gerekir. **EntityName** yerine kesme işleminin uygulanacağı **Entity** tipinin adı verilir.*
- *Metodun dönüş tipi **Expression<Func<T,bool>>** olmalıdır. **Func** temsilcisinde yer alan **T entity** tipidir.*
- *Metod parametre almaz.*
- *Metodun işleyişi sırasında bir **istisna(Excpetion)** oluşsa bile istemci talebi tamamlanır ve kendisine hata mesajı uygun **HTTP Statu Code** değeri ile döndürülür.*

Eğer kesme operasyonu veri güncellenmesi,eklenmesi veya silinmesi işlemleri sırasında yapılacaksa, izlenilmesi gereken kurallar aşağıdaki gibidir.

- *Metodun **public** erişim belirleyicisi olmalıdır.*
- *Metoda **[ChangeInterceptor("EntityName")]** niteliği uygulanmalıdır. Buradaki **EntityName**, **entity** tipinin adıdır.*
- *Metodun dönüş tipi yoktur. Bu nedenle **void** olarak tanımlanır.*
- *Metodun iki parametresi vardır. İlki **entity tipi** ikincisi ise **UpdateOperations enum** sabitidir. Bu enum sabiti ile metodu içerisinde **değiştirme, silme, ekleme** operasyonları ele alınabilir.*
- *Metodun işleyişi sırasında bir **istisna** oluşursa, istemciden gelen talep tamamlanır ve kendisine uygun olan **Http Statu Code** değerine sahip hata gönderilir. Bu istisna sonrasında sunucu tarafındaki asıl veri kaynağında herhangibir değişiklik kesinlikle olmaz.*

Makalemizi sonlandırmadan önce önemli bir noktayı daha vurgulamakta yarar vardır. Servisin doğrulanması için istemci tarafından gönderilen kullanıcı adı ve şifre bilgileri açık metinler olarak gitmektedir. Eğer örnekler test edilirken **Fiddler** aracı yardımıyla arka plandaki paketler izlenirse aşağıdaki ekran görüntüsünde yer alan durum ile karışalışır.



Bu nedenle en uygun çözüm servisin **HTTPS** tabanlı bir iletişim üzerinden hizmet vermesinin sağlanması olarak düşünülebilir.

Bu yazımızda **Ado.Net Data Service**' lerde **doğrulama** ve **yetkilendirme** işlemlerinin nasıl ele alınabileceğini, bir başka deyişle güvenliğin nasıl sağlanabileceğini en temel hatlarıyla incelmeye çalıştık. **Ado.Net Data Service** konusunda geliştirmeler devam etmektedir. Güvenlik ile ilişkili olaraktan farklı yaklaşımların getirilmeside bu nedenle söz konusu olabilir. Ancak en azından, host uygulamanın bu işte önemli bir rol üstlendiği gözden kaçırılmamalıdır. Bu yazıda kullanılan tekniğe göre, doğrulama işlemini **Asp.Net Web** uygulaması devralmıştır. Size tavsiyem bunu bir **WCF** sitesinden host ederkende gerçekleştiriyor olmanızdır. Konunun daha kolay pekişmesi adına aşağıdaki görsel

dersleride izlemenizi tavsiye ederim. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

Ado.Net Data Services-Security(1)

Ado.Net Data Services-Security(2)

Ado.Net Data Services-Security(3)

(Boyutun küçük olması için ASPNETDB.MDF ve log dosyası çıkartılmıştır. Bu nedenle örneği deneyebilmek için söz konusu veritabanını Asp.Net Web Site Administration Tool ile oluşturmanız gerekmektedir.)

Guvenlik.rar (312,54 kb)

Dayanıklı WCF Servisleri (Custom Persistence Providers) (2009-01-23T22:31:00)

wcf,

Hatırlayacağınız üzere bir önceki makalemizde, **dayanıklı WCF servislerinin(Durable WCF Services)** nasıl geliştirilebileceğini incelemeye başlamıştık. Kısaca hatırlatmak gerekirse dayanıklı WCF servislerini şu şekilde tanımlayabiliriz; belirli bir süre için durumlarını bir depolama alanında saklayarak koruyabilen ve t süre sonrasında istemci uygulama tarafından bırakıldığı haliyle kullanılabilen servisler. Konunun ilk aşamasında, varsayılan olarak SQL sunucusu üzerindeki bir depolama alanının kullanıldığı senaryo üzerinde durmuştuk. *(Hatırlayaım, SQL sunucusu üzerinde veriyi saklamak yönetsel açıdan pek çok işi kolaylaştırmaktadır.)* Bu makalemizde ise kaldığımız yerden devam edeceğiz ve aşağıdaki iki soruya yönelik çözümler geliştirmeye gayret edeceğiz.

1- İstemci uygulama servis örneğini kullandıktan sonra kapatılır*(İstemcinin kapatma işlemi sırasında servis örneğinin saklama alanından silinmesini sağlayacak metodu çağırmadığı varsayılır)*. Bu durumda t süre sonrasında istemci uygulama aynı servis örneğinin içeriğini kullanmak isterse nasıl bir süreç izlenmeli ve kodlama yapılmalıdır?

2- Varsayılan olarak kullanılan **SqlPersistenceProviderFactory** tipinin sağladığı **SQL** tabanlı saklama stratejisi yerine özelleştirilmiş başka bir sağlayıcı kullanılabilir mi?*(örneğin servis örnekleri dosya tabanlı bir sistemde veya farklı bir veritabanı üzerinde saklanabilirler mi? **Oracle, Access** vb.)*

İlk olarak 1nci sorumuza cevap arayarak yazımıza devam edelim. Bilindiği üzere istemci uygulama, servise ait bir proxy nesnesi ürettiğinde ve bunun üzerinde bazı uzak operasyonları çağırdığında, her iki taraf arasında **oturum(Session)** bazlı olarak dolaşan bir **InstanceId** değeri söz konusudur. Buna göre istemci uygulamanın daha önceden depolama alanına attığı bir servis içeriğini tekrardan kullanabilmesi için, depolama alanında duran bu servis örneğinin **InstanceId** değerine ihtiyacı vardır. Bu değer bilindiği

üzere servis tarafında üretilip istemciye içerik ile birlikte gönderilir ve operasyon çağrılarında karşılıklı olarak kullanılır. Dolayısıyla istemci uygulama, kullanmak istediği dayanıklı WCF servisine ait **GUID** tipinden olan **InstanceId**değerine sahip ise, saklanan veri içeriğini kullanıp işlemlerine bıraktığı yerden devam edebilir. Bu noktada tabiki istemcinin doğru **InstanceId** değerine sahip olması önemlidir.

Diğer taraftan **Exception** oluşmasına neden olabilecek pek çok durum vardır. Söz gelimi istemci uygulama servis örneğinin depolama alanından kaldırılmasına neden olan bir operasyon çağrısı yaptıktan sonra, **InstanceId** değerini kullanarak aynı servisin içeriğine kesinlikle ulaşamaz. çünkü servis tarafındaki depolama alanında bu **InstanceId** değerine sahip içerik artık bulunmamaktadır.*(Hemen hatırlayalım, varsayılan olarak **CompleteInstance=true** değerine sahip olan servis operasyonu, **Instance** içeriğinin depolama alanından silinmesine neden olmaktadır.)* Geliştirme sırasında bu ve benzeri istisnai durumlara dikkat edilmesi önemlidir. En azından istisna yönetim mekanizmalarından yararlanılmalı ve sistemin işleyişi çok fazla aksatılmamalıdır.

Şimdi dilerseniz ilk sorumuzun cevabını bulmak için kodlamaya yavaş yavaş geçelim. Bu noktada işlemleri daha kolay izleyebilmek adına bir önceki makalede geliştirdiğimiz **CommonService**isimli serviste küçük bir değişiklik yapıyoruz. Söz konusu değişiklikler ile en büyük amacımız, istemci tarafında **CommonValue** değerini takip ederek, ilk sorumuzu daha kolay analiz edebilmektir. Buna göre **IncreaseValue** metodunun, servis içerisindeki **CommonValue** alanının değerini geriye döndürmesini sağlıyoruz. Tabi bu değişikliğin hem servis sözleşmesinde hemde servis sözleşmesini uyarlayan tipin içerisinde yapılması gerekmektedir**.**

***NOT :** Servis tarafındaki sözleşmelerde değişiklik olması halinde(örneğin operasyonun dönüş tipi veya parametrelerinde değişiklik yapılması yada yeni operasyonların eklenmesi vb...) bu hizmeti kullanan istemci uygulamaların, servis referanslarını **Update** etmeleri mutlak suretle gereklidir.*

örneğimizde servis kütüphanesinde yapmış olduğu basit değişiklikler aşağıdaki gibidir.

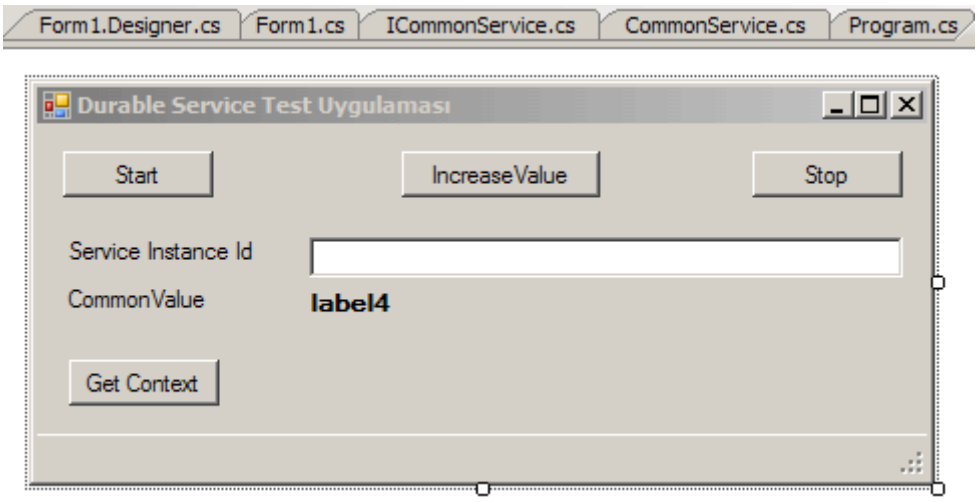
ICommonService isimli servis sözleşmesinde yapılan değişiklik;

```
[OperationContract]
int IncreaseValue(int value);
```

Servis sözleşmesini uyarlayan **CommonService** sınıfında yapılan değişiklik;

```
[DurableOperation()]
public int IncreaseValue(int value)
{
   commonValue += value;
   return commonValue;
}
```

Gelelim istemci tarafına. İstemci tarafında bu kez basit bir **Windows** uygulaması kullanıyor olacağız. *(İstemci uygulamamız sadece test amaçlıdır bu nedenle çok fazla beklentimiz olmamalıdır :)* İstemcimizin Form1 tasarımı aşağıdaki gibidir.



İstemci uygulamamızın kod içeriği ise şu şekildedir;

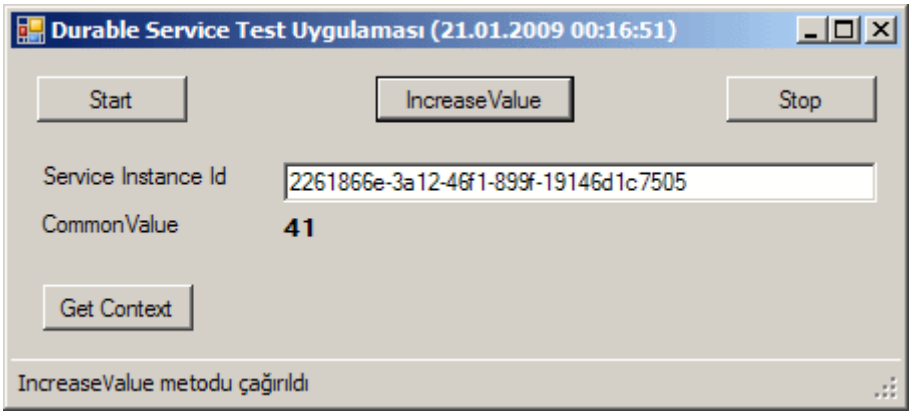
```
using System;
using System.Collections.Generic;
using System.Windows.Forms;
using WinClientApp.CommonServiceReference;
using System.ServiceModel;
using System.ServiceModel.Channels;

namespace WinClientApp
{
   public partial class Form1
      : Form
   {
      private ServiceCommonClient client=null;

      public Form1()
      {
         InitializeComponent();
         // Proxy nesnesi örneklenir.
         client = new
ServiceCommonClient("WSHttpContextBinding_ServiceCommon");
         lblStatus.Text = "Servis için proxy nesnesi örneklendi";
         Text += " (" + DateTime.Now.ToLongTimeString() + ")";
      }

      private void btnStart_Click(object sender, EventArgs e)
      {
```

```
        // Start metodu çağırılır. Bu metod çağrısı ile servis tarafında InstanceData tablosuna kayıt atılır.
        client.Start();
        lblStatus.Text = "Start metodu çağırıldı";
        txtInstanceId.Text = client.GetInstanceId().ToString();
    }

    private void btnIncreaseValue_Click(object sender, EventArgs e)
    {
        // Sembolik olarak değer arttırımı servis üzerinden yapılır
        lblCommonValue.Text=client.IncreaseValue(10).ToString();
        lblStatus.Text = "IncreaseValue metodu çağırıldı";
    }

    private void btnStop_Click(object sender, EventArgs e)
    {
        // Stop metodu çağırılır. Bu metod çağırıldığında servis tarafında InstanceData tablosunda saklanan satır silinir
        client.Stop();
        lblStatus.Text = "Stop metodu çağırıldı";
    }

    // Daha önceden kayıt edilmiş bir instance' ı kullanmak için aşağıdaki kod parçası kullanılır
    private void btnGetContext_Click(object sender, EventArgs e)
    {
        // IContextManager için System.WorkflowServices.dll assembly' ının referans edilmesi gerekir
        IContextManager conMng = ((IContextChannel)client.InnerChannel).GetProperty<IContextManager>();
        IDictionary<string, string> context = conMng.GetContext();

        if (!context.ContainsKey("instanceId"))
        {
            context.Add("instanceId", txtInstanceId.Text);
            conMng.SetContext(context);
        }
    }
  }
}
```

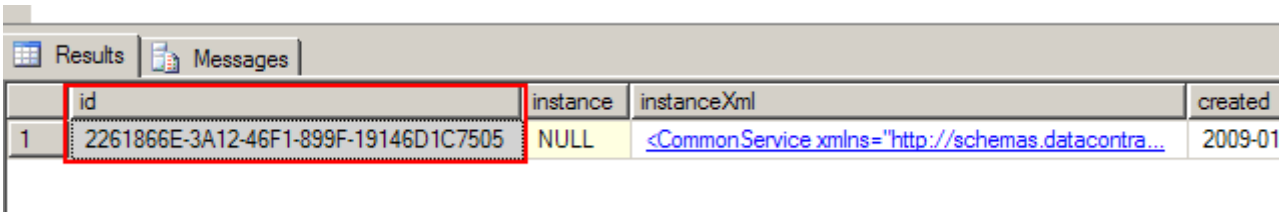
Bu uygulamada daha önceden depolanan bir servis örneğine ulaşılmak istendiğinden, **System.ServiceModel.Channels** isim alanındaki **IContextManager** isimli arayüzden yararlanılmaktadır. Bu arayüz **System.WorkflowServices.dll assembly**' ı içerisinde yer aldığından söz konusu **.Net Framework 3.5** kütüphanesinin istemci

uygulamaya referans edilmesi gerekmektedir. örneğimiz tabiki geliştiriciler için yazılmıştır. Son kullanıcıya verildiği takdirde çalışma zamanında pek çok istisnanın oluşması muhtemel ve kaçınılmazdır.*(Unutmayın şu anda bir test uygulaması geliştiriyoruz ve analiz yapmaya çalışıyoruz.)*

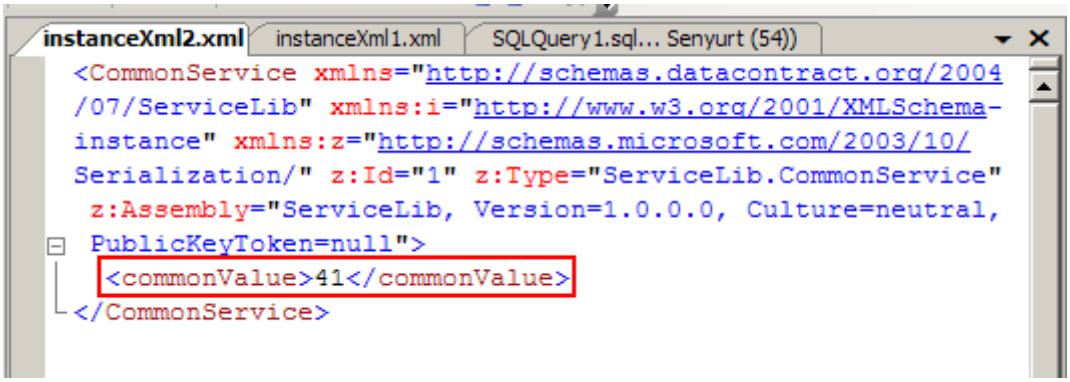
İlk olarak servis kütüphanesini*(Bir önceki makalemizden hatırlayacağınız gibi **WCF** kütüphaneleri otomatik olarak **WcfSvcHost** programı yardımıyla host edilebilmekte ve WcfTestClient ile test edilebilmektedir.)* ve windows uygulamamızı çalıştıralım. Sonrasında ilk olarak **Start** düğmesine basacağız.
Ardından **IncreaseValue** değerini arttırmak için bir kaç işlem yaptırılabiliriz. Ancak hiç bir şekilde **Stop** düğmesine basmayacağız. Bu en önemli test noktamız.
Nitekim **Stop** düğmesine basılması sonrasında servis tarafında çağırılan operasyon, sahip olduğu **DurableOperation** niteliğinin **CompleteInstace** özelliğinin **true** değerine sahip olması nedeni ile **instance**' ın kaldırılmasına neden olacaktır. Testin bu aşamasında uygulamamızın ekran görüntüsüde aşağıdakine benzer olacaktır. Yine burada testin ikinci aşaması için önem arz eden konulardan biriside servis tarafından üretilen**InstanceId** değeridir. Bu değeri saklamamız gerekiyor.*(Lütfen Form üzerindeki tarihe dikkat edin.)*



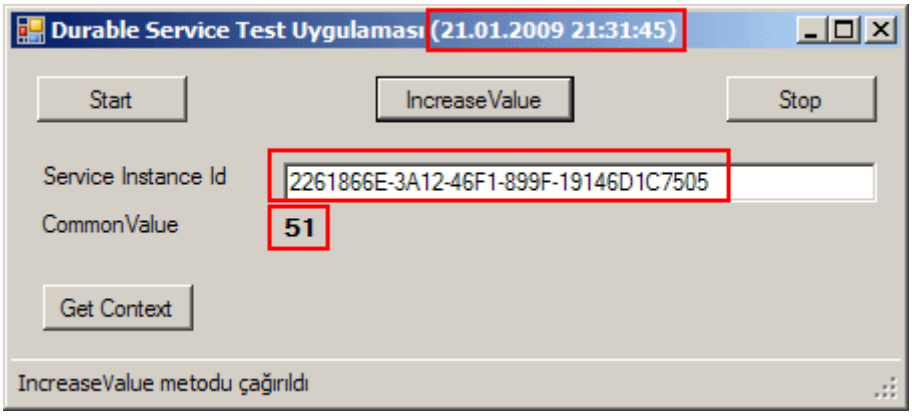
Testimizin ikinci aşamasında uygulamamızı kapatıp çıktığımızı düşünelim. Eğer veritabanına gider ve **InstanceData** tablosuna bakacak olursak, servis örneğinin satır bazında eklenmiş olduğunu rahatlıkla görebiliriz.



XML verisinde ise servis tarafındaki **CommonValue** değişkeninin aşağıdaki şekilde görüldüğü gibi **41** olarak tutulduğu gözlemlenebilir.



Gel zaman git zaman aradan uzunca bir süre geçer ve biz aynı uygulamayı bir kere daha çalıştırırız. Ancak bu sefer, **Start** metodu yerine daha önceden sakladığımız servis **InstanceId**değerini **TextBox** kontrolüne ekleyip **GetContext** isimli metodu çağırmalıyız. Buna göre daha önceden kayıt edilmiş olan servis örneğin içeriğinin kullanılması için ilk adımı atmış oluruz ki uygulamanın çalışma zamanı ekran görüntüsüde aşağıdaki gibi olacaktır.



Dikkat edilecek olursa uygulamayı çalıştırdığımıda **Form**' un **Text**' i üzerinde saat bilgisi olarak **00:16:51** yazmaktaydı.*(Gerçektende aynı günün akşamında örneği tekrardan çalıştırdığımı söyleyebilirim).* Ancak şu anda saat **21:31:45** ve servis örneğinin veritabanında kayıtlı olan içeriğini kullanaraktan, **CommonValue** değişkeninin son değerini 51 olarak değiştirmiş bulunuyoruz. Dolayısıyla bir servisin nasıl dayanıklı hale getirilebileceğini ve bununla birlikte istemcilerin daha önceki **instance**' lara nasıl ulaşabileceklerini görmüş olduk.

Elbetteki gerçek hayat vakalarında söz konusu **instanceId** değerlerini istemci tarafında daha düzenli bir şekilde saklamak gerekebilir. Belikde uygulama kapatılırken **instanceId** değeri konfigurasyon düzeyinde saklanabilir ve bir sonraki açılışta değerlendirilerek istemcinin, servis içeriğinin son hali üzerinden otomatik olarak devam etmeside sağlanabilir. Tahmin edeceğiniz üzere dayanıklı WCF servisleri, **Workflow** tarzındaki uygulamalarda daha yaygın olarak kullanılmaktadır. Nitekim **WF** uygulamaları kendi içlerinde zaten var olan bir sürerlik yapısına sahiptir ve bir akışın anlık olarak uzun süreliğine saklanması mümkündür. öyleki **WCF** servislerini

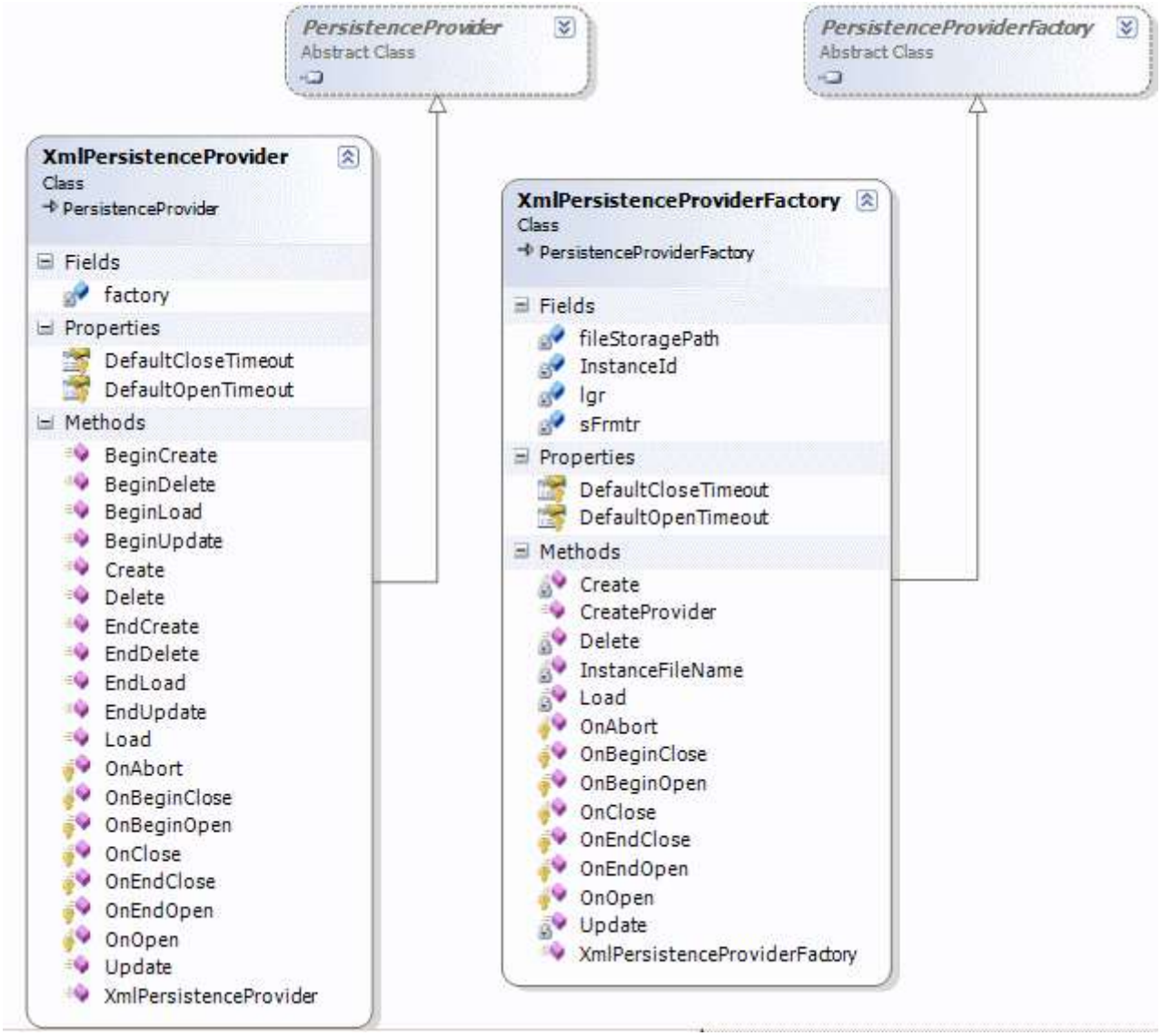
dayanıklı hale getirmek içinde Workflow' un ana assembly' larından olan **System.WorkflowServices.dll**' dan yararlanılmaktadır.

Gelelim makalemize konu olan ikinci sorumuza. **özelleştirilmiş bir sürerlik sağlayıcısını(Custom Persistence Provider) nasıl geliştirebiliriz?** Burada iş biraz daha zorlaşmaktadır. Bu konudaki en güzel örnek **Microsoft** tarafından geliştirilmiş olup **MSDN** den indirip inceleyebilirsiniz. Biz çok daha basit haliyle ele alacağız ve bu nedenle özellikle asenkron yaklaşımları göz ardı edeceğiz.

İlk olarak **PersistenceProviders** isimli bir **sınıf kütüphanesi(Class Library)** oluşturarak işe başlayabiliriz. Söz konusu kütüphane içerisinde özel bir **sürerlik sağlayıcısı(Persistence Provider)** kullanacağımız için **System.ServiceModel.dll**, **System.WorkflowServices.dll assembly**' larını referans etmemiz gerekiyor. Bunların haricinde örneğimizde kullanacağımız özel sağlayıcı, çalışma zamanındaki servis örneklerine ait verileri **SOAP(SimpleObjectAccessProtocol)** serileştirme yaparak saklayacaktır.

NOT : *Elbetteki özel sürerlik sağlayıcısının hangi tipte veri kaynağı ile çalışacağı tamamen geliştiriceye ve ortam şartlarına bağlıdır. Sadece dosya sistemi değil**Oracle** gibi **SQL Server** harici veritabanları da kolaylıkla ele alınabilir.*

Biz örneğimizde **SOAP** serileştirmesini kullanacağımızda **System.Runtime.Serialization.Formatters.Soap.dll assembly**' ınında sınıf kütüphanesine referans edilmesi gerekmektedir. Model temel olarak **Abstract Factory tasarım desenine** uygun olarak geliştirilmelidir.



Doğruyu söylemek gerekirse biraz karmaşık görünen bir yapı. Ancak incelendiği takdirde içerisinde **asenkron** operasyonların dahi yer aldığını*(ki bunun uyarlamalarını eklemedik kodlarımıza)*, **Abstract Factory** deseninin uygulandığını rahatlıkla görebiliriz. Bir başka deyişle **XmlPersistenceProviderFactory** sınıfı, özel depolama işlemleri için**XmlPersistenceProvider** tipini örneklemektedir. Bu, **WCF** servislerinde özel sağlayıcıların üretimi sırasında kullanılan standart yoldur. örnekte kullandığımız**XmlPersistenceProviderFactory** tipi, **abstract PersistenceProviderFactor y** tipinden türemektedir. Bu sınıfın içerisinde **ezilmek(override)** zorunda olan **CreateProvider** isimli metod ise**PersistenceProvider** tipinden bir referans döndürmektedir. örneğimizde söz konusu dönüş referansı olarak **PersistenceProvider** sınıfından türetilen **XmlPersistenceProvider** tipi kullanılmaktadır. Buna ek olarak **PersistenceProviderFactory** tipi **CommunicationObject**' ten türemektedir ve bu sebeptende **CreateProvider** haricinde ek metodlarıda uyarlamaktadır. Söz gelimi**OnBeginClose**, **OnEndClose** gibi metodlar veya **DefaulfCloseTimeout**, **DefaultOpenTimeout** gibi

özellikler **CommunicationObject** tipi içerisinde **abstract** olarak tanımlandıklarından otomatik olarak**XmlPersistenceProviderFactory** sınıfınada uygulanmaktadırlar. Ek olarak, **XmlPersistenceProvider** tipinin ortaklaşa kullanacağı **Create**,**Load**,**Update**,**Delete** gibi işlevselliklere ait metodlar,**XmlPersistenceProviderFactory** tipi içerisinde yer almaktadır. İki sınıf arasındaki iletişimin sağlanması amacıyla **XmlPersistenceProvider** tipinin **yapıcı metodu(Constructor Method)**içerisine, **XmlPersistenceProviderFactory** sınıfının çalışma zamanı referansı aktarılmaktadır. Böylece **XmlPersistenceProvider** tipi içerisinden, factory sınıfında tanımlanan genel üyelere erişilebilir. *(**PersistenceProvider** türevli sınıf istenirse **ProviderPersistenceFactory** türevli sınıf içerisinde **dahili tip-inner type** olarakta tasarlanabilir. Microsoft'un örneğinde bu tarz bir kullanım söz konusudur.)* **XmlPersistenceProviderFactory** tipi ile ürettiği **XmlPersistenceProvider** sınıfının içerikleri ise aşağıdaki gibidir.

XmlPersistenceProviderFactory içeriği;

```
using System;
using System.ServiceModel.Persistence;
using System.IO;
using System.Runtime.Serialization.Formatters.Soap;

namespace PersistenceProviders
{
   public class XmlPersistenceProviderFactory
      : PersistenceProviderFactory
   {
      #region Gerekli Sınıf Değişkenleri

      // Servis örneklerinin saklanacağı varsayılan klasörü tutan değişken
      private string fileStoragePath = Path.Combine(System.Environment.CurrentDirectory,
"Instances");
      // Soap serileştirmesini yapacak olan SoapFormatter değişkeni
      SoapFormatter sFrmtr = null;
      Logger lgr = null;
      Guid InstanceId;

      #endregion

      #region Yardımcı Metodlar

      private string InstanceFileName()
      {
         // Dosya adı belirlenir. Ad belirlenirken ayırt edici olması için base sınıftan Id
özelliğinin değeri kullanılır
         string fileName = Path.Combine(fileStoragePath, InstanceId.ToString() + ".sxml");
```

```
        return fileName;
    }

    #endregion

    #region XmlPersistenceProvider sınıfı için kullanılan ortak CRUD metodları

    // Servis örneği için depolama kaynağının oluşturulması ve içeriğinin serileştirilmesi amacıyla kullanılır
    public object CreateInstance(object instance, TimeSpan timeout)
    {
        string fileName = InstanceFileName();

        // Soap formatter tipinden yararlanılarak servis örneğinin serileştirilmesi sağlanır
        using (FileStream stream = new FileStream(fileName, FileMode.Create, FileAccess.Write))
        {
            sFrmtr.Serialize(stream, instance);
        }

        lgr.AddEventEntry(fileName + " için Create işlemi yapıldı, Serileştirme gerçekleştirildi.", System.Diagnostics.EventLogEntryType.Information);
        return null;
    }

    // Servis örneğinin depolama kaynağının silinmesi amacıyla kullanılır
    public void DeleteInstance(object instance, TimeSpan timeout)
    {
        // Dosya adı elde edilir
        string fileName = InstanceFileName();
        // Eğer dosya sistemde var ise
        if (File.Exists(fileName))
        {
            // Dosya silinir
            File.Delete(fileName);
            // Silme bilgsi loglanır
            lgr.AddEventEntry(fileName + " için Delete işlemi yapıldı.", System.Diagnostics.EventLogEntryType.Information);
        }
        else // yok ise
        {
            // Dosya sistemde bulunamadıysa normal şartların aksine başka bir nedenle silinmiş olabilir. Bu durum loga aktarılır
            lgr.AddEventEntry(fileName + " sistemde bulunamadığından Delete işlemi yapılamadı", System.Diagnostics.EventLogEntryType.Warning);
```

```
        }
    }

    // Kaydedilmiş olan servis örneğinin kaynaktan alınıp kullanılabilir object haline
getirilmesi için kullanılır
    public object LoadInstance(TimeSpan timeout)
    {
        string fileName = InstanceFileName();
        object result = null;
        if (File.Exists(fileName)) // dosya mevcutsa
        {
            // loglama yap
            lgr.AddEventEntry(fileName + " için Load işlemi yapıldı",
System.Diagnostics.EventLogEntryType.Information);
            // servis örneğinin dosya içerisinde tutulan serileştirilmiş örneğini deSerialize
ederk geriye döndür
            using (FileStream stream = new FileStream(fileName, FileMode.Open,
FileAccess.Read))
            {
                result = sFrmtr.Deserialize(stream);
                stream.Close();
            }
            return result;
        }
        else // dosya bulunamıyorsa
        {
            // uyarı mesajı niteliğinde loglama yap
            lgr.AddEventEntry(fileName + " sistemde bulunamadığından Load işlemi
gerçekleştirilemedi", System.Diagnostics.EventLogEntryType.Warning);
            return null;
        }
    }

    // Kaynakta tutulan servis örneğinin güncellenmesi işlemini gerçekleştirir.
    public object UpdateInstance(object instance, TimeSpan timeout)
    {
        // Dosya adı alınır
        string fileName = InstanceFileName();

        // Soap formatter tipinden yararlanılarak servis örneğinin serileştirilmesi sağlanır
        using (Stream stream = new FileStream(fileName, FileMode.Create,
FileAccess.Write))
        {
            // Güncellenen servis örneği tekrardan aynı dosya üzerine serileştirilir
            sFrmtr.Serialize(stream, instance);
```

```
        }

        lgr.AddEventEntry(fileName + " için Update işlemi yapıldı, Serileştirme
gerçekleştirildi.", System.Diagnostics.EventLogEntryType.Information);
        return null;
    }

    #endregion

    public XmlPersistenceProviderFactory()
    {
        // Eğer servis örneklerinin saklanacağı varsayılan klasör yoksa oluştur
        if (!Directory.Exists(fileStoragePath))
            Directory.CreateDirectory(fileStoragePath);

        // Soap Formatter değişkeni oluşturulur
        sFrmtr = new SoapFormatter();

        // Loglama işlemini yapacak olan sınıf örneği oluşturulur
        lgr = new Logger("CustomXmlPersistenceProvider", "Custom Xml Persistence
Log(WCF)");

        lgr.AddEventEntry("XmlPersistenceProviderFactory örneği oluşturuldu",
System.Diagnostics.EventLogEntryType.SuccessAudit);
    }

    public override PersistenceProvider CreateProvider(Guid id)
    {
        InstanceId = id;
        return new XmlPersistenceProvider(id, this);
    }

    protected override TimeSpan DefaultCloseTimeout
    {
        get { return TimeSpan.FromSeconds(10); }
    }

    protected override TimeSpan DefaultOpenTimeout
    {
        get { return TimeSpan.FromSeconds(10); }
    }

    protected override void OnAbort()
    {
    }
```

```
        protected override IAsyncResult OnBeginClose(TimeSpan timeout,
AsyncCallback callback, object state)
        {
            return null;
        }

        protected override IAsyncResult OnBeginOpen(TimeSpan timeout,
AsyncCallback callback, object state)
        {
            return null;
        }

        protected override void OnClose(TimeSpan timeout)
        {
        }

        protected override void OnEndClose(IAsyncResult result)
        {
        }

        protected override void OnEndOpen(IAsyncResult result)
        {
        }

        protected override void OnOpen(TimeSpan timeout)
        {
        }
    }
}
```

Görüldüğü gibi basit **serileştirme(Serialization)** ve **ters-serileştirme(DeSerialization)** işlemleri gerçekleştirilmektedir. Bununla birlikte örneğin bizim için*(en azından benim için)* kolay anlaşılabilir olması adına asenkron metodların içerikleri uyarlanmamıştır.

XmlPersistenceProvider sınıfının içeriği;

```
using System;
using System.ServiceModel.Persistence;

namespace PersistenceProviders
{
    public class XmlPersistenceProvider
        : PersistenceProvider
```

```
{
    XmlPersistenceProviderFactory XmlPrsFactory;

    public XmlPersistenceProvider(Guid id, XmlPersistenceProviderFactory
xmlFactory)
        : base(id)
    {
        XmlPrsFactory = xmlFactory;
    }

    #region PersistenceProvider ve CommunicationObject üyeleri

    // Servis örneği depolama alanında ilk oluşturulduğunda çalışır.
    public override IAsyncResult BeginCreate(object instance, TimeSpan timeout,
AsyncCallback callback, object state)
    {
        return null;
    }

    // Silme işlemi başladığında devreye girer.
    public override IAsyncResult BeginDelete(object instance, TimeSpan timeout,
AsyncCallback callback, object state)
    {
        return null;
    }

    // Yükleme işlemi başladığından devreye girer.
    public override IAsyncResult BeginLoad(TimeSpan timeout, AsyncCallback
callback, object state)
     {
        return null;
    }

    // Servis örneği güncellemeye başladığında devreye girer
    public override IAsyncResult BeginUpdate(object instance, TimeSpan timeout,
AsyncCallback callback, object state)
    {
        return null;
    }

    // Depolama alanında servis örneğinin kayıt altına alınmasında kullanılır
    public override object Create(object instance, TimeSpan timeout)
    {
        return XmlPrsFactory.CreateInstance(instance, timeout);
    }
```

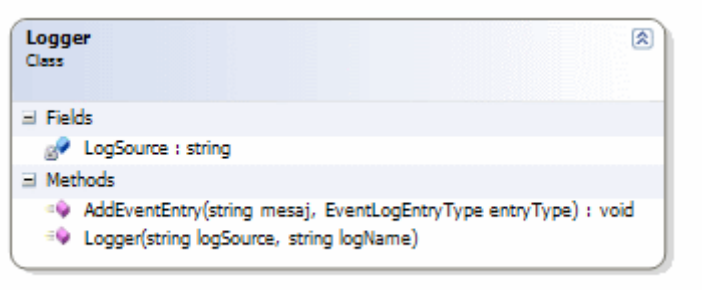
```
// Servis örneğinin depolama alanında silinmesinin ele alındığı metoddur
public override void Delete(object instance, TimeSpan timeout)
{
   XmlPrsFactory.DeleteInstance(instance, timeout);
}

// Servis örneğinin veri içeriğinin depolama alanına ilk kayıt edildiği aşamanın sonunda devreye giren metoddur
public override object EndCreate(IAsyncResult result)
{
   return null;
}

// Servis örneğine ait veri içeriğinin depolama alanında silindiği aşamının sonunda devreye giren metoddur
public override void EndDelete(IAsyncResult result)
{
}

// Servis örneği verisinin depolama alanında ortama yüklendiği aşamın sonunda devreye giren metoddur
public override object EndLoad(IAsyncResult result)
{
   return null;
}

// Servis örneği verisinin depolama alanında güncelleştirilmesi sonrasında devreye giren metoddur
public override object EndUpdate(IAsyncResult result)
{
   return null;
}

// Servis örneği verisinin depolama alanından yükenmesi aşamasının ele alındığı metoddur
public override object Load(TimeSpan timeout)
{
   return XmlPrsFactory.LoadInstance(timeout);
}

// Servis örneği verisinin depolama alanında güncelleştirilmesi aşamasının ele alındığı metoddur
public override object Update(object instance, TimeSpan timeout)
{
```

```
        return XmlPrsFactory.UpdateInstance(instance, timeout);
    }

    // Kapatma operasyonun tamamlanması için gereken varsayılan sürenin belirlendiği
özelliktir
    protected override TimeSpan DefaultCloseTimeout
    {
        get
        {
            return TimeSpan.FromSeconds(10);
        }
    }

    // Açma operasyonunun tamamlanması için gereken varsayılan sürenin belirlendiği
özelliktir
    protected override TimeSpan DefaultOpenTimeout
    {
        get
        {
            return TimeSpan.FromSeconds(10);
        }
    }

    protected override void OnAbort()
    {
    }

    protected override IAsyncResult OnBeginClose(TimeSpan timeout,
AsyncCallback callback, object state)
    {
        return null;
    }

    protected override IAsyncResult OnBeginOpen(TimeSpan timeout,
AsyncCallback callback, object state)
    {
        return null;
    }

    protected override void OnClose(TimeSpan timeout)
    {
    }

    protected override void OnEndClose(IAsyncResult result)
    {
```

```
        }

        protected override void OnEndOpen(IAsyncResult result)
        {
        }

        protected override void OnOpen(TimeSpan timeout)
        {
        }

        #endregion
    }
}
```

Dikkat edileceği üzere **XmlPersistenceProvider** tipi ağırlıklı olarak **CRUD** işlemleri için gerekli olan zorunlu metod çağrılarını **PersistenceProvider abstract** sınıfından devralmaktadır. **Create**, **Update**, **Delete** ve **Load** isimli zorunlu olarak ezilen metodlar, **XmlPersistenceProviderFactory** tipi içerisinde tanımlanmış olan **CreateInstance**, **UpdateInstance**,**DeleteInstance** ve **LoadInstance** fonksiyonellikler ini çağırmaktadır. Aslında **Factory** sınıfı kendisini kullanan birden fazla **PersistenceProvider** türevli tipide ele alabilir. Hepsi için ortak olabilecek pek çok fonksiyonellik fabrika tipi içerisinde toplanabileceği gibi, **PersistenceProvider** türevli tiplerede yayılabilir. *(Bence bu konuyu araştırabilirsiniz. Tek bir **PersistenceProviderFactory**türevli tip üstünden birden fazla **PersistenceProvider** türevli tipi kullanabilmek ve bunları farklı vakalar için ele alabilmek.)*

Son olarak, kütüphanemizde süreci, oluşabilecek çalışma zamanı hatalarını loglamak amacıyla **Logger** isimli birde yardımcı sınıfımız bulunmaktadır. Bu sınıf yardımıyla kodun gerekli yerlerinde ilgili bilgileri günlüklere işleme şansına sahip oluyoruzki, **geliştirme(Development)** ve özellikle **test** aşamalarında bu oldukça işimize yarayacak bir özelliktir. Söz gelimi **WCF** kütüphanesinin bir **Windows Servisi** içerisinde **host** edilmesi halinde, hataların ayıklanması için ekstra çaba sarfetmek gerekmektedir. Bu nedenle **Windows Servisinin host** olarak kullanıldığı bir senaryoda **log** kayıtları hayati bilgiler verebilir ve hataların tespit edilmesi çok daha kolaylaşabilir.



Logger sınıfımızın içeriği ise aşağıdaki gibidir;

```
using System.Diagnostics;

namespace PersistenceProviders
{
   /// <summary>
   /// Loglama işlemleri için yardımcı sınıftır
   /// </summary>
   public class Logger
   {
      private string LogSource;

      /// <summary>
      /// Event Viewer içerisinde ayrı bir log sekmesi oluşturur
      /// </summary>
      /// <param name="logSource">Log verisinin kaynağı(Genellikle işlem adı yada
program adı olarak kullanılabilir)</param>
      /// <param name="logName">Event Viewer altındaki boğumun adıdır</param>
      public Logger(string logSource, string logName)
      {
         LogSource = logSource;
         if(!EventLog.Exists(logName))
            EventLog.CreateEventSource(logSource, logName);
      }

      /// <summary>
      /// Yapıcı metodda oluşturulan log kaynağına event yazdırır
      /// </summary>
      /// <param name="mesaj">Event ile ilişkili bilgi</param>
      /// <param name="entryType">Event in EventLogEntryType tipinden
değeridir</param>
      public void AddEventEntry(string mesaj, EventLogEntryType entryType)
      {
         EventLog.WriteEntry(LogSource, mesaj, entryType);
      }
   }
}
```

örnek sürerlik sağlayıcımız, uygulamanın olduğu yerde*(yada serivis host eden uygulamanın olduğu yerde)* **Instances** adlı bir klasör açmaktadır.

NOT : *Uygulamada dosyaların kaydedileceği klasör bilgisi konfigurasyon seviyesinde de tutulabilir. Hatırlayalım,* ***SqlPersistenceProviderFactory*** *sağlayıcısı kullanıldığında*

örneklerin saklanması için kullanılacak veritabanı bağlantısı, ***ConnectionString*** *elementinin üzerinden alınmaktadır.*

çalışma zamanındaki servislere ait örnekler bu klasör altında **sxml** uzantılı olarak tutulmaktadır. İçerikleri tahmin edileceği üzere **SOAP** formatındadır. Dosya adları servis örneğine ait **InstanceId** değerleri olarak verilmektedir. Akış son derece basittir. **DurableOperation** niteliğinin **CanCreateInstance** özelliğinin değeri **true** olan operasyon çağrısı ile bu klasörde bir dosya oluşturulmakta ve servis örneğine ait içerik serileştirilmektedir. Sonrasında yapılan operasyon çağrılarında bu dosya içeriğinin ters serileştirilerek **object** olarak elde edilmesi ve içeriğinin kullanılması söz konusudur. Yine **DurableOperation** niteliğinin **CompleteInstance** özelliği **true** olan operasyon çağırıldığında söz konusu servis örneğine ait dosya, **Instances** klasöründen silinmektedir. Tabi istenirse söz konusu dosyanın silinmeyip yeniden isimlendirilerek örneğin belirli süreliğine sistemde tutulması gibi işlemlerde yapılabilir.

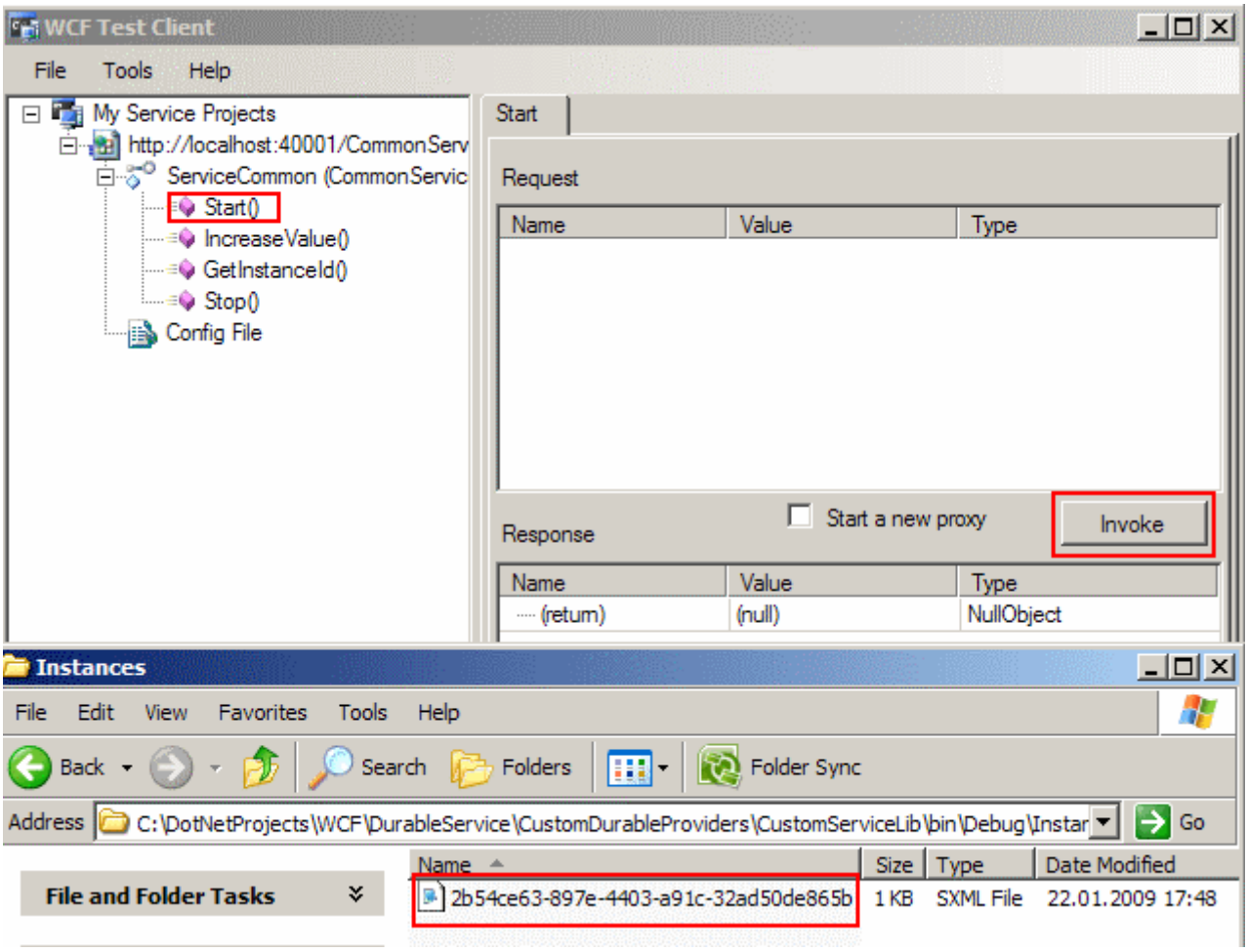
Bu teorik akışı elbette test ederek onaylamamız gerekmektedir. Bunun için basit bir istemci ve servis host uygulaması yazabilir yada özel sürerlik sağlayıcılarını kullanan servis kütüphanesini doğrudan **Visual Studio 2008** ortamında çalıştırabiliriz. Hangisi kullanılırsa kullanılsın servis tarafındaki konfigurasyon içeriğinde bir önceki makalemizde yaptığımız gibi **PersistenceProviderFactory** türevli tipin belirtilmesi gerekmektedir. örnek uygulamada bu amaçla geliştirilmiş olan **CustomServiceLib** isimli servis kütüphanesi bir önceki makalede geliştirdiğimiz **ICommonService** sözleşmesi ile **CommonService** tipini kullanmaktadır. **App.config** dosyasının içeriği ise aşağıdaki gibidir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="CommonServiceBehavior">
          <serviceDebug includeExceptionDetailInFaults="true" />
          <serviceMetadata httpGetEnabled="true" />
          <persistenceProvider
type="PersistenceProviders.XmlPersistenceProviderFactory,PersistenceProviders,
Verison=1.0.0.0" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service behaviorConfiguration="CommonServiceBehavior"
name="CustomServiceLib.CommonService">
        <endpoint
address="" binding="wsHttpContextBinding" bindingConfiguration=""
name="CommonServiceWsHttpEndPoint"
contract="CustomServiceLib.ICommonService" />
```

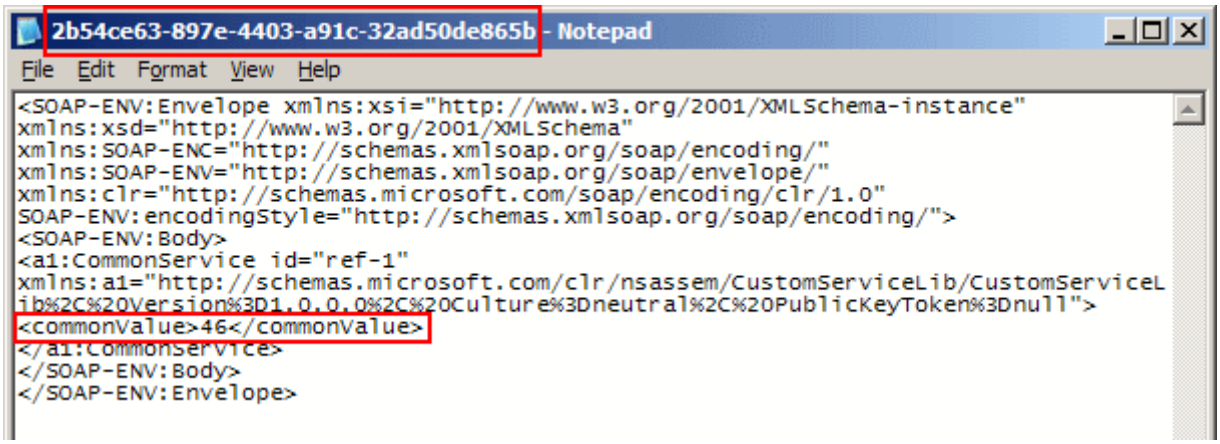
```
            <endpoint binding="mexHttpBinding" bindingConfiguration=""
name="MexEndPoint" contract="IMetadataExchange" address="Mex" />
            <host>
               <baseAddresses>
                  <add baseAddress="http://localhost:40001/CommonServiceV2" />
               </baseAddresses>
            </host>
         </service>
      </services>
   </system.serviceModel>
</configuration>
```

Görüldüğü üzere, **persistenceProvider** elementi içerisinde özelleştirilmiş olan **XmlPersistenceProviderFactory** tipi belirtilmektedir. Söz konusu tipin bulunduğu kütüphane referans etme yoluyla kullanıldığından br önceki örnekte olduğu gibi **Culture** veya **public key** bilgileri bulunmamaktadır. Elbetteki özelleştirilmiş depolama sağlayıcısını bünyesinde barındıran kütüphaneyi **Global Assembly Cache**' e atabiliriz. Bu durumda konfigurasyon içerisinde **Full Qualified Name**' in belirtilmesi gerekmektedir.

CustomServiceLib kütüphanesini direkt olarak **Visual Studio 2008** üzerinden çalıştırarak ilerleyebiliriz. **WcfSvcHost** ve paralelinde **Wcf Test Client** çalıştırıldıktan sonra ilk yapmamız gereken **Start** metodunu çağırmak olmalıdır. Bu noktada uygulamanın çalıştığı yerdeki **Instances** klasörüne baktığımızda aşağıdaki ekran görüntüsünde olduğu gibi **sxml** uzantılı ve **GUID** değeri ile adlandırılmış bir dosyanın oluşturulduğunu görürüz.



Bu adımdan sonra test olarak **IncreaseValue** metodu değişik değerler ile çağırlabilir. Operasyonun her çağırılmasında, servis örneğine ait içeriğin oluşturulan dosyadan yüklenmesi, ters serileştirildikten sonra içeride yer alan **CommonValue** değerinin arttırılması ve örneğin son halinin tekrardan aynı dosya üzerine serileştirilmesi adımları gerçekleşir. örneğin şu noktada **sxml** isimli dosyanın içeriğine bakılırsa aşağıdaki **XML** bilgisinin tutulduğu görülebilir.



Bu andan itibaren **Stop** metodu çağırılmadan **Wcf Test Client** uygulamasından çıkılabilir. Hatta aynı örnek defalarca çağırılıp **Stop** operasyonu kullanılmadan test edilebilir. Her

durumda **Instances** klasöründe birer dosya oluşturulacak ve herhangibir nedenle silinene kadar sahip oldukları **InstanceId** değerleri için servis tarafı istemcilere hizmet verebilecektir. Ancak Stop metodu çağırılırsa bu durumda Instances klasöründe bununla ilişkili olan dosyanın silindiği açık bir şekilde görülebilir. İşleyişi daha net bir şekilde anlayabilmek için kodu **debug**ederek çalıştırmanızı öneririm. Bu durumda **Create**, **Update**, **Delete**, **Load** metodları arasındaki geçişleri çok daha kolay bir şekilde analiz edebilirsiniz. Son olarakta servis kütüphanesini kullanan bir sunucu uygulama ve istemci geliştirerek yeni sağlayıcıyı test etmenizi ve sistemin düzgün yürüdüğünden emin olmanızı öneririm.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde dayanıklı WCF servisleri ilişkili iki önemli soruya cevap bulmaya çalışltık. İlk olarak istemcilerin var olan **InstanceId**' leri kullanarak servislerin, sunucu üzerinde uzun süreliğine saklanabilen hallerini nasıl kullanabileceklerine değinik. Sonrasında ise özel bir sürerlik sağlayıcısının**(Custom Persistence Provider)** nasıl yazılabileceğini gördük. Tabiki buradaki örnekleri geliştirmek, genişletmek tamamen sizin elinizdedir. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

CustomDurableProviders.rar (63,41 kb)

Dayanıklı WCF Servisleri (Durable WCF Services) (2009-01-16T22:24:00)

wcf,

Uzun zamandır Windows Communication Foundation(WCF) konulu bir araştırma üzerinde durmamıştık. Aslında **WCF 4.0** ve **WF 4.0** ile ilgili yeniliklerin uzun süredir konuşulduğu şu günlerde, WCF tarafında oldukça önemli bir yere sahip olan ve hatta Workflow Foundation açısından da değer arz eden bir konuya değiniyor olacağız. **Dayanıklı WCF Servisleri(Durable WCF Services)**.

Durable deyince aklıma gelen ilk şey, uzun ömürlü bir pil markası oluyor :) Sürer...Sürer...Sürerrr...İşin mizahi yanı bir tarafa dursun, **Servis Yönelimli Mimari(ServiceOrientedArchitecture)** uygulamalarında, bir servisin herhangibir zaman dilimindeki konumunun içeriği ile birlikte uzun süreliğine saklanabiliyor olması bazen elzem olan gereksinimlerden bir tanesidir. Nitekim bazı servis operasyonlarının gerçekleştirilmesi sırasında servisin kapanması veya istemcinin tekrardan başlatılması gibi vakaları olasıdır. Yada çok uzun süren, içerisinde insan faktörü olan bir sürecin ele alındığı herhangibir servisin, belirli zaman dilimlerinde son içeriğinden yüklenerek çalıştırılması gerekebilir. Normal şartlarda **WCF**servisleri istemci ile **oturum bazlı(Session Based)** haberleşmektedir. Buna göre istemciler her proxy nesnesi örnekleyip servise bağlandıklarında bir oturum açmış olurlar. Ne varki servisin herhangibir nedenle düşmesi, istemcinin programı kapatması gibi durumlarda bu oturum bilgileri çok doğal olarak kaybolabilir. Servisin düşmesi sonrasında, üzerinde taşıdığı tüm veriler bir tedbir alınmadığı sürece yok olabilir. İstemci uygulama yeniden başlatıldığında, servis ayakta bile

olsa yeni bir **proxy** örneği oluşturduğundan, daha önceki servis verilerine çok doğal olarak ulaşılamayadabilir. Halbuki belkide istemci tarafından çağırılan operasyonların bazıları, kendisini zaman dilimleri içerisinde koruyan verilerin son haline ihtiyaç duyabilir.

özellikle **WF** yapısı içerisinde de, bu tip durağansızlıklar oldukça önemli vakalar arasında yer almaktadır. öyleki bir WF içerisinde **çok uzun süren operasyonların(Long Running Operations)** olması çok daha yüksek bir ihtimaldir. Bu nedenle WF' in kendi doğasında olan akışın herhangibir zamandaki durumunu o anki verileri ile saklamak gibi bir misyonuda var**(Persistence)**. Günümüzde WF ile WCF' in özellikle **.Net Framework 3.5** sürümünden itibaren birbirleri ile daha sıkı fıkı olduklarıda göz önüne alındığında, konunun önemi bir kat daha artmaktadır. Bu basit bilgiler dahi, bir servisin içerisindeki operasyon bazlı verilerin kalıcı olarak saklanabilmesinin gerekliliğinide ortaya çıkarmaktadır. öyleyse biraz daha derinlere dalmaya ne dersiniz?

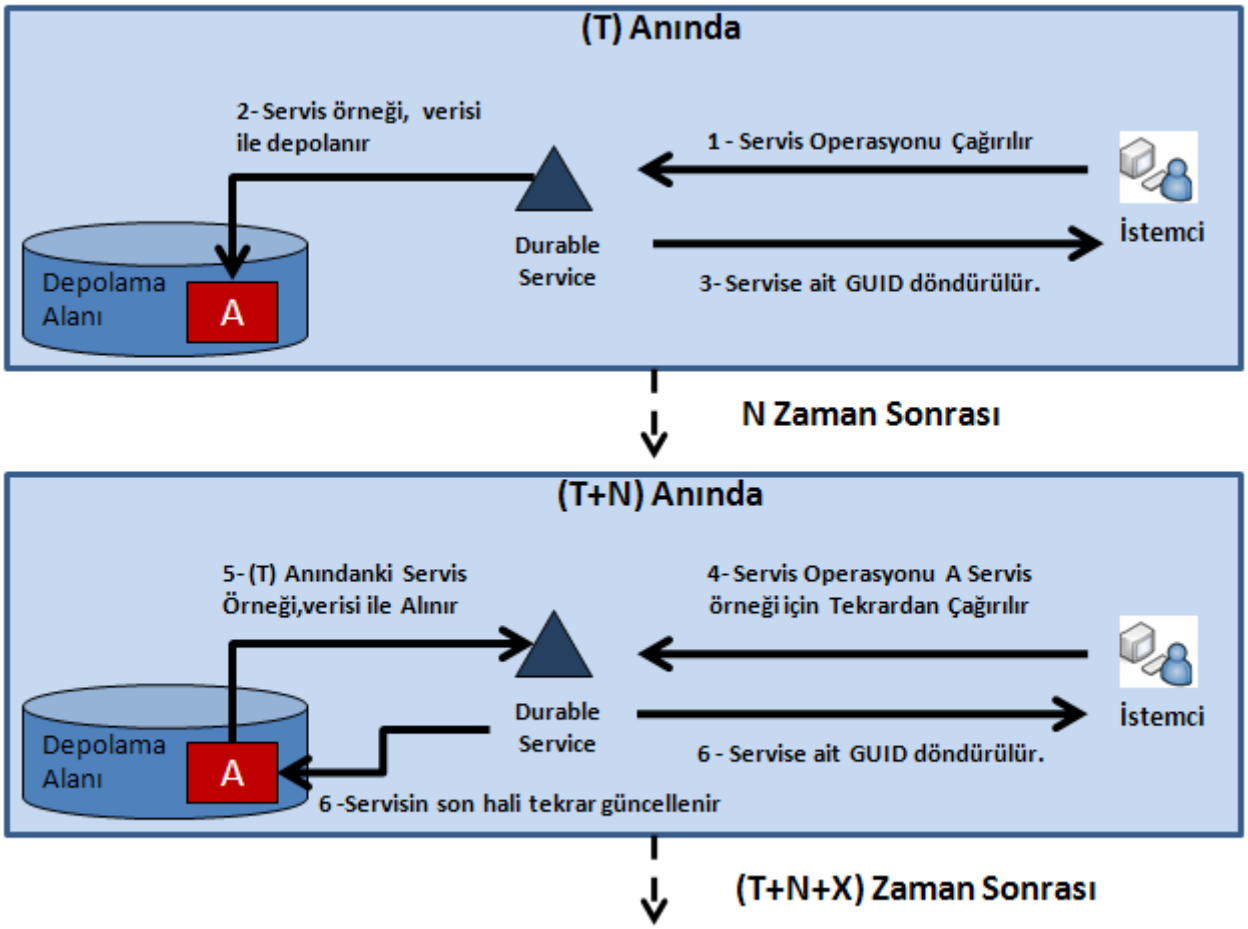
Herşeyden önce servis tarafında kalıcı olarak saklanacak olan nesne nedir? Bu çok doğal olarak servis sınıfına ait sunucu tarafından üretilen örneğin kendisi olmalıdır. Diğer bir noktada, servis nesnesinin nerede saklanacağıdır? Burada akla gelen en pratik çözüm çok doğal olarak bir veritabanıdır. Nitekim veritabanı içerisinde fiziki olarak servis verilerin saklanması, saklanan satırların kolayca yönetilmesi ve daha pek çok avantaj söz konusudur. çok şükürki veritabanı tarafında gerekli **tabloların(Tables)** ve **saklı yordamların(Stored Procedures) betikleri(scripts) .Net Framework 3.5** ile birlikte gelmektedir.*(.Net Framework 3.0 içerisindede bu script dosyalarını olduğunu vurgulayalım. Ancak 3.5 tarafında pek çok bug' ın düzeldiğide bir gerçektir.)* Her ne kadar buradaki veritabanı yapısını kurmak düşünüldüğünde çok zor olmasada, **.Net Framework 3.5** içerisinde bu işlem için gerekli tipler(Types) ve veritabanı oluşturma betiklerinin hazır olarak gelmesi geliştiricilerin işini oldukça kolaylaştırmaktadır.

NOT : *Servis nesnelerinin depolanması için bir **SQL** veritabanının kullanılması şart değildir. Varsayılan olarak **System.WorkflowServices.dll** assembly' ı içerisindeki**System.ServiceModel.Persistence isim alanı(Namespace)** altında yer alan **SqlPersistenceProviderFactory** tipi kullanılmaktadır.Bu tip doğrudan belirtilen **SQL** bağlantı cümlesindenki depolama alanını kullanmaktadır. Ancak istenirse, **PersistenceProviderFactory abstract** sınıfından türetme yapılarak özel bir depolama alanının kullanılmasıda sağlanabilir.*

Depolama alanının belirtilmesi elbetteki yeterli değildir. Bununla birlikte **WCF çalışma zamanının(Runtime)**, **niteliklerden(Attributes)** yararlanarak depolama alanına servis örneklerini ekleme veya kaldırma gibi işlemleri hangi operasyonlar gerçekleştiğinde yapacağının bildirilmesi gerekmektedir. Hatta hangi servis tipinin **durable** olarak bırakılacağınında WCF çalışma zamanına söylenmesi önemlidir. Bu noktada devreye **Durable**, **DurableOperation** gibi **nitelikler(Attribute)** girmektedir.

Aslında **SQL** depolama alanı varsayılan olarak kullanıldığında, sistem son derece basit bir çalışma mantığına sahiptir. Dayanıklı WCF servislerinde istemcinin servis tipine ait bir proxy oluşturmasının ve Durable hale gelme özelliğini başlatan operasyonu çağırmasının

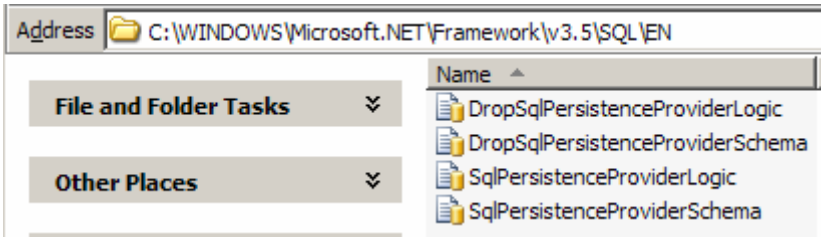
ardından, sunucu tarafındaki depolama alanında servis örneğinin serileştirilmiş bir örneği hemen ilgili tabloya kayıt olarak eklenmektedir. Bu kayıt bir **GUID** ile ilişkilendirilmekte ve böylece istemci söz konusu **GUID**' e sahip olduğunda, istediği zaman*(elbetteki tablodaki verinin başına bir şey gelmediği, istemci ile servis arasındaki ağ bağlantısında bir aksaklık olmadığı sürece vb...)* aynı servis örneğinin verisine erişip operasyonlarında kullanabilir. Anlattığımız bu çalışma sistemi varsayılan olarak **SQL** depolama alanı kullanıldığı durum için geçerlidir. Nitekim **GUID** üretimi, serileştirilen servis nesne örneğinin tabloya eklenmesi, bilgisinin güncellenmesi, silinmesi gibi adımlar veritabanı bağımlı işlemlerdir. Bu durumu aslında aşağıdaki şekil ile kafamızda biraz daha netleştirebiliriz.



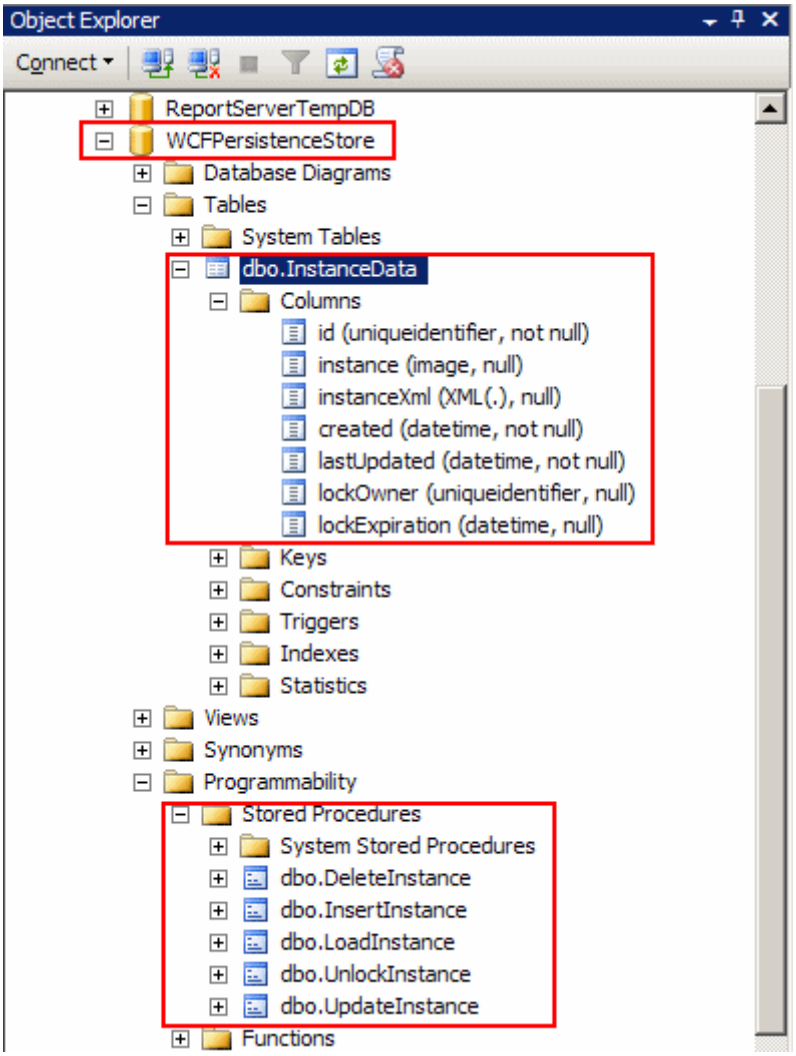
Bu görselde sadece iki farklı zaman dilimi için örnek bir durum göz önüne alınmaktadır. İlk olarak istemci proxy örneğini oluşturduktan sonra bir operasyon çağrısı gerçekleştirir. Bu operasyon çağrısı sonrasında, servis örneğinin depolama alanına atılması ve bir **GUID**' in üretilerek sonraki zaman dilimleri için istemciye gönderilmesi söz konusudur. Senaryo gereği ileri bir zaman diliminde istemci aynı servis örneğini daha önceden kaydettiği içeriği ile yeniden kullanmak istemektedir. Bu durumda servisin **T** zamanındaki örnek içeriği depolama alanında durduğundan ve istemcinin bu servisi bulması için gerekli **GUID** değeri var olduğundan*(ki bu değer aynı zamanda servisin instanceId değeri ile*

eşittir), **T+N** zamanında istemci uygulama, daha önceki servis içeriğini son haliyle kullanmaya devam edebilir. Elbette zaman dilimi ilerledikçe servisin durağanlığı devam ettirilebilir. Ne zamanki istemci, servis örneğinin tablodan silinmesi için gerekli talepte bulunursa*(ki bunu ilerleyen kısımlarda göreceğiz)*, ilgili örnek depolama alanından kaldırılır.

Dilerseniz bu kadar konuşmayı bir kenara bırakalım ve adım adım dayanıklı, sapa sağlam bir WCF Servis örneği nasıl yazılabilir öğrenmeye ve en önemliside anlayama çalışalım. İlk olarak veritabanı tarafındaki gerekli hazırlıkları yapmamız gerekiyor. Bir başka deyişle depolama tablolarını, ve **CRUD(CreateReadUpdateDelete)** işlemleri için gerekli **StoredProcedure**' lerin oluşturulması lazım. Bu amaçla varsayılan olarak Windows XP tabanlı bir sistemde varsayılan olarak **C:\WINDOWS\Microsoft.NET\Framework\v3.5\SQL\EN** adresinde bulunan**SqlPersistenceProviderSchema** ve **SqlPersistenceProviderLogic** isimli **SQL** bet iklerini sırasıyla çalıştırıyoruz. Hemen belirtelim, ilk olarak şemaları oluşturan**SqlPersistenceProviderSchema** betiğinin çalıştırılması gerekiyor.



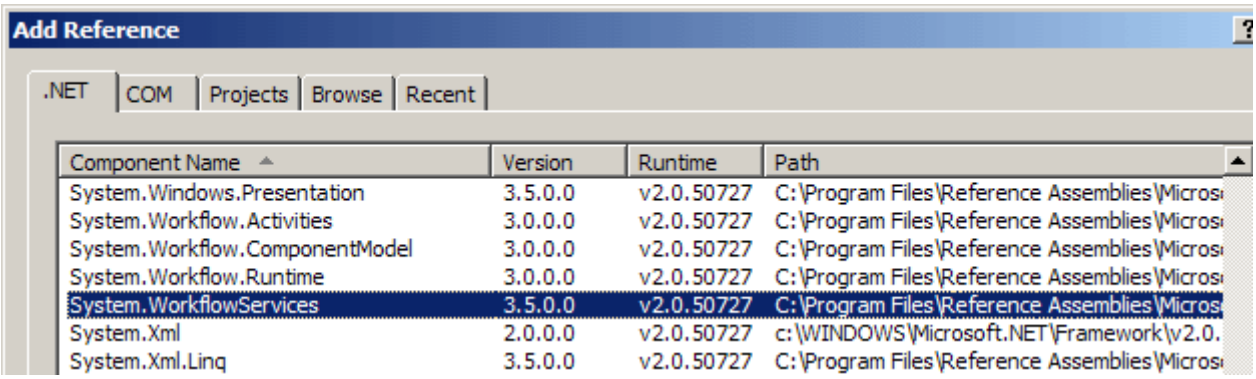
Yine önemli bir noktada şu. Aman **Master** veritabanı altında çalıştırmayın :) Depolama alanının veritabanı adını istediğiniz gibi belirleyebilirsiniz. örnekte söz konusu betikler, **SQLServer 2008** üzerinde*(ki 2005 üzerindede yapabilirsiniz)* **WCFPersistenceStore** isimli veritabanı altında çalıştırılmaktadır. Sonuç olarak aşağıdaki şekilde görülen nesneler oluşur.



Görüldüğü gibi servis örneklerini tutacak olan **InstanceData** isimli basit bir tablo ve **CRUD** operasyonları ile kilit açma işlemleri için birer **saklı yordam** oluşturulmuştur. Tablo alanları şöyle bir gözden geçirildiğinde **uniquidentifier** tipinden bir **id** alanının olduğunu görüyoruz ki bu aslında istemci tarafı içinde önem arz eden **instanceId** değeri olacaktır. öyleki istemcinin uygulamayı kapatıp tekrar açtıktan sonra daha önceden var olan bir **instance**' a ait servis ve verisine erişebilmesi istenebilir. Bu durumu bir sonraki makalemizde ele almaya çalışacağız. Bunlara ek olarak **instanceXML** isimli **XML** veri tipindeki alanında bizim için şu aşamada dikkate değer olduğunu söyleyebiliriz. Nitekim servis örneğinin veri içeriği bu alanda serileştirilmiş olarak tutulacaktır. Dolayısıyla servisin serileştirilebilir olmasının gerektiği ortaya çıkmaktadır. *(Bu durum göz önüne alındığında aklıma hep web tarafında Session' ların veritabanında tutulması gelmektedir. Nitekim bu durumdada **Session** içeriğinin veritabanındaki küçük bir alana olduğu gibi aktarılabilmesi için serileştirilebilir olması şarttır :))*

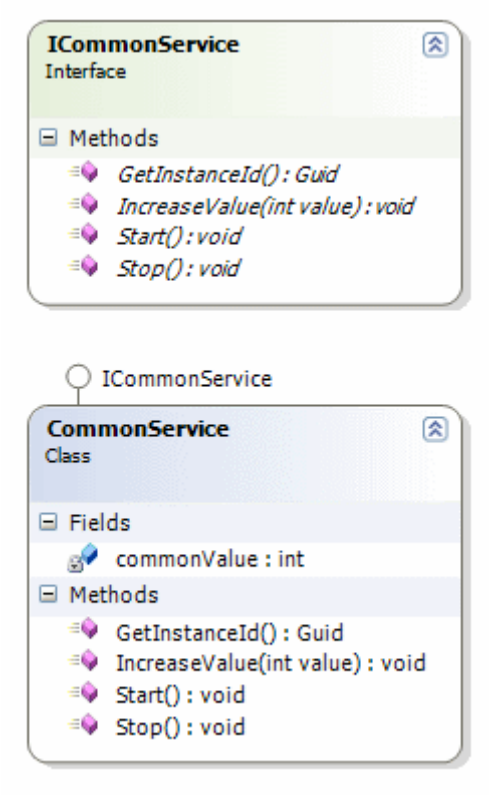
Veritabanı tarafındaki hazırlıklarıda bu şekilde tamamlandıktan sonra artık örnek bir servisin yazılmasına başlayabiliriz. örnek WCF servisimiz **.Net Framework 3.5** tabanlı bir **WCF Service Library**' dir ve kütüphane için belkide en önemli

nokta **System.WorkflowServices** (.Net Framework 3.5) **assembly**' ını referans edilmesi gerekliliğidir.



Bu nedenle sınıf kütüphanesine söz konusu **assembly**' ı ekleyerek yolumuza devam edebiliriz. Servis kütüphanesi içerisinde yer alan tiplerin kod içerikleri, sınıf diagramı görüntüsü ve app.config içeriği ise aşağıda olduğu gibidir.

Sınıf diagramı;



ICommonService isimli servis sözleşmesi içeriği;

```
using System;
using System.ServiceModel;

namespace ServiceLib
```

```
{
   [ServiceContract(
      Name="ServiceCommon"
      ,Namespace="http://www.bsenyurt.com/CommonService")]
   interface ICommonService
   {
      [OperationContract]
      void Start();

      [OperationContract]
      void IncreaseValue(int value);

      [OperationContract]
      Guid GetInstanceId();

      [OperationContract]
      void Stop();
   }
}
```

Görüldüğü üzere servis sözleşmesi içerisinde sembolik olarak 4 farklı operasyon tanımı bulunmaktadır. Şu anda amacımız sadece dayanıklı bir WCF servisi yazmak olduğundan bize çokda anlamlı gelmeyen operasyonlarımızın mevcut olduğunu söyleyebiliriz. Operasyonların uyarlamasının yapıldığı **CommonService** isimli sınıf içeriği ise aşağıdaki gibidir.

CommonService sınıfının içeriği;

```
using System;
using System.ServiceModel.Description;

namespace ServiceLib
{
   [Serializable]
   [DurableService]
   class CommonService
      :ICommonService
   {
      int commonValue;

      #region ICommonService Members

      [DurableOperation(CanCreateInstance=true)]
      public void Start()
      {
```

```
        commonValue = 1;
    }

    [DurableOperation()]
    public void IncreaseValue(int value)
    {
        commonValue += value;
    }

    [DurableOperation()]
    public Guid GetInstanceId()
    {
        return System.ServiceModel.Dispatcher.DurableOperationContext.InstanceId;
    }

    [DurableOperation(CompletesInstance=true)]
    public void Stop()
    {
    }

    #endregion
  }
}
```

Aslında burada daha önceki servis geliştirmelerimizden farklı olarak dikkate değer bir kaç nokta bulunmaktadır. Herşeyden önce sınıfın **Serializable** ve **DurableService** nitelikleri ile imzalandığını görüyoruz. Serileştirilebilir olması bir gereklilik. Nitekim servis tipinin çalışma zamanı
örneğinin, **InstanceData** tablosundaki **instanceXML** veya **instance** alanlarına serileşmesi söz konusudur. Hangi alana serileşeceği bilgisi ise konfigurasyon dosyasında belirtilebilir. Yani servis nesne örneğinin veri içeriğinin **XML** olarak yada **binary** formatta tutulması sağlanabilir.

Dayanıklı bir servis tanımlamasında en önemli
noktalar **DurableService** ve **DurableOperation** nitelikleridir. örneğin servis operasyonlarında olan **Start** metodu çalıştığında **SQL**üzerinde **CommonService** örneği için bir satır oluşturulacaktır. Diğer taraftan bu satırın devamlılığı herhangibir istisnai durum oluşmassa **Stop** operasyonuna yapılan çağrı ile son bulacaktır.
Nitekim **Stop** operasyonundaki **DurableOperation** niteliğinde **CompleteInstance** özelliği ne **true** değeri verilmiştir. Dayanıklı servisler tasarlanırken oturum bazlı çalışma önemlidir. Bu nedenle aşağıdaki koşullara dikkat edilmesi gerekmektedir.

- Dayanıklı WCF servislerinde **içerik bazlı bağlayıcı tipler(Content Based Binding Types)** kullanılmalıdır.

örneğin **WSHttpContextBinding**, **BasicHttpContextBinding** veya**NetTcpContext Binding** gibi.

- **InstanceContextMode** özelliğinin değeri **PerSession** olmalıdır ki varsayılan olarak örneğimizde böyledir.
- **ConcurrencyMode** özelliğinin değeri **Multiple** olmamalıdır.
- Eğer sözleşmede sessionlar desteklenmiyorsa tüm operasyonlara ait **DurableOperation** niteliklerinin **CanCreateInstance** özelliklerine **true** değeri atanmalıdır ve eğer böyle bir durum söz konusu ise **IsOneWay true** olarak set edilmemelidir.
- Eğer **DurableService** niteliğinin **SaveStateInOperationTransaction** değeri **true** ol arak belirlenirse, tüm operasyonların **OperationBehavior** niteliği ile imzalanması ve**TransactionScopeRequired** özelliklerinede **true** değeri atanması gerekmektedir. Yada, **TransactionFlowOption** özelliğinin değerinin **Mandatory** olması gerekir. Tüm bunlara ilavetende**ServiceBehavior** niteliğinin **ConcurrencyMode** özelliğinin değerinin **Single** olarak belirlenmesi gerekmektedir.

Görüldüğü gibi oldukça kafa karıştırıcı bir şartname dizisi ile karşı karşıyayız. Ancak gerçek hayat vakalarında bu durumlar altın değerinde önem taşımaktadır. Gelelim servis tarafındaki konfigurasyon dosyası içeriğine.

Servis tarafındaki App.config içeriği;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <connectionStrings>
    <add name="WCFPerstConStr" connectionString="data
source=.;database=WCFPersistenceStore;integrated security=SSPI"/>
  </connectionStrings>
  <system.web>
    <compilation debug="true" />
  </system.web>
  <system.serviceModel>
    <services>
      <service name="ServiceLib.CommonService"
behaviorConfiguration="ServiceLib.CommonServiceBehavior">
        <host>
          <baseAddresses>
            <add baseAddress =
"http://localhost:8731/Design_Time_Addresses/ServiceLib/CommonService/" />
          </baseAddresses>
        </host>
        <endpoint address ="" binding="wsHttpContextBinding"
contract="ServiceLib.ICommonService">
          <identity>
            <dns value="localhost"/>
```

```
            </identity>
          </endpoint>
          <endpoint address="mex" binding="mexHttpBinding"
contract="IMetadataExchange"/>
        </service>
      </services>
      <behaviors>
        <serviceBehaviors>
          <behavior name="ServiceLib.CommonServiceBehavior">
            <serviceMetadata httpGetEnabled="True"/>
            <serviceDebug includeExceptionDetailInFaults="False" />
              <persistenceProvider
                type="System.ServiceModel.Persistence.SqlPersistenceProviderFacto
ry, System.WorkflowServices, Version=3.5.0.0, Culture=neutral,
PublicKeyToken=31bf3856ad364e35"
                connectionStringName="WCFPerstConStr"
                persistenceOperationTimeout="00:00:10"
                lockTimeout="00:01:00"
                serializeAsText="true"/>
          </behavior>
        </serviceBehaviors>
      </behaviors>
    </system.serviceModel>
</configuration>
```

Bu şekilde bakıldığında kimimize korkutucu gelebilen konfigurasyon içeriğindeki ayarların çoğunu **Microsoft Service Configuration Editor** aracılığıyla yapabileceğimizi unutmayalım. Konfigurasyon dosyasındanda görüldüğü üzere **bağlayıcı tip(Binding Type)** olarak içerik tabanlı **wsHttpContextBinding** örneği kullanılmaktadır. Daha öncedende belirttiğimiz
gibi **NetTcpContextBinding** veya **BasicHttpContextBinding** bağlayıcı tipleride ele alınabilir.

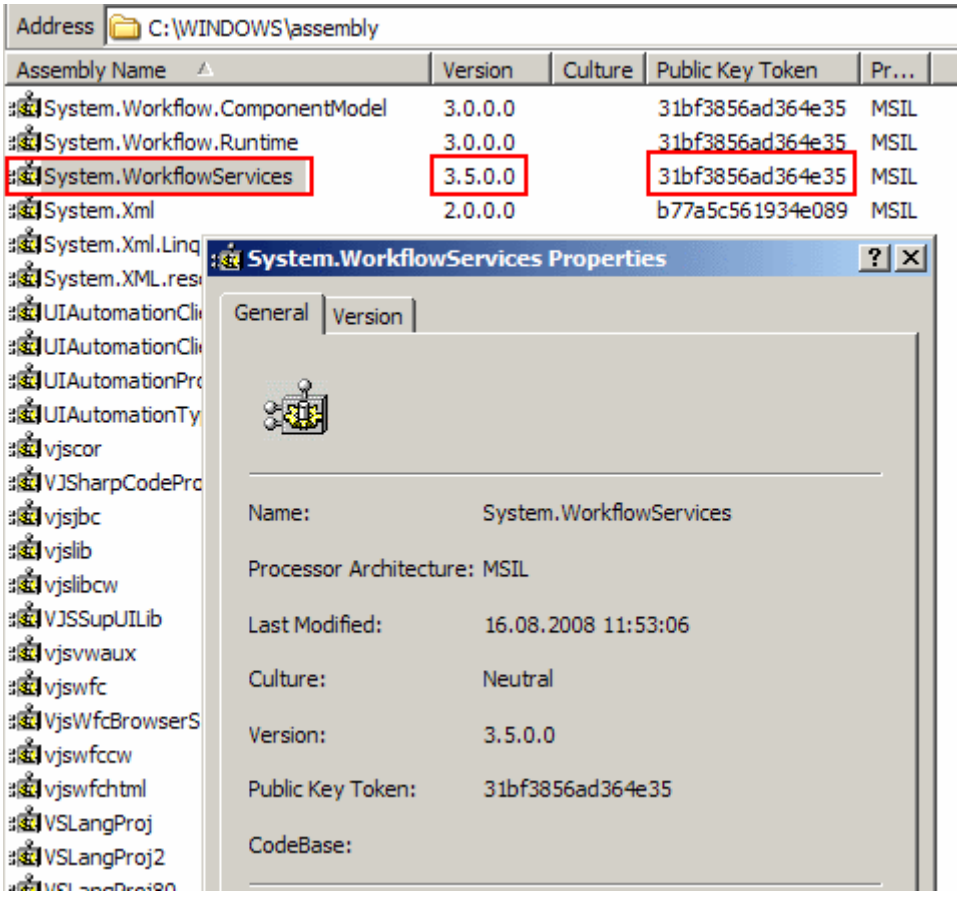
önemli noktalardan ilki **connectionString** elementi değeridir. Hatırlayacağınız gibi örneğimizdeki servis verilerini kalıcı olarak **WCFPersistenceStore** isimli örnek bir veritabanı üzerinde tutmak amacıyla bir takım ön hazırlıklar yapmıştık. Bu bilgi **SqlPersistenceProviderFactory** tipi için önemlidir. Nitekim provider' ın hangi bağlantıyı kullanarak işlemler yapacağını bilmesi gerekmektedir. Bu sebepten dolayıda **persistenceProvider** isimli servis davranışı içerisinde bir **connectionStringName** niteliği bulunmaktadır.

İkinci önemli nokta elbetteki **persistenceProvider** isimli servis davranışı ve içeriğidir. Burada dayanıklı depolama alanı için gerekli pek çok ayar yapılmaktadır. örneğin servis verisinin**text** tabanlı olarak serileştirilip serileştirilmeyeceği **serializeAsText** değeri ile belirtilir. Kilitleme ve operasyonlar için zaman aşımı

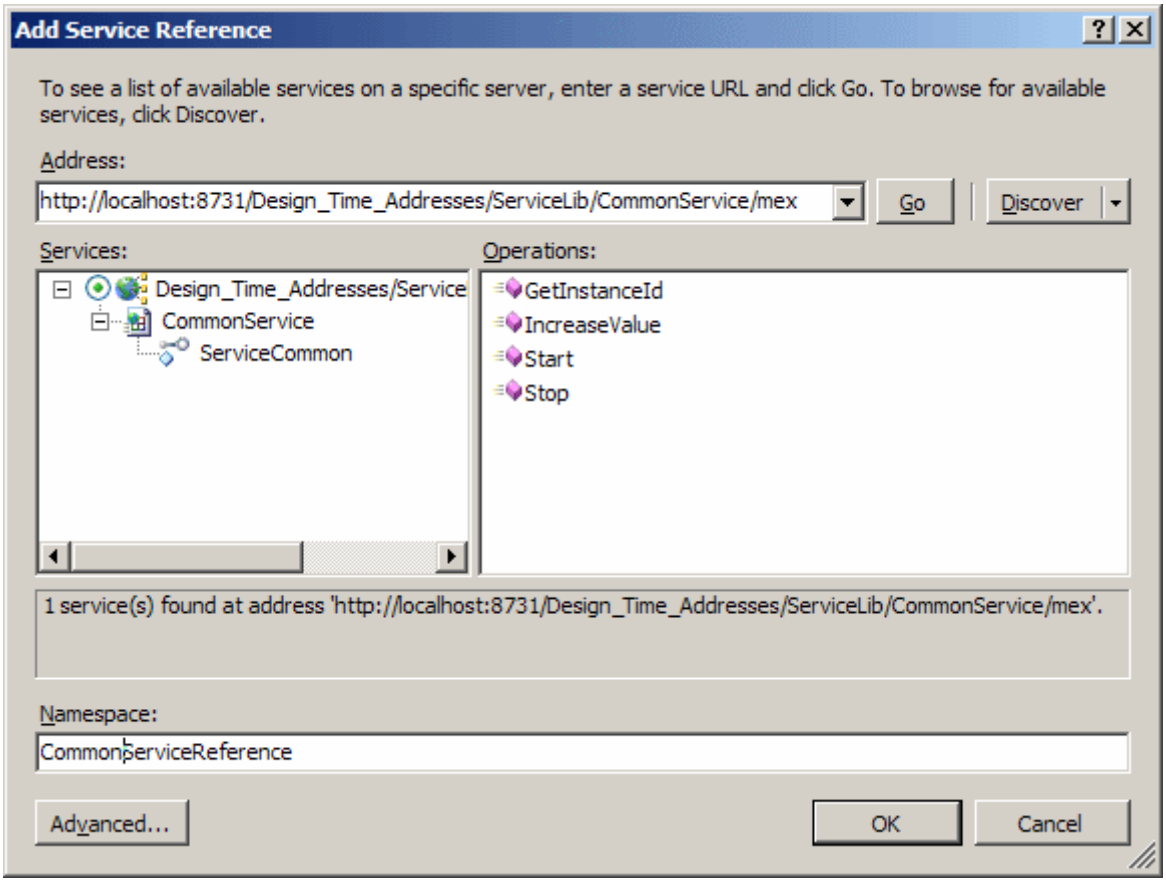
süreleri **lockTimeout** ve **persistenceOperationTimeout**niteliklerine atanan değerler ile belirtilir.

üçüncü önemli nokta ise, **persistenceProvider** elementi içerisinde yer alan **type** niteliğine **SqlPersistenceProviderFactory** değerinin atanmasıdır. Bu varsayılan olarak ve sürekli vurguladığımız üzere **SQL** üzerinde depolama yapılacağını belirtmektedir. type niteliğindeki bu tanımlama sırasında tipin **qualified name** değerinin tam olarak verilmesi gerekir *(**TipAdı, AssemblyAdı, Assembly Versiyonu,Assembly Culture bilgisi, Assembly PublicKeyToken değeri**)*.

***NOT : PublicKeyToken** değerini kolay bir şekilde **Global Assembly Cache(GAC)** içerisinden de bulabiliriz. Bunun için aşağıdaki ekran görüntüsünde olduğu gibi **Assembly**klasörüne gitmemiz yeterlidir.*



Servis tarafında yaptığımız bu hazırlıkların ardından artık istemci tarafını da geliştirmeye başlayabiliriz. Dayanıklı bir WCF servisini test ederken en basit haliyle bir **ConsoleApplication** bizim için biçilmiş kaftan olacaktır. Console uygulamamızı servis ile aynı **Solution** üzerinde geliştirdiğimizden, servise referansını eklememizde son derece kolay olacaktır. Bu amaçla **Console** uygulamasında **Add Service Reference** seçeneğini kullandıktan sonra aşağıdaki ekran görüntüsünde olduğu gibi WCF servisine ulaşılabilinir.

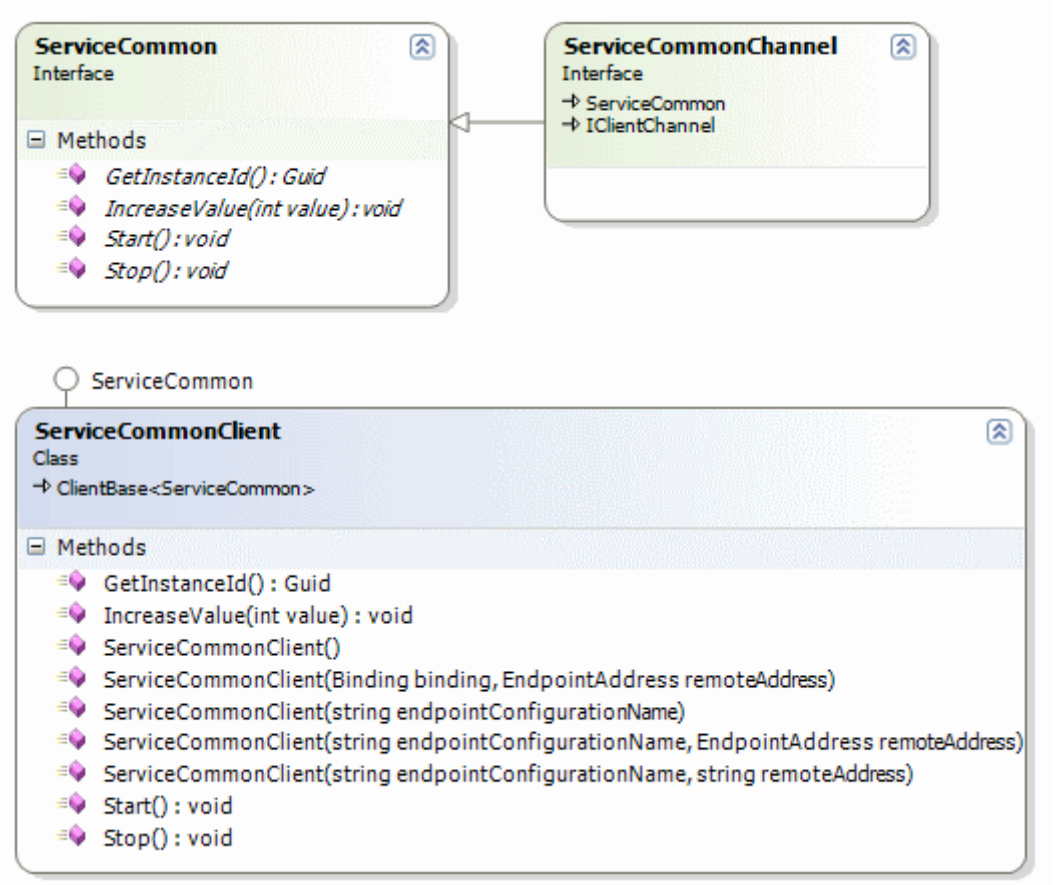


Servis referansını istemci tarafına ekledikten sonra otomatik olarak üretilen konfigurasyon dosyası içeriği aşağıdaki gibi olacaktır.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <bindings>
         <wsHttpContextBinding>
            <binding name="WSHttpContextBinding_ServiceCommon"
closeTimeout="00:01:00"
               openTimeout="00:01:00" receiveTimeout="00:10:00"
sendTimeout="00:01:00"
               bypassProxyOnLocal="false" transactionFlow="false"
hostNameComparisonMode="StrongWildcard"
               maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
               messageEncoding="Text" textEncoding="utf-8" useDefaultWebProxy="true"
               allowCookies="false" contextProtectionLevel="Sign">
               <readerQuotas maxDepth="32" maxStringContentLength="8192"
maxArrayLength="16384" maxBytesPerRead="4096" maxNameTableCharCount="16384"
/>
               <reliableSession ordered="true" inactivityTimeout="00:10:00"
enabled="false" />
               <security mode="Message">
```

```
                <transport clientCredentialType="Windows" proxyCredentialType="None"
realm="" />
                <message clientCredentialType="Windows"
negotiateServiceCredential="true" algorithmSuite="Default"
establishSecurityContext="true" />
              </security>
            </binding>
          </wsHttpContextBinding>
        </bindings>
        <client>
          <endpoint
address="http://localhost:8731/Design_Time_Addresses/ServiceLib/CommonService/" bin
ding="wsHttpContextBinding" bindingConfiguration="WSHttpContextBinding_Service
Common" contract="CommonServiceReference.ServiceCommon"
name="WSHttpContextBinding_ServiceCommon">
            <identity>
              <dns value="localhost" />
            </identity>
          </endpoint>
        </client>
    </system.serviceModel>
</configuration>
```

Dikkat edileceği üzere istemci tarafında oluşturulan **EndPoint** içerisinde de aynen servis tarafındaki konfigurasyon dosyasında olduğu gibi **wsHttpContextBinding** bağlayıcı tipi kullanılmaktadır. Sınıf diagramındada izlenebileceği gibi servis referansının eklenmesi ile birlikte proxy üretimi için gerekli tiplerde otomatik olarak istemci tarafına eklenmektedir.

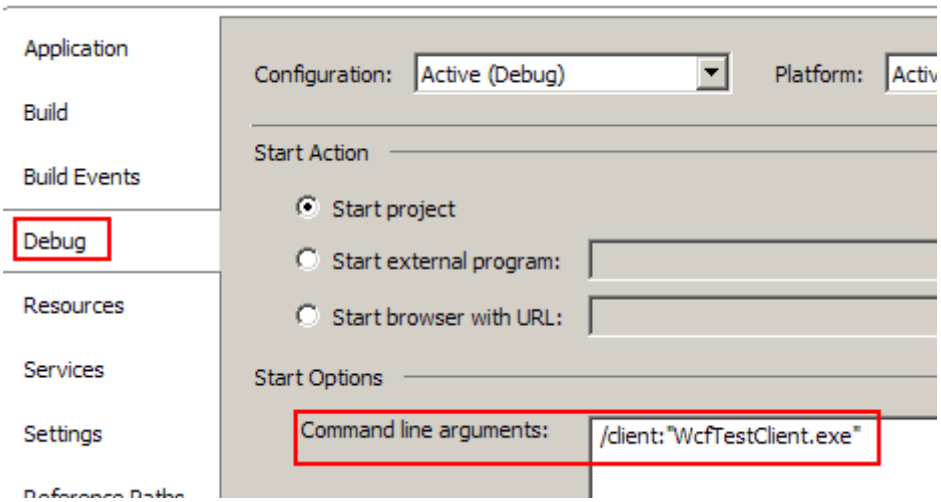


Program kodlarımızı ise test amaçlı olarak aşağıdaki gibi geliştirebiliriz.

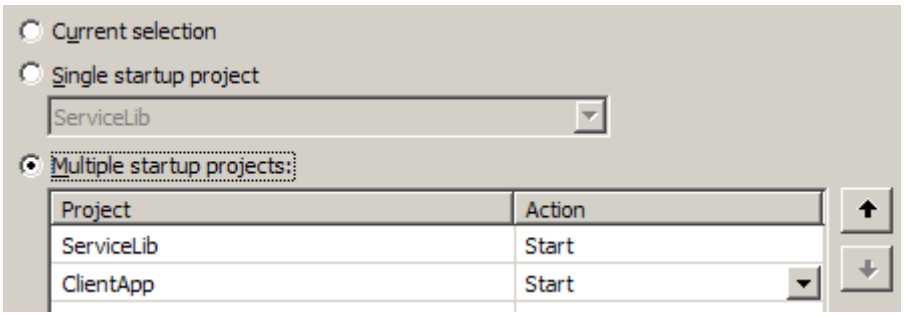
```
using System;
using ClientApp.CommonServiceReference;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         ServiceCommonClient client = new
ServiceCommonClient("WSHttpContextBinding_ServiceCommon");

         client.Start();
         Console.WriteLine("Service başlatıldı");
         client.IncreaseValue(10);
         Guid instanceId=client.GetInstanceId();
         Console.WriteLine("Instance değeri {0}",instanceId.ToString());
         client.Stop();
         Console.WriteLine("Uygulamadan çıkmak için bir tuşa basınız");
         Console.ReadLine();
```

```
        }
    }
}
```

İlk olarak proxy sınıfına ait nesne örneği oluşturulmaktadır. Hemen ardından **Start**, **IncreaseValue**, **GetInstanceId** ve **Stop** servis metodları sırasıyla çağırılır. Hatırlanacağı üzere **Start** metodu ile **Servis** örneğinin tabloya atılması, **Stop** metodundan sonra ise tablodan servise ait ilgili satırın silinmesi gerekmektedir. Bu ilk vakayı analiz etmek için servis kütüphanesi ve istemci uygulamayı aynı anda çalıştırmalıyız. Bu nedenle iki ön hazırlık yapılmalıdır. Bildiğiniz gibi **Visual Studio 2008** ile birlikte **WCF Servis**kütüphanelerini başlatılabilir ve built-in gelen servis, istemci uygulamalarını kullanarak testler yapılabilir. Bu örneğimizde istemci tarafını kendimiz geliştirdiğimiz için servis kütüphanesi özelliklerinden **WcfTestClient.exe**' nin çalıştırıldığı komut satırı parametresini kaldırmamız gerekmektedir.



Bu işlemin ardından ise **Solution** özelliklerine gidip önce sınıf kütüphanesinin sonrasında ise istemci uygulamanın çalıştırılması gerektiğini belirtmeliyiz.



Böylece servis kütüphanesine ait test sunucusu önce çalışacak sonrasında ise istemci uygulamamız yürütülecektir ki bu hazırlık özellikle kodu debug etmemizi kolaylaştıracaktır. İlk test için istemci tarafındaki **Start** metodu çağrısına bir **breakpoint** koymamız gerekmektedir.

```
ClientApp.Program
1  using System;
2  using ClientApp.CommonServiceReference;
3
4  namespace ClientApp
5  {
6      class Program
7      {
8          static void Main(string[] args)
9          {
10             ServiceCommonClient client =
11                 new ServiceCommonClient("WSHttpContextBinding_ServiceCommon");
12
13             client.Start();
14             Console.WriteLine("Service başlatıldı");
15             client.IncreaseValue(10);
16             Guid instanceId=client.GetInstanceId();
17             Console.WriteLine("Instance değeri {0}",instanceId.ToString());
18             client.Stop();
19             Console.WriteLine("Uygulamadan çıkmak için bir tuşa basınız");
20             Console.ReadLine();
21         }
22     }
23 }
```

Şimdi uygulamayı çalıştırıp **Start** metodunu **Step Over** ile geçtiğimizde **SQL** sunucusu üzerindeki **instanceData** tablosunda yeni bir satır oluşturulduğunu görebiliriz.

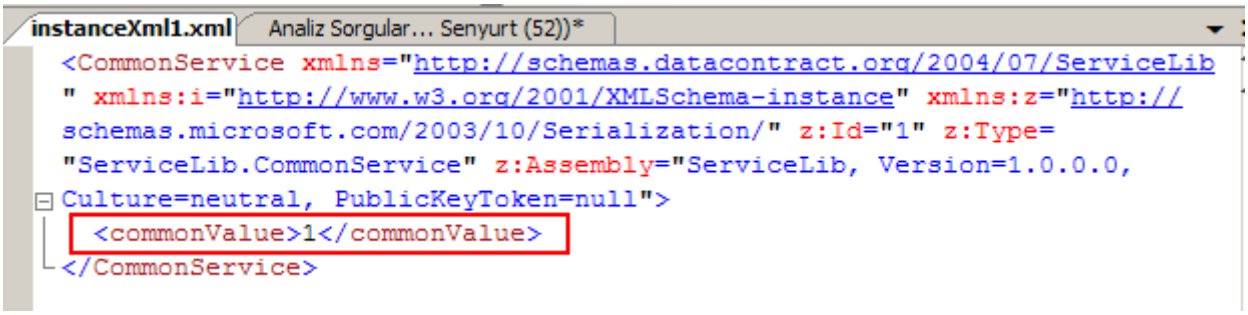
```
Analiz Sorgular... Senyurt (52))*
Use WCFPersistenceStore
Go

Select
    id
    ,instance
    ,instanceXml
    ,created
    ,lastUpdated
    ,lockOwner
    ,lockExpiration
From InstanceData
```

Results | Messages

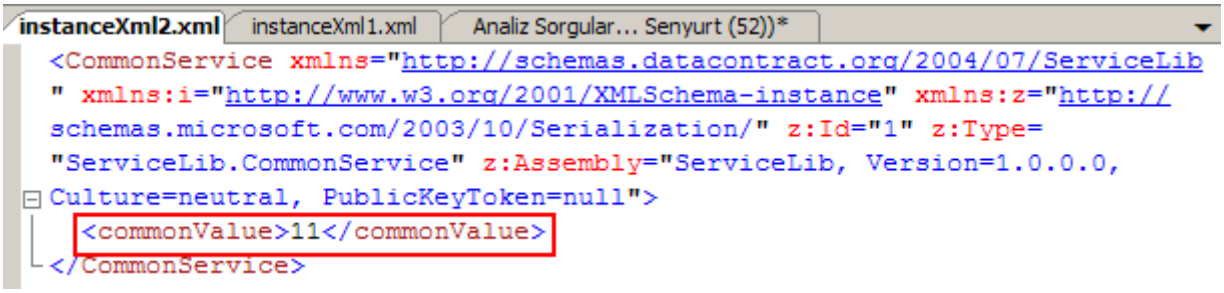
	id	instance	instanceXml
1	4E022418-DD27-4A96-9FB4-718A28B98E8E	NULL	<CommonService xmlns="http://s

Dikkat ederseniz servis örneklendiğinde değil, **DurableOperation** niteliğinde **CanStartCreateInstance** değeri **true** olan **Start** metodu çağrısından sonra bu satır eklenmiştir. Satırın **id**değeri, servis çağrısı için üretilen **GUID** değeridir ve aslında bu değer içerik ile birlikte istemciyede gönderilmektedir. Bu nedenle oturum içerisinde yapılacak olan diğer operasyon çağrılarında bu GUID numarası kullanılır. **instanceXml** isimli alanın içeriğine

bakıldığında *(özellikle **IncreaseValue** metodundan önce)* servisin aşağıdaki içeriğe sahip olduğu görülebilir.



CommonService içerisinde tanımlanmış olan **commonValue** alanının ilk atanan değeri burada açık bir şekilde görülmektedir. Elbetteki serileşen tipin içerisinde kaç tane alan(field) varsa, bunların anlık olarak değerlerinin tamamı serileştirilen bu içeriğe atanmaktadır. Eğer kodda **F10(Step Over)** ile ilerlenmeye devam edilirse **IncreaseValue** metodu geçildikten sonra **XML**içeriğinin aşağıdaki hale geldiği görülür.



Bir başka deyişle servis nesne örneğinin depolama alanında tutulan içeriğindeki alanda da güncelleme yapılmıştır. Kod bu şekilde sonuna kadar devam ettirildiğinde ve **Stop** metodu çağrısıda **F10** ile geçildiğinde artık **CommonService**' in şu anki örneği için açılan satırın artık olmadığı açık bir şekilde gözlemlenebilir. Ki burada Stop metodundaki**DurableOperation** niteliğinde **CompleteInstance** özelliğine **true** değerini atadığımızı hatırlamamızda yarar vardır. Bu basit işleyişi tekrar edip **SQL Server Profiler** aracı yardımıyla arka plana bakıldığında ise aşağıdaki gibi bir sorgu dizisinin çalıştırıldığı gözlemlenmektedir.

	InsertInstance SP' si için çağrı
Start Metodu çağrısında	declare @p3 xml set @p3=convert(xml,N'<CommonService xmlns="http://schemas.da instance" xmlns:z="http://schemas.microsoft.com/2003/10/Serializati Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"><common declare @p7 int set @p7=0 **exec InsertInstance** @id='**1B9F9AA9-4F5C-46F1-97ED-C520716** @instanceXml=@p3,@unlockInstance=1,@hostId='97CE292D-D46

	select @p7
IncreaseValue Metodu çağrısında	**LoadInstance SP' si için çağrı**
	declare @p5 int set @p5=0 **exec LoadInstance** @id='**1B9F9AA9-4F5C-46F1-97ED-C520716C** @lockTimeout=60,@result=@p5 output select @p5
	UpdateInstance SP' si için çağrı
	declare @p3 xml set @p3=convert(xml,N'<CommonService xmlns="http://schemas.dat instance" xmlns:z="http://schemas.microsoft.com/2003/10/Serializati Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"><commonV declare @p7 int set @p7=0 **exec UpdateInstance** @id='**1B9F9AA9-4F5C-46F1-97ED-C52071** @instanceXml=@p3,@unlockInstance=1,@hostId='97CE292D-D465 select @p7
GetInstanceId Metodu çağrısında	**LoadInstance SP'si için çağrı**
	declare @p5 int set @p5=0 **exec LoadInstance** @id='**1B9F9AA9-4F5C-46F1-97ED-C520716C** 4A55BCE2D7B3',@lockTimeout=60,@result=@p5 output select @p5
	UpdateInstance SP'si için çağrı
	declare @p3 xml set @p3=convert(xml,N'<CommonService xmlns="http://schemas.dat instance" xmlns:z="http://schemas.microsoft.com/2003/10/Serializati Version=1.0.0.0, Culture=neutral, PublicKeyToken=null"><commonV declare @p7 int set @p7=0 **exec UpdateInstance** @id='**1B9F9AA9-4F5C-46F1-97ED-C52071** @instanceXml=@p3,@unlockInstance=1,@hostId='97CE292D-D465 select @p7
Stop Metodu çağrısında	**LoadInstance SP' si için çağrı**
	declare @p5 int set @p5=0 **exec LoadInstance** @id='**1B9F9AA9-4F5C-46F1-97ED-C520716C** 4A55BCE2D7B3', @lockTimeout=60,@result=@p5 output select @p5

	DeleteInstance SP' si için çağrı
	declare @p4 int set @p4=0 **exec DeleteInstance** @id='**1B9F9AA9-4F5C-46F1-97ED-C520716C** 4A55BCE2D7B3',@lockTimeout=60,@result=@p4 output select @p4

Biraz karmaşık ve gereksiz gibi görülebilir ama bu sorgular içerisinde çağırılan saklı yordamlara ve atamalara dikkat etmenizi, ayrıca incelemenizi şiddetle tavsiye ederim. İlk testimiz başarılı bir şekilde çalıştı. Depolama alanına WCF servis örneğimizin başarılı bir şekilde eklendiği, operasyon çağrıları sırasında güncellendiğini ve son olarakta silindiğini gördük. Ancak başka vakalarda söz konusudur. örneğin istemci uygulama süreci başlattıktan sonra herhangibir sebeple sonlanırsa, servis tarafında başlattığı **instance**' ın içeriğine tekrar nasıl ulaşabilir? Bir başka deyişle, daha önceden başlattığı servisin verilerine nasıl ulaşabilir? Diğer taraftan, servis tarafında **SQL** tabanlı bir depolama alanı kullanılmaktadır. Peki ya özel bir depolama yapmak istersek. Yani SQL dışından bir ortam kullanmak istersek. Söz gelimi dosya tabanlı bir sistem kullanılabilir mi? Yada örneğin bir **Access** tablosu bu iş için göz önüne alınabilir mi? Bu durumda nasıl ayarlamalar yapılması gerekmektedir? Kalıcı servislerde transaction' lar söz konusu olabilir mi, eğer olursa süreçler nasıl kontrol altına alınabilir? İşte bu ve benzer sorularımız cevabını ilerleyen makalelerimizde bulmaya çalışıyor olacağız. Şimdilik hevesimizi burada dayanıklı olarak saklı tutuyor ve bir sonraki makalemizde görüşmek üzere diyoruz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

DurableService.rar (50,92 kb)

Ado.Net Senkronizasyon Servisleri(Sync Services for Ado.Net) (2009-01-03T22:19:00)

ado.net sync services,microsoft sync framework,

Birbirleri ile sürekli bağlantı
halinde olamayan **istemci/sunucu(Client/Server)** mimarilerinde en büyük problemlerden biriside verilerin karşılıklı veya tek taraflı olaraktan senkronize edilmeleridir. çoğu büyük çaplı saha uygulamasında, sunucu tarafındaki veri kaynaklarının istemcide kullanıldığı durumlar söz konusudur. Bu noktada istemcilerin sürekli bağlı kalamadıkları bir ortamın var olması olasıdır **(Occasionally Connected Enivronments)**. Nitekim istemci ve sunucu arasında kablosuz bağlantı olma ihtimali oldukça yüksektir. Nitekim günümüz teknolojileri düşünüldüğünde istemci uygulamaların bir çoğu mobil cihazlar ile, diz üstü bilgisayarlar üzerinde koşmaktadır. Bu tabiki daha çok saha elemanlarının işin içerisine girdiği senaryolardır.

Söz gelimi günlük ürün fiyatlandırmalarını kullanarak satış yapan personelin mobil cihazlara sahip olduğu bir durumda, istemcilerin sunucuya sürekli olarak bağlı kalmaları

saha üzerinde son derece zor olabilir. *(Mobil cihazlar, modern cep telefonları ve PDA' ler olabileceği gibi dizüstü bilgisayarlarda olabilir.)* Buna rağmen istemcinin söz konusu veriyi kullanarak çalışabilmesi de istenebilir. Diğer taraftan istemcinin sunucuya bağlandığı hallerde ortak veri üzerindeki değişikliklerini göndermesi ve hatta var olan farklılıkları kendi sistemine çekmeside istenebilir. Bu iki taraflı bir senkronizasyon anlamına gelmektedir ki, iki tarafında birbirleri üzerindeki verileri senkronize etmeleri sırasında pek çok güçlükle karşılaşılacaktır. Nitekim bu vakada eş zamanlı bağlı olan kullanıcıların ortak verilerde yapacağı değişikliklerin ele alınması gerekir ki bunlar çakışmalara**(Conflicts)** neden olmaktadır. Bazı durumlarda ise sadece sunucudaki farklılıkların istemciye aktarılması veya tam tersi söz konusudur. Bu durumlarda senkronizasyonu yönetmek nispeten biraz daha kolaydır. Ancak hangi vaka olursa olsun her iki uçta yer alan verinin kolay kodlanabilir, yönetilebilir bir biçimde senkronize edilmeleri istenir. En azından geliştiriciler için bu önemlidir.

***NOT : Ado.Net senkronizasyon** servisleri esas itibariyle **Microsoft Sync Framework(MSF)** altyapısının bir parçasıdır. MSF altyapısı içerisinde **Sync Services for File Systems** ve **Sync Services for FeedSync** isimli iki alt açılım daha vardır. **Sync Services For Ado.Net** özel olarak istemci ve sunucu arasındaki veri senkronizasyonun sağlanmasında **Ado.Net**' in üstüne gelmiş olan bir tamamlayıcı olarak düşünülebilir. (Microsoft Sync Framework için **MSDN**' den detaylı bilgi alınabilir.)*

Senkronizasyon işlemlerinde bilinen ve kullanılan farklı teknikler de söz konusudur. örneğin **Remote Data Access(RDA)** veya **Merge Replication**. **RDA**, **SQL Server Compact 3.5** ile diğer bir **SQL**veritabanı arasındaki senkronizasyon işlemlerinde ele alınır. **Merge Replication** ise **SQL** veritabanlarının herhangi versiyonları arasındaki senkronizasyon süreçlerinde kullanılır. özellikle**Merge Replication** veritabanı yöneticilerine hitap eder ve **SQL** kaynaklarını hedefler. Ancak **ADO.Net Senkronizasyon Servisleri** ile **WCF(Windows Communication Foundation)** hizmetlerini kullanarak, sunucu tarafında farklı veri kaynaklarına erişebilmek mümkündür. Diğer taraftan **Ado.Net Senkronizasyon Servisleri** daha çok uygulama geliştiricileri hedef alır. Eğer istemci tarafının senkronize edeceği veri kümesi **SQL** dışında bir kaynak ise mutlaka **Ado.Net Senkronizasyon Servisi** göz önüne alınmalıdır. **RDA**, **Merge Replication** ve **Ado.Net Senkronizasypn Servisleri** arasındaki karşılaştırmaları aşağıdaki tablodan da inceleyebilirsiniz. özellikle karar verme aşamasında bu tablodaki bilgilerden de yararlanılabilir.

Anahtar özellik
Servisler üzerinden senkronizasyon sağlanması
Farklı veri kaynakları için destek
Değişimsel(Incremental) farklılıkların takibi

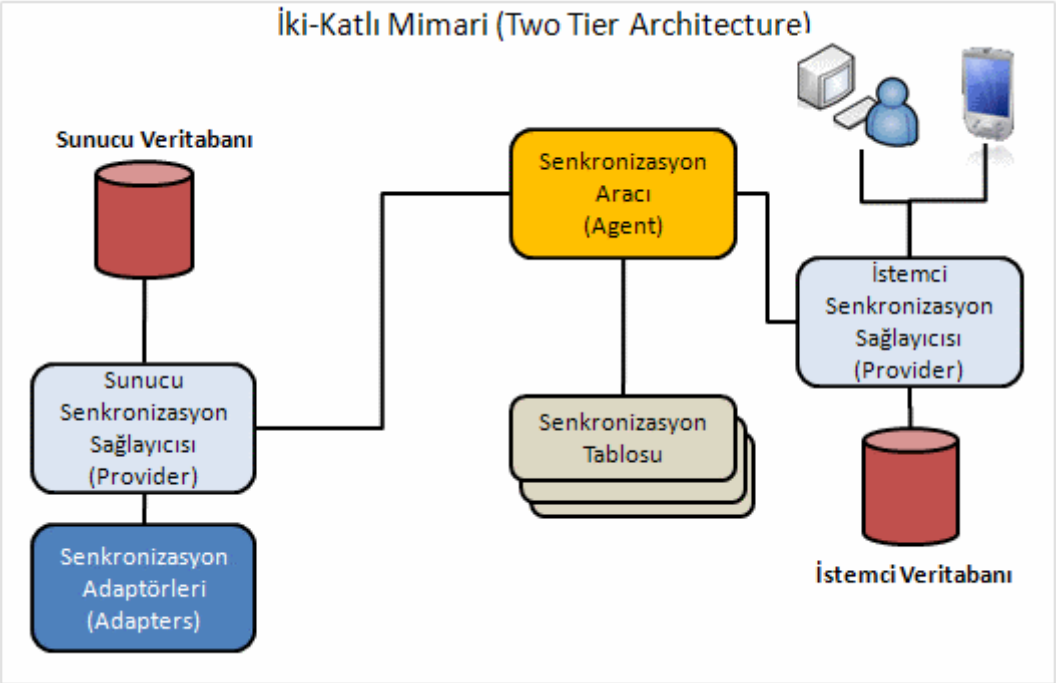
çakışma(Conflict) kontrolü ve çözümleri
İstemci tarafında View' ların kolayca oluşturulması*(Görsel derslerde ele alınacaktır)*
Otomaik şema(Schema) ve veri oluşturma
Büyük boyutlu DataSet desteği
Otomaik olarak şema değişikliklerini üretmek
Veriyi tekradan birleştirmek(Repartition)
***** *RDA değişimsel upload' ları destekler. Verinin istemci tarafına alınmasında Snapshot modelini*

Teknik açıdan bakldığında **Ado.Net Senkronizasyon Servisleri** temel olarak aşağıdaki 3 assembly' dan oluşmaktadır. **Ado.Net Senkronizasyon Servisleri** tarafların sahip olduğu veri sağlayıcılarına göre **2 katlı(Two Tier), N-katlı(N-Tier)** ve **Servis Bazlı Mimariye(Service Oriented Architecture)** uygun olacak şekilde kullanılabilmektedir. Eğer istemci ve sunucu**Ado.Net** veri sağlayıcıları üzerinden konuşuyorlarsa iki katlı veya n katlı modeller tercih edilebilir. Ancak sunucu tarafından **SQL** dışı bir veri kaynağı var ise*(Söz gelimi bir XML deposu, Active Directory vb...)* bu durumda servis yönemlimli olacak şekilde bir geliştirme yapılmalıdır. Senkronizasyon her zaman için istemci tarafında başlatılan bir olaydır ve temel olarak 4 farklı tipte senkronizasyon tekniği kullanılmaktadır.

Kullanılan Teknik	**Açıklama**
Snapshot	Bu teknikte senkronizasyon işlemi başlatıldığında sunucu tarafındaki verinin tam göz önüne alınmaz, sürekli olarak son hal indirilir.
Download-Only	Bir önceki senkronizasyona göre farklı olan verilerin download edilmesi söz kor
Upload-Only	Senkronizasyon işleminde istemci tarafındaki veriler üzerinde yapılan değişiklik ekibinin herhangibir ürün satışı için yaptığı girişler veya güncellemeler buna örn
Bidirectional	çift yönlü senkronizasyon söz konusudur. Söz gelimi bir kargo dağıtım firmasını sunucu üzerine güncellemesi gibi bir vaka örnek olarak düşünülebilir. Bu noktad veri**çakışmalarının(Conflicts)** kontrol altına alınması gerekir.

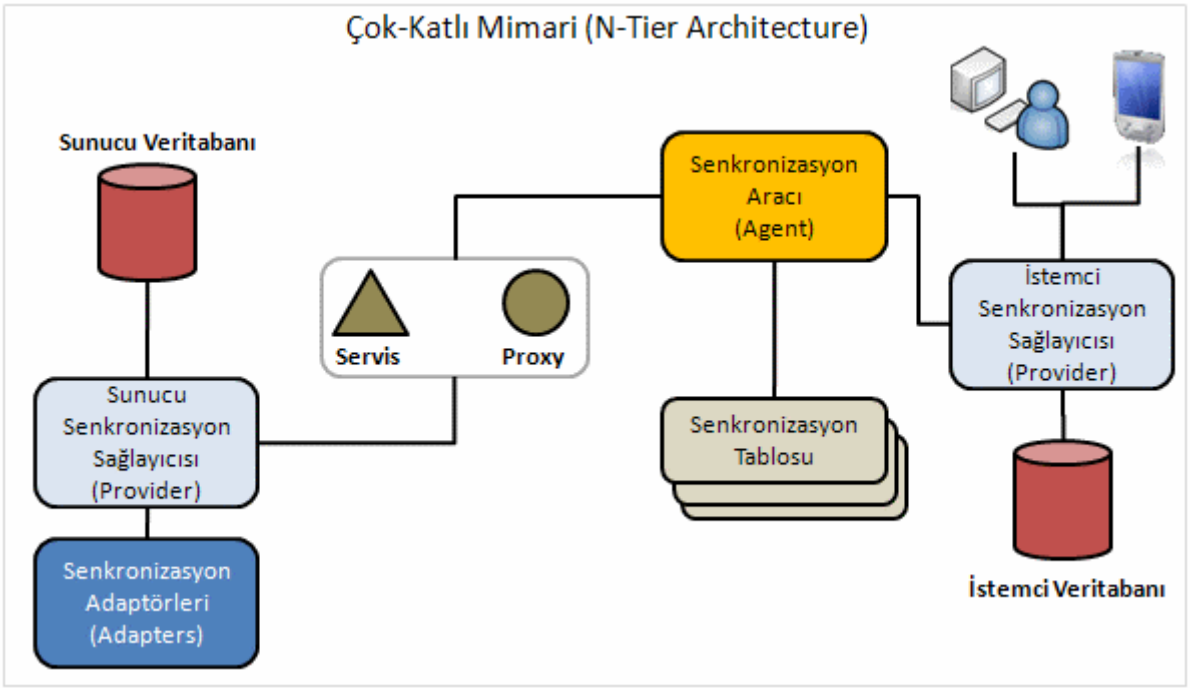
Bu noktada belkide senkronizasyon servislerinin katlı mimarideki konumlarını ele almak yararlı olabilir. Bu amaçla aşağıdaki çizelgelerden yararlanabiliriz.

2 Katlı Mimari;



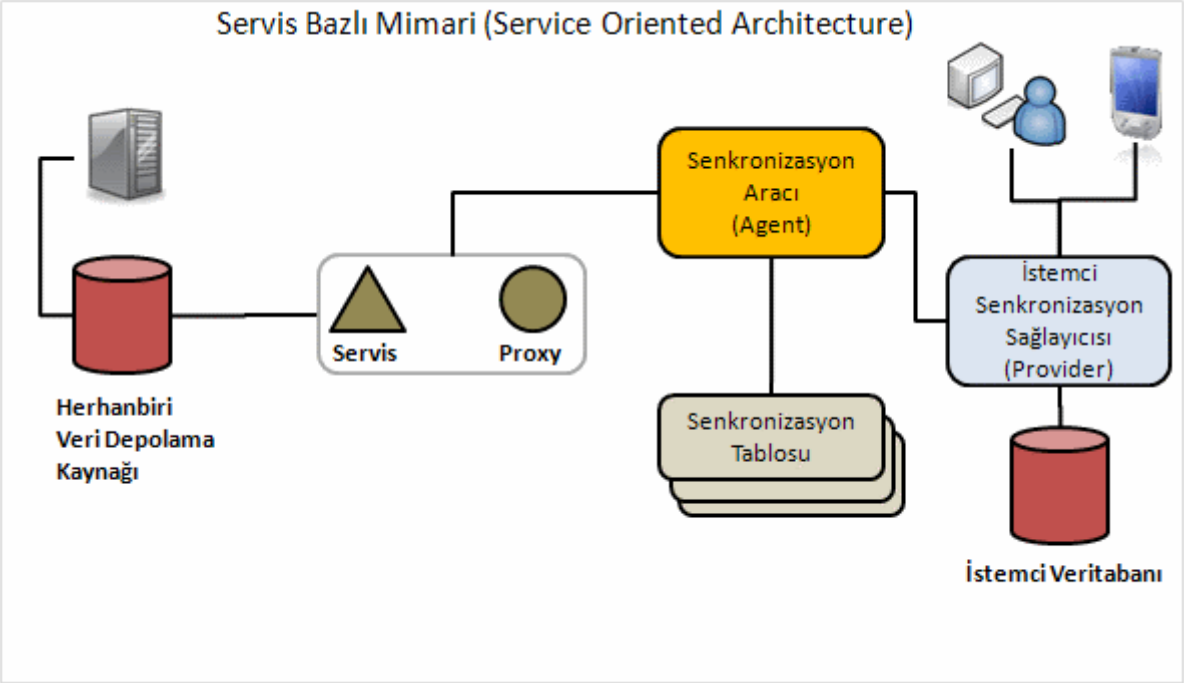
İki katlı modelde sunucu ve istemci tarafında senkronizasyon işlemine tabi olan nesneler bazen aynı uygulama üzerinde bulunmaktadır. Her ne kadar çok fazla tavsiye etmesemde iki katlı modelin uygulanması özellikle **Visual Studio 2008** ortamında son derece kolaydır. Ancak gerçek hayat uygulamalarında çoğunlukla sunucu senkronizasyonunu bir servis veya başka bir uygulama tek başına üstlenir.

N-Katlı Mimari;



N-katlı modelde istemci ve sunucu üzerindeki veritabanları arasındaki iletişim sırasında devreye giren ve ayrı bir katta duran **Servis-Proxy** tipleri mevcuttur. Genellikle istemci ve sunucu veribanları arasında doğrudan bağlantıyı gerektirmediği için 2 katlı mimariye göre daha fazla tercih edilmektedir.

Servis Bazlı Mimari;



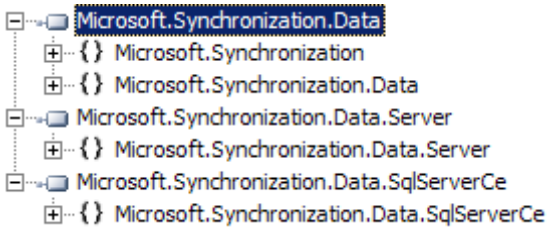
SOA modeli sunucu tarafındaki veri kaynağının **SQL** olmadığı durumlarda ele alınabilir. Bu sebepten dolayı sunucu tarafında sunucu senkronizasyon sağlayıcısı veya senkronizasyon adaptörleri bulunmamaktadır. Bu mimaride istemcinin sunucu tarafı ile mutlak suretle bir servis üzerinden konuşuyor olması gerekmektedir. Bir başka deyişle sunucu senkronizasyon sağlayıcısı ile senkronizasyon adaptörlerinin görevini servis tarafı üstlenmektedir. Bu noktada servis tarafında **WCF** gibi gelişmiş modellerin kullanılmasıda mümkündür.*(İlerleyen görsel derslerimizde bu konuyuda incelemeye çalışıyor olacağız.)*

*NOT : **.Net Framework** tarafından bakıldığında **Ado.Net Sync Service**' ler aşağıdaki 3 temel **assembly** ve tiplerinden yaralanmaktadır.*

***Microsoft.Synchronization.Data.dll** assembly (Synchronization Agent, Synchronization Tables ve Synchronization Groups)*

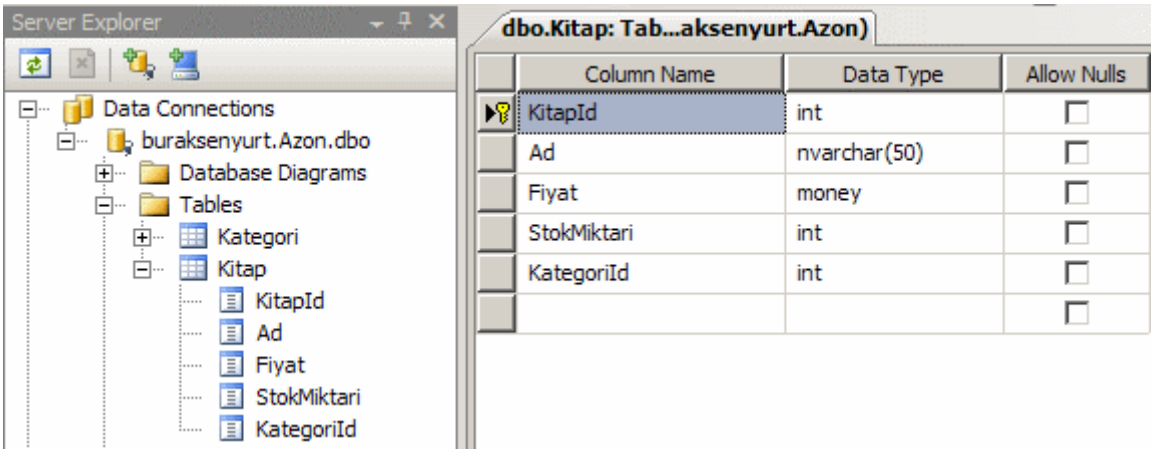
***Microsoft. Synchronization.Data.SqlServerCe.dll** (Client Synchronization Provider)*

***Microsoft. Synchronization.Data.Server.dll** (Server Synchronization Provider ve Synchronization Adapters)*

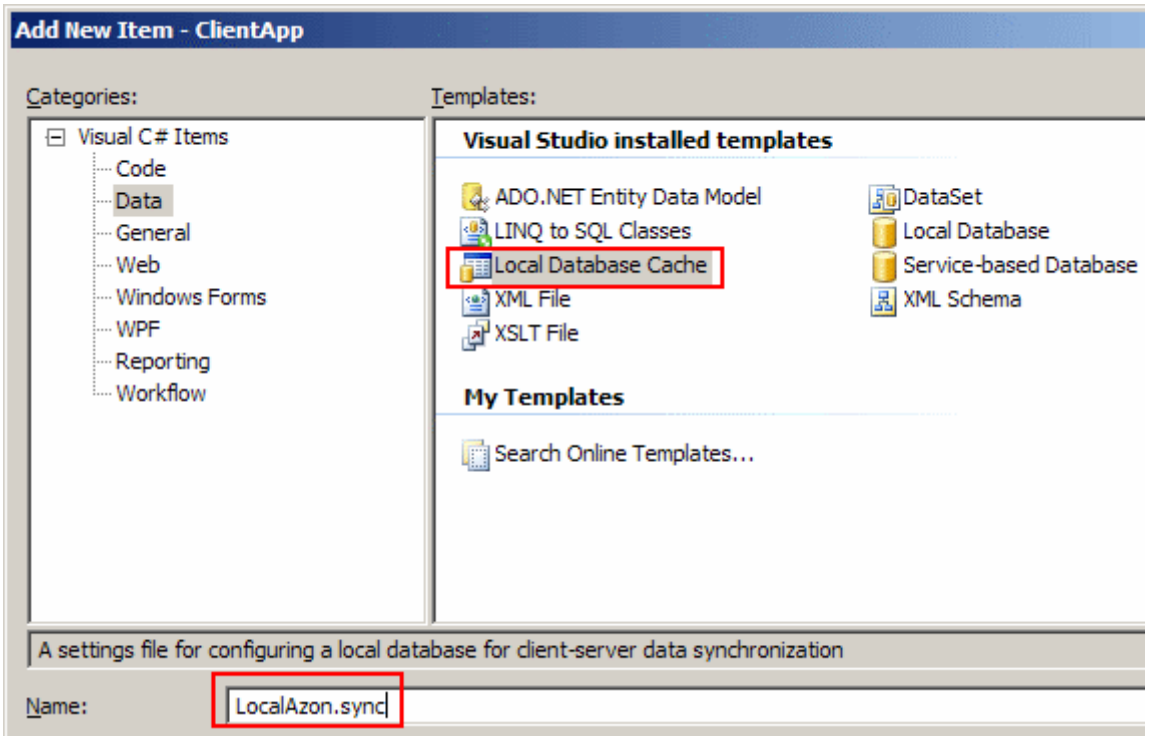


*(Tabi **Sync Service For Ado.Net**' i kullanabilmek için ilgili sürümünü indirip kurmanız gerekmektedir. Makalenin yazıldığı tarihten sonra farklı versiyonların çıkması ve muhtemel değişimlerin olmasınında söz konusu olduğunu belirtmek isterim. Makalemizde **Microsoft Sync Framework 2.0 CTP** sürümü içerisindeki **Sync Services For Ado.Net** alt yapısı kullanılmaktadır.)*

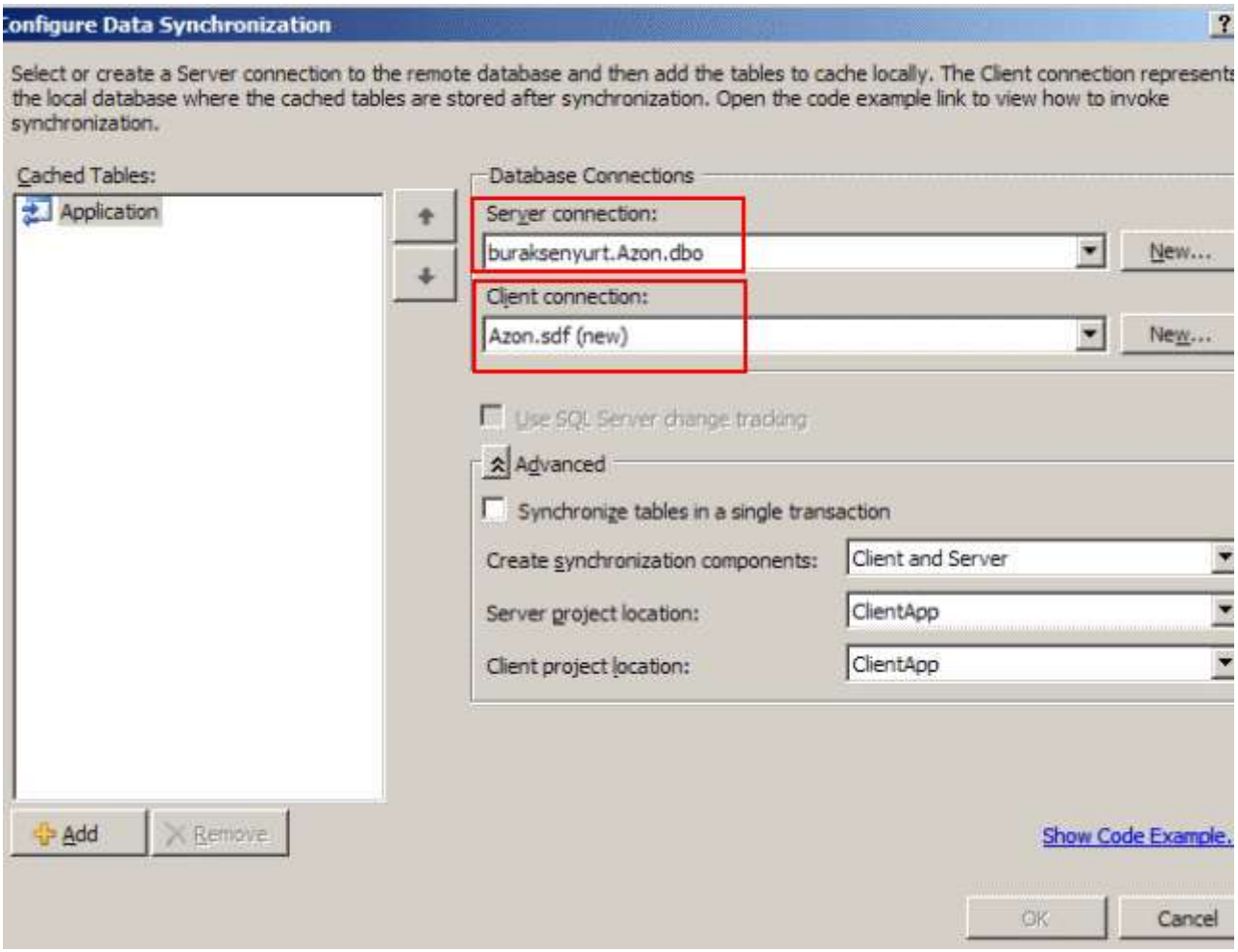
Makalemizin bundan sonraki bölümünde teknik detayları bir kenara bırakıp çok basit bir örnek üzerinden konuyu daha net bir şekilde kavramaya çalışacağız. örnekte kullanılmakta olan **Windows** uygulaması üzerinde, **Azon** isimli örnek veritabanında yer alan **Kitap** isimli tablo için **çift yönlü(Bidirectional)** senkronizasyon işlemleri yapılmaktadır. Ancak elbette istediğiniz tipte bir veritabanı ve tablolarını kullanabilirsiniz. Tablomuzun ilk hali aşağıdaki şekilde görüldüğü gibidir ve bu noktadaki hali oldukça önemlidir.



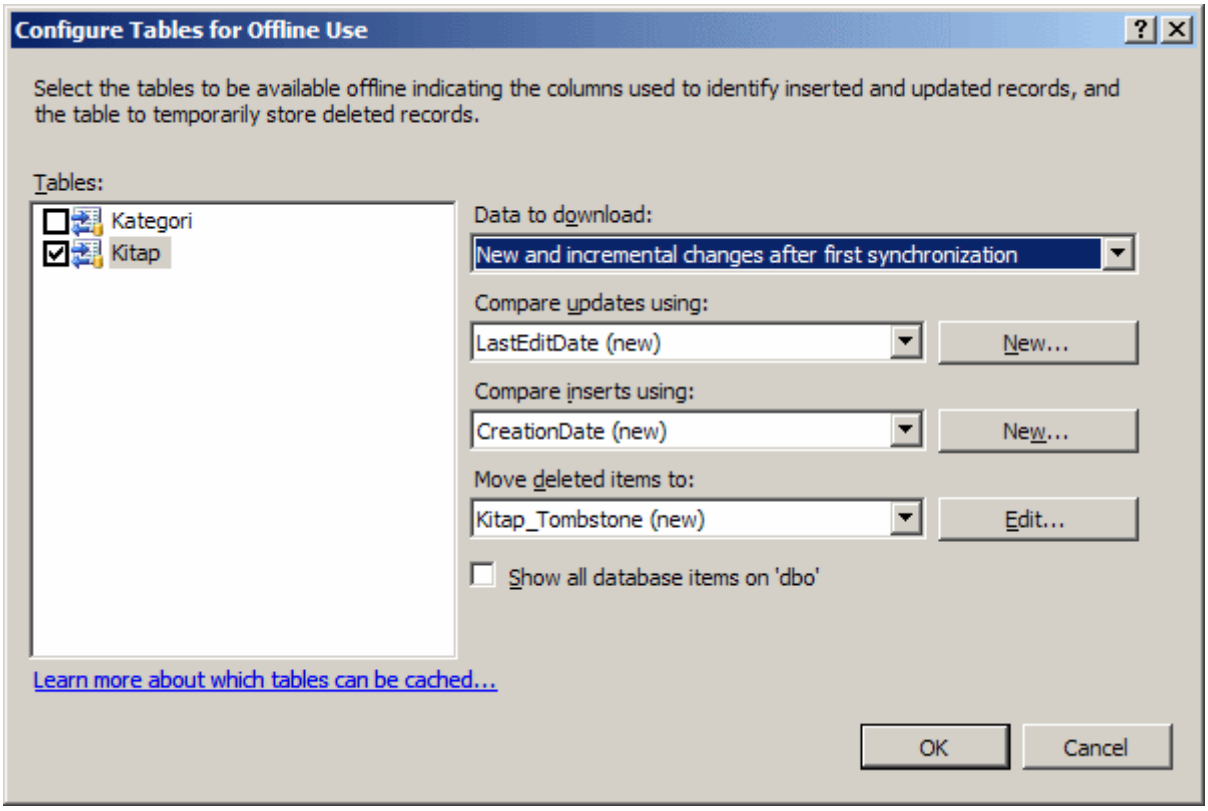
Biraz sonra tablonun şu anki alan yapısının senkronizasyon desteği için değiştiğini göreceğiz :) Şimdi örnek Windows uygulamamıza **Local Database Cache** isimli yeni bir **şabloun öğe(Template Item)** ekliyoruz.



Sync uzantılı **LocalAzon** isimli öğe temel olarak senkronizasyon işlemleri ile ilişkili ayarları tutacaktır. örneğin senkronizasyon tablolarına ait **şema(Schema)** bilgileri, **bağlantı(Connection)**ayarları, kodlama kısımları vb... Bu nedenle ilk olarak karşımıza aşağıdaki pencere çıkacaktır.

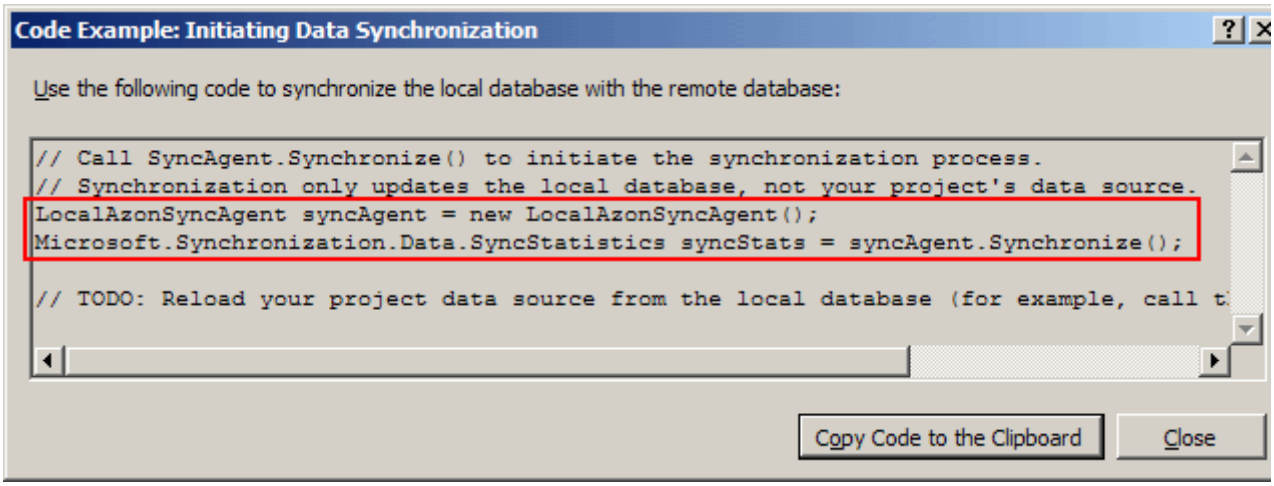


Burada ilk etapta sunucu veri kaynağı ve istemci veri kaynağına ait bağlantılar belirtilir. Dikkat edileceği üzere sunucu bağlantısı **Sql Server***(ki örnekte SQL Server 2008 üzerinde durmaktadır)* üzerindeki veritabanını işaret etmektedir. İstemci bağlantısında ise **sdf** uzantısı ile dikkat çeken ve biraz sonra oluşturulacak olan **Sql Server Compact 3.5** sürümünde bir veritabanı dosyası adı bulunmaktadır. **Advanced** kısmına baktığımızda ise senkronizasyon için **transaction** tanımlaması yapılabildiği de dikkati çekmektedir. Yani tüm tabloların senkronizasyon işlemlerinin ortak bir transaction içerisinde gerçekleştirilmesi isteği belirtilebilmektedir. Diğer taraftan önemli noktalardan biriside sunucu ve istemci proje lokasyonlarıdır. Şu an itibariyle her iki değerde geliştirmekte olduğumuz projeyi işaret etmektedir. Elbetteki gerçek hayat vakalarında özellikle sunucu proje lokasyonu başka bir uygulamayı işaret etmektedir.*(N-Tier veya SOA uyarlaması)*. Bu adımı tamamlamadan önce sol taraftaki **Application** kısmına yeni bir **offline table** eklenmesi gerekmektedir. Bu amaçla, **Add** düğmesine basıldıktan sonra aşağıdaki ekran görüntüsü ile karşılaşılacaktır.

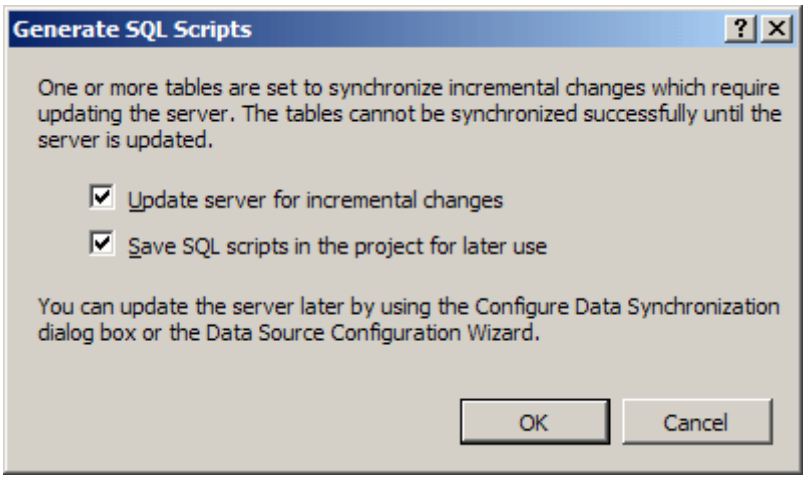


Bu kısımda senkronizasyon sürecine dahil olacak tablo(tablolar) veya veritabanı nesneleri seçilir. Dikkat edileceği üzere verinin istemci tarafına indirilme şekli belirlenebilmektedir. Şu andaki seçime göre ilk senkronizasyon işleminden sonra **değişimsel(Incremental)** ve yeni eklemelerin indirilmesi seçeneği etkindir.

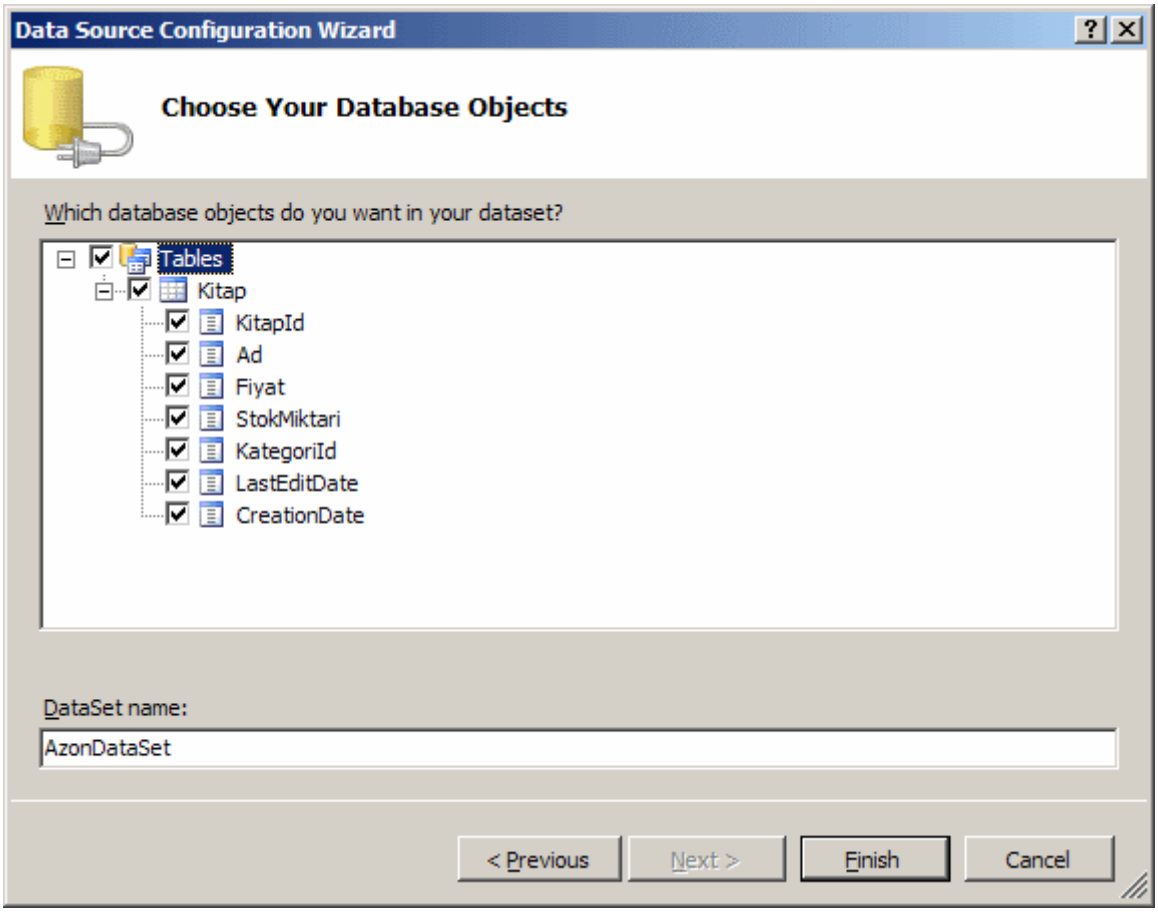
Senkronizasyon işlemleri sırasındaki önemli noktalardan biriside, istemci ve sunucu arasındaki bağlantının tekrardan sağlanması sonrasında karşılıklı olarak verilerde yapılan işlemlerin nasıl ayırt edilebileceğidir. Söz gelimi yeni güncelleştirmeler, silmeler veya eklemeler nasıl ayırt edilebililr. Bu sebepten **update** işlemleri için varsayılan olarak **LastEditDate**, **insert**işlemleri için **CreationDate** isimli iki yeni alanın **Kitap** isimli tabloya eklenmesi söz konusudur. Diğer taraftan silinen satırlar **_Tombstone** uzantılı bir tabloda saklanacaktır ve bu şekilde takip edilebilecektir. **New** veya **Edit** düğmelerinden yararlanarak söz konusu parametrelerin farklı isimlerde oluşturulmalarıda sağlanabilir. Burada geliştiricilerin hayatını kolaylaştıran bir linkte bulunmaktadır. Tablo eklenmesi tamamlandıktan sonra **Configure Data Synchronization** penceresinde yer alan **Show Code Example** linkine tıklandığında örnek bir kod parçası görülür.



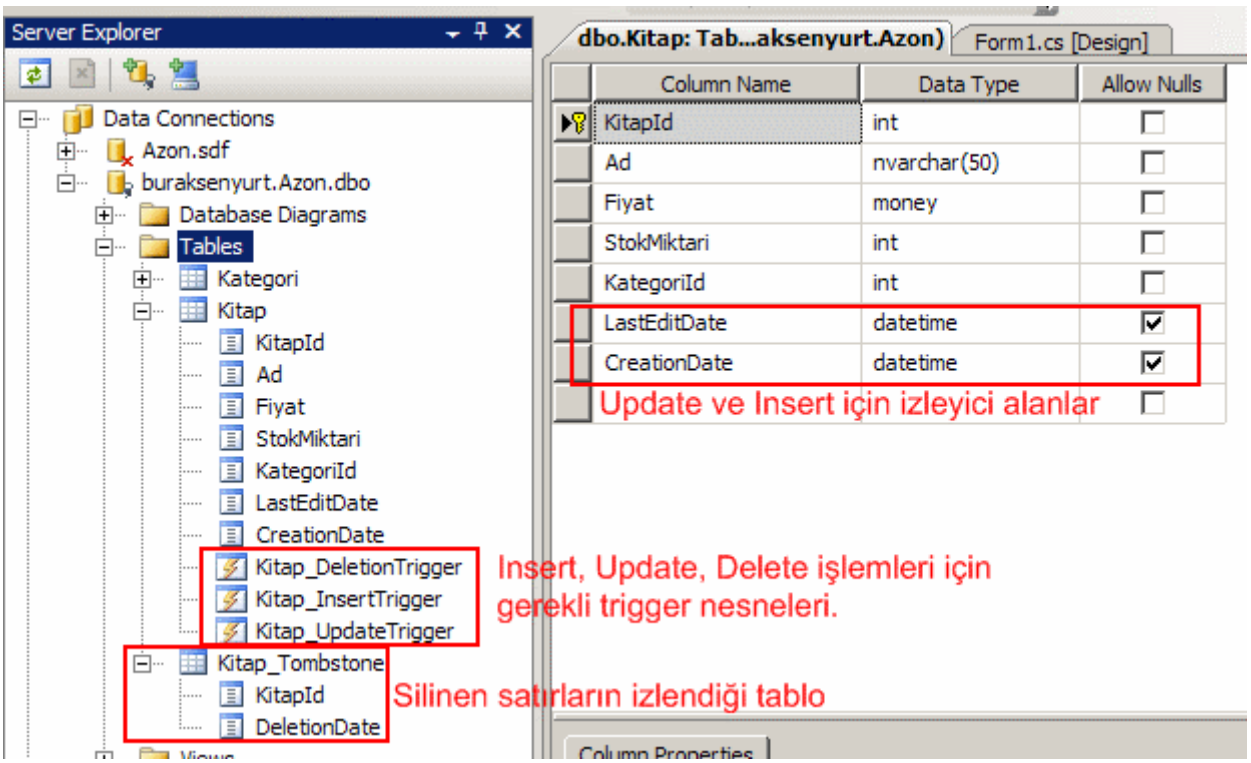
Bu kod parçası senkronize işleminin istemci tarafında başlatılacağı yerde kolayca kullanılabilmektedir ki biz de öyle yapıyor olacağız :) **Configure Data Synchronization** pencersinden çıkılırken istenirse senkronizasyon işlemleri için kullanılacak **SQL Script**' lerinin istemci uygulamaya eklenmesi sağlanabilir. Bunun için aşağıdaki penceredeki seçenekleri varsayılan halleri ile bırakmak yeterli olacaktır.



Artık istemci tarafı için gerekli olan **türlendirilmiş(Typed) DataSet**, **DataTable** ve **DataAdapter** üretimleri yapılacaktır. Bunun için var olan Kitap tablosunun seçilmesi yeterlidir.



Senkronizasyon kodlarını eklemeden önce, istemci uygulamada ve sunucu veritabanında olan düzenleme ve ilaveleri analiz etmeye başlayabiliriz. İlk dikkat çekici nokta sunucu üzerindeki Azon veritabanında olan ilavelerdir.



Görüldüğü üzere güncelleme ve ekleme işlemlerinin takibi için Kitap tablosuna **LastEditDate** ve **CreationDate** isimli **datetime** tipinden iki alan eklenmiştir. Silinen satırların bilgisi için**Kitap_Tombstone** isimli bir tablo oluşturulmuştur. Bu tablo **KitapId** ve **DeletionDate** isimli alanları içermektedir. Böylece hangi satırın ne zaman silindiği bilgisi tutulabilmektedir. Diğer taraftan**Insert**, **Update** ve **Delete** işlemlerinden sonra devreye giren t**etikleyicilerinde(triggers)** eklendiği görülebilir. Triggerların içerikleri ve ne iş yaptıkları kısaca aşağıdaki tabloda açıklanmaktadır.

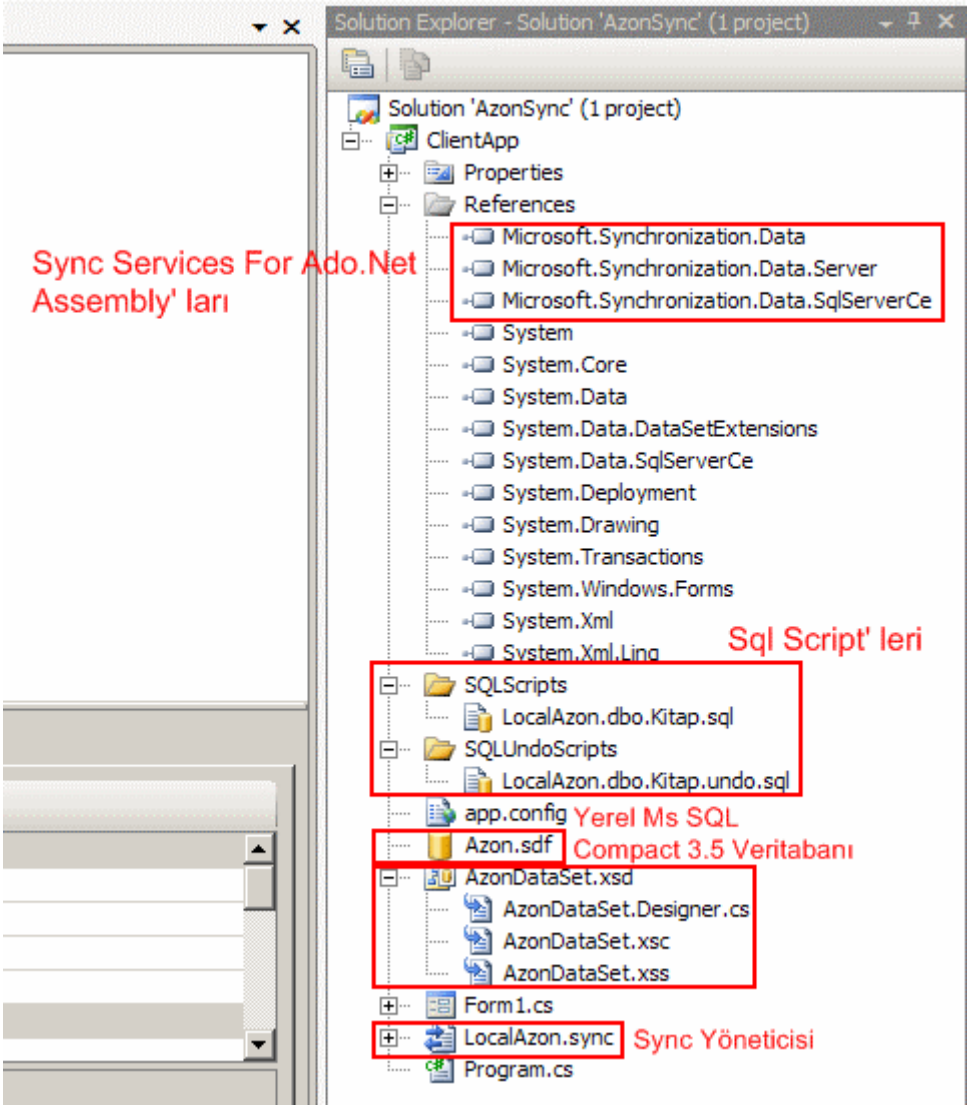
Trigger	**Query**
Kitap_DeletionTrigger	ALTER [dbo].[K ON [dbo AFTER AS SET NO UPDA SET [I FROM WHEF deleted [dbo].[K IF @@ BEGIN INSE ([Kit SEL deleted END

Kitap_UpdateTrigger	ALTER
	ON [dbo
	AFTER
	AS
	BEGIN
	SET NO
	UPDA
	SET [I
	inserted
	WHEF
	[dbo].[K
	END;

Kitap_InsertTrigger	ALTER ON [dbo

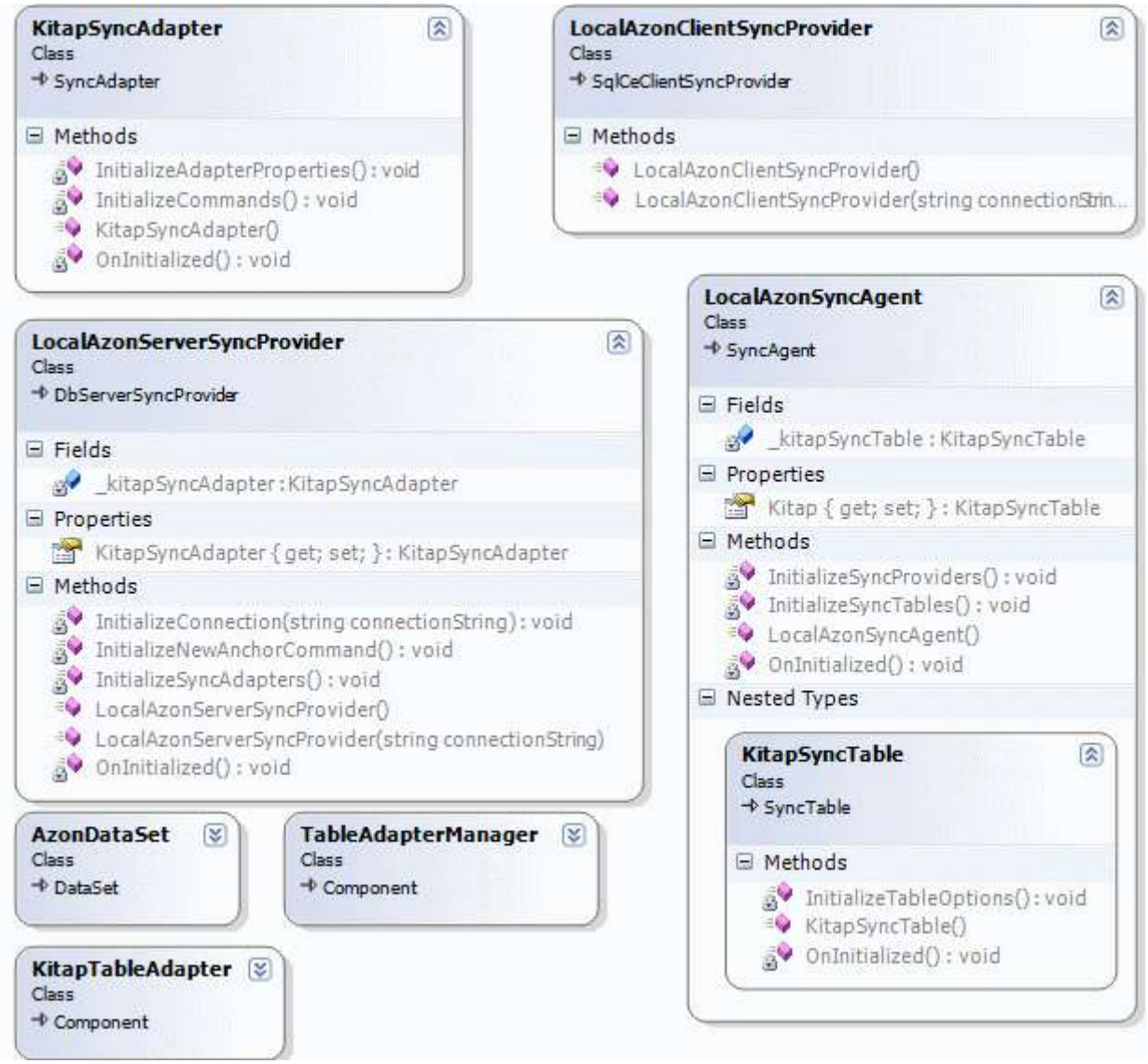
	AFTER
	AS
	BEGIN
	SET NO
	UPDA
	SET [
	inserted
	WHEF
	[dbo].[K
	END;

Peki ya uygulama tarafındaki değişiklikler nelerdir?



Görüldüğü üzere **Sync Services for Ado.Net** için gerekli olan **Microsoft.Synchronization.Data, Microsoft.Synchronization.Data.Server** ve **Microsoft.Synchronization.Data.SqlServer Ceassembly**' ları projeye referans olarak gelmektedir. Tüm senkronizasyon alt yapısına ait

tipler bu **assembly**' lar içerisinde gelmektedir. Diğer taraftan istemci uygulamada yerel bir **sdf** dosyasıda oluşturulmuştur. Bu dosya **local** olarak çalışabilen **Sql Server Compact 3.5** versiyonunda bir veritabanıdır. İstemci uygulamadaki görsel veri bağlı bileşenler için türlendirilmiş **DataSet**içeriği **AzonDataSet.xsd** adıyla oluşturulmuştur. Senkronizasyon ayarlarını ve işlemlerini üstlenen tipleri **LocalAzon.sync** öğesi barındırmaktadır. Bunlara ek olarak istemci tarafı için üretilen tiplere baktığımızda aşağıdaki sınıf diagramında yer alan temel öğeler dikkat çekmektedir.

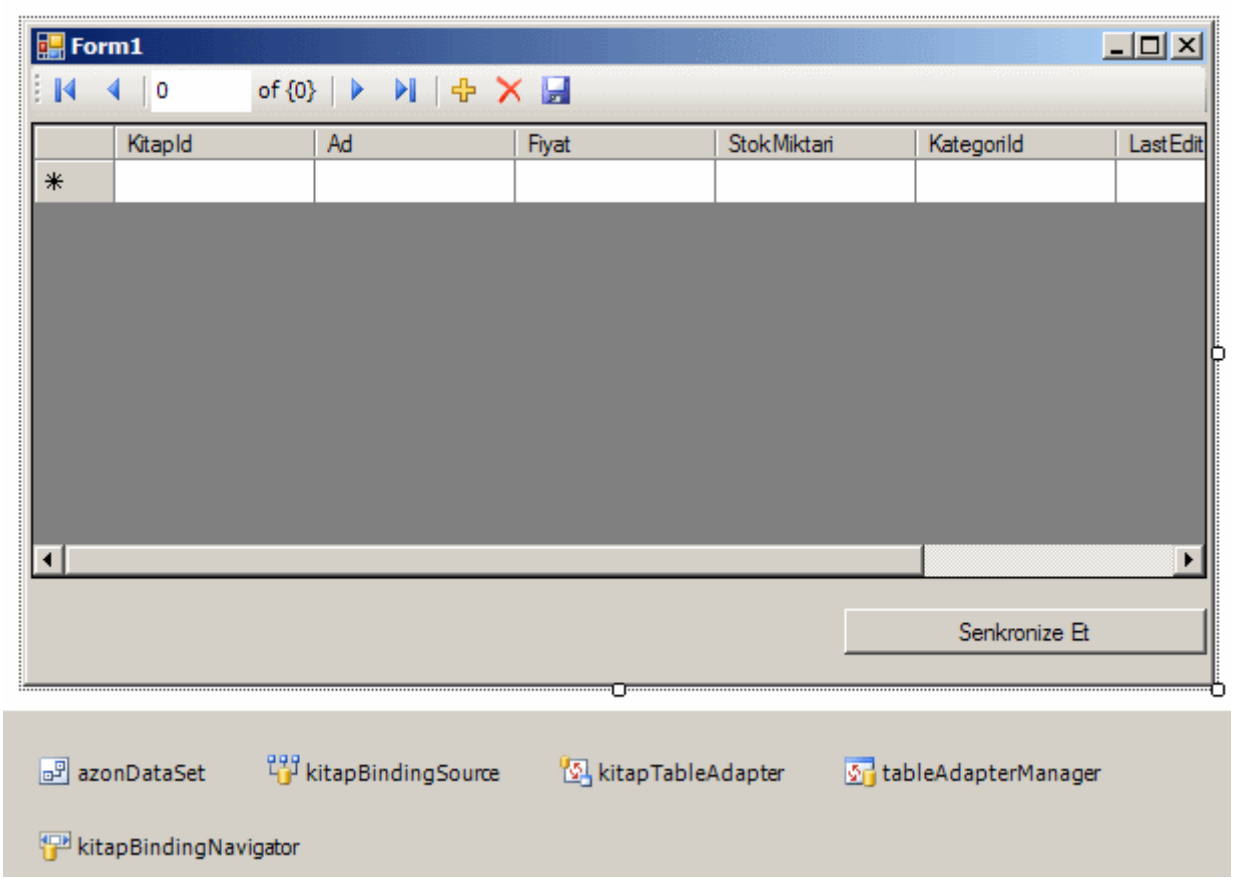


Buradaki tiplerin temel işlevleri aşağıdaki tabloda belirtilmektedir.

Kullanılan Sınıf	**Açıklama**
DbServerSyncProvider	**Microsoft.Synchronization.Data.Server.dll** assembly' ı içerisinde yer Sunucu üzerindeki senkronizasyon tablolarının bilgilerinin tutulması, s edilmesi, sunucu veritabanına **değişimsel(incremental)** farklılıkların a

SqlCeSyncProvider	**Microsoft.Synchronization.Data.SqlServerCe.dll** assembly'ı içerisin İstemci tarafında senkronizasyona dahil edilmiş tabloların bilgilerinin s elde edilmesi, istemci veritabanına değişimsel farklılıkların aktarılması
SyncAdapter	**Microsoft.Synchronization.Data.Server.dll** assembly' ı içerisinde ye görmektedir. Senkronize işleminde ele alınan her tablo için bu tipten türeyen bir sını olan **DbCommand** örneklerini bir başka deyişle **SQL** sorgularını içeri
SyncAgent	**Microsoft.Synchronization.Data.dll** assembly'ı içerisinde yer almakta
SyncTable	**Microsoft.Synchronization.Data.dll** assembly'ı içerisinde bulunmakta Senkronizasyon işlemine tabi olan tüm tablolar için istemci tarafında b oluşturulmaktadır. Senkronizasyona dahil olan istemci tablolarına ait a
*Makalede geliştirdiğimiz örnekte sunucu tarafı senkronizasyon tiplerininde istemci uygulama üze **Synchronization** kısmındaki**Application** ayarlarına bakıldığında istemci ve sunucu uygulamaları iletişimi sağlayan ayrık bir uygulamanın servis bazlı olması söz konusudur ki bu durumda **N-Tie***	

Testlere başlamadan önce istmeci uygulamanın tasarımını basit olarak aşağıdaki gibi değiştirelim.



Burada **GridView** kontrolü otomatik olarak **AzonDataSet** içerisindeki **Kitap** tablosuna bağlıdır ve üzerinde bağlantısız olarak veri ekleme, çıkarma, güncelleştirme işlemleri yapılabilmektedir. Diğer taraftan **Senkronize Et** başlıklı düğmenin içeriği aşağıdaki gibidir.

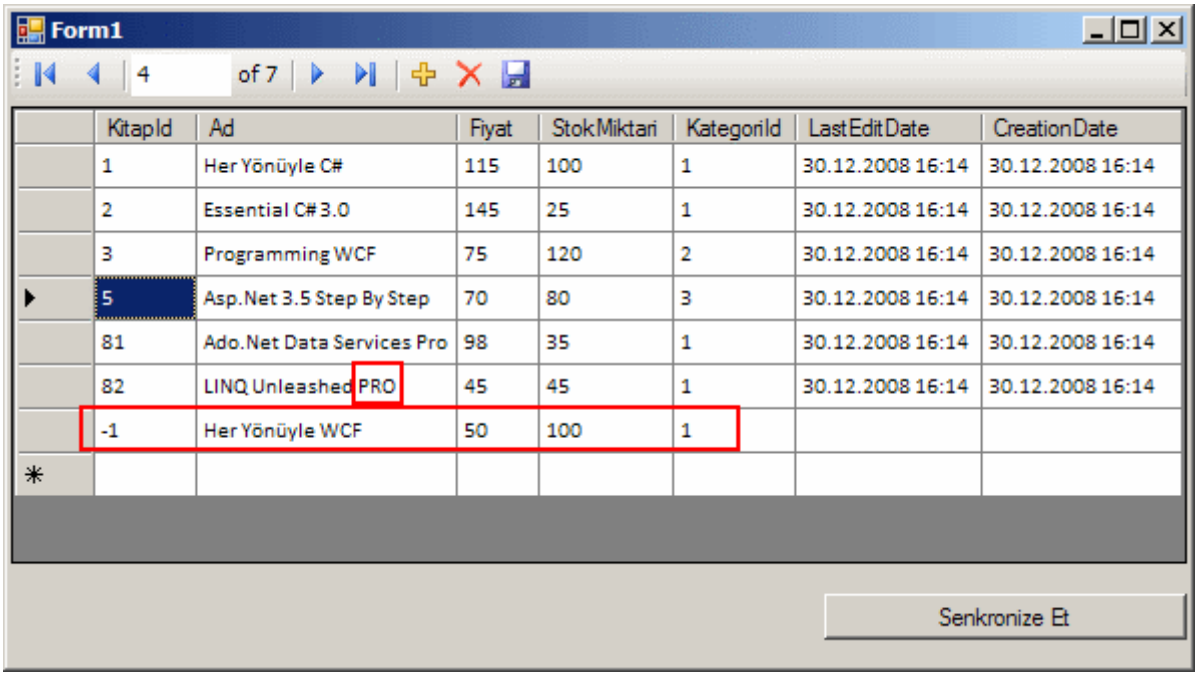
```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;

namespace ClientApp
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void kitapBindingNavigatorSaveItem_Click(object sender, EventArgs e)
      {
         this.Validate();
         this.kitapBindingSource.EndEdit();
         this.tableAdapterManager.UpdateAll(this.azonDataSet);
      }

      private void Form1_Load(object sender, EventArgs e)
      {
         this.kitapTableAdapter.Fill(this.azonDataSet.Kitap);
      }

      private void btnSenkronizeEt_Click(object sender, EventArgs e)
      {
         LocalAzonSyncAgent syncAgent = new LocalAzonSyncAgent();
         Microsoft.Synchronization.Data.SyncStatistics syncStats =
syncAgent.Synchronize();
         Form1_Load(null, null);
      }
   }
}
```

Tahmin ettiğiniz gibi biz sadece button içeriğini ekliyoruz. Burada ilk olarak bir **Agent** nesnesi örnekleniyor. Sonrasında ise **Synchronize** metodu ile senkronizasyon işlemi başlatılıyor. Bu işlemin sonuçlarını istersek üretilen **SyncStatistics** tipi üzerinden elde edebiliriz ki bunu loglama amacıyla kullanabiliriz. Yaptığımız bu değişiklikler sonrasında uygulamayı test ettiğimizde özellikle çift yönlü olarak bir senkronizasyon işlemi yapılamadığını göreceğiz. Bu sorunu çözmek için aşağıdaki kod parçasında da görüldüğü gibi **LocalKitap.sync**kod dosyasının içeriğini değiştirmemiz ve senkronizasyon tipini söz konusu **Kitap** tablosu için belirtmemiz gerekmektedir.

```
namespace ClientApp {

public partial class LocalAzonSyncAgent {
   partial void OnInitialized(){
         Kitap.SyncDirection =
Microsoft.Synchronization.Data.SyncDirection.Bidirectional;
      }
   }
}
```

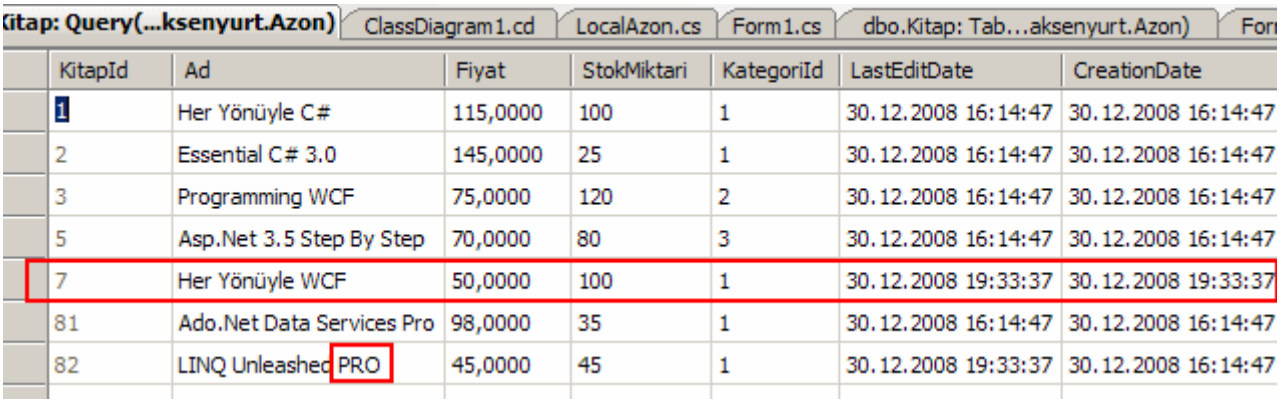
Artık Kitap tablosu için çift yönlü olaraktan senkronizasyon kontrolü yapılacaktır. Bir başka deyişle hem istemci hemde sunucu tarafındaki değişiklikler senkronizasyon işlemleri sonrasında karşı tarafa iletilecek ve verilerin eşlenmesi sağlanacaktır. Elbette birden fazla tablo kullanılması halinde bu tabloların her biri için ayrı ayrı senkronizasyon yönleri seçilebilir.

Artık kısa bir kaç test yapılabilir. İlk olarak istemci uygulama çalıştırıldığında Kitap tablosunun tüm içeriğinin çekildiği gözlemlenecektir. Burada tüm verinin indirilmesi son derece normaldir. Program çalıştıktan sonra örneğin, istemci tarafındaki veri içeriğinde çeşitli değişiklikler yaptığımızı varsayalım. örneğin KitapId alanının değeri 4 olan satırı sildiğimizi, yeni bir kitap eklediğimizi ve 83 numaralı kitabın adı için bir güncelleştirme yaptığımızı düşünelim.



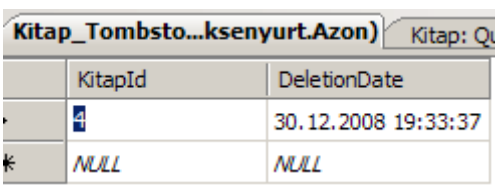
KitapId	Ad	Fiyat	StokMiktari	KategoriId	LastEditDate	CreationDate
1	Her Yönüyle C#	115	100	1	30.12.2008 16:14	30.12.2008 16:14
2	Essential C# 3.0	145	25	1	30.12.2008 16:14	30.12.2008 16:14
3	Programming WCF	75	120	2	30.12.2008 16:14	30.12.2008 16:14
5	Asp.Net 3.5 Step By Step	70	80	3	30.12.2008 16:14	30.12.2008 16:14
81	Ado.Net Data Services Pro	98	35	1	30.12.2008 16:14	30.12.2008 16:14
82	LINQ Unleashed PRO	45	45	1	30.12.2008 16:14	30.12.2008 16:14
-1	Her Yönüyle WCF	50	100	1		

Bu işlemlerin arkasından değişiklikleri **DataSet** üzerinde kaydedip **Senkronize Et** düğmesine basarsak, farklılıkların sunucu tarafınada aktarıldığını görebiliriz. Bu noktada özellikle **LastEditDate** ve **CreationDate** alanlarının istemci ve sunucudaki değerleri farklılıkların tespitini kolaylaştırmaktadır. Bunu daha rahat görebilmek amacıyla Azon.sdf dosyası içerisindeki Kitap tablosu ile sunucudaki Azon veritabanında yer alan Kitap tablosunun içeriklerini karşılaştırmanızı öneririm.



KitapId	Ad	Fiyat	StokMiktari	KategoriId	LastEditDate	CreationDate
1	Her Yönüyle C#	115,0000	100	1	30.12.2008 16:14:47	30.12.2008 16:14:47
2	Essential C# 3.0	145,0000	25	1	30.12.2008 16:14:47	30.12.2008 16:14:47
3	Programming WCF	75,0000	120	2	30.12.2008 16:14:47	30.12.2008 16:14:47
5	Asp.Net 3.5 Step By Step	70,0000	80	3	30.12.2008 16:14:47	30.12.2008 16:14:47
7	Her Yönüyle WCF	50,0000	100	1	30.12.2008 19:33:37	30.12.2008 19:33:37
81	Ado.Net Data Services Pro	98,0000	35	1	30.12.2008 16:14:47	30.12.2008 16:14:47
82	LINQ Unleashed PRO	45,0000	45	1	30.12.2008 19:33:37	30.12.2008 16:14:47

Yeni kayıt ekleme ve güncelleme işlemleri dışında istemci tarafında birde satır silmiştik. Bu nedenle sunucu veritabanındaki **Kitap_Tombstone** tablosunda aşağıdaki şekildende anlaşılacağı üzere 4 numaralı satır(Silinen kitabın KitapId değeri) için bir ekleme yapıldığı gözlemlenebilir.



KitapId	DeletionDate
4	30.12.2008 19:33:37
NULL	NULL

İkinci test olarak sunucu üzerinde değişikliker yapıp bunları istemci tarafına, uygulamadaki **Senkronize Et** düğmesini kullanarak alabiliriz. Şu nokta unutulmamalıdır; Senkronize etme işlemi program her açıldığında yapılmamaktadır. Nitekim veritabanında değişiklik yapılsa ve istemci ile sunucu arasında bir bağlantı olmasa bile, istemci uygulama yerel veritabanı(Local Database-sdf) üzerindeki veriler ile çalışarak, değişiklikler, eklemeler ve silmeler yapabilir ama, senkronize etme işlemi başlamadığı sürece hem değişiklikleri alamaz hem de bunları sunucu veritabanına iletemez.

Bir açıdan bakıldığında, yapılan işlemler verilerin normal yollarla karşılıklı olarak uç kaynaklara gönderilmesinden pek farklı değildir. özellikle bağlantısız katman modelinde geliştirme yapıldığında benzer işlevsellikleri sağlayabiliriz. Ne varki **Sync Services for Ado.Net** alt yapısı sunucu ve istemci tarafındaki senkronizasyonun sağlanması adına geliştiriciye daha güçlü ve yönetilebilir tipler sunmaktadır. Ayrıca senkronize işlemlerinin yapılması sırasında sunucu veri kaynağının sabit bir **SQL** veritabanı olması şartı bulunmamakla birlikte, **N-Tier** yada **SOA** tabanlı geliştirme yapmakta mümkündür. Bu açılardan bakıldığında avantajlar ortadadır.

Konuyu daha iyi kavrayabilmek ve örneğimizin gerçek ortamdaki çalışmasını izleyebilmek adına video dersimizi izlemenizi öneririm. İlerleyen makalelerimizde **Ado.Net Senkronizasyon Servisleri**konusuna değinmeye devam ediyor olacağız. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

AzonSync.rar (154,25 kb)

Client Application Services (2008-12-16T06:35:00)

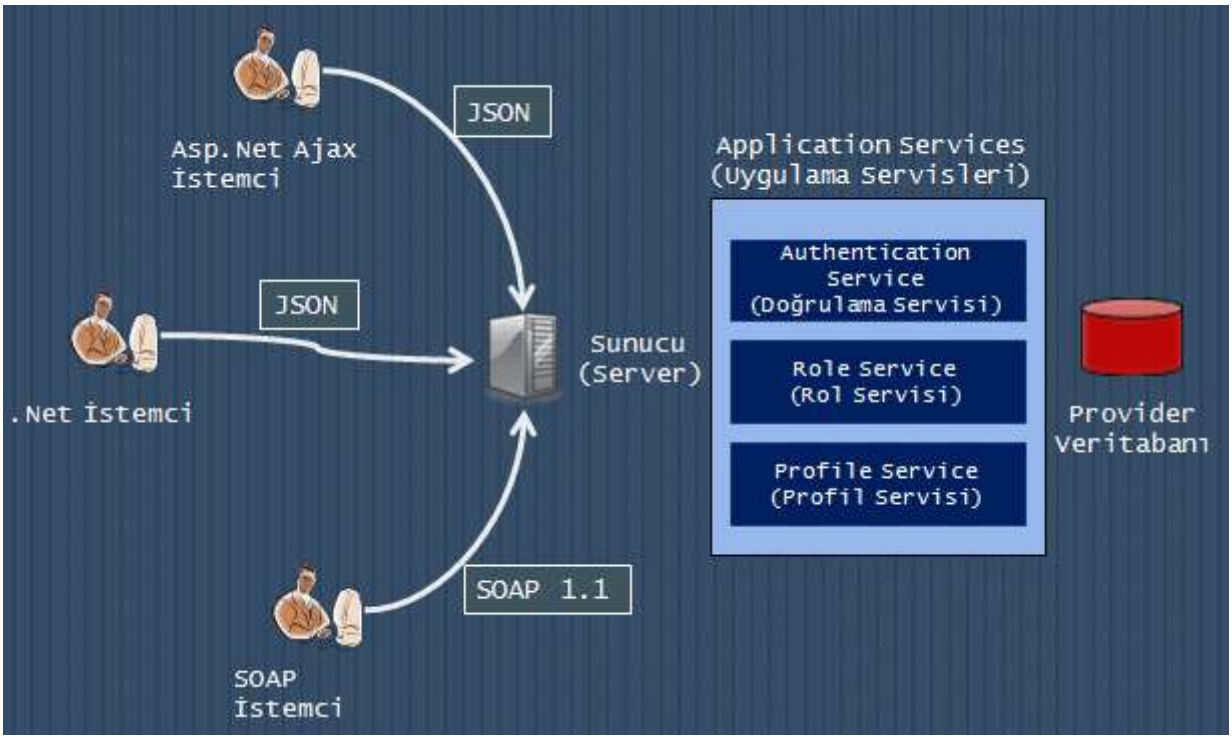
client application services,

Pek çok istemci uygulama için önem arz eden konular arasında **doğrulama(Authentication), Rol Yönetimi (Roles Management), profile(Profile)** göre kişiselleştirme yer almaktadır. Özellikle Web tabanlı uygulamalarda bu kıstaslar daha çok ön plana çıkmaktadır. Nitekim **Client/Server** mimarinin en güzel uyarlamalarından birisi olan web tabanlı geliştirmelerde, istemcilerin doğrulanması, rollerine göre ne yapabileceklerinin belirlenmesi, profillerine göre istekte bulundukları sayfaların kişiselleştirilmesi önemlidir. Bu noktada **Asp.Net 2.0** sürümünden itibaren saymış olduğumuz bu kriterlerin çok daha kolay bir şekilde uygulanabilmesi sağlanmıştır. Hatırlanacağı üzere **Asp.Net 2.0 Web Site Administration Tool** veya kod tarafında **Membership API** içerisinde yer alan tipler yardımıyla, kullanıcı hesaplarının yönetilmesi, çeşitli rollere atanması görsel olarak kolayca yapılabilmektedir. Bununla birlikte **Profile API** içerisinde yer alan tipler yardımıylada, bir web sitesinin kullanıcı bazında özelleştirilebilmesi son derece kolaylaşmıştır. Ancak bu noktada söz konusu kriterlerin windows tabanlı istemciler**(Windows Clients)**açısındanda değerlendirilebilir olması oldukça kıymetlidir. İşte tam bu noktada söz konusu kriterlerin servis haline

getirilmesi gerekliliği ortaya çıkmaktadır. Nitekim **authentication, role management, profile** gibi kıstaslar aslında servis haline getirilirlerse **Asp.Net Web** uygulamaları dışında da kullanılabilir hale gelirler. İşte bu günkü makalemizin konusu olan **İstemci Uygulama Servisleri(Client Application Services)** ile, **.Net** tabanlı Windows uygulamalarının(**Windows Presentation Foundation-WPF** yada **Windows Forms**) söz konusu kriterleri, .Net içerisine gömülü olan **Asp.Net Uygulama Servisleri(Asp.Net Application Services)** üzerinden gerçekleştirebilmeleri mümkündür.

NOT : Asp.Net Application Service*' ler olmasada windows istemcilerinin doğrulama, rol ve profil yönetimi için servis bazlı mimarilerinin ele alması mümkündür. Sonuç itibariyle burada basit servisler yazılarak bu ihtiyaçlar karşılanabilir. Ancak Asp.Net Application Service' ler Framework içerisine gömülü olduklarından tüm istemci çeşitleri için bir standardizasyon getirmektedir. Bununla birlikte* ***Visual Studio 2008*** *sürümünde gelen bir kaç basit yenilikte servislerin kullanımını kolaylaştırmaktadır.*

Asp.Net Uygulama Servisleri(Asp.Net Application Services) sadece **Asp.Net AJAX** web uygulamalarında değil, **.Net** tabanlı her hangibiri istemci tarafından ele alınabilir. Bu noktada her iki istemci uygulama çeşidide **JSON(JavaScript Object Notation)** tabanlı bir mesaj iletişimini doğal olarak HTTP üzerinden gerçekleştirmektedir. Ne varki Asp.Net Uygulama Servisleri, **SOAP 1.1(Simple Object Access Protocol)** temelli mesaj gönderen istemcilerede hizmet verebilmektedir. Buda çok doğal olarak **.Net** dışındaki uygulamaların söz konusu servisleri kullanabilmesi anlamına gelmektedir. Bir başka deyişle örneğin **Java** tabanlı bir uygulama bile Asp.Net Uygulama Servislerini çağırabilir ve **doğrulama(Authentication), rol ve profile yönetimlerini(Role and Profile Management)**kullanabilir. İstemci çeşitleri ve servisler arasındaki ilişkiler basitçe aşağıdaki şekil ile özetlenebilir.



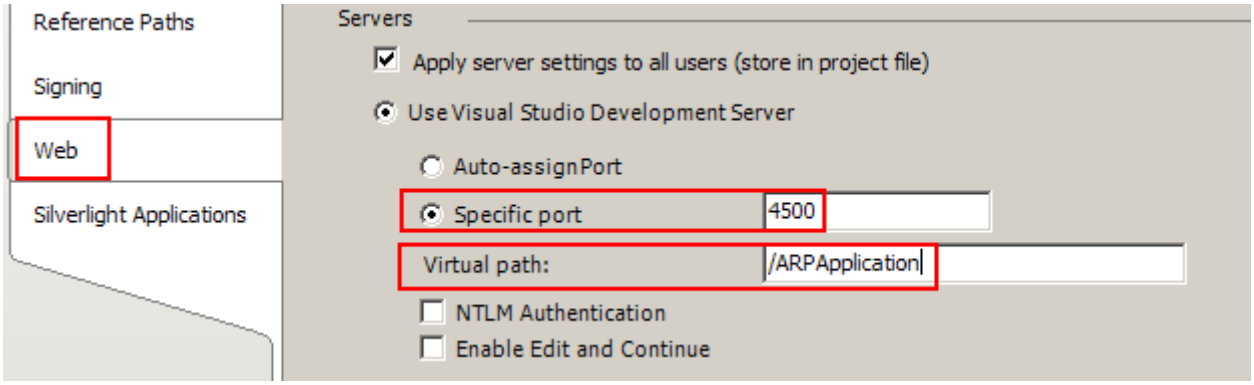
Görüldüğü üzere uygulama servisleri aslında birer Web Servisi mantığında olduğundan **HTTP** üzerinden her tür istemcinin ulaşabilmesi mümkündür.*(WCF bazlı bir kullanım şeklide mümkündür. Bunu ilerleyen makalelerimizde yada bir görsel dersimizde ele almaya çalışacağız)* Bu servislerin temel işlevleri arasında istemcilerin doğrulanması, doğrulama sonrasında istemci tarafında biletler(ticket) açılmasının sağlanması, istemcinin hangi rolde olduğunun tespit edilmesi ve profiline göre uygulamanın içeriğinin kişiselleştirilmesi sayılabilir. Dikkat edileceği üzere söz konusu hizmetler için bir veri saklama ortamı olması şarttır. Varsayılan olarak bu ortam bilindiği üzere **SQL** sunucusu(veya Express sürümü) üzerindeki**Membership** veritabanlarıdır. Ancak istenirse bu veri kaynaklarını kullanan provider' lar servis tarafında özelleştirilebilir ve farklı depoların kullanılması sağlanabilir. Bunun için basitçe konfigurasyon içeriğinde bir kaç değişiklik yapmak yeterli olacaktır. Servislerin temel görevleri aşağıdaki tabloda olduğu gibi özetlenebilir.

Servis	**Görevi**
Authentication Service(Doğrulama Servisi)	İstemcinin doğrulanması ve uygulamaya giriş yapabilmesi(Login) amacıyla kullanılır. **Login** işlemi sonrasında istemci tarafında saklanacak bir **bilet(Ticket)** oluşturulur. Tahmin edileceği üzere bu bilet bir **cookie** olarak depolanır ve bir geçerlilik süresi vardır.
Roles Service(Rol servisi)	Uygulama bazlı olaraktan istemcinin hangi rolde olduğunun **Asp.Net Role Provider** tarafından denetlenmesi hizmetini üstlenir. Buna göre istemci uygulamada, örneğin menülerin veya kontrollerin rol bazlı olarak kullanılabilmesi sağlanabilir.
Profile Service(Profil Servisi)	İstemci uygulamanın sunucu üzerinde tutulan kullanıcı verilerine göre herhangibir zamanda farklı biçimlerde gösterilebilmesine veya davranış sergilemesine hizmet eden servistir. Burada servis tarafında tüm kullanıcılar için ortaklaşa tanımlanmış profil özellikleri söz konusudur. Ancak özellik değerleri her istemci için farklı olarak tutulabilmektedir.

Bu genel açıklamalardan sonra sanıyorumki uygulama servisleri hakkında biraz fikir sahibi olunmuştur. Makalemizin bundan sonraki bölümünde yer alan hedefimiz ise bir windows istemcisi üzerinden söz konusu servislerin kullanılmasını sağlamaktır. Bu noktada **Visual Studio 2008** sürümünün **İstemci Uygulama Servisilerinin(Client Application Service)** ele alınmasında büyük kolaylıklar sağladığıda unutulmamalıdır. Öyleyse hiç vakit kaybetmeden işe başlayalım. Öncelikli olarak doğrulama, rol ve profil yönetimi hizmetlerini üstlenecek bir web servisi geliştiriyor olacağız.

İlk olarak bir **Web Service Application** projesi oluşturarak başlayabiliriz. Bu uygulama içerisinde varsayılan olarak gelen **Service.asmx** dosyasının söz konusu senaryoda herhangibir kullanım alanı bulunmamaktadır. Servisin tek görevi istemci uygulamaya **Authentication, Roles ve Profile** servis hizmetlerini sağlamaktır. Bu sebepten

asmx dosyası silinebilir. Şu an için test servisimiz file-based olarak açılmıştır. Ancak tabiki production ortamlarında IIS gibi bir sunucu altında yayınlanması önerilir. File-based kullanım nedeniyle Web Servis uygulamasının özelliklerinden basit olarak port ve sanal yol(Virtual Path) ayarlamalarını aşağıdaki şekildeki gibi yapmamız yeterli olacaktır.



Burada belirtilen bilgiler istemci uygulamada ele alınacaktır. Şimdi web.config dosyası içerisinde bazı basit ayarlamalar yapılması gerekmektedir. Bu amaçla web.config dosyası içeriğini şimdilik aşağıdaki gibi değiştirelim.

```
19
20
21    <appSettings/>
22    <connectionStrings/>
23  <system.web.extensions>
24    <scripting>
25      <webServices>
26        <authenticationService enabled="true"/>
27        <roleService enabled="true"/>
28        <profileService enabled="true"/>
29      </webServices>
30    </scripting>
31  </system.web.extensions>
32
33    <system.web>
34
35      <roleManager enabled="true"/>
36      <profile enabled="true"/>
```

Konfigurasyon dosyası içeriğinde **authentication**, **role** ve **profile** servislerini etkinleştirmek için **system.web.extensions** elementi altında bazı tanımlamalar yapılmıştır. Bununla birlikte**provider** bazında **role** ve **profile** yönetimlerininde etkinleştirilmesi amacıyla **roleManager** ve **profile** elementleri ele alınmaktadır. Bu basit ayarlamaların hemen ardından kullanıcı ve rol tanımlamalarının yapılmasına başlanabilir. Bunun için web servisi uygulamasında **Web Site Administration Tool**' dan yararlanılabilir.

NOT : *Örneğin geliştirildiği makinede yer alan* ***machine.config(C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\CONFIG)*** *i çeriğinde* ***LocalSqlServer*** *isimli* ***connection string*** *bilgisi aşağıdaki gibidir.*

```
124    <runtime/>
125    <connectionStrings>
126      <add name="LocalSqlServer" connectionString="data source=.\SQLEXPRESS;
       Integrated Security=SSPI;AttachDBFilename=|DataDirectory|aspnetdb.mdf;User
       Instance=true" providerName="System.Data.SqlClient"/>
127    </connectionStrings>
```

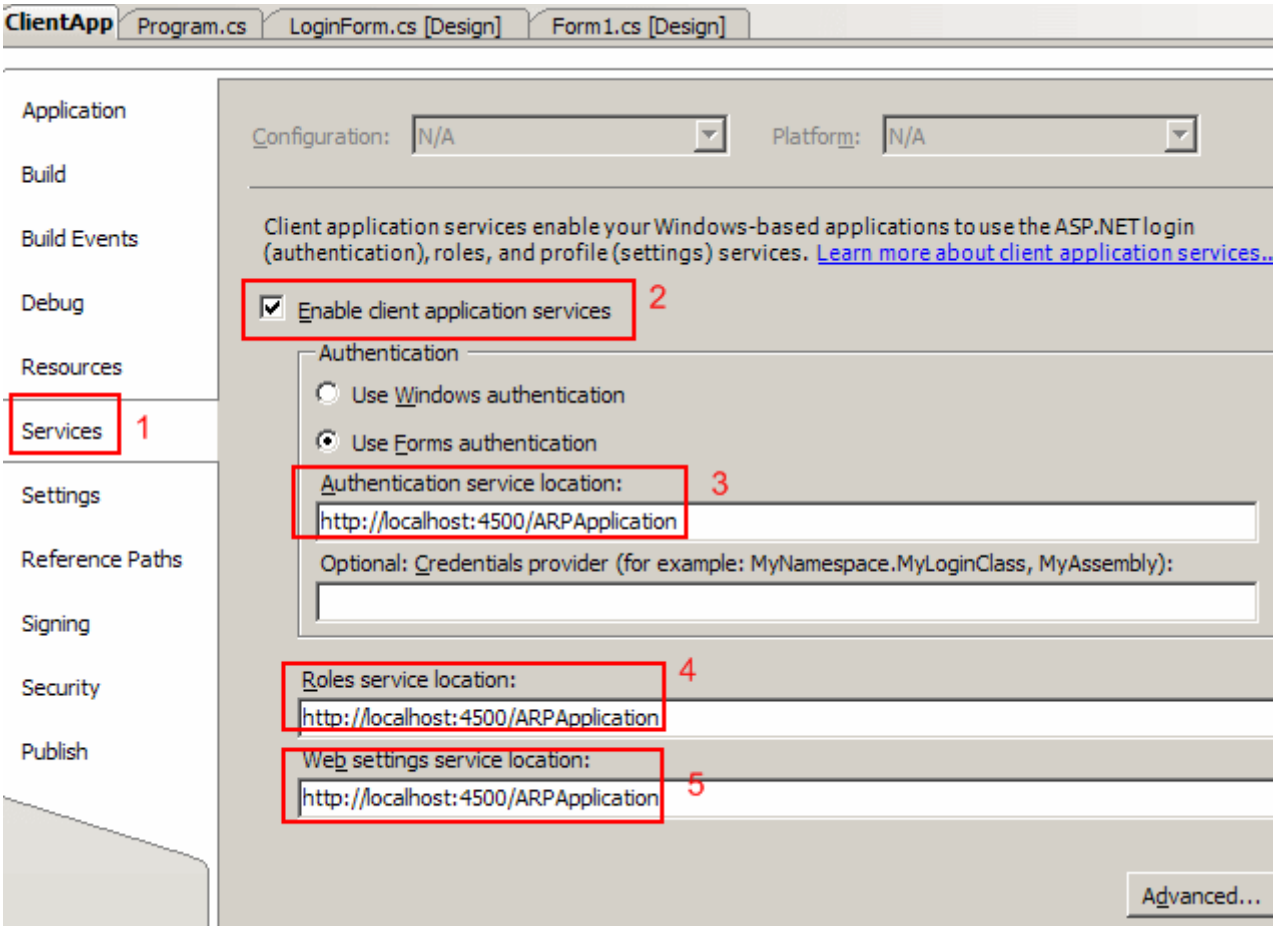
*Bu nedenle **Asp.Net Web Site Administration Tool**, web servisi uygulamasının içerisinde **membership** yönetimi için gerekli veritabanını**(Aspnetdb.mdf)** oluşturmaktadır. Elbetteki**LocalSqlServer** değerinin web servisi uygulamasının **web.config** dosyası içerisinde ezilmesi ve farklı bir lokasyonun işaret edilmeside sağlanabilir. Eğer makinedeki sunucu üzerinde veritabanının oluşturulması istenirse **aspnet_regsql** komut satırı aracından yararlınmalıdır. Veritabanı oluşturulduktan sonra ise config dosyası içeriğinde oluşturulan veritabanı adresi işaret edilerek devam edilebilir.*

Asp.Net Web Site Administration Tool üzerinde **From Internet** seçeneği ile **Form Based Authentication** açılır. Testler için iki farklı rol ve kullanıcı oluşturulur. Bu kullanıcılara ait bilgileri örnek senaryomuzda aşağıdaki gibi tanımlayabiliriz.

Kullanıcı	Şifre	Email	Gizli Soru	Gizli Cevap	Rol
buraks	buraks1234.	buraks@azon.com	buraks	buraks	Yonetici
bili	bili1234.	bili@azon.com	bili	bili	Calisan

Bu işlemlerin arkasından **AppData** klasörü altında **aspnetdb.mdf** veritabanı dosyasının açıldığı ve yukarıdaki kullanıcı bilgilerinin ilgili tablolara eklendiği görülebilir.

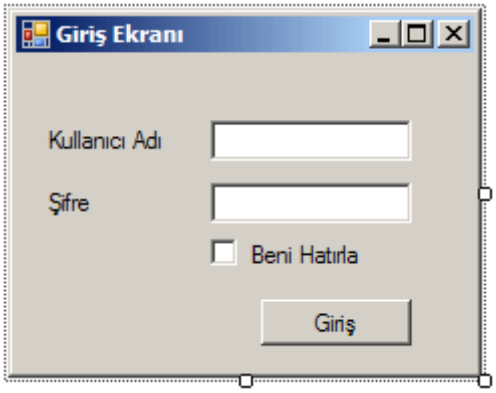
Artık windows tabanlı istemci uygulamanın geliştirilemesine başlanabilir. İstemci uygulama açısından en önemli nokta elbetteki yukarıda tanımlanmış olan web servisine erişilebilmesidir. Bu amaçla windows uygulamasının özelliklerinde aşağıdaki ekran görüntüsünde yer alan ayarların yapılması gereklidir.



İlk olarak projenin özelliklerinden **Services** kısmına geçilir. Burada **Enable client application services** seçeneği işaretlenmelidir. Sonrasında ise **Authentication**, **Roles** ve **Profile** servisleri için erişim adresleri belirtilir. Dikkat edileceği üzere her üç servis içinde **SecurityService** isimli web servis uygulamasının erişim bilgileri verilmektedir.

NOT : *Servis bildirimlerinde her bir kıstas için ayrı adresler verilebilmektedir. Dolayısıyla* ***Doğrulama, rol ve profile yönetimi*** *görevlerini üstlenen ve farklı lokasyonlarda duran 3 farklı servisin tanımlanması mümkündür.*

Windows uygulamasında basit bir giriş formu yer almaktadır. Bu form üzerinden kullanıcı bilgileri girilmekte ve servise gönderilmektedir. İşte bu noktada önemli bir referansa ihtiyacımız vardır. İstemci uygulamanın **System.Web.dll assmebly**' ını referans etmesi gereklidir. Nitekim ilk örneğimizde kullanacığımız **Membership** sınıfı bu assembly içerisinde yer alan**System.Web.Security isimalanında(Namespace)** yer almaktadır.




```
using System;
using System.Windows.Forms;
using System.Web.Security;

namespace ClientApp
{
   public partial class LoginForm
      : Form
   {
      public LoginForm()
      {
         InitializeComponent();
         txtSifre.PasswordChar = '*';
      }

      private void btnGiris_Click(object sender, EventArgs e)
      {
         try
         {
            if (Membership.ValidateUser(txtKullanici.Text, txtSifre.Text))
            {
               Form1 frm = new Form1();
               frm.Show();
               this.Hide();
            }
            else
            {
               MessageBox.Show("Giriş yetkiniz yok", "Yetkisiz Erişim",
MessageBoxButtons.OK, MessageBoxIcon.Stop);
               Application.Exit();
            }
         }
         catch (Exception excp)
         {
```

```
            MessageBox.Show(excp.Message);
            Application.Exit();
        }
    }
  }
}
```

Giriş formunda kullanıcı adı ve şifre bilgisi istenmektedir. Sonrasında ise **Membership** sınıfının **static ValidateUser** metodu ile kullanıcı servis üzerinden doğrulanmaya çalışılmaktadır. **ValidateUser** metodu **true** veya **false** değer döndürmektedir. **True** dönmesi halinde kullanıcı bilgisi doğrulanmıştır. Windows uygulaması çalıştırıldığında,**4500** numaralı **port** üzerinden **Asp.Net Development Server**'ında çalıştığı gözlemlenir. Doğru kullanıcı bilgisi girildiğinde formun açıldığı ancak hatalı bilgi girilmesi halinde ise programın kapandığı gözlemlenebilir.

Örnekte dikkat edilmesi gereken noktalardan biriside istemci tarafında herhangibi **proxy** tipinin bilinçli bir şekilde fiziki olarak oluşturulmayışıdır. Normal şartlarda özellikle windows tabanlı istemcilerin, servisleri kullanabilmeleri için servise ait proxy tiplerine ihtiyaçları vardır. Oysa Uygulama Servislerinin ele alınmasında esas olan nokta mesaj alış verişidir.

Şimdi örneğimizi biraz daha geliştireceğiz. Bu sefer **Credential Provider** kullanaraktan bir **Login** işlemi gerçekleştirmeye çalışacağız. Az önceki örnekte **LoginForm** isimli bir form tasarlamıştık. Bu form içerisindeki düğmeye bastığımızda **Membership** sınıfının static **ValidateUser** metodunu kullanıyorduk. Uygulamanın ana formuna geçiş yapmadan önce bu formun çıkmasını sağlamak içinse **Main** metodu içerisinde aşağıdaki kodlamayı kullanmıştık.

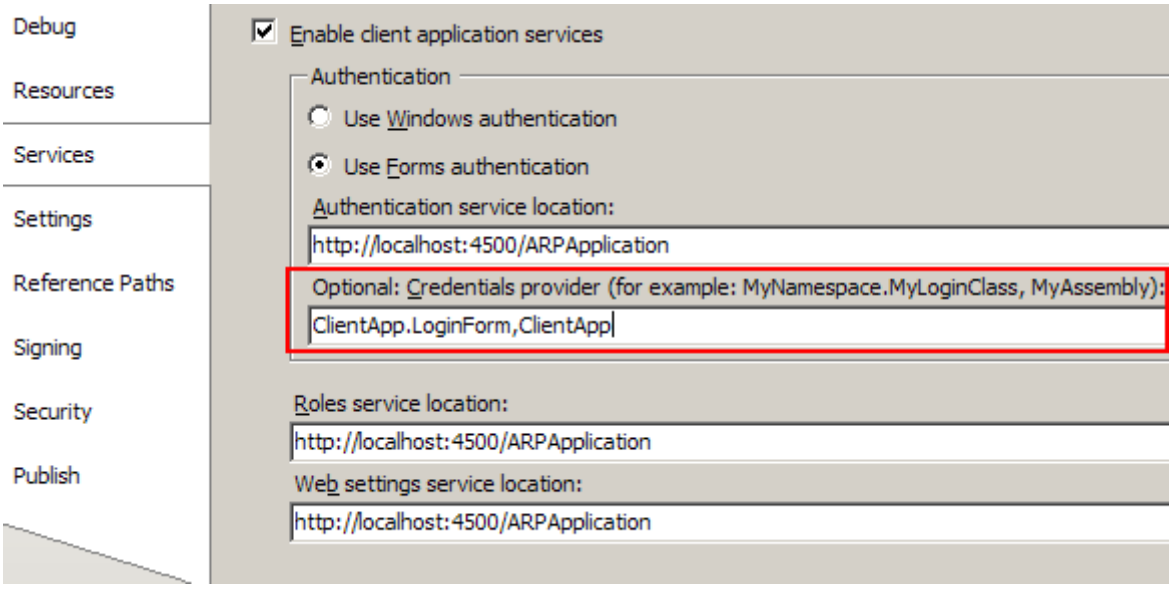
```
Application.Run(new LoginForm());
```

Crendentail Provider kullanaraktan aslında uygulamaya giriş yapıldığı sırada otomatik olarak **Login** formunun gösterilmesi sağlanabilir. Olayı daha kolay kavramak için örnek üzerinde adım adım ilerleyelim. Bu amaçla **LoginForm** üzerinden bazı değişiklikler yapmamız gerekmektedir. İlk olarak formumuzun kod yapısını aşağıdaki gibi düzenlemeliyiz.

```
using System;
using System.Windows.Forms;
using System.Web.Security;
using System.Web.ClientServices.Providers;

namespace ClientApp
{
   public partial class LoginForm
      : Form, IClientFormsAuthenticationCredentialsProvider
```

```
    {
        public LoginForm()
        {
            InitializeComponent();
            txtSifre.PasswordChar = '*';
            FormBorderStyle = FormBorderStyle.FixedSingle;
            btnGiris.DialogResult = DialogResult.OK;
            btnIptal.DialogResult = DialogResult.Cancel;
        }

        #region IClientFormsAuthenticationCredentialsProvider Members

        public ClientFormsAuthenticationCredentials GetCredentials()
        {
            if (this.ShowDialog() == DialogResult.OK)
            {
                return new ClientFormsAuthenticationCredentials(txtKullanici.Text,
txtSifre.Text, chkHatirla.Checked);
            }
            else
            {
                return null;
            }
        }

        #endregion
    }
}
```

Dikkat edilecek olursa **LoginForm** sınıfına **IClientFormsAuthenticationCrendentialsProvider arayüzü(Interface)** uygulanmaktadır. Bu arayüz içerisinde **ClientFormsAuthenticationCredentials**tipinden nesne örneği döndüren **GetCrendentials** isimli metod bildirimi yer almaktadır. **ClientFormsAuthenticationCredentials** sınıfının oluşturulması sırasında parametre olarak kullanıcı adı ve şifre bilgileri gönderilmektedir. Peki bu form uygulama içerisinde otomatik olarak nasıl kullanılacaktır. Bir başka deyişle uygulama çalıştığında **LoginForm** otomatik olarak nasıl başlatılacaktır. Bunun için projenin özelliklerinde aşağıdaki ekran görüntüsünde olduğu gibi küçük bir bildirim yapılması yeterlidir.



Burada opsiyonel bir **Crendential Provider** tanımlaması yapılmaktadır ve LoginForm tipi işaret edilmektedir. Dolayısıyla uygulama başlatıldığında LoginForm tipinin **Crendential Provider**olarak kullanılacağı belirtilmektedir. Bu işlemin hemen ardından **Form1** üzerinde **Load** metoduna aşağıdaki kodlamalar yapılabilir.

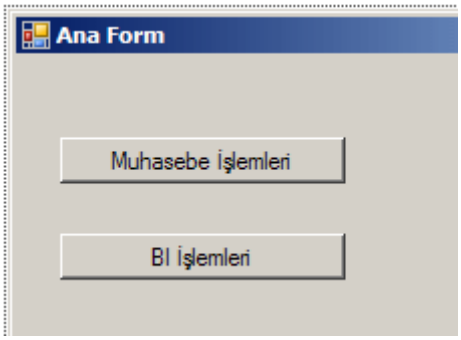
```
using System;
using System.Windows.Forms;
using System.Web.Security;
using System.Net;

namespace ClientApp
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void Form1_Load(object sender, EventArgs e)
      {
         try
         {
            if (!Membership.ValidateUser(String.Empty, String.Empty))
            {
               MessageBox.Show("Giriş Yetkiniz Yok", "Yetkisiz giriş",
MessageBoxButtons.OK, MessageBoxIcon.Stop);
               Application.Exit();
            }
         }
```

```
            catch (WebException excp)
            {
               MessageBox.Show(excp.Message);
               Application.Exit();
            }
         }
      }
   }
```

Dikkat edilecek olursa **Membership.ValidateUser** metod çağrısında String.Empty parametre değerleri kullanılmıştır. Bu son derece doğaldır nitekim bu noktada devreye **LoginForm** sınıfı girecektir. Bir başka deyişle durum uygulama **breakpoint**' ler yardımıyla **debug modda** izlendiğinde, **Form1_Load**' daki **ValidateUser** çağrısına gelindikten sonra LoginForm' un oluşturulduğu ve modal bir pencere olarak çalıştığı gözlemlenebilir.

Uygulama test edildiğinde doğru kullanıcı bilgileri girilmesi halinde ana formun(Form1) açıldığı görülür. Bununla birlikte yanlış kullanıcı bilgileri girilmesi halinde **LoginForm**' un varsayılan olarak 3 hak tanıdığı görülecektir. 3ncü girişte başarı sağlanamadığı takdirde yine program sonlanacaktır. Beni Hatırla başlıklı **CheckBox** kontrolü işaretlendiği takdirde, sonraki açılışlarda uygulamanın kullanıcı bilgilerini sormadığı görülecektir. Yani kullanıcı bileti istemci tarafında belirli süreliğine saklanacaktır. *(Bu noktada size güzel bir araştırma konusu sunabiliriz. Saklama halinde istemci bilgisi ne kadar süre ile tutulur. Bu süre nasıl özelleştirilebilir?)* Aslında şu noktada web uygulamalarında gerçekleştirdiğimiz **Forms Authentication** bazlı kullanıcı yönetiminin Windows versiyonunu yazmış bulunuyoruz.

Şimdi windows uygulamamızda role bazlı işlemler yapmaya çalışacağız. Örneğin Form1 üzerinde bulunan iki button bileşeninin sadece ilgili rollerde görünmesini sağlayabiliriz.

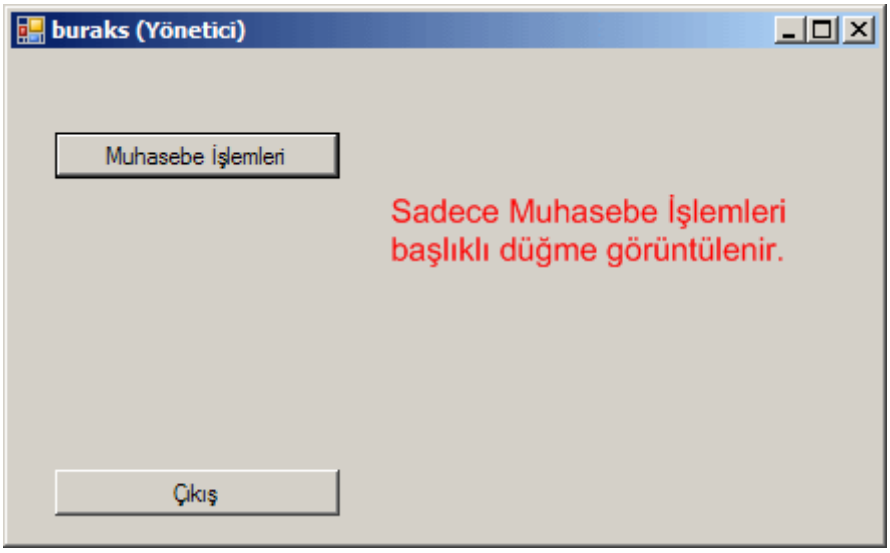


Söz gelimi **Yonetici** rolünde olan bir kullanıcı sadece **Muhasebe** işlemleri yapabilecekken, **Calisan** rolündeki bir kullanıcı ise sadece **BI** işlemlerini yapabilecektir. Bunun için örnek olarak**Form1_Load** metodu içerisinde aşağıdaki kodlamaları yapmamaız yeterlidir.

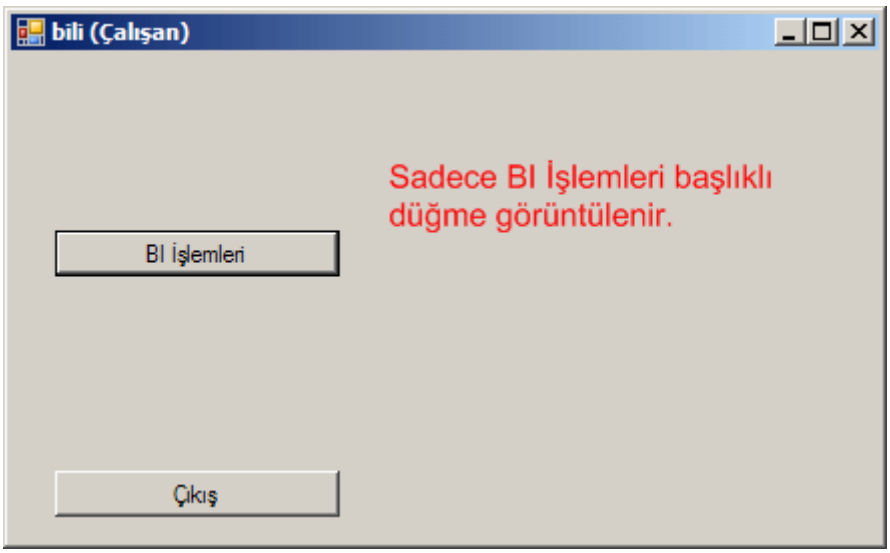
```
using System;
using System.Windows.Forms;
using System.Web.Security;
using System.Net;
using System.Web.ClientServices.Providers;
using System.Threading;
using System.Security.Principal;

namespace ClientApp
{
   public partial class Form1 : Form
   {
      public Form1()
      {
         InitializeComponent();
      }

      private void Form1_Load(object sender, EventArgs e)
      {
         try
         {
            if (!Membership.ValidateUser(String.Empty, String.Empty))
            {
               MessageBox.Show("Giriş Yetkiniz Yok", "Yetkisiz giriş",
MessageBoxButtons.OK, MessageBoxIcon.Stop);
               Application.Exit();
            }
            else
            {
               IPrincipal principal = Thread.CurrentPrincipal;

               if (principal.IsInRole("Yonetici"))
               {
                  Text = String.Format("{0} ({1})", principal.Identity.Name, "Yönetici");
                  btnMuhasebe.Visible = true;
                  btnBI.Visible = false;
               }
               else if (principal.IsInRole("Calisan"))
               {
                  Text = String.Format("{0} ({1})", principal.Identity.Name, "Çalışan");
                  btnMuhasebe.Visible = false;
                  btnBI.Visible = true;
               }
            }
         }
```

```
            catch (WebException excp)
            {
                MessageBox.Show(excp.Message);
                Application.Exit();
            }
        }

        private void btnLogout_Click(object sender, EventArgs e)
        {
            ClientFormsAuthenticationMembershipProvider prvd
=(ClientFormsAuthenticationMembershipProvider)System.Web.Security.Membershi
p.Provider;

            try
            {
                prvd.Logout();
                Application.Restart();
            }
            catch (WebException excp)
            {
                MessageBox.Show(excp.Message);
            }
        }
    }
}
```

Kullanıcının hangi rolde olduğunu öğrenmek için **Thread** sınıfı üzerinden **CurrentPrincipal** referansına gidilir. **CurrentPrincipal**, **IPrincipal** arayüzü tipinden bir refernas taşımaktadır ve **IsInRole** metodu yardımıyla çalışmakta olan uygulamanın sahibi olan kullanıcının parametre olarak belirtilen rolde olup olmadığı öğrenilebilir. Buna göre windows uygulamasının rol bazlı özellikleri ayarlanabilir. Form1 içerisinde ayrıca **Logout** işlemi içinde bir kod eklentisi yer almaktadır. Nitekim **LoginForm** üzerinde **Beni Hatırla** başlıklı**CheckBox** tıklandığı takdirde uygulama kullanıcıyı her açılışta hatırlayacaktır. **Logout** olmak istendiği durumlarda **ClientFormsAuthenticationMembershipProvider** tipine ulaşmak ve **Logout** metodunu çağırmak yeterlidir. Uygulama tekrar test edildiğinde önreğin **buraks** isimli kullanıcı ile giriş yapıldığında aşağıdaki ekran görüntüsü ile karşılaşılır.



Benzer şekilde **bili** isimli kullanıcı ile girildiğinde ise aşağıdaki ekran görüntüsü ile karşılaşılır.

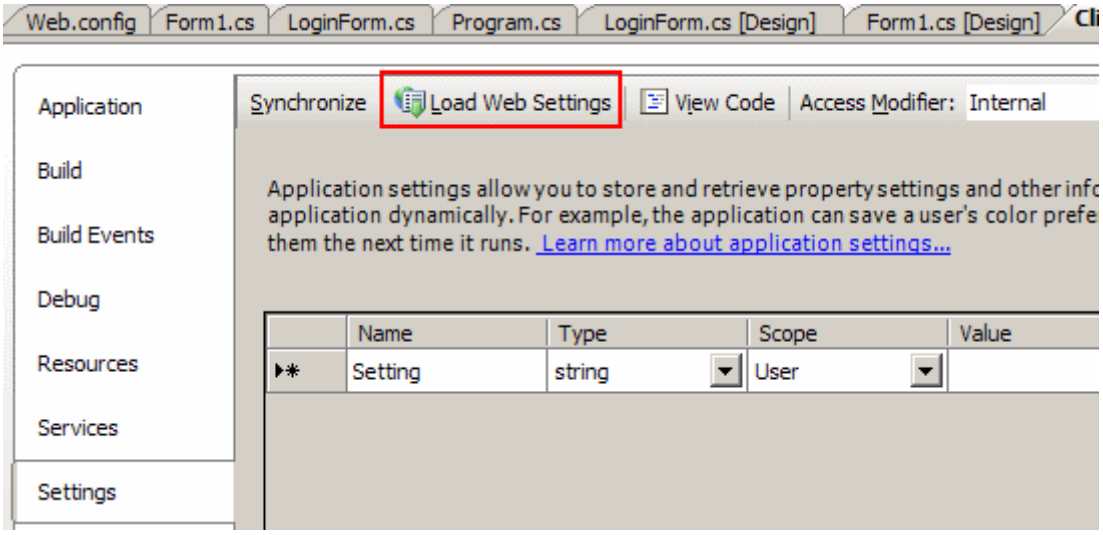


Son olarak makalemizde **profil** servisinin nasıl kullanılabileceğini ele almaya çalışalım. Burada amaç sunucu tarafında tanımlanan **özelliklerin(Properties)** her kullanıcı için farklı veriler tutacak şekilde tasarlanmasıdır. Böylece doğrulanan her kullanıcı, aynı isimli özelliğin farklı değerlerini sunucu üzerinde saklayabilir ve kendi istemci uygulaması üzerinde kullanabilir. Bu bir anlamda web uygulamalarında sıklıkla kullanılan **kişiselleştirmedir(Personalization)**. Aynı hizmet **İstemci Uygulama Servisleri** sayesinde, windows tabanlı programlara da uygulanabilir. Dilerseniz bu basit işlemi nasıl yapabileceğimize bakalım. Öncelikli olarak sunucu tarafında yer alan **web.config** dosyası içerisinde basit bir kaç bildirimi aşağıdaki gibi yapmamız gerekmektedir.

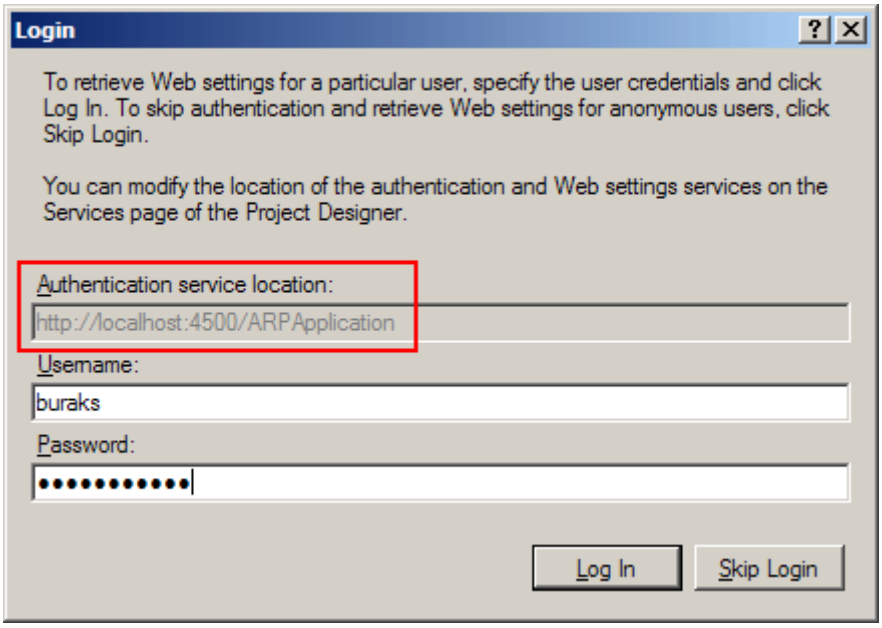
```
<system.web.extensions>
 <scripting>
  <webServices>
   <authenticationService enabled="true"/>
   <roleService enabled="true"/>
   <profileService enabled="true"
                   readAccessProperties="GirisMesaji,SonGirisZamani"
                   writeAccessProperties="GirisMesaji,SonGirisZamani"/>
  </webServices>
 </scripting>
</system.web.extensions>

<system.web>

 <roleManager enabled="true"/>
 <profile enabled="true">
  <properties>
   <add
    name="GirisMesaji" defaultValue="Giriş Mesajınız" type="System.String"
    allowAnonymous="false" readOnly="false" serializeAs="String"/>
   <add
    name="SonGirisZamani" type="System.DateTime"
    allowAnonymous="false" readOnly="false" serializeAs="String"/>
  </properties>
 </profile>

```

Dikkat edeceğiniz üzere **profile** elementi altında yer alan **properties** alt boğumu(Child Node) içerisinde iki adet özellik bildirimi yapılmaktadır. **GirisMesaji** isimli özellik **string** tipindendir. Varsayılan bir değeri vardır. **String** formatta serileştirilmektedir. **İsimsiz kullanıcılar(Anonymous Users)** için bu özelliğe değer atanmasına izin verilmemektedir. Ayrıca **yanlız okunabilir(readonly)** bir özellikte değildir. Benzer kriterler **SonGirisZamani** isimli ikinci özellik içinde yer almaktadır. Ancak **SonGirisZamani** isimli özelliğin veri tipi **DateTime**' dır. **profileService**elementi içerisinde ise söz konusu özelliklere okuma ve yazma hakları, **readAccessProperties** ve **writeAccessProperties nitelikleri(Attribute)** ile verilmektedir. Burada birden fazla özelliğin aralarına virgüller konularak belirtilebildiğinde dikkat edilmelidir. Bu işlemler sonrasında sunucu tarafında her istemci için farklı değerlere sahip olabilecek özellik tanımlamaları yapılmış olmaktadır.

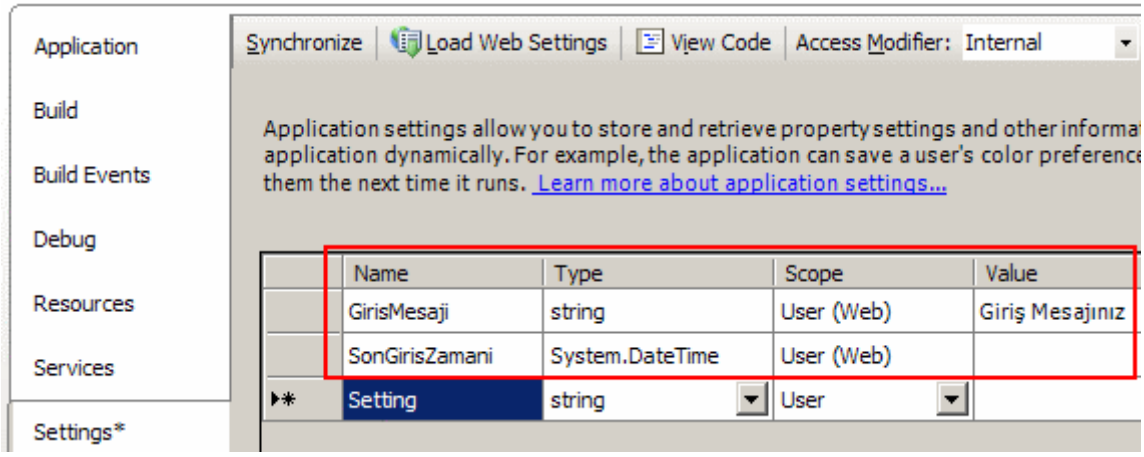
Şimdi istemci tarafında bazı işlemlerin yapılması gerekmektedir. Öncelikli olarak istemci uygulamada **Properties->Settings** kısmına geçilir ve **Load Web Settings** düğmesine basılır.



Bu işlemin ardından sunucu tarafındaki profile servisine bağlanılmak istenir ki var olan geçerli bir kullanıcı ile bu gerçekleştirilebilir.



Eğer bağlantı başarılı bir şekilde sağlanırsa, sunucu tarafındaki **web.config** dosyasında tanımlanan profil özelliklerinin istemci uygulamadaki **settings** kısmına eklendiği açık bir şekilde görülebilir.



Bir başka deyişle artık istemci tarafında **GirisMesaji** ve **SonGirisZamani** isimli özelliklere tip bazında erişilebilecektir. Kod tarafında, formun yüklenmesi ile birlikte bu özelliklerin değerleri okunabilir ve herhangibir amaçla kullanılabilir. Form kapanırken veya uygulamadan çıkılırkende istenirse, bu özelliklerin yeni değerlerinin set edilip kaydedilmesi, bir başka deyişle servis tarafındaki **aspnetdb.mdf** veritabanında yer alan **aspnet_Profile** tablosuna yazdırılması sağlanabilir. Bu örneğimizde söz konusu senaryoyu çok basit bir şekilde değerlendirmeye çalışacağız. Örneğin Form1 in Load metodunu aşağıdaki gibi güncellediğimizi düşünelim.

```
private void Form1_Load(object sender, EventArgs e)
{
   try
   {
      if (!Membership.ValidateUser(String.Empty, String.Empty))
      {
         MessageBox.Show("Giriş Yetkiniz Yok", "Yetkisiz giriş",
MessageBoxButtons.OK, MessageBoxIcon.Stop);
         Application.Exit();
      }
      else
      {
         txtGirisMesaji.DataBindings.Add("Text", Properties.Settings.Default,
"GirisMesaji");
         lblSonGirisZamani.Text =
Properties.Settings.Default["SonGirisZamani"].ToString();

         IPrincipal principal = Thread.CurrentPrincipal;

         if (principal.IsInRole("Yonetici"))
         {
            Text = String.Format("{0} ({1})", principal.Identity.Name, "Yönetici");
            btnMuhasebe.Visible = true;
            btnBI.Visible = false;
```

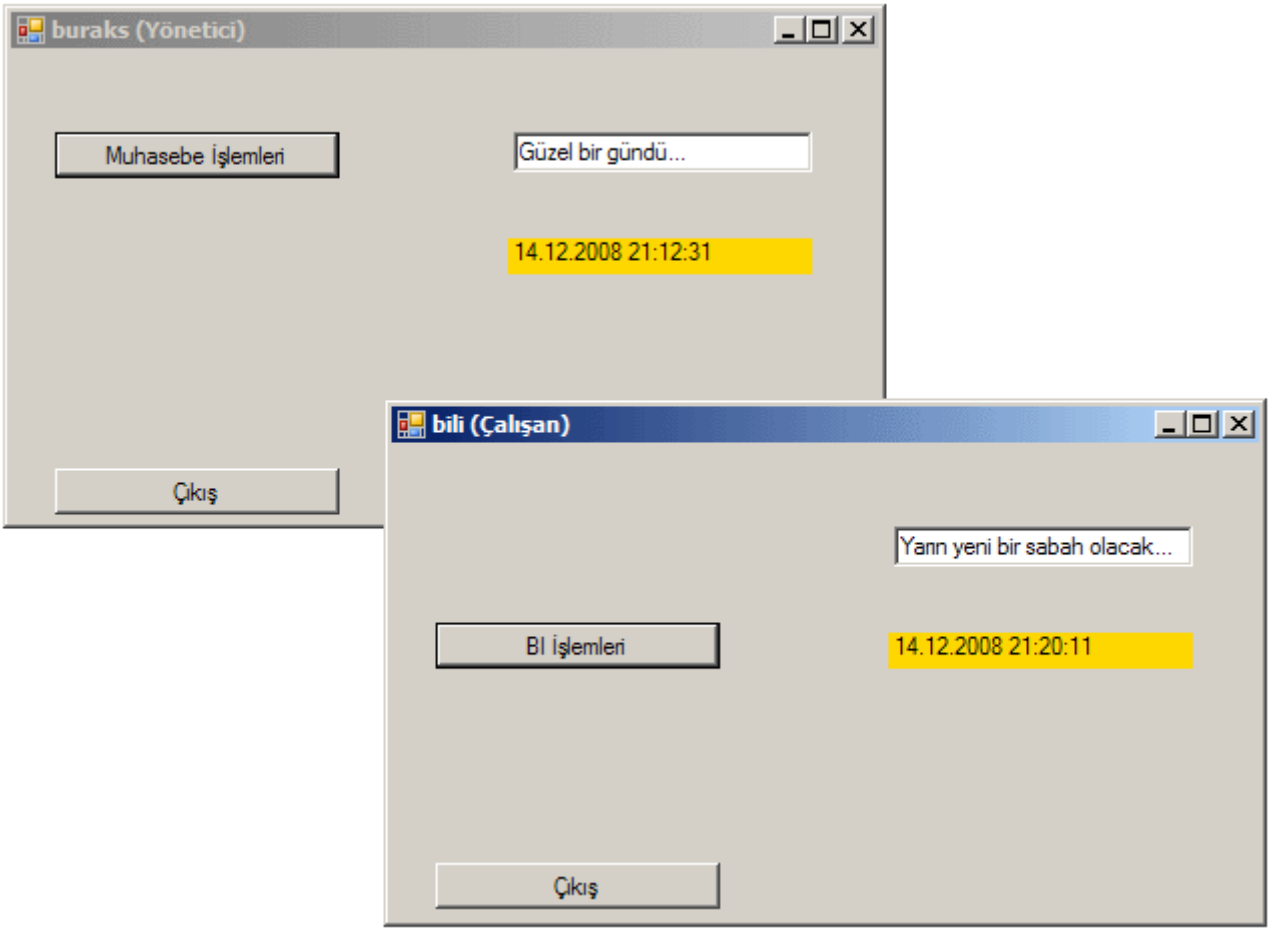
```
            }
            else if (principal.IsInRole("Calisan"))
            {
               Text = String.Format("{0} ({1})", principal.Identity.Name, "Çalışan");
               btnMuhasebe.Visible = false;
               btnBI.Visible = true;
            }
         }
      }
      catch (WebException excp)
      {
         MessageBox.Show(excp.Message);
         Application.Exit();
      }
   }
```

Dikkat edilecek olursa **txtGirisMesaji** isimli TextBox kontrolüne **veri bağlaması(DataBinding)** sırasında **Properties.Settings.Default** veri kaynağı(Data Source) olarak gösterilmiştir. Son parametrede ise **Settings** kısmında tanımlı olan özellik adları verilmektedir. **txtGirisMesaji** isimli **TextBox** kontrolü için **çift-yönlü veri bağlama(two-way databinding)** söz konusudur. Böylece kontrol içerisinde veri değiştirildiğinde söz konusu değişiklik özellik tarafınada otomatik olarak yansıtılacaktır. **lblSonGirisZamani** isimli **Label** kontrolü ise **tek yönlü(one-way)** olarak **SonGirisZamani** özelliğine bağlanmıştır.

Form' dan çıkılırken bu verilerin servis tarafına gönderilmesi gerekir ki bir sonraki girişte son değerler kullanılabilsin. Bu durumda örneğin **Form1_Closing** olay metodu içerisinde aşağıdaki kodlamalar yapılabilir.

```
private void Form1_FormClosing(object sender, FormClosingEventArgs e)
{
   if (Thread.CurrentPrincipal.Identity.AuthenticationType.Equals("ClientForms"))
   {
      try
      {
         Properties.Settings.Default["SonGirisZamani"] = DateTime.Now;
         Properties.Settings.Default.Save();
      }
      catch(WebException excp)
      {
         MessageBox.Show(excp.Message);
      }
   }
}
```

İlk olarak kullanıcının **ClientForms** tipinde **authentication** gerçekleştirip gerçekleştirmediğine bakılır. Bir başka deyişle kullanıcının **Client Application Services** kullanarak **Login**olup olmadığı anlaşılmaya çalışılmaktadır. Eğer bu şekilde bir bağlantı sağlanmış ise **SonGirisZamani** özelliğinin değeri değiştirilir ve **Save** metodu çağırılır. Dikkat edileceği üzere **GirisMesaji** isimli profil özelliği için bir atama yapılmamıştır. Bunun sebebi iki yönlü veri bağlama işleminin **txtGirisMesaji** kontrolü için kullanılıyor olmasıdır. Bu metodun çalışması sırasında bazı hatalar oluşabilir. Söz gelimi **Save** metodu çalıştığı sırada istemcinin **cookie** bilgisi silinmiş olabilir. Yada kullanıcı**Logout** olmuş durumdadır ancak halen daha uygulama içerisindedir ve Save metodu çağırılmıştır. Dolayısıyla işlemin gerçekleştirilebilmesi için tekrardan **Login** işleminin yapılması gerekmektedir. Diğer taraftan kayıt işlemi sırasında servis ile olan bağlantının kurulamama ihtimalide vardır. Bu gibi durumlar ele alınaraktan daha güçlü bir profil kaydetme süreci oluşturulabilir. (*Burada nasıl bir tedbir alınabileceğinide bir araştırma konusu haline getirebilirsiniz)* Artık uygulamayı bu haliyle test edebiliriz. Bilhassa aynı uygulamadan iki örnek başlatıp iki farklı kullanıcı ile girilmesini tavsiye ederim. Bu durumda her iki kullanıcı içinde farklı profil verileri tutulduğu gözlemlenebilir. Aşağıdaki test ekranlarında olduğu gibi.



Bu testin sonrasında sunucu tarafındaki **aspnetdb.mdf** veritabanında yer alan **aspnet_Profile** tablosu içeriğine bakıldığında her iki kullanıcı içinde ilgili satırların oluşturulduğu görülebilir.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde **İstemci Uygulama Servislerini(Client Application Service)**, **Visual Studio 2008** ortamı üzerinde basit bir şekilde ele alarak, **Windows**tabanlı istemcilerde **Authentication,Roles** ve **Profile** hizmetlerinin nasıl ele alınabileceğini basit bir şekilde incelemeye çalıştık. Söz konusu kriterlerin birer servis olarak **.Net Framework** içerisine dahil edilmesi sonrasında herhangibir .**Net** istemcisinin bunları kullanabileceğini anladık. Burada **.Net** tabanlı istemcilerin **JSON** formatında mesajlaşma yaptığını ancak .**Net** dışı uygulamalarında **SOAP 1.1** formatını kullanarak bu servisleri ele alabileceklerine değindik ki bunu ilerleyen makalelerimizde ele almaya çalışacağız.

Makalemizde geliştirmeye çalıştığımız örneğin bir benzerinin adım adım yapılışını C#Nedir? adresinde yer alan görsel derstende eş zamanlı olarak takip edebilirsiniz. Tekradan görüşünceye dek hepinize mutlu günler dilerim.

Ado.Net Data Services Ders Notları - Optimistic Concurrency (2008-10-30T05:45:00)

ado.net data services,

İstemci-Sunucu(Client-Server) bazlı uygulamalar göz önüne alındığında, istemcilerin aynı veriler üzerinde birbirlerinden habersiz şekilde değişiklikler yapabilme ihtimali oldukça meşhur bir vaka olarak bilinmektedir. özellikle .Net tarafında **bağlantısız katman(Disconnected Layer)** uygulamalarında bu tip vakalar son derece önemlidir. Zaman zaman bu tip vakalar ile mücadele etmek ve tedbirler almak gerekir. Vaka aslında şu şekilde ifade edilebilir; "sunucu üzerinden aynı veri içeriklerini çeken istemci programlar, sunucu ile bağlantılarını kestikten sonra kendi uygulama alanları üzerine aldıkları verilerde değişiklik yapabilirler. Ancak bu noktada sunucu ile sürekli bir bağlantıları olmadığından, başka istemcilerin aynı veriler üzerinde değişiklikler yapıp yapmadıklarını tam olarak bilemezler. Bu sebepten aynı veriler üzerinde birbirlerinden habersiz olacak şekilde yaptıkları değişiklikleri sunucuya gönderebilirler." İşte bu noktada sunucu tarafında durumun nasıl ele alınacağı önem kazanır. Bu amaçla çeşitli denetleme mekanizmaları kullanılabilir. Bu yazımızda hepinizin kulağında bol bol **Optimistic Concurrency**kelimelerinin çınlayacağını şimdiden söyleyebilirim.

Bahsetmiş olduğumuz bu vaka bazen görmezden gelinebilecek olmasına karşın çoğu durumda kontrol altına alınması gereken bir sorun olarak değerlendirilir. İşin içerisine birde değişikliklerin sunucu üzerindeki veritabanına gönderilmesi sırasında devreye alınan **Transaction**' lar girerse, vaka kendi içerisinde dahada karmaşıklaşır. Ancak bizim şu andaistemediğimiz tek şey bu vakayı dahada karıştırmaktır. Bunlara karşın söz konusu vakada çözümsel olarak **optimistic(iyimser)** yada **pesimistic(kötümser)** yaklaşımların uygulanabilir olduklarınıda bilmek gerekir. Peki bu durumun Ado.Net Data Service' ler ile olan bağlantısı nedir? Herşeyden önce Ado.Net Data Service hangi modeli baz alır? Baz aldığı model nasıl uygulanır?

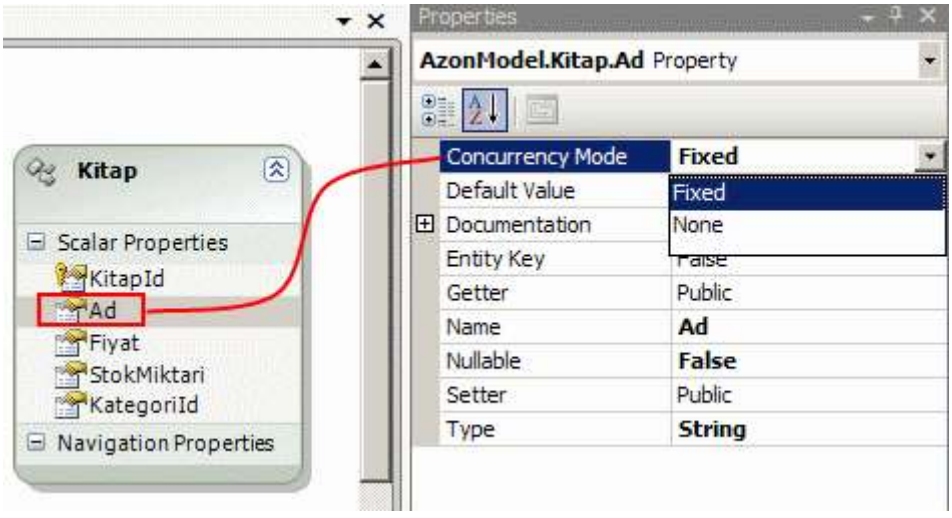
Bildiğiniz gibi bu ana kadarki ders notlarımızda ve görsel derslerimizde **Ado.Net Data Service**' lerin, web programlama modeline uygun olarak çalıştığını ve **EDM(Entity Data Model)** yada**Custom LINQ Provider** gibi katmanlar üzerinde veri sunumu gerçekleştirdiğine değindik. Ayrıca, **Ado.Net Data Service**' ler bir sunucu uygulama üzerinden **host** edilmek zorunda olmakla birlikte, bunları tüketen farklı istemci uygulamalar yazılabilmektedir. Bir başka deyişle tipik bir **istemci-sunucu** modeli söz konusudur. Bunlara ilaveten işin içerisinde, istemci tarafına çekilebilen veriler ve tabiki **CRUD(CreateRetriveUpdateDelete)** operasyonları söz konusudur. Bu operasyonlar içerisinde yer alan **CUD** fonksiyonellikleri ve istemcilerin veriyi kendi uygulama alanlarına çektikten sonra sunucu ile herhangibir bağlantılarının kalmayışı, istemcilerin aynı veriler üzerinde birbirlerinden habersiz değişikliker yapabilecekleri sonucunu doğurmaktadır. Peki bu tarz bir sorun ile nasıl mücadele edilebilir? **Ado.Net Data Service** çözüm olarak, **Optimistic Concurrency** yaklaşımının ele alınmasına izin vermektedir.

NOT : Optimistic Concurrency *yaklaşımına göre, istemci tarafına çekilen veriler güncelleştirilmek üzere sunucu tarafına gönderildiklerinde ilk hali ile karşılaştırılırlar. Böylece ilk okumadan sonra başka bir istemcinin aynı veriyi değiştirip değiştirmediği kontrol altına alınabilir. Eğer bir değişiklik var ise istemcinin bu konuda uyarılması gerekir. Bu uyarıl üzerinde çalışılmakta olan modele göre çoğunlukla bir* ***istisna(Exception)*** *olarak ele alınır. Söz gelimi* ***Ado.Net*** *tarafından bildiğimiz* ***DBConcurrencyViolation*** *bu tip bir istisnadır. Tabi işin içerisine servis yönelimli bir çözüm girdiğinde bu, çoğunlukla bir* ***Fault Message*** *formatına uygun olacak şekilde hata bilgisi içeren bir* ***XML*** *verisidir. Eğer veriler başkası tarafından değiştirilmemişse tabiki güncelleme işlemi sunucu tarafındada onaylanacaktır.*

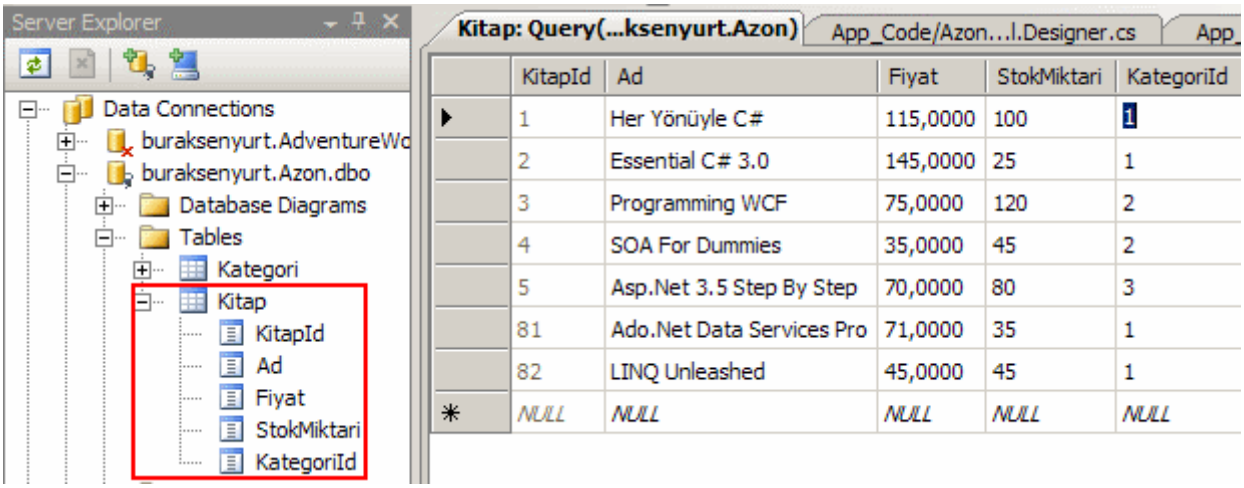
Peki **Ado.Net Data Service** tarafında bu yaklaşım nasıl ele alınmaktadır? Artık bu noktadan sonra adım adım basit bir örnek üzerinde ilerlenilmesinde yarar olacağı kanısındayım. örneğimizi geliştirirken en büyük yardımcılarımızdan biriside **Fiddler** isimli **HTTP Debugging Proxy** aracı olacaktır. Nitekim çakışma olması halinde istemciler ve sunucu arasında gidip gelen **HTTP** paketlerinin incelenmesi gerekmektedir. Test senaryomuz son derece basittir. Aynı veri satırı üzerinde değişiklik yapacak en az iki istemci uygulamanın çalıştırılması ve bu esnada oluşacak **istisnaların(Exceptions)** ve **HTTP** paketlerinin izlenmesi hedeflenmektedir. Bunların yanında SQL tarafında neler olduğunu gözlemlemek adınada **SQL Server Profiler** aracından yararlanmamız gerekecektir. Tabi öncelikli olarak basit bir **WCF Service** uygulaması geliştirerek başlamalıyız. Söz konusu servisimiz daha önceden geliştirdiğimiz **Azon** isimli veritabanını ve içerisinde bir kaç satır veri içeren **Kitap** isimli tabloyu istemci tarafına sunacak şekilde geliştirilecektir. Bu nedenle **WCF Service** uygulamımız için gerekli olan ön hazırlıklar aşağıdaki gibidir.

EDM(Entity Data Model) içeriğimiz;

Burada hemen bir özelliği vurgulamak gerekiyor. **EDM** diagramında yer alan **Kitap Entity** tipi içerisinde yer alan özelliklerin her birisi için **Concurrency Mode** isimli bir özellik yer almaktadır. **Properties**penceresinden ulaşılabilen bu özelliğin değeri varsayılan olarak **None** şeklindedir. Buna **Mixed** değerini vermemiz halinde **Optimistic Concurrency** için söz konusu özellik değerlerinin hesaba katılacağı belirtilmiş olunur. örneğimizde bu amaçla **Ad** ve **Fiyat** özelliklerinin **Concurrency Mode** değerleri **Mixed** olarak belirlenmiştir. Ki senaryomuzda sadece bu alanların değerleri için**Optimistic Concurreny** kontrolü yapılacaktır.



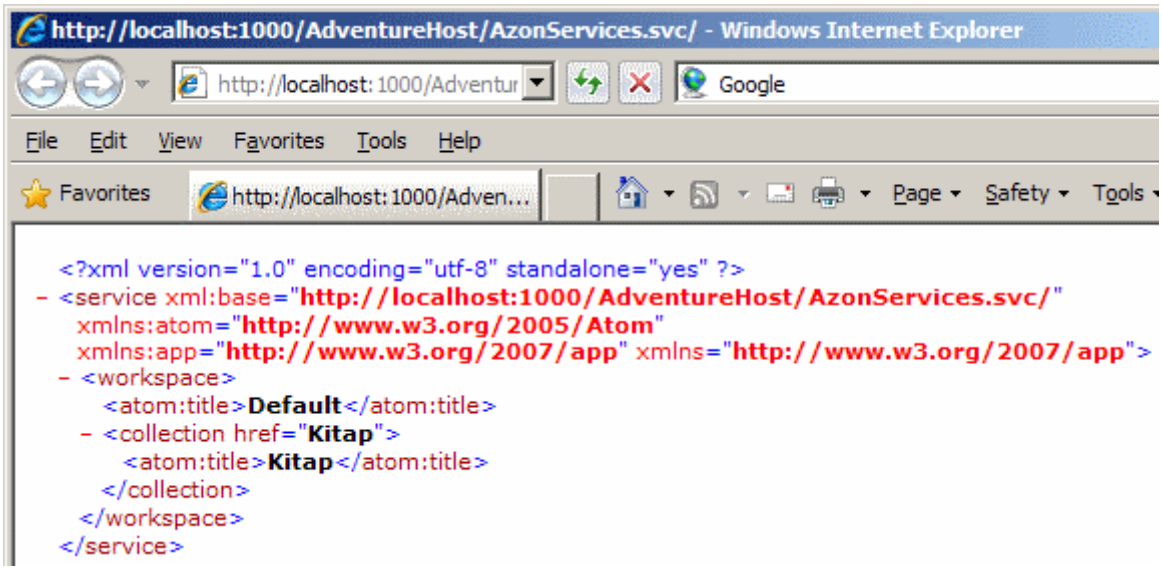
Kitap tablosuna ait bir kaç satırlık veri içeriği;



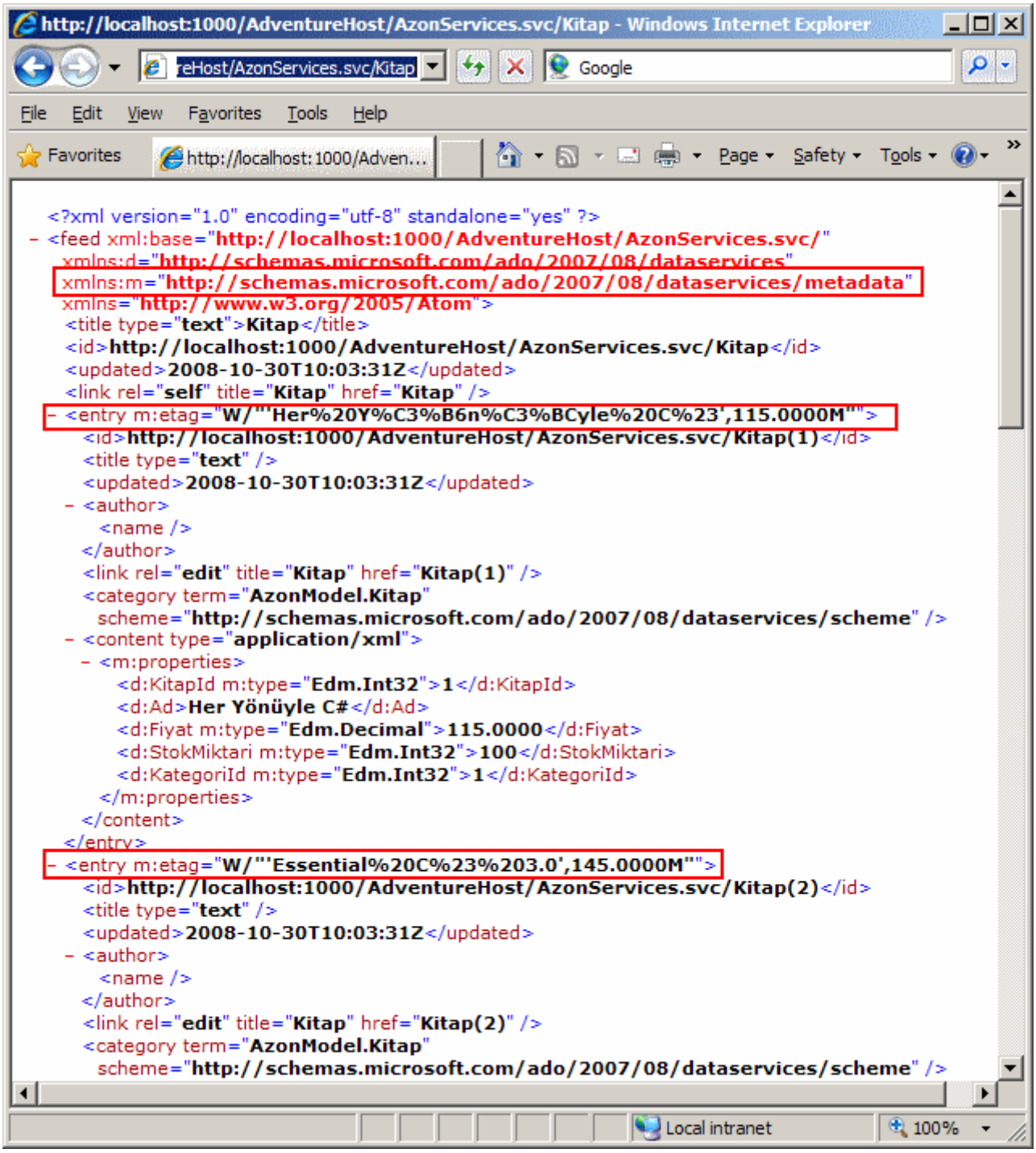
AzonServices.svc.cs içeriğimiz;

```
using System;
using System.Data.Services;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel.Web;
using AzonModel;

public class AzonServices
   : DataService<AzonEntities>
{
   public static void InitializeService(IDataServiceConfiguration config)
   {
      config.SetEntitySetAccessRule("*", EntitySetRights.All);
   }
}
```

Tahmin edeceğiniz gibi istemci tarafında **CRUD** operasyonları yapılabileceğinden **EntitySetRights.All enum** sabiti değeri kullanılmıştır. Buraya kadar geldikten sonra **AzonServices.svc** isimli**Ado.Net Data Service** örneğinin çalıştığından emin olmakta yarar vardır. Bunun için servisi basit bir tarayıcı uygulama içerisinde açmamız yeterli olacaktır. Aşağıdakine benzer bir ekran görüntüsü ile karşılaşırız.



Yanlız burada **Kitap entity** içeriği talep edildiğinde, **atom** formatında üretilen **XML** verisinde yeni bir **attribute** tanımlaması karşımıza gelecektir.



Her bir **entity** elementi içerisinde **m:etag** isimli bir **attribute(nitelik)** tanımlanmıştır. **m** takma adlı isim alanına sahip bu nitelikler içerisinde her bir kitabın **Ad** ve **Fiyat** bilgilerinin yer aldığına dikkat edin. Hatırlayacağınız gibi, **EDM** diagramında bu özelliklerin **Concurrency Mode** değerlerini **Mixed** olarak belirlemiştik. Bu nedenle çakışma kontrolü için ilgili özelliklerin değerleri **XML** çıktısına dahil edilmiştir. Bundan dolayı **etag** yada **entitytag** adı verilen nitelikler içerisinde taşınan özellik değerleri, çakışma kontrolü için istemci ile sunucu arasında gidip gelen paketlerde önem kazanmaktadır.

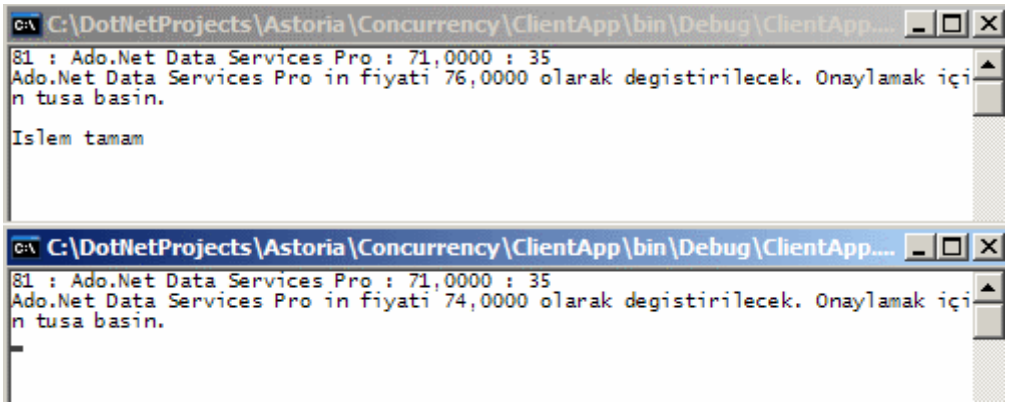
Gelelim istemci uygulamamıza. Amacımız **Optimistic Concurrency** modelini Ado.Net Data Service' ler üzerinde incelemek olduğundan şimdilik işimizi görecek basit bir program yazmamız yeterli olacaktır. Sanıyorumki ne demek istediğimi anladınız :) Basit bir **Console Application** geliştiriyoruz. Uygulamamıza aynı **solution** içerisinde yer alan servisimizide ekledikten sonra istemci uygulama kodlarımızı aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ClientApp.AzonServiceReference;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         // Proxy nesne örneği oluşturulur.
         AzonEntities proxy = new AzonEntities(new
Uri("http://buraksenyurt:1000/AdventureHost/AzonServices.svc"));

         // Concurrency testi için ID si 81 olan Kitap verisi çekilir
         Kitap kitap81 = (from k in proxy.Kitap
                          where k.KitapId==81
                             select k).First<Kitap>();

         // 81 nolu ID' ye ait kitap bilgileri gösterilir
         Console.WriteLine("{0} : {1} : {2} :
{3}",kitap81.KitapId,kitap81.Ad,kitap81.Fiyat,kitap81.StokMiktari);

         // Test amacıyla rastgele bir artış değeri üretilir ve 81 nolu Kitap nesne örneğinin
Fiyat değeri değiştirilir
         Random rnd = new Random();
         int yeniFiyatArtisi = rnd.Next(1, 10);
         kitap81.Fiyat = kitap81.Fiyat + yeniFiyatArtisi;

         // Burası test noktası
         Console.WriteLine("{0} in fiyatı {1} olarak değiştirilecek. Onaylamak için tuşa
basın.", kitap81.Ad, kitap81.Fiyat);
         Console.ReadLine();

         // Nesne güncellenir
         proxy.UpdateObject(kitap81);
```

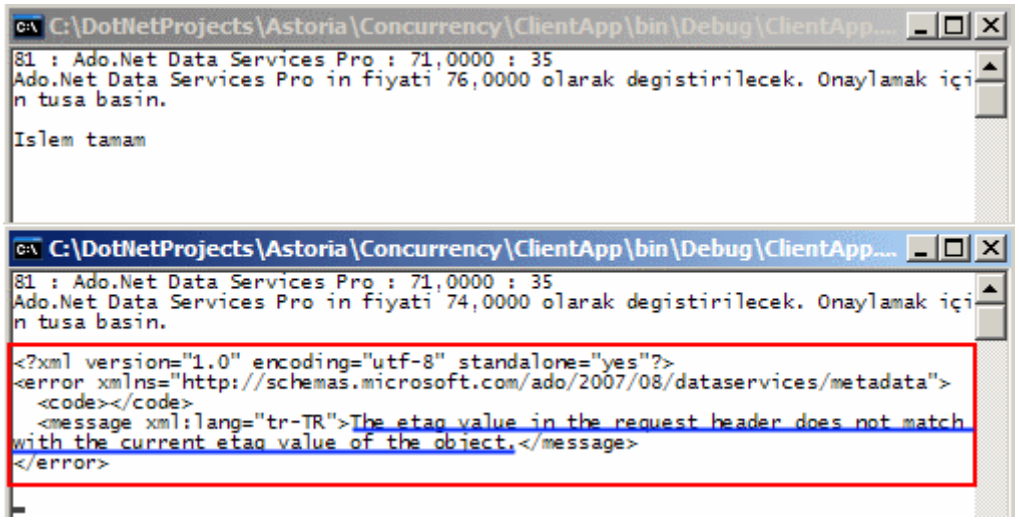
```
        // Bir istisna bloğu içerisinde SaveChanges metodu çağırılır.
        try
        {
           proxy.SaveChanges();
           Console.WriteLine("İşlem tamam");
        }
        catch (Exception excp)
        {
           // Burada beklenen hata mesajı InnerException içerisinde gelir
           Console.WriteLine(excp.InnerException.Message);
        }

        Console.ReadLine();
     }
  }
}
```

Kısaca istemci uygulamada neler yaptığımızdan bahsedelim. öncelikli olarak **proxy** nesnesi örnekleniyor. **Fiddler** aracını **Web Development Server** üzerinden çalıştıracağımız için daha önceki makalemizde bahsettiğimiz gibi **1000** numaralı portu ve makine adını kullanıyoruz. İlerleyen kısımlarda test amacıyla **KitapID** değeri **81** olan kitap bilgilerini istemci tarafına**LINQ** sorgusu üzerinden çekiyoruz. Elde edilen **Kitap** nesne örneğinin **Fiyat** özelliğini **Random** sınıfı ile üretilen rastgele bir değer kadar arttırıyoruz. Sonrasında ise bir**Console.ReadLine** çağrısı görmekteyiz. Bu çağrının olduğu yer aynı uygulamadan aynı makinede birden fazla çalıştırdığımızda test yapmamızı kolaylaştıracaktır. Bu çağrının ardından **UpdateObject** metodu işletiliyor. Sonrasında ise **try...catch** blokları içerisinde alınmış olan bir **SaveChanges** metodu çağrısı görüyoruz. Bu çağrı istemci tarafındaki güncelleştirmenin sunucuya iletilmesine neden oluyor. İşte bu noktada eğer bir çakışma söz konusu ise istemci tarafına bir **exception** döndürülecektir. Gelin hemen bir test yaparak işe başlayalım. Testimizde aynı uygulamadan iki adet çalıştırıyor ve birinde değişiklikleri sunucuya gönderdikten sonra , ikincisi içinde aynı işlemi yapmayı deniyoruz. Sonuç olarak bu test sonrasında aşağıdaki ekran görüntüleri oluşacaktır.

Her iki uygulama açılıp ilk uygulamadaki güncellemeler servis tarafında gönderildiğinde;

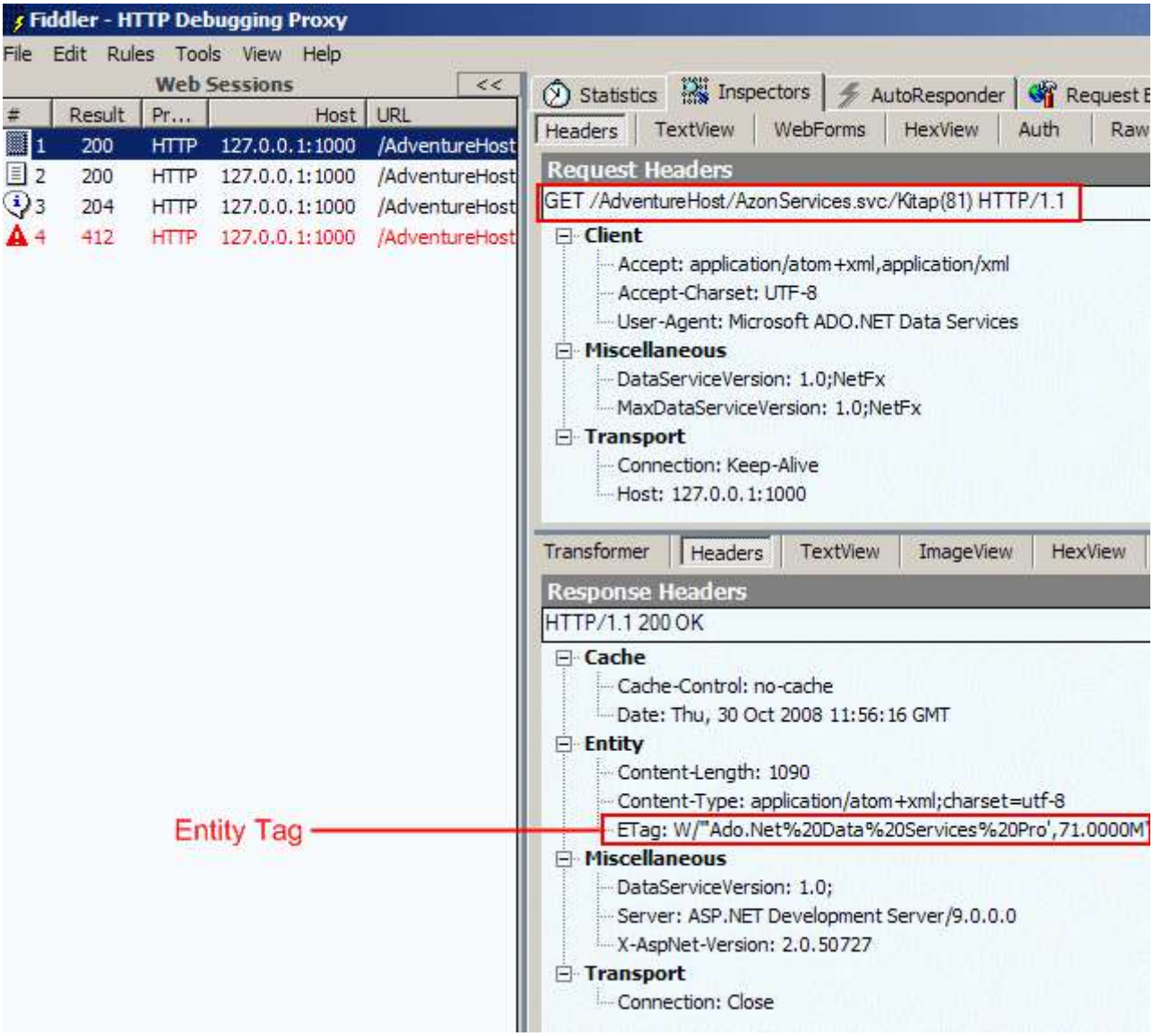


İkinci uygulamada devam edilip yeni değerler ile aynı veri güncellenmek üzere servise gönderildiğinde;

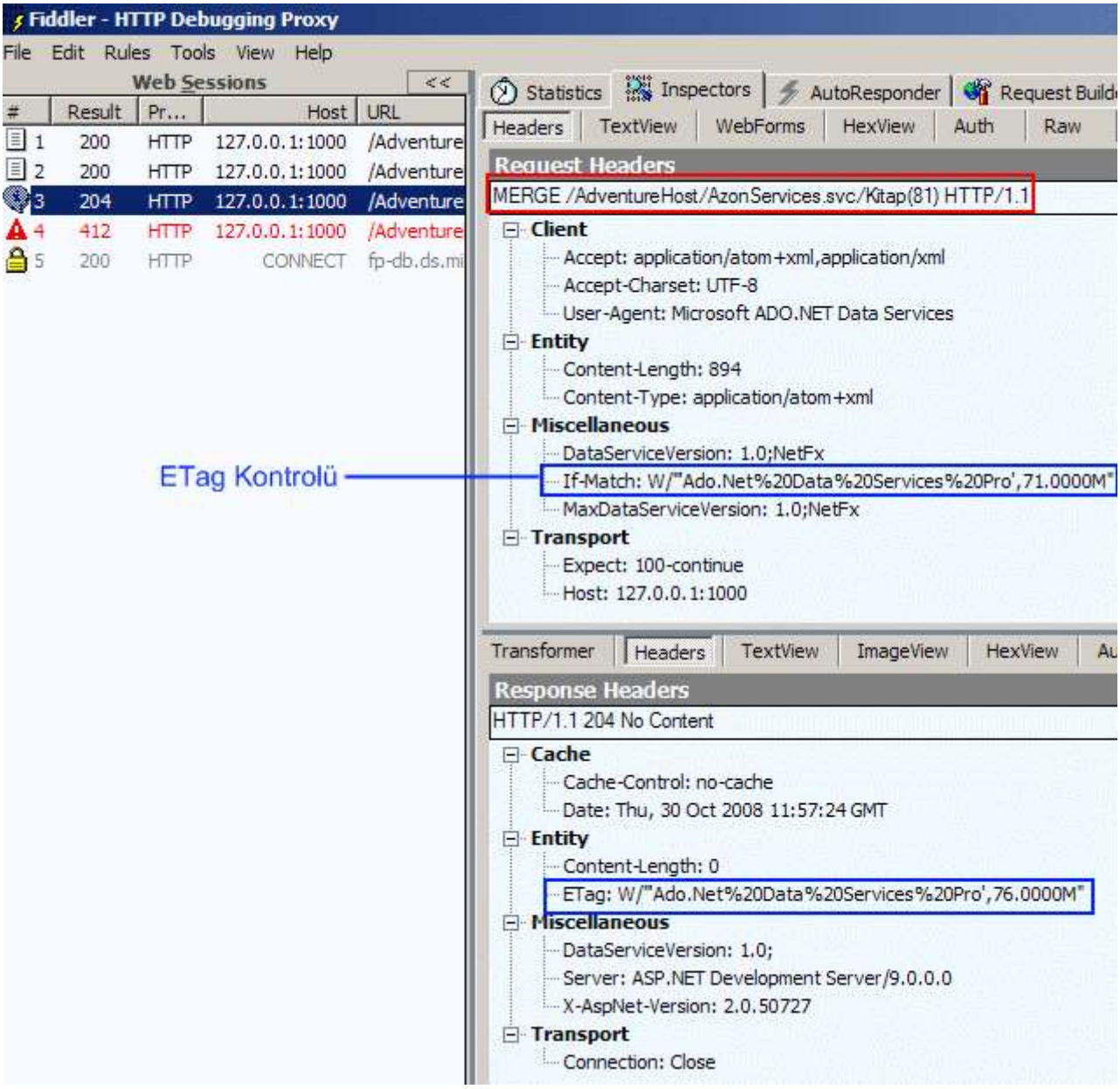


Görüldüğü gibi ikinci uygulamaya bir adet **Fault Exception** gönderilmiş ve **etag** değerinin **Request Header**' daki güncel **etag** değeri ile uyuşmadığı belirtilmiştir. Bir başka deyişle ikinci uygulamanın güncelleştirmek istediği satır başkası tarafından güncellenmiştir.

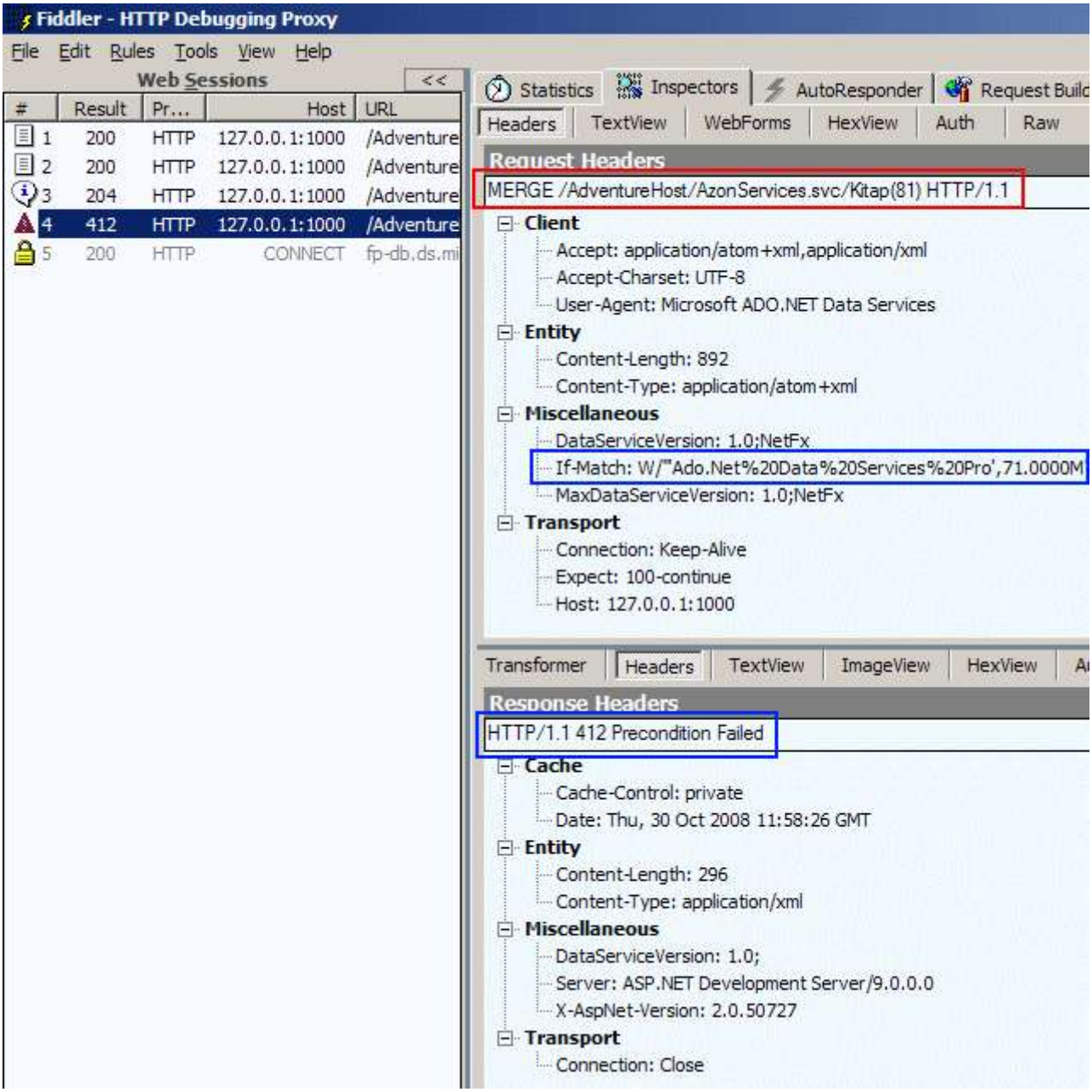
Hemen **Fiddler** aracı ile arka planda olanları inceleyelim. Birinci uygulama çalıştırıldığında ilk olarak **81** numaralı **KitapID** değerine sahip veri çekilmektedir. Bu tipik olarak **HTTP Get**çağrısıdır ve **Request(İstek)** ile **Response(Cevap)** paketlerine ait **Header(Başlık)** içerikleri aşağıdaki ekran görüntüsünde olduğu gibidir.



Standart bir iletişim olduğu gözlemlenmekle birlikte **Respons Header** içerisinde **ETag** isimli bir bilgi daha yer almaktadır. Bu bilgiye göre ilk uygulamaya çekilen kitap satırında **Ad** değeri **Ado.Net Data Services Pro**, **Fiyat** değeri ise **71.0000** dır. İkinci uygulamada aynı talepte bulunacaktır ve yine yukarıdaki ekran görüntüsünde yer alan paket alışverişi söz konusudur. Bu durumda ikinci uygulama için söz konusu olan **Response Header** içerisindeki **ETag** değeride aynıdır. Gelelim 3ncü paket alışverişine.



Request Header içerisinde **If-Match** isimli bir bilgi yer aldığı görülmektedir. Bu bilgiye göre **Ado.Net Data Services Pro** ve **71.0000** değerlerinin doğrulanması istenmektedir. Şu durumda başka bir uygulama yada veritabanı üzerinden doğrudan olacak şekilde, **81** numaları kayıtta bir değişiklik olmadığından **Response Headers** içerisinde söz konusu verinin güncellenen değerlerine ait bilgiler istemci tarafına gönderilmektedir. Bir başka deyişle şimdi **ETag** değerinin içeriği **Ado.Net Data Services Pro** ve **76.0000** dır. Dikkat edileceği üzere **Fiyat**değişmiştir. Ancak arkada unutmamamız gereken ikinci uygulamamız vardır. Bu uygulmada tuşa basıp devam edildiğinde, **Fiddler** üzerinden 4ncü paket alışverişi aşağıdaki gibi yakalanmaktadır.



Bu kez **Request Header** içerisindeki **ETag** değeri **71.0000** değeri için talepte bulunur. Ancak az önceki uygulamada **ETag** değerinde yer alan **Fiyat 76.0000** olarak değişmiştir. Dolayısıyla **If-Match** karşılaştırması başarılı olmayacaktır. Bunun sonucu olarakta geriye **HTTP/1.1 412** kodu **(Precondition Failed)** döner.

NOT : *HTTP 1.1 durum kodları(Status Codes) ve açıklamaları için WC3 üzerinden yayınlanan adresinden bilgi alabilirsiniz.*

çok doğal olarak bu hata için istemci tarafına bir **exception** bilgisi gönderilmiştir. Bu arada **SQL** tarafında neler olduğunuda bilmekte yarar vardır. Aynı süreç **SQL Server Profiler** üzerinden incelendiğinde ilk uygulamanın güncelleştirme işleminden hemen önce **81** numaları **KitapID** için bir **Select** sorgusu çalıştırıldığı sonrasında ise aşağıdaki **SQL** ifadesinin devreye girdiği görülür.

exec sp_executesql N'update [dbo].[Kitap]
set [Ad] = @0, [Fiyat] = @1, [StokMiktari] = @2, [KategoriId] = @3
where ((([KitapId] = @4) and ([Ad] = @5)) and ([Fiyat] = @6))
',N'@0 nvarchar(25),@1 decimal(19,4),@2 int,@3 int,@4 int,@5 nvarchar(25),@6 decimal(19,4)',@0=N'Ado.Net Data Services Pro',@1=76.0000,@2=35,@3=1,**@4=81,@5=N'Ado.Net Data Services Pro',@6=71.0000**

Dikkat edelim! **Where** ifadesinden sonra **KitapID**, **Ad** ve **Fiyat** alanları hesaba katılmıştır. Bunun en büyük nedeni **EDM** diagramında **Concurrency Mode** değeri **Fixed** olarak belirlenen özelliklerdir. Dolayısıyla bu **Where** kriteri bozulmadığından güncelleştirme işlemi yapılmıştır. Oysaki ikinci uygulama tuşa basılarak devam ettirildiğinde, **Update** sorgusunun çalıştırılmadığı onun yerine **81** nolu **KitapID** için bir **Select** sorgusunun işlediği görülür. Akabindede zaten **ETag** verilerindeki uyuşmazlık nedeni ile istemciye bir hata mesajı gönderilmektedir. Buraya kadar anlattıklarımızın özetini aşağıdaki tablo ile özetleyebiliriz.

İşlem	**Açıklama (HTTP Trafiği**
Birinci Uygulama çalışır.	
Kitap kitap81 = (from k in proxy.Kitap where k.KitapId==81 select k).First<Kitap>();	HTTP Get Paketi Gönderi Select sorgusu 81 no için Response bilgisi HTTP 20 ETag bilgisi ***"Ado.Net Da***
İkinci Uygulama çalışır.	
Kitap kitap81 = (from k in proxy.Kitap where k.KitapId==81 select k).First<Kitap>();	HTTP Get Paketi Gönderi Select sorgusu 81 no için Response bilgisi HTTP 20 ETag bilgisi ***"Ado.Net Da***
Birinci Uygulamada Fiyat bilgisi güncellenir.	
Birinci Uygulamada SaveChanges metodu çalışır.	HTTP Merge paketi gider. If-Match bilgisi ***"Ado.Net*** SQL tarafında Update sorg Response için ETag değer
İkinci Uygulamada Fiyat bilgisi güncellenir.	
İkinci Uygulamada SaveChanges metodu çalışır.	HTTP Merge paketi gider. If-Match bilgisi ***"Ado.Net*** SQL tarafında 81 için veri If-Match bilgisindeki Fiya ***HTTP/1.1 412 (Precondit***

Tabi işin bir de diğer şeklini ele almak gerekir. Yani **Fixed** değerlerini kullanmadığımız durum. Burada sadece **servis tarafındaki EDM** diagramında değişiklik yapmak yeterli olacaktır. Bir başka deyişle istemci tarafında, **Concurrency** modelinin değiştirildiğine dair bir servis güncellemesi yapılmasına gerek yoktur. Tabi böyle bir durumda her iki uygulamanın güncelleme işlemleride geçerli olacaktır ve buna görede en son yazanın verisi tabloya yansıtılacaktır. Aynı örneği buna göre test ettiğimizde **SQL** tarafına giden **Update** sorgularının aşağıdakine benzer olduğu görülmektedir.

```
exec sp_executesql N'update [dbo].[Kitap]
set [Ad] = @0, [Fiyat] = @1, [StokMiktari] = @2, [KategoriId] = @3
where ([KitapId] = @4)
',N'@0 nvarchar(25),@1 decimal(19,4),@2 int,@3 int,@4 int',@0=N'Ado.Net Data Services Pro',@1=88.0000,@2=35,@3=1,@4=81
```

Dikkat edileceği üzere sadece **Primary Key** alanı hesaba katılmıştır. Yine **Fiddler** aracı ile istemci ve servis arasındaki **HTTP** trafiği incelendiğinde **If-Match** yada **ETag** gibi bilgilerin **Request**veya **Response Header**' ları içerisinde yer almadığı görülür. Görüldüğü üzere senaryonun gerektirdiklerine göre servis tarafında Optimistic Concurrency modeli tercih edilebilir veya edilmez. Eğer bu model tercih edilirse istemci tarafındaki uygulamalarda mutlaka Exception kontrolünün yapılması gerekmektedir. Böylece geldik bir yazımızın daha sonuna. Bir sonraki yazımızda görüşünceye dek hepinize mutlu günler dilerim.

Concurrency.rar (39,02 kb)

Ado.Net Data Services Ders Notları - Custom LINQ Provider ve CUD Operasyonları (2008-10-24T05:38:00)

ado.net data services,

Ado.Net Data Services konusu ile ilintili bir önceki ders notlarımızda, **EDM(Entity Data Model)** üzerinden **CUD(CreateUpdateDelete)** işlemlerinin nasıl yapılabileceğini incelemeye çalışmıştık. Ancak durum özel **LINQ Provider** kullanımı söz konusu olduğunda biraz daha karmaşıklaşmakta. Nitekim **Custom LINQ Provider** kullanılması halinde istemci tarafından gelen**CUD** taleplerine karşılık servis tarafında özel kodlamaların yapılması gerekiyor. Bu noktada ders notlarımız içerisinde belkide çoğumuzun korkup fazla bulaşmak istemediği bir konuya kısacada olsa değineceğimizi şimdiden ifade etmek isterim. **Reflection(Yansıma)** :)

"Hayda brea nereden çıktı bu reflection" diyenelerimiz eminim ki vardır. öyleyse kısaca bu kavramı hatırlamaya çalışalım. **Reflection** teknikleri ile **çalışma zamanında(Runtime) .Net CLR** tiplerine ait(ister kullanıcı tanımlı ister önceden tanımlanmış tipler) **metadata** bilgilerine ulaşılabilmektedir. Bu açıdan bakıldığında özellikle **plug-in** tabanlı uygulama geliştirmelerde, **IDE** tasarımlarında kullanılmaktadır. Hatta çalışma

zamanında tiplere ait canlı nesne örneklerinin üretilip kullanılması bile mümkündür. Söz gelimi **.Net Reflector**gibi araçlar **Reflection** teknikleri yardımıyla geliştirilirler. Peki konunun **Ado.Net Data Service**' ler ile olan ilişkisi nedir? Neden bu tekniklere ihtiyaç vardır?

***NOT : Reflection** tekniklerinin kalbinde **Type** isimli tip yer alır. Bu basit tipten yararlanarak çalışma zamanında herhangibir tipe ait bilgileri elde etmek, tiplere ait nesne örnekleri oluşturmak gibi işlemler yapılabilir. Bu basit işlemler ile **plug-in** uygulamaları, **Reflection** gibi **.Net Assembly**' larının içeriğini gösteren programlar yazılabilir. Hatta **IDE** geliştirmelerinde**Reflection** tekniklerinden yararlanılmaktadır.*

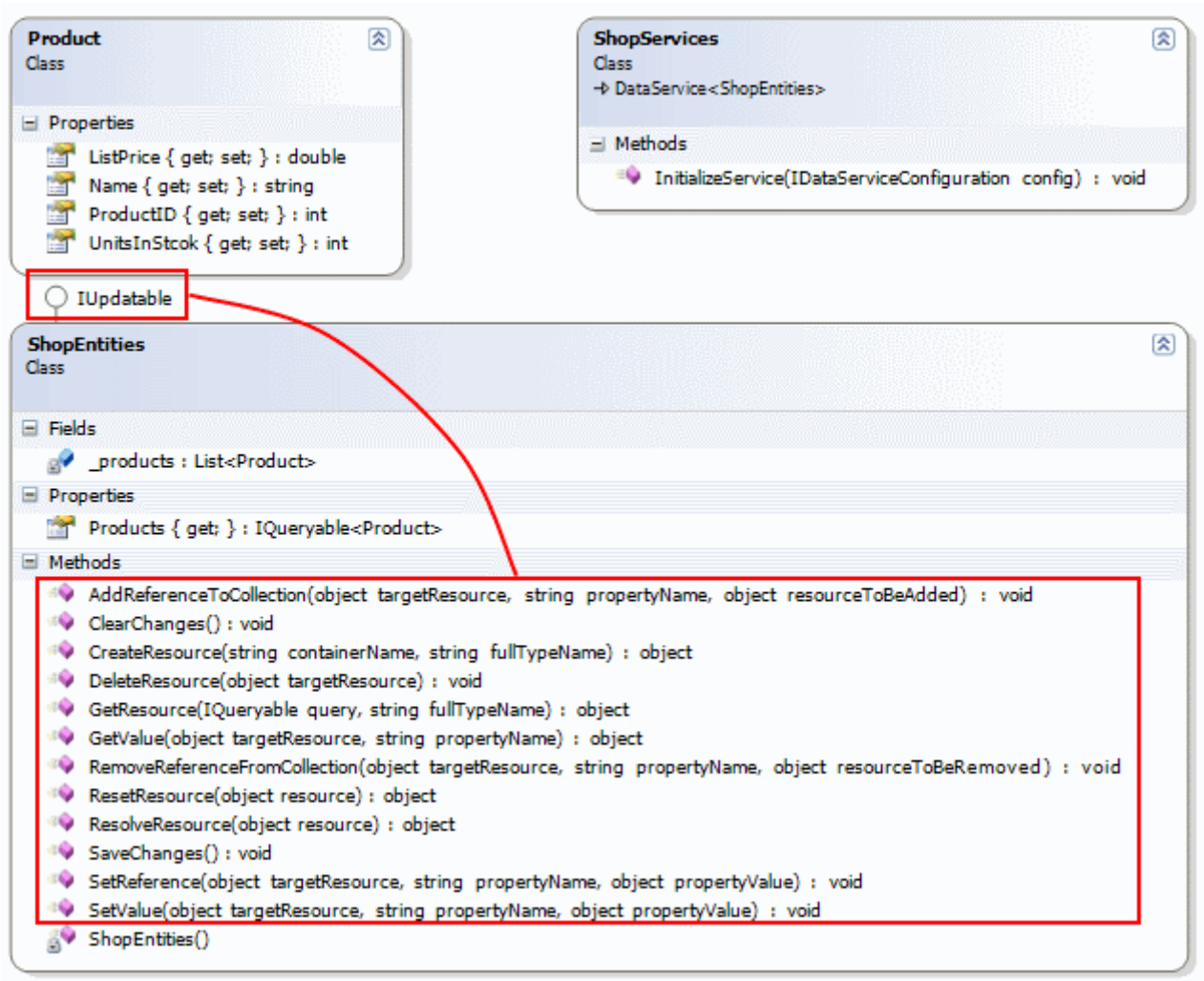
Bu sorunun sorulmasını nedeni servis tarafında **Custom LINQ Provider** kullanılmasıdır. İstemciler **CUD** işlemleri için servis tarafına nesne verisi içeren **HTTP** paketleri gönderirler. Servis tarafında yer alan herhangibir **LINQ Provider**' ın bu nesne içeriklerini ilgili veri kaynaklarına eklemesi, çıkartması yada değiştirmesi için çalışma zamanının anlayabileceği belirli kurallara uyması gerekir. Böylece herhangibir **Custom LINQ Provider** alt yapısının **CUD** işlemlerini gerçekleştirebilmesi bir standart altına alınmış olunur. öyleyse burada servis tarafında uyulması gereken bir kurallar dizisi söz konusudur. Bu kurallar öyle bir yapı içerisinde tanımlanmalıdırki **.Net CLR**' a özgü olmalıdır. İşte bu noktada nasıl bir tip olabilir sorusunu kendinize sormanız gerekmektedir? **Interface(Arayüz)**. Bilindiği üzere bir arayüz, uygulandığı tipin uyması gereken kuralları belirtmektedir. Ancak bunun yanında çok biçimlilik özelliğine sahiptir ve bu nedenle çalışma zamanında kendisini implemente eden tiplere ait nesne örneklerini taşıyabilmektedir. Buda **plug-in** tabanlı mimarilerde önem arz eden bir konudur. Bu şimdiki senaryomuzda gerekli bir bilgi değildir belki ama arayüzlerin ne işe yaradığınıda anlatan güzel bir tanımlamadır.

***NOT : Interface(Arayüz)** tipi sadece kendisini uyarlayan tiplerin uyması gereken **üye(member)** tanımlamalarını içerir. Bunun dışında iş yapan **üyeler(members)** içermez. Ayrıca**polimorfik** bir başka deyişle çok biçimliliğe destek verirler. Dolayısıyla bir arayüz tipini kullanarak çalışma zamanında kendisinden türeyen birden fazla nesneyi işaret etmek ve bunların hepsi için ortak işlemler yürütmek mümkündür. Nitekim bu ortak işlemler arayüz içerisinde tanımlanmış olup, arayüzü implemente eden tiplerde yazılma zorunluluğu bulunan ve farklı şekillerde uygulanabilen üyelerdir. (Bu noktada ah keşke şu C# derslerindeki temel konuları biraz daha araştırsaydım diyenleriniz olabilir. Vakit çok geç değil...)*

Bu anlatılanlardan özet olarak şu sonucu çıkartabiliriz. **Ado.Net Data Service** içerisinde kullanılan **taşıyıcı varlık tipinin(Container Entity Type) CUD** işlemleri için belirli kurallara uyması ve bu kuralları uygulaması gerekmektedir. Bu dayatma için **.Net CLR** içerisinde **System.Data.Services** isim alanında yer alan **IUpdatable** adlı bir **interface** tipi tanımlanmıştır. Bu arayüzün üye metodları ile **CUD** işlemleri için gerekli uyarlamaların yapılması istenir. Bakın henüz **Reflection** tekniklerinden birisini kullanmaktan bahsetmediğimizi belirtelim. Bunu bir nesneyi servis tarafında örneklerken kullanıyor olacağız.

Şu anda neler hissetiğinizi biliyoru ve size hak veriyorum. Bu yazılanlar biraz sıkıcı. Aslında adım adım bir örneğe geçerek ilerlemekte yarar var. Gelin hiç vakit kaybetmeden bir **WCF Service**uygulaması geliştirelim ve içerisinde **Custom LINQ Provider** kullanan bir **Ado.Net Data Service** öğesi oluşturalım. Bu amaçla açılan **WCF Service** uygulamasına sırasıyla aşağıdaki tipleri entegre etmemiz yeterli olacaktır. İlk olarak servis tarafındaki geliştirmelerimizin genel görünümüne sınıf diagramı görüntüsünden bir bakalım.

Sınıf diagramı(Class Diagram);



öncelikli olarak **Entity** tipimizi geliştirelim. **Product** isimli sınıfımız basit olarak bir ürüne ait Id, ad, fiyat, stok miktari gibi bilgileri taşıyacak şekilde tasarlanmıştır.

Product sınıfı;

```
using System;
using System.Linq;

namespace ServerApp
```

```
{
   public class Product
   {
      public int ProductID { get; set; }
      public string Name { get; set; }
      public double ListPrice { get; set; }
      public int UnitsInStcok { get; set; }
   }
}
```

Product **entity** tipini içerisinde kullanan ve istemcilere sunan taşıyıcı tipimiz ise aşağıdaki gibidir. Yanlız burada dikkat edilmesi gereken en önemli nokta söz konusu tipin **IUpdatable**arayüzünü uygulamış olmasıdır.

ShopEntites sınıfı;

```
using System;
using System.Linq;
using System.Reflection;
using System.Data.Services;
using System.Collections.Generic;

namespace ServerApp
{
   // CUD işlemlerine destek vermesi amacıyla ShopEntities tipine IUpdatable arayüzü uyarlanmıştır.
   public class ShopEntities
      :IUpdatable
   {
      // Product tipinden listeyi tutacak generic koleksiyonumuz
      static List<Product> _products;

      // Static yapıcı metod(Constructor) içerisinde bir kereliğine _product koleksiyonu örnek veriler ile doldurulur.
      static ShopEntities()
      {
         _products = new List<Product>
            {
               new Product{ ProductID=1, Name="Dvd Player", ListPrice=100, UnitsInStcok=100},
               new Product{ ProductID=2, Name="Mp3 Player 8 Gb", ListPrice=50, UnitsInStcok=150},
               new Product{ ProductID=3, Name="15.4 inch LCD", ListPrice=190, UnitsInStcok=50},
               new Product{ ProductID=4, Name="320 Gb Hdd", ListPrice=120,
```

```
UnitsInStcok=200},
                new Product{ ProductID=5, Name="1 Tb Hdd 3.5inch", ListPrice=250,
UnitsInStcok=175}
            };
    }

    // İstemci tarafına sunulacak olan readonly(yanlız okunabilir) özellik
    public IQueryable<Product> Products
    {
        get
        {
            // AsQueryable<> ile generic List koleksiyonunun sorgulanabilir olması sağlanır
            return _products.AsQueryable<Product>();
        }
    }

    #region IUpdatable Members

    // Yeni bir Product nesnesinin oluşturulması ve eklenmesi sırasında devreye girer
    public object CreateResource(string containerName, string fullTypeName)
    {
        // Activator kullanılarak tipin full adından bir nesne oluşturulması sağlanır
        var newProduct = Activator.CreateInstance(Type.GetType(fullTypeName));
        // Oluşturulan nesne örneği Product tipine dönüştürülerek _products isimli generic
koleksiyona eklenir
        _products.Add((Product)newProduct);
        // Eklenen yeni nesne örneği metoddan geriy döndürülür
        return newProduct;
    }

    // Silme işlemi sırasında devreye giren metod
    // Parametre olarak silinecek entity nesne örneği gelir
    public void DeleteResource(object targetResource)
    {
        // Gelen nesne örneği Product tipine dönüştürülür ve _products isimli
koleksiyondan Remove metodu ile çıkartılır
        _products.Remove((Product)targetResource);
    }

    // Update(güncelleme) işlemi sırasında devreye giren metoddur.
    public object GetResource(IQueryable query, string fullTypeName)
    {
        object r = null;
        var numarator = query.GetEnumerator();
        while (numarator.MoveNext())
```

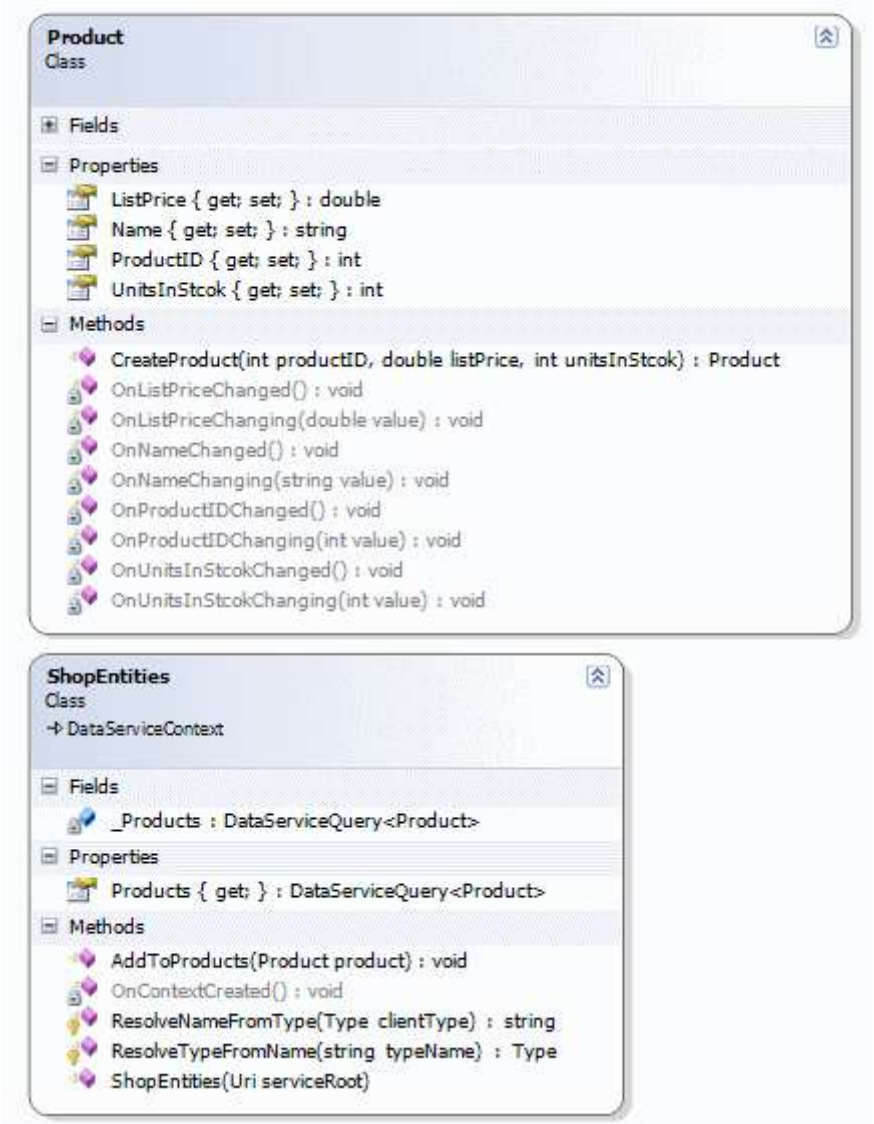
```
        {
            if (numarator.Current != null)
            {
                r = numarator.Current;
                break;
            }
        }
        return r;
    }

    // gelen nesnenin belirtilen özelliğinin değerini geriye döndüren metoddur
    public object GetValue(object targetResource, string propertyName)
    {
        var targetType = targetResource.GetType();
        PropertyInfo targetProperty = targetType.GetProperty(propertyName);
        return targetProperty.GetValue(targetType, null);
    }

    // Gelen nesnenin ilgili özelliğine gelen değeri atayan metoddur.
    // Bu metod veri ekleme ve güncelleştirme işlemleri sırasında ilgili nesne örneğinin her bir özelliği için çalışır
    public void SetValue(object targetResource, string propertyName, object propertyValue)
    {
        Type targetType = targetResource.GetType();
        PropertyInfo targetProperty = targetType.GetProperty(propertyName);
        targetProperty.SetValue(targetResource, propertyValue, null);
    }

    public object ResolveResource(object resource)
    {
        return resource;
    }

    // Değişikliklerin kaydedilmesini sağlayan metoddur.
    // Söz konusu örnekte veriler bellek üzerinden tutulduğundan uygulanmasına gerek yoktur.
    public void SaveChanges()
    {
    }

    public void RemoveReferenceFromCollection(object targetResource, string propertyName, object resourceToBeRemoved)
    {
        throw new NotImplementedException();
```

```
        }

        public object ResetResource(object resource)
        {
            throw new NotImplementedException();
        }

        public void SetReference(object targetResource, string propertyName, object
propertyValue)
        {
            throw new NotImplementedException();
        }

        public void AddReferenceToCollection(object targetResource, string propertyName,
object resourceToBeAdded)
        {
            throw new NotImplementedException();
        }

        public void ClearChanges()
        {
            throw new NotImplementedException();
        }

        #endregion
    }
}
```

Woooww! Evet biraz korkutucu bir implemantasyon. Ama en azından **Custom Membership Provider** yazarkenki kadar korkutucu değil :) Aslında tüm arayüzün uyarlamasını gerçekleştirmiş değiliz. Nitekim söz konusu örnekte kullanılan **Product** tipi içerisinde herhangibir navigasyon özelliği bulunmamaktadır. Diğer taraftan sadece **CUD** işlemleri için gerekli temel metodların uygulandığını ifade edebiliriz. Herşeyden önce **CreateResource** metoduna dikkat etmemiz gerekiyor. Notlarımızın başında belirtmiş olduğumuz gibi **Reflection** tekniklerinin burada iyi bir kullanımını görüyoruz. Nitekim ilk olarak parametre olarak gelen **string** ifadeden yararlanılarak bir nesne örneği oluşturuluyor ki bu noktada **.Net Remoting** günlerinden hatırlayacağımız meşhur **Activator** sınıfı kullanılmakta. Yine **GetValue** ve **SetValue** metodları içerisinde parametre olarak gelen bilgilerden yararlanılarak bir özellik değerinin set edilmesi veya elde edilmesi işlemlerinde **Reflection** teknikleri kullanılıyor. Tabi bizim odaklanmak istediğimiz nokta **Ado.Net Data Service** içerisinde **Custom LINQ Provider** kullanılması halinde **CUD** işlemlerinin nasıl gerçekleşeceği. Buraya kadar yaptıklarımız ile işin zor kısmını az da olsa aştık. Şimdi aşağıdaki **svc** içeriğine sahip **Ado.Net Data Service** öğesinide **WCF** projemize ekleyelim.

```
using System;
using System.Data.Services;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel.Web;
using ServerApp;

public class ShopServices
    : DataService<ShopEntities>
{
   public static void InitializeService(IDataServiceConfiguration config)
   {
      config.SetEntitySetAccessRule("*", EntitySetRights.All);
   }
}
```

CUD işlemeri gerçekleştireceğimiz için **SetEntitySetAccessRule** metodu içerisinde **All** sabit değerini kullanıyoruz. Artık istemci tarafını ve istemci için gerekli test kodlarını yazabiliriz. Her zamanki gibi basit bir **Console** uygulaması işimizi görecektir. Elbette servisimizide **Add Service Reference** seçeneği ile istemci uygulamaya eklememiz gerekiyor. İstemci tarafında yapılan servis ekleme işlemi sonrasında aşağıdaki sınıf diagramında görülen tipler oluşacaktır.



Daha önceki ders notlarımızdan da hatırlayabileceğimiz gibi, veri ekleme işlemleri için **AddToProducts** metodu oluşturulmuştur. Bunun dışında **DataServiceContext** üzerinden**UpdateObject**, **DeleteObject** metodlarınıda kullanıyor olacağız. Nitekim bu metodlar ile, silme ve güncelleme operasyonları gerçekleştirilecektir. İstemci uygulamamızın kod içeriği ise örnek olarak aşağıdaki gibi geliştirilebilir.

İstemci uygulama kodları;

```
using System;
using System.Linq;
using System.Collections.Generic;
using ClientApp.ShopServiceReference;

// Not: Kod içerisinde Exception kontrolleri yapılmamıştır
namespace ClientApp
{
```

```
class Program
{
   static void Main(string[] args)
   {
      ShopEntities proxy = new ShopEntities(new
Uri("http://buraksenyurt:1000/ServerApp/ShopServices.svc"));

      // Yeni bir Product nesne örneği oluşturulur
      Product newProduct = new Product
         {
            ProductID = 6,
            Name = "HP Mouse",
            ListPrice = 12,
            UnitsInStcok = 100
         };

      // Nesne örneği eklenir
      proxy.AddToProducts(newProduct);

      // Ekleme operasyonu servis tarafına gönderilir
      proxy.SaveChanges();

      // ProductID değeri 2 olan Product nesnesi istenir.
      var prd = (from p in proxy.Products
                 where p.ProductID==2
                 select p).First<Product>();

      // Bazı özelliklerin değerleri sembolik olarak değiştirilir
      prd.ListPrice += 10;
      prd.UnitsInStcok += 15;
      // Nesne güncellemesi yapılır
      proxy.UpdateObject(prd);

      // Update operasyonu servis tarafına gönderilir
      proxy.SaveChanges();

      // Silme işlemine örnek olması için son eklenen Product istenir
      var lastPrd = (from p in proxy.Products
                     where p.ProductID == newProduct.ProductID
                     select p).First<Product>();

      // Nesne silinmesi yapılır
      proxy.DeleteObject(lastPrd);
      // Delete operasyonu servis tarafına gönderilir
      proxy.SaveChanges();
```

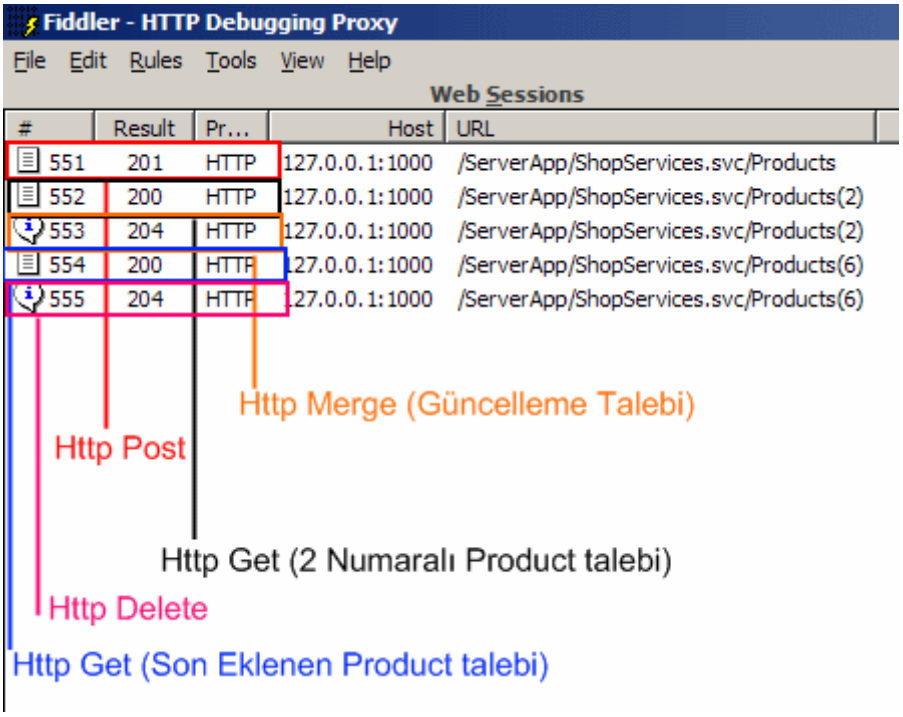
```
        }
    }
}
```

örnekte ilk olarak bir **Product** nesne örneği oluşturulmakta ve sonra **AddToProducts** metodu ile ilave edilmektedir. Tabiki bu ekleme işleminin servis tarafına gönderilmesi için, **SaveChanges** metodunun çağırılması gerekir. Güncelleme işlemi için **UpdateObject** metodu kullanılmaktadır. Silme işlemi içinse **DeleteObject** metodu. Elbette her iki işleminde **HTTP** paketleri içerisine gömülerek servis tarafına gönderilmesi için **SaveChanges** metodu çağırılmalıdır. İstemci uygulama bu haliyle çalıştırıldığında herhangibir sorun olmadan işlemlerin gerçekleştiği görülür. *(Ancak siz bu uygulamayı test ederken mutlaka **Debug** modda çalıştırın ve küçük bir tavsiye; **WCF** tarafındaki **debug** işlemleri için hem servis hemde istemci uygulamayı debug modda çalıştırmalısınız.)*

Uygulama test edilirken eğer **SaveChanges** metodlarında **breakpoint**' ler ile ilerlenir ve arka planda **Fiddler** gibi bir **Http Debugging Proxy** aracı kullanılırsa uygulamanın tamamının çalışması sonrasında aşağıdaki sonuçların elde edildiği görülür.

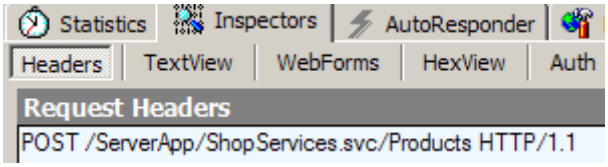
NOT : ***Fiddler*** *aracı basit bir **HTTP Debugging Proxy** uygulamasıdır ve **IIS** üzerinde **80** numaralı porta gelen ve giden tüm **HTTP** paketlerini izleyebilmenizi, içeriğini görebilmenizi sağlar. Lakin **Asp.Net Development Server** ile kullanımında bazı ön ayarlar yapılması gerekmektedir. **Fiddler** aracının kullanımı **Ado.Net Data Services**' lerde **Batch Processing**işlemlerinin ele alındığı **görsel dersimizde** incelenmiştir.*

HTTP Paketlerinin Fiddler aracı üzerinden incelenmesi;

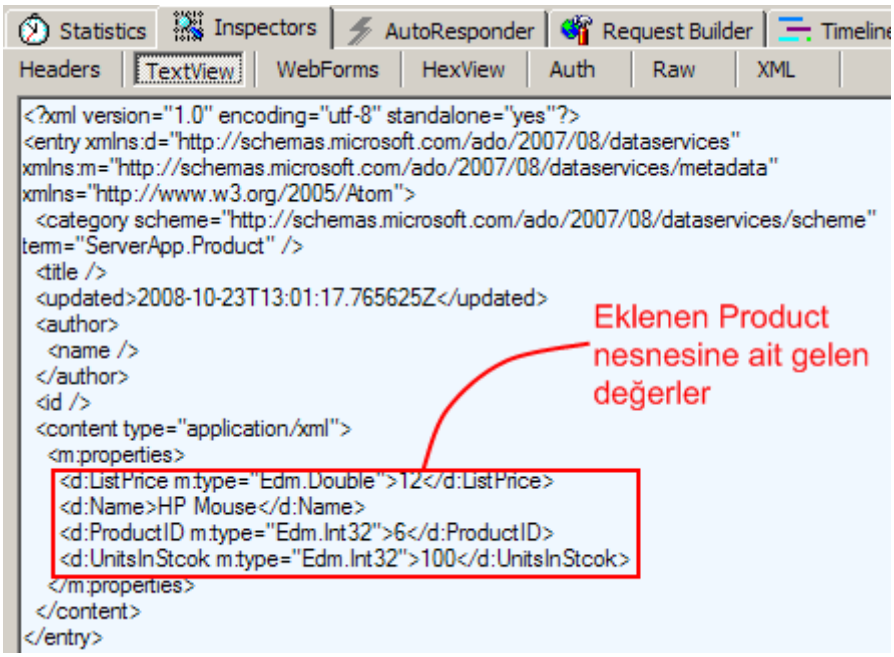


Bu durumu biraz analiz edelim. İlk olarak yeni bir **Product** ekleniyor. Bu ekleme işlemi istemci tarafında yapılan **SaveChanges** metodu çağrısı sonrasında, servis tarafına bir **HTTP** paketi olarak gidiyor ki bu **POST** metoduna göre hazırlanmış bir pakettir. Yine Fiddler yardımıyla paketin içeriğinin aşağıdaki gibi olduğu görülebilir.

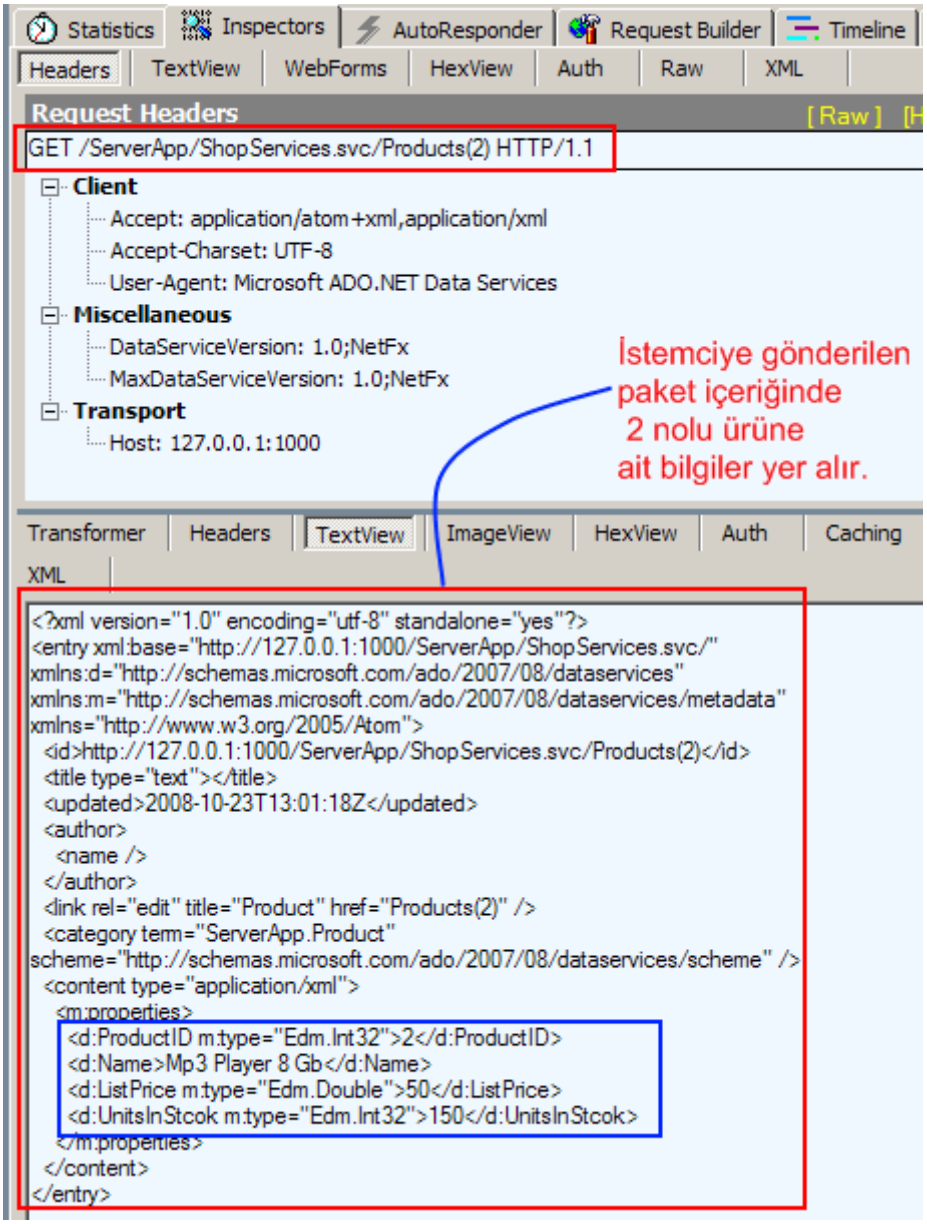
Header içeriği;



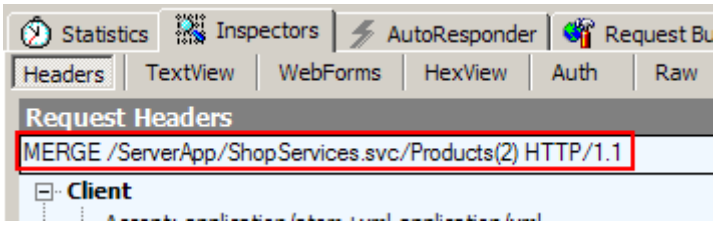
Paket içeriği;



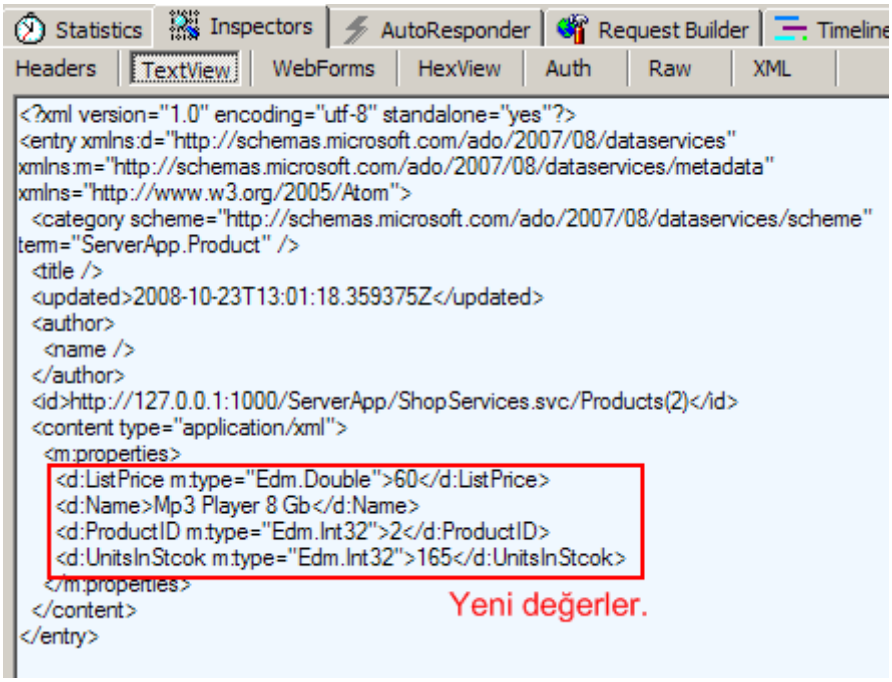
Sonrasında ise bir **HTTP Get** talebi gelmektedir. Nitekim kod içerisinde güncelleme örneği için, **ProductID** değeri **2** olan ürün bilgisi istenmiştir. Bunun sonucu olarak tabiki istemci tarafına da bir içerik gönderilmektedir. Yine Fiddler aracı yardımıyla bu içeriğe bakılabilir.



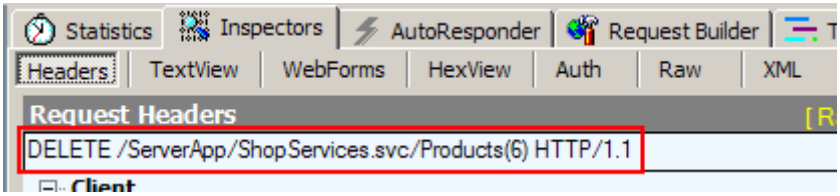
ProductId değeri 2 olan ürünün istemci tarafına çekilmesinin ardından bir güncelleme işlemi gerçekleştirilir. Koda dikkat edecek olursak bu işlemler için **UpdateObject** ve ardından**SaveChanges** metodları çağırılmaktadır. **SaveChanges** metodu bu kez istemciden servis tarafına **HTTP Merge** paketi gönderir ve bu paket içerisinde güncelleştirilen yeni değerler yer alır. Buna göre servise gelen **Request** paketinin **Header** içeriği aşağıdaki gibidir.



Paket ile gönderilen bilgiler ise yine Fiddler aracı ile görülebilir.



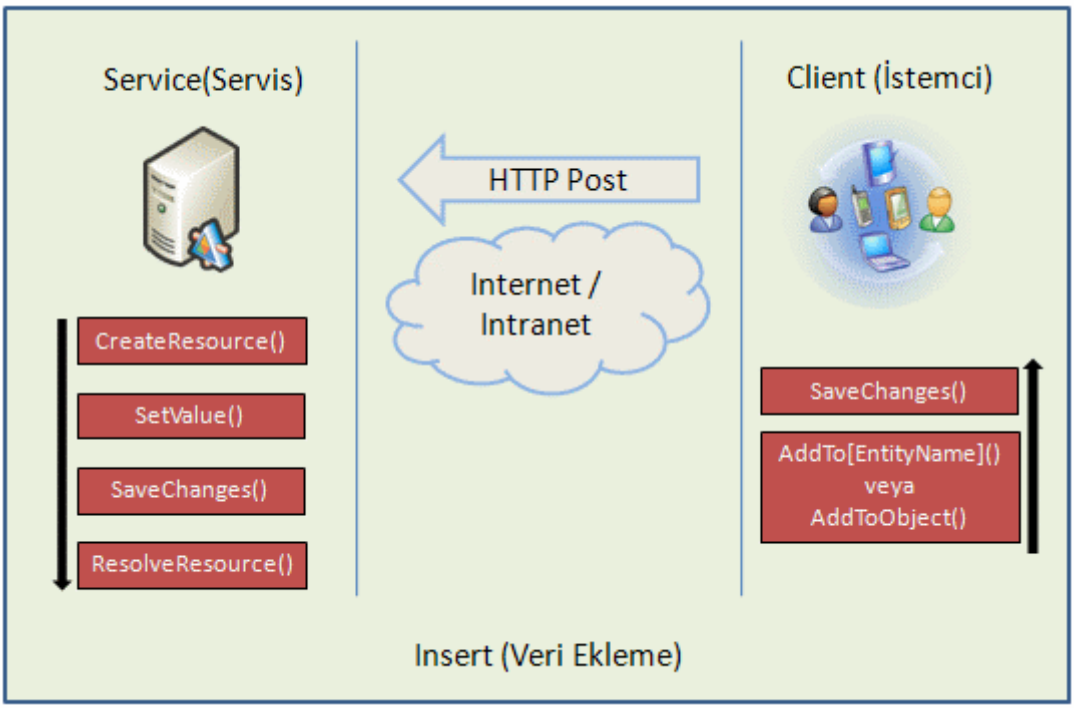
Son olarak silme operasyonunda önce silinmek istenen veri servis tarafından talep edilmiştir. Bu noktada yine bir **HTTP Get** çağrısı gerçekleşir. Bu işlemden sonra istemci kodlarında**DeleteObject** ve ardından yine **SaveChanges** metodları çağırılmıştır. **SaveChanges** çağrısı sonrasında ise bu kez bir **HTTP Delete** talebi istemciden servis tarafına doğru gönderilecektir.



Arka planda HTTP paketlerinde neler gittiğini gördük. Burada Fiddler aracına çoğunuzun aşık olduğunu hisseder gibiyim. Aynı **SQL Server Profiler** gibi son derece başarılı bir izleme aracı.

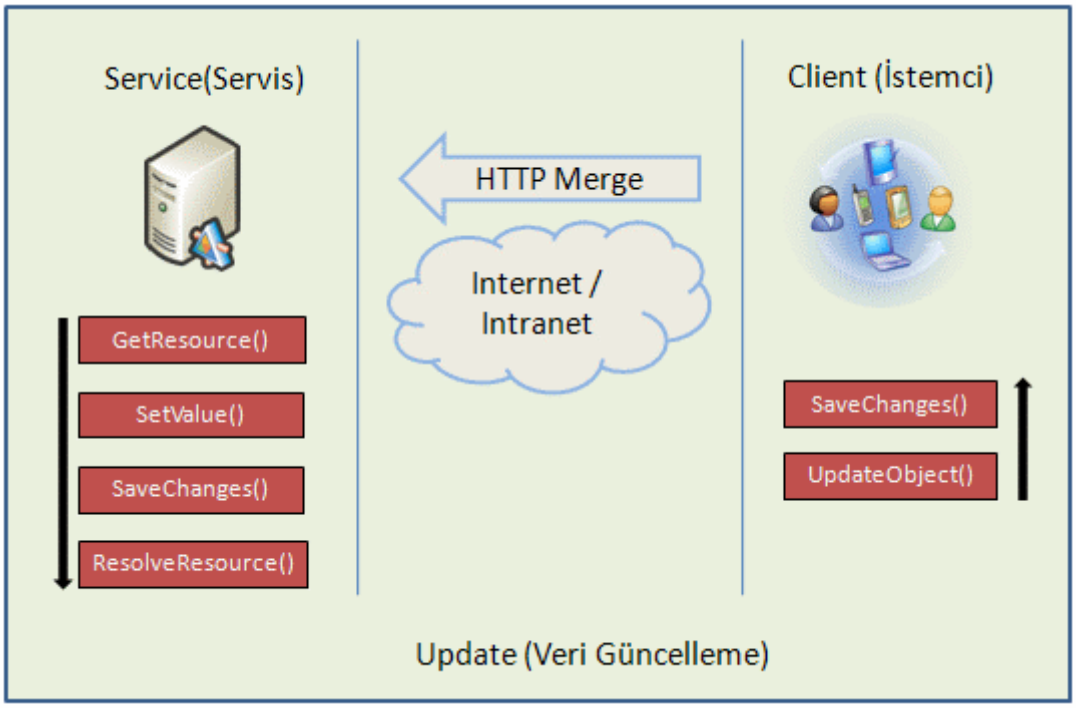
Gelelim kod tarafındaki metod işleyişlerine. Söz gelimi veri ekleme işlemi sırasında istemci tarafında **AddToProducts** ve **SaveChanges** metodlarını kullanıyoruz. Peki ya servis tarafında uyguladığımız **IUpdatable** arayüzüne ait hangi metodlar devreye giriyor. Bu durumu analiz etmek için **Debug** modda biraz dolaşmam gerektiğini ifade etmek isterim. Sonunda aşağıdaki sonuçlara ulaşabildim. Tabi söz konusu süreçler yukarıdaki geliştirdiğimiz örneğe göre işlemektedir.

Insert işlemine ait süreç;



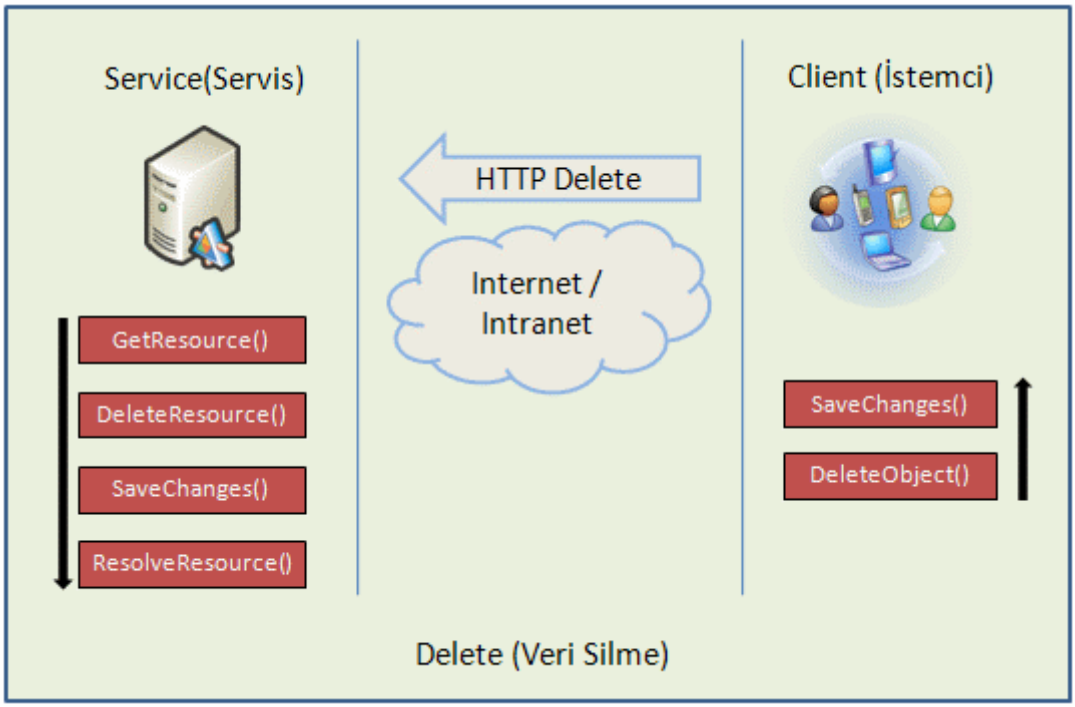
İstemci tarafında ilk olarak **AddTo[EntityName](örneğin AddToProducts)** metodu çağırılır. Sonrasında ise **SaveChanges** metodu yürütülür. **SaveChanges** metodu **insert** işlemi için gerekli **HTTP** paketinin **Post** metoduna göre servis tarafına gönderilmesinde rol oynar. Bunun karşılığında servis tarafında sırasıyla **CreateResource**, **SetValue**, **SaveChanges** ve**ResolveResource** metodları çağırılır. **CreateResource** metodu ile **HTTP** paketinde gelen istekte yer alan tipin, çalışma zamanında oluşturulması ve ilgili veri kaynağına eklenmesi işlemleri gerçekleştirilir. **SetValue** metodu ise oluşturulan tipin her özelliği için çalışır. Bir başka deyişle özelliklerin değerlerinin verilmesinde devreye girer. örnekte veriler bellekteki koleksiyonlarda tutulduğu için geri dönüş değeri olmayan ve parametre almayan **SaveChanges** metodu içerisinde herhangibir işlem yapılmamıştır.

Update işlemine ait süreç;



İstemci tarafında bu kez öncelikli olarak **UpdateObject** ve sonrasında **SaveChanges** metodları çağırılır. Servis tarafında ise öncelikli olarak güncellenecek verinin elde edilmesi için**GetResource** metodu devreye girer. **Insert** sürecine benzer bir şekilde **SetValue** metodu güncellenecek özellikler için tek tek çalışır ve yine sırasıyla **SaveChanges**, **ResolveResource**metodları devreye girer.

Delete işlemine ait süreç;



Silme işleminde istemci tarafında sırasıyla **DeleteObject** ve **SaveChanges** metodları çalışır. Bunun karşılığında **Delete** metodunu içeren bir **HTTP** paketi servis tarafına gönderilir. Servis tarafında ise yine silinmek istenen verinin elde edilmesi için **GetResource** metodu ilk olarak devreye girer. Sonrasında ise **DeleteResource** ile bu verinin kaynaktan çıkartılması sağlanır. örnekte bir koleksiyon kullanıldığından bu basit bir **Remove** işleminden öte değildir. Son olarak yine **SaveChanges** ve **ResolveResource** metodları çağırılır.

Görüldüğü üzere kendi geliştirdiğimiz **Custom LINQ Provider**' ların kullanıldığı senaryolarda en kritik nokta **IUpdatable** arayüzünün kullanılmasıdır. Bu arayüzün yazımızda kullanılan örnek implemantasyonu ile basit CUD işlemleri gerçekeştirilebilir. Böylece geldik **Ado.Net Data Service**' ler ile ilişkili bir ders notumuzun daha sonuna. Bir sonraki yazımızda görüşünceye dek hepinize mutlu günler dilerim.

UsingIUpdatable.rar (39,03 kb)

Ado.Net Data Services Ders Notları - CUD Operasyonları (2008-10-16T05:33:00)

ado.net data services,

Ders notlarımızı tutmaya devam ediyoruz. Bu gün **Ado.Net Data Service**' ler yardımıyla istemcilerden **veri ekleme(Insert), silme(Delete) ve güncelleme(Update)** işlemlerinin nasıl yapılabileceğini incelemeye karar verdim. Tabiki **Ado.Net Data Services** konusu halen daha **Astoria** kod adıyla anılmakta. Dolayısıyla zaman içerisinde uygulanan metod adlarında ve kullanılış biçimlerinde değişiklikler olması muhtemel. Yine şu an itibariyle neler yapabileceğimize bakmakta yarar var nitekim bir **WCF** fanatiği olarak **Ado.Net Data Services** açılımı beni son derece heyecanlandırıyor. Bu kadar laf salatasından sonra kısaca konuya girmeye ve basit bir örnek geliştirmeye ne dersiniz?

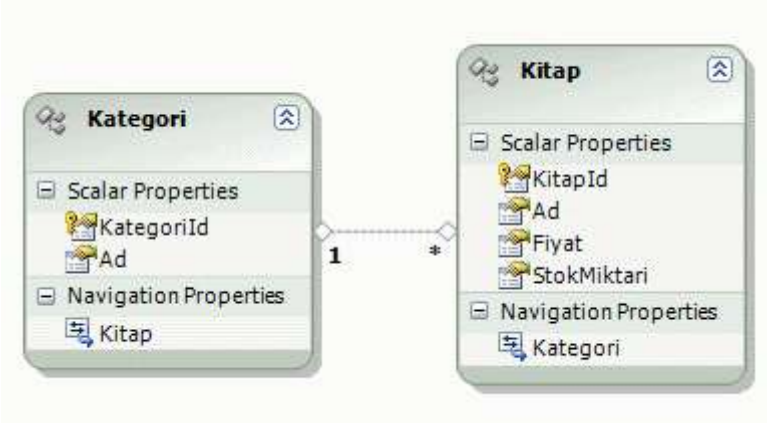
Ado.Net Data Service operasyonlarına yapılan istemci çağrılarının **HTTP** bazlı olduklarını ve **GET,POST,PUT,DELETE** gibi metodlara göre uygulandıklarını biliyoruz. İstemci tarafından servis operasyonlarına doğru eklenmek, silinmek veya güncellenmek amacıyla gönderilen verilerin çoğunluğuda **POST** metoduna uygun olacak şekilde paketlenmektedir. Ancak elbetteki istemci tarafında bu paketin manuel olarak hazırlanması gibi işlemler ile uğraşmamıza gerek yoktur. Nitekim istemci tarafında oluşturulan servis örneğine ait üye metodlar yardımıyla bu paketlerin otomatik olarak hazırlanması, gönderilmesi sağlanabilmektedir. Her zamanki gibi adım adım ilerleyeceğimiz bir örnek konuyu pekiştirmek açısından çok daha yararlı olacaktır.

İlk olarak veritabanı üzerindeki hazırlıklarımızı yapalım. örneğimizde **Azon** isimli*(Benim seminerlerimi takip edenler bu isimdeki hayali şirketi hatırlayacaktır :))* bir veritabanını ve bunun üzerinde yer alan **Kategori** ve **Kitap** isimli tabloları kullanıyor olacağız. Şimdi hiç vakit kaybetmeden aşağıdaki **SQL Script**' ini **SQL Management Studio** üzerinde çalıştırabilir ve örnek veritabanı, tablo ve test verilerinin eklenmesini sağlayabilirsiniz.

```
--Test veritabanı oluşturulur
Create Database Azon
GO

--Test veritabanını kullan
Use Azon
GO

-- Kategori tablosu oluşturulur
Create Table Kategori
(
   KategoriId int identity(1,1) not null,
   Ad nvarchar(20) not null,
   Constraint Pk_Kategori Primary Key(KategoriId)
)
GO

--Kategori tablosu için test verileri girilir
Insert into Kategori (Ad) Values ('Programlama');
Insert into Kategori (Ad) Values ('SOA çözümleri');
Insert into Kategori (Ad) Values ('Web Programlama');

--Kitap tablosu oluşturulur
Create Table Kitap
(
   KitapId int Identity(1,1) not null,
   Ad nvarchar(50) not null,
   Fiyat money not null,
   StokMiktari int not null,
   KategoriId int not null,
   Constraint Pk_Kitap Primary Key(KitapId)
)
GO

--Kitap tablosu için test verileri eklenir
Insert into Kitap (Ad,Fiyat,StokMiktari,KategoriId) Values ('Her Yönüyle C#',50,100,1);
Insert into Kitap (Ad,Fiyat,StokMiktari,KategoriId) Values ('Essential C# 3.0',80,25,1);
Insert into Kitap (Ad,Fiyat,StokMiktari,KategoriId) Values ('Programming
WCF',75,120,2);
Insert into Kitap (Ad,Fiyat,StokMiktari,KategoriId) Values ('SOA For Dummies',35,45,2);
Insert into Kitap (Ad,Fiyat,StokMiktari,KategoriId) Values ('Asp.Net 3.5 Step By
Step',70,80,3);

--Relation Oluşturulur
ALTER TABLE Kitap WITH CHECK ADD CONSTRAINT FK_Kitap_Kategori
```

```
FOREIGN KEY(KategoriId)
REFERENCES Kategori (KategoriId)
GO

ALTER TABLE Kitap CHECK CONSTRAINT FK_Kitap_Kategori
GO
```

Kategori ve **Kitap** tabloları yine bir birlerine bağlı kümeler olarak tasarlanmıştır. Böylece bir **Kategori** ve buna bağlı **Kitap** satırlarının **Entity** örnekleri üzerinden eklenmesinin analizi için gerekli ortam hazırlanmış olur. Her iki tablo arasındaki ilişkinin veritabanı üzerinde tanımlanmış olmasıda son derece önemlidir. Nitekim bu tanımla **yapılmadığı** takdirde**EDM(Entity Data Model)** içerisinde oluşturulan **Entity** tipleri arasındada bir **Association** en azından otomatik olarak oluşmayacaktır. Sıradaki aşamada servisimiz için gerekli **host** uygulamanın yazılması gerekmektedir. **Ado.Net Data Service**' leri bu ana kadarki ders notlarımızda sürekli olarak **WCF Service** şablonları üzerinde tuttuk. Şimdilik geleneği bozmuyoruz. Yine işlerimizi kolaylaştırması açısından **EDM** katmanını kullanıyor olacağız. Daha önceki ders notlarımızda ve görsel derslerimizde bu konuya değindiğimiz için tekrar etmeyeceğiz ancak oluşan **EDM** modelinin aşağıdakine benzer olması gerektiğinide hemen vurgulayalım.*(Bu aşamayı tamamlarken önceki ders notları veya görsel derslere bakmadan ilerlemeye çalışmanız sizin yararınıza olacaktır. Bildiğiniz gibi pratik mükemmelleştirir.)*

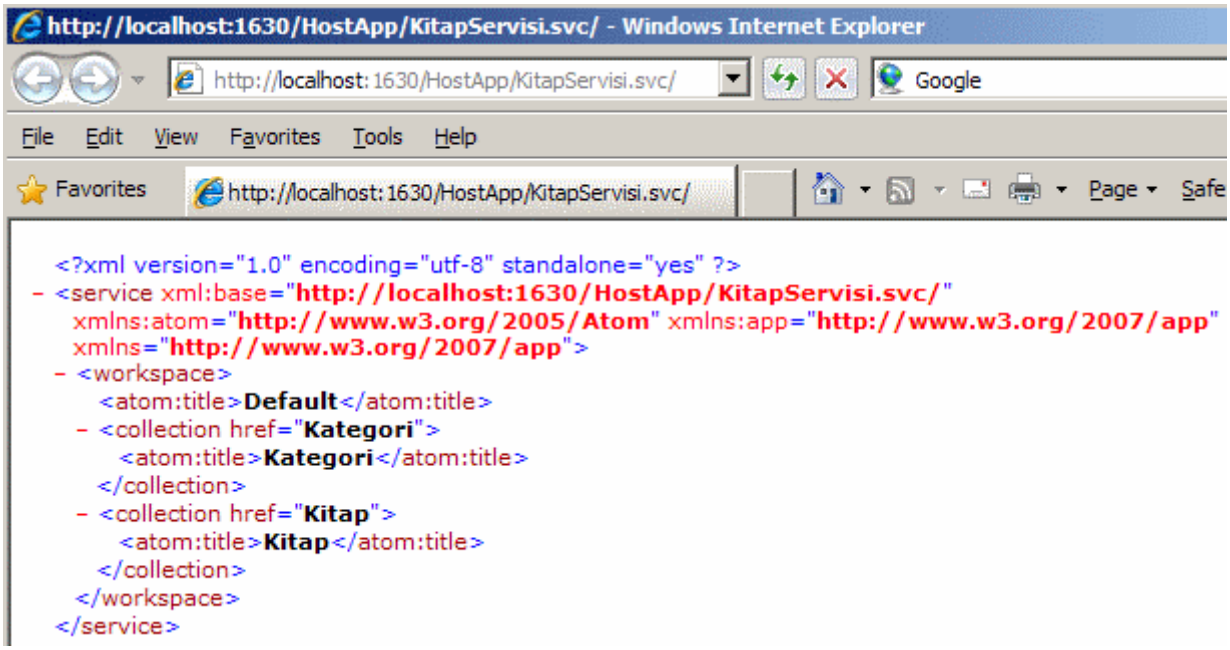


KitapServisi olarak isimlendirdiğimiz **svc** dosyasına ait kod içeriği ise aşağıdaki gibi olmalıdır.

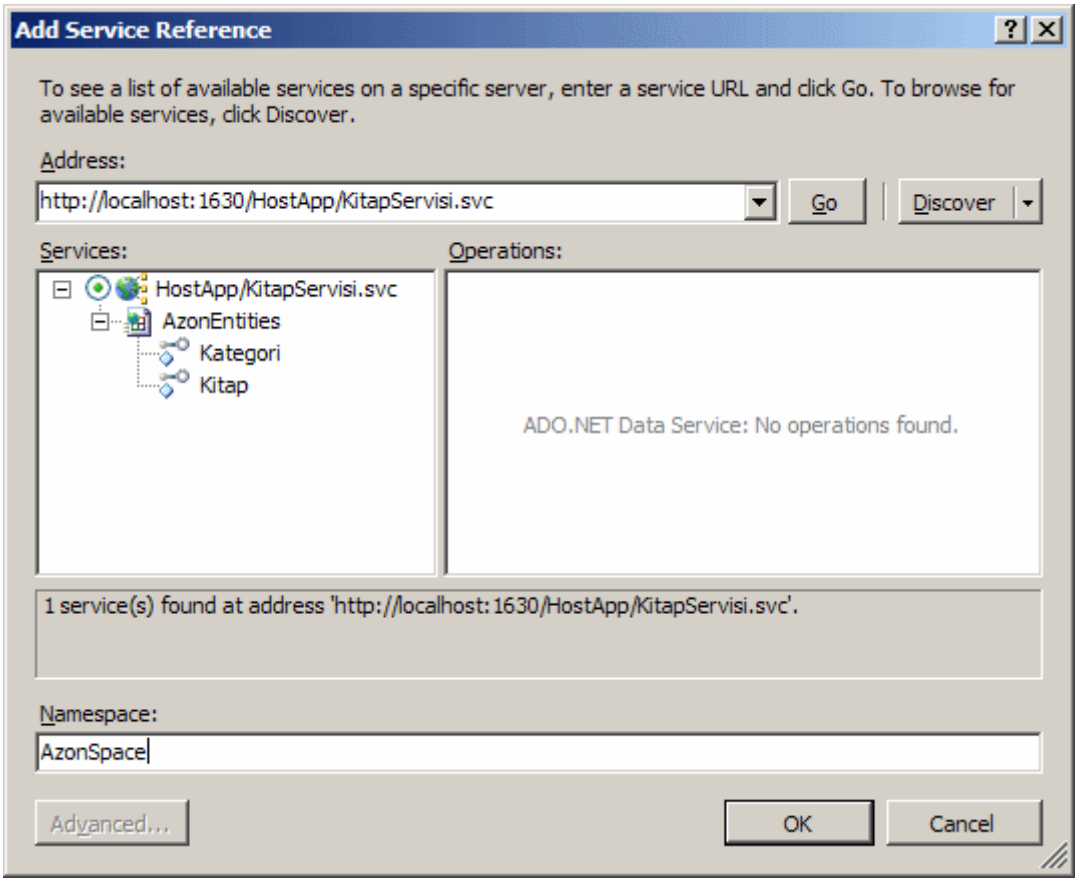
```
using System;
using System.Linq;
using System.Data.Services;
using System.ServiceModel.Web;
using System.Collections.Generic;
using AzonModel;

public class KitapServisi
    : DataService<AzonEntities>
```

```
{
   public static void InitializeService(IDataServiceConfiguration config)
   {
      config.SetEntitySetAccessRule("*", EntitySetRights.All);
   }
}
```

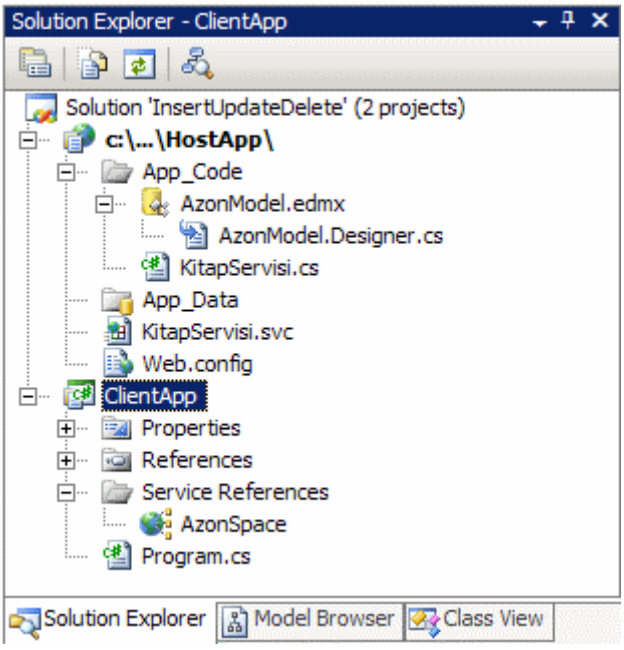
Tahmin edileceği üzere **AzonEntites** isimli taşıyıcı tip içerisindeki tüm **Entity** tipleri tüm haklar ile istemcilere açılmaktadır. Burada varsayılan olarak bulunan **AllRead** değerini kullanamayız. Nitekim verilerin eklenmesi, güncellenmesi ve silinmesi operasyonları için izin verilmesi gerekir. Burada kolaya kaçarak **All enum** sabiti değerini kullandık. Bu adıma kadar geldikten sonra yapmamız gereken ilk iş servisin çalışıp çalışmadığını test etmek olmalıdır. Bu amaçla servisin herhangibir tarayıcıda açılması yeterlidir. Eğer bir sorun yoksa aşağıdaki ekran görüntüsüne benzer bir çıktının elde edilmesi gerekir.



Sıradaki adımımızda istemci uygulamanın oluşturulması ve servis referansının eklenmesi yer almaktadır. Yine çok sevineceğiniz ve yazarken büyük keyif alacağınız bir **ConsoleApplication** :) geliştiriyor olacağız. **Console** uygulamamıza aynı **solution** içerisinde oluşturduğumuz **Ado.Net Data Service** örneğini ise **Add Service Reference** seçeneği ile ekleyeceğiz. Aynen aşağıdaki ekran görüntüsünde olduğu gibi.



Bu işlemin ardından **solution** içeriğinin aşağıdakine benzer olmasını bekleyebiliriz.



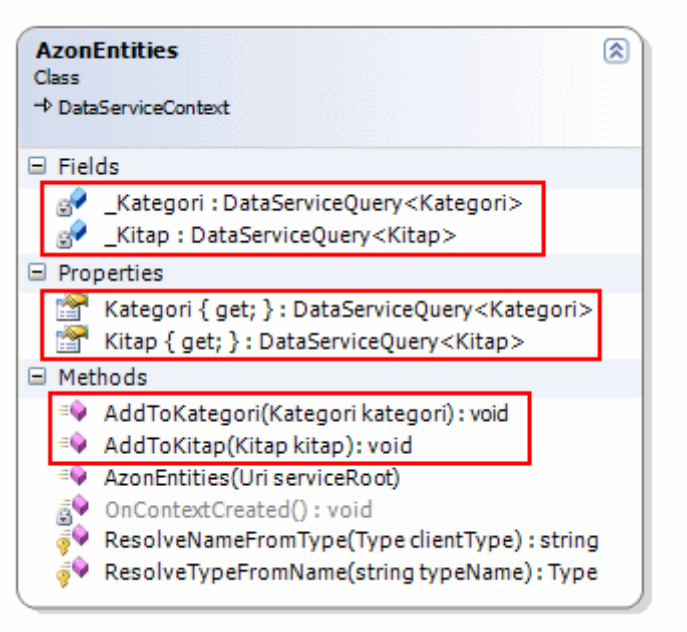
Artık birazda kod yazalım. İlk kod örneğinde toplu bir **insert** işleminide gerçekleştiriyor olacağız. İlk önce bir **Kategori** örneğini oluşturup istemci tarafındaki **Entity** nesnesine ilave edecek ve bu değişikliği veritabanına doğru göndereceğiz. Sonrasında ise

bu **Kategori** altında olacak **Kitap** nesnelerini örneklerinin değerlerini veri tabanına göndereceğiz. İşte kodlarımız;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using ClientApp.AzonSpace;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         // Proxy nesnesi örneklenir
         AzonEntities proxy = new AzonEntities(new
Uri("http://localhost:1630/HostApp/KitapServisi.svc"));

         // Yeni bir kategori nesnesi örneklenir
         Kategori windowsClient = new Kategori { Ad = "Windows Programlama" };

         // Oluşturulan Kategori nesne örneği bellek üzerindeki Entity örneğine ilave edilir
         proxy.AddToKategori(windowsClient);
         // Yeni kategorinin veritabanındaki tabloya eklenmesi için SaveChanges metodu
çağırılır.
         // (Bu noktada SaveChanges çağırılması şart değildir. Bu sadece KategoriId' nin
tablodan elde edilmesini sağlamada rol oynamaktadır)
         proxy.SaveChanges();
         Console.WriteLine("{0} ID si ile {1} Kategorisi
eklendi",windowsClient.KategoriId.ToString(),windowsClient.Ad);

         // Generic bir List koleksiyonunda tutulacak şekilde Kitap nesne örnekleri
oluşturulur.
         List<Kitap> kitaplar = new List<Kitap>()
            {
               new Kitap { Ad = "Windows Form 2.0 Programming", Fiyat = 90, StokMiktari
= 10 },
               new Kitap { Ad = "Pro WPF", Fiyat = 75, StokMiktari = 12 },
               new Kitap { Ad = "Core Windows Programming", Fiyat = 90, StokMiktari =
16 }
            };

         // Tüm kitaplar dolaşılır
         foreach (Kitap k in kitaplar)
```

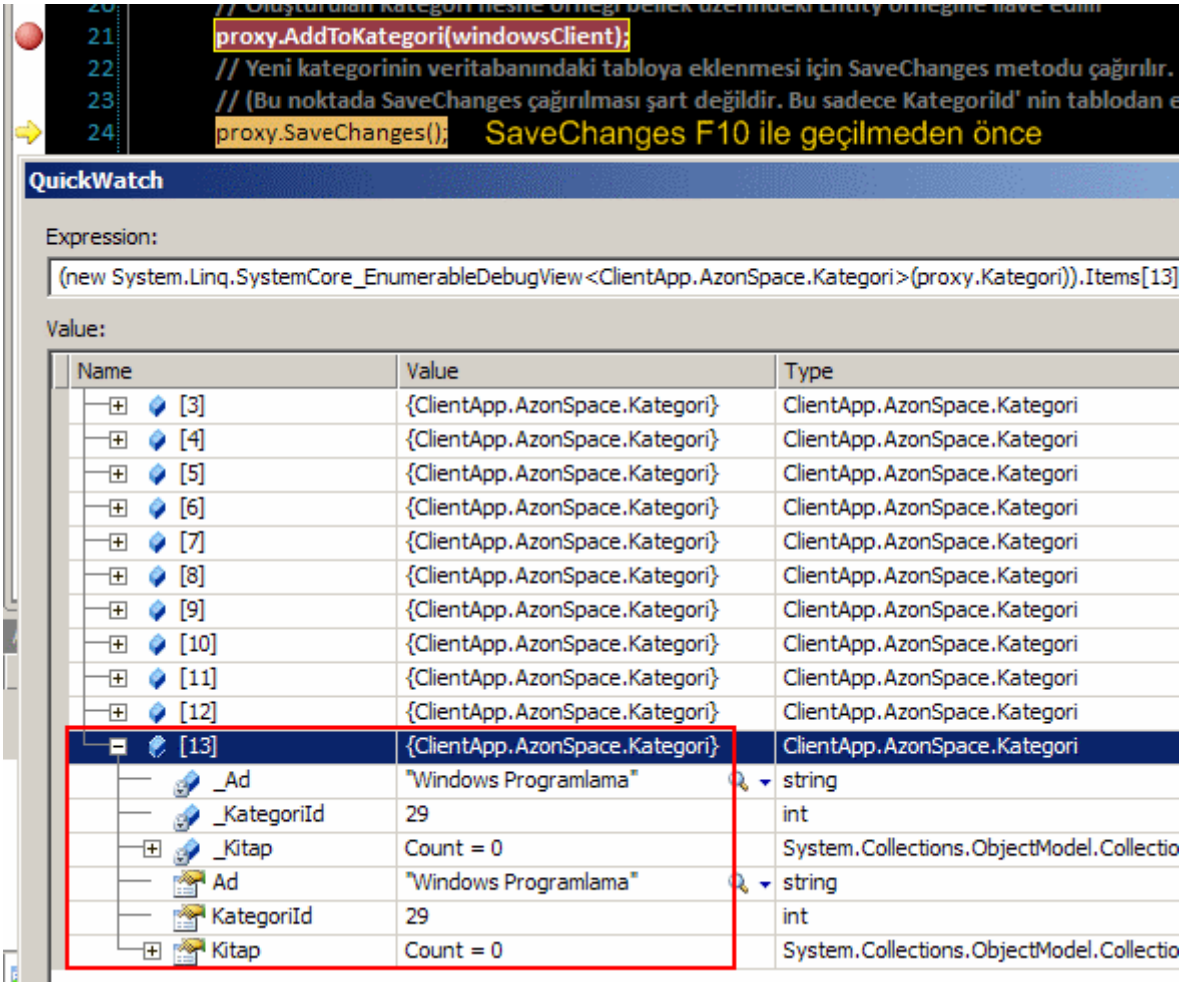
```
        {
            // Her bir Kitap nesne örneği ilgili Entity örneğine ilave edilir.
            proxy.AddToKitap(k);
            // O andaki kitap ile yukarıda oluşturulan Kategori arasındaki ilişki kurulur
            proxy.AddLink(windowsClient, "Kitap", k);
        }

        // Değişiklikler veritabanına gönderilir
        // Batch enum sabit değeri ile tüm isteklerin tek bir HTTP paketinde gönderilmesi
sağlanır
        proxy.SaveChanges(System.Data.Services.Client.SaveChangesOptions.Batch);

        // Eklenen kategori servis tarafından talep edilir
        var eklenenKategori = (from k in proxy.Kategori
                               where k.KategoriId == windowsClient.KategoriId
                               select k).First();
        // Elde edilen kategoriye ait kitap bilgilerinin yüklenmesi istenir
        proxy.LoadProperty(eklenenKategori, "Kitap");

        Console.WriteLine("\n{0} kategorisine eklenen kitaplar\n",eklenenKategori.Ad);
        // Eklenen kategoriye bağlı kitaplar listelenir
        foreach (Kitap k in eklenenKategori.Kitap)
        {
            Console.WriteLine("{0} {1} {2}
{3}",k.KitapId.ToString(),k.Ad,k.StokMiktari.ToString(),k.Fiyat.ToString("C2"));
        }
    }
  }
}
```

Kodlarda özellikle üzerinde durmamız gereken nokta bir **Entity** nesne örneğinin oluşturulması sonrasında kullanılan **AddTo[EntityAdı]**, **AddLink** ve **SaveChanges** isimli metodlardır.**AddTo[EntityAdı]** metodları, servis referansının eklenmesi sırasında istemci tarafında oluşturulan **Entity** tiplerinin her birisi için service tipine eklenir. Söz gelimi **Kitap** için **AddToKitap**,**Kategori** içinse **AddToKategori**. Bunu sınıf diagramındanda görebiliriz.

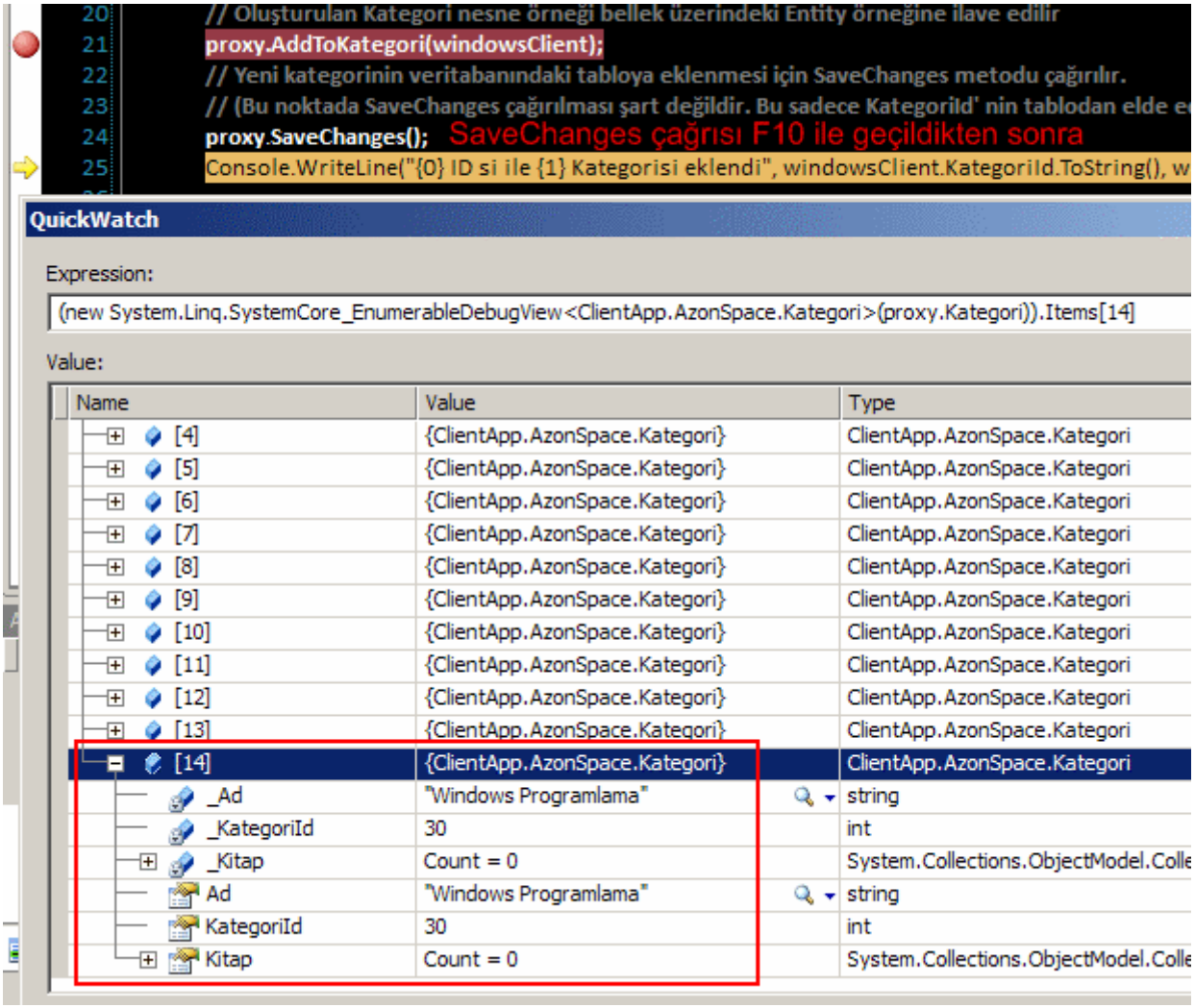


AddTo[EntityAdı] metodları parametre olarak aldıkları **Entity** nesne örneklerini taşıyıcı servis tipinin takip etmesinde rol oynarlar. Aslında kendi içlerinde **AdoToObject** metodunu çağırmaktadırlar. Bu izleme işlemi aslında **SaveChanges** metodu için önem arz etmektedir. Nitekim eklenen, silinen veya güncellenen nesne değerlerinin veritabanına doğru yansıtılması işlemini gerçekleştirmektedir. Bu noktada durup kodu **debug** ederek ilerlemenizi öneririm. özellikle **SaveChanges** metodu çağırılmadan önce **proxy** nesne örneği içerisinde **Kategori** veya **Kitap**özellikleri içeriği ile **SaveChanges** çağrısı sonrası içeriklere bakıldığında durum daha net bir şekilde görülebilir. Söz gelimi ben testleri yaparken aşağıdaki sonuçları elde ettim.

SaveChanges çağrısı öncesi;



SaveChanges çağrısı sonrası;



SaveChanges metodunun çağırılması sırasında birde **System.Data.Services.Client.SaveChangesOptions.Batch enum** sabiti değeri kullanılmıştır. Bu değer ile birden fazla **HTTP**paketinin yerine tüm isteğin tek bir **HTTP** paketi içerisinde gönderilmesi sağlanabilmektedir bu servis ile istemci arasındaki trafik akışının yoğunluğunu azaltıcı bir etkendir. *(Bu konu ile ilişkili olarak(Batching) bir görsel ders hazırlıyor olacağım.)*

Tabi **SaveChanges** çağrısı sırasında sunucu tarafındaki veri kaynağı üzerindede bir takım **SQL** sorgu ifadeleri çalışacaktır. Burada **SQL Server Profiler** aracının kullanmanızı şiddetle tavsiye ederim. örneğin veri ekleme testleri sırasında benim yakaladığım örnek sql ifadeleri aşağıdaki gibi olumuştur.

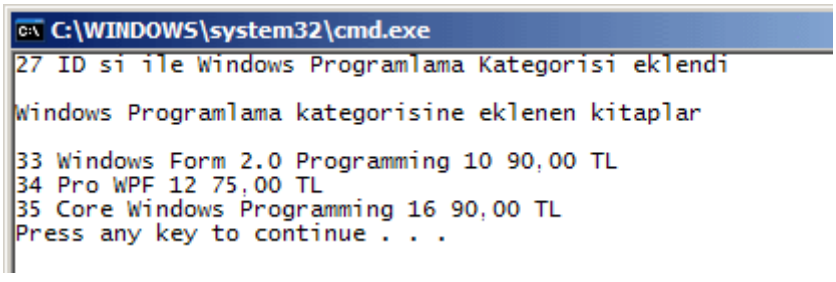
-- Kategori için Insert çağrısı
exec sp_executesql N'insert [dbo].[Kategori]([Ad])
values (@0)
select [KategoriId]
from [dbo].[Kategori]
where @@ROWCOUNT > 0 and **[KategoriId] = scope_identity()**',N'@0 nvarchar(19)',@0=N'Windows Programlama'

-- İlk Kitap için Insert çağrısı
exec sp_executesql N'insert [dbo].[Kitap]([Ad], [Fiyat], [StokMiktari], **[KategoriId]**)
values (@0, @1, @2, @3)
select [KitapId]
from [dbo].[Kitap]
where @@ROWCOUNT > 0 and **[KitapId] = scope_identity()**',N'@0 nvarchar(28),@1
decimal(19,4),@2 int,@3 int',@0=N'Windows Form 2.0
Programming',@1=90.0000,@2=10,**@3=27**

-- İkinci kitap için Insert çağrısı
exec sp_executesql N'insert [dbo].[Kitap]([Ad], [Fiyat], [StokMiktari], **[KategoriId]**)
values (@0, @1, @2, @3)
select [KitapId]
from [dbo].[Kitap]
where @@ROWCOUNT > 0 and [KitapId] = scope_identity()',N'@0 nvarchar(7),@1
decimal(19,4),@2 int,@3 int',@0=N'Pro WPF',@1=75.0000,@2=12,**@3=27**

-- üçüncü kitap için Insert çağrısı
exec sp_executesql N'insert [dbo].[Kitap]([Ad], [Fiyat], [StokMiktari], **[KategoriId]**)
values (@0, @1, @2, @3)
select [KitapId]
from [dbo].[Kitap]
where @@ROWCOUNT > 0 and [KitapId] = scope_identity()',N'@0 nvarchar(24),@1
decimal(19,4),@2 int,@3 int',@0=N'Core Windows
Programming',@1=90.0000,@2=16,**@3=27**

İlk kodda dikkat çekici noktalardan biriside **Kategori** nesne örneği eklendikten sonra **Kitap** nesne örneklerinden oluşan bir koleksiyonun nasıl ilave edildiğidir. Burada döngü içerisinde çağırılan **AddToKitap** metodu haricinde **AddLink** isimli bir fonksiyon yer almaktadır. Bu metod, o anki **Kitap** nesne örneğinin hangi **Kategori**' ye bağlı olacağının belirlenmesinde rol oynamaktadır. Bu sebepten dolayıda kodun **SQL** tarafındaki üretiminde 3 Kitap için çalıştırılan **Insert** sorgularında **KategoriID** değerleri otomatik olarak set edilmiştir. **AddLink** metodu çağırılmadığı takdirde bu ilişkinin sağlanması mümkün olmamaktadır. İlk örnek kodumuzu tamamlamış bulunuyor. İşte testler sırasında oluşan örnek bir ekran çıktısı.



Sırada güncelleştirme işlemleri var. Bu amaçla aşağıdaki örnek kod satırlarını göz önüne alabiliriz;

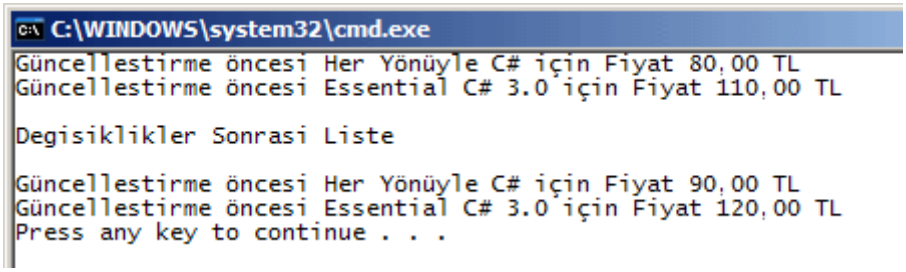
```
// Güncellenecek veri kümesi çekilir.
// örneğin KategoriId değeri 1 olan Kitaplar çekilir
var tumKitaplar = from k in proxy.Kitap
                  where k.Kategori.KategoriId==1
                  select k;

// Elde edilen sonuç kümesindeki her bir Kitap nesne örneği üzerinde basit bir
güncelleştirme yapılır
foreach (Kitap k in tumKitaplar)
{
   Console.WriteLine("Güncelleştirme öncesi {0} için Fiyat
{1}",k.Ad,k.Fiyat.ToString("C2"));
   k.Fiyat += 10;
   // Yapılan güncellemeler entity üzerinde onaylanır
   proxy.UpdateObject(k);
}
// Değişiklikler veritabanına gönderilir
proxy.SaveChanges();

// Sonuçları test etmek için servis tarafından 1 numaralı kategoriye bağlı kitaplar tekrar
istenir
Console.WriteLine("\nDeğişiklikler Sonrası Liste\n");
var kategori1Kitaplari = from k in proxy.Kitap
                         where k.Kategori.KategoriId == 1
                            select k;
// Her bir kitabın bilgisi ekrana yazdırılır
foreach (Kitap k in tumKitaplar)
{
   Console.WriteLine("Güncelleştirme öncesi {0} için Fiyat {1}", k.Ad,
k.Fiyat.ToString("C2"));
}
```

Bu kod parçasında örnek olarak **KategoriId** değeri **1** olan Kategoriye bağlı **Kitap** nesnelerinin fiyatlarının **10** birim arttırılması sağlanmaktadır. Bizim için bu kod parçasında dikkat edilmesi gereken fonksiyonellikler **UpdateObject** ve yine **SaveChanges** metodlarıdır. **SaveChanges** metodu **SQL** tarafında aşağıdaki sorgu ifadelerinin oluşmasına neden olur.

```
-- İki Update yakalanır. Nitekim 1 numaralı kategoride sadece iki Kitap vardır.
exec sp_executesql N'update [dbo].[Kitap]
set [Ad] = @0, [Fiyat] = @1, [StokMiktari] = @2
where ([KitapId] = @3)
',N'@0 nvarchar(14),@1 decimal(19,4),@2 int,@3 int',@0=N'Her Yönüyle
C#',@1=90.0000,@2=100,@3=1
```

```
exec sp_executesql N'update [dbo].[Kitap]
set [Ad] = @0, [Fiyat] = @1, [StokMiktari] = @2
where ([KitapId] = @3)
',N'@0 nvarchar(16),@1 decimal(19,4),@2 int,@3 int',@0=N'Essential C#
3.0',@1=120.0000,@2=25,@3=2
```

Burada iki adet **Update** sorgusunun oluşturulması son derece doğaldır.
Nitekim **1** numaralı **Kategori**' ye bağlı sadece iki adet **Kitap** bulunmaktadır. Kodun çalıştırılması sonrasında ise programın ekran görüntüsü aşağıdakine benzer olacaktır.



Burada hemen bir test yapmanızı öneririm.
İlk **Console.WriteLine** çağrısını **UpdateObject** metodunun sonrasına koyduğunuz takdirde **Kitap** nesne örneklerinin değerlerinin anında güncellenip güncellenmediğini(Entity tarafında) analiz edebilirsiniz.

Son olarak basit bir silme operasyonu işlemini ele alıyor olacağız. Bu son kod parçasındaki amacımız bir **Kategori** ve buna bağlı **Kitap** verilerinin silinmesini sağlamak. İşte örnek kod parçamız;

```
// önce kullanıcıya Kategori listesi sunulur
Console.WriteLine("\nSilme Operasyonu\n");
var kategoriler = from k in proxy.Kategori
                  select k;
foreach (Kategori kategori in kategoriler)
{
   Console.WriteLine("{0} {1}",kategori.KategoriId.ToString(),kategori.Ad);
}
// Kullanıcıdan silmek istediği kategorinin KategoriId değeri istenir
Console.WriteLine("Silmek istediğini kategori id' yi seçin");
int secilenKategoriId;

// Eğer ekrandan alınan değer Int32' ye Parse edilebilirse
if (Int32.TryParse(Console.ReadLine(),out secilenKategoriId))
{
   Kategori secilenKategori = null;
   try
   {
      // Ekrandan girilen ID değerine ait Kategori nesne örneği talep edilir
```

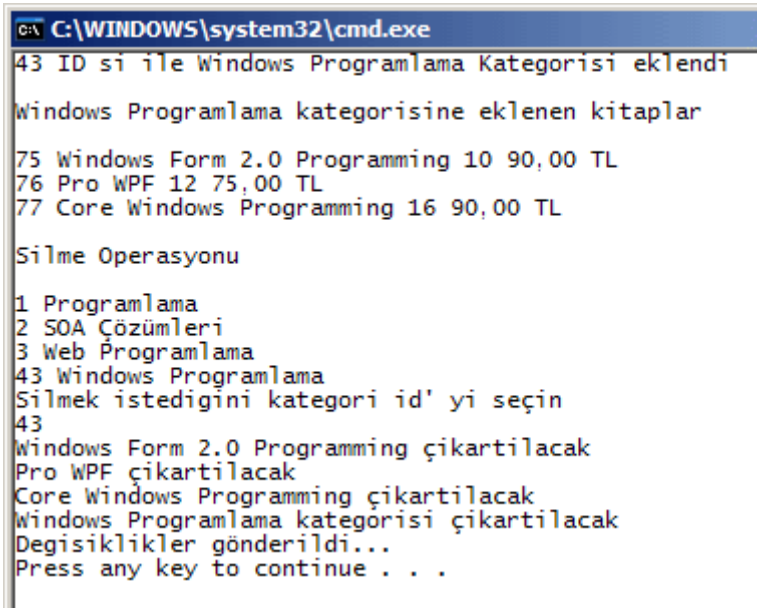
```
        secilenKategori = (from k in proxy.Kategori
                           where k.KategoriId == secilenKategoriId
                              select k).First<Kategori>();

        // önce bu Kategorinin KategoriId değerine sahip Kitap listesi alınır
        var kitapListesi = from k in proxy.Kitap
                           where k.Kategori.KategoriId == secilenKategoriId
                              select k;
        // Elde edilen her bir Kitap nesne örneği DeleteObject metodu ile çıkartılır
        foreach (Kitap kitap in kitapListesi)
        {
           proxy.DeleteObject(kitap);
           Console.WriteLine("{0} çıkartılacak",kitap.Ad);
        }

        // Son olarak seçilmiş olan Kategori nesnesi çıkartılır
        proxy.DeleteObject(secilenKategori);
        Console.WriteLine("{0} kategorisi çıkartılacak", secilenKategori.Ad);

        // Değişikliklerin veri kaynağı üzerinde de yapılması için SaveChanges metodu
çağırılır.
        proxy.SaveChanges(System.Data.Services.Client.SaveChangesOptions.Batch);
        Console.WriteLine("Değişiklikler gönderildi...");
    }
    catch
    {
    }
}
```

Kodda öncelikli olarak kullanıcıya var olan **Kategori** listesi gösterilir ve silmek istediği **Kategoriye** ait **KategoriId** değerini girmesi istenir. Bunun sonrasında söz konusu **Kategori** ve buna bağlı**Kitaplar** bulunur. önce **Kitap** nesne örnekleri tek tek **DeleteObject** metodu ile çıkartılmak üzere işaretlenir. Sonrasında ise aynı işlem seçilen **Kategori** için yapılır. Son olarak tüm işlemlerin **SaveChanges** metodu ile veritabanına gönderilmesi sağlanır. Bu noktada **SQL** tarafında oluşturulan sorgu ifadeleri aşağıdakilere benzer olacaktır. *(Bu ifadelerin yakalanması için SQL Server Profiler aracını kullandığımızı hatırlayalım)*

```
-- 43 nolu KategoriId değerine sahip Kitap verileri silinir
exec sp_executesql N'delete [dbo].[Kitap]
where (([KitapId] = @0) and ([KategoriId] = @1))',N'@0 int,@1 int',@0=75,@1=43

exec sp_executesql N'delete [dbo].[Kitap]
where (([KitapId] = @0) and ([KategoriId] = @1))',N'@0 int,@1 int',@0=76,@1=43
```

```
exec sp_executesql N'delete [dbo].[Kitap]
where (([KitapId] = @0) and ([KategoriId] = @1))',N'@0 int,@1 int',@0=77,@1=43

-- 43 nolu KategoriId değerine sahip Kategori silinir
exec sp_executesql N'delete [dbo].[Kategori]
where ([KategoriId] = @0)',N'@0 int',@0=43
```

Sonuç olarak örnek ekran çıktısı aşağıdaki gibi olacaktır.



Burada hemen bir noktayı vurgulayalım. Normal şartlar altında aynı silme operasyonunu veritabanı üzerinde gerçekleştirmek istesek, önce **Kitap** verilerini sonra ise **Kategori** verilerini silmemiz gerekirdi. Bildiğinin gibi bunun nedeni **Kategori** ve **Kitap** tabloları arasında tanımlı olan **Foreign Key** bağımlılığı ve bunun sonucu olan **Constraint** tir.Aynı mantık kod tarafında şart değildir. Bir başka deyişle önce **Kategori** için **DeleteObject** sonrasında ise **Kitap** topluluğu için **DeleteObject** metodu çağırılabilir. Nitekim organizasyon ve doğru **SQL** çalıştırma sırası **SaveChanges** metodu sonrasında oluşmaktadır. Bunu kod üzerinde deneyerek test etmenizi öneririm.

Buraya kadar yaptıklarımızı kısaca özetlersek eğer veri ekleme, güncelleme ve silme işlemleri sırasında **SaveChanges** metodunun asıl işi yüklendiğini ve veri kaynağında sorgu ifadelerinin oluşturulması için gerekli **HTTP** paketlerini hazırladığını düşünebiliriz. İstemci tarafında nesne örneği eklemek için **AddTo[EntityAdı]** yada **AddToObject** metodlarını kullanabileceğimizi gördük*(Sonuçta AddTo[EntityAdı] kendi içinde AddToObject metodunu çağırmakta).* Ayrıca silme işlemlerinde **DeleteObject** ve güncelleştirme işlemlerinde ise **UpdateObject** fonksiyonlarını ele aldık. Diğer taraftan ilişkisel nesnelerin bağlanması içinse **AddLink** fonksiyonelliğinin ele alınması gerektiğini öğrendik. Nihayetinde bir ders notumuzun sonunda daha geldik. Ancak akıllarda(en azından benim aklımda ve biraz sonra

sizin aklınızda) şöyle bir soru oluşabilir. **Eğer servis tarafında özel bir LINQ Provider ve buna bağlı tipler kullanılıyorsa Insert,Update ve Delete işlemleri nasıl gerçekleştirilebilir?** Nitekim **EDM** modelinde veri kaynağında **SQL** olduğundan bu işlemler için **SQL** sorgu ifadeleri kullanılmaktadır. İşte bu analizi bir sonraki ders notlarımızda inceliyor olacağız. Böylece geldik bir makalemizin daha sonuna. Makalemizde yer alan **CRUD** işlemlerini şu ve bu görsel derslerden de inceleyebilirsiniz. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

InsertUpdateDelete.rar (47,82 kb)

Ado.Net Data Services Ders Notları - İstemci Geliştirmek (2008-10-06T05:25:00)

ado.net data services,

Hatırlayacağınız gibi daha önceki iki ders notumuzda **Ado.Net Data Service** örneklerinin nasıl geliştirilebileceğini incelemeye çalışmıştık. Hatırlatmak gerekirse, **Ado.Net Data Service**' ler ile verilerin **Entity Data Model(EDM)** veya **Custom LINQ Provider** bazlı katmanlar üzerinden **REST** modeline göre sunulması mümkün olmaktadır. Bu noktada söz konusu servislerin**WCF**' in **REST** modelini kullanan ve **Ado.Net** üzerine odaklanmış bir açılımı olduğu görüşünde hem fikir olabiliriz. Ne varki **Servis Yönelimli Mimari(Service Oriented Architecture-SOA)** temelli çözümlerde yap-bozun en önemli iki parçasını servis ve istemciler oluşturmaktadır. Bir başka deyişle, servislerin tamamlayıcısı olan ve ilgili hizmetleri kullanacak **istemci uygulamalar(Client Applications)** olmalıdır. İşte bu yazımızda istemci uygulamaları göz önüne alacağız.

Ado.Net Data Service ve istemci arasında geçen bu hikayede, anahtar öneme sahip bir kaç kelimede yer almaktadır. **WCF**, **Ado.Net**, **REST** vb. Bunlar az çok istemcilerin kimler olabileceğinide ortaya çıkartan terimlerdir. Aslında bir servis istemcisinin herhangibir uygulama olabilmesi istenir. Platform kriterleri gözetilmeksizin. Fakat geçmiş zamanlarda sadece belirli platformlara yönelik çözümler de ele alınmamış değildir ki halen daha popüler olarak pek çok alt yapıda kullanılmaktadır. Buna verilebilecek en güzel örnek belkide **.Net Remoting** çözümleridir. .**Net Remoting** temelli uygulamalar sadece **.Net** tabanlı istemci ve sunucuları baz almaktadır. Bu bir kısıtlamadır ama performans ve verimlilik gibi avantajlarıda getirmektedir. Ancak zaman ilerledikçe farklı tipte platformların ortaklaşa haberleşebilmesi daha büyük önem arz etmeye başlamıştır. Buda **Xml Web Service**' lerin popüler olmasının nedenlerinden birisidir :) Ama uzun zamandır elimizde çok daha güçlü bir kozun olduğunu da belirtmek isterim; **Windows Communication Foundation**.

Tekrardan sihirli kelimelerimize dönelim. **WCF** kelimesi, geliştireceğimiz servisin WCF kurallarının bir sonucu olarak ortaya çıktığının açık bir göstergesidir. Buda kendi içerisinde WCF' in nimetlerini barındıracak bir servis çözümünü ifade eder ki buna **JSON(JavaScript Object Notation)**, **Syndication**, **Web Programming Model** gibi pek çok önemli kriterde dahil olur. Dolayısıyla **WCF** servislerini ele alabilen tüm istemci

çeşitleri bu senaryoda olasıdır. Ancak **Ado.Net** kelimesi, servisimizi belirli bir yöne doğru odaklamaktadır. Buna göre geliştirilen servis tamamen verilerin(Data) sunumu üzerine konuşlandırılmaktadır. Bu zorunluluk olmamakla birlikte bütünlüğü sağlayıcı bir hedef olarak görülmelidir. Buda istemci tipini belirleyici diğer bir etkendir.
Ancak **REST(REpresentationalStateTransfer)** kelimesi olayı biraz daha belirginleştirmektedir. Söz konusu istemciler **REST** modeline göre talepte bulunabilmelidir. Yani **HTTP** protokolü üzerinde **GET**,**HEAD**,**POST**,**DELETE** gibi metodlara göre talepte bulunabilmeli ve gelen sonuçlarıda irdeleyebilmelidirler.

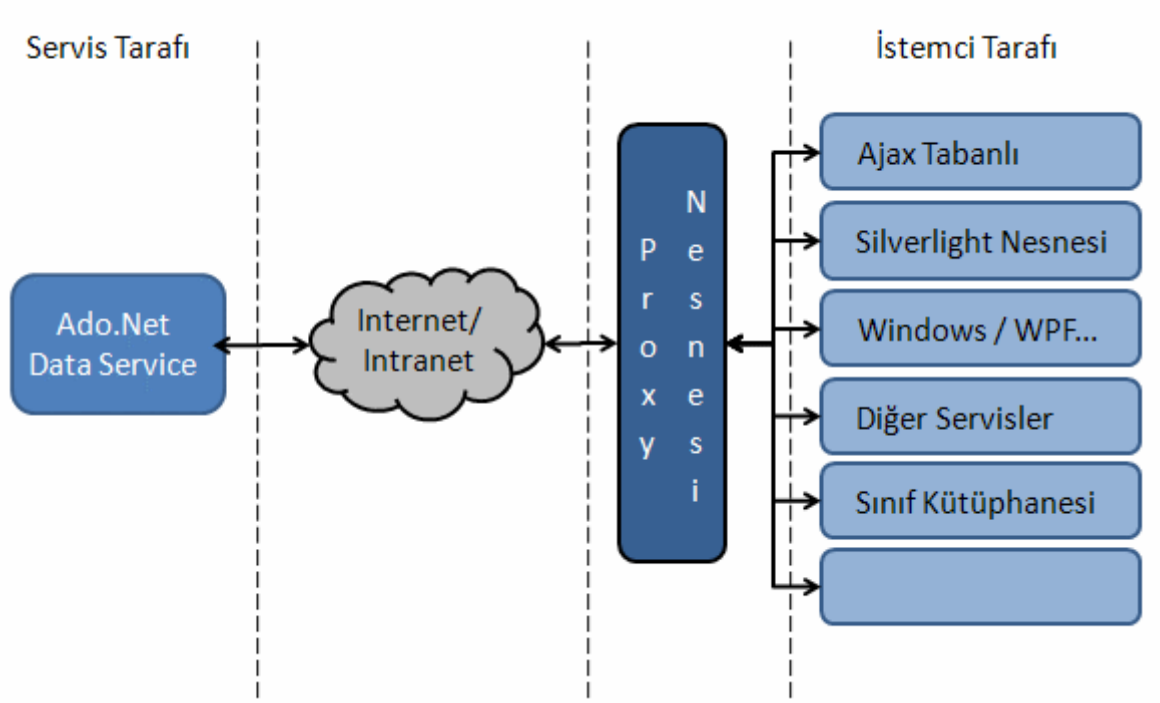
Doğruyu söylemek gerekirse bu kelimeleri bir kenara bırakıp herhangi çeşit istemci uygulama ele alınabilir diyerekten işin içerisinden de sıyrılabiliriz :) Yinede güncel teknolojiler göz önüne alındığında aşağıdaki maddelerde yer alan istemci tiplerinin dikkat çekeceği ortadadır.

- Ajax Tabanlı Web Sayfaları (Ajax Based Web Pages)
- Silverlight Nesneleri
- WPF(Windows Presentation Foundation), WinForms gibi Windows Uygulamaları
- Diğer Servisler (WCF Servisleri, Xml Web Servisleri, .Net Remoting Uygulamaları, Windows Servisleri vb...)
- Sınıf Kütüphaneleri-Class Libraries

Bu tipler çoğaltılabilir. Ancak benimde dikkatimi çeken ve özellikle üzerinde durulmaya değer çeşitler **Ajax** tabanlı web sayfaları ve **Silverlight** nesneleridir ki bunlar şu zaman itibariyle son derece popüler uygulamalardır. Yanlız dikkat edilmesi gereken başka noktalarda vardır. Söz gelimi bir **Ado.Net Data Service** örneği farklı servisler tarafındanda tüketilebilir. Böyle bir durumda tüketici servisin kendisi, aslında tükettiği servis için bir istemci olmaktadır. Yine extreme senaryolar göz önüne alınabilir. Söz gelimi **Active Directory** hizmetini özel bir **LINQ Provider** ile güvenli bir şekilde farklı bir lokasyona bir **Ado.Net Data Service** olarak sunabiliriz. örneğin dünya üzerindeki bir otele ait tüm nesnel verilerin **Active Directory** kökenli olaraktan tek bir merkezde tutulduğunu düşünün. Diğer lokasyonlardaki oteller bu merkezi verileri kullanmak isteyecektir. Bu noktada tüketici istemciler bir servis olup söz konusu hizmeti kapalı ağ içerisinde(**Intranet**diyebiliriz) ele alabilir ve diğer istemcilere sunabilir. Ki bu kapalı ağ istemcileride söz konusu lokasyondaki otelin içerisinde yer alan çeşitli tipteki uygulamalardır. Bu tabiki gerçek bir vaka değil ancak sizlerde bu cümlede bir kaç dakikalığına durup çeşitli **Ado.Net Data Service** senaryoları düşünebilir ve bunları yakın çevrenizdeki yetkin kişiler ile tartışarak analiz edebilirsiniz.

İstemci hangi çeşitten olursa olsun servis ile olan iletişimini kod seviyesinde kolaylaştırmak açısından genellikle **Proxy** nesneleri göz önüne alınır. Bu bir zorunluluk değildir. Nitekim **REST**modeline göre servise gidecek olan **HTTP** paketlerinin manuel olarak hazırlanıp gönderilmeside mümkündür. öyleki bu işlem için **HttpWebRequest** yada **HttpWebResponse** tipleri kolayca göz önüne alınabilir. Bir anlamda örneğin, **XML Web Servislerinde** bir **SOAP(Simple Object Access Protocol)** paketinin bahsettiğimiz tipler ile hazırlanıp gönderilmesinden ve geri gelen

cevabın açılarak ele alınmasından farklı bir işlem değildir. Ne varki **Proxy** kullanımı kodlamacının işini oldukça kolaylaştırır. çünkü bu sayede geliştirici bildik kodları yazarken sanki kendi ortamındaki bir nesneyi kullanıyormuş hissine kapılır. Gelip giden paket içerikleri ile uğraşmak zorunda kalmaz. Halbuki söz konusu servis talepleri, proxy tarafından servisin anlayacağı paketler haline getirilerek gönderilir. Benzer şekilde servisten gelen paketlerde proxy tarafından açılarak istemcideki **çalışma zamanı(RunTime)** nesnelerine devredilir. Bu konuda aşağıdaki şekil biraz daha aydınlatıcı bilgi verebilir.

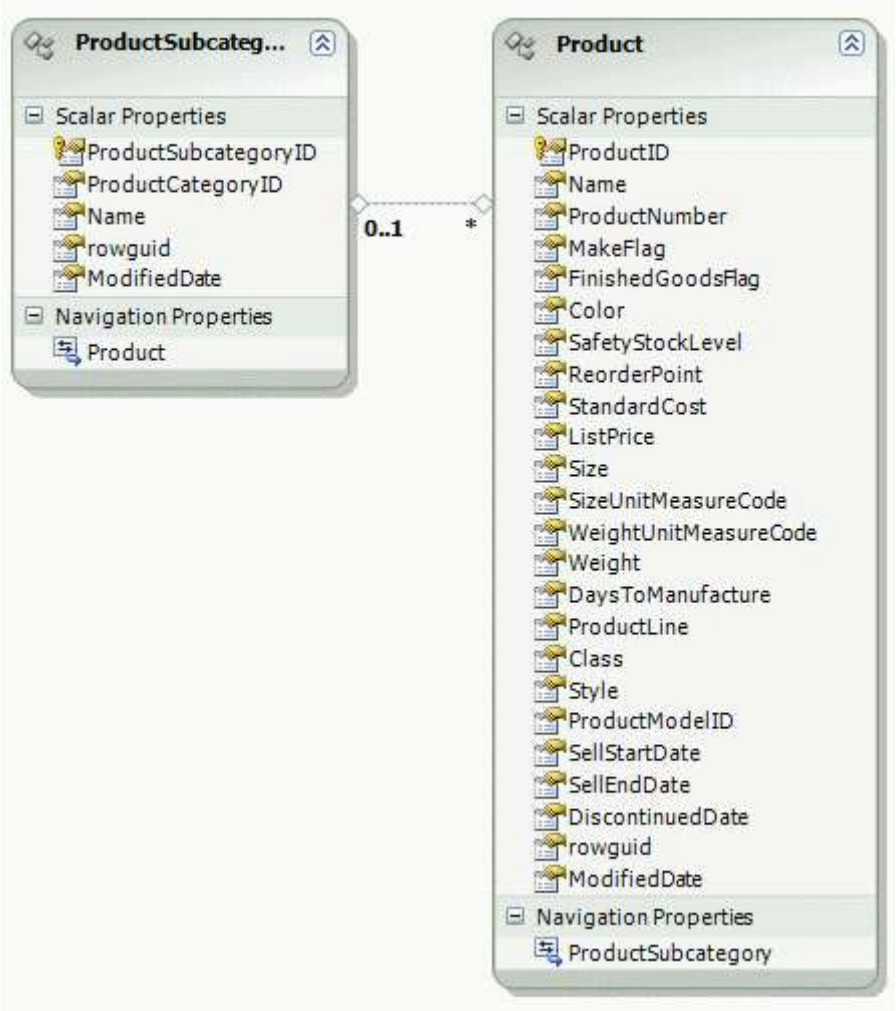


Tabi burada proxy tiplerinin geliştirme zamanında eklenmesi gibi bir zorunluluk yoktur. Bazı uygulama çeşitlerinde örneğin **Ajax** temelli web sayfalarında söz konusu proxy tiplerine ait nesne örnekleri çalışma zamanında transparant olarak oluşturulup kullanılabilirler. Ajax tabanlı bir web istemcisi üzerinden bir Ado.Net Data Service' in kullanımı çokda kolay değildir. İşin özellikle benim açımdan keyifsizleştiği nokta istemci tarafı için **javascript** kodları döktürülmesi gerekliliğidir. Ajax tabanlı bir web formunda bir **Ado.Net Data Service**' inin nasıl kullanılabileceğini ilgili görsel dersten takip edebilirsiniz.

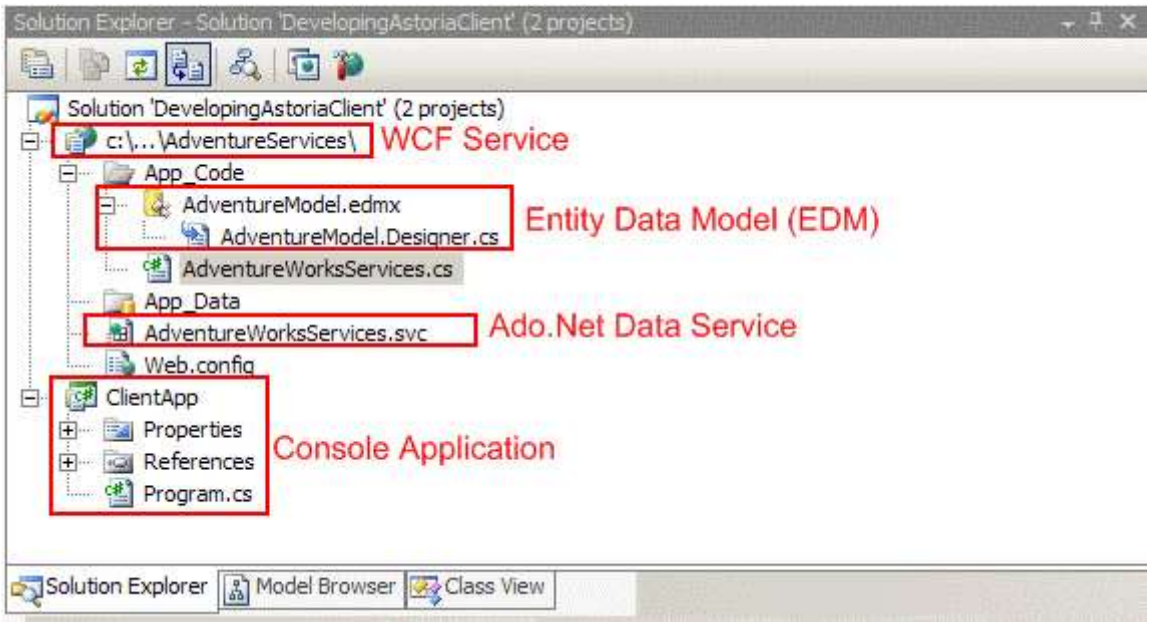
Peki biz bu yazımızda neler yapacağız? Aslında yukarıdaki listede yer almayan bir istemci uygulama geliştiriyor olacağız :) Tahmin edeceğiniz üzere bir **Console** uygulaması. Sonuçta amacımız bir istemci uygulamada basit **REST** taleplerinde bulunabilmek. Bunun içinde görsel detayların fazla olmadığı ve odağın tamamen koda kaydığı bir ortam kullanmamız öğrenmemiz açısından önemli olacaktır. İşte **Console** uygulaması seçmemizin(yada seçmemin) nedenide budur? Her zaman olduğu gibi basit bir **Ado.Net Data Service**' i geliştireceğiz. Senaryomuzda yine**AdventureWorks** veritabanını ve buradaki **ProductSubCategory**, **Product** tablolarını ele alacağız. Bu tablolar arasındaki bire çok ilişki istemci tarafında ilişkisel veri çekme işlemlerini analiz etmemizi

sağlayacaktır. Ado.Net Data Service' in nasıl geliştirileceğini daha önceki notlarımızda ve görsel derslerimizde yeterince ele almıştık. Bu nedenle sadece **EDM(Entity Data Model)** grafiğinin, **AdventureWorksServices.svc.cs** kodlarının ve **Solution** içeriğininin aşağıdakilere benzer olmasına özen göstermeniz yeterli olacaktır.*(Servis uygulamasının bir**WCF Service** şablonu olduğunu hatırlatalım.)*

EDM Grafiği;



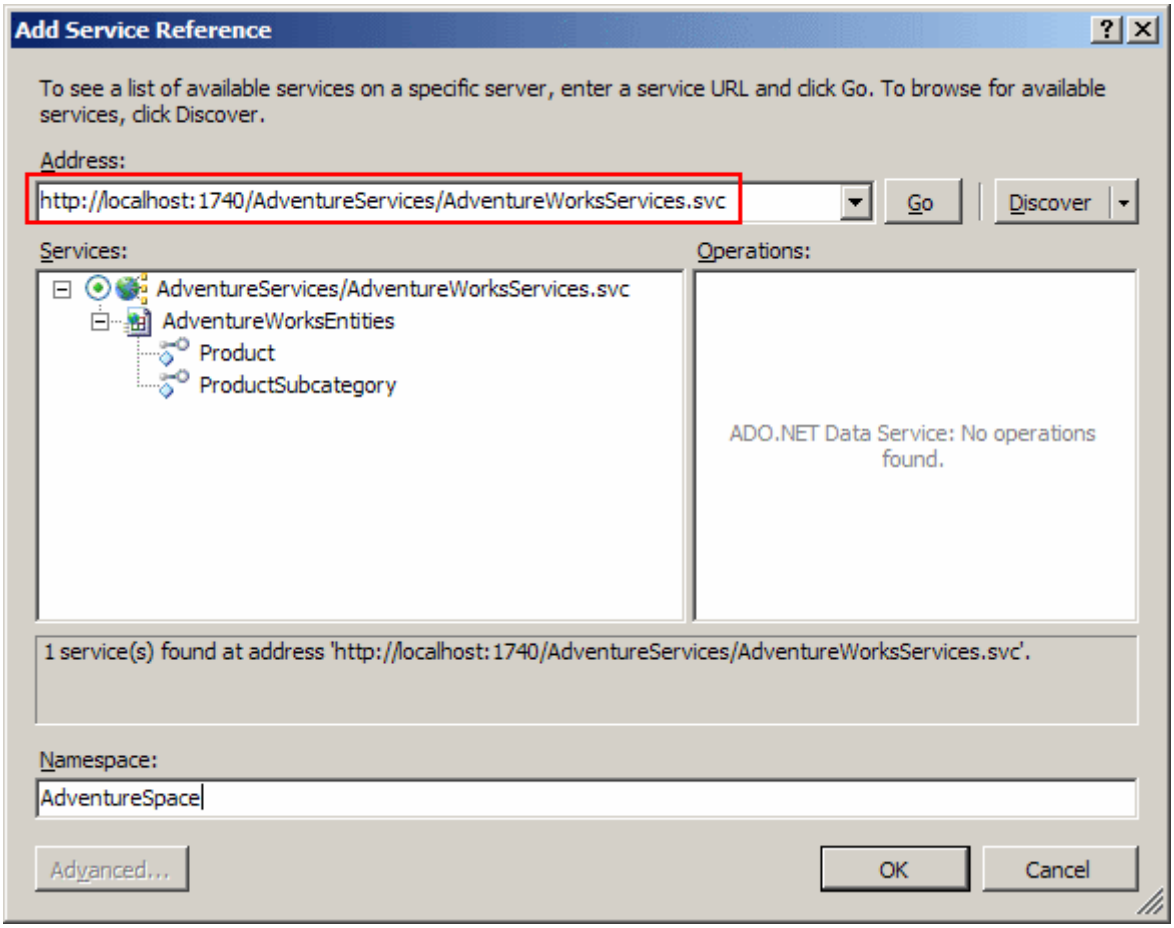
Solution İçeriği;



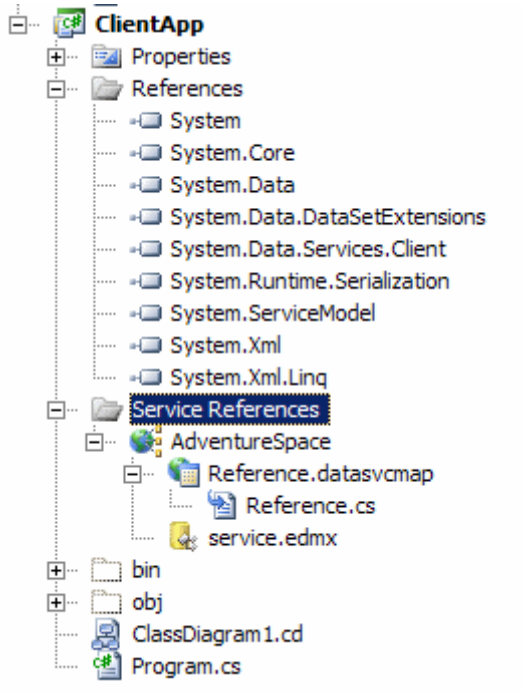
AdventureWorksServices.svc.cs içeriği;

```
using System;
using System.Data.Services;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel.Web;
using AdventureWorksModel;

public class AdventureWorksServices
    : DataService<AdventureWorksEntities>
{
    public static void InitializeService(IDataServiceConfiguration config)
    {
        config.SetEntitySetAccessRule("*", EntitySetRights.All);
    }
}
```

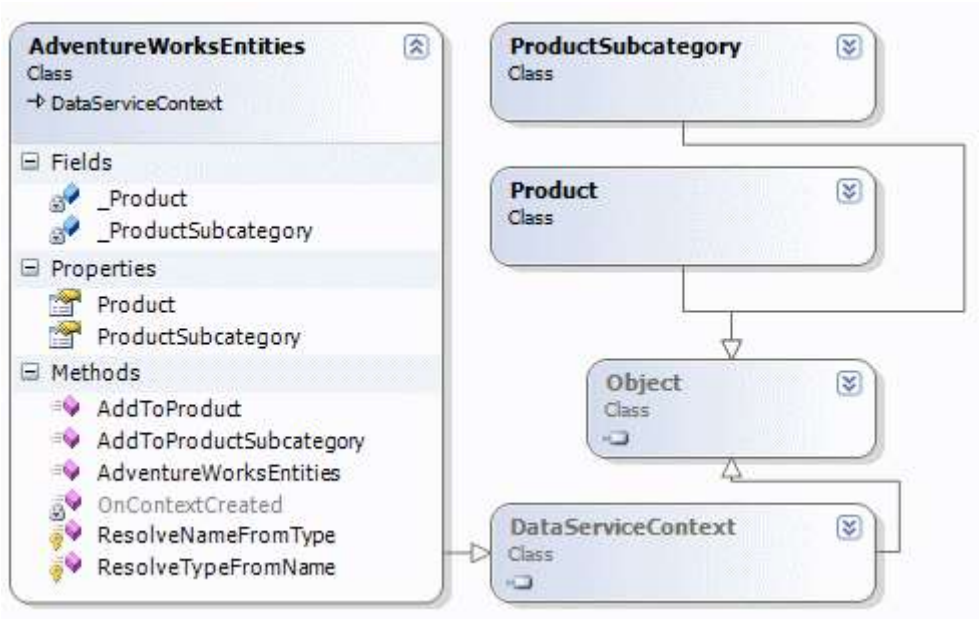
Sonrasında ise istemci **Console** uygulaması için gerekli **proxy** tiplerini üreteceğiz. **Proxy** üretimi için iki farklı seçeneğimiz bulunmaktadır. Bunlardan birisi **WCF** Servislerindende aşina oluğumuz **Add Service Reference** seçeneğidir. Bu seçeneği kullandığımızda karşımıza gelen dialog penceresinde **Ado.Net Data Service** adresinin yazılması yeterli olacaktır. Elbette geliştirilen örnek gereği **Discover** menüsünden **Services in Solution** seçeneği ele alınabilir. Nitekim servis ve istemci uygulamalarımız aynı **solution** içerisinde yer almaktadır.



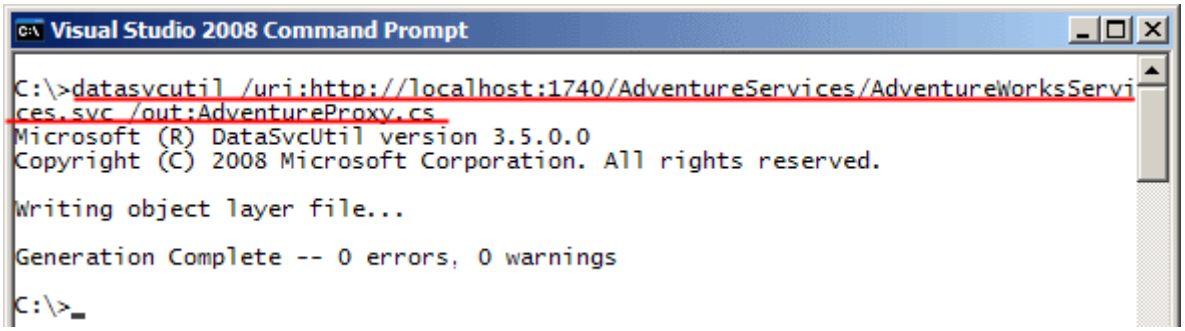
Bu noktada normal bir **WCF Service** referansı eklerken karşımıza çıkan seçeneklerden tamamının aktif olmadığı hemen göze çarpmaktadır. öyleki bir **WCF** servisi bu teknik ile eklenirken aktif olan **Advanced** düğmesi **Ado.Net Data Service** eklenirken aktif değildir. Buda olay bazlı asenkron ayarlamalar, erişim belirleyicileri(Internal veya Public) gibi bazı seçeneklerin kullanılamadığı anlamına gelmektedir. Ekleme işlemi sonrasında istemci uygulama tarafında servis ile ilişkili proxy referanslarının oluşturulduğu açık bir şekilde görülebilir.



Yine burada **edmx** uzantılı tip dikkati çekmektedir. Bu açıkçası bir **XML** içeriğidir ve istemci tarafına indirilmiş olan serileştirilebilir tipler ile ilgili eşleştirmeleri üzerinde taşımaktadır. Yine dikkat çekici noktalardan birisi istemci tarafı için bir **config** dosyası oluşturulmamış olmasıdır. Nitekim bu modelde **EndPoint** kullanımı söz konusu değildir. Zaten talepler basit**HTTP** metodları olacak şekilde servise ulaştırılmaktadır ki bu aşamada üretilen **taşıyıcı(Container)** sınıf devreye girmektedir(AdventureWorksEntities). Oluşturulan tiplere bakıldığında ise aşağıdaki sınıf diagramında yer alan açılımların oluştuğu görülür. Dikkat edileceği üzere servis tarafından sunulan entity tipleri istemci tarafındada oluşturulmuştur. Asıl yüklenici tip ise **AdventureWorksEntities** isimli **DataServiceContext** türevli sınıftır. Bu sınıf sayesinde **CRUD** operasyonlarının tamamı kolay bir şekilde istemci tarafında ele alınabilir.



Diğer **Proxy** üretme tekniği ise **SvcUtil** aracının **Ado.Net Data Service'** ler için geliştirilmiş olan versiyonu **DataSvcUtil** komut satırı programıdır. Komut satırından **proxy** üretimi
için **Visual Studio 2008 Command Prompt** üzerinden **DataSvcUtil** aracının aşağıdaki resimde olduğu gibi kullanılması yeterli olacaktır. çıktı AdventureProxy.cs isimli dosya içerisine yapılmaktadır. Uri parametresinden sonra ise Proxy üretimi için kaynak olan Ado.Net Data Service adresi verilmiştir.



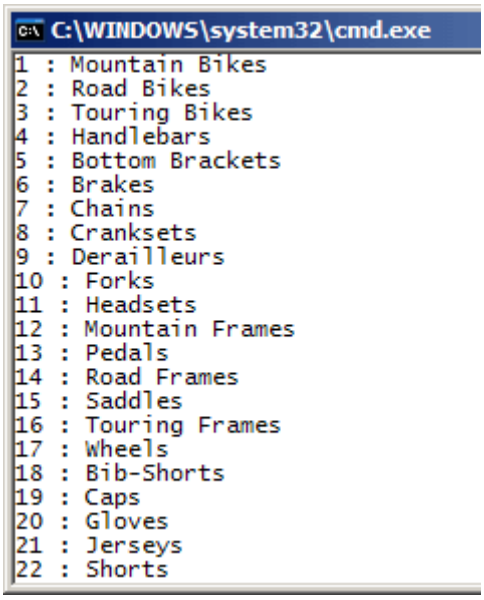
Elbette elimizde **Visual Studio 2008** gibi bir **IDE** olmadığı durumlarda **proxy** üretimi için **DataSvcUtil** aracını kullanmak gerekmektedir. Aksi durumda
ise **Add Service Reference** seçeneği çok daha mantıklıdır. Sonuç olarak ben örneğimizde **Add Service Reference** seçeneğini ele aldım.

Artık ve nihayet istemci tarafındaki kodlarımızı geliştirmeye başlayabiliriz. öncelikli olarak küçük bebek adımları ile başlamakta yarar vardır. örneğin
tüm **ProductSubCategory**listesini elde etmek istediğimizi düşünelim. Bu durumda kodlarımızı aşağıdaki gibi geliştirmemiz yeterli olacaktır.

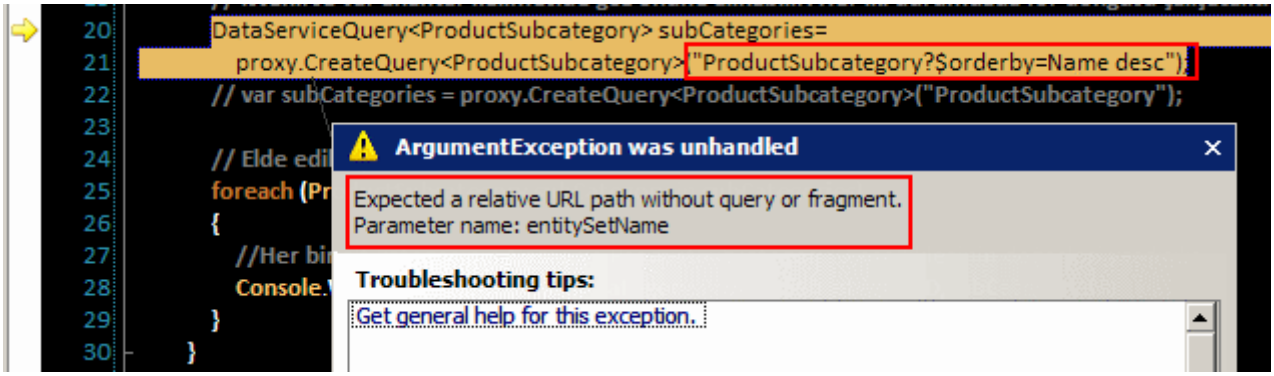
```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Data.Services.Client;
using ClientApp.AdventureSpace;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         // öncelikli olarak Proxy nesnesi örneklenir.
         // Parametre olarak Ado.Net Data Service' in URL bilgisi kullanılır.
         AdventureWorksEntities proxy = new AdventureWorksEntities(new
Uri("http://localhost:1740/AdventureServices/AdventureWorksServices.svc"));

         // CreateQuery metodu parametre olarak Entity adını almaktadır.
```

```
        // Metodun döndürdüğü sonuç kümesi DataServiceQuery tipi ile ele alınabilir.
        // İstenirse var anahtar kelimeside göz önüne alınabilir. Her iki durumdada for döngüsü çalışacaktır.
        DataServiceQuery<ProductSubcategory> subCategories=proxy.CreateQuery<ProductSubcategory>("ProductSubcategory");
        // var subCategories = proxy.CreateQuery<ProductSubcategory>("ProductSubcategory");

        // Elde edilen sonuç kümesinin her bir elemanı ProductSubcategory sınıfı tipindendir.
        foreach (ProductSubcategory subCategory in subCategories)
        {
           //Her bir alt kategorinin Name ve ProductSubcategoryID özelliklerinin değerleri yazdırılır.
           Console.WriteLine("{0} : {1}",subCategory.ProductSubcategoryID,subCategory.Name);
        }
     }
  }
}
```

Bunun sonucu olarak aşağıdakine benzer bir ekran görüntüsü elde ederiz.



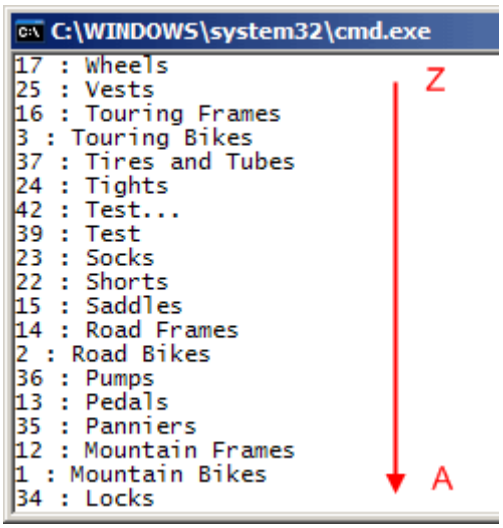
Şimdi işi biraz daha ilerletelim. Söz gelimi bu alt kategorilerin isimlerine göre tersten sıralı bir şekilde gelmesini istediğimizi düşünelim. Bu durumda **CreateQuery** metodu içerisinde**ProductSubcategory?$orderby=Name desc** şeklinde bir ifade kullanmamız kaçınılmazdır. Ne varki **CreateQuery** metodu sadece **Entity** adları ile çalışmaktadır ve bu nedenle ek parametreler alamaz. Dolayısıyla bu denemenin sonucu olarak aşağıdaki ekran görüntüsünde yer alan **istisnaya(Exception)** düşülür.



öyleyse çare nedir? Parametrik bir sorgu söz konusu ise eğer bu durumda **Execute** metodunun kullanılması gerekmektedir. Bir başka deyişle kodları aşağıdaki şekilde değiştirmek yeterli olacaktır.

```
AdventureWorksEntities proxy = new AdventureWorksEntities(new
Uri("http://localhost:1740/AdventureServices/AdventureWorksServices.svc"));

// Parametrik sorgu gönderimi için Execute metodu kullanılmalıdır.
var subCategories = proxy.Execute<ProductSubcategory>(new
Uri("/ProductSubcategory?$orderby=Name desc", UriKind.Relative));

foreach (ProductSubcategory subCategory in subCategories)
{
   Console.WriteLine("{0} :
{1}",subCategory.ProductSubcategoryID,subCategory.Name);
}
```

Program çalıştırıldığında aşağıdaki ekran görüntüsü elde edilir. Dikkat edileceği üzere alt kategoriler isimlerine göre tersten sıralı olacak şekilde elde edilmektedir.

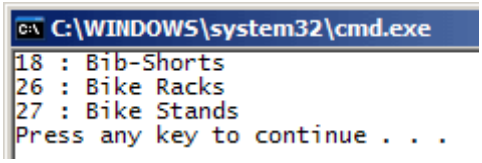


Bu sorgular son derece basittir. İşi biraz daha karıştırmaya ne dersiniz? örneğin A dan Z' ye sıralanmış alt kategorilerden ilk üçünü ve bunlara bağlı ürünleri elde etmek istediğimizi

düşünelim. Bu noktada **ProductSubcategory** ve **Product** tipleri arasındaki **ilişki(Association)** son derece önemlidir. Bu sonuçları elde etmek için kodları ilk etapta aşağıdaki gibi geliştiririz.

```
AdventureWorksEntities proxy = new AdventureWorksEntities(new
Uri("http://localhost:1740/AdventureServices/AdventureWorksServices.svc"));

var subCategories = proxy.Execute<ProductSubcategory>(new
Uri("/ProductSubcategory?$orderby=Name&$top=3", UriKind.Relative));

foreach (ProductSubcategory subCategory in subCategories)
{
   Console.WriteLine("{0} :
{1}",subCategory.ProductSubcategoryID,subCategory.Name);
   // O andaki alt kategoriye bağlı ürünleri gezmek için Product özelliğinden yararlanılır.
   foreach (Product product in subCategory.Product)
   {
      Console.WriteLine("\t {0}, {1},
{2}",product.ProductID.ToString(),product.Name,product.ListPrice.ToString());
   }
}
```

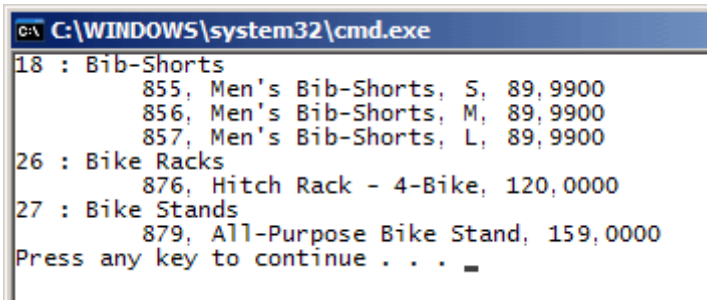
Ancak program çalıştırıldığında hiç beklenmedik bir sonuç elde edilir. Aynen aşağıdaki resimde olduğu gibi. Dikkat edileceği üzere sadece Alt kategori adları ve ID değerleri elde edilmiş bağlı olan ürün listeleri gelmemiştir.



Sebep son derece açıktır. çünkü sadece **ProductSubcategory** içeriği servis tarafından istenmiştir. Bunlara bağlı **Product** nesne toplulukları alınmamıştır. Görsel derslerimizdende hatırlayacağınız üzere **expand** anahtar kelimesininin kullanılmasının sebebide budur. Dolayısıyla kodda aşağıda görüldüğü gibi küçük bir değişiklik yapmak gerekecektir.

```
var subCategories = proxy.Execute<ProductSubcategory>(new
Uri("/ProductSubcategory?$orderby=Name&$top=3&$expand=Product",
UriKind.Relative));
```

Bu haliyle kod çalıştırıldığında aşağıdaki sonuçlar elde edilir.



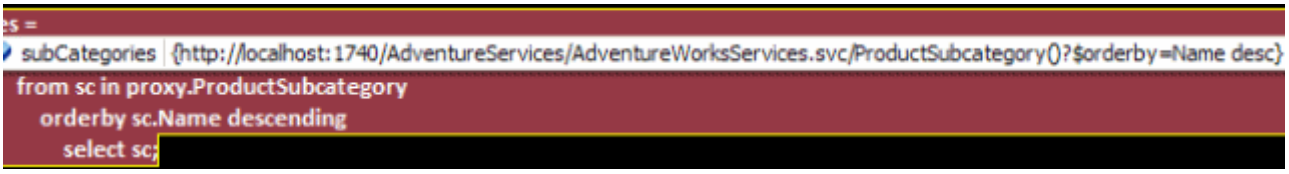
Makalenin bu kısımlarında içimden **"böylesine önemli ve bir o kadar da güzide bir teknoloji içerisinde LINQ kullanılmaz mı?"** diye geçirmiyor değilim. Tahmin ediyorumki sizlerinde bu yönde bazı beklentileri vardır. öyleyse gelin kolları sıvayalım ve aşağıdaki kod parçasını göz önüne alalım.

```
using System;
using System.Linq;
using System.Collections.Generic;
using System.Data.Services.Client;
using ClientApp.AdventureSpace;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         AdventureWorksEntities proxy = new AdventureWorksEntities(new
Uri("http://localhost:1740/AdventureServices/AdventureWorksServices.svc"));

         var subCategories = from sc in proxy.ProductSubcategory
         orderby sc.Name descending
         select sc;

         foreach (ProductSubcategory subCategory in subCategories)
         {
            Console.WriteLine("{0} :
{1}",subCategory.ProductSubcategoryID,subCategory.Name);
         }
      }
   }
}
```

Bu kez gördüğünüz gibi **Execute** yada **CreateQuery** gibi metodlar kullanmadık. Bunların yerine doğrudan bir **LINQ** sorgusu yazdık ve işte sonuç;

```
C:\WINDOWS\system32\cmd.exe
17 : Wheels
25 : Vests
16 : Touring Frames
3 : Touring Bikes
37 : Tires and Tubes
24 : Tights
42 : Test...
39 : Test
23 : Socks
22 : Shorts
15 : Saddles
14 : Road Frames
2 : Road Bikes
36 : Pumps
13 : Pedals
35 : Panniers
12 : Mountain Frames
1 : Mountain Bikes
34 : Locks
33 : Lights
21 : Jerseys
32 : Hydration Packs
31 : Helmets
```

Aslında yazılan **LINQ** sorgusu istemci tarafında bir **HTTP** ifadesinin oluşturulmasına ve servise doğru gönderilmesine neden olmaktadır.
öyleki **debug** modda **subCategories** değişkenine bakıldığında aşağıdaki ekran görüntüsünde olduğu gibi bir **QueryString** ifadesi oluştuğu gözlemlenir.



Buna göre **ProductSubcategory** ve bunlara bağlı ürünlerin elde edilmesi için yazılmış olan kod parçasında **LINQ** ifadelerini aşağıdaki gibi kullanarak aynı sonuçların elde edilebileceği açık bir şekilde görülebilir.

```
AdventureWorksEntities proxy = new AdventureWorksEntities(new
Uri("http://localhost:1740/AdventureServices/AdventureWorksServices.svc"));

// Take metodu ile A...Z ye sıralanmış listenin ilk 3 elemanı alınmış olunur.
var subCategories = (from sc in proxy.ProductSubcategory
orderby sc.Name
select sc).Take<ProductSubcategory>(3);

// Elde edilen alt kategoriler dolaşışır
foreach (ProductSubcategory subCategory in subCategories)
{
   Console.WriteLine("{0} :
{1}",subCategory.ProductSubcategoryID,subCategory.Name);
   // O andaki alt kategoriye bağlı ürünlerin çekilmesi için LoadProperty metodu kullanılır.
İkinci parametre ilişkinin taşındığı özellik adıdır.
```

```
    proxy.LoadProperty(subCategory, "Product");
    // Artık o andaki alt kategori için yüklenen Product satırları dolaşılabili
    foreach (Product product in subCategory.Product)
    {
        Console.WriteLine("\t{0} {1}
{2}",product.ProductID.ToString(),product.Name,product.ListPrice.ToString());
    }
}
```

Bu kod parçasında tek dikkat edilmesi gereken nokta **LoadProperty** özelliğinin kullanımıdır. Nitekim bu özellik ile ilişkisel veriler yüklenmediği takdirde alt kategoriye bağlı ürün listeleri servis tarafında çekilmez.

NOT : *İster **Execute** metodu olsun ister **LINQ** sorgusu olsun, ilişkisel özelliklerce taşınan verilerin çekilmesi için sırasıyla **expand** anahtar kelimesinin yada **LoadProperty** metodunun kullanılması gerekir.*

Artık sizde farklı sorgulama örnekleri deniyerek istemci tarafında neler yapılabileceğini analiz edebilirsiniz. Görüldüğü üzere bir Ado.Net Data Service' in istemci tarafından ele alınması standart bir servis kullanımına çok benzemektedir. Proxy tipleri burada işi kolaylaştırmakla birlikte LINQ sorgularınında kullanılabiliyor olması kişisel görüşüme göre son derece önemlidir.

Böylece bugünkü ders notlarımızında sonuna gelmiş bulunuyoruz. Bu ders notlarımızda basit bir istemcinin nasıl geliştirilebileceğini incelemeye çalıştık. Konu ile ilişkili olaraktan ilgiligörsel dersi takip etmenizi öneririm. Bir sonraki ders notlarımızda istemci tarafında **CRUD(CreateReadUpdateDelete)** operasyonlarının nasıl ele alınabileceğini analiz etmeye çalışacağız; ve eğer mümkün olursa özel **LINQ Provider** kullanılması halinde, servis tarafında **Insert**, **Update**, **Delete** oparasyonlarına olanak sağlamak için neler yapılması gerektiğine değiniyor olacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

DevelopingAstoriaClient.rar (57,66 kb)

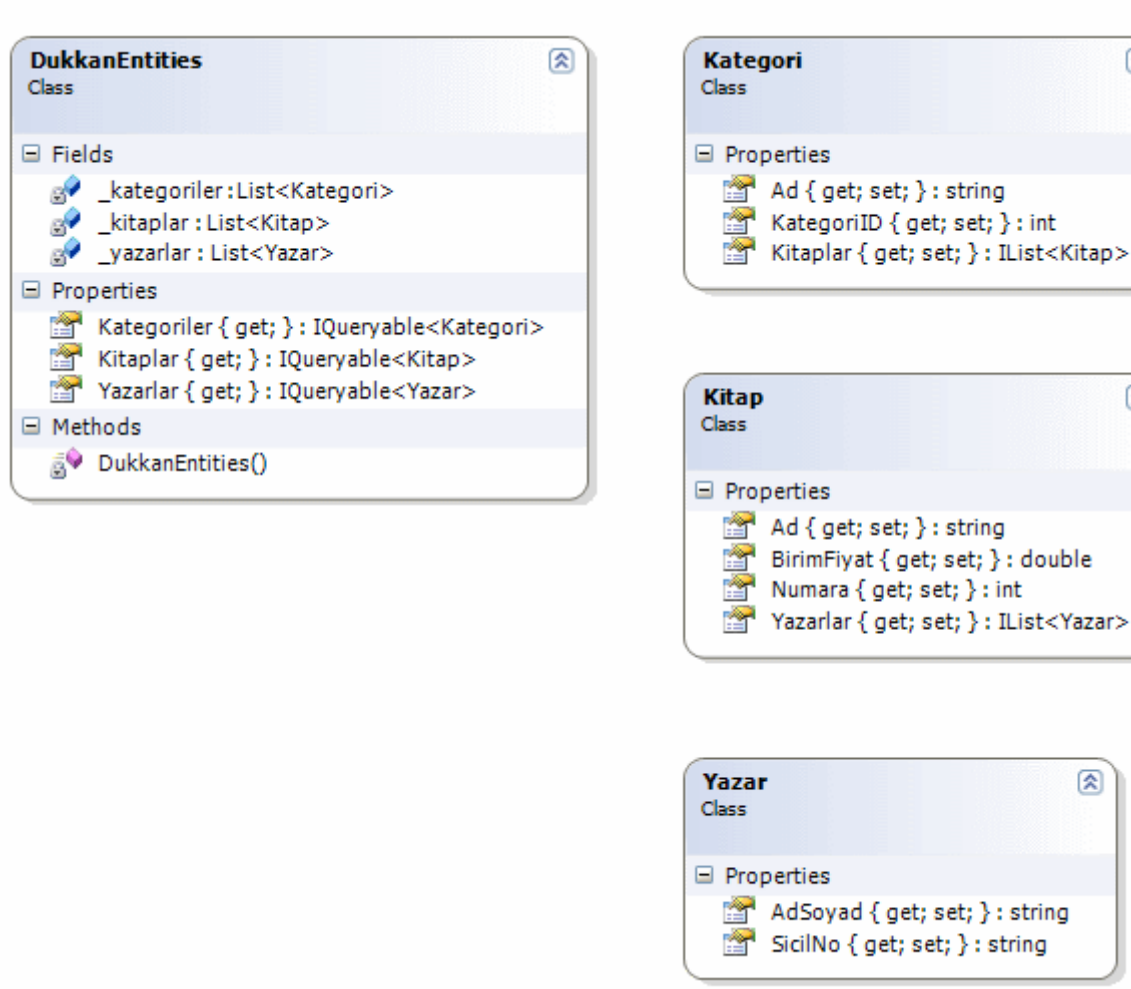
Ado.Net Data Services Ders Notları - Custom LINQ Provider (2008-09-24T05:18:00)

ado.net data services,

Son bahar yada kış gibi mevsimler ile özellikle yağmurlu ve kasvetli günlerde her geliştiricinin araştırma ve öğrenme süreci ve verimliliğinde belirgin bir artış gözlemlenir. Bu herkes için böyle olmasada en azından benim için geçerli bir durumdur. İşte bu felsefe ve ruh haliyle çıktığımız yolda son makalemizde **Ado.Net Data Services** konusuna değinmeye başlamış ve ders notlarımızı kaleme almıştık. İkinci ders notlarımızın konusu ise **LINQ Provider** kullanarak özel bir bağlama işleminin nasıl yapılabileceğini görmek.

Hatırlayacağınız gibi **Ado.Net Data Service Entry Point**' leri istemcilerden gelen **HTTP** taleplerini ele almak üzere tasarlanmaktadır. Bu tipteki servislerin belirgin amacı, veriler üzerindeki hizmetleri operasyonel bazda istemcilere sunabilmektir. Bu nedenle arka planda bir **veri erişim katmanı(Data Access Layer)** mutlaka söz konusudur. Temel olarak **DAL** içerisinde iki farklı açılım olduğundan bir önceki yazımızda kısaca bahsetmiştik. Buna göre **Entity Data Model** veya **LINQ Provider** seçenekleri söz konusudur. Bu günkü ders notlarımızda ele alacağımız **LINQ Provider**, çoğunlukla **Entity** tiplerinin geliştirici tarafından tasarlanıp farklı veri kaynaklarına bağlanması istendiği durumlarda düşünülür. Bilindiği üzere günümüzde pek çok veri kaynağı için yazılmış **LINQ Provider** araçları söz konusudur. Söz gelimi Active Directory üzerinde çalışacak şekilde tasarlanmış **LINQ to Active Directory** sağlayıcısı mevcuttur. Hatta daha ilginçlerinden biriside kütüphanemde yer alan sayısızı kitabı getirdiğim Amazon sitesi için yazılmış olan **LINQ to Amazon** sağlayıcısıdır ki kitap arama işlemleri için geliştirilmiştir. Sayısız LINQ sağlayıcısının olduğunu ve **Robert Shelton** un blog adresinden bazılarını inceleyebileceğinizi belirtmek isterim. Madem özel LINQ Provider tipleri yazılabiliyor ve çok farklı veri kaynaklarına bağlanılabiliyor, bağlanırken **LINQ** mimarisinin nimetlerinden yararlanıyor; bu sistemler neden servis bazlı olacak şekilde istemcilere yayınlanamasınlar? İşte bu noktad devreye **Ado.Net Data Services** girmektedir.

Sanıyorumki bu kadar laf kalabalığından sonra bir örnek geliştirerek hareket etmekte yarar vardır. İşe yine **Ado**.**Net Data Service** örneğinin **Host** edileceği bir **WCF Service** uygulaması ile başlanacaktır. Sonrasında ise kendi **LINQ Provider** ortamımız için gerekli tipler ve üyeleri geliştirilecektir. Tüm bu işlemler tamamlandığında ise her zamanki gibi istemci uygulama yazmaya gerek kalmadan basit bir tarayıcı üzerinde gerekli testler yapılacaktır. öncelikli olarak **DepoServisleri** isimli bir **WCF Service** uygulamasını **Visual Studio 2008 Professional Service Pack 1**üzerinde oluşturalım. Sonrasında ise sınıf diagramı ve kod içeriği aşağıdaki gibi olan tipleri projemize ilave edelim.




```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using System.Data.Services.Common;

public class Kategori
{
   public int KategoriID { get; set; }
   public string Ad { get; set; }
   // Kategori bağlı Kitaplara ulaşmak için IList<Kitap> tipinden bir özellik
kullanılmaktadır
   public IList<Kitap> Kitaplar { get; set; }
}

// DataServiceKey niteliği ile Kitap sınıfının anahtar-Key özelliğinin Numara olduğu
belirtilir.
[DataServiceKey("Numara")]
public class Kitap
```

```
{
   public int Numara { get; set; }
   public string Ad { get; set; }
   public double BirimFiyat { get; set; }
   // Bir kitabın yazalarına geçiş yapmak için IList<Yazar> tipinden bir özellik kullanılır
   public IList<Yazar> Yazarlar { get; set; }
}

// Yazar sınıfı için Key özelliği string tipinden olan SicilNo' dur.
[DataServiceKey("SicilNo")]
public class Yazar
{
   public string SicilNo { get; set; }
   public string AdSoyad { get; set; }
}

// Entity tiplerini içerisinde barındıran sınıf
public class DukkanEntities
{
   static List<Kategori> _kategoriler;
   static List<Kitap> _kitaplar;
   static List<Yazar> _yazarlar;

   // static yapıcı DukkanEntities' e ait taleplerin sayısı ne olursa olsun ilk seferde bir
kereliğine çalıştığı için, bellek üzerinde tutulacak nesne topluluklarının oluşturulması
sırasında performans kazanımı sağlamaktadır.
   static DukkanEntities()
   {
      // örnek kategori verileri oluşturulur.
      _kategoriler = new List<Kategori>
         {
            new Kategori{ KategoriID=1, Ad="Bilgisayar Kitapları"},
            new Kategori{ KategoriID=2, Ad="Bilim Teknik Kitapları"},
            new Kategori{ KategoriID=3, Ad="çizgi Roman"},
         };

      // örnek kitap verileri oluşturulur.
      _kitaplar = new List<Kitap>
         {
            new Kitap{ Numara=9001, Ad="Red Kit", BirimFiyat=10},
            new Kitap{ Numara=9002, Ad="Ten Ten' in Maceraları", BirimFiyat=9.99},
            new Kitap{ Numara=9003, Ad="örümcek Adım", BirimFiyat=8.99},
            new Kitap{ Numara=9004, Ad="Batman", BirimFiyat=6.99},
            new Kitap{ Numara=9005, Ad="Barbar Conan", BirimFiyat=13.44},
            new Kitap{ Numara=9006, Ad="Superman", BirimFiyat=12.34},
```

```
            new Kitap{ Numara=9007, Ad="Martin Myster", BirimFiyat=23.45},
            new Kitap{ Numara=8002, Ad="Programming C#", BirimFiyat=45},
            new Kitap{ Numara=8003, Ad="LINQ Unleashed", BirimFiyat=44.45},
            new Kitap{ Numara=7450, Ad="Essential WCF For .Net Framework 3.5",
BirimFiyat=44.50},
            new Kitap{ Numara=1240, Ad="Fermant' nın Son Teoremi", BirimFiyat=5},
            new Kitap{ Numara=2450, Ad="Sayıların Gücü", BirimFiyat=7.5},
            new Kitap{ Numara=2470, Ad="Evrenin Kısa Tarihi", BirimFiyat=9.99}
        };

    // Kategori ve Kitaplar arasındaki ilişkiler veriler üzerinden sağlanır
    for (int i = 0; i < _kategoriler.Count; i++)
        _kategoriler[i].Kitaplar = new List<Kitap>();
    for (int i = 0; i <= 6; i++)
        _kategoriler[2].Kitaplar.Add(_kitaplar[i]);
    for (int i = 7; i <= 9; i++)
        _kategoriler[0].Kitaplar.Add(_kitaplar[i]);
    for (int i = 10; i <= 12; i++)
        _kategoriler[1].Kitaplar.Add(_kitaplar[i]);

    // örnek yazarlar oluşturulur
    _yazarlar = new List<Yazar>
        {
            new Yazar{ SicilNo="Y100", AdSoyad="Ali"},
            new Yazar{ SicilNo="Y101", AdSoyad="Veli"},
            new Yazar{ SicilNo="Y104", AdSoyad="Mehmet"},
            new Yazar{ SicilNo="Y103", AdSoyad="Burak"},
            new Yazar{ SicilNo="Y107", AdSoyad="Selim"},
            new Yazar{ SicilNo="Y108", AdSoyad="Kamil"},
            new Yazar{ SicilNo="Y109", AdSoyad="Cemil"},
            new Yazar{ SicilNo="Y110", AdSoyad="Nazlı"},
            new Yazar{ SicilNo="Y120", AdSoyad="Ayşe"},
            new Yazar{ SicilNo="Y111", AdSoyad="Fatma"},
            new Yazar{ SicilNo="Y110", AdSoyad="Melike"}
        };

    // Kitaplar ile yazalar arasındaki verisel bağlantılar sağlanır
    Random rnd = new Random();
    for (int i = 0; i < _kitaplar.Count; i++)
    {
        _kitaplar[i].Yazarlar = new List<Yazar>();
        for(int j=0;j<3;j++)
            _kitaplar[i].Yazarlar.Add(_yazarlar[rnd.Next(0,_yazarlar.Count-1)]);
    }
}
```

```
    // REST modelinde talep edilebilecek özellikler IQueryable<T> tipinden tanımlanır
    public IQueryable<Kategori> Kategoriler
    {
        get
        {
            return _kategoriler.AsQueryable<Kategori>();
        }
    }
    public IQueryable<Kitap> Kitaplar
    {
        get
        {
            return _kitaplar.AsQueryable<Kitap>();
        }
    }
    public IQueryable<Yazar> Yazarlar
    {
        get
        {
            return _yazarlar.AsQueryable<Yazar>();
        }
    }
}
```

Yukarıdaki kod satırları ilk etapta karmaşık görünebilir ancak işin teorisi oldukça kolaydır. Herşeyden önce **REST** modeline göre dışarıya sunmak isteyeceğimiz tiplerin bir tasarımının servis tarafında olması gerekir. Bu tasarımın karşılığı tahmin edileceği üzere sınıftır. örnek senaryoda bir kitap dükkanında olması muhtemel materyallere ait sınıflar tasarlanmıştır. **Kategori**, **Kitap** ve **Yazar**. Elbetteki bunlar tamamen farazi tasarımlardır. önemli olan ve kavramamız gereken nokta, **Entity Data Model**' dan bağımsız ve **LINQ** yapısını kullanarak **Ado.Net Services** için gerekli **Data Access Layer**' ın geliştirilmesidir.

Bir önceki makaleyi hatırlıyorsak eğer, **URL** satırından **ProductSubcategory(2)** gibi taleplerde bulunabileceğimizi görmüştük. Buradaki **2** sayısalı aslında, **ProductSubCategory**tablosunun **PrimaryKey** olarak set edilmiş **ProductSubCategoryID** alanının değeriydi. Dolayısıyla **EDM** içerisinde bunu işaret edecek biçimde tasarlanmış **Property**' ler söz konusudur. Şimdi tekrardan Kategori, Kitap ve Yazar sınıflarına dönelim. Dikkat edileceği üzere Kitap ve Yazar sınıflarının başında **DataServiceKey** isimli bir **nitelik(Attribute)** kullanılmaktadır. Bu nitelik sayesinde sınıfın **anahtar özelliği(Key Property)** belirlenir. **Kitap** için bu özellik **int** tipinden olan **Numara** iken, **Yazar** için **string** tipinden olan **SicilNo**' dur. Tabi burada diğer bir vaka daha vardır. Kategori sınıfı için bu tip bir nitelik kullanılmamıştır. Bunun sebebi ise, çalışma zamanının **Key Property**' lere bakarken

ya **[SınıfAdı][ID]** yada **ID** isiminde özellikler aramasıdır. Dikkat edilecek olursa **Kategori** sınıfında **KategoriID** isimli bir özellik bulunmaktadır.

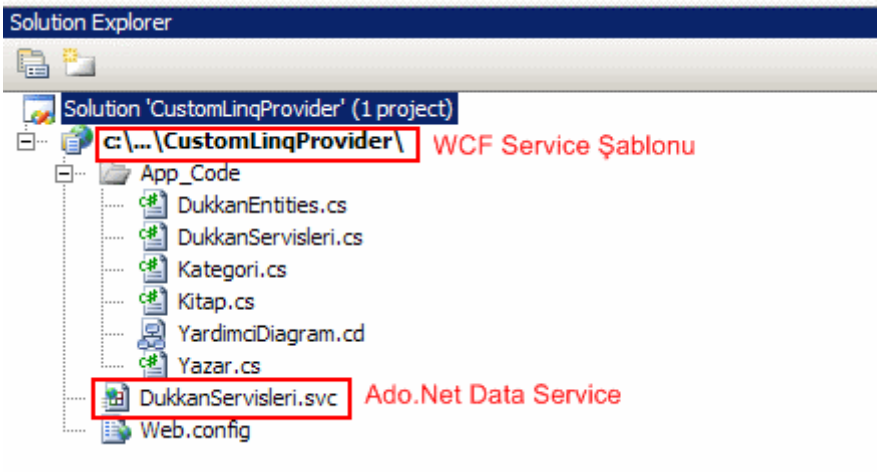
NOT : DataServiceKey *niteliği(Attribute) sadece sınıf seviyesinde(Class Level) uygulanabilir ve* ***System.Data.Services.Common*** *isim alanı(Namespace) altında yer almaktadır. İki adet aşırı yüklenmiş yapıcısı(Overloaded Constructor) vardır. İstenirse bir sınıfın birden fazla özelliğine* ***Key Property*** *niteliğinin kazandırılmasını sağlayabilir.*

Sınıf tasarımlarında dikkat çekici noktalardan bir diğeri ise **IList<T>** tipinden özelliklerin kullanılmasıdır. örneğin **Kategori** ve **Kitap** sınıfları içerisinde sırasıyla **IList<Kitap>** ve**IList<Yazar>** tipinden özellikler yer almaktadır. Bu yaklaşım **EDM**' nin kullandığının neredeyse aynısıdır. Amaç; bir entity nesne örneği üzerinden ilişkisel diğer entity nesne örneklerine geçiş yapabilmektir. Söz konusu entity nesne örnekleri birer sınıf olduğundan bu geçişte liste bazlı özelliklerin kullanılması son derece doğaldır.

Buraya kadarki kısımlarda kafa karıştırıcı herhangibir nokta olmadığı düşüncesindeyim. Eğer sıkıldıysanız benim gecenin ilerleyen şu saatlerinde yaptığım gibi sıcak bir nescafeyi yudumlanızı yada demlenmiş güzel bir çayı içmenizi tavsiye ederim.

Artık kalan detaylara bakabiliriz. Servis tarafında dışarıya sunulacak olan tipler birer koleksiyon içerisinde tutulmalıdır. Hatta bu koleksiyonları birer özellik olarak barındıracak bir başka deyişle **entity** tiplerini içerecek bir **taşıyıcının(Container)** var olmasıda gerekir. Bu nedenle **DukkanEntity** isimli bir sınıf tasarlanmalıdır. Bu sınıf içerisinde bellek üzerinde tutulacak olan **Entity** nesneleri için gerekli üyeler ve kodlar yer almaktadır. Ancak dikkat edilmesi gereken en önemli unsur **Entity** özelliklerinin **IQueryable<T>** tipinden tanımlanmış olmamılardır ki**Ado.Net Data Services** tarafı için bu sorgulanabilmeyi sağlayan küçük bir detaydır. Bu amaçla **Kategoriler**, **Kitaplar** ve **Yazarlar** isimli **yanlız okunabilir özellikler(Read Only Properties)**oluşturulmuş ve **DukkanEntities** sınıfı içerisine dahil edilmiştir. Söz konusu özelliklerin **get** bloklarında **AsQueryable** metodunun kullanıldığı hemen dikkat çekmektedir. AsQueryable fonksiyonu, IList<T> tipinden olan sınıf için liste bazlı koleksiyonların IQueryable<T> arayüzü(Interface) tarafından taşınabilmesi için geliştirilmiş bir metoddur.

Sanıyorumki yazıyı okumakta olan herkesin içinde(ve hatta yazmakta olan benim) sistemin nasıl çalışacağına dair yoğun bir merak var. Bu merakı gidermeden önce yapmamız gereken küçük bir iş daha var. Oda **Ado.Net Data Service**' i WCF Service tipindeki projemize Add New Item seçeneği ile eklemek. *(Bu noktada hemen şunu vurgulayalım; Ado.Net Data Service tiplerinin illede bir WCF Service şablonunda olması şart değildir. İstenirse bir Web projesine eklenebilir. Hatta istenirse manuel olarak bir host uygulamada yazılabilir ki bu konuya ilerleyen yazılarımızda değiniyor olacağız.)*

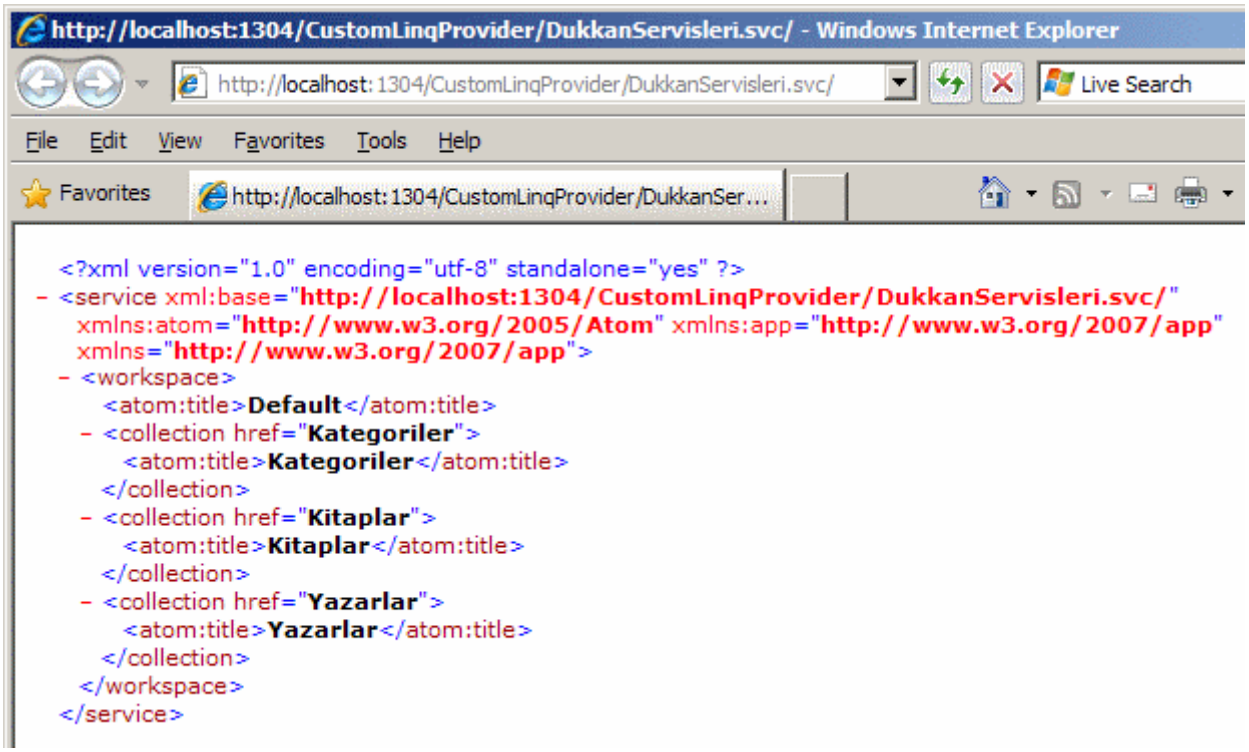


örneğe **DukkanServisleri.svc** olarak eklenen **Ado.Net Data Service**' in kod içeriği ise aşağıdaki gibi geliştirilebilir.

```
using System;
using System.Data.Services;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel.Web;

public class DukkanServisleri
    : DataService<DukkanEntities>
{
   public static void InitializeService(IDataServiceConfiguration config)
   {
      config.SetEntitySetAccessRule("*", EntitySetRights.AllRead);
   }
}
```

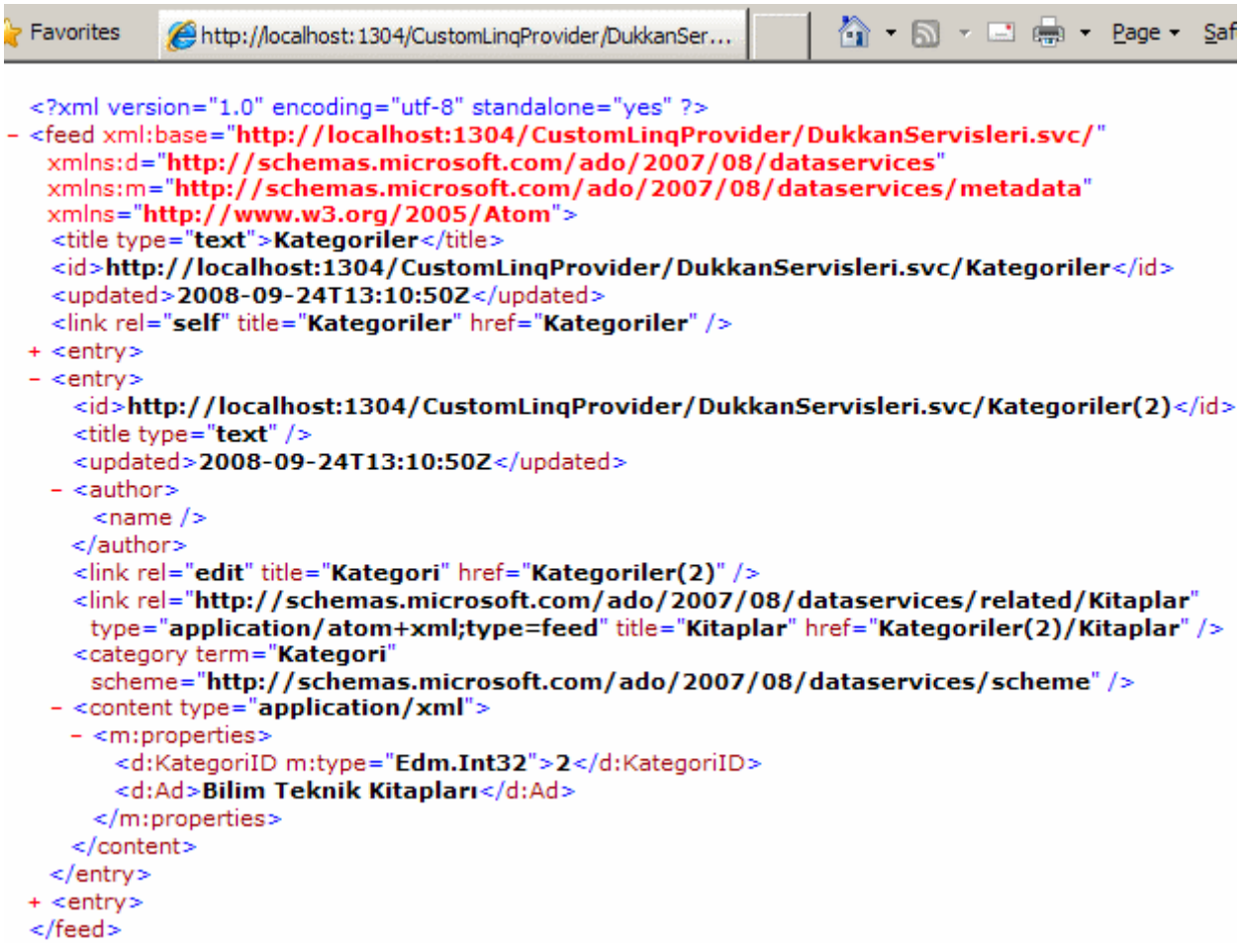
Her zamanki gibi servis örneklenirken devreye giren **InitializeService** metodu içerisinde, gerekli erişim hakları tanımlanır. ***** işaretini kullandığımız için **DukkanEntities** içerisinde tanımlı**IQueryable<T>** temelli tüm özellikler dışarıya açılmaktadır. **AllRead** sabit değeri kullanıldığı içinde, söz konusu tiplerin sadece okuma amaçlı olaraktan dış ortama sunulması sağlanır.

Artık örnekler test edilebilir. Bir önceki makalemizdede belirttiğimiz gibi herhangibir istemci uygulama yazmamıza gerek yoktur(Bu noktada biraz tembellik ettiğimi açıkça belirtebilirim). Basit bir tarayıcı uygulama, örneğin **Internet Explorer** bizim için yeterlidir. örnekteki kodları doğru olarak geliştirdiysek eğer, DukkanServisleri.svc için çalışma zamanında aşağıdakine benzer bir çıktının alınması gerekir. Eğer sizlerde eş zamanlı geliştirdiğiniz örneğinizde benzer sonuçları alıyorsanız herşey yolunda gidiyor demektir.



Sistem çalıştığına göre bir kaç sorgulama denemesi yapılabilir. Aşağıda test amaçlı bir kaç sorgu yer almaktadır.

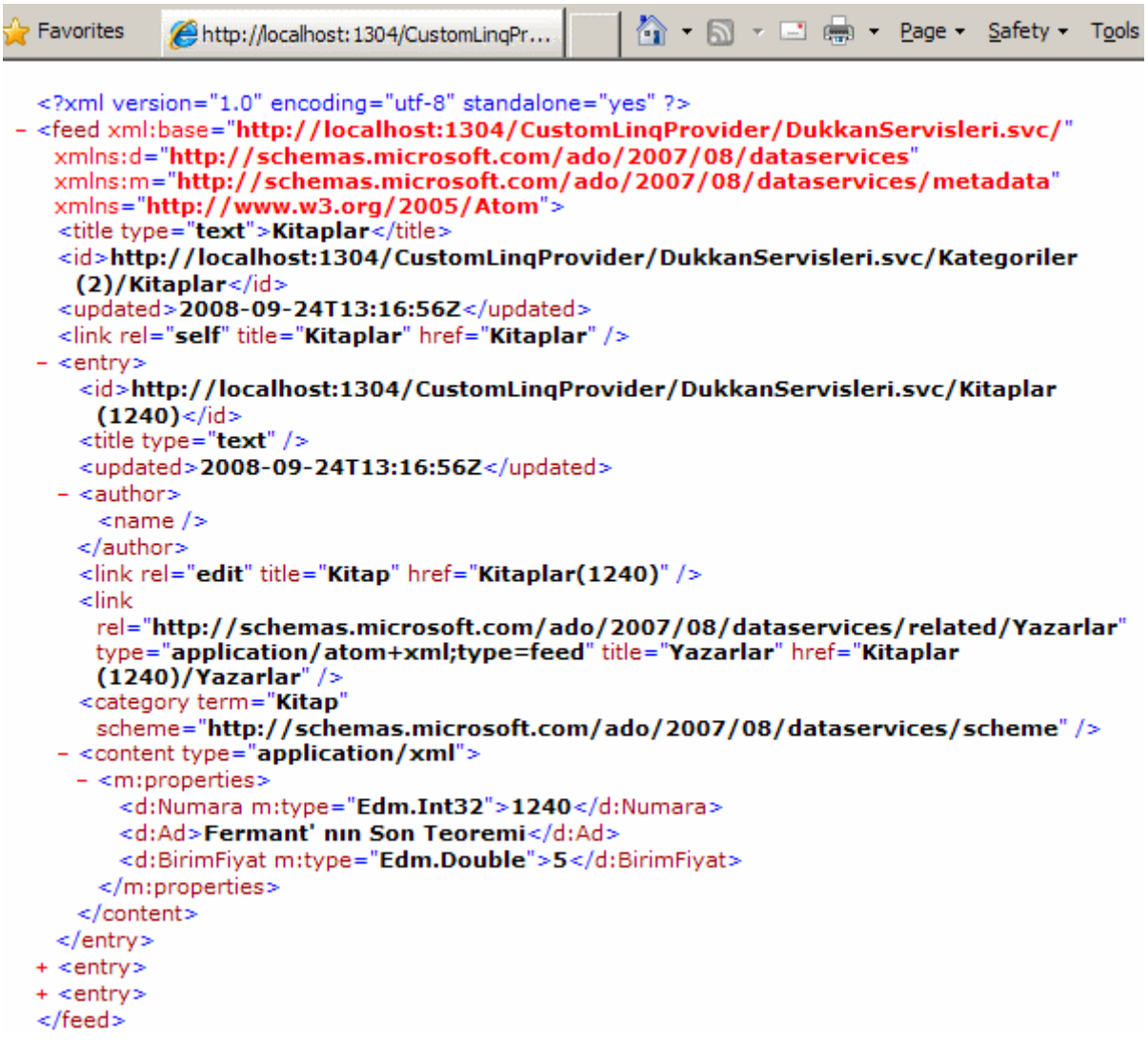
örnek 1: Tüm Kategorilerin elde edilmesi.
URL : http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/**Kategoriler**




```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <feed xml:base="http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns="http://www.w3.org/2005/Atom">
    <title type="text">Kategoriler</title>
    <id>http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/Kategoriler</id>
    <updated>2008-09-24T13:10:50Z</updated>
    <link rel="self" title="Kategoriler" href="Kategoriler" />
  + <entry>
  - <entry>
      <id>http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/Kategoriler(2)</id>
      <title type="text" />
      <updated>2008-09-24T13:10:50Z</updated>
    - <author>
        <name />
      </author>
      <link rel="edit" title="Kategori" href="Kategoriler(2)" />
      <link rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Kitaplar"
        type="application/atom+xml;type=feed" title="Kitaplar" href="Kategoriler(2)/Kitaplar" />
      <category term="Kategori"
        scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    - <content type="application/xml">
      - <m:properties>
          <d:KategoriID m:type="Edm.Int32">2</d:KategoriID>
          <d:Ad>Bilim Teknik Kitapları</d:Ad>
        </m:properties>
      </content>
    </entry>
  + <entry>
  </feed>
```

örnek 2: İki numaralı kategorideki kitapların elde edilmesi

URL : http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/**Kategoriler(2)/Kitaplar**

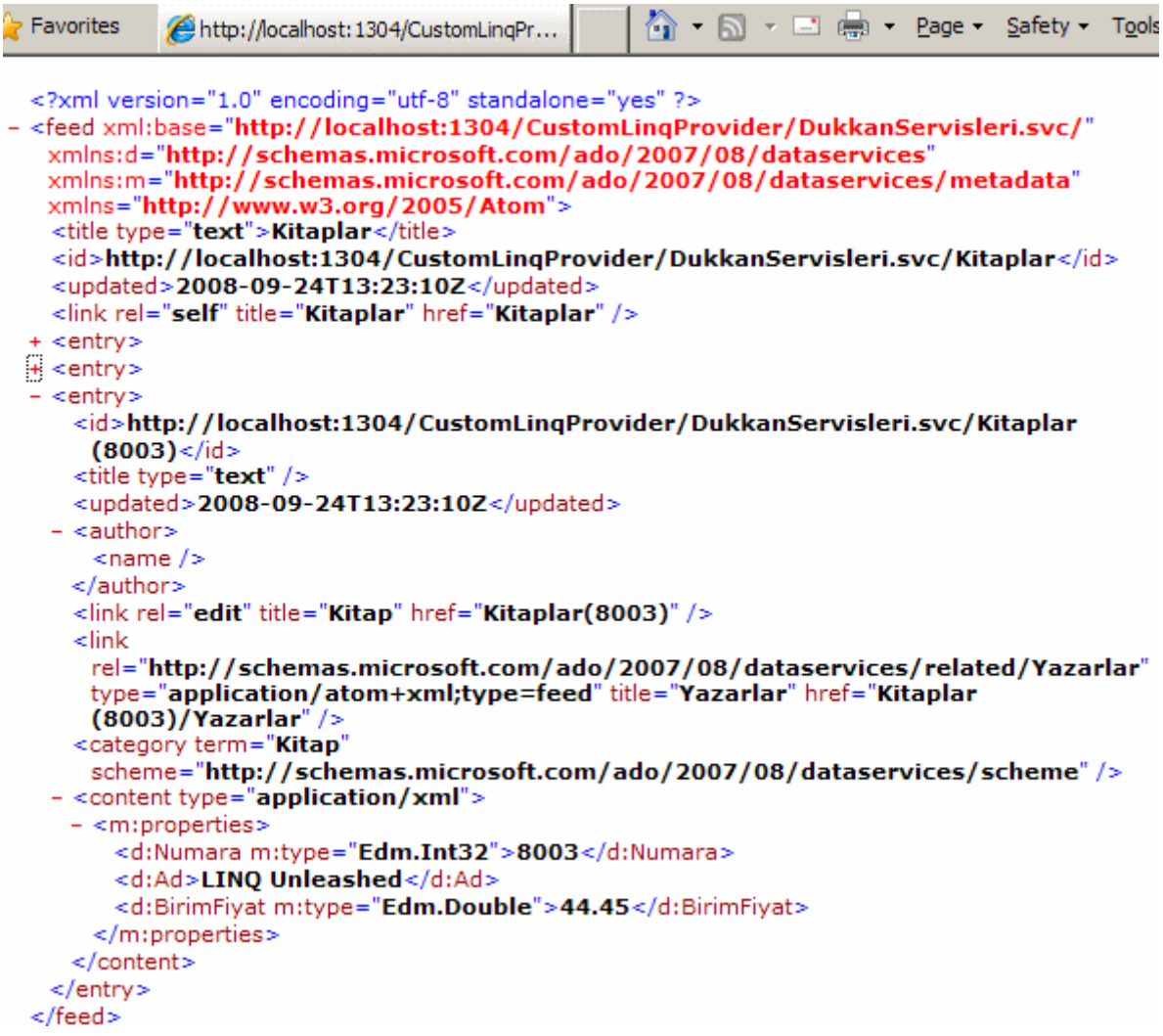



```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <feed xml:base="http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns="http://www.w3.org/2005/Atom">
    <title type="text">Kitaplar</title>
    <id>http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/Kategoriler
      (2)/Kitaplar</id>
    <updated>2008-09-24T13:16:56Z</updated>
    <link rel="self" title="Kitaplar" href="Kitaplar" />
  - <entry>
      <id>http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/Kitaplar
        (1240)</id>
      <title type="text" />
      <updated>2008-09-24T13:16:56Z</updated>
    - <author>
        <name />
      </author>
      <link rel="edit" title="Kitap" href="Kitaplar(1240)" />
      <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Yazarlar"
        type="application/atom+xml;type=feed" title="Yazarlar" href="Kitaplar
        (1240)/Yazarlar" />
      <category term="Kitap"
        scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    - <content type="application/xml">
      - <m:properties>
          <d:Numara m:type="Edm.Int32">1240</d:Numara>
          <d:Ad>Fermant' nın Son Teoremi</d:Ad>
          <d:BirimFiyat m:type="Edm.Double">5</d:BirimFiyat>
        </m:properties>
      </content>
    </entry>
  + <entry>
  + <entry>
  </feed>
```

örnek 3: Kitaplar içerisinden en pahalı 3 ünün elde edilmesi

URL : http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/**Kitaplar?$orderby=Bir**

Burada & kullanılarak birden fazla anahtar kelimenin birlikte kullanımı ele alınmaktadır.

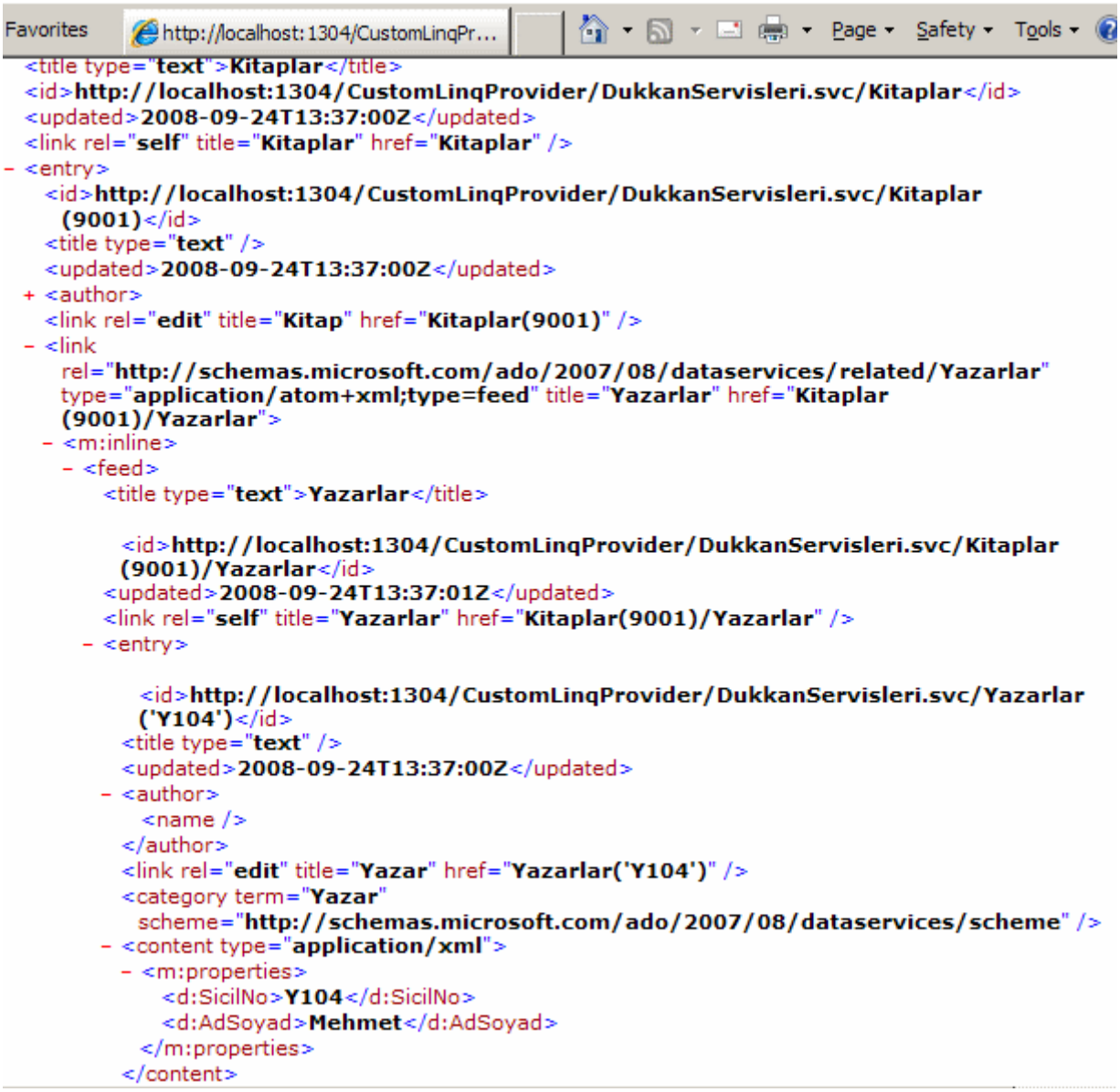



```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <feed xml:base="http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/"
    xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
    xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
    xmlns="http://www.w3.org/2005/Atom">
    <title type="text">Kitaplar</title>
    <id>http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/Kitaplar</id>
    <updated>2008-09-24T13:23:10Z</updated>
    <link rel="self" title="Kitaplar" href="Kitaplar" />
  + <entry>
  + <entry>
  - <entry>
      <id>http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/Kitaplar
        (8003)</id>
      <title type="text" />
      <updated>2008-09-24T13:23:10Z</updated>
    - <author>
        <name />
      </author>
      <link rel="edit" title="Kitap" href="Kitaplar(8003)" />
      <link
        rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/Yazarlar"
        type="application/atom+xml;type=feed" title="Yazarlar" href="Kitaplar
        (8003)/Yazarlar" />
      <category term="Kitap"
        scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
    - <content type="application/xml">
      - <m:properties>
          <d:Numara m:type="Edm.Int32">8003</d:Numara>
          <d:Ad>LINQ Unleashed</d:Ad>
          <d:BirimFiyat m:type="Edm.Double">44.45</d:BirimFiyat>
        </m:properties>
      </content>
    </entry>
  </feed>
```

örnek 4: Kitaplar ve her kitaba ait Yazar listelerinin elde edilmesi

URL : http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/**Kitaplar?$expand=Yaz**

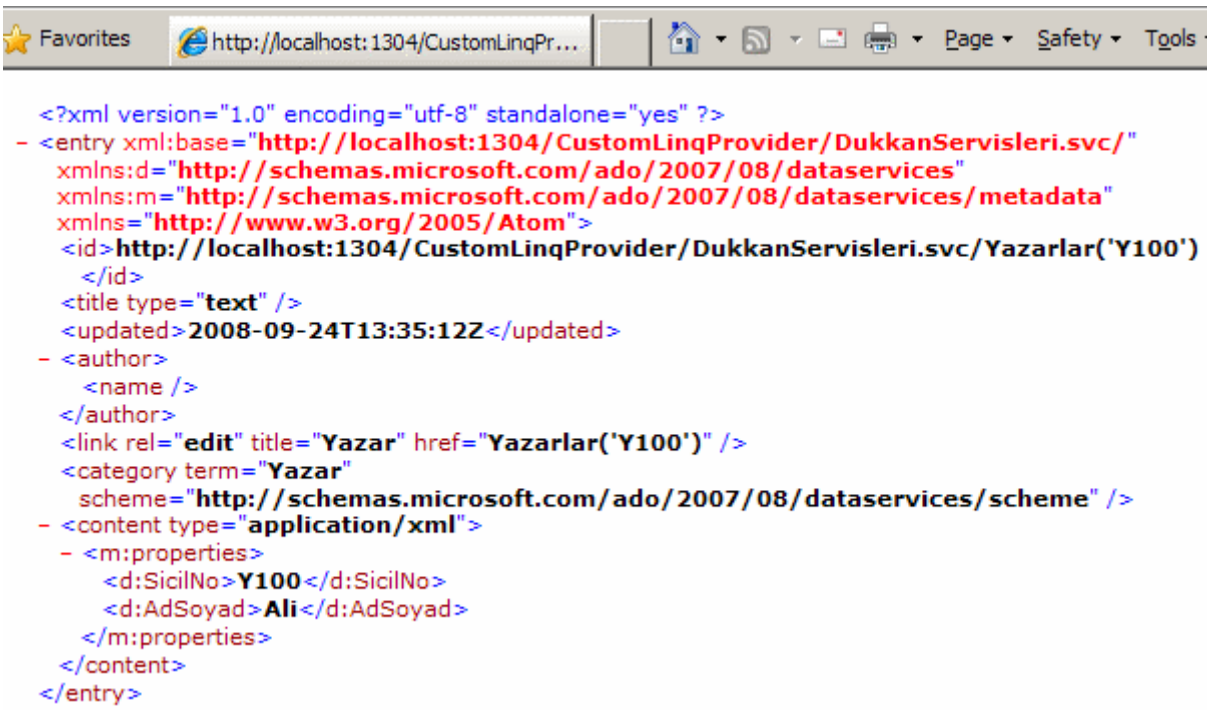
expand anahtar kelimesi sayesinde her Kitaba ait Yazar listelerinin çekilmesi ve XML içerisine g



örnek 5: SicilNo değeri Y100 olan yazarın elde edilmesi

URL : http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/**Yazarlar('Y100')**

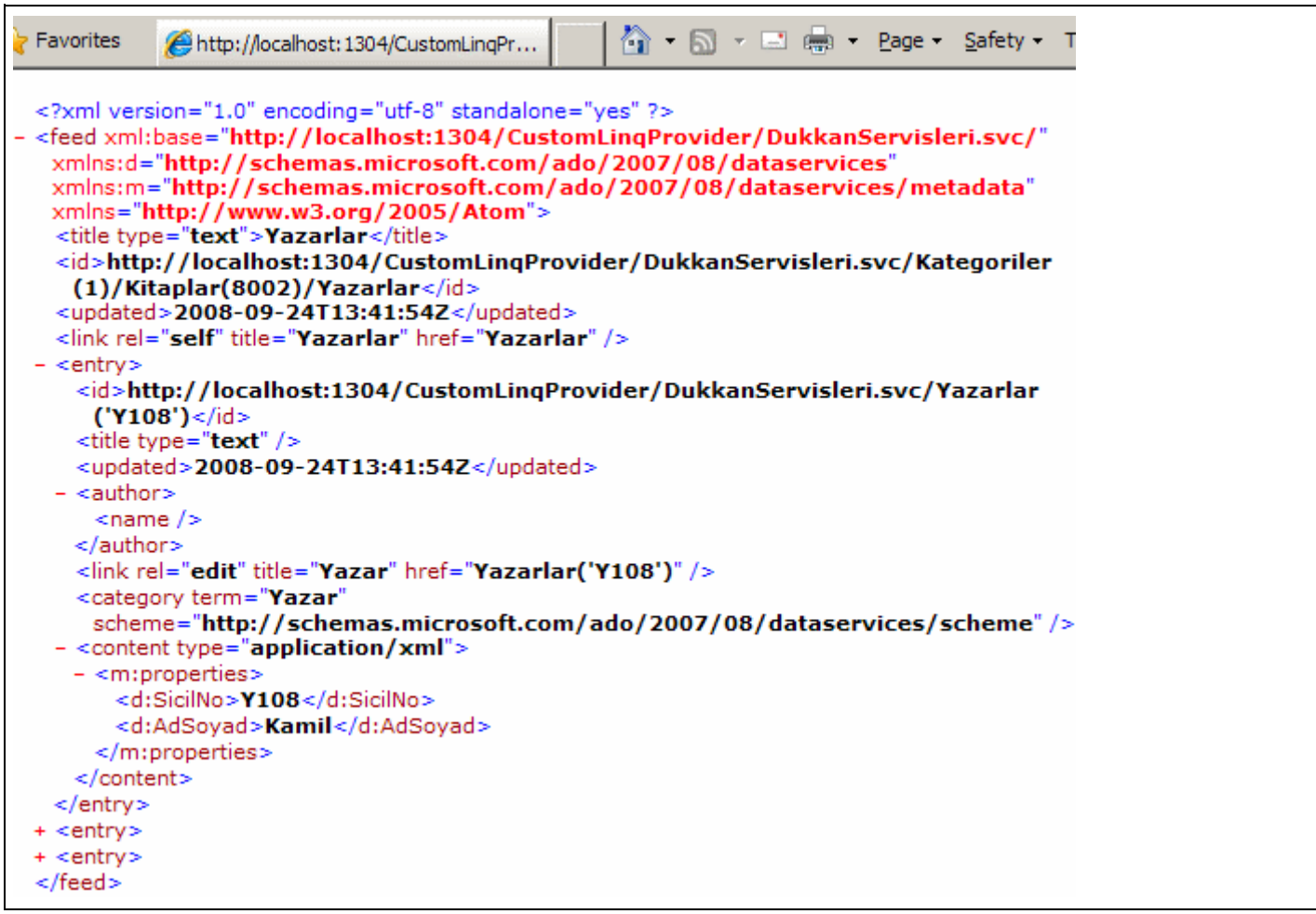
*Burada SicilNo **string** bir özellik olduğunda parantezler içerisinde **tek tırnak** işareti kullanılmıştı*



örnek 6 : 1 numaralı Kategorideki 8002 numaralı kitabın yazalarının elde edilmesi

URL : http://localhost:1304/CustomLinqProvider/DukkanServisleri.svc/**Kategoriler(1)/Kitaplar**

Bu örnektede özellikler arasında iki kademeli geçiş yapılmaktadır. Kategori' den Kitap' a, Kitap' öneririm.



Lütfen sizlerde değişik sorgulamaları deneyerek konuyu özümsemeye ve başka neler yapılabileceğini görmeye çalışınız.

Görüldüğü gibi **EDM**' den bağımsız olarak **Ado.Net Data Services** üzerinden geliştirme yapmak oldukça kolaydır. Yazımızın başlarındada belirttiğimiz gibi, farklı **LINQProvider** geliştirmeleri yapabilir ve bunları **Ado.Net Data Services** üzerinden **REST** modeline göre sunabiliriz. Biz makalemizde ele aldığımız örnekte bellek üzerinde tutulan(Memory Based) nesne koleksiyonlarını kullandık. Burada verinin kaynağı **XML** tabanlı dosyalarda olabilirdi. Size tavsiyem **XML** üzerinde tutulan verileri özel **LINQ Provider** yardımıyla bir **Ado.Net Data Service**üzerinden dışarıya sunmaya çalışmanızdır**.**

Bu makalede işlediklerimizi dilerseniz **görsel** olarakta izleyebilirsiniz. Her ne kadar ilk iki makale **Ado.Net Data Services** hakkında bir fikir vermiş olsada henüz bir istemci uygulama yazmadığımızı ve kolaya kaçarak tarayıcı uygulamaları kullandığımızı(Kullandığımı) farkedebilirsiniz :) İşte bir sonraki makalemizde bu konuya değinmeye çalışacağız. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

CustomLinqProvider.rar (5,38 kb)

Ado.Net Data Services Ders Notları - Hello World (2008-09-22T05:02:00)

ado.net data services,

Uzun bir aradan sonra yeni bir makale ile daha birlikteyiz. Sağnak yağışlı ve tamda "bu havada bir makale yazılır" dedirten bir günde hazırladığımız bu yazımızda, daha şimdiden gelecek vaat etmiş görünen, **.Net Framework 3.5 Service Pack 1** ile birlikte hazır olarak gelen, **Visual Studio 2008** ortamına entegre edilen ve **WCF** mimarisinin en güzel uyarlamalarından birisi olan**ADO.Net Data Services(Astoria)** üzerinde konuşuyor olacağız.

Bilindiği üzere **Windows Communication Foundation(WCF)**, **Microsoft**' un **.Net Framework 3.0** ile duyurduğu ve **3.5** ile getirdiği yeni ilavelerle ön plana çıkardığı yeni **Service YönelimliMimari(Service Oriented Architecture/SOA)** yaklaşımıdır. Bu yaklaşımın etkileri güncel projelerde kendini uzun zamandır göstermektedir. **SOA** yaklaşımlarının ağırlıklı bir biçimde kabul gördüğü günümüz çözümlerinde **veri(Data)** ile olan ilişkiler göz önüne alındığında **Microsoft** cephesinde uzun süre önce getirilen yeni bir proje karşımıza çıkmaktadır. **Ado.Net Data Services**.

Astoria kod adı ile anılan ve İstanbul Mecidiyeköy' deki alış veriş merkezinin adaşı olan bu mimarinin uygulanış biçimi **Visual Studio 2008 Service Pack 1** ile daha da kolay hale gelmiş ve**IDE** içerisine başarılı bir şekilde entegre edilmiştir. Peki Astoria neler vaat etmektedir ve nasıl bir mimari modele sahiptir? Dilerseniz kısaca bu konulara değinerek yazımızı sabırla okumaya devam edelim.

öncelikli olarak **Ado.Net Data Services** bir **WCF** servis yaklaşımıdır. Bununla birlikte, **WCF** mimarisine **.Net Framework 3.5** ile getirilen yeniliklerden biriside; servislerin, **Web Programlama Modeline**uygun olacak şekilde yayınlanabilmeleridir. Bu **Representational State Transfer(REST)** modeline uygun servislerin yazılabileceği anlamına gelmektedir.

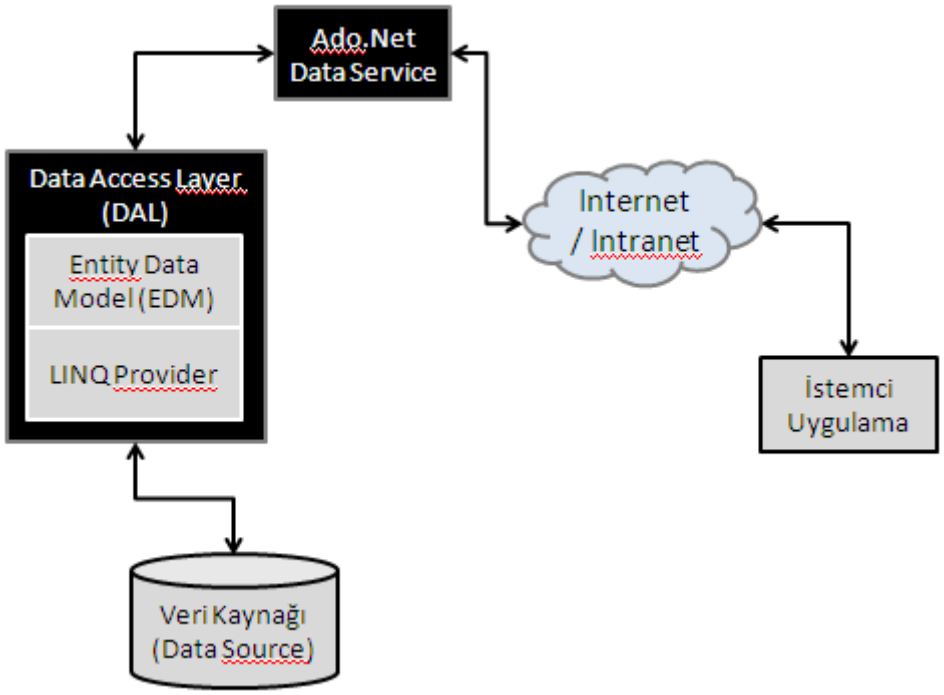
***NOT :** Ebellteki **Web programlama modelinin(Web Programming Model)** **WCF** tarafına kazandırdığı tek avantaj **QueryString** bazlı operasyon desteği değildir. Bunun yanında**JSON(Java Script Object Notation)** formatında yayınlama ve **RSS**, **Atom** bazlı **Syndication** desteğide gelmektedir.*

Aslında kafayı çok fazla karıştırmaya gerek yoktur. Aşağıdaki tablo durumu daha net bir şekilde özetlemektedir.

HTTP İşlemi	**CRUD Karşılığı**
Post	Create, Update, Delete
Get	Read

Put	Create, Overwrite/Replace
Delete	Delete

Bu tabloda anlatılmak istenen şudur; **HTTP** üzerinden yapılabilecek olan **Post, Get, Put, Delete** gibi çağrılar tablonun sağ tarafında yer alan veri operasyonlarına dönüştürülebilirler. O halde işin içersine veri kelimesinin girdiği ortadadır ve ne varki **Astoria** açılımı **Ado.Net Data Services** olarak geçmektedir. O halde Astoria için, Ado.Net tabanlı verileri **REST** modelinin belirttiği kriterlere uygun olacak şekilde dışarıya sunan servis mimarisidir tanımlamasını yapmak yerinde olacaktır. **Servis talepleri(Requests) HTTP** protokolüne göre **QueryString** bazlı olmaktadır. Bu talepler servis tarafına ulaştıklarında ise arka planda bir **Data Access Layer** tarafından karşılanmakta ve operasyonel olarak **CRUD(CreateReadUpdateDelete)** işlemlerine dönüştürülmektedir. Sonrasında istemciye gönderilecek olan cevaplar **XML** bazlı olarak ele alınmaktadır. Standart olarak **ATOM** formatında bir **XML** çıktısı istemci tarafına gönderilmektedir. Bu tanımlamalar kısaca bir fikir versede mimari detaylara bakmakta yarar vardır. Aşağıdaki şekil **Astoria** mimarisini kısaca özetlemektedir.



çok kısaca mimari üzerinden konuşarak devam edelim. öncelikli olarak **internet** veya **intranet** üzerinden talepte bulunabilecek bir **istemci(Client)** uygulama söz konusudur. Bu uygulama standart bir .**Net**programı olabilir. örneğin bir **Windows/WPF** yada basit bir **Console** uygulaması. çok doğal olarak istemci başka bir **servisde** olabilir. Ancak günümüzde **Ado.Net Data Service** örneklerini kullanacak en popüler istemciler web tabanlı olanlarıdır. Bir başka deyişle **Ajax Based Client** ve **Silverlight** gibi uç birimler örnek olarak verilebilirler.

İlerleyen kısımlarda **Ajax** tabanlı bir istemcinin nasıl geliştirileceğine de değinilecektir. İstemci uygulamalar, servise doğru QueryString benzeri formatta bir talepte bulunabilir. örneğin;

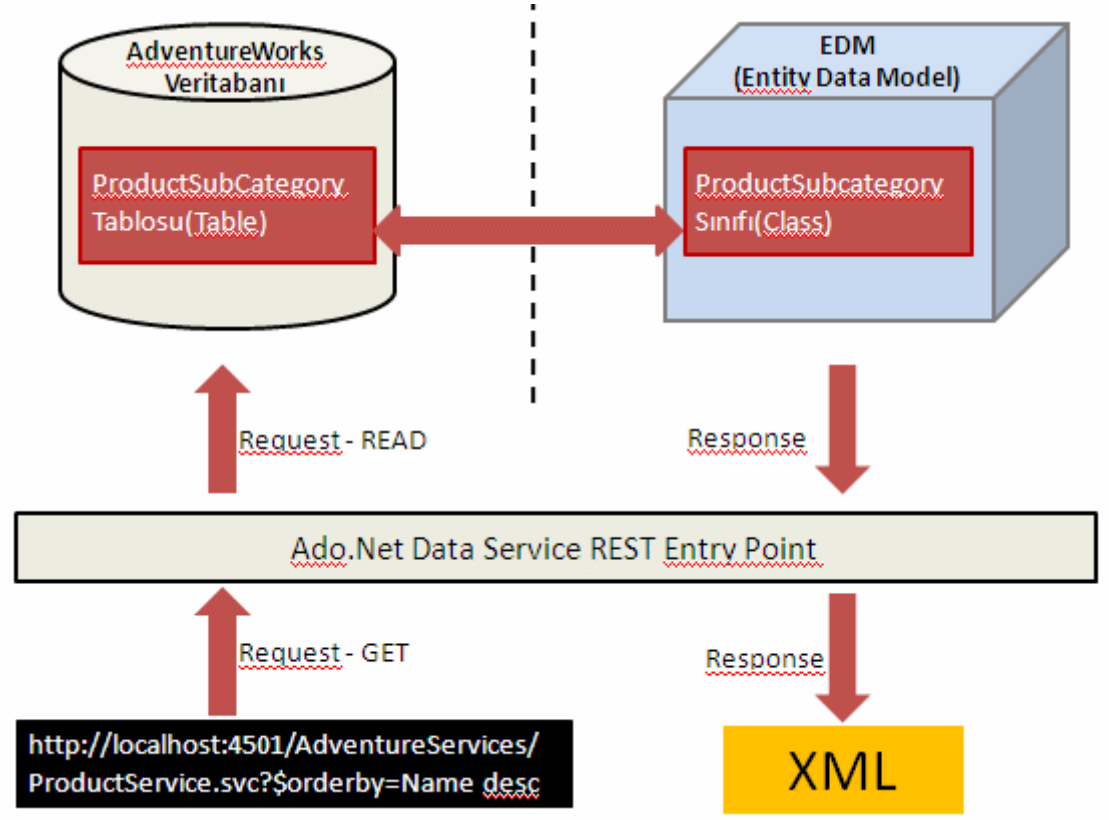
http://localhost:4501/AdventureServices/ProductService.svc/**ProductSubcategory?$order by=Name desc**

gibi.

Aslında bu ifade son derece açıktır. Tahmin edileceği üzere **HTTP** tabanlı olaraktan **localhost** isimli yerel makinede **4501** numaralı port üzerinden yayın yapan **ProductService.svc** isimli bir **WCF**servisi söz konusudur. Servise giden talep ise şunu ifade etmektedir; **"Lütfen ProductSubcategory verilerini Name alanlarına göre ters sırada olacak şekilde gönderin"**. İşte **svc** uzantısından sonra gelen ifade basit bir **QueryString** tanımlamasıdır. Elbetteki arada kullanılan **$** işareti ifadeyi **Ado.Net Data Services** için biraz özelleştirmektedir. Bu noktada talebin servis tarafına ulaştığını düşünebiliriz. Peki servis bu noktadan sonra ne yapmaktadır?

Service tarafında **çalışma zamanında(RunTime)** devrede olan **DataServiceHost** *(**WebServiceHost** sınıfından türeyen ki buda **WCF** Servislerinde çekirdek olan **ServiceHost** sınıfından türemektedir.)* nesne örneği, gelen talebi arka planda ele alır. Bu noktada elde iki seçenek yer almaktadır. Bunlardan birisi yine **Visual Studio 2008 Service Pack 1** ile **IDE** ortamına dahil olan**Entity Data Model(EDM)** açılımının kullanılmasıdır. Diğeri ise özel **LINQ Provider** kullanıldığı seçenektir. **LINQ Provider** kullanımı yardımıyla **REST** taleplerinin nesneler üzerindede ele alınması sağlanabilmektedir ki bununla ilişkili bir örneği ilerleyen bölümlerde geliştiriyor olacağız.

EDM modeline göre veritabanı bazlı **Entity tipler(Types)** ve bu tiplere ait **üyeler(Members)** söz konusudur. Burada temel amaç veritabanı üzerindeki nesnel yapıların **OOP(Object Oriented Programming)** ortamında karşılıkları olan **sınıf(Class)** tiplerinde ve üyelerinde*(Her alanın karşılığı olan bir özellik ile)* ele alabilmektir. Böylece kod ortamından veritabanı üzerine geçiş yapmaya gerek kalmadan **CRUD** operasyonları kolayca icra edilebilir. **EDM** açısından olaya baktığımızda sadece yukarıdaki basit **okuma(READ)** talebi için aşağıdaki şekil biraz daha aydınlatıcı ve fikir verici olabilir.

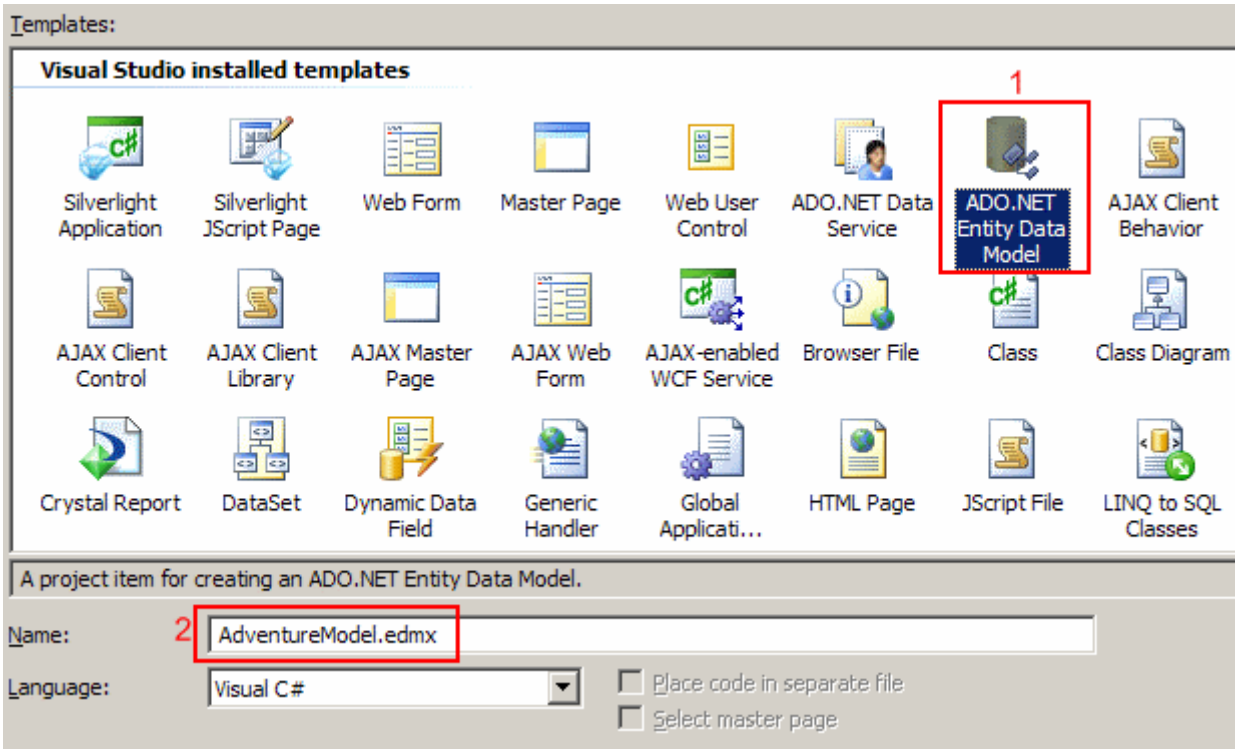


Görüldüğü üzere **REST** bazlı talep, **Ado.Net Data Service Entry Point**' e ulaştıktan sonra servis çalışma zamanı tarafından veritabanına doğru basit **sorgular(Queries)**şeklinde gönderilirler. Bunun doğal sonucu olarak bazı veri kümeleri elde edilir. Elde edilen sorgu sonuçları **EDM** içerisinde yer alan **Entity** nesneleri ve üyeleri tarafından değerlendirilir. Nitekim veritabanı tarafındaki nesnelerin karşılığı olan varlıklar, **EDM** içerisinde yer almakta olup çalışma zamanında servis operasyonları tarafından ele alınmaktadır. Bir başka deyişle örnek baz alındığında, **ProductSubCategory** tablosu içerisindeki herhangibir **satırın(Table Row)** karşılığı olan **ProductSubcategory** sınıfına ait nesne örneklerinden oluşan bir **koleksiyon(Collection)** üretimi gerçekleşir. Bu üretim sonrasında ilgili koleksiyon, servis çalışma ortamı tarafından **XML** çıktısı haline getirilir ve istemciye gönderilir.

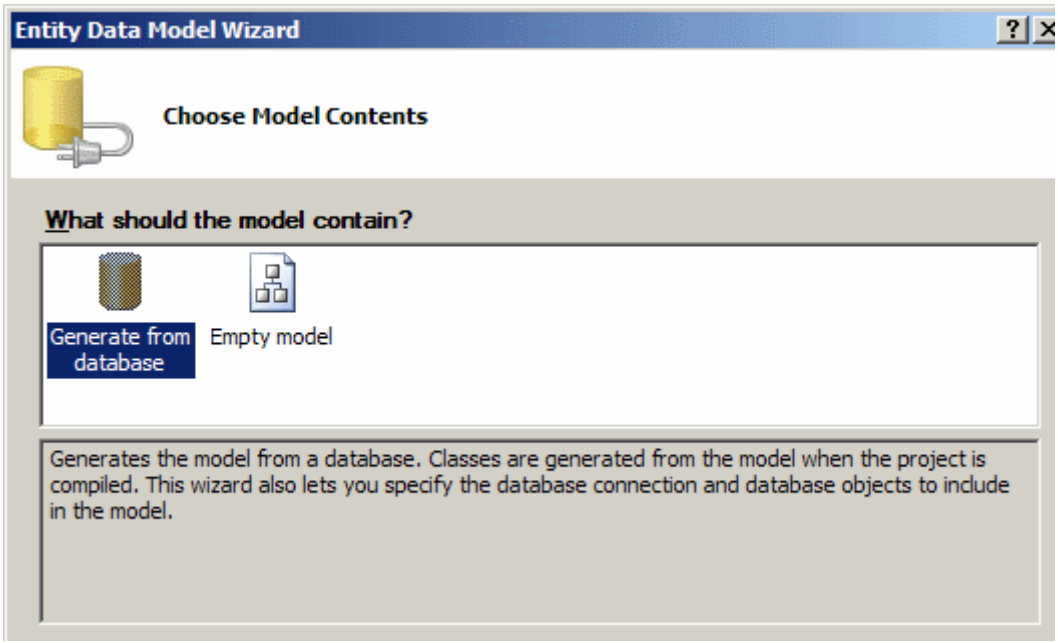
Sanıyorumki artık bir örnek geliştirerek konuyu pekiştirmenin ve makalenin yazıldığı bu yağmurlu günde ekranımızda bir güneş açtırmanın zamanı geldi. İlk olarak örneğin **Visual Studio 2008 Professional Service Pack 1** üzerinde geliştirildiğini belirtelim.

Ado.Net Data Service' ler esas itibariyle birer **WCF** servis öğesi olarak tanımlanırlar. Bu sebepten dolayı söz konusu servislerin bir sunucu uygulama üzerinde **host** edilmeleri gerekmektedir. Burada istenirse bir **WCF Service** uygulaması baz alınabilir. Yada herhangibir **Asp.Net Web Site/Asp.Net Web Application** üzerindede bu işlem gerçekleştirilebilir. **Web** tarafındaki geliştirme kurallarının buradada geçerli olduğunu ve buna göre **dosya tabanlı(File-Based)** yada doğrudan **IIS** üzerinde geliştirme yapabileceğimizi hatırlayalım. Biz
örneğimizde **AdventureServices** adlı **WCFService** şablonunu

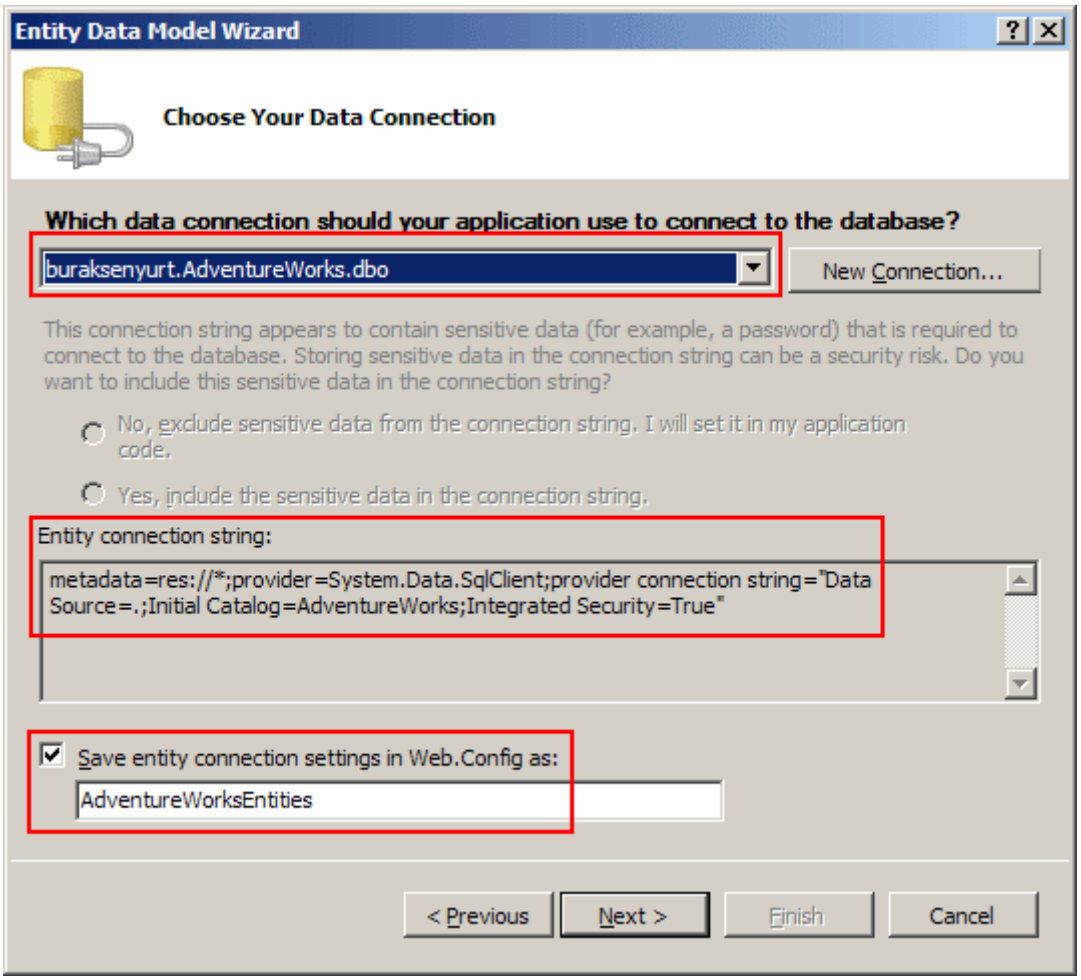
kullanacağız. **AdventureServices** isimli proje **File-Based** olarak geliştirilecektir. Servis uygulaması oluşturulduktan sonra ilk yapılması gereken, **Ado.Net Data Service**' in kullanacağı **Data Access Layer** ortamını hazırlamaktır. Burada daha öncedende belirtildiği üzere **EDM**, **LINQ Provider** seçenekleri mevcuttur.
örneğimizde **Entity Data Model** kullanılmaktadır. **EDM** nesnesini eklemek için projeye sağ tıkladıktan sonra aşağıdaki resimde yer alan **Ado.Net Entity Data Model** şablonunu seçmek yeterli olacaktır.



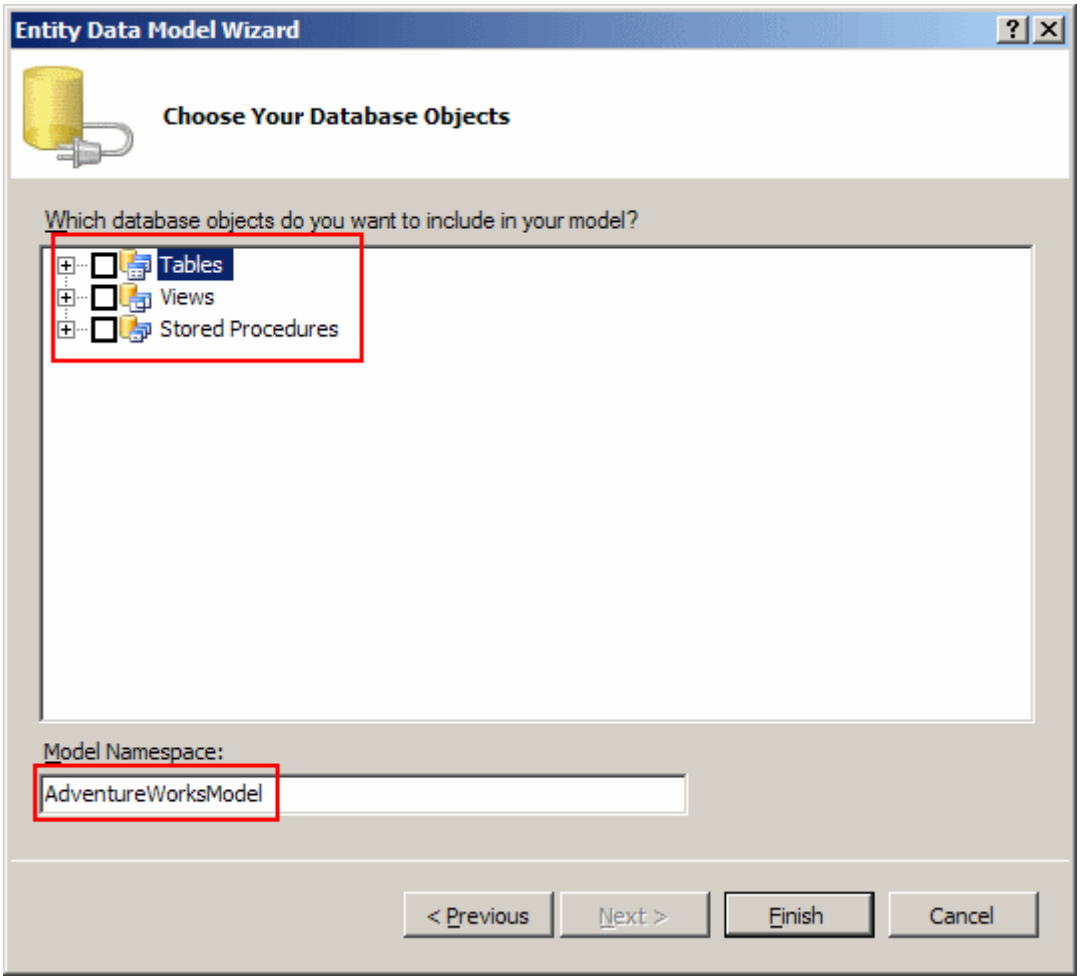
İsim olarak **AdventureModel** adı kullanılabilir. **edmx** uzantılı dosya seçimi yapıldıktan sonra bir dizi adımdan oluşan sihirbaz arabirimi ile karşılaşılır.



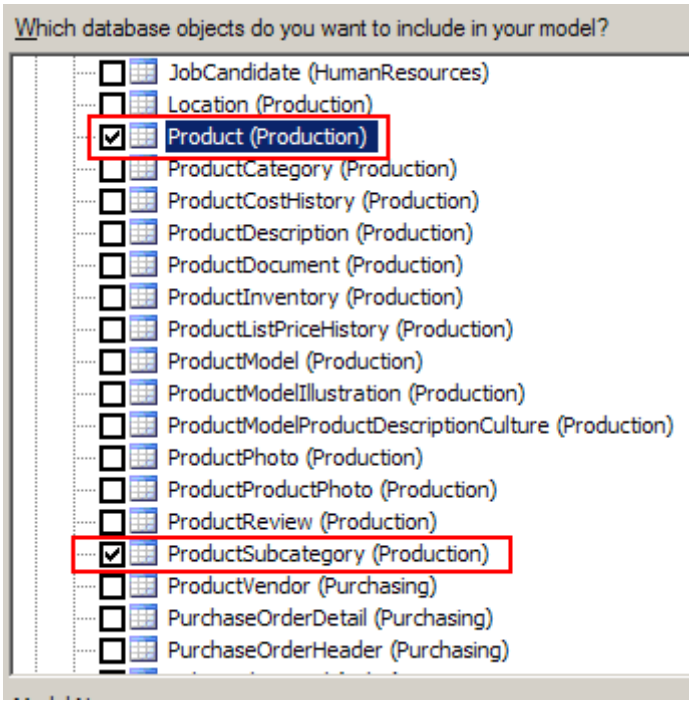
İlk adımda **EDM** modelinin var olan bir veritabanından oluşturulacağı seçimi yapılır**(Generate from database)**. Ancak istenirse **Empty Model** kullanılarak, **EDM** tiplerinin görsel olarak veritabanından bağımsız tasarlanması ve ilgili sınıfları ve üyelerinin oluşturulması sağlanabilir. Bu çoğunlukla **Entity** tasarımının önceden yapılıp sonrasında veriye bağlama kararının verileceği durumlarda ele alınabilir. Ki bu şekilde oluşturulan **Entity** nesneleri içerisinde **LINQ Provider**' lar kullanılarak farklı sağlayıcılara*(örneğin **XML** veya**Object** tiplerine)* doğru eşleştirmelerde gerçekleştirilebilir.



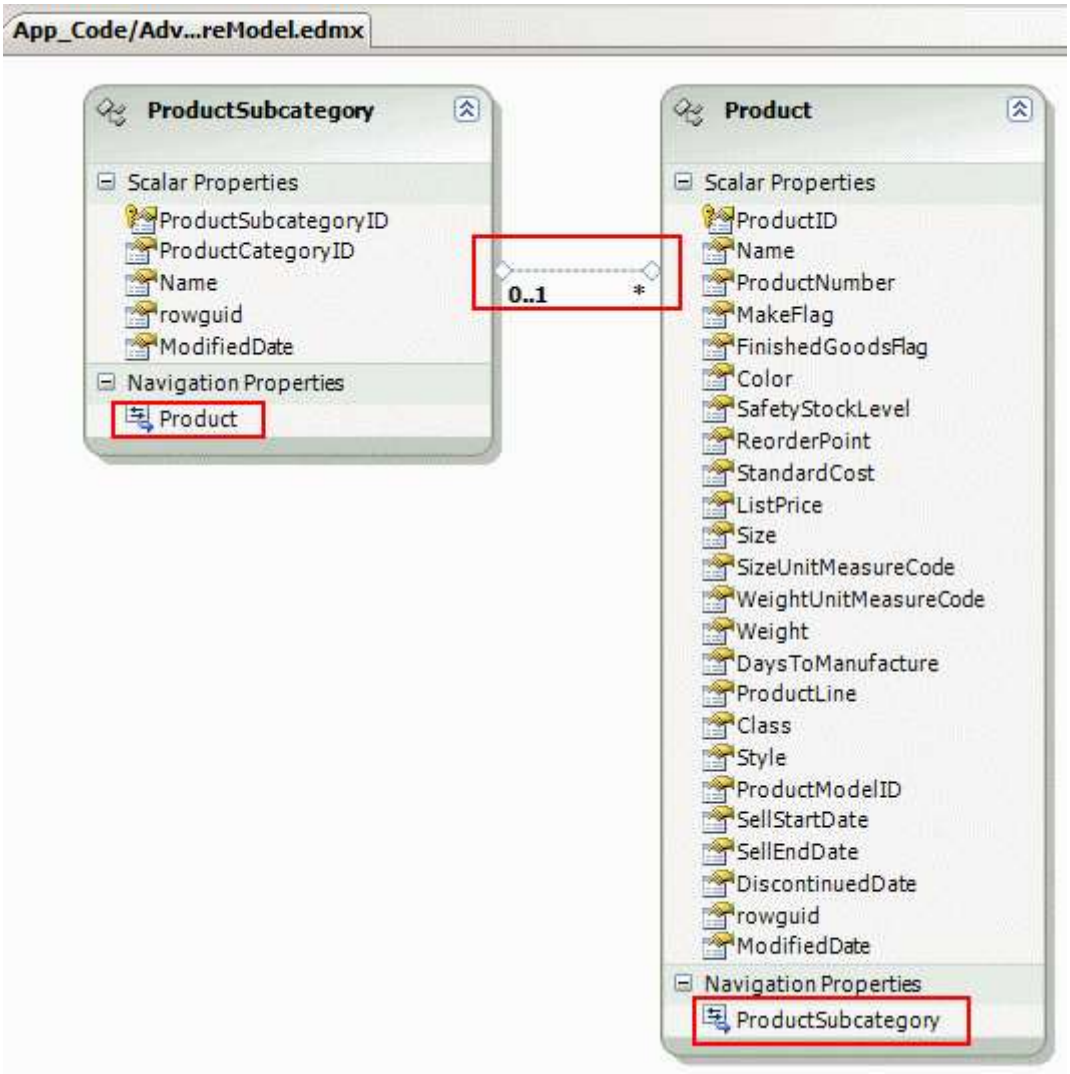
İkinci adımda **bağlantı(Connection)** seçimi yapılır. Buna göre herhangibir veritabanı bağlantısı kullanılabilir. Söz konusu bağlantılara ilişkin bilgiler ise istenirse **Web.config** dosyası içerisinde saklanabilir.



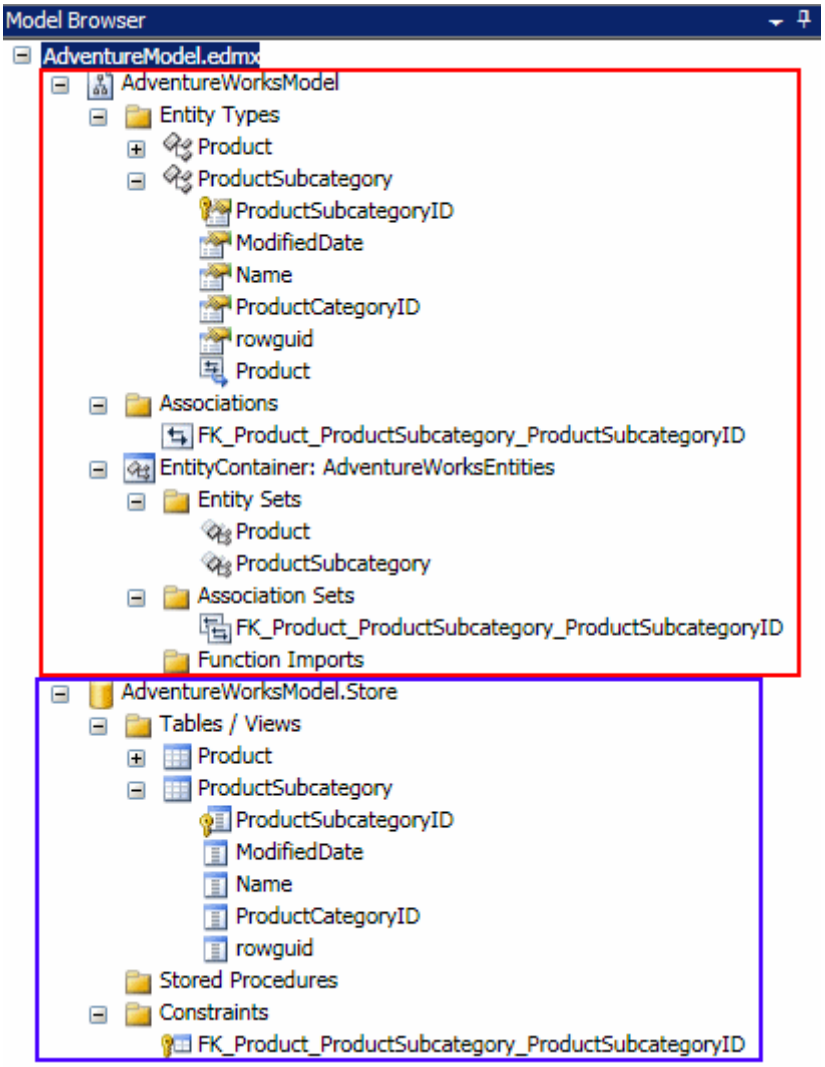
üçüncü adımda, oluşturulan bağlantı üzerindeki veritabanı içeriği görülür. Burada **tablolar(Tables)**, **görünümler(Views)** ve **saklı yordamlar(Stored Procedures)** yer almaktadır. Dolayısıyla bu adımda, **EDM** içerisindeki tiplerin eş düştüğü veritabanı objeleri işaretlenir.



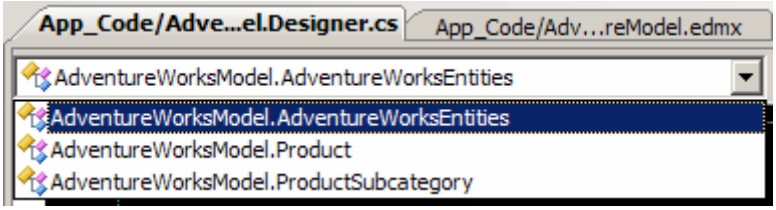
örnekte yukarıdaki şekildende görüleceği üzere **ProductSubCategory** ve **Product** tabloları ele alınmaktadır. Bu tablolar arasında **bire-çok(One to many)** ilişki olması nedeni ile ilişkisel yapılarıda inceleyebilme fırsatımız olacaktır. Tüm bu işlemler tamamlandıktan sonra aşağıdaki **EDM** diagramının oluştuğu görülecektir.



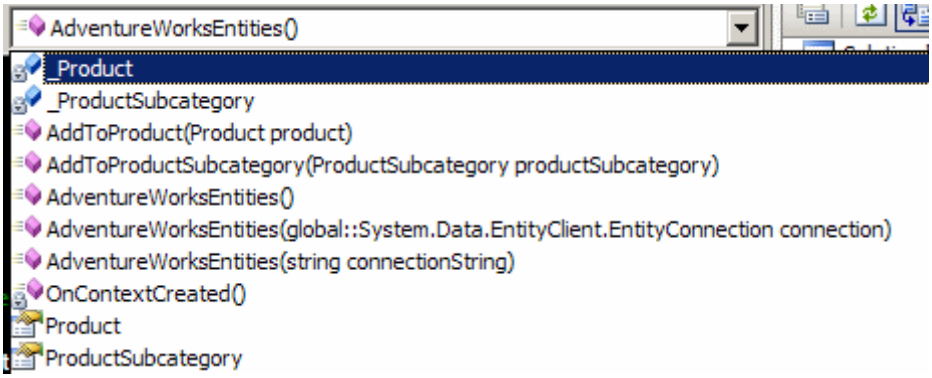
Dikkat edilecek olursa **ProductSubCategory** ve **Product** tablolarının kendileri birer sınıf olarak oluşturulmuş, alanları birer üye olarak ilave edilmiştir. Bunlara ek olarak her iki tablo arasındaki**ilişki(Relation)**, **EDM** içerisinde bir **Association** olarak tanımlanmıştır. Dikkat çekici özelliklerden bir diğeri ise sınıflara ait nesne örnekleri üzerinden birbirlerine geçiş yapılmasını sağlayacak **özelliklerin(Properties)** eklenmiş olmasıdır. Söz gelimi bir alt kategoriye bağlı ürünleri elde etmek için **ProductSubcategory** tipine ait nesne örneği üzerinden **Product**özelliği kullanılabilir. Oluşturulan bu sınıflar ve eş düştükleri veritabanı objeleri arasındaki ilişkiler istenirse **Model Browser** aracılığıyla aşağıdaki şekilde görüldüğü gibide izlenebilir.



örnekteki **Model Browser** görselinde, **AdventureWorksModel** kısmında **EDM** içeriği haritalanmaktadır. Diğer taraftan **AdventureWorksModel.Store** boğumu(Node) altında ise, **EDM**içeriğinin karşılıkları olan veritabanı unsurları listelenmektedir. **AdventureModel.Designer.cs** dosyasına bakıldığında ise 3 adet **sınıfın(Class)** oluşturulduğu görülür.



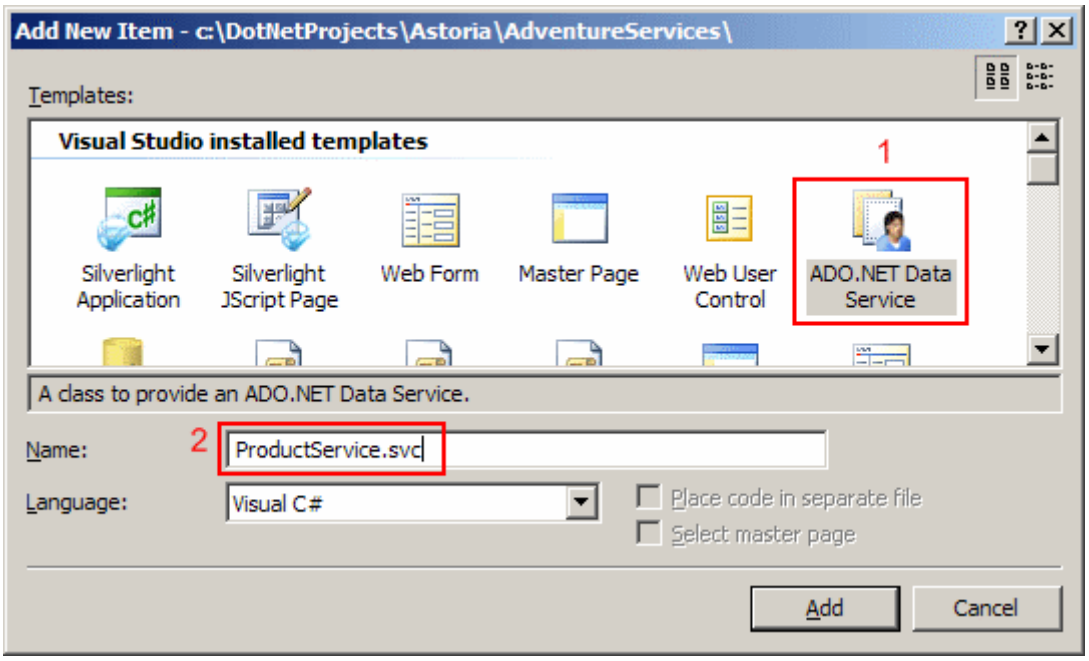
Dikkat edileceği üzere **Product** ve **ProductSubcategory** tipleri dışında **AdventureWorksEntities** isimli bir tipin daha olduğu görülmektedir ki üyeleri aşağıdaki şekilde olduğu gibidir.



AdventureWorksEntites, **ObjectContext** sınıfından türemiştir
ve **Entity** koleksiyonlarının yönetimi, **EDM** nesnelerinin eş düştüğü veri objeleri ile olan fonksiyonelliklerin ele alınması gibi kritik görevleride üstlenmek üzere tasarlanmıştır. öyleki, **Product** ve **ProductSubcategory** nesne toplulukları bu sınıf içerisinde aşağıdaki kod parçasında olduğu gibi tutulmaktadır.

```
public global::System.Data.Objects.ObjectQuery<Product> Product
{
   get
   {
      if ((this._Product == null))
      {
         this._Product = base.CreateQuery<Product>("[Product]");
      }
      return this._Product;
   }
}
private global::System.Data.Objects.ObjectQuery<Product> _Product;

public global::System.Data.Objects.ObjectQuery<ProductSubcategory>
ProductSubcategory
{
   get
   {
      if ((this._ProductSubcategory == null))
      {
         this._ProductSubcategory =
base.CreateQuery<ProductSubcategory>("[ProductSubcategory]");
      }
      return this._ProductSubcategory;
   }
}
private global::System.Data.Objects.ObjectQuery<ProductSubcategory>
_ProductSubcategory;
```

ObjectQuery<T> tipinden birer **readonly özellik(Property)** yardımıyla! Ve yine bir ürünün veya alt kategorinin eklenmesi için **AddToProduct** ve **AddToProductSubcategory** isimli metodlarda yer almaktadır. Bu noktada akla şöyle bir soru gelebilir. **Update**, **Delete** işlemleri için niye metodlar bulunmamaktadır? Söz konusu sorunun cevabı ilerleyen bölümlerde verilecektir ve bu nedenle sizlere biraz düşünme ve araştırma süresi kalmaktadır. Bu detayları şimdilik geride bırakarak asıl konumuza geri dönmenin yararlı olacağı kanısındayım.
Artık **WCF Service**uygulamasına bir **Ado.Net Data Service** eklenebilir. Tek yapılması gereken projeye **Add New Item** seçeneği ile aşağıdaki şekildede görülen **Ado.Net Data Service** öğesini eklemektir.

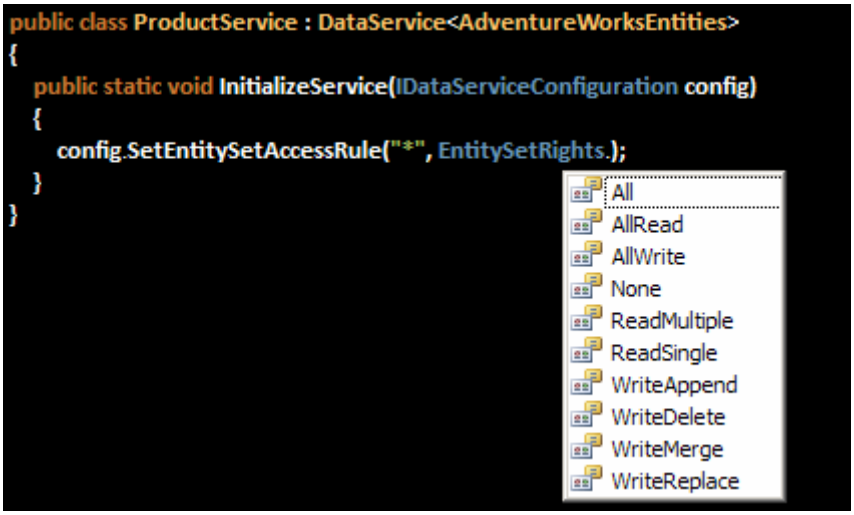


Bu işlemin ardından proje şablounan **ProductService.svc** servis ve **ProductService.cs code-behind** dosyaları eklenecektir. *(Söz konusu işlemlerde biz **WCF** geliştiricilerini şaşırtan herhangibir nokta bulunmamaktadır. Nitekim **Web** üzerinden **host** edilen bir **WCF** serviside aynı prensiplerde oluşturulmaktadır. Bir **svc** içeriği ve çoğunlukla **code-behind** üzerinde tutulan kod içeriği.)* **ProductService.cs** içeriği kısaca incelendiğinde bir **başlatma işleminin(Initialization)** yapılması gerektiği görülmektedir. Nitekim servis nesnesi örneklendiğinde **EDM**içerisindeki hangi tiplerin hangi şartlarda yayınlanacağının belirlenmesi gerekmektedir. Bu bir anlamda yetkilendirme süreci olarakta düşünülebilir. Söz konusu **ProductService.cs** içeriği örnek için aşağıdaki gibi değiştirilmelidir.

```
using System;
using System.Data.Services;
using System.Collections.Generic;
using System.Linq;
using System.ServiceModel.Web;
using AdventureWorksModel;
```

```
public class ProductService
   : DataService<AdventureWorksEntities>
{
   public static void InitializeService(IDataServiceConfiguration config)
   {
      config.SetEntitySetAccessRule("*", EntitySetRights.AllRead);
   }
}
```

Burada dikkat edilmesi gereken noktalardan birisi **ProductService** sınıfının **System.Data.Services** isim alanında**(Namesapce)** yer alan **DataService<T> generic** sınıfından türemiş olmasıdır. Bunun dışından şu an için önem arz eden nokta **static InitializeService** metodudur. Bu metod, servis örneği ilk oluşturulduğunda bir kereliğine devreye girer. **config** değişkeni üzerinden yapılan çağrı ise önemlidir. **SetEntitySetAccessRule** metoduna gönderilen ilk parametrede ***** sembölü kullanılarak, **Entity** set içerisindeki tüm tiplerin ele alınacağı belirtilmektedir. İkinci parametre ise **EntitySetRights** enum sabiti tipinden olup aşağıdaki değerleri alabilir.

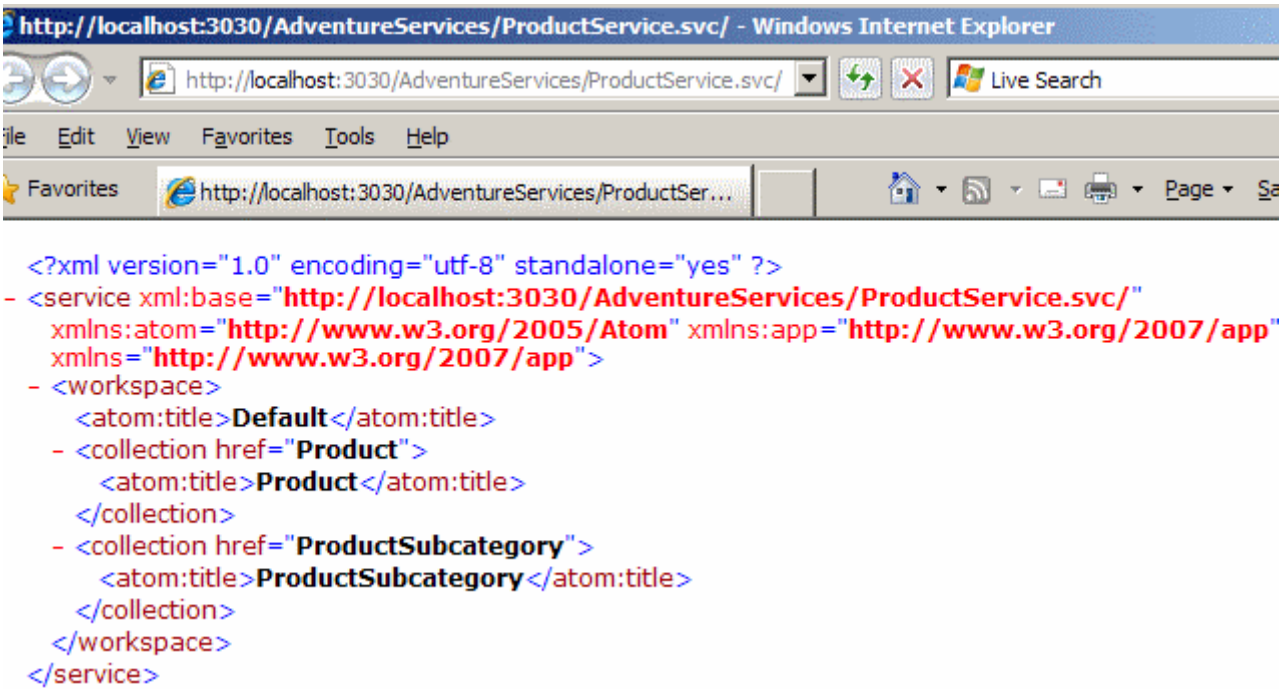


Burada **AllRead**' in anlamı tüm **Entity** nesneleri üzerinde her çeşit **veri okuma(Read)** işleminin yapılabileceğidir. Bir başka deyişle **EntitySetRights enum** sabitinin değerleri ile, hangi **Entity**objelerine hangi haklarla erişebileceği belirtilmektedir. Söz gelimi aşağıdaki kod örneğini ele alalım.

```
config.SetEntitySetAccessRule("Product", EntitySetRights.AllRead);
config.SetEntitySetAccessRule("ProductSubcategory", EntitySetRights.AllWrite);
```

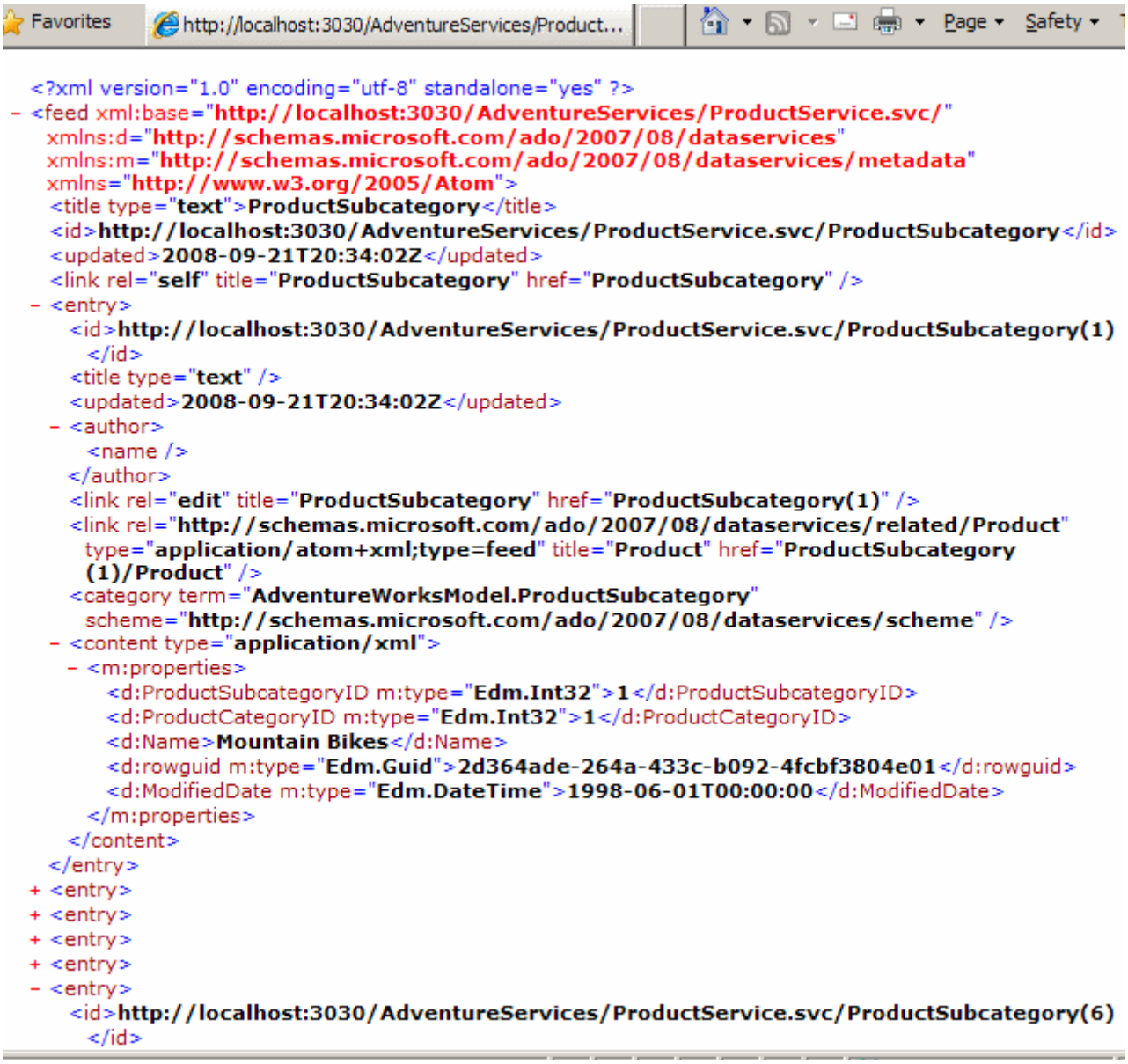
Bu ifadelere göre **Product** nesneleri için sadece okuma işlemi yapılabilirken, **ProductSubcategory** nesneleri içinde sadece yazma işlemleri yapılabilmektedir. Artık herhangibir istemci yazmadan,**ProductService.svc** servisi test edilebilir. Nitekim hepimizin iştahının kabarmış olduğunu ve bir an önce sonuçları görmek istediğinizi hissetmekteyim. öyleyse gelin **F5** ile projemizi çalıştıralım. Uygulama ilk

çalıştırıldığında **ProductService.svc** dosyası tarayıcı pencere içerisinde aşağıdaki gibi görünecektir.



Buradan çıkartılması gereken ilk sonuç **Product** ve **ProductSubcategory** elementleri için taleplerde bulunulabileceğidir. öyleyse test sorgularına başlanabilir. Sorgulardan kastımız elbetteki **URL**satırına girilen **QueryString** ifadeleri ve
bunların **ATOM** tabanlı **XML** çıktılarının nasıl olacağıdır.

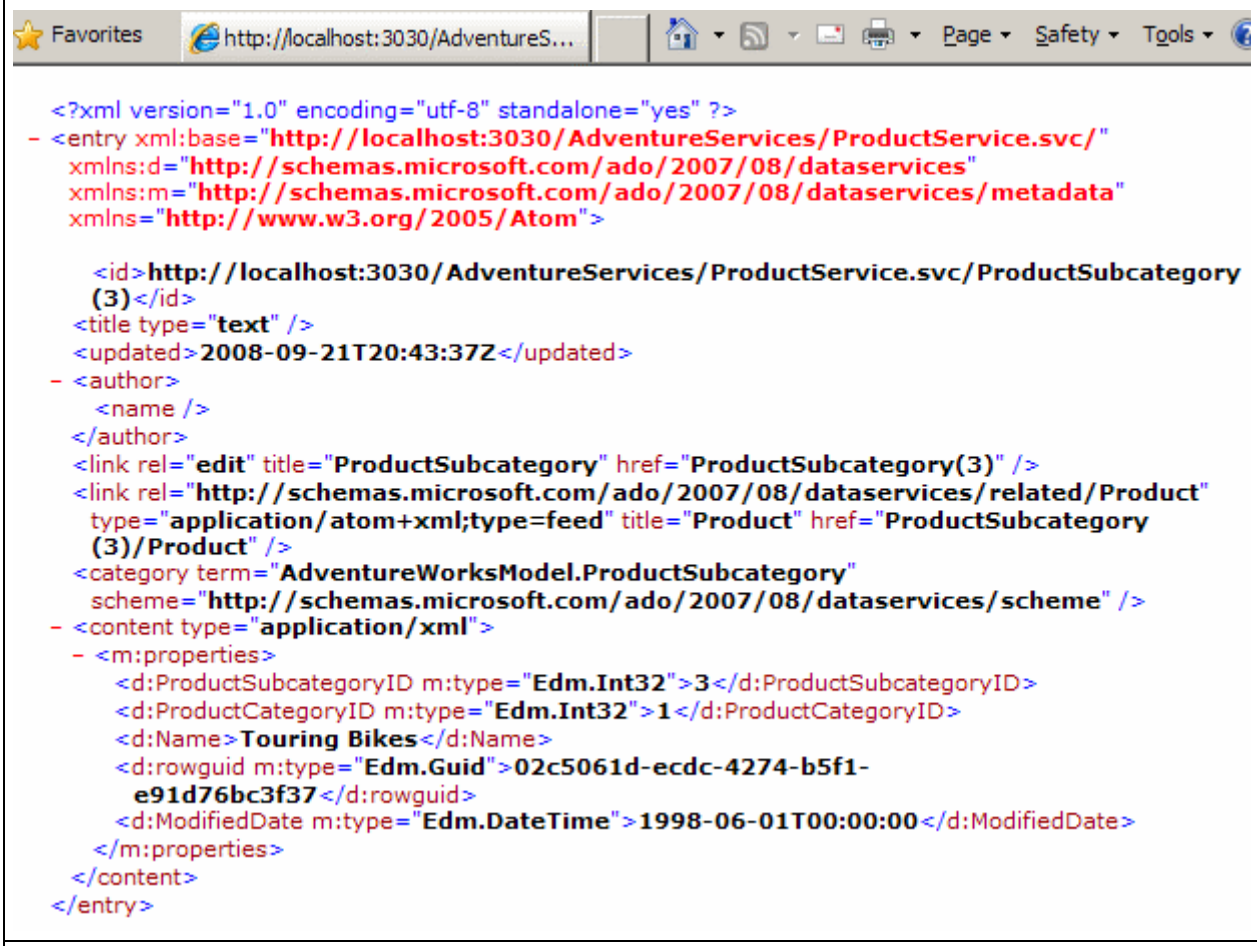
örnek 1 ; **Tüm ProductSubcategory satırlarının elde edilmesi**
URL Satırı ifadesi : **http://localhost:3030/AdventureServices/ProductService.svc/ProductSubcategory** *(Burada hemen bir hatırlatma yapalım. Eğer **URL** satırında **ProductSubcategory** yerine **Produc*** ***Found** çıktısı alınır. Dolayısıyla **QueryString**' leri yazarken **Case-Sensitive** olmalarına dikkat et*
Sonuç Ekran görüntüsü ;



örnek 2 : 3 Numaralı ProductSubcategory bilgisinin elde edilmesi

URL Satırı ifadesi :
http://localhost:3030/AdventureServices/ProductService.svc/ProductSubcategory(3)

Sonuç Ekran görüntüsü ;



Not : İkinci örnekte dikkat edilmesi gereken bir nokta vardır. URL satırında parantez içerisinde 3 edilmiştir. Pekiya servis tarafından nasıl olmaktadırda, parantez içerisindeki değerin ProductSubC nokta **ProductSubcategory** sınıfındaki ProductSubCategoryID özelliğidir.

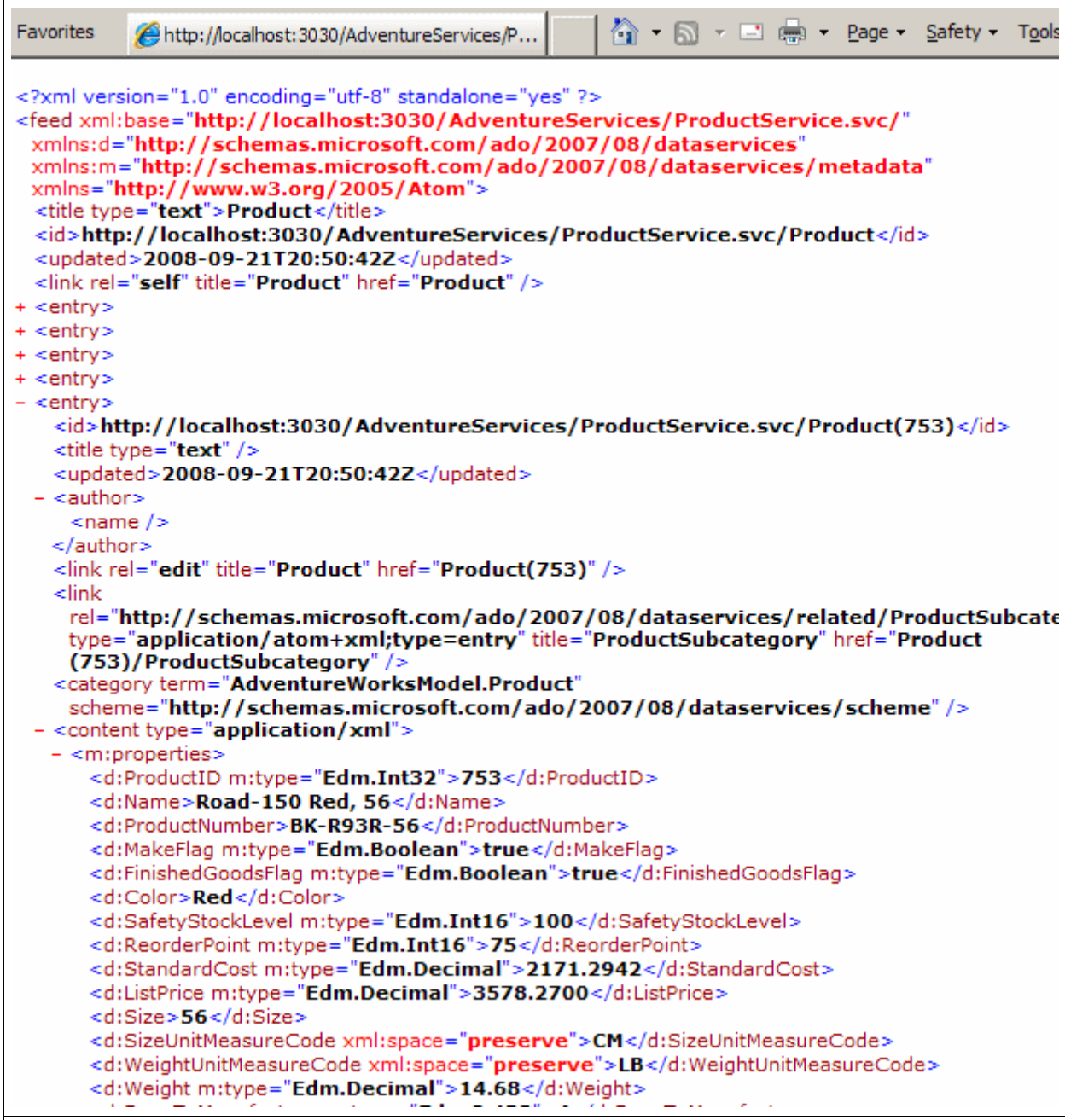
```
[global::System.Data.Objects.DataClasses.EdmScalarPropertyAttribute(EntityKeyProperty=true, IsNullable=false)]
[global::System.Runtime.Serialization.DataMemberAttribute()]
public int ProductSubcategoryID
{
    get
    {
        return this._ProductSubcategoryID;
    }
    set
    {
        this.OnProductSubcategoryIDChanging(value);
        this.ReportPropertyChanging("ProductSubcategoryID");
        this._ProductSubcategoryID = global::System.Data.Objects.DataClasses.StructuralObject.SetValidValue(value);
        this.ReportPropertyChanged("ProductSubcategoryID");
        this.OnProductSubcategoryIDChanged();
    }
}
private int _ProductSubcategoryID;
```

Şekildende görüleceği üzere **EdmScalarPropertyAttribute** niteliğinin içerisinde **EntityKeyPro** gelen ifadenin bu özelliğe ait olduğunu bilmektedir.

örnek 3 : Product tablosundan ListPrice değeri 3500 birim üzerinde olanların elde edilmesi

URL Satırı ifadesi :
http://localhost:3030/AdventureServices/ProductService.svc/Product?$filter=ListPrice gt 35
(Burada gt=grater then anlamındadır.Buna göre küçüktür için lt=less then)

Sonuç Ekran görüntüsü ;

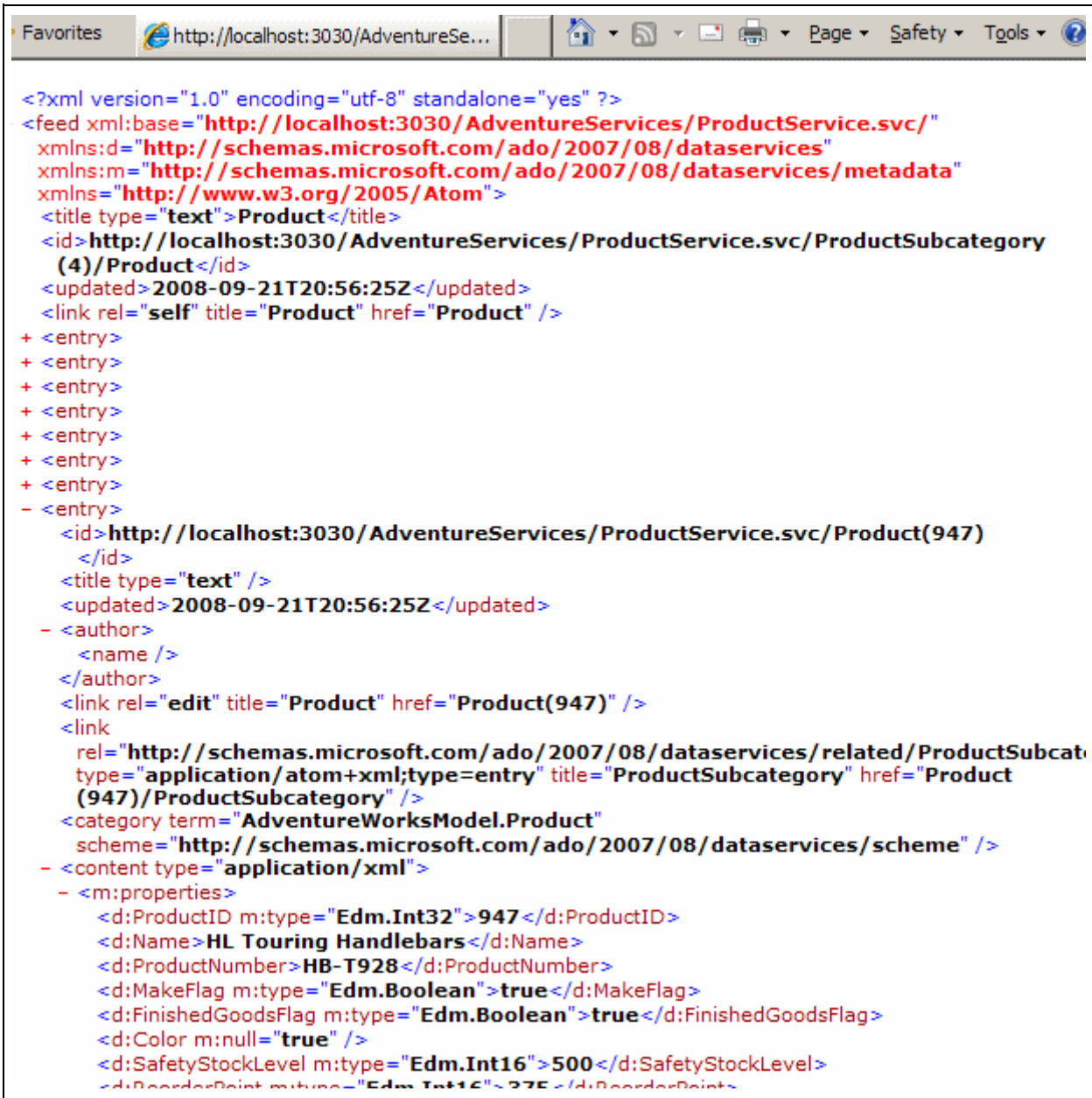



```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
<feed xml:base="http://localhost:3030/AdventureServices/ProductService.svc/"
  xmlns:d="http://schemas.microsoft.com/ado/2007/08/dataservices"
  xmlns:m="http://schemas.microsoft.com/ado/2007/08/dataservices/metadata"
  xmlns="http://www.w3.org/2005/Atom">
   <title type="text">Product</title>
   <id>http://localhost:3030/AdventureServices/ProductService.svc/Product</id>
   <updated>2008-09-21T20:50:42Z</updated>
   <link rel="self" title="Product" href="Product" />
+ <entry>
+ <entry>
+ <entry>
+ <entry>
- <entry>
     <id>http://localhost:3030/AdventureServices/ProductService.svc/Product(753)</id>
     <title type="text" />
     <updated>2008-09-21T20:50:42Z</updated>
   - <author>
       <name />
     </author>
     <link rel="edit" title="Product" href="Product(753)" />
     <link
       rel="http://schemas.microsoft.com/ado/2007/08/dataservices/related/ProductSubcate
       type="application/atom+xml;type=entry" title="ProductSubcategory" href="Product
       (753)/ProductSubcategory" />
     <category term="AdventureWorksModel.Product"
       scheme="http://schemas.microsoft.com/ado/2007/08/dataservices/scheme" />
   - <content type="application/xml">
     - <m:properties>
         <d:ProductID m:type="Edm.Int32">753</d:ProductID>
         <d:Name>Road-150 Red, 56</d:Name>
         <d:ProductNumber>BK-R93R-56</d:ProductNumber>
         <d:MakeFlag m:type="Edm.Boolean">true</d:MakeFlag>
         <d:FinishedGoodsFlag m:type="Edm.Boolean">true</d:FinishedGoodsFlag>
         <d:Color>Red</d:Color>
         <d:SafetyStockLevel m:type="Edm.Int16">100</d:SafetyStockLevel>
         <d:ReorderPoint m:type="Edm.Int16">75</d:ReorderPoint>
         <d:StandardCost m:type="Edm.Decimal">2171.2942</d:StandardCost>
         <d:ListPrice m:type="Edm.Decimal">3578.2700</d:ListPrice>
         <d:Size>56</d:Size>
         <d:SizeUnitMeasureCode xml:space="preserve">CM</d:SizeUnitMeasureCode>
         <d:WeightUnitMeasureCode xml:space="preserve">LB</d:WeightUnitMeasureCode>
         <d:Weight m:type="Edm.Decimal">14.68</d:Weight>
```

örnek 4 : ProductSubcategoryID değeri 1 olan alt kategoriye bağlı ürünlerin listesinin elde

URL Satırı ifadesi :
http://localhost:3030/AdventureServices/ProductService.svc/ProductSubcategory(1)/Produc

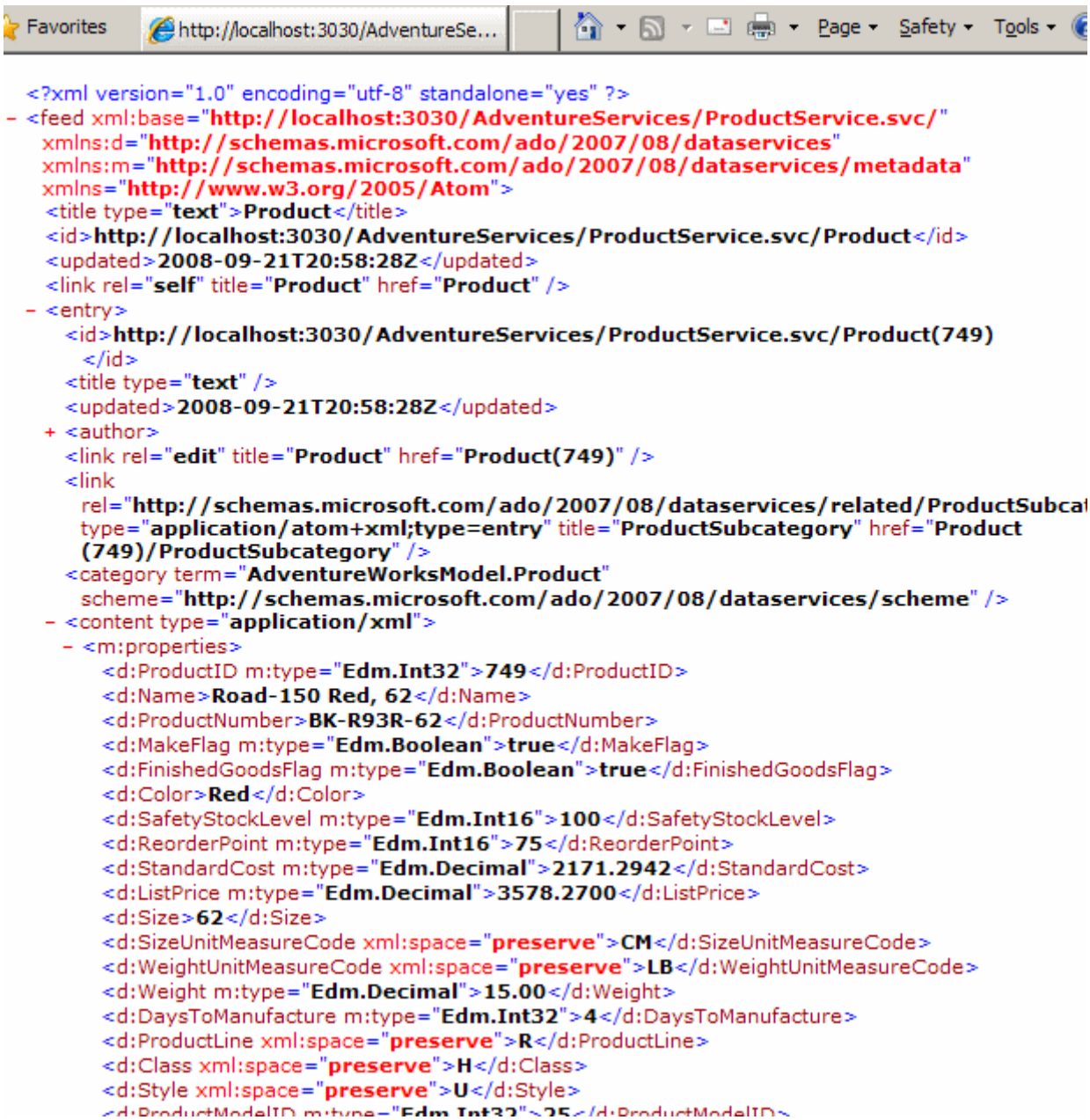
Sonuç Ekran görüntüsü ;



örnek 5 : Product tablosundan ListPrice değeri 3500 birim üzerinde olanların isimlerine gö

URL Satırı ifadesi :
http://localhost:3030/AdventureServices/ProductService.svc/Product?$filter=ListPrice gt 35
(İki ayrı sorgu ifadesi birleştirilirken & kullanılır)

Sonuç Ekran görüntüsü ;

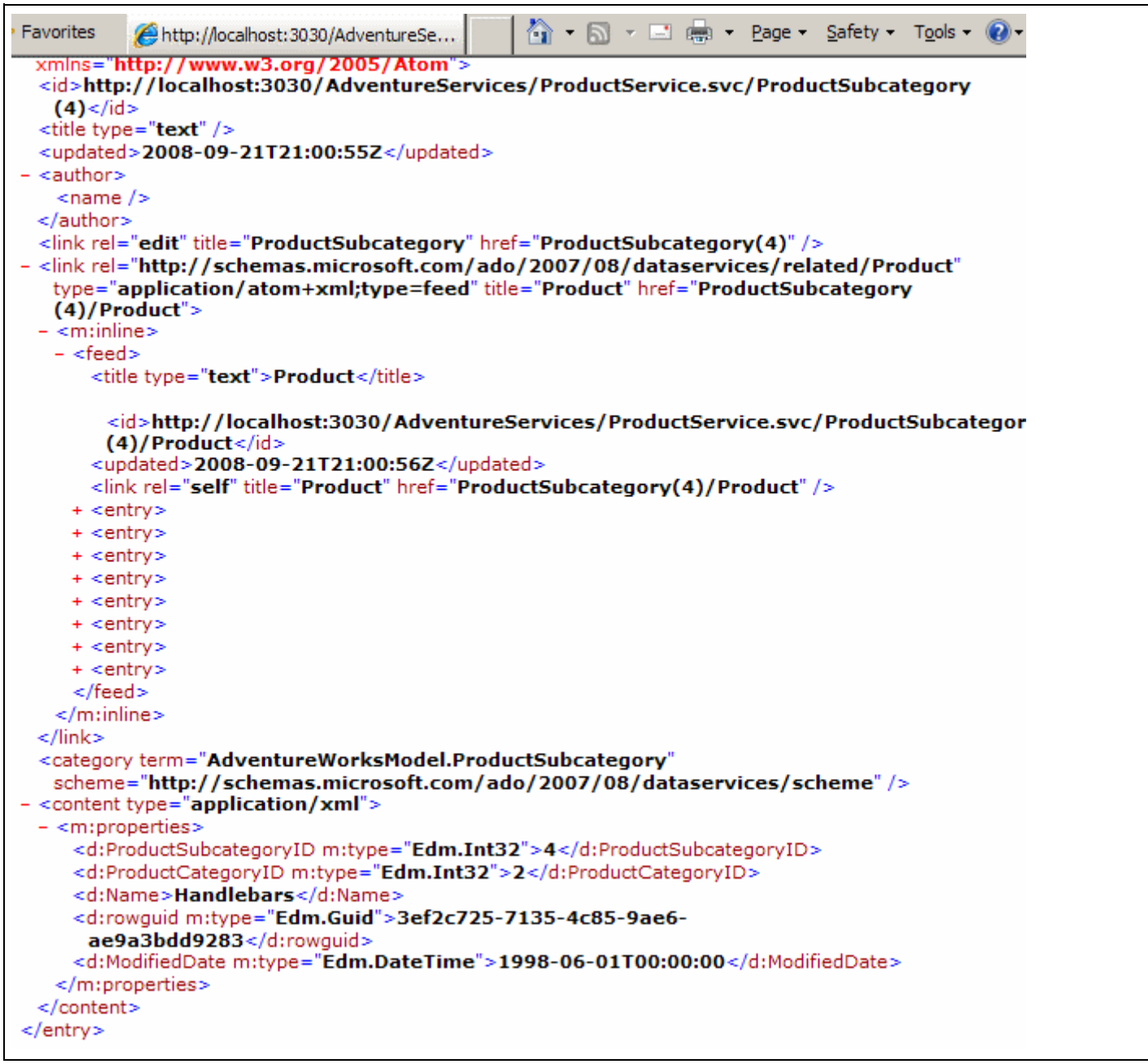


örnek 6 : ProductSubCategoryID değeri 4 olan alt kategorinin bilgilerinin elde edilmesi ve l

URL Satırı ifadesi :
http://localhost:3030/AdventureServices/ProductService.svc/ProductSubcategory(4)?$expa

Sonuç Ekran görüntüsü ;

Dikkat edileceği üzere inline elementi altında 4 numaralı alt kategoriyi bağlı Product örneklerinin
komutunun kullanılabilmesini sağlayan üyenin ProductSubcategory sınıfındaki Product özelliği c



Buraya kadar anlatıklarımız umarım sizlere **Ado.Net Data Services** hakkında biraz fikir verebilmiştir. Eğer buraya kadar makaleyi zevkle okuduysanız işte size yapmanız gereken bir kaç ödev. öncelikli olarak tarayıcı uygulama üzerinden sorgu gönderdiğinizde **SQL** tarafında nasıl komutlar çalıştırıldığına bakmanızı öneririm. Kod tarafında ilgili noktalara**breakpoint**' ler ekleyerek nesnelerin ne zaman örneklendiklerine*(REST sorgusu SQL sorgusuna dönüştürülmeden öncemi, sonra mı gibi)* bakmanızı öneririm. Başka ne çeşit sorgular yazabileceğinizi **filter, orderby, expand, ()** dışında ne gibi **query** komutları olabileceğini araştırın. Bu araştırmalarıda başarı ile yaparsanız şöyle güzel bir sütlü nescafe' yi deniz kenarında yudumlamayı hak etmişsiniz demektir, üstelik güneş batarken.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde kısaca **Ado.Net Data Services(Astoria)** konusuna değinmeye çalıştık. özet olarak, **ADO.Net EDM(Entity Data Model)** veya **LINQ Provider** seçeneklerini kullanaraktan,
verilerin **REST(REpresentational State Transfer)** modele uygun bir servis üzerinden yayınlanabileceğini gördük. Bu makalemizde herhangibir istemci uygulama geliştirmemiş olmamıza rağmen, sonuçları değerlendirmek adına bir tarayıcı uygulama kullandığımızı unutmayalım. Nitekim tarayıcı uygulamalarda hangi çeşitten olurlarsa olsunlar potansiyel olarak birer servis istemcisidir. Elbette ilerleyen makalelerimizde istemci uygulamaların nasıl geliştirilebileceğinede değinebileceğimizi belirtmek isterim. **Ado.Net Data Service**' ler ile ilişkili yazı dizimizin bu ilk bölümüne ait görsel bir derside **.Net TV'** den de izleyebilirsiniz. Bununla birlikte **Ado.Net Data Services** ile ilişkili blog bilgilerine Microsoft' un **şu** adresinden ulaşabilirsiniz. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

AstoriaHelloWorld.rar (8,36 kb)

OPML İçeriği (2008-07-01T16:10:00)

opml,

Şu anda takip etmekte olduğum önemli kişi, topluluk ve takımların blolarını içeren OPML dosyasıdır.

feedreader.opml (21,53 kb) **[11.Eylül.2010 tarihli güncel içeriktir]**

WCF ile P2P(Peer To Peer) Programlama (2008-05-25T03:42:00)

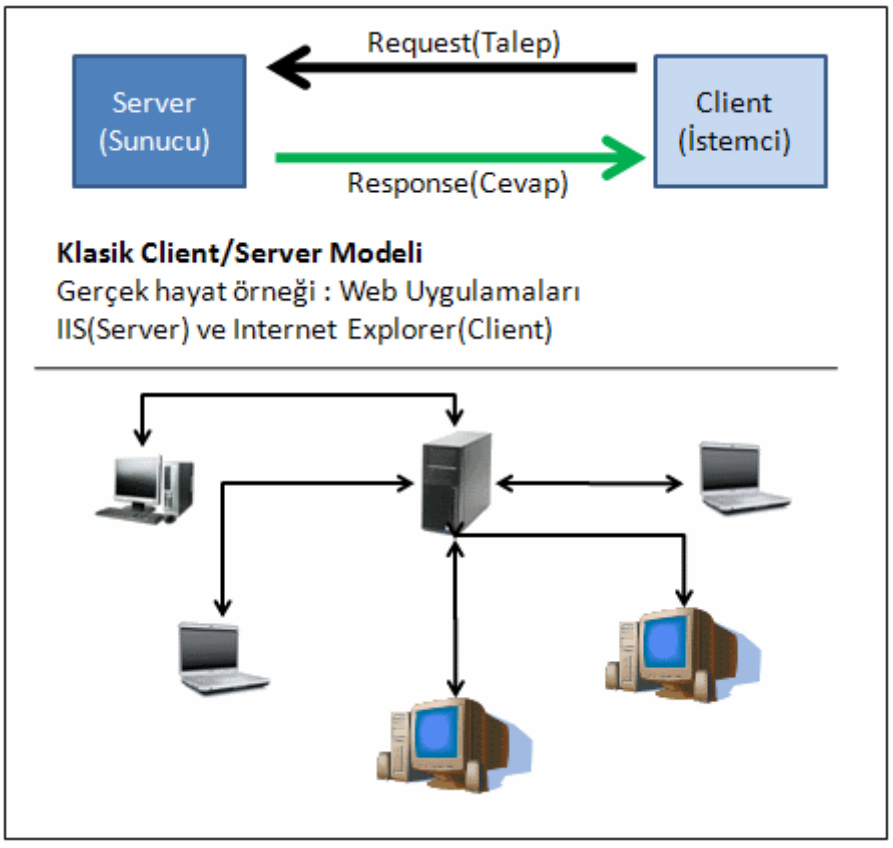
wcf,

Dağıtık mimari uygulamaları(Distributed Applications) geliştirilirken çoğunlukla **Client/Server** veya **N-Tier** modelleri göz önüne alınmaktadır. Oysaki dağıtık mimari uygulamaları için **Peer-to-Peer(P2P)** modelide söz konusudur. **P2P** modelinde istemci ve sunucu arasında bir fark yoktur ve alt yapı hazırlıkları diğer modellere göre biraz daha karmaşıktır. Programlama zorluğu nedeni ile geliştiriciler zaman zaman bu modelden kaçınırlar. Oysaki günümüzde P2P üzerine kurulu olan pek çok sistem yer almaktadır. P2P uygulamalarına örnek olarak,**IRC(Internet Relay Chat), anında mesajlaşma(Instant Messaging), dosya paylaşım(File Sharing), Oyunlar, görsel ve sesli veri aktarımı, veri kopyalama(Data Replication)** programları gösterilebilir. Dolayısıyla P2P modelinin dağıtık mimari çözümlerinde aslında önemli bir yeri bulunmaktadır. P2P tabanlı sistemlerde özellikle **ölçeklenebilirlik(Scalability)** ve**güvenilirlik(Reliability)**, diğer dağıtık mimari modelleri ile kıyaslandığında daha yüksek verimlilik göstermektedir.

NOT : *P2P programlar başarılı bir şekilde tesis edildiklerinde* ***scalability*** *ve* ***reliability*** *manasında belirgin üstünlük ve avantajlar sağlamaktadır.*

- ***Scalability*** *: Kaynak kullanımının artmasıyla sistem performansının değer kaybetmeden yükselebilmesi ölçeklenebilirlik olarak ifade edilebilir. Bu oranın yüksek olması bir avantaj olarak görülebilir.*
- ***Reliability*** *: Kurulu olan sistemin güvenilirliğini, aynı zamanda devamlılığını ifade eder.*

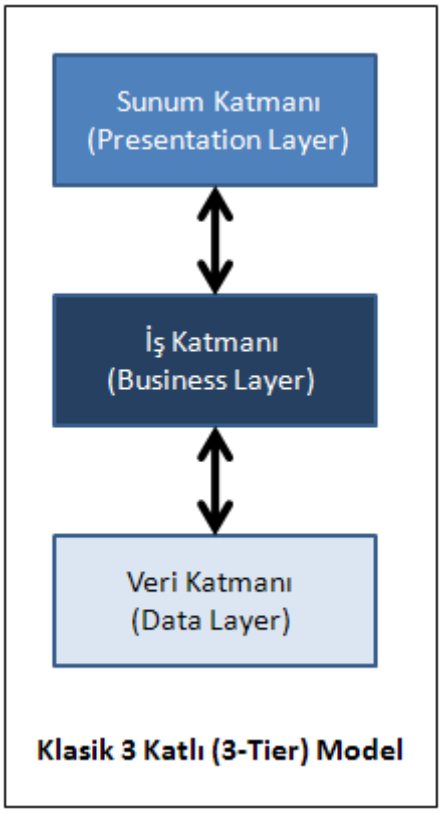
çok doğal olarak **Windows Communication Foundation(WCF)** içerisindede **P2P** desteği bulunmaktadır. Bu bölümde WCF mimarisi ile P2P çözümlerinin nasıl geliştirilebileceğine giriş yapılmaktadır. Başlamadan önce diğer dağıtık mimari modellerine kısaca bir göz atmakta ve P2P ile aralarındaki farkları analiz etmeye çalışmakta yarar bulunmaktadır.**Client/Server(İstemci/Sunucu)** modeli aşağıdaki şekil ile basitçe özetlenebilir.



Bu modelde istemci, sunucu üzerinden yayınlanan fonksiyonellikler için **talepte(Request)** bulunur. Sunucunun görevi ise bu taleplere karşılık **cevaplar(Response)** üretmektir. Bu mimariye verilebilecek en güzel örnek **Web sunucuları** ve **tarayıcı uygulamalardır(Browser Applications)**. Söz gelimi **IIS(Internet Information Services)** sunucu görevini üstlenirken, **Internet Explorer, Netscape Navigator, Mozilla Firefox** gibi tarayıcı uygulamalar istemci rolünü yüklenmektedir. Bu

modelde aslında istemci ve sunucu uygulamalar aynı sistemin bir parçasıdır. Her zaman **IIS** yada **Internet Explorer** gibi hazır program arayüzleri olmayabilir. Söz gelimi WCF mimarisinde **Self Hosting** tekniğine göre sunucu program, **Console**, **Windows**, **WPF**, **Windows Service, WAS(Windows Activation Service)** uygulaması olacak şekilde geliştirilebilir. Buna paralel olarak istemci tarafı içinde aynı durum geçerlidir. Yinede sonuç itibariyle istemci tarafı talepte bulunan, sunucu tarafı ise bu talepleri karşılayan roldedir. Client/Server modelinde merkezileştirme olanağının bulunması bir avantaj olarak görülebilir.

N-Tier yada çok katmanlı mimaride, dağıtık uygulama geliştirme modellerinden birisidir. Aslında bu model dağıtık mimari tarafında çoğunlukla **3 katmanlı(3-Tier)** olacak şekilde uygulanmaktadır. Aşağıdaki şekilde bu durum kısaca ifade edilmeye çalışılmaktadır.

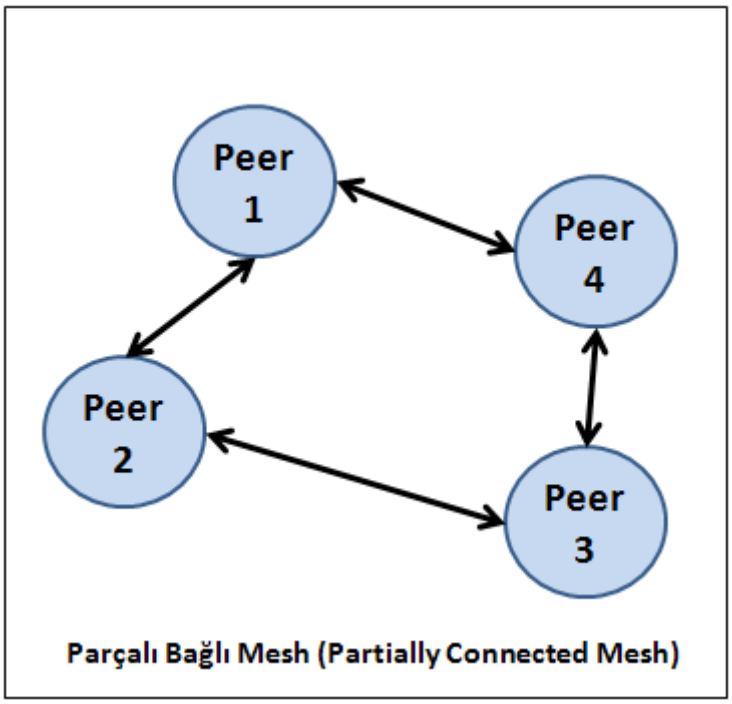


Bu model aslında **Client/Server** mimarinin genişletilmiş bir hali olarak düşünülebilir. Katmanlar ayrı fiziki parçalara bölümlenebilmektedir. Dağıtık mimari açısından özellikle **iş mantığının(Business Logic)** farklı bir makineye alınması **ölçeklenebilirliği(Scalability)** arttıran bir faktördür. Hatta bu fiziki bölünme ile **yük dengesinin dağıtılması(Load Balancing)**, daha **güvenli(Secure)** bir ortamın tesis edilmesi gibi avantajlarda sağlanabilmektedir. Tabiki, bu fiziki bölünüm için donanıma yapılan yatırımda bu işin cabası olarak görülebilir.

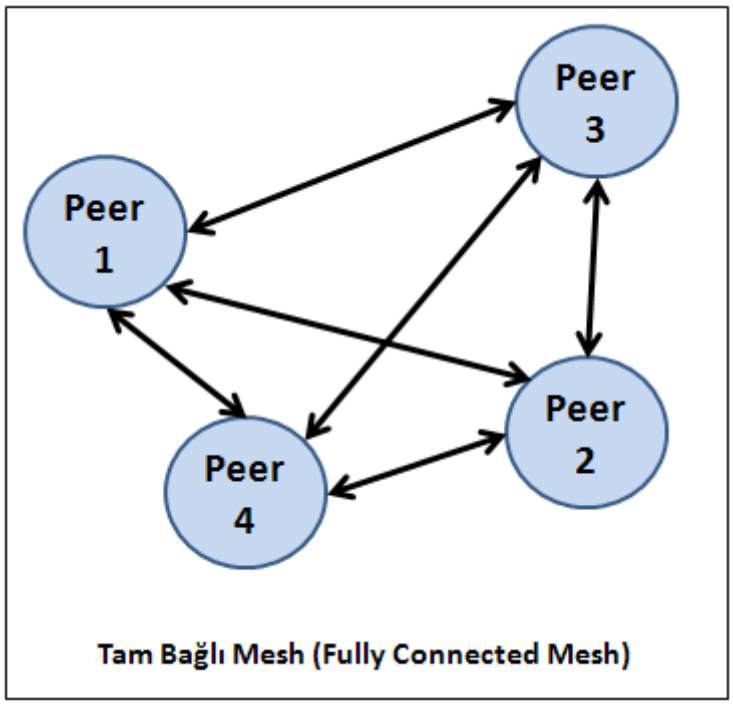
P2P modelinde ise sistemin tüm katılımcıları*(ki bunlar peer node-boğum olaraktanda adlandırılmaktadır)* çoğunlukla hem istemci hemde sunucu görevini üstlenebilmektedir.**Client/Server** veya **N-Tier** modelleri ile dağıtık uygulama çözümleri

geliştirilmesi **P2P**' e göre daha kolaydır. Ayrıca yönetimin merkezileştirilmesi yada güvenliğin güçlü bir şekilde sağlanmasıda söz konusudur. Fakat bölümün başındada belirtildiği üzere, **ölçeklenebilirlik(Scalability)** ve **güvenilirlik(Reliability)** gibi konularda P2P mimarisi gerçekten öne çıkmaktadır. *(Tabiki, ölçeklenebilirlik olgusu, P2P dışındaki modellerdede vardır ancak maliyeti daha yüksek olmaktadır. özellikle sunucu yatırımları göz önüne alındığında.)* Elbetteki bu farklılıklar gerçek hayat vakalarında duruma göre tercih edilebilirler. Hatta bazı vakalarda karma modellerin kullanıldığıda görülmektedir.

Genel olarak **P2P** modelinde yer alan uygulamalar bir **Mesh Network** içerisinde gruplanırlar. Söz konusu **Mesh Network'** lerin iki farklı uygulanış biçimi vardır. Bunlardan birisi **Parçalı Bağlı Mesh(Partially Connected Mesh)** modelidir ve aşağıdaki şekildekine benzer bir yaklaşım sunmaktadır.

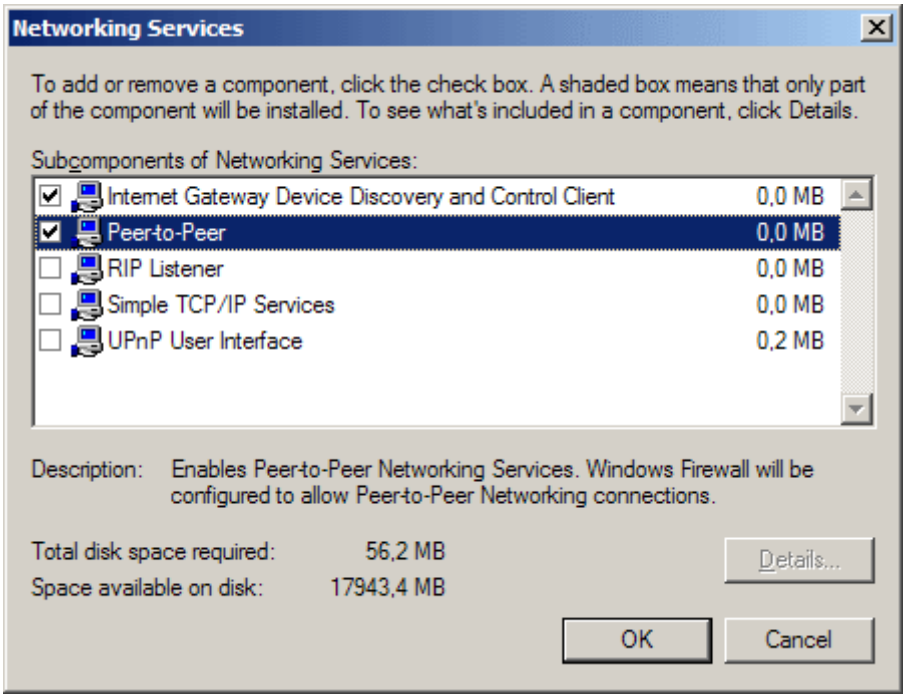


Bu modele göre **Mesh Network** içerisinde yer alan **boğumlar(Peer Nodes)** yakınlarındaki komşularına doğrudan bağlıdır. Bir başka deyişle sistem içerisindeki programlar en yakın bilgisayardaki ile konuşabilmektedir. *(Birbirlerinin komuş boğumları olarakta görülebilirler.)* Bu modelde boğumların tamamı birbirlerine bağlı değildir. Bu nedenle örneğin komşu olmayan bir boğumda yer alan katılımcıya mesaj aktarımı için, mesajın sırayla birbirlerine bağlı olan boğumlar üzerinden hareket etmesi gerekmektedir. Bu bir dezavantaj olarak görülebilir. Diğer taraftan bu tip sistemlerde Mesh Network' e dahil olan katılımcı sayısının artması ile birlikte, **ölçeklenebilirliğinde arttrığı** gözlemlenmektedir. Doğal olarak bu bir avantajdır. Diğer modelde ise **Mesh Network** içerisinde yer alan tüm katılımcılar(PeerNode) birbirlerine bağlıdır. Bu durum aşağıdaki şekil ile özetlenebilir.



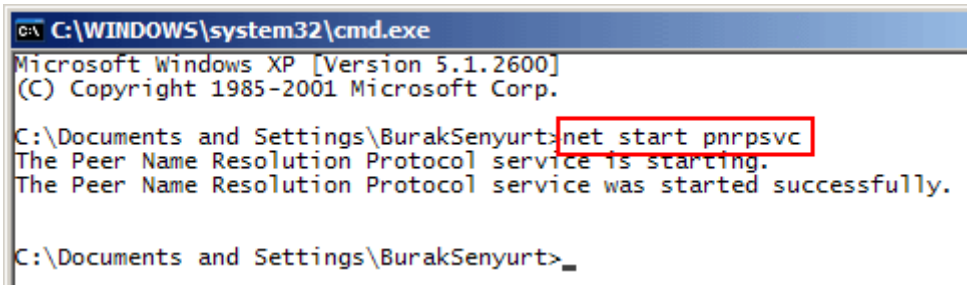
çoğunlukla **Mesh Network** içerisindeki katılımcı sayılarının düşük olduğu durumlarda tercih edilen bir modeldir. çok doğal olarak Mesh ağlarında katılımcı uygulamaların birbirlerini bulabilmesi veya haberleşebilmesi için bir takım protokollerin rol oynaması gerekmektedir. Söz gelimi **UDP(User Datagram Protocol)** gibi. **Windows Communication Foundation** bu amaçla**Peer Name Resolution Protocol(PNRP)** isimli protokolü varsayılan olarak kullanmaktadır. Bu protokol **Windows XP Service Pack 2** ve **Windows Vista** ile birlikte gelmektedir. Ancak istenirse **PNRP** yerine kullanılacak özel **çözücüler(Resolver)** ele alınabilir ki bu bölümde ele alınacaktır.

NOT : PNRP(Peer Name Resolution Protocol) *protokolü sistemde yüklü değilse,* ***Windows XP Service Pack 2*** *için* ***Add Remove Windows Components*** *kımsına girip,* ***Network Services****bileşenlerinden* ***Peer to Peer*** *birimini eklemek yeterli olacaktır.*



özellikle uygulama geliştirildikten sonra ***"System.InvalidOperationException: Resolver must be specified. The default resolver (PNRP) is not available."*** *gibi bir çalışma zamanı istisnasının(Runtime Exception) alınmasının nedeni ilgili PNRP hizmetinin yüklenmemiş olmasıdır.*

PNRP yüklenmesi tek başına yeterli değildir. ***Peer Name Resolution Protocol Service*** *çalışıyor olması gerekmektedir. Bunun için Administrator Tools bölümden yer alan* ***Services*** *kısmından yararlanılabilir yada komut satırından* ***net start pnrpsvc*** *emri aşağıdaki ekran görüntüsünde oldugu gibi uygulanabilir.*



Ne varki ***PNRP*** *sadece* ***XP*** *ve* ***Vista*** *için geçerlidir. Söz gelimi* ***Windows Server 2003*** *sisteminde çalıştırılamamaktadır. Diğer taraftan* ***PNRP IPv6*** *alt yapısını kullanmaktadır. Bu nedenle* ***IPv6*** *desteği olmayan yönlendiricilerin(Routers) olduğu sistemlerde ala alınamamaktadır. Ayrıca* ***Vista*** *ile birlikte* ***PNRP 2.0*** *versiyonu gelmektedir. Buna göre* ***XP*** *ile* ***Vista*** *sistemlerin birbirleriyle konuşması gerektiği durumlarda XP sistemler için ek yükleme yapılması gerekmektedir. İşte bu tip sorunların olabileceği vakalarda,* ***özel peer çözücülerinin(Custom Peer Resolver)*** *geliştirilmesi ve kullanılması önerilmektedir. Bu sayede* ***Mesh Network*** *içerisine dahil olan katılımcıların tutuluş şekilleride özelleştirilebilir. Söz gelimi veritabanı üzerinde saklanabilir.*

P2P programlamada önem arz eden konulardan biriside mesajların ulaştırılma şeklidir. Bu noktada iki farklı teknik
bulunmaktadır. **Directional** ve **Flooding**. **Directional** mesajlaşmaya göre, **Mesh Network** içerisinde yer alan herhangibir boğumdan(Node) çıkan mesaj, **hedef boğuma(node)** ulaşıncaya kadar komşu boğumlar üzerinden
yönlendirilir. **Flooding** haberleşmede ise mesaj,**Mesh Network** içerisindeki tüm boğumlara gönderilir ve mesajı alması gereken boğum tarafından yakalanır. çok doğal olarak **Flooding** modelinde aynı **Mesh Network** içerisinde yer alan bir boğumda çalışan uygulamaya mesajın defalarca gelmesi söz konusu
olabilmektedir. **WCF** mimarisi, **P2P** programlamada **Flooding** mesajlaşma modelini kullanmaktadır. Bununla birlikte **WCF**geliştiricisi için en önemli olan noktalardan birisininde bağlayıcı tip seçimi olduğu bilinmektedir. **WCF,** P2P için **bağlayıcı tip(Binding Type)** olarak **NetPeerTcpBinding** sınıfını ele alınmaktadır.

Bu basit teknik detaylardan sonra **WCF** tarafından **P2P** tarzı bir uygulamanın nasıl geliştirilebileceği adım adım incelenebilir. örnekte **PNRP** hizmeti
yerine **CustomPeerResolverService** tipi kullanılarak bir servis uygulaması tasarlanacak ve **IRC** benzeri basit bir program ortamı geliştirilecektir. Bu çözümsel yaklaşım aynı zamanda **Microsoft** tarafındanda önerilmektedir. Bu amaçla özel çözücüyü içeren ayrı bir servis uygulaması yazılmalıdır. örnek olarak bu bir **Console** uygulaması olabilir. Söz konusu uygulamanın kod içeriği aşağıdaki gibidir.

özel çözücü Service Kodu;

```
using System;
using System.ServiceModel;
using System.ServiceModel.PeerResolvers;

namespace IRCServer
{
   class Program
   {
      static void Main(string[] args)
      {
         CustomPeerResolverService resolver = new CustomPeerResolverService();
         ServiceHost customResolver = new ServiceHost(resolver);

         resolver.Open();
         customResolver.Open();
         Console.WriteLine("çözümleyici aktif. çıkmak için bir tuşa basınız...");
         Console.ReadLine();
      }
```

```
    }
}
```

Burada dikkat edilmesi gereken ilk nokta **CustomPeerResolverService** sınıfına ait bir nesne örneği kullanılmasıdır. **CustomPeerResolverService** sınıfı özel peer çözücü için gerekli **temel uyarlamaları(Basic Implementations)** içermektedir. Söz konusu tip **IPeerResolverContract arayüzünden(Interface)** türetilmiştir. **IPeerResolverContrac t** arayüzü özellikle peer uyarlamasında özel işlemler yapılması istendiği durumlarda ele alınmaktadır. Söz gelimi peer çözümleyicisinin veritabanı ilişkili olması isteniyorsa, **IPeerResolverContract**arayüzünü uygulayan bir sınıfın yazılması ve ilgili üyelerin ezilmesi gerekmektedir. Sonrasında ise uygulama içerisinde bu sınıfın, peer çözücü olarak kullanılacağı belirtilmelidir. örnekte böyle bir ihtiyaç olmadığından **.Net Framework 3.0** ile birlikte gelen hazır **CustomPeerResolverService** sınıfı kullanılmaktadır.

çözücü Servise ait konfigurasyon içeriği;

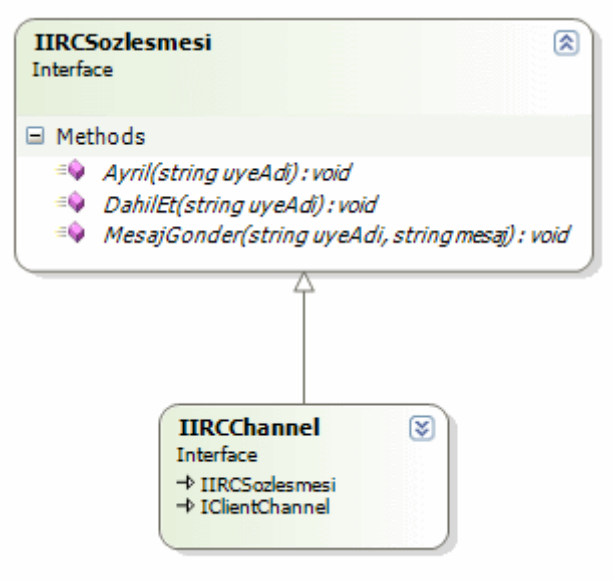
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service
name="System.ServiceModel.PeerResolvers.CustomPeerResolverService">
        <endpoint address="net.tcp://localhost:4500/IRCResolver"
binding="netTcpBinding" bindingConfiguration="ResolverBindingConfig"
contract="System.ServiceModel.PeerResolvers.IPeerResolverContract" />
      </service>
    </services>
    <bindings>
      <netTcpBinding>
        <binding name="ResolverBindingConfig">
          <security mode="None"/>
        </binding>
      </netTcpBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

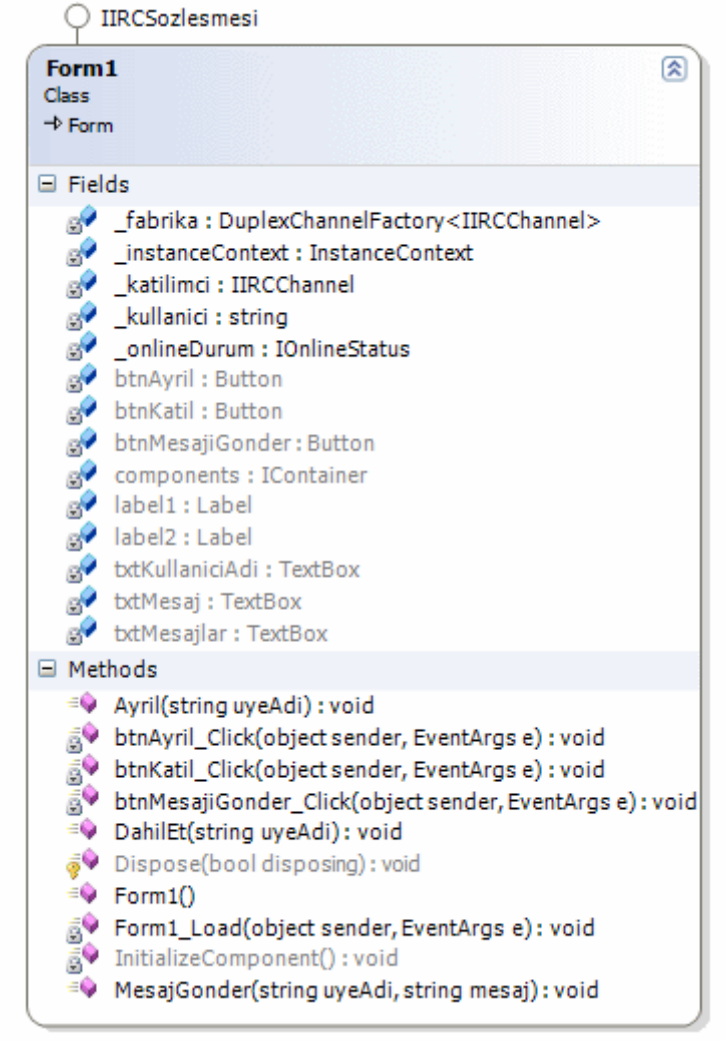
services elementi içerisine dikkat edilecek olursa, **net.tcp://localhost:4500/IRCResolver** adresi üzerinden **NetTcpBinding** bağlayıcı tipini ele alan, herhangibir güvenlik modunun kullanılmadığı ve **IPeerResolverContract sözleşmesini(Service Contract)** bildiren bir **EndPoint** tanımlaması yapıldığı görülmektedir. Elbette, **Mesh Network** içerisine dahil olacak olan katılımcıların*(PeerNode)* birbirleriyle haberleşebilmeleri için tasarlanan bu özel peer çözücü servis uygulamasının çalışıyor olması gerekmektedir. çok doğal olarak bu

uygulama ağ ortamında bir sunucu üzerinde host edildiği takdirde*(Söz gelimi **Windows Server 2003** üzerinde koşan bir **Windows Service** içerisinden sunulabilir)* ağa bağlı tüm istemci uygulamaların birbirleriyle**P2P** üzerinden haberleşebilmesi sağlanabilir. üstelik TCP bazlı yayınlama yapıldığı için performans açısından da oldukça tatmin edici sonuçlar alınacaktır.

İstemci tarafında yer alacak ve Mesh ağına katılımcı(PeerNode) olarak dahil olacak program basit bir **Windows** uygulaması olarak tasarlanmaktadır. Bu uygulamanın görsel arayüzü aşağıdaki gibidir.

Kullanıcılar istedikleri bir isim ile **Mesh ağına** dahil olabileceklerdir. Bununla birlikte mesaj göndermek, diğer katılımcıların mesajlarını görmek veya **IRC** kanalından ayrılmak gibi işlemleride yapabileceklerdir. Katılımcılar için önemli olan üç operasyon söz konusudur. **Mesh** ağına dahil olmak, ağdan ayrılmak ve ağdaki herkese mesaj gönderebilmek. Bunların bir servis sözleşmesine ait operasyonlar olduğu açıktır. Dolayısıyla istemci tarafında bu işlevsellikleri barındıran bir sözleşme arayüzü tasarlanmalıdır. Diğer taraftan bu tip bir **P2P**çalışmada, kanalın **çift yönlü iletişim(Duplex Communication)** sağlayacak şekilde çalışabiliyor olması ve hatta söz konusu operasyonların asenkron olarak yürütülebilmesi gereklidir. Bu sebepten **DuplexChannel** kullanılması ve operasyonların **OneWay** olarak işaretlenmesi gereklidir. **Windows** uygulamasına ait olan **Form** sınıfının kendisi bu senaryoda**sözleşmeyi(Service Contract)** uygulayan tip olacaktır. Bunlara ek olarak istemci tarafından açılacak olan kanal içinde **IClientChannel** arayüzünün hesaba katılması gerekmektedir. Katılımcı uygulama tarafından bakıldığında **sınıf diagramı(Class Diagram)** içeriği aşağıdaki gibidir.





Katılımcı uygulamanın kodları aşağıdaki gibidir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.ServiceModel;
using System.ServiceModel.Channels;

namespace ChatApp
{
   public partial class Form1
      : Form
      ,IIRCSozlesmesi
   {
      string _kullanici=null;
      InstanceContext _instanceContext=null;
      DuplexChannelFactory<IIRCChannel> _fabrika = null;
      IIRCChannel _katilimci = null;
      IOnlineStatus _onlineDurum = null;

      public Form1()
      {
         InitializeComponent();
      }

      private void Form1_Load(object sender, EventArgs e)
      {
         btnKatil.Enabled = true;
         btnAyril.Enabled = false;
         btnMesajiGonder.Enabled = false;
         txtMesajlar.ScrollBars = ScrollBars.Vertical;
      }

      #region IChat Members

      public void DahilEt(string uyeAdi)
      {
         txtMesajlar.Text += String.Format("*** {0} IRC' ye katıldı. ***", uyeAdi);
         txtMesajlar.Text += Environment.NewLine;
      }

      public void MesajGonder(string uyeAdi, string mesaj)
```

```
    {
       txtMesajlar.Text += String.Format("({0}) -> {1}", uyeAdi,mesaj);
       txtMesajlar.Text += Environment.NewLine;
    }

    public void Ayril(string uyeAdi)
    {
       txtMesajlar.Text += String.Format("*** {0} IRC' den ayrıldı. ***", uyeAdi);
       txtMesajlar.Text += Environment.NewLine;
    }

    #endregion

    private void btnKatil_Click(object sender, EventArgs e)
    {
       if (!String.IsNullOrEmpty(txtKullaniciAdi.Text))
          _kullanici = txtKullaniciAdi.Text;
       else
          _kullanici = "İsimsiz";

       _instanceContext = new InstanceContext(this);
       _fabrika = new DuplexChannelFactory<IIRCChannel>(_instanceContext,
"ClientEndPoint");
       _katilimci = _fabrika.CreateChannel();
       _onlineDurum = _katilimci.GetProperty<IOnlineStatus>();

       _onlineDurum.Online += delegate(object snd, EventArgs ea)
                              {
                                 txtMesajlar.Text += "*** Hat Açık ***";
                                 txtMesajlar.Text += Environment.NewLine;
                                 txtMesajlar.Text += _onlineDurum.ToString();
                                 txtMesajlar.Text += Environment.NewLine;
                              };
       _onlineDurum.Offline += delegate(object snd, EventArgs ea)
                              {
                                 txtMesajlar.Text += "*** Hat Kapalı ***";
                                 txtMesajlar.Text += Environment.NewLine;
                                 txtMesajlar.Text += _onlineDurum.ToString();
                                 txtMesajlar.Text += Environment.NewLine;
                              };
       _katilimci.Open();
       _katilimci.DahilEt(_kullanici);

       btnKatil.Enabled = false;
       btnAyril.Enabled = true;
```

```
            btnMesajiGonder.Enabled = true;
        }

        private void btnAyril_Click(object sender, EventArgs e)
        {
            _katilimci.Ayril(_kullanici);
            _katilimci.Close();
            _fabrika.Close();

            btnKatil.Enabled = true;
            btnAyril.Enabled = false;
            btnMesajiGonder.Enabled = false;
        }

        private void btnMesajiGonder_Click(object sender, EventArgs e)
        {
            _katilimci.MesajGonder(_kullanici, txtMesaj.Text);
            txtMesaj.Clear();
        }
    }

    public interface IIRCChannel
        : IIRCSozlesmesi, IClientChannel
    {
    }

    [ServiceContract(CallbackContract = typeof(IIRCSozlesmesi))]
    public interface IIRCSozlesmesi
    {
        [OperationContract(IsOneWay = true)]
        void DahilEt(string uyeAdi);

        [OperationContract(IsOneWay = true)]
        void MesajGonder(string uyeAdi, string mesaj);

        [OperationContract(IsOneWay = true)]
        void Ayril(string uyeAdi);
    }
}
```

IIRCSozlemesi isimli arayüz(Interface), servis sözleşmesini tanımlamaktadır. Bu sözleşme içerisinde yer alan **DahilEt**, **MesajGonder** ve **Ayril** isimli operasyonlar asenkron(asynchronous) olarak çalışmaktadır ve tek yönlüdür(OneWay). Bu sebepten **IsOneWay** özelliklerine **true** değeri atanmıştır. Diğer taraftan **CallbackContract** olarak **IIRCSozlesmesi** arayüzünün kendisi bildirilmiştir.

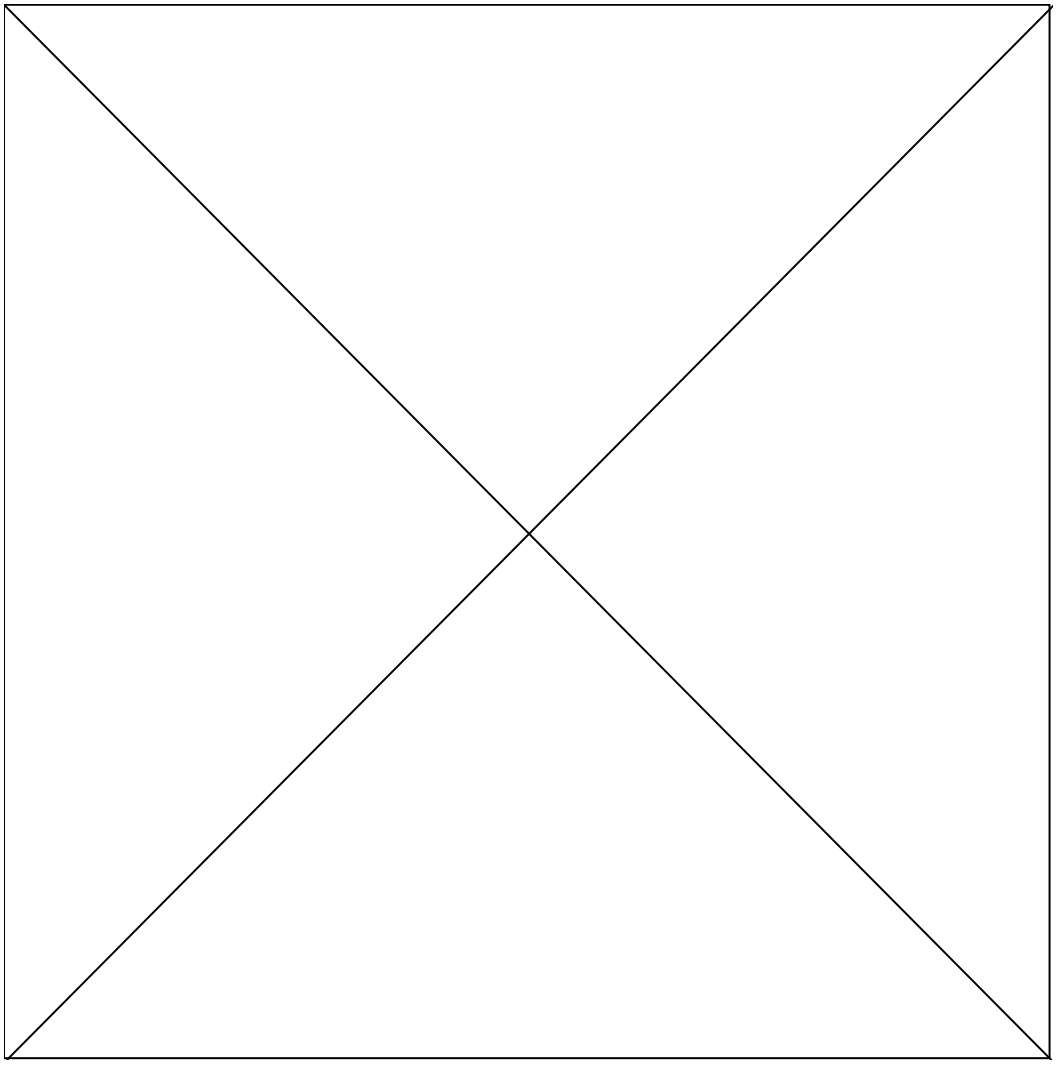
Bilindiği gibi **çift yönlü iletişim(Duplex Communication)** kurulacağından bir **geri bildirim sözleşmesinin(Callback Contract)** tanımlanması gerekmektedir. Böylece istemciler birbirleri üzerinden bağımsız olarak operasyon çağrıları gerçekleştirebilir ki, **P2P** haberleşmede tarafların birbirleri üzerinde operasyon geri bildirimlerinde bulunması gereklidir. Bu sayede bir katılımcının göndereceği bir mesaj Mesh ağa dahil olan diğer kullanıcılarada iletilebilir.

örnekte yer alan **form** uygulamasının kendisine ait nesne örneği, **Mesh ağın** bir katılımcısı olarak rol oynamaktadır. Bu sebepten **InstanceContext** örneği oluşturulurken parametre olarak **this** verilmiştir. *(Burada this doğal olarak çalışma zamanında Form1 nesne örneğini işaret etmektedir.)* Sonrasında ise bir **DuplexChannelFactory<T>** nesne örneği, InstanceContext nesne örneğinin kendisi ve konfigurasyon dosyasında belirtilen **EndPoint** değeri için örneklenmektedir. **DuplexChannelFactory<T>** generic sınıfı ile, çift yönlü iletişimi ele alacak olan bir referansın üretilmesi sağlanmaktadır. Bu katılımcıların her biri çift yönlü iletişimi sağlayabilecek şekilde birer kanalı, **Mesh** ağına doğru açmaktadır. Bu kanalın oluşturulması sırasındada devreye **CreateChannel** metodu girmektedir. Mesh ağına dahil olan kullanıcı referansından yararlanılarak **online** ve **offline** olma durumlarına ait olaylar da ele alınabilir. Bu amaçla **IOnlineStatus** arayüzünün çalışma zamanında taşıdığı referanstan yararlanılmaktadır. örnekte önemli olan noktalardan biriside konfigurasyon içerisinde yer alan istemci taraflı **EndPoint** bildirimidir. Söz konusu uygulama için bu bildirim aşağıdaki gibidir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <client>
         <endpoint address="net.p2p://BizimIRC" binding="netPeerTcpBinding"
bindingConfiguration="Varsayilan" contract="ChatApp.IIRCSozlesmesi"
name="ClientEndPoint" />
      </client>
      <bindings>
         <netPeerTcpBinding>
            <binding name="Varsayilan" port="0">
               <security mode="None" />
               <resolver mode="Custom">
                  <custom address = "net.tcp://localhost:4500/IRCResolver"
binding="netTcpBinding" bindingConfiguration="ResolverBindingConfig" />
               </resolver>
            </binding>
         </netPeerTcpBinding>
         <netTcpBinding>
            <binding name="ResolverBindingConfig">
               <security mode="None"/>
            </binding>
```

```
        </netTcpBinding>
      </bindings>
   </system.serviceModel>
</configuration>
```

Konfigurasyon dosyasında dikkat edilmesi gereken en önemli noktalardan birisi **netPeerTcpBinding** elementi içerisinde yer alan resolver boğumudur. Bu boğum içerisinde **mode niteliğine(attribute)** verilen**Custom** değeri ile özel bir peer çözücü servisinin kullanılacağı ifade edilmektedir. Diğer tarafan söz konusu özel peer çözücü servisin hangi adres üzerinden **host** ediliği , hangi bağlayıcı tipin(Binding Type) kullanıldığı bilgileri ise **custom** isimli elementin **address** ve **binding** niteliklerinde belirtilmektedir. Dikkat edilecek olursa **address** niteliğinin değeri servis uygulamasının **host** ettiği adrestir. **client** elementi içerisinde tanımlanan **endpoint** boğumunda ise **address** kısmında **net.p2p://BizimIRC** adresi yer almaktadır. Bu adres Mesh ağının adresi olarakta düşünülebilir. Bir başka deyişle katılımcıların dahil olacağı Mesh ağının yolu tayin edilmektedir. Ayrıca bağlayıcı tip olarak **netPeerTcpBinding** kullanılmaktadır. Geliştirilen örnek güvenlik ile ilişkili olaraktan ek bir özellik içermemektedir. Bu nedenle **security** elementlerinde yer alan **mode** değerleri **None** olarak bildirilmiştir. **netPeerTcpBinding** elementinde yer alan **port** niteliğinin değeri **0** olduğu için, katılımcı uygulamalar çalıştıkları makinede boş buldukları bir port üzerinden kanal oluşturacaklar ve Mesh ağına açacaklardır.***(.Net Remoting*** *mimarisinde geliştirilen örneklerdede port değerine 0 verilmesi ile aynı anlamda olduğuna dikkat edelim.)* Artık uygulamalar test edilebilir. Aşağıdaki Flash animasyonunda senaryoya ait sonuçlar irdelenmektedir. *(Flash Video boyutu 560 Kb olup yüklenmesi sabır isteyebilir.)*



Görüldüğü gibi B isimli katılımcı Mesh ağına dahil olduktan sonra sistem online moda geçmiştir. Bununla birlikte Mesh ağına giren her kullanıcı bilgisi diğer bağlı PeerNode' larada anında iletilir. Böylece katılımcılar ağa dahil olan kullanıcıları görebilmektedir. Katılımcıların gönderikleri mesajlar diğer tüm katılımcıların ekranında anında görülebilmektedir. Diğer taraftan ağdan ayrılan katılımcıların bilgileride diğerlerine iletilmektedir. Burada ilgi çekici noktalardan biriside her katılımcı için ayrı Peer Node ID değerlerinin üretilmesidir. Sistemin offline moda geçmesi ise sadece bir katılımcı kalması durumunda meydana gelmektedir.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde **WCF** mimarisi üzerinde **P2P(Peer-to-Peer)** programlamaya giriş yapmaya çalıştık ve örnek olarak basit bir **IRC** senaryosu geliştirdik. Senaryoda, var olan **PNRP(Peer Name Resolution Protocol)** servisi yerine **.Net Framework** ile birlikte gelen **CustomPeerResolverService** sınıfını kullandık. İlerleyen makalelerimizde P2P tabanlı senaryolarda, geliştirici tarafından yazılmış özel bir peer çözümleyici servisin nasıl ele alınabileceğini ve güvenliği nasıl sağlayabileceğimizi incelemeye çalışacağız. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

PeerToPeerProgramming.rar (61,67 kb)

WCF - Performans (2008-05-20T03:36:00)

wcf,

Uzun bir aradan sonra tekrar birlikteyiz. **Windows Communication Foundation** mimarisinin geliştirilmesininin tek amacı, var olan dağıtık mimari modellerini bir çatı altında birleştirmek değildir. Buna paralel olaraktan WCF mimarisi, **.Net Remoting, Xml Web Servisleri, WSE(Web Service Enhancements), MSMQ(Microsoft Message Queue), COM+** gibi pek çok dağıtık uygulama geliştirme modelinin çalışma zamanı alt yapısının kolayca oluşturulabilmesinide hedeflemektedir. Burada dekleratif programlama modelinin benimsenmesinin önemli katkısı vardır. Sonuçta geliştiricinin sıklıkla yapmak zorunda kaldığı alt yapı hazırlıklarının **attribute(nitelik)** veya konfigurasyon bazlı olacak şekilde ayarlanabilmesi son derece avantajlıdır. Bu açıdan bakıldığında olayın kilit noktası WCF' in **ABC(AddressBindingContract)**' sinde yer alan **bağlayıcı tiplerdir(Binding Types)**. Bilindiği üzere WCF mimarisi **.Net Framework 3.0** ile birlikte gelmiştir. Bununla birlikte şu an itibariyle **.Net Framework 3.5** sürümünde ek yeniliklerde içermektedir*(örneğin **Web Programlama Modeli, WorkFlow Foundation ile Tam Entegrasyon, JSON, AJAX** desteği gibi)*. Bağlayıcı tiplerin sayısının çok olmasının, geliştiricilerin karar verme noktasındaki işlerini zorlaştırdığıda bir gerçektir. Bu nedenle çoğunlukla yardımcı tablolardan veya diagramlardan yararlanılmaktadır.

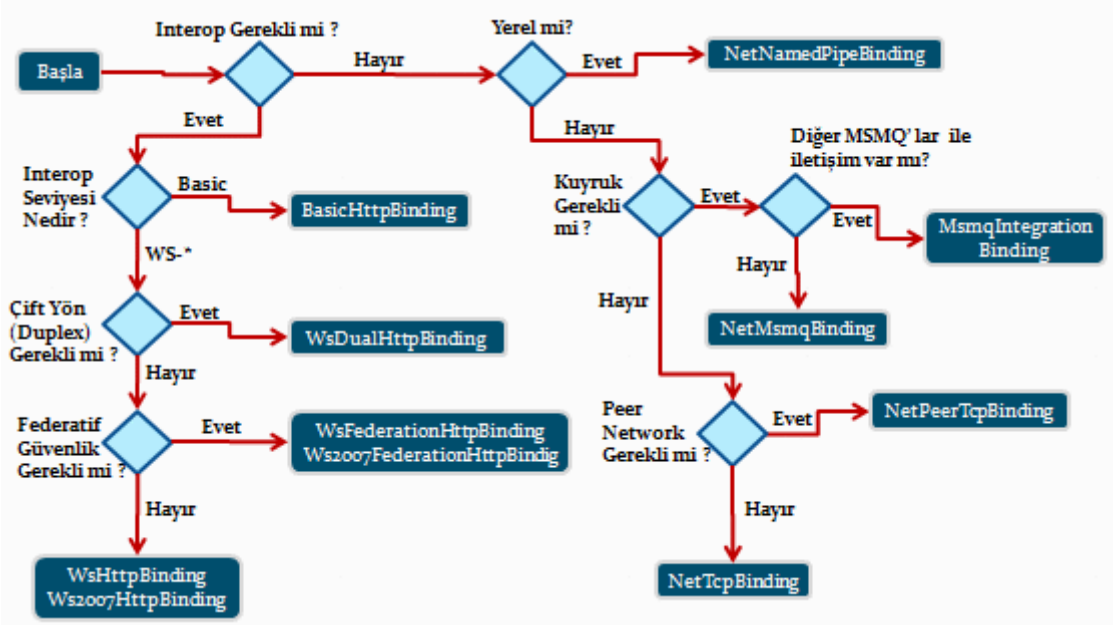
Yazın geldiği bu sıcak günlerde yazdığımız bu makalemizde, biraz hafiften ilerleyecek ve bu bağlayıcı tiplerden çok sık kullanılanlar arasındaki performans farklarını incelemek için nasıl bir yol izleyebileceğimizi incelemeye çalışacağız. Nitekim senaryoya göre seçilebilecek bağlayıcı tiplerin birden fazla olması halinde performans kriterleri ön plana çıkmaktadır. öncelikli olarak şu anda kullanılan bağlayıcı tipler ve aralarındaki belirgin farklıları göz önüne almakta yarar vardır. İşte aşağıdaki tablo bu farklılıkları dile getirmektedir.

Bağlayıcılar(Bindings)		
Bağlayıcı Adı	**Açıklama**	**Framework Versiyonu**
BasicHttpBinding	*Asmx web servis modelini ve WS-I Basic 1.1 profilini destekler. (Eski web servisleri ile konuşalacağı senaryolarda ele alınabilir)*	**3.0/ 3.5**
WsHttpBinding	*WS-* şartnamalerine uygun Web Servisi modelini destekler.*	**3.0/ 3.5**
WsDualHttpBinding	*İki yönlü seri iletişime(Dublex Communication) izin verir.*	**3.0/ 3.5**
WebHttpBinding	*XML ve JSON serileştirme desteği ile servisler üzerinde REST/POX tabanlı iletişime izin verir.*	**3.0/ 3.5**
WsFederationHttpBinding	*WS-* için Federation şartnamelerini destekler.*	**3.0/ 3.5**
Ws2007HttpBinding	*2007 WS-* standartlarını destekler.*	**3.5**
Ws2007FederationHttpBinding	*WsFederationHttpBinding için 2007 standartlarını destekler.*	**3.5**
NetTcpBinding	*İki .Net tabanlı sistem arasında Tcp bazlı iletişimi destekler.*	**3.0/ 3.5**
NetNamedPipeBinding	*Makine üzerinde bir veya daha fazla .Net tabanlı uygulama arasında iletişimi destekler.*	**3.0/ 3.5**
NetMsmqBinding	*MSMQ ile asenkron iletişimi destekler*	**3.0/ 3.5**
NetPeerTcpBinding	*Peer-To-Peer ağ uygulamaları arası iletişimi destekler.*	**3.0/ 3.5**
MsmqIntegrationBinding	*MSMQ tabanlı kuyruk modelini kullanarak mesaj göndirilip alınmasını destekler.*	**3.0/ 3.5**

Görüldüğü gibi bağlayıcı tiplerin birbirlerine göre farklı kullanım sahaları ve senaryoları bulunmaktadır. Buradaki bağlayıcı tiplerde yeterli gelmiyorsa bu durumda **özel bağlayıcı tiplerin(Custom Binding Type)** yazılmasıda tercih edilebilir ki bazı senaryolarda*(örneğin **Front-End Service** yazılması gereken durumlar veya **Reply Attack** gibi vakalarda)* bu gereklidir. Yukarıdaki tablo karar vermek için tam olarak yeterli değildir. Karar verirken ele alınması gereken sorular arasında **platform bağımsızlık(Interoperability), güvenilirlik(Relaibiliyu), performans, WS-* şartnameleri(Specifications), güvenlik(Security)** gibi pek çok kriter bulunmaktadır. İşte bu noktada da devreye aşağıdaki gibi bir tablo girmektedir.

								İletişim Çeşidi		
Bağlayıcı (Binding)	İletişim Seviyesinde Güvenlik (Transport Level Security)	Mesaj Seviyesinde Güvenlik (Message Level Security)	WS-* Desteği	WS-* Transaction Desteği	Güvenilir Mesajlaşma (Reliable Messaging)	Güvenilir Oturumlar (Reliable Sessions)	Performans	İstek / Cevap (Request / Reply)	Tek Yön (One - Way)	Çift Yön (Duplex)
BasicHttpBinding	1	1	1	0	0	0	İyi	1	1	0
WsHttpBinding	1	1	1	1	0	1	İyi	1	1	0
WsDualHttpBinding	1	1	1	1	0	1	İyi	1	1	1
WsFederationHttpBinding	1	1	1	0	0	1	İyi	1	1	0
Ws2007HttpBinding	1	1	1	1	0	1	İyi	1	1	0
Ws2007FederationHttpBinding	1	1	1	0	0	1	İyi	1	1	0
NetTcpBinding	1	1	0	1	0	1	Daha İyi	1	1	1
NetNamedPipeBinding	1	0	0	1	0	0	En İyi	1	1	1
NetMsmqBinding	1	1	0	0	1	0	Daha İyi	0	1	0
NetPeerTcpBinding	1	0	0	0	0	0	İyi	0	1	1
MsmqIntegrationBinding	1	0	0	0	1	0	Daha İyi	0	1	0

Görüldüğü gibi bu tabloda yer alan kriterlerden yararlanarak hangi bağlayıcı tipin kullanılacağı kolay bir şekilde kararlaştırılabilir. Söz gelimi **çift yönlü(Duplex)** iletişimin söz konusu olduğu bir vaka varsa, kriterler sınırlıdır. Bir başka deyişle böyle bir senaryoda **WsDualHttpBinding**, **NetTcpBinding**, **NetNamedPipeBinding** ve **NetPeerTcpBinding** tiplerinden birisi kullanılabilir. Ancak karar vermek halen daha zor olabilmektedir. Belkide bir başlangıç noktası olsa bu iş daha da kolaylaşabilir. O halde bir akış şeması son derece yerinde olur. Mesela;



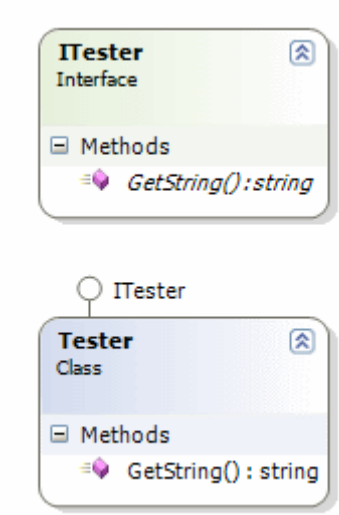
öncelikli olarak başka servisler ile uyumlu bir şekilde konuşulup konuşulmayacağına karar verilerek başlayan bu iş akışının farklı versiyonlarıda bulunmaktadır. Ancak **Juval Löwy ve Steve Resnic**' in önerdiği örnek karar verme adımlarından biriside bu çizelgedir. Karar vermede önemli olan kriterlerden biriside elbetteki **Performans**' tır. çoğunlukla gerçek hayat ortamlarında, yazılan uygulamaların gerçek değeri performansları ile ölçülmektedir. Bu durumda akla gelen sorulardan ilki vakaya göre hangi bağlayıcı tipin en iyi performansı verdiğidir. İşte yazımızın bu bölümünden itibaren 4 temel ve sık kullanılan bağlayıcı tip arasındaki **saniye başına düşen çağrı(Calls Per Seconds)** sayılarını test edeceğimiz örnek bir senaryo üzerinde durmaya çalışacağız.
Vakamızda **NetTcpBinding**, **WsHttpBinding**, **BasicHttpBinding** ve **NetNamedPipeBinding** bağlayıcı tipleri ele alınmaktadır.

Bilindiği üzere **NetTcpBinding** bağlayıcı tipi çoğunlukla **intranet** tabanlı sistemlerde yüksek performansı ile göz önüne çıkmaktadır. Ne varki **WS** destekli servisler ile konuşulması gereken vakalarda **WsHttpBinding** bağlayıcı tipi sıkılıkla değerlendirilmektedir. Tüm bunların yanında özellikle **WS Basic Profile 1.1** ve öncesi yazılmış **Web Servisleri** ile olan haberleşmede **BasicHttpBinding** biçilmiş kaftandır. Elbette öyle vakalar vardırki aynı makine üzerinde koşmakta olan servislerin bir birleriyle haberleşmesi söz konusudur. çok doğal olarak böyle bir durumda **NetNamedPipeBinding** tipi ön tarafa çıkmaktadır.

NOT : .Net Framework 3.0 *ile birlikte, WCF tarafında geliştirilen servislerinin performans değerlerinin ölçülebilmesi amacıyla,* ***ServiceModelEndPoint, ServiceModelOperation, ServiceModelService*** *isimli* ***performans nesneleri(Performance Objects)*** *gelmektedir. Bu değerler* ***Performance Monitor*** *aracı yardımıyla ele alınabilir yada kod tarafından değerlendirilebilir.*

Dilerseniz örnek üzerinden devam edelim. öncelikli olarak kendimize bir adet **WCF Servis Library** geliştiriyor olacağız. *(Söz konusu servis kütüphanesi* ***Visual Studio***

2008 kullanılarak ***.Net Framework 3.5*** *şablonunda geliştirilmiştir.)* Servis tarafına sunulacak olan **sözleşme(Contract)** ve uygulayıcı tipin içeriği aşağıdaki gibidir.



ITester arayüzü(Interface);

```
using System;
using System.ServiceModel;

namespace ProcessLib
{
   [ServiceContract]
   public interface ITester
   {
      [OperationContract]
      string GetString();
   }
}
```

Sözleşmeyi uygulayan Tester sınıf(Class);

```
using System;

namespace ProcessLib
{
   public class Tester
      :ITester
   {
      #region ITester Members

      public string GetString()
      {
```

```
            return "".PadRight(1000, 'Q');
        }

        #endregion
    }
}
```

Servis sözleşmesinin sunduğu operasyon sadece **string** döndüren bir metoda sahiptir. Test sırasında bu metod kullanılaraktan istemci tarafından yapılan yüklü operasyon çağrılarının sonuçları irdelenmeye çalışılacaktır. Artık **Host** uygulamanın yazılmasına başlanabilir. Olayların basit şekilde analiz edilebilmesi için **Host** uygulama **Console** olarak tasarlanmaktadır. Bu uygulama kendi içerisinden 4 farklı **EndPoint** sunmaktadır. Bunlar vakada yer alan bağlayıcı tiplerin her biri için ele alınmaktadır. Bu amaçla konfigurasyon dosyasının içeriği aşağıdaki gibi geliştirilebilir. *(Bu aşamaya gelmeden önce Host uygulamaya **System.ServiceModel.dll** ve WCF Sınıf Kütüphanesine ait assembly nesnelerinin eklenmesi gerektiğinide hatırlayalım.)*

Host uygulama tarafındaki konfigurasyon (App.Config) içeriği;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <diagnostics performanceCounters="All" />
    <services>
      <service name="ProcessLib.Tester">
        <endpoint address="net.tcp://localhost:9000/TesterService"
binding="netTcpBinding" bindingConfiguration="" name="TesterServiceTcpEndPoint"
contract="ProcessLib.ITester" />
        <endpoint address="http://localhost:9001/TesterService"
binding="basicHttpBinding" name="TesterServiceHttpEndPoint"
contract="ProcessLib.ITester" />
        <endpoint address="http://localhost:9002/TesterService"
binding="wsHttpBinding" bindingConfiguration=""
name="TesterServiceWsHttpEndPoint" contract="ProcessLib.ITester" />
        <endpoint address="net.pipe://localhost/TesterService"
binding="netNamedPipeBinding" bindingConfiguration=""
name="TesterServicePipeEndPoint" contract="ProcessLib.ITester" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

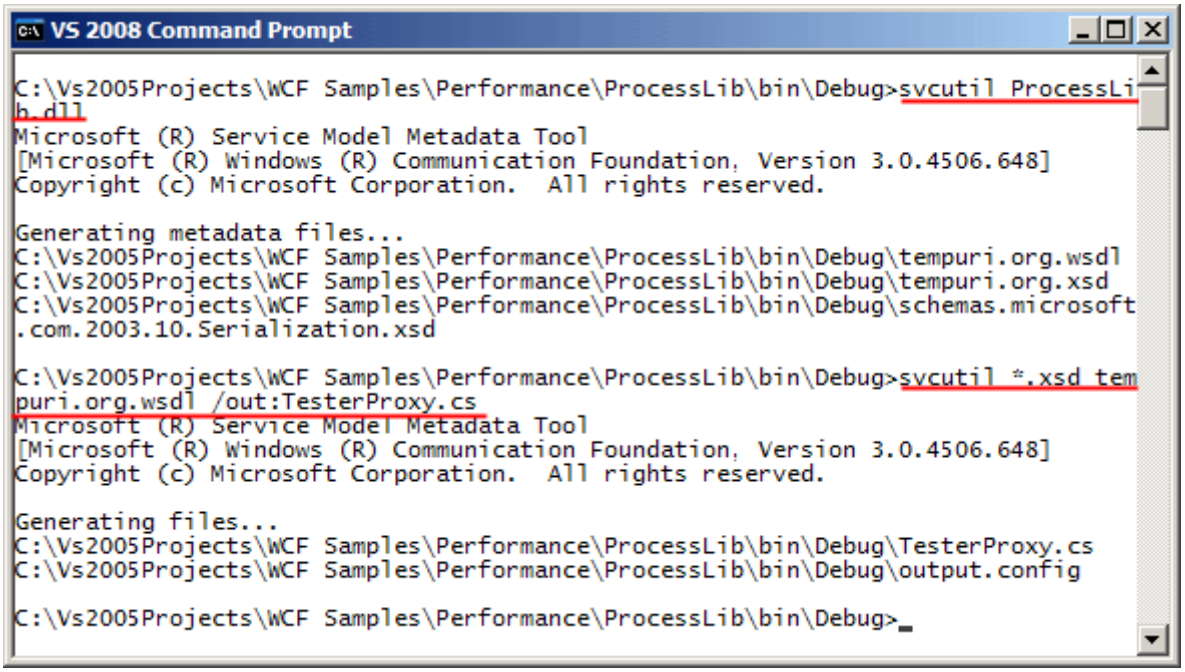
Konfigurasyon dosyasında en çok dikkat çeken noktalardan biriside **diagnostics** isimli elementtir. Bu elementin içerisinde yer alan **performanceCounters** niteliğine

atanan **All** değeri sayesinde **Performance Monitor** üzerinden ölçümler yapılabilmektedir. Bu sebepten servis tarafındaki konfigurasyon dosyasında mutlaka bildirilmelidir.

Host Uygulama Kodları;

```
using System;
using System.ServiceModel;
using ProcessLib;

namespace ServerApp
{
   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(Tester));
         host.Open();
         Console.WriteLine("Servis dinlemede\nKapatmak için bir tuşa basın");
         Console.ReadLine();
         host.Close();
      }
   }
}
```

Host uygulama 4 farklı **EndPoint** üzerinden hizmet vermektedir. Tüm **EndPoint**'ler **Tester** isimli servis tipini kullanmaktadır. Buna göre 4 farklı tipte istemcinin bu servis üzerinden hizmet alması mümkündür. İstemci tarafını geliştirmeden önce gerekli **proxy** tipinin üretimi için **svcutil** aracından aşağıdaki ekran görüntüsünde yer aldığı gibi yararlanılabilir.



Artık istemci uygulamanın geliştirilmesine başlanabilir. Buradada örneği basit bir şekilde ele alabilmek için bir **Console** uygulaması göz önüne alınmaktadır. Yine önemli olan nokta konfigurasyon dosyasının içeriğidir. Dört **EndPoint** için testler aynı istemci üzerinden yapılacağından konfigurasyon içeriği aşağıdaki gibi tasarlanmıştır.

İstemci uygulama konfigurasyon(App.config) içeriği;

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <system.serviceModel>
      <client>
         <endpoint address="http://localhost:9001/TesterService"
binding="basicHttpBinding" contract="ITester" name="ClientHttpEndPoint" />
         <endpoint address="net.tcp://localhost:9000/TesterService"
binding="netTcpBinding" bindingConfiguration="" contract="ITester"
name="ClientTcpEndPoint" />
         <endpoint address="http://localhost:9002/TesterService"
binding="wsHttpBinding" bindingConfiguration="" contract="ITester"
name="ClientWsHttpEndPoint" />
         <endpoint address="net.pipe://localhost/TesterService"
binding="netNamedPipeBinding" bindingConfiguration="" contract="ITester"
name="ClientPipeEndPoint" />
      </client>
   </system.serviceModel>
</configuration>
```

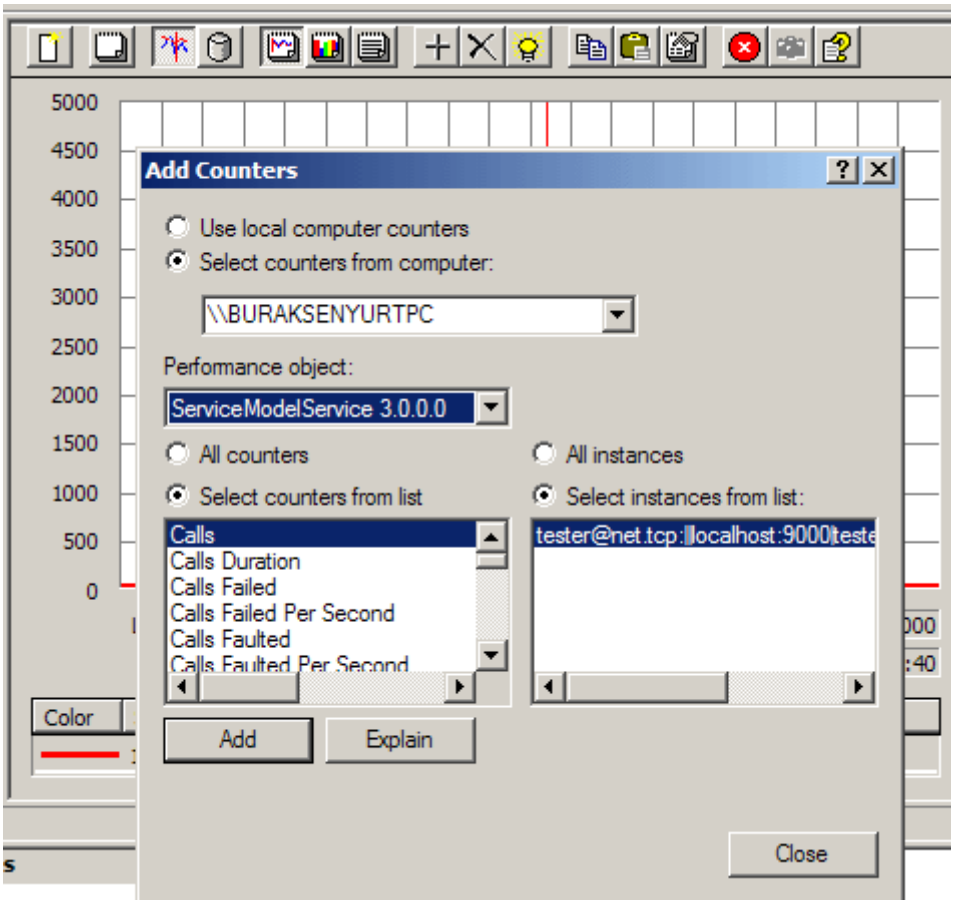
İstemci uygulamanın kodları;

```
using System;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Bağlayıcı tipi seçiniz...");
         Console.WriteLine("NetTcpBinding için 1");
         Console.WriteLine("BasicHttpBinding için 2");
         Console.WriteLine("WsHttpBinding için 3");
         Console.WriteLine("NetNamedPipeBinding için 4");

         string secim=Console.ReadLine();
         Console.WriteLine("Sayaç Değerini Giriniz");
         int sayac;
         if (!Int32.TryParse(Console.ReadLine(), out sayac))
            sayac = 100000;

         switch (secim){
            case "1":
               TestiBaslat("ClientTcpEndPoint",sayac);
               break;
            case "2":
               TestiBaslat("ClientHttpEndPoint", sayac);
               break;
            case "3":
               TestiBaslat("ClientWsHttpEndPoint", sayac);
               break;
            case "4":
               TestiBaslat("ClientPipeEndPoint", sayac);
               break;
            default:
               TestiBaslat("ClientTcpEndPoint", sayac);
               break;
         }
      }

      private static void TestiBaslat(string baglayici,int testSayisi)
      {
         Console.WriteLine("Test Başladı\nDeneme Sayisi {0}\nSeçilen EndPoint
{1}",testSayisi,baglayici);
         TesterClient client = new TesterClient(baglayici);
         for (int i = 0; i < testSayisi; i++)
```

```
        {
           string result = client.GetString();
        }
        Console.WriteLine("Test tamamlandı...");
        client.Close();
      }
   }
}
```

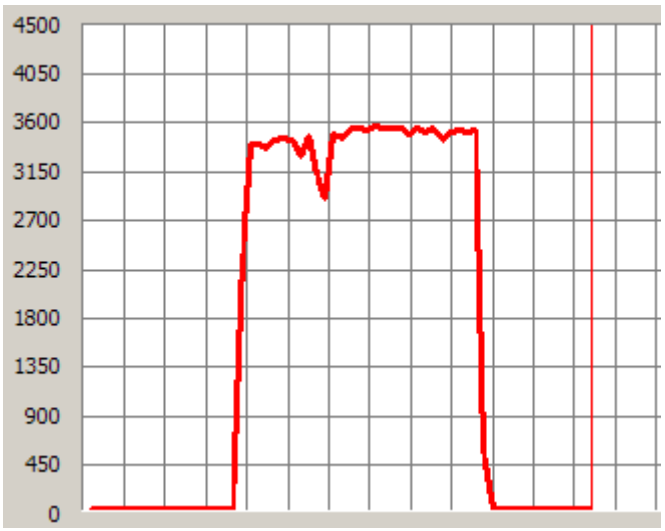
İstemci uygulama **GetString** isimli metodu sembolik olarak girilen değer kadar *(varsayılan olarak **100000(Yüzbin)** kere)* çağırmaktadır. Bu bize test için yeterli süreyide vermektedir. Diğer taraftan elbetteki bu test değerleri makinenin özelliklerine göre farklılık gösterecektir. Diğer taraftan bağlayıcı tipler arasındada belirgin farklılıklar olduğu gözlenecektir. Bunun en büyük nedenlerinden birisi ilgili bağlayıcı tipin arka tarafta kullandığı **kodlama(Encoding)** ve **iletişim protokolüdür(Transport Protocol)**. *(Makalede yer alan örneğin testlerinin yapıldığı makine 1 Gb RAM kapasiteli olup, Intel Centrino işlemcilidir. Ayrıca Windows XP Service Pack 2 işletim sistemine sahiptir. Testlerde yer alan istemci ve sunucu uygulama aynı makine üzerinde koşmaktadır.)*

Artık testlere başlanabilir. **Performance Counter** üzerinden çalışma zamanı ölçümlerinin yapılabilmesi için servis uygulamasının çalışıyor olması gerekmektedir. Sonrasında ise **Performance Monitor** kullanılarak ölçümler yapılabilir. **Performance Monitor** uygulamasına komut satırından **perfmon** yazılarak ulaşılabilir. Servis uygulaması başlatıldığında **Counter** olarak eklenebilecek seçeneklerde ilgili servis uygulamasına ait örnek görülebilecektir.

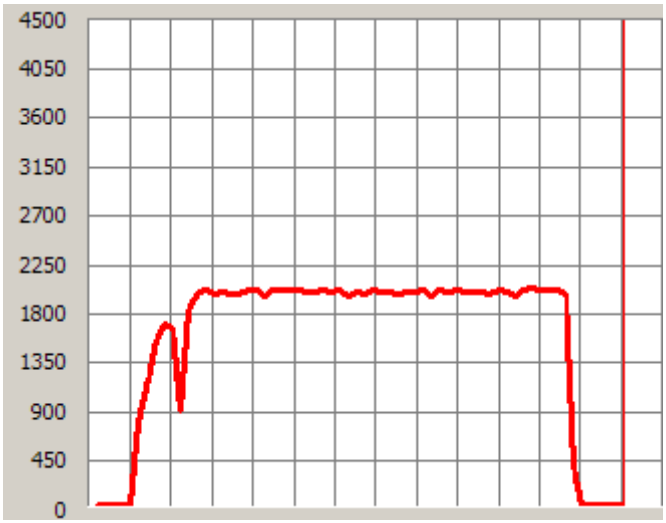


Biz testlerimizde **ServiceModeService 3.0.0.0** Performans nesnesinin **Calls Per Second** isimli sayacını kullanıyor olacağız. Bu sayaç servise gelen saniye başına operasyon çağrılarını göstermekte olup bağlayıcı tiplerin alt yapı hazırlıkları nedeni ile ne kadar yavaş veya ne kadar hızlı olduklarını göstermektedir. **Performance Monitor** sonuçları test programında kullanılan her bağlayıcı tip için aşağıdaki ekran çıktılarında olduğu gibidir.

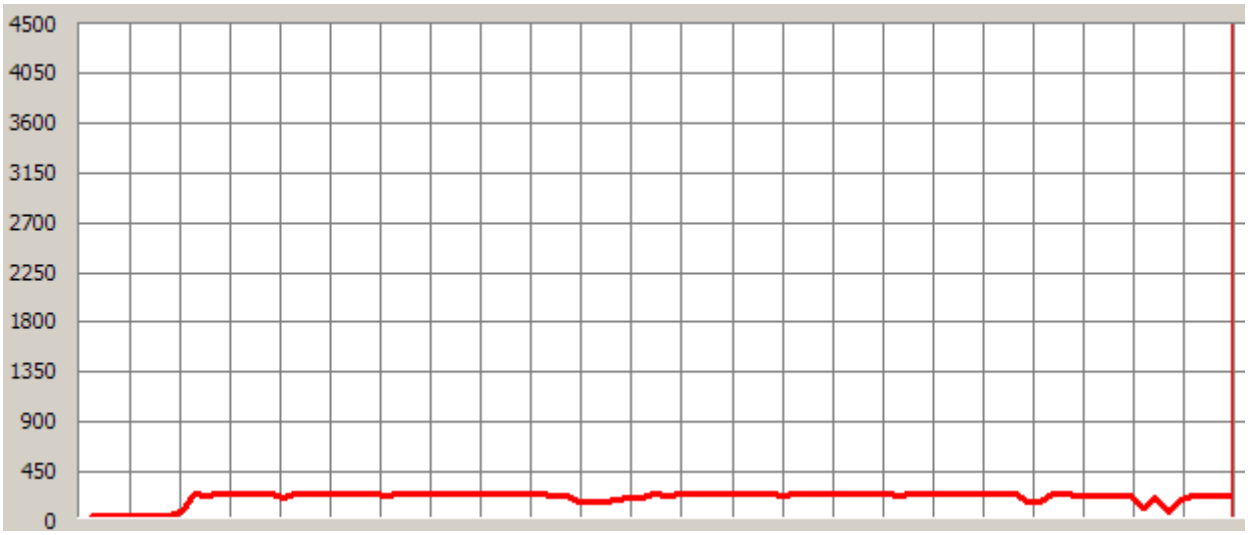
NetTcpBinding için Calls Per Second sonuçları;



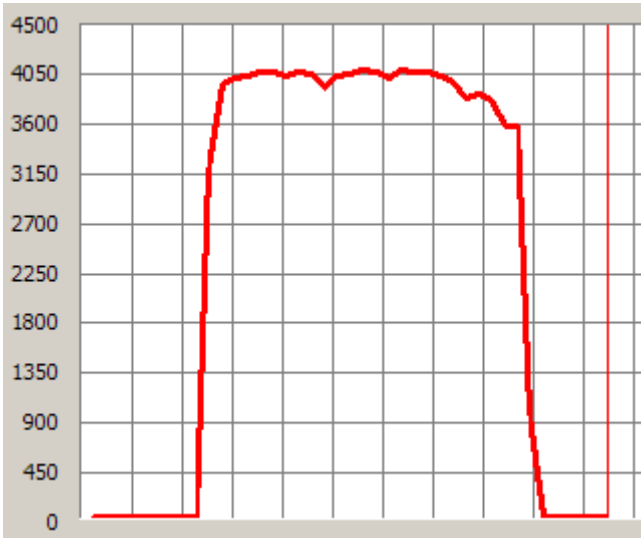
BasicHttpBinding için Calls Per Second sonuçları;



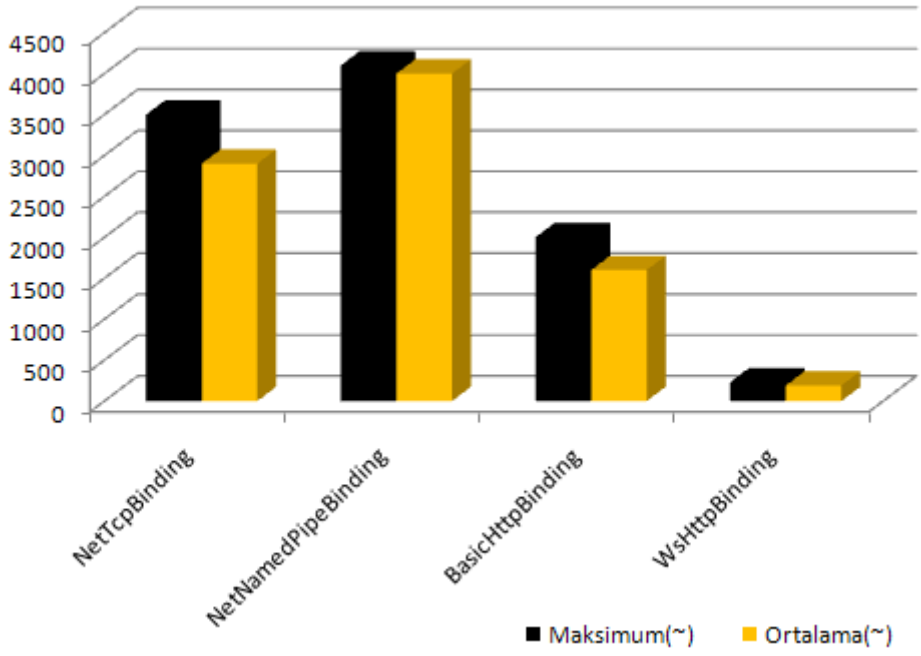
WsHttpBinding için Calls Per Second sonuçları;



NetNamedPipeBinding için Calls Per Second sonuçları;



Tüm bunları bir arada değerlendirildiğinde, **saniye başına düşen operasyon çağrıları** için aşağıdaki grafikte görülen sonuçlar ortaya çıkmaktadır. Bu grafikte tepe noktalarının yaklaşık maksimum ve ortalama değerleri ele alınmaktadır. Bu en azından tahmini olarak, bu dört bağlayıcı tip arasındaki performans farklılıkları hakkında fikir vermektedir.



çok doğal olarak aynı makine üzerinde çalışan servisler arası haberleşmede tercih edilen **NetNamedPipeBinding** için en yüksek değerler ortaya çıkmaktadır. Diğer taraftan özellikle**TCP** ve **Binary** serileştirme kullanan **NetTcpBinding** değerleride oldukça yüksektir. Bunların yanında **WsHttpBinding** gerçekten çok düşük değerler ile göze çarpmaktadır. Bunun en büyük nedeni **WS** standartlarına göre mesajların hazırlanması sırasında geçen süre kayıplarıdır. Tabiki bu kriter tek başına yeterli değildir. Söz gelimi servisin **Host** edildiği sunucunun **operasyon başına maliyet(Cost Per Operation)** gibi performans değerleride göz önüne alınabilir. Bu ve daha pek çok performans testi yapılabilir.

Buraya kadar anlatılanlar göz önüne alındığında sadece bağlayıcı tipler arasındaki farklılıkları analiz etmek için basit olarak nasıl bir yol izlenebileceği ele alınmıştır. Bunun dışında WCF ile geliştirilen servislerin Xml Web Serviceleri, .Net Remoting veya COM+ ile geliştirilen modellere göre belirgin performans farklılıkları olduğu bilinmektedir. Bu konu ile ilişkili daha detaylı bilgiye http://msdn.microsoft.com/en-us/library/bb310550.aspx ulaşabilirsiniz. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

Performance.rar (58,27 kb)

WCF ile WF Entegrasyonu - 2 (2008-04-23T02:32:00)

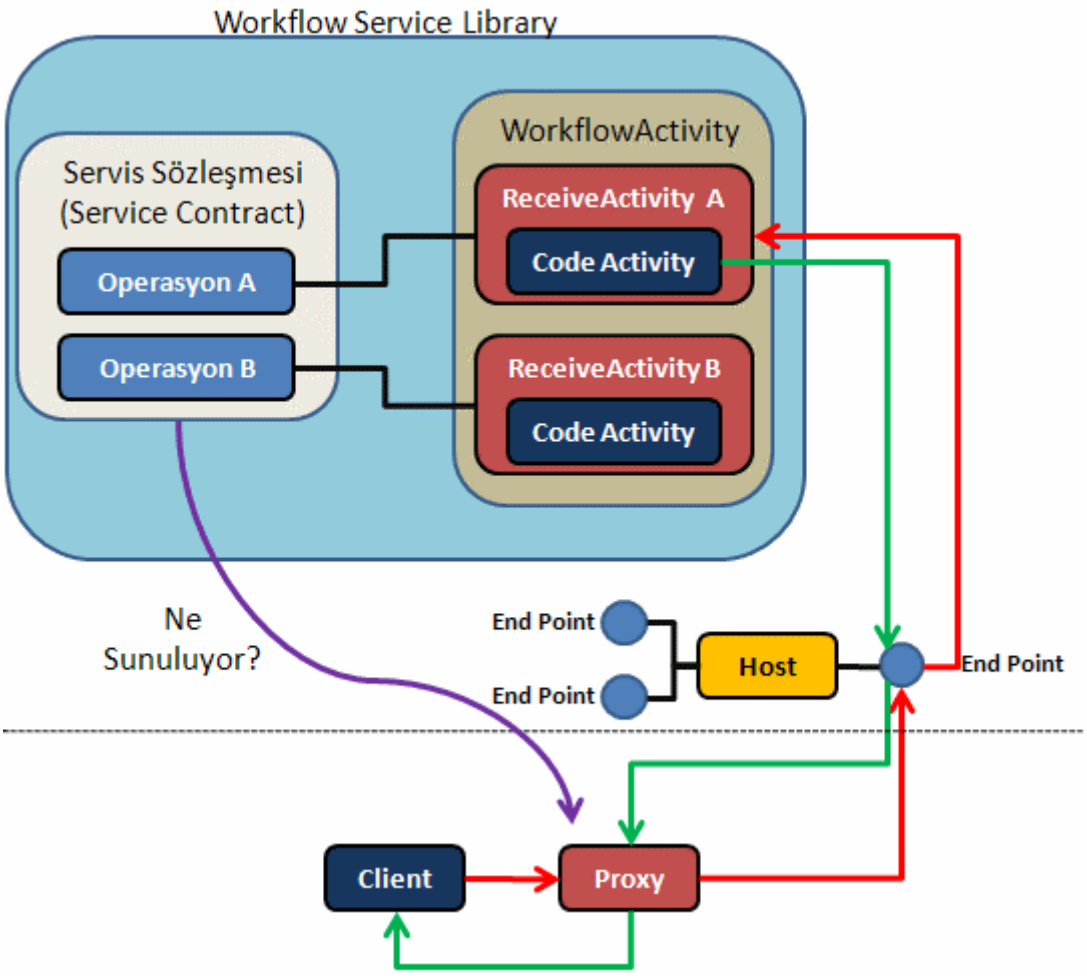
wcf,wf,

Bir önceki yazımızda **WCF(Windows Communication Foundation)** servislerinin, **WF(Windows WorkFlow)** uygulamaları içerisinden nasıl çağırıldığını incelemiştik. Bu yazımızda ise tam tersine, bir **Workflow** örneğinin servis olarak nasıl sunulabileceğini analiz ediyor olacağız. Bazı durumlarda kod akışlarının birer servis olarak istemcilere sunulması gerekebilir. Burada söz konusu kod akışlarının **Servis Yönelimli Mimarinin(Service Oriented Architecture)** imkanlarından yararlanıyor olması isteği ön plana çıkmatadır. çok doğal olarak servis gibi yayınlanan **akış tipleri(Workflow Instance)**, istemci ile olan mesajlaşmalarında **SOA** temelli olanakları kullanabilir hale gelmektedir. Bu noktada **WCF** ile **WF** entegrasyonu göz önüne alınmalıdır.

WCF servisleri **WF** uygulamaları içerisinden çağırılırken ağırlıklı olarak **.Net Framework 3.5** ile birlikte gelen **SendActivity** bileşeni kullanılmaktadır. **WF** örneklerinin servis olarak yayınlamasında ise başrol oyuncusu yine .Net Framework 3.5 ile birlikte gelen **ReceiveActivity** isimli activity bileşenidir. özellikle **Visual Studio 2008** kullanılarak servis destekli **Workflow**kütüphaneleri kolay bir şekilde geliştirilebilmektedir. Bu amaçla **Visual Studio 2008** ortamına **Sequential Workflow Service Library** ve **State Machine Workflow Service Library** proje şablonları eklenmiştir. Tabi çok doğal olarak geliştirilen iş akışı servislerinin bir uygulama tarafından barındırılması(Hosting) ve yayınlanmasıda gerekmektedir. **Host** seçenekleri **WF** servisleri içinde aynıdır. **IIS** üzerinde, **WAS(Windows Activation Service)** yardımıyla veya **Self-Hosting** seçeneklerine göre barındırma ve yayınlama yapılabilir. önemli olan noktalardan birisie normal WCF servislerinden farklı olaraktan **Host** çalışma ortamını **WorkflowServiceHost** tipinin yönetmesidir.

WF uygulamasının servis olarak yayınlanması için istemcilere bir **sözleşme(Contract)** bildirimi yapılması gerekmektedir. çok doğal olarak bu **sözleşme(Contract)** bir **arayüz(Interface)**olarak tanımlanmalıdır. Arayüz içerisinde yer alan operasyonlar dışarıya **ReceiveActivity** bileşeni ile sunulabilirler. Buna göre sözleşme içerisinde tanımlanan her operasyon için*(**OperationContract** niteliği ile imzalanmış metodlar)* birer **ReceiveActivity** oluşturulmalıdır. Bu noktada karşılaşılan önemli sorulardan biriside, istemcinin bu operasyona nasıl parametre göndereceği veya cevap alacağıdır. Nitekim bir WCF kütüphanesinde çoğunlukla asıl işi yapan bileşen, sözleşme tipinin uygulandığı bir sınıftır(Class). WF açısından bakıldığında ise bu görevi **Workflow** sınıfı içerisinde tanımlanan **özellikler(Properties)** yada **alanlar(fields)** üstlenmektedir. Bir başka deyişle, **ReceiveActivity** irtibatta olduğu operasyon için gerekli parametreler ile haberleşmek adına söz konusu özellik veya alanlardan yararlanır. Dolayısıyla asıl iş **ReceiveActivity** içerisinden yapılmalıdır.**ReceiveActivity** bileşeninin **composite** bir bileşen olarak tanımlanmasının sebebide budur. **ReceiveActivity** içerisine örneğin **CodeActivity** gibi bileşenler dahil edilerek operasyonun asıl işinin yapılacağı kod

bloklarının işletilmesi sağlanabilir. İstemci açısından olaya bakıldığında yine bir **proxy** nesnesinin servis ile olan haberleşmeyi sağladığı ortadadır. Aslında aşağıdaki şekil durumu biraz daha kolay bir şekilde açıklamaktadır.

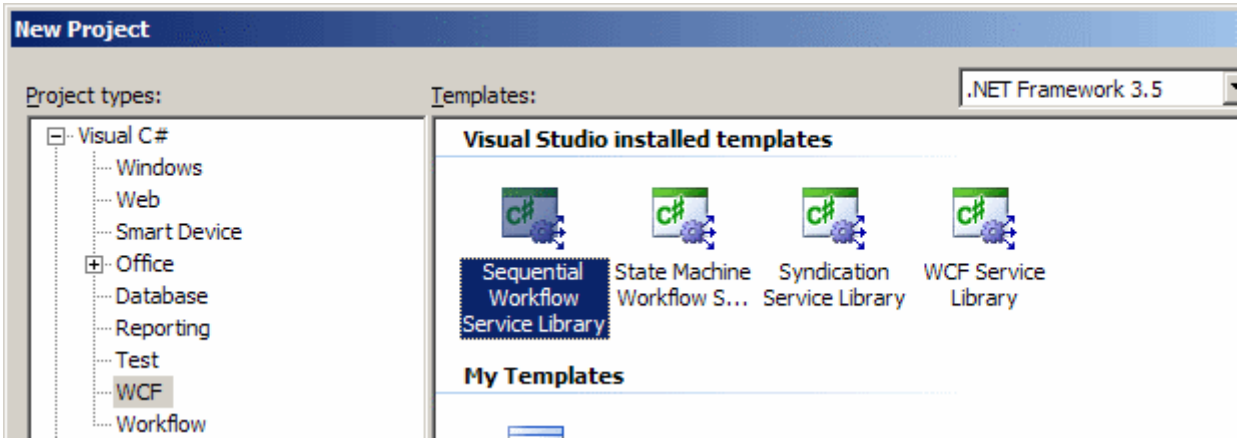


Şekle göre **WorkflowActivity** sınıfının **ReceiveActivity** bileşenleri, dışarıya servis üzerinden sunulan operasyonlar ile ilişkili talep alma ve cevap gönderme işlemlerini üstlenmektedir. çok doğal olarak servisi bir **EndPoint** üzerinden dışarıya sunmak gerekmektedir. **EndPoint** tanımında dikkat edilmesi gereken noktalardan biriside, **bağlayıcı tipin(Binding Type)basicHttpContextBinding**, **netTcpContextBinding** yada **wsHttpContextBinding** bileşenlerinden birisi olmaslıdır. Bu bağlayıcı tiplerin ortak özelliği servis destekli iş akışları(Service-Enabled Workflow) ile haberleşilebilmesini sağlamalarıdır. Ancak istenirse özel bağlayıcılara **ContextBindingElement** tipi uygulanarak workflow destekli hale getirilmeleride sağlanabilir. İstemci uygulama çok doğal olarak servis ile olan haberleşme sırasında proxy sınıfından yararlanmak durumundadır.

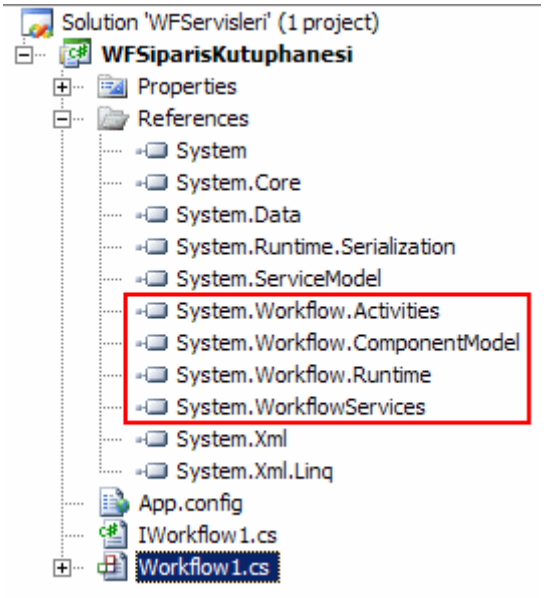
Not : *WF ve* ***WCF*** *entegrasyonun bir sonucu olarak istemciler bir* ***Workflow*** *aktivitesini servis bazlı olacak şekilde çağırıp kullanabilmektedir. Böylece istemciler bir kod akışı sürecini****servis tabanlı*** *düşünerek ele alabilirler.*
Buna ***JSON(JavaScriptObjectNotation)*** *gibi mesajlaşma desteklerinin eklenebileceği*

*düşünüldüğünde bir **Workflow** örneğinin herhangibir platform üzerinden kullanılabilmeside mümkün hale gelmektedir. Bu WCF tanımında yer alan "her hangibir CLR tipinin servis olarak yayınlanabilmesi" ilkesinin bir sonucu olarak görülebilir.*

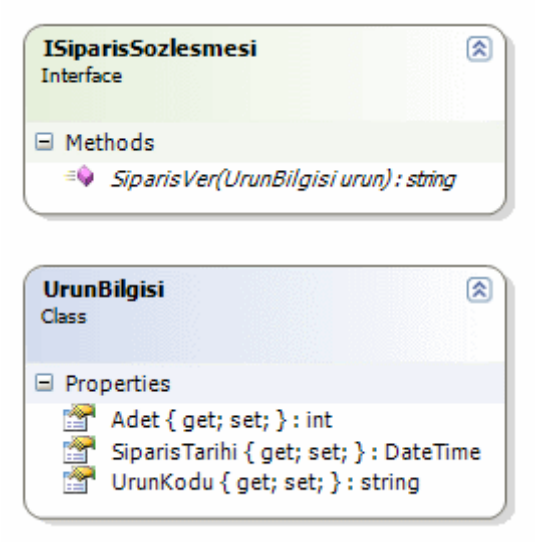
Bu kadar karmaşık ve teorik bilgiden sonra örnek bir uygulama üzerinden hareket ederek **WF** uygulamaları içerisinde servis yayınlamasının nasıl yapılabileceğini adım adım incelemeye çalışalım. İşe ilk olarak servis sözleşmesini ve operasyonlar için gerekli aktivite bileşenlerini içerecek olan kütüphaneyi tasarlamak ile başlamak doğru olacaktır. Bu amaçla **Visual Studio 2008** ortamında bir **Sequential Workflow Service Library** projesi açtığımızı düşünelim.*(Bu şablon **New Project->WCF** sekmesi altında yer almaktadır.)*



Söz konusu **servis kütüphanesine(WFSiparisKutuphanesi)** bakıldığında ilk dikkati çeken nokta referanslar kısmına **Workflow** desteği için eklenen **assembly**' lardır.



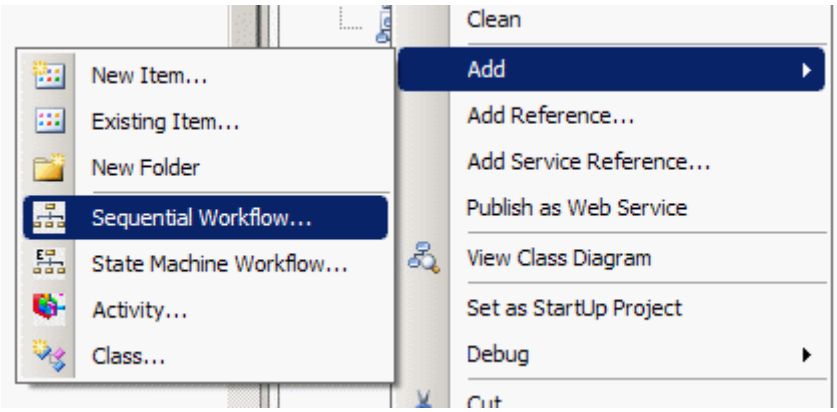
Bu noktada şablon olarak getirilen **IWorkflow1.cs, Workflow1.cs** ve **App.config** isimli dosyalar silinmiştir. örnekte kullanılan servis sözleşmesi içeriği ise aşağıdaki sınıf diagramında görüldüğü gibidir.



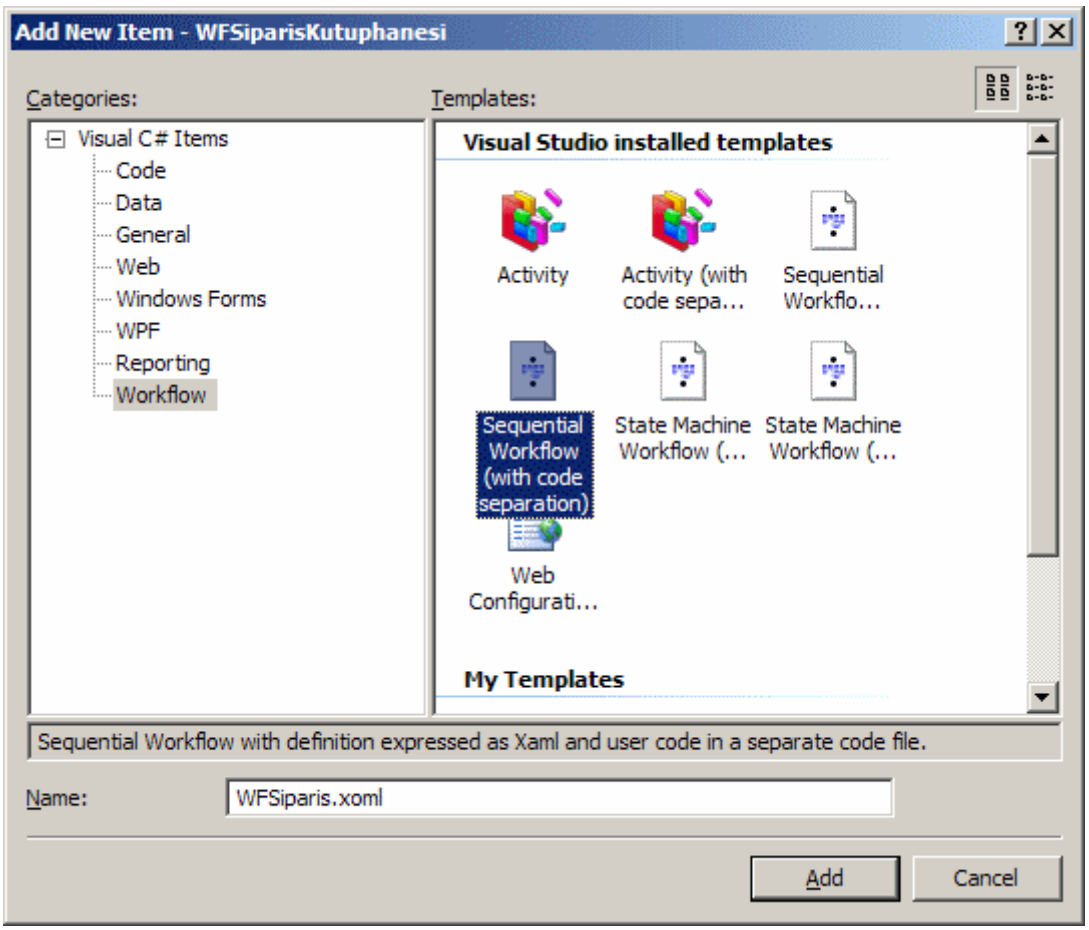

```
using System;
using System.ServiceModel;
using System.Runtime.Serialization;

namespace WFSiparisKutuphanesi
{
   [ServiceContract(Namespace = "http://www.bsenyurt.com/UrunSiparisServisi", Name =
"UrunSiparisServisi")]
   public interface ISiparisSozlesmesi
   {
      [OperationContract]
      string SiparisVer(UrunBilgisi urun);
   }

   [DataContract]
   public class UrunBilgisi
   {
      [DataMember]
      public string UrunKodu { get; set; }
      [DataMember]
      public int Adet { get; set; }
      [DataMember]
      public DateTime SiparisTarihi { get; set; }
   }
}
```

ISiparisSozlesmesi isimi **servis sözleşmesi(Service Contract) SiparisVer** isimli tek bir operayon sunmaktadır. **SiparisVer** metodu parametre olarak **UrunBilgisi** sınıfına ait bir nesne örneği almaktadır. Bu nesneye ait veri, çalışma zamanında istemci tarafından servise doğru geleceğinden serileştirilebilir olması gerekmektedir. Bu sebepten doğal olarak **veri sözleşmesi(Data Contract)** olacak şekilde tasarlanmıştır. SiparisVer isimli operasyon aynı

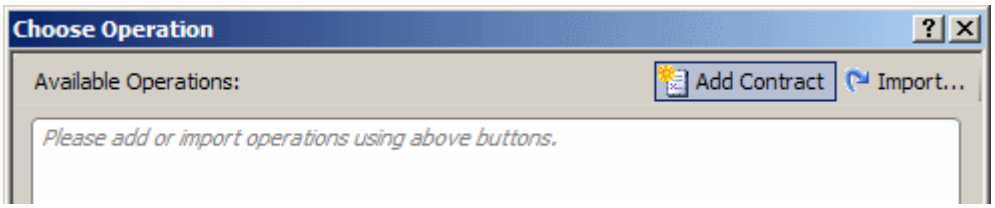
zamanda geriye **string** tipinden bir değerde döndürmektedir. Böylece dışarıya sunulacak olan servis sözleşmesi tanımlanmıştır. Artık bu sözleşmeyi kullanacak olan aktivite sınıfının yazılması gerekmektedir. örnekte bunun için bir adet **Sequential Workflow Activity** sınıfı ele alınacaktır. Bunun için projeye **Add->Sequential Workflow** seçeneği ile yeni bir akış sınıfı eklenmelidir.



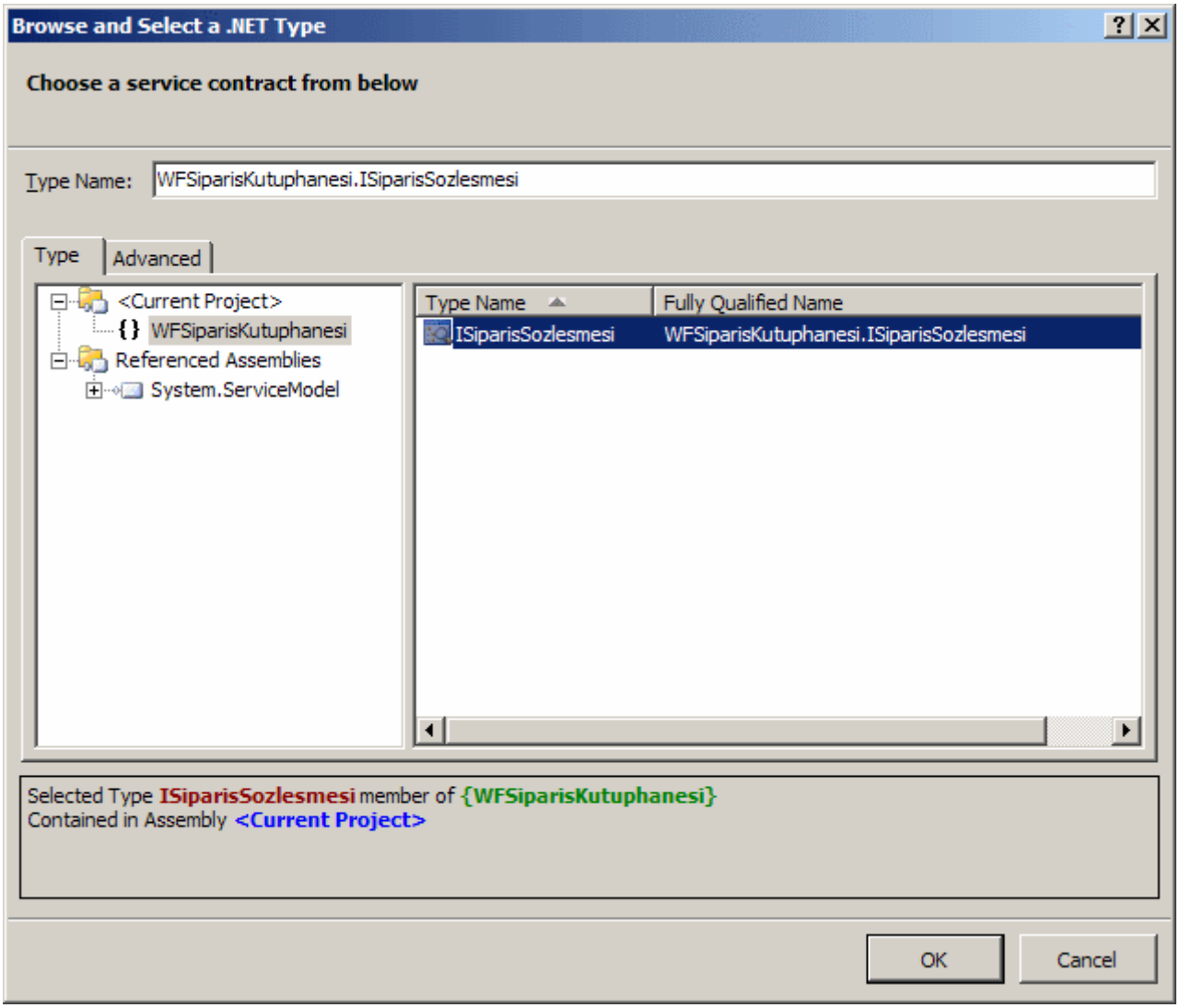
Bu adımda **Sequential Workflow(with code seperation)** seçeneği işaretlenerek devam edilebilir. Böylece dizayn tarafı ile kodu birbirinden ayrı tutacak bir akış tipi oluşturulacaktır.



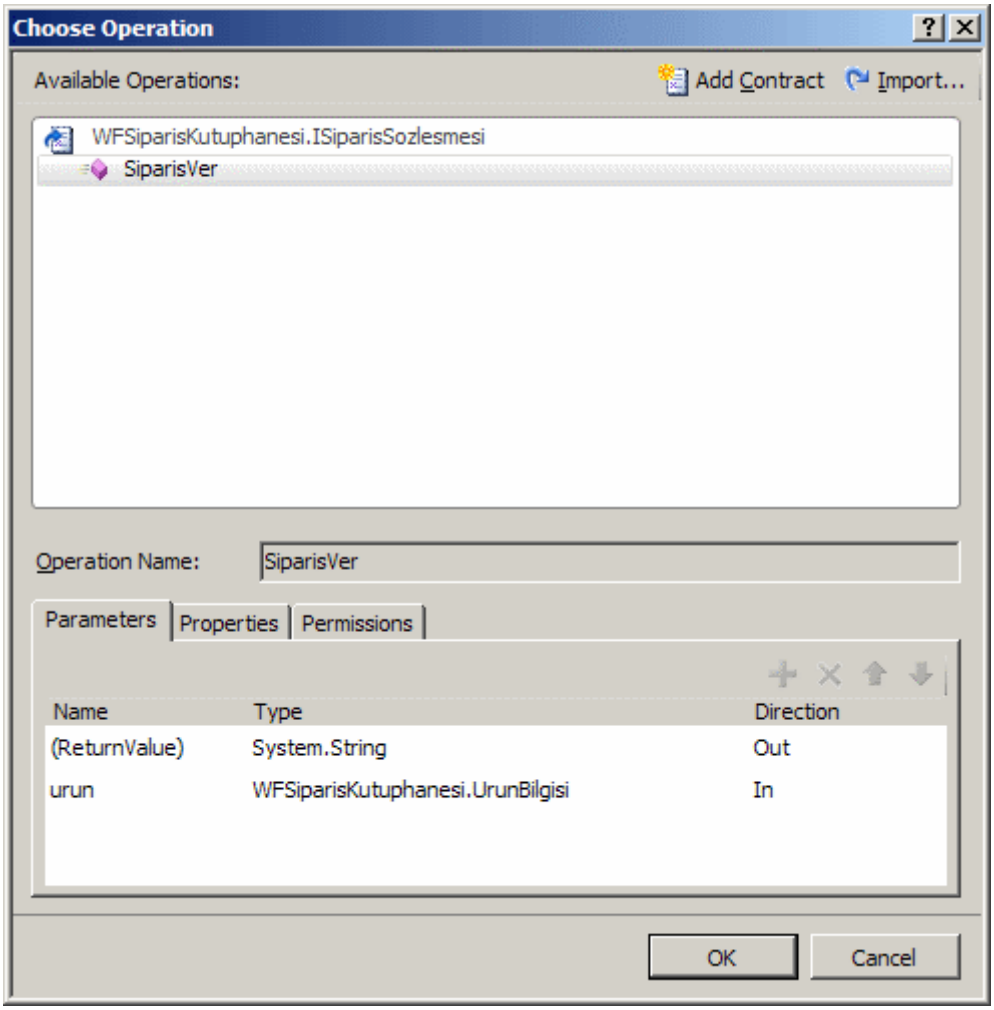
WFSiparis.xoml (XOML-eXtensible Object Markup Language) tasarım zamanında gerekli aktivite tiplerini barındıracaktır. **WFSiparis** isimli sınıf **SequentialWorkflowActivity** tipinden türemektedir. Servis sözleşmesi içerisinde yer alan **SiparisVer** metodu bu akış nesnesi içerisinde **ReceiveActivity** bileşeni tarafından ele alınmalıdır. Bununla birlikte **SiparisVer** metodunun**UrunBilgisi** ve döndüreceği değer için birer **özellik/alan(Property/Field)** içermesi gerekmektedir. öncelikli olarak bir **ReceiveActivity** bileşeni tasarım ekranına sürüklenip bırakılmalıdır. **ReceiveActivity** bileşeni **SendActivity** bileşenine benzer olaraktan **ServiceOperationInfo** özelliğini içermektedir. çok doğal olarak bu özellikten yararlanılarak hangi operasyonun ele alınacağı ve kullanılacak*(yada otomatik olarak üretilecek)* olan parametreler belirlenmelidir. **ServiceOperationInfo** özelliğinde yer alan üç nokta düğmesine basıldıktan sonra aşağıdaki arayüz ile karşılaşılır.



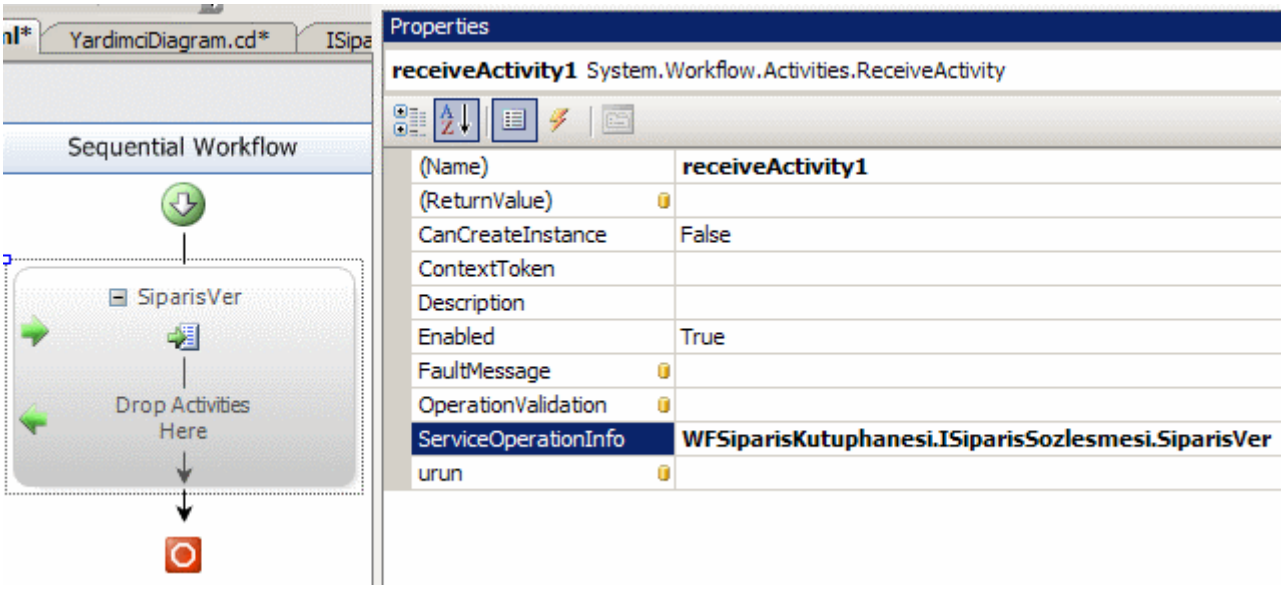
Burada yine **Import** seçeneği kullanılarak aşağıdaki ekran görüntüsünde olduğu gibi ilgili servis sözleşmesinin seçilmesi gerekmektedir.



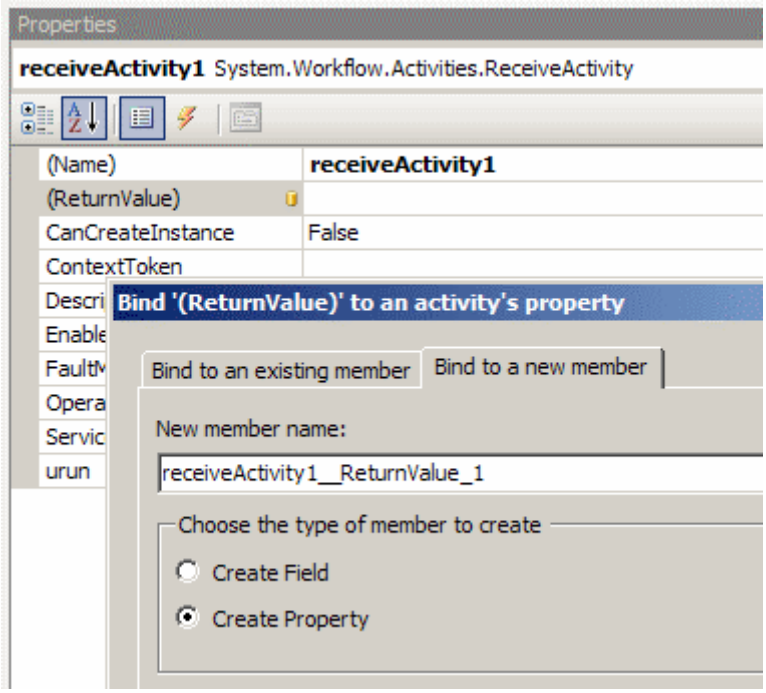
çok doğal olarak **ISiparisSozlesmesi** otomatik olarak gelecektir. Bu noktada **Workflow** içerisinde birden fazla servis sözleşmesi tutulabileceği de unutulmamalıdır. Bu seçim işlemini takiben aşağıdaki ekran görüntüsü elde edilir ve artık operasyon seçimi ve bunun için gerekli sınıf özelliklerinin oluşturulması adımına geçilebilir.



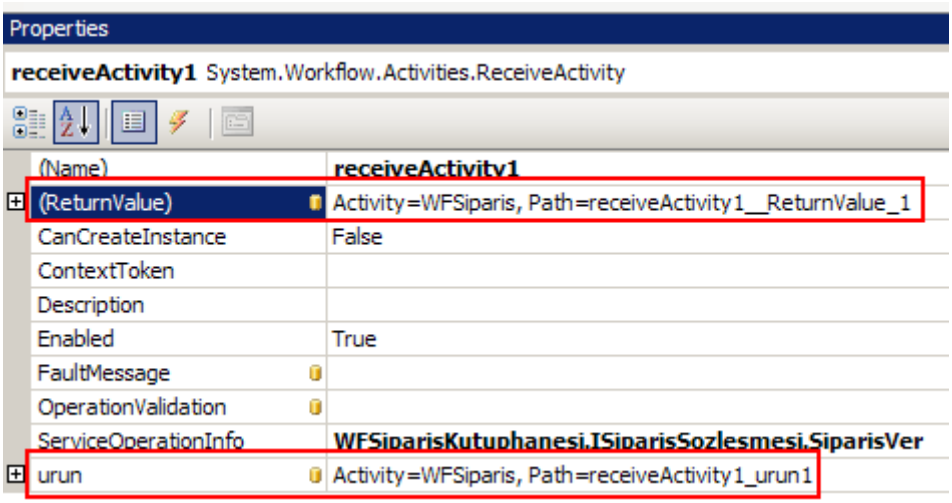
Dikkat edileceği üzere **Parameters** kısmında **String** tipinden yönü **Out** olan ve **UrunBilgisi** tipinden yönü **In** olan birer parametre görülmektedir. Bu parametrelerin akış içerisinde ele alınması için karşılığı olan özelliklerin veya alanların sınıf içerisine ya manuel yada otomatik olarak dahil edilmesi şarttır. Sonuç olarak aşağıdaki ekran görüntüsü ile karşılaşılacaktır.



Burada **ReturnValue** ve **urun** özellikleri için otomatik özellik ürettirilmesi sağlanabilir. Söz gelimi aşağıdaki ekran görüntüsünde örnek olarak **SiparisVer** metodunun geri dönüş tipi için otomatik özellik ürettirilmesinin nasıl sağlandığı gösterilmektedir.



Aynı işlem urun özelliği içinde yapıldıktan sonra **receiveActivity1** için son durum aşağıdaki gibi olacaktır**.**

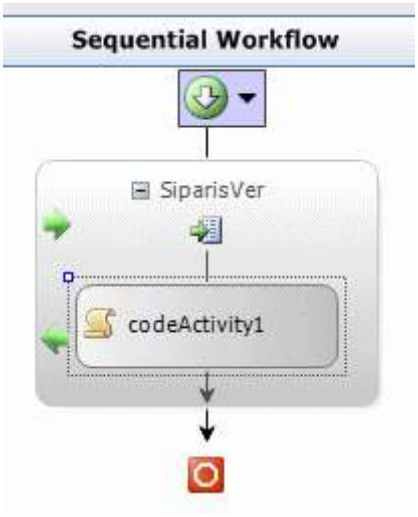


çok doğal olarak bu yapılan değişiklikler sonrasında **WFServis** isimli sınıfın içeriği aşağıdaki gibi değişecektir.

```
namespace WFSiparisKutuphanesi
{
   public partial class WFSiparis : SequentialWorkflowActivity
   {
```

```
        public static DependencyProperty receiveActivity1_urun1Property =
DependencyProperty.Register("receiveActivity1_urun1",
typeof(WFSiparisKutuphanesi.UrunBilgisi), typeof(WFSiparisKutuphanesi.WFSiparis));
        public static DependencyProperty receiveActivity1__ReturnValue_1Property =
DependencyProperty.Register("receiveActivity1__ReturnValue_1", typeof(System.String),
typeof(WFSiparisKutuphanesi.WFSiparis));

        [DesignerSerializationVisibilityAttribute(DesignerSerializationVisibility.Visible)]
        [BrowsableAttribute(true)]
        [CategoryAttribute("Parameters")]
        public UrunBilgisi receiveActivity1_urun1
        {
            get
            {
                return
((WFSiparisKutuphanesi.UrunBilgisi)(base.GetValue(WFSiparisKutuphanesi.WFSiparis.re
ceiveActivity1_urun1Property)));
            }
            set
            {
                base.SetValue(WFSiparisKutuphanesi.WFSiparis.receiveActivity1_urun1Propert
y, value);
            }
        }

        [DesignerSerializationVisibilityAttribute(DesignerSerializationVisibility.Visible)]
        [BrowsableAttribute(true)]
        [CategoryAttribute("Parameters")]
        public string receiveActivity1__ReturnValue_1
        {
            get
            {
                return
((string)(base.GetValue(WFSiparisKutuphanesi.WFSiparis.receiveActivity1__ReturnValue
_1Property)));
            }
            set
            {
                base.SetValue(WFSiparisKutuphanesi.WFSiparis.receiveActivity1__ReturnValu
e_1Property, value);
            }
        }
    }
}
```

Artık **ReceiveActivity** içerisine istemcilere döndürülecek olan cevap için gerekli **activity** bileşeninin eklenmesi adımına geçilebilir. Sonuç itibariyle istemciler **SiparisVer** metoduna çağrıda bulunduktan sonra bir aktivitenin işletilmesi gerekmektedir. Bu aktivite **ReceiveActivity** içerisinde tanımlanabilir. Söz gelimi **CodeActivity** bu işlem için idealdir. Operasyona istemciden gelen bilgi, aktivite sınıfının **receiveActivity1_urun1** özelliği üzerinden elde edilebilir. Metoddan istemciye döndürülecek olan **string** değer ise **receiveActivity1_ReturnValue1** özelliğine set edilmelidir. Bu atama işlemide CodeActivity bileşeni içerisinde yapılabilir. *(Burada **CodeActivity** kullanılması şart değildir. önemli olan aktivite sınıfı içerisine eklenen özellikler yardımıyla istemciden operasyona gelen parametre bilgilerinin alınabilmesi veya geriye döndürelecek bir sonucun üretilebilmesi ve istemci tarafından ele alınabilmesidir.)* Şimdi **ReceiveActivity** içerisine bir**CodeActivity** eklediğimizi düşünelim.



CodeActivity1 bileşeninin **ExecuteCode** özelliğine örnek olarak **SiparisiIsle** değerini verip kod içeriğini aşağıdaki gibi geliştirdiğimizi düşünelim. *(Şu anda amaç WF servislerinin nasıl yazılacağını görmek olduğundan **Console** tabanlı host ve istemci uygulamalar yazılacaktır. Bu nedenle **CodeActivity** bileşeninin işaret edeceği metod içerisinden o andaki talep bilgilerini değerlendirmek amacıyla Console ekranına bilgi yazdırılmaktadır.)*

```
private void SiparisiIsle(object sender, EventArgs e)
{
   // Operasyona gelen istemci çağrısında UrunBilgisi nesne örneğine ait veriler bulunmaktadır. Bu verilere aktivite sınıfının özellikleri üzerinden erişilebilir.
   Console.WriteLine("Adet talebi : {0} Urun Numarası : {1} İstek Tarihi : {2}", receiveActivity1_urun1.Adet, receiveActivity1_urun1.UrunKodu, receiveActivity1_urun1.SiparisTarihi.ToString());
   receiveActivity1__ReturnValue_1 = "Istek Alınmıştır"; // İstemciye operasyonda dönecek olan sonuç
}
```

Artık servisi barındıracak olan **Host** uygulamanın yazılmasına başlanabilir. Host uygulama daha öncedende bahsedildiği gibi **WCF** mimarisinin izin verdiği herhangibir çeşitte olabilir ***(IIS, Console, WPF, WAS, Windows Service...)***. örnekte **Host** uygulama basit bir **Console** projesi olarak geliştirilmektedir. Host uygulamada önemli olan noktalardan birisi, **Workflow Service** kütüphanesi ile birlikte, **Workflow** ve **WCF** çalışma ortamları için gerekli **assembly**' lara ihtiyaç olduğudur. Bu nedenle ilk etapta Console uygulamasında **System.ServiceModel, System.Workflow.Activites, System.Workflow.ComponentModel, System.WorkflowServices** kütüphanelerinin referans edilmesi gerekmektedir. Host uygulama için önem arz eden noktalardan biriside çalışma zamanı ortamı için gerekli konfigurasyon ayarlarıdır. Aynen WCF servislerinin yazılmasında olduğu gibi config dosyalarından yararlanılabilir yada kod bazında gerekli ayarlamalar yapılabilir. örnekte kullanılan config dosyası içeriği aşağıdaki gibidir.

Host Uygulama App.config;

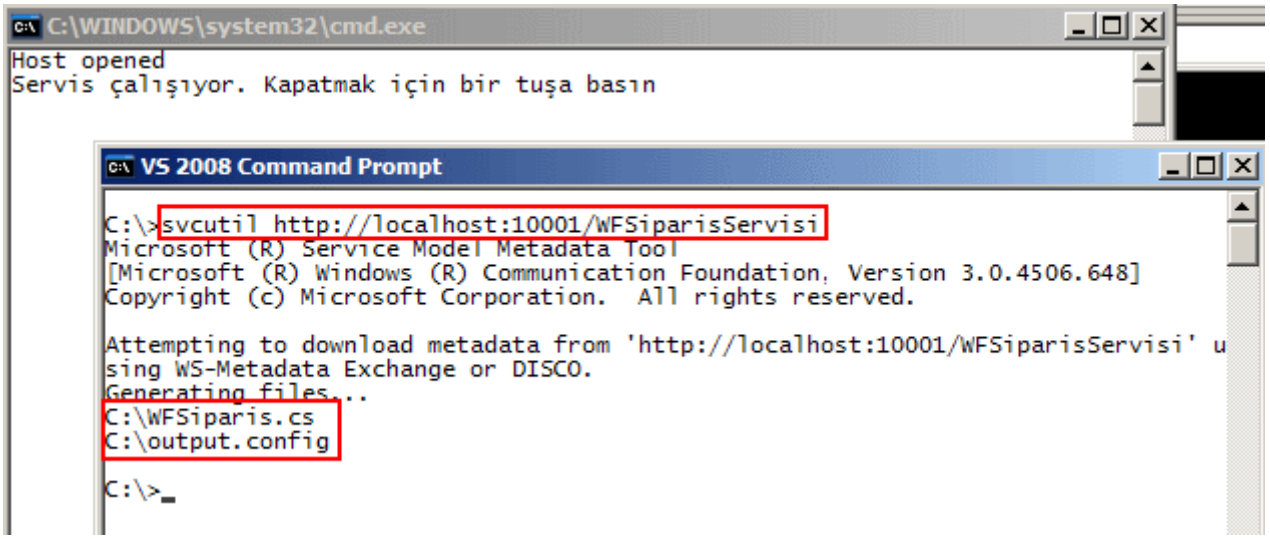
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <services>
         <service behaviorConfiguration="WFSiparisBehavior"
name="WFSiparisKutuphanesi.WFSiparis">
            <endpoint address="" binding="wsHttpContextBinding"
name="WfSiparisWsHttpEndPoint" contract="WFSiparisKutuphanesi.ISiparisSozlesmesi"
/>
            <endpoint address="mex" binding="mexHttpBinding" name="MexEndPoint"
contract="IMetadataExchange" />
            <host>
               <baseAddresses>
                  <add baseAddress="http://localhost:10001/WFSiparisServisi" />
               </baseAddresses>
            </host>
         </service>
      </services>
      <behaviors>
         <serviceBehaviors>
            <behavior name="WFSiparisBehavior" >
               <serviceMetadata httpGetEnabled="true" />
                  <serviceDebug includeExceptionDetailInFaults="true"/>
            </behavior>
         </serviceBehaviors>
      </behaviors>
   </system.serviceModel>
</configuration>
```

Konfigurasyon dosyasında görüldüğü gibi iki adet **EndPoint** tanımlaması yapılmaktadır. Bunlardan birisi bağlayıcı olarak **wsHttpContextBinding** tipini kullanmaktadır. Diğer**EndPoint** ise **base address** üzerinden **metadata** erişimine izin veren bir **Mex(Metadata Exchange) EndPoint** olarak tanımlanmıştır. *(Elbette host uygulamanın IIS üzerinde tutulduğu bir senaryoda Mex EndPoint bildirimine gerek yoktur.)* Host uygulamanın **Main** metoduna ait kodlar ise aşağıdaki gibi geliştirilebilir.

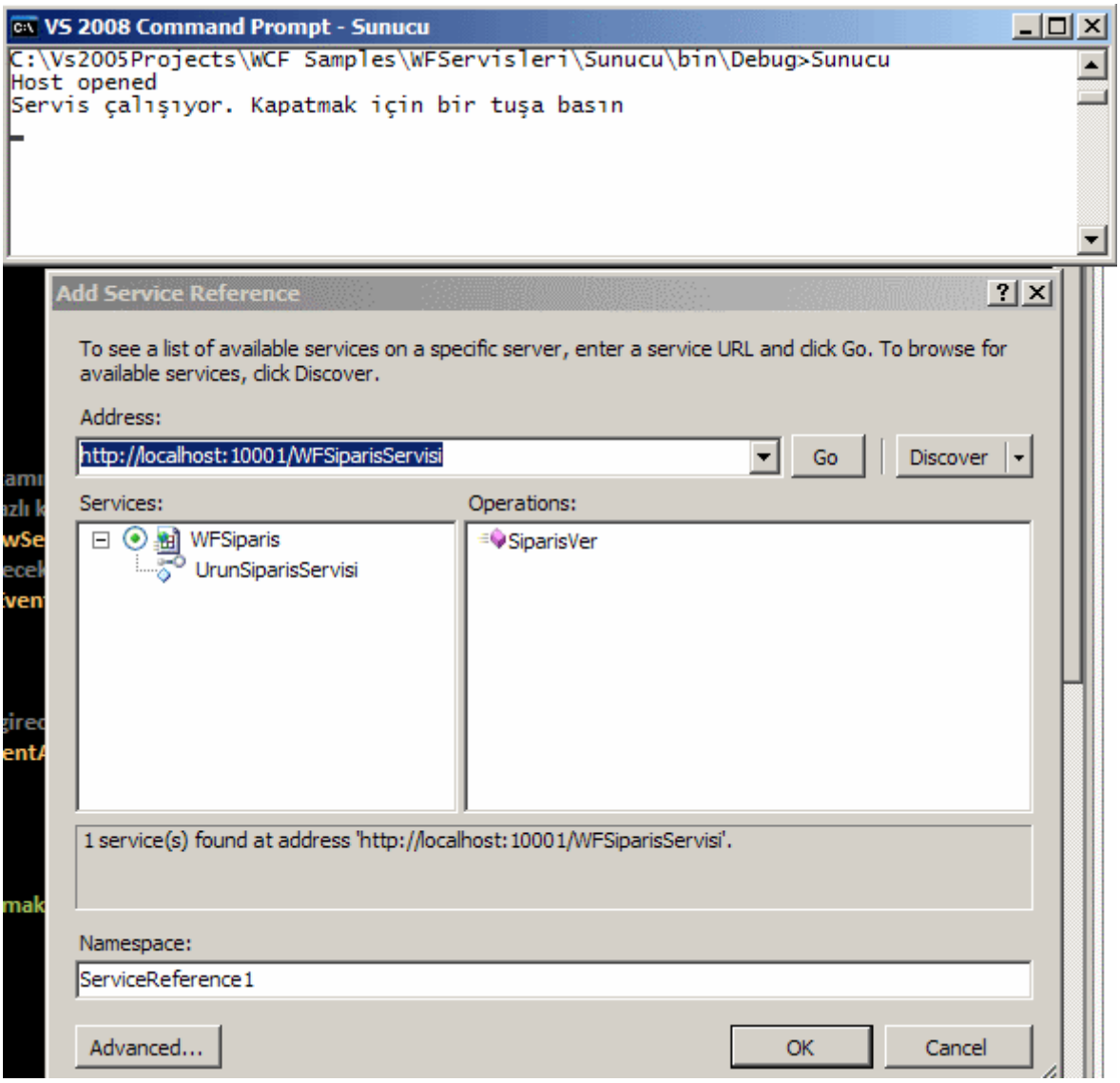
```
using System;
using System.ServiceModel;
using System.Workflow.Runtime;
using WFSiparisKutuphanesi;

namespace Sunucu
{
   class Program
   {
      static void Main(string[] args)
      {
         // Workflow servisi için gerekli çalışma ortamının hazırlanmasını
WorkflowServiceHost tipi üstlenir
         // Parametre olarak yayınlanacak servis bazlı kullanılacak olan aktivite sınıfı
belirtilir.
         WorkflowServiceHost host = new WorkflowServiceHost(typeof(WFSiparis));
         // Host uygulama açıldığından devreye girecek olay metodu
         host.Opened += delegate(object sender, EventArgs arg)
         {
            Console.WriteLine("Host opened");
         };
         // Host uygulama kapatıldığında devreye girecek olan olay metodu
         host.Closed += delegate(object sender, EventArgs arg)
         {
            Console.WriteLine("Host Closed");
         };
         host.Open(); // Host açılır
         Console.WriteLine("Servis çalışıyor. Kapatmak için bir tuşa basın");
         Console.ReadLine();
         host.Close(); // Host kapatılır
      }
   }
}
```

Artık istemci uygulamanın yazılmasına geçilebilir. Ama öncesinde istemci için gerekli **proxy** tipinin ve **konfigurasyon** dosyası içeriğinin üretilmesi gerekmektedir. İki seçenek vardır. **Svcutil**aracı ve **Visual Studio 2008** ortamında ele alınabilen **Add Service Reference**. **Svcutil** aracı komut satırından aşağıdaki ekran görüntüsünde yer aldığı gibi

kullanılabilir. Tabiki bu işlem sırasında **Host** uygulamanın çalışıyor olması ve servisin dışarıdan erişilebilir durumda bulunması gerekmektedir.



Visual Studio 2008 ortamında **Add Service Reference** seçeneği yardımıyla da Proxy ve config üretimi gerçekleştirilebilir. Elbette bu yaklaşımda da Host uygulamanın çalışıyor olması gerekmektedir.

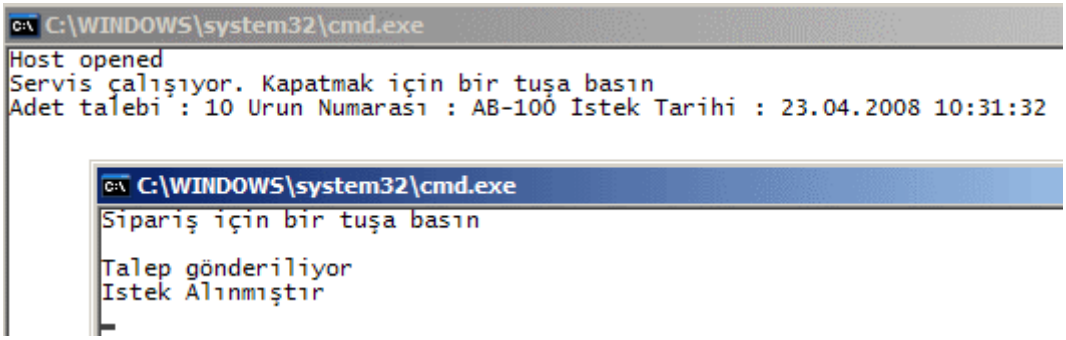


örneğimizde yer alan **proxy** ve **config** dosyalarının üretimi için **Add Service Reference** yaklaşımı kullanılmıştır. Yukarıdaki ekran görüntüsündende takip edilebileceği gibi istemci uygulama **base address** ile belirtilen **Url** adresi üzerinden servis sözleşmesine erişebilmekte ve yayınlanan servis operasyonlarını görebilmektedir. İstemci uygulamada basit bir **Console** projesi olarak tasarlanmıştır ve **Main** metodunun kod içeriği aşağıdaki gibidir.

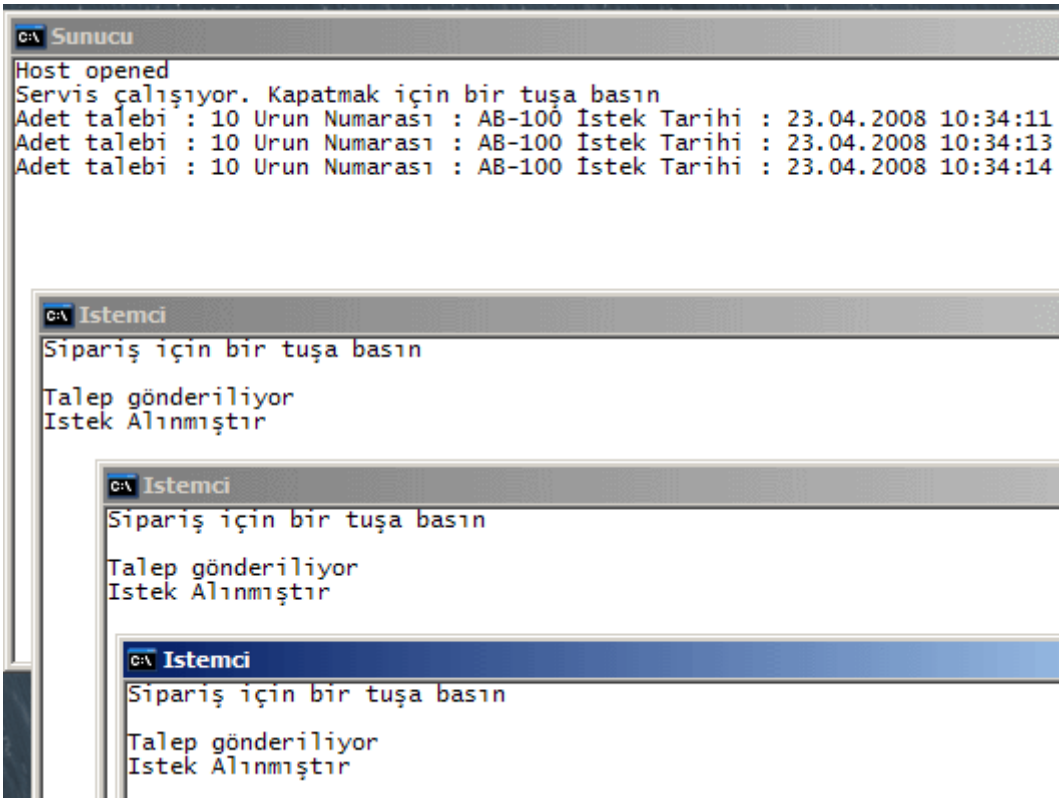
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Istemci.WFSiparisServisi;

namespace Istemci
{
   class Program
   {
```

```
        static void Main(string[] args)
        {
            Console.WriteLine("Sipariş için bir tuşa basın");
            Console.ReadLine();

            // SiparisVer metodu için gerekli parametre üretilir
            UrunBilgisi urn = new UrunBilgisi()
                {
                    Adet = 10,
                    UrunKodu = "AB-100",
                    SiparisTarihi = DateTime.Now
                };

            // Proxy üretimi gerçekleştirilir
            UrunSiparisServisiClient servis = new UrunSiparisServisiClient();
            Console.WriteLine("Talep gönderiliyor");
            // WF Servis operasyonu çağırılır
            string cevap=servis.SiparisVer(urn);
            // Operasyon sonucu gösterilir
            Console.WriteLine(cevap);

            Console.ReadLine();
        }
    }
}
```

önce **Host** uygulama sonrasında istemci uygulama çalıştırılarak test edildiğinde aşağıdaki ekran görüntüsü elde edilir. Görüldüğü gibi istemci uygulama **Workflow**operasyonunu başarılı bir şekilde kullanabilmiştir.



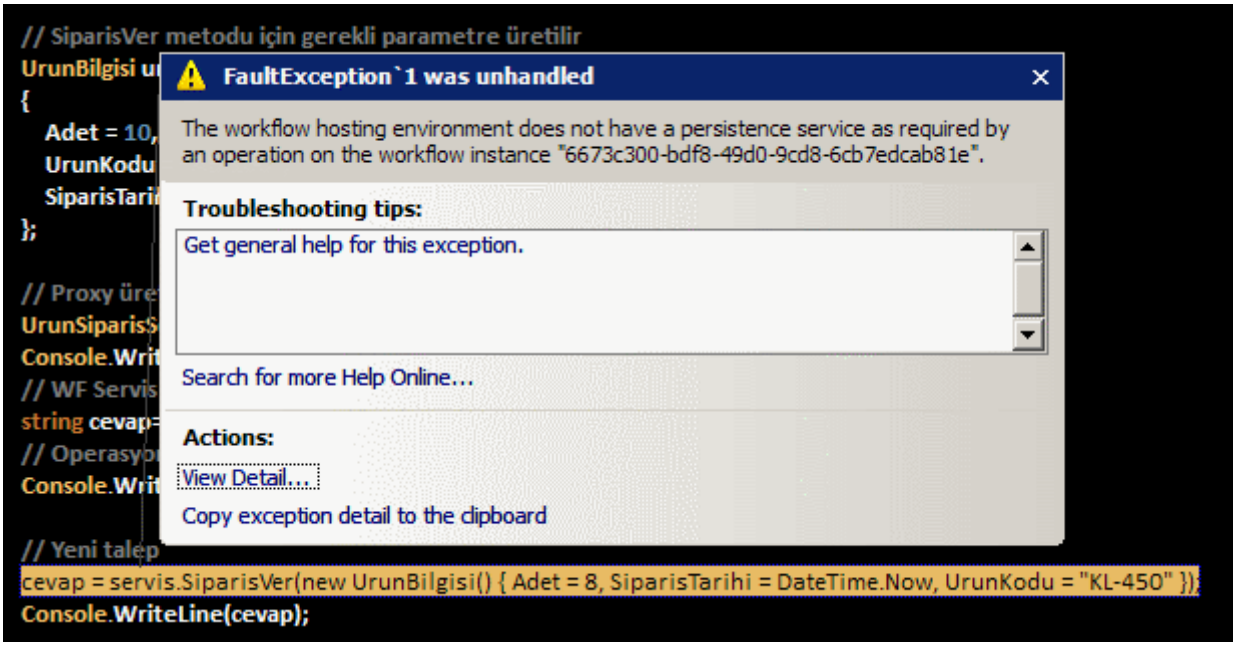
Hatta birden fazla istemci uygulama çalıştırıldığındada WF servisinin başarılı bir şekilde her istemciye cevap verdiği görülmektedir.



Ancak dikkat edilmesi gereken önemli bir durum vardır. Aynı istemci uygulama tarafından ikinci bir sipariş isteği geldiğinde, bir başka deyişle **SiparisVer** metodu aynı proxy örneği üzerinden ikinci bir kez çağırıldığında ne olacaktır? Bunun için koda aşağıdaki satırları eklediğimizi düşünelim.

```
cevap = servis.SiparisVer(new UrunBilgisi() { Adet = 8, SiparisTarihi = DateTime.Now,
UrunKodu = "KL-450" });
Console.WriteLine(cevap);
```

Uygulama test edildiğinde çalışma zamanında ikinci **SiparisVer** metodu çağrısında aşağıdaki istisna(Exception) mesajının alındığı görülür.*(Detaylı hata mesajının istemci tarafındanda görülebilmesi için servis tarafında* ***<serviceDebug includeExceptionDetailInFaults ="true"/>*** *bilgisinin eklenmiş olması gerekmektedir.)*



Bunun sebebi gelen talep sonrasında sunucu tarafında ilgili istemci için üretilen Workflow servis örneğinin cevap verdikten sonra artık olmayışıdır. Bu nedenle ikinci talep aslında istemcinin daha önceden kullandığı Workflow servis örneği için yaptığı istektir. Oysaki **host** uygulama tarafında bu Workflow servis örneğinin işi önceki talebin sonuçlanması ile bitmiştir. *(Tabi burada kolaya kaçılarak pratik bir çözüm olarak istemci uygulama içerisinde her operasyon çağrısı öncesinde yeni bir proxy üretimi yoluna gidilebilir ki bu tavsiye edilen bir yol değildir.)* İşte burada uzun süreli durağan olması gereken bir Workflow servisi örneği söz konusudur. Yani **Workflow** örneğinin durumunu koruması gerekmektedir. **Persistence** servisleri kullanılarak bu sorun çözümlenebilir. Bu amaçla **SqlWorkflowPersistenceService** tipinden yararlanılarak ilgili durağanlığın gerçekleştirilmesi sağlanabilir. Tabi bu amaçla öncelikli olarak SQL üzerinde ilgili veritabanının oluşturulması ve tabloların hazırlanması gerekmektedir. Sonrasında ise ilgili PersistenceService' in örneklenip Workflow servisi çalışma ortamına kod yardımıyla yada konfigurasyon bazında bildirilmesi gerekmektedir. Bu konu yazının kapsamı dışına çıktığından burada ele alınmayacaktır.

Böylece geldik bir makalemizin daha sonuna. Bu makalede basit olarak Workflow örneklerinin birer servis olarak nasıl yayımlanabileceğini adım adım incelemeye çalıştık. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

WFServisleri.rar (89,02 kb)

WCF ile WF Entegrasyonu - 1 (2008-04-17T02:24:00)

wcf,wf,

Bilindiği üzere **Window Communication Foundation** ve **Windows Workflow Foundation**, **.Net Framework 3.0** ile birlikte gelen önemli teknolojilerdendir. WCF **servis**

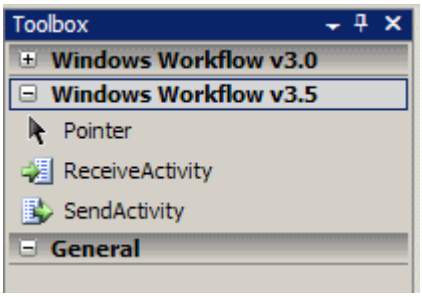
yönelimli mimariye(Service Oriented Architecture) yeni bir yaklaşım getirip, dağıtık mimari uygulama geliştirme kavramlarını bir çatı altında toplayarak güçlü, daha platform bağımsız ve güvenilir bir ortamda geliştirme yapılabilmesini olanaklı kılan bir alt yapı sunmaktadır. WF ise, birden fazla adımdan oluşan kod süreçlerinin **iş akışı(Workflow)** tarzında **olay güdümlü(Event-Driven)** yada **sırasal (Sequential)** olacak şekilde tasarlanarak çeşitli .Net uygulamalarında kullanılabilmelerini mümkün kılan bir alt yapı tesit etmektedir. **Workflow**, süreçlere ait akışların hazırlanmasında ağırlıklı olarak **aktivite tiplerini(Activity Types)** kullanmaktadır. Bu aktivitelerin dallara ayrılması, çatallanması, birbirlerine katılması gibi pek çok işlem de, Workflow ortamı tarafından sunulmaktadır. WF ile geliştirilen iş akışları kısa süreceği gibi uzunda**(Long-Running Workflows)** sürebilir. Bu nedenle WF ortamı özellikle uzun süren akışların, **sistem yeniden başlatmaları(reboot)** gibi vakalara karşı ayakta durabilmesi için kalıcı olarak saklama işlemlerine destek de vermektedir. WF mimarisinin yetenekleri sadece bununlada sınırlı değildir. örneğin Transaction desteği diğer yetenekleri arasında gösterilebilir.

***Not : Windows Workflow Foundation** ile **Windows Communication Foundation** arasındaki entegrasyon gerçek anlamda **.Net Framework 3.5** ve **Visual Studio 2008** ile birlikte sağlanmıştır.*

WF aslında, **.Net Framework 3.5** ve **Visual Studio 2008** ile gelen yenilikler sayesinde **WCF** ortamının gerçek anlamda bir **tamamlayıcısı** teknoloji olarak görülmektedir. Bu anlamda WF uygulamaları içerisinden WCF servislerinin çağırılması ve kod akışının ilgili **servis noktaları(Service EndPoint)** üzerinden yürütülmesi sağlanabilmektedir. Tam tersine WF içerisindende servis yayınlaması yapılabilmektedir. Bu entegrasyon sayesinde süreçlerin istemcilere **güçlü(Robust), güvenilir(Reliable)** ve **güvenli(Secure)** bir şekilde sunulmasıda sağlanmaktadır. öyleki bu entegrasyonun doğal sonucu olarak iş mantığının(Logic) farklı formatlarda***(MTOM, SOAP, Binary, JSON, X509 vb...)*** dolaşımı, **IIS(Internet Information Services), WAS(Windows Activation Service) ve Windows Service** gibi ortamlar üzerinden **host** edilme imkanı gibi pek çok fonksiyonellik ele alınabilmektedir.

WCF ile **WF** entegrasyonu sırasında servis ile olan etkileşimin modellenebilmesi için **Send, Receive** gibi aktivitelerden yararlanılmaktadır. **Send** aktivitesi sayesinde **WF** içerisinden **bir WCF servisine mesaj gönd**erilmesi sağlanabilir ki bu noktada **Proxy** nesneleride devreye girmektedir. Diğer taraftan **Receive** aktivitesi sayesinde **workflow**' un kendisinin **bir servis gibi sunulabilmesi** olanaklı hale gelmektedir.

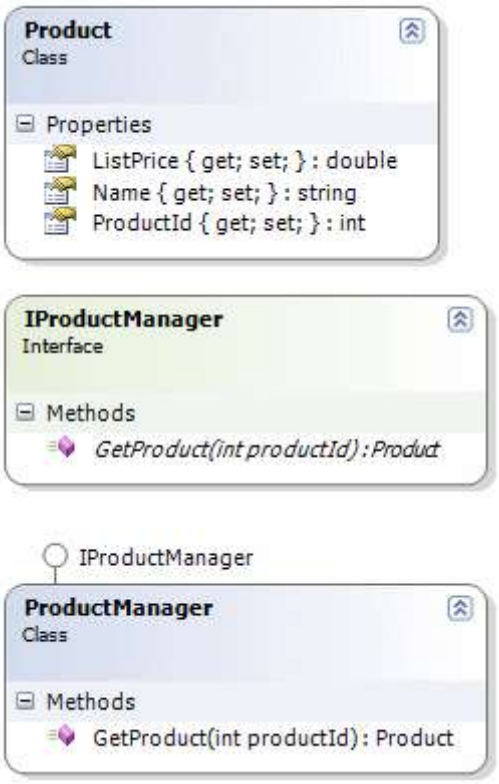
***SendActivity** ve **ReceiveActivity** tipleri .Net Framework 3.5 ile birlikte gelmiştir.*



Aktivasyon alt yapısının host edilmesi içinse **ServiceHostBase abstract** sınıfından türeyen ve **.Net Framework 3.5** ile birlikte gelen **WorkflowServiceHost** tipinden yararlanılmaktadır. Servis ve istemci arasındaki önemli konulardan biriside korelasyonun sağlanmasıdır. Bu bir anlamda istemcinin doğru servis örneği ile iletişime geçebilmesi demektir ki bunun sağlanabilmesi için eklenmiş olan yeni **davranış(Behavior)** ve **bağlayıcılar(Bindings)** söz konusudur.

Bu teorik bilgilerden sonra örnekler ile devam edelim. İlk olarak **bir WF uygulaması içerisinde bir WCF servisinin nasıl çağırılabileceğini** incelemeye çalışacağız. WF içerisinden bir WCF servis noktasına ulaşmak için **SendActivity, InvokeWebServiceActivity, CodeActivity** aktivite tiplerinden yararlanılabilir. Bunlardan en güçlü olanı SendActivity dir.**InvokeWebServiceActivity Web servislerinin proxy** sınıfı aracılığıyla çağırılmasını sağlamaktadır. WCF tarafından **asmx** modeline uygun yayınlama yapılabildiğinden bu aktivite tipide tercih edilebilir. Nevarki **SendActivity** tipine göre herhangibir üstünlüğü bulunmamaktadır. **SendActivity** WCF servisi ile **senkron(synchronous)** olarak haberleşilmesini sağlar ve**tek yönlü(One-Way), talep-cevap(Request-Response), talep-hata(Request-Fault)** desenlerini ele alır. Tek yönlü desene göre servise talepte bulunulduktan sonra bir cevap beklenmez. Talep-Cevap desenine göre ise servisten yapılan isteğe bir sonuç gelinceye kadar beklenir(Synchronous). Son desene göre ise ya cevap gelir yada **hata mesajı(Fault Message)**. Bunların dışında **özel aktiviteler(Custom Activity)** yazılarak iş mantığının söz konusu aktivite tip içerisine gömülmesi ve servisin ele alınmasıda sağlanabilir.

Elbette ilk olarak bir servis kütüphanesinin(WCF Service Library) ve host uygulamanın tasarlanması gerekmektedir. örnekte kullanılacak olan servis kütüphanesinin içeriği aşağıdaki gibidir.



Product sınıfı;

```
[DataContract]
public class Product
{
   [DataMember]
   public int ProductId { get; set; }
   [DataMember]
   public string Name { get; set; }
   [DataMember]
   public double ListPrice { get; set; }
}
```

Product sınıfı **Production.Product** tablosundaki herhangibir satıra ait **ProductID**,**Name** ve **ListPrice** bilgilerini taşımak üzere tasarlanmış ve bu sebepten bir **veri sözleşmesi(DataContract)**olacak şekilde tanımlanmıştır.

IProductManager arayüzü(Interface);

```
[ServiceContract(Name="Urun
Servisi",Namespace="http://www.bsenyurt.com/UrunServisi")]
public interface IProductManager
{
   [OperationContract]
```

```
    Product GetProduct(int productId);
}
```

Servis sözleşmesi(Service Contract) Product tipinden nesne örnekleri döndüren tek bir operasyon tanımı içermektedir.

ProductManager sınıfı;

```
public class ProductManager
        : IProductManager
{
    #region IProductManager Members

    public Product GetProduct(int productId)
    {
        Product prd = null;
        using (SqlConnection conn = new SqlConnection("data
source=.;database=AdventureWorks;integrated security=SSPI"))
        {
            SqlCommand cmd = new SqlCommand("Select ProductId,Name,ListPrice From
Production.Product Where ProductId=@PrdId", conn);
            cmd.Parameters.AddWithValue("@PrdId", productId);
            conn.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            if (reader.Read())
                prd = new Product { ProductId = Convert.ToInt32(reader["ProductId"]), Name =
reader["Name"].ToString(), ListPrice = Convert.ToDouble(reader["ListPrice"]) };
            reader.Close();
        }
        return prd;
    }

    #endregion
}
```

ProductManager sınıfında yazılan **GetProduct** metodu, parametre olarak gelen değere göre **Production**.**Product** tablosundan bir satır verisini çekmektedir. Eğer parametre olarak gelen id değerine bağlı bir satır varsa **ProductId**,**Name**,**ListPrice** değerlerinin toplandığı **Product** nesne örneği geri döndürülmektedir. Sınıf kütüphanesinin tanımlanmasından sonra host servis uygulamasının yazılması gerekmektedir. örnekte Host, basit bir Console uygulaması olacak şekilde aşağıdaki gibi tasarlanmıştır.

Program sınıfı içeriği;

```
using System;
using System.ServiceModel;
using ProductServices;

namespace Sunucu
{
   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(ProductManager));
         host.Open();
         Console.WriteLine("Sunucu dinlemede...");
         Console.ReadLine();
         host.Close();
      }
   }
}
```

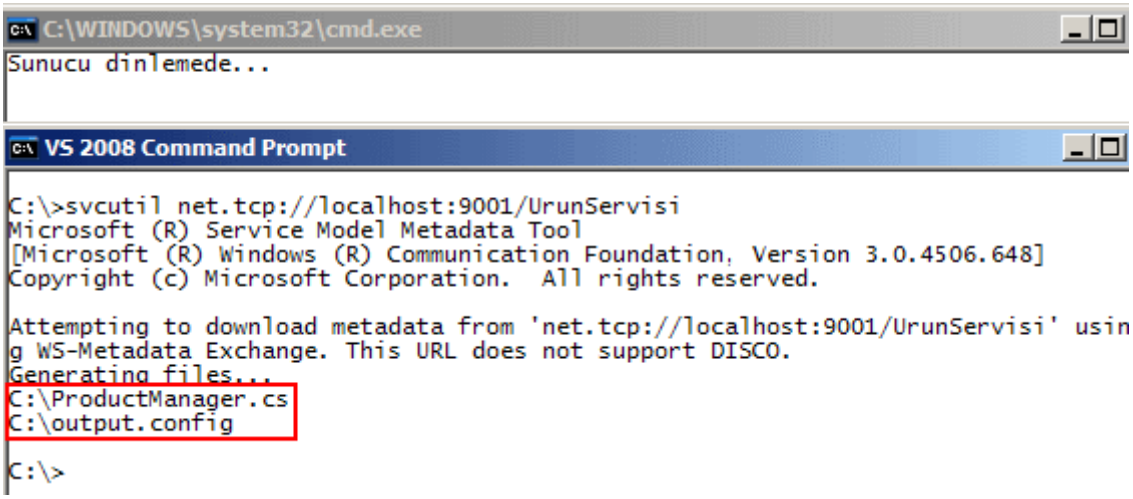
App.config içeriği;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <behaviors>
         <serviceBehaviors>
            <behavior name="UrunServisiBehavior">
               <serviceMetadata/>
            </behavior>
         </serviceBehaviors>
      </behaviors>
      <services>
         <service behaviorConfiguration="UrunServisiBehavior"
name="ProductServices.ProductManager">
            <endpoint address="" binding="netTcpBinding" bindingConfiguration=""
name="UrunServisiTcpEndPoint" contract="ProductServices.IProductManager" />
            <endpoint address="mex" binding="mexTcpBinding"
name="UrunServisiMexEndPoint" contract="IMetadataExchange" />
            <host>
               <baseAddresses>
                  <add baseAddress="net.tcp://localhost:9001/UrunServisi" />
               </baseAddresses>
            </host>
         </service>
      </services>
```

```
    </system.serviceModel>
</configuration>
```

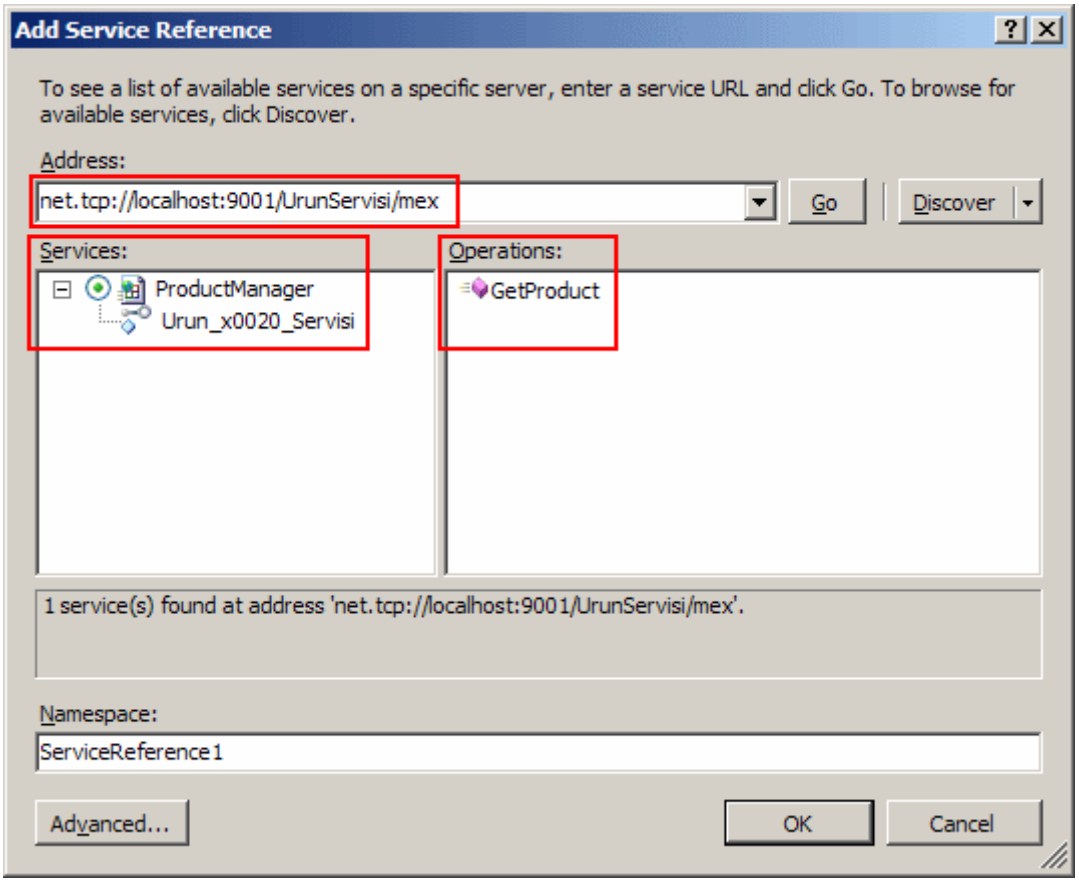
İlerlemeden önce **App.config** dosyası içeriğinin incelenmesinde yarar vardır. Servis uygulaması iki adet **EndPoint** sunmaktadır. **UrunServisiTcpEndPoint** isimli servis noktası TCP*(netTcpBinding bağlayıcısını kullandığına dikkat edelim)* bazlı olacak şekilde bir yayınlama yapmaktadır. **address** bilgisi verilmemiş olmasına rağmen **host** elementi içerisinde tanımlanan**baseAdress** bilgisi kullanılmaktadır. Diğer EndPoint ise **IMetadaExchange** arayüzünü servis sözleşmesi olarak kullanan ve **mexTcpBinding** bağlayıcısını baz alarak **mex** isimli bir adres üzerinden yayınlama yapan bir servis noktası sunmaktadır. Bu EndPoint sayesinde baseAddress elementi ile bildirilen **TCP** adresi üzerinden **Metadata Exchange** işlemi gerçekleştirilebilir. Bir başka deyişle çalışmakta olan servis uygulamasına bağlı adres üzerinden **proxy** nesnesi üretimi gerçekleştirilebilir. önemli olan noktalardan biriside proxy üretiminin sağlanmasıdır. Bu nedenle servise metadata yayınlaması için bir **servis davranışı(Service Behavior)** eklenmiştir. Böylece servis operasyonlar servis dışına sunulabilir hale gelmiştir.

*WF uygulaması, dışarıdan bir WCF servisini çağırmak için **proxy** nesnesinden yararlanmaktadır. Bu proxy nesnesi **Visual Studio 2008** ortamında **Add Service Reference** ile kolay bir şekilde eklenebileceği gibi **svcutil** aracı yardımıylada aşağıdaki şekilde olduğu gibi çekilebilirde.*

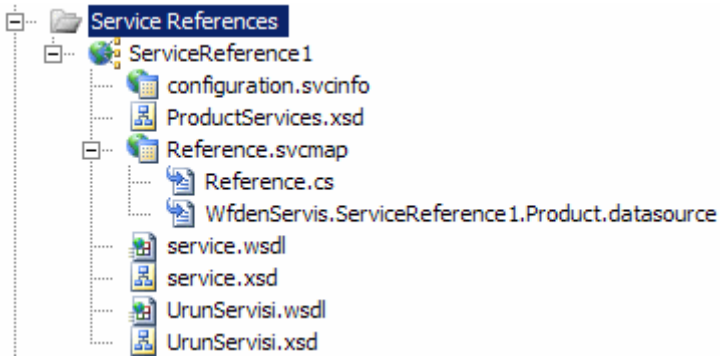


Hangisi tercih edilirise edilsin örneğe göre servis uygulamasının çalışır durumda olması gerekmektedir.

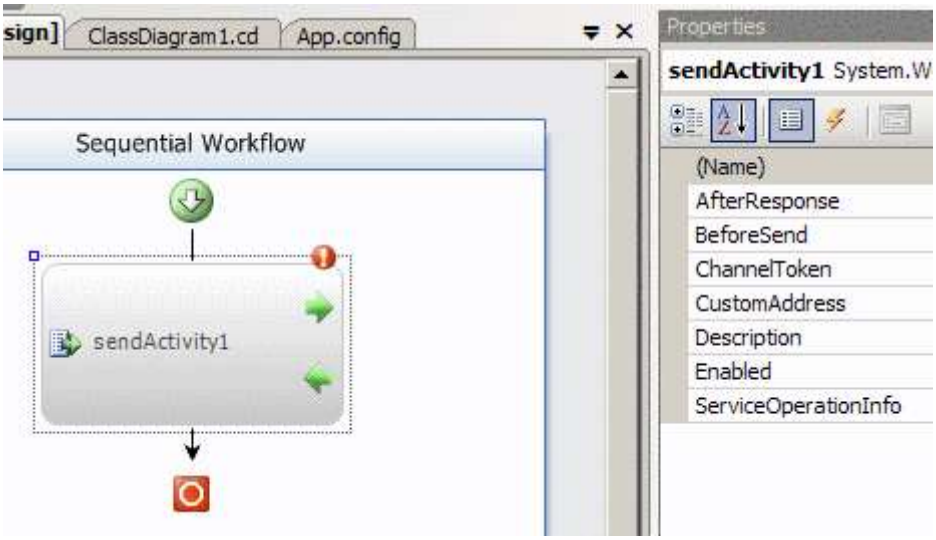
Artık WF uygulamasının yazılmasına başlanabilir. örnekte **Sequential Workflow Console Application** şablonu kullanılmaktadır. Proje açıldıktan sonra servis uygulamasının çalışıyor olmasına dikkat edereken, **Add Service Reference** seçeneği ile proxy sınıfının üretilmesi ve WF uygulamasına eklenmesi sağlanabilir.



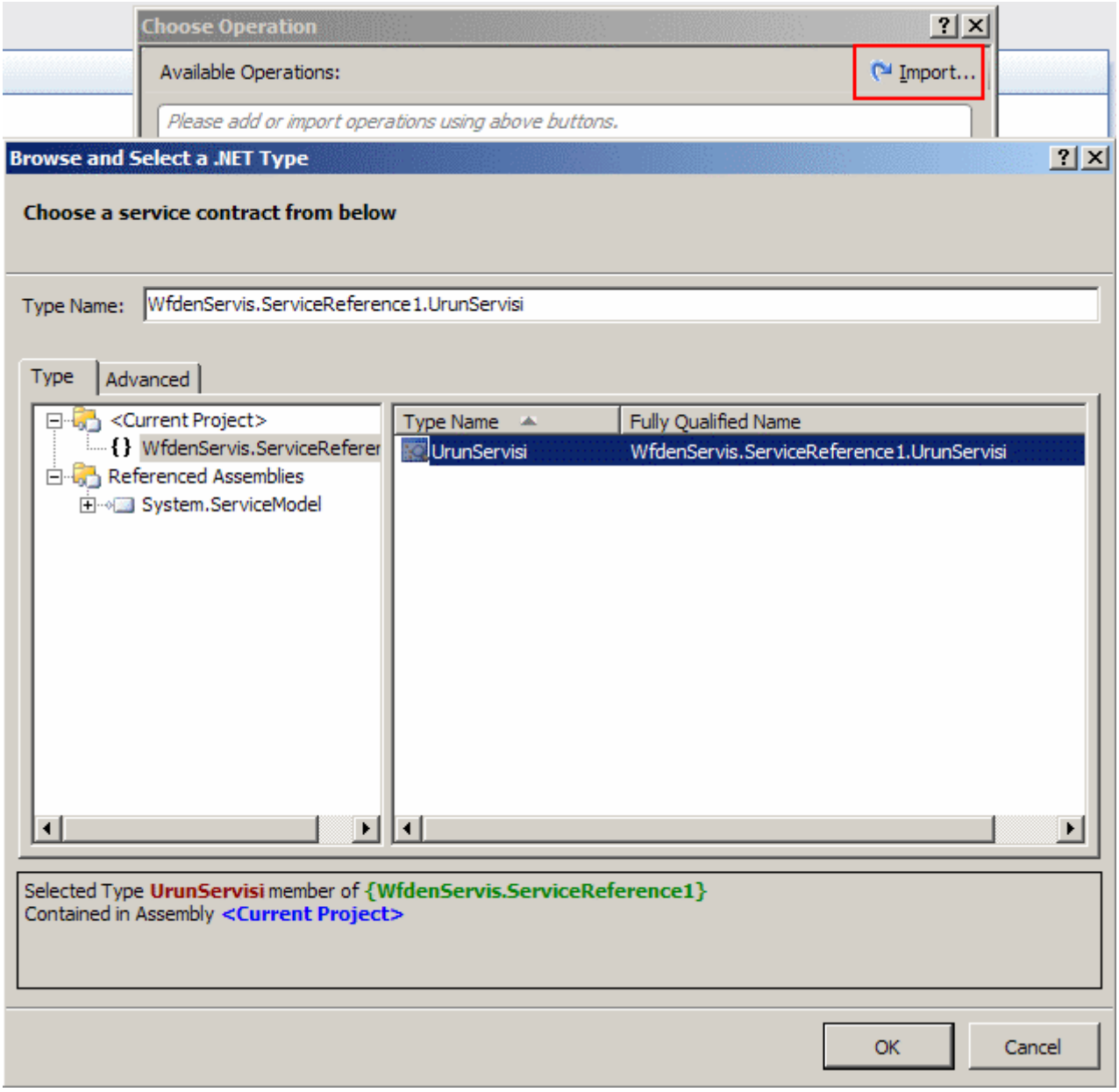
Bu işlemin arkasından WF uygulaması içerisinde aşağıdaki gibi servisa ait bilgilerin indirildiği ve proxy sınıfının*(Reference.cs içerisinde yer almaktadır)* üretildiği görülebilir.



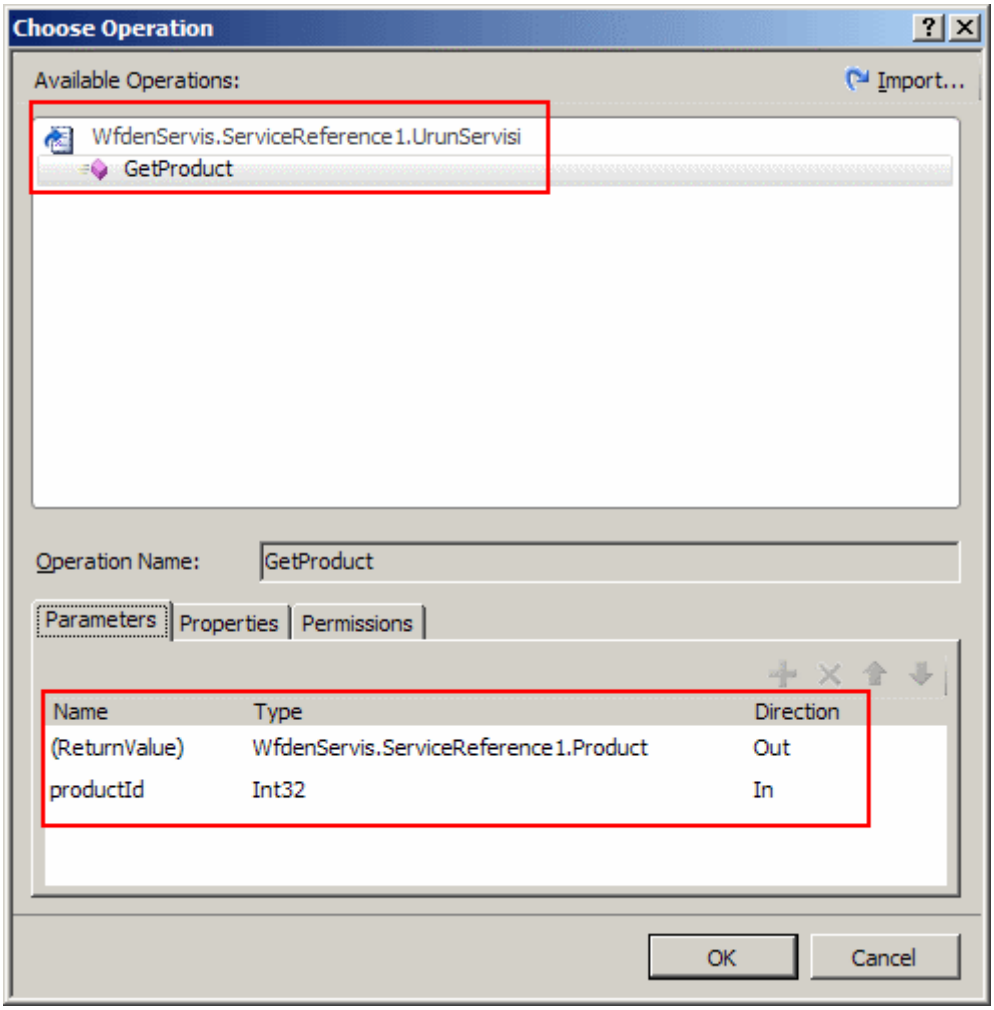
örnekte WF içerisinden servisi çağırmak için **SendActivity** tipi kullanılacaktır. İlk olarak Workflow1 tasarım ortamına **SendActivity** bileşeni sürüklenmelidir.



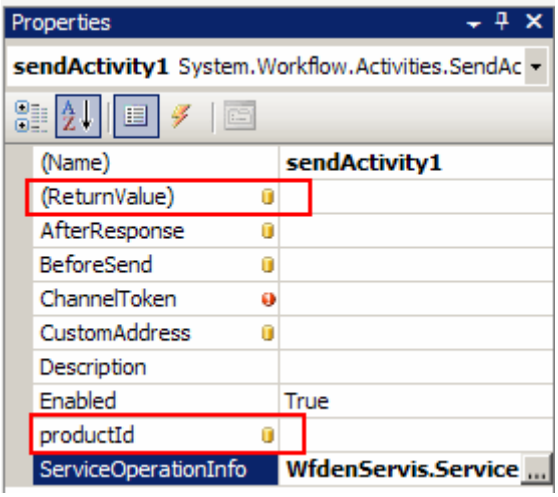
SendActivity bileşeninin önemli olan bazı üyeleri vardır.
Bunlardan **ServiceOperationInfo** özelliği yardımıyla etkileşimde bulunulacak olan servisin ve ilgili operasyonunun seçilmesi sağlanır.**AfterResponse** alanının işaret ettiği olay sayesinde, servis tarafından cevap geldikten sonra işletilmesi istenen kodların icra edilmesi sağlanmaktadır. Benzer şekilde **BeforeSend** alanının işaret ettiği olay sayesinde, servise mesaj gönderilmeden önce işletilmesi gereken bir kod mantığı var ise, bunun icra edilmesi sağlanmaktadır. İstenirse özel servis adresleri **CustomAdress**özelliği ile tanımlanabilir. örnekte ilk olarak **ServiceOperationInfo** özelliğinden yararlanılarak kullanılacak operasyon seçilmelidir.



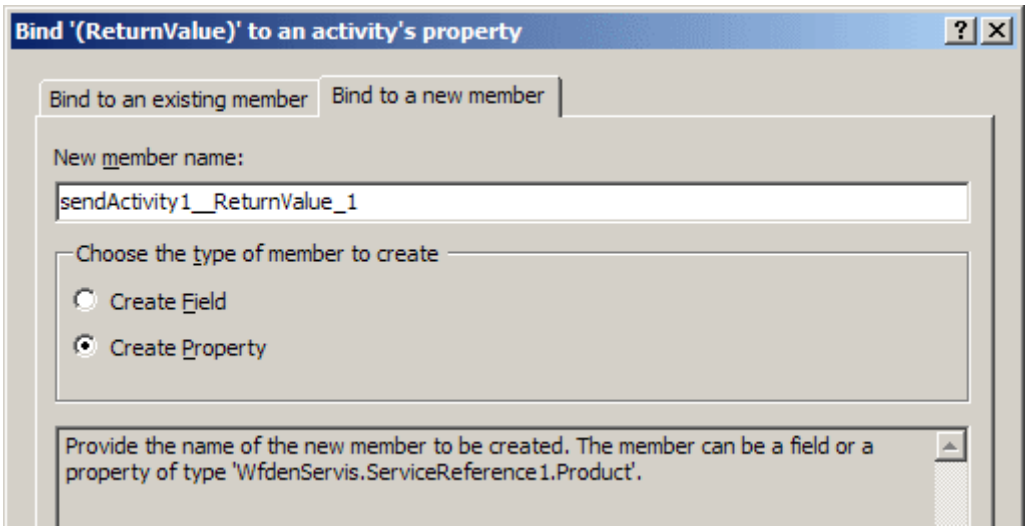
Import düğmesine basıldıktan sonra ekrana gelen ara birimde, projeye az önce referans edilmiş olan serviste görülecektir*(ServiceReference1)*. Bu işlemin hemen arkasından **UrunServisi** tipine çift tıklanırsa aşağıdaki ekran görüntüsü elde edilir.



Görüldüğü gibi servise ait **GetProduct** operasyonu eklenmiştir. **Parameters** kısmında, operasyonun aldığı ve geri döndürdüğü değişkenlere ait bir takım bilgiler yer almaktadır. Buna operasyon productId isimli Int32 tipinden bir parametre almakta ve Product tipinden bir sonuç döndürmektedir. **OK** düğmesine basıldıktan sonra **Parameters** kısmında yer alan değişkenlerin **Properties** penceresine birer özellik olaraktan eklendiği izlenebilir.



Bu özelliklerin yanında yer alan üç nokta düğmelerine basıldığında, ilgili alanların servis operasyonuna bağlanması için gerekli **özellik/alan(Property/Field)** tanımlamalarının yapılacağı bir ekran ile karşılaşılır. Bu ekrandan yararlanılarak **Workflow** sınıfı içerisinde yazılmış var olan özelliklere/alanlara bağlama yapılabileceği gibi **Bind to a new member** sekmesinden faydalanılarak yeni özelliklerin anında oluşturulmasıda sağlanabilir. örneğin **ReturnValue** özelliği için aşağıdaki ekran görüntüsünde yer alan seçimler kullanılmış ve anında **sendActivity1_ReturnValue1** isimli bir üyenin oluşturulması sağlanmıştır.

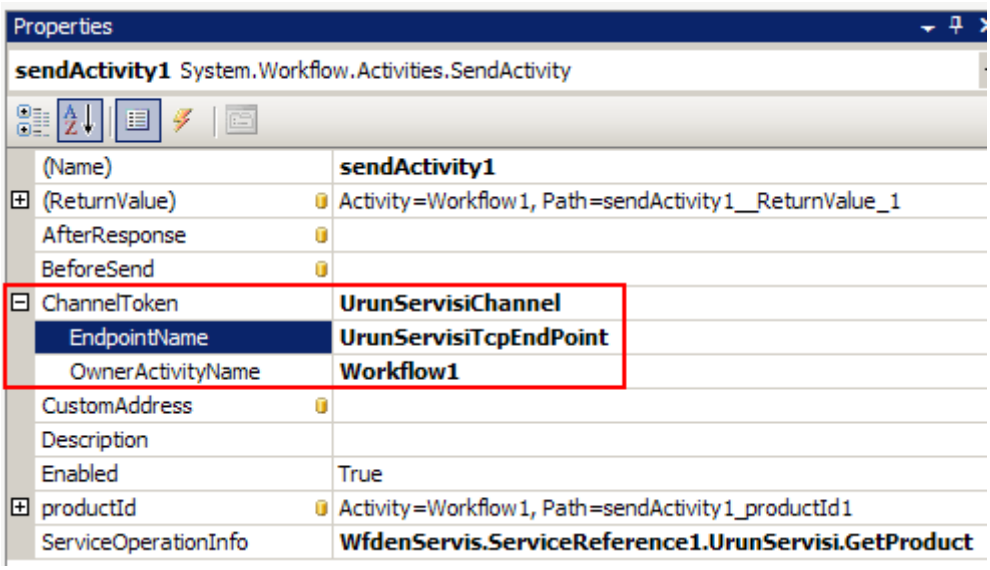


Aynı işlem **productId** isimli aktivite özelliği içinde yapılmalıdır. Bu işlemlerin ardından Workflow1 sınıfı içerisine DependencyProperty tipinden iki yeni özelliğin aşağıdaki gibi eklendiği görülür.

```
namespace WfdenServis
{
   public sealed partial class Workflow1: SequentialWorkflowActivity
   {
      public Workflow1()
      {
         InitializeComponent();
      }

      public static DependencyProperty sendActivity1__ReturnValue_1Property =
DependencyProperty.Register("sendActivity1__ReturnValue_1",
typeof(WfdenServis.ServiceReference1.Product), typeof(WfdenServis.Workflow1));

      [DesignerSerializationVisibilityAttribute(DesignerSerializationVisibility.Visible)]
      [BrowsableAttribute(true)]
      [CategoryAttribute("Parameters")]
      public WfdenServis.ServiceReference1.Product sendActivity1__ReturnValue_1
      {
         get
```

```
        {
            return
((WfdenServis.ServiceReference1.Product)(base.GetValue(WfdenServis.Workflow1.send
Activity1__ReturnValue_1Property)));
        }
        set
        {
            base.SetValue(WfdenServis.Workflow1.sendActivity1__ReturnValue_1Property
, value);
        }
    }

    public static DependencyProperty sendActivity1_productId1Property =
DependencyProperty.Register("sendActivity1_productId1", typeof(System.Int32),
typeof(WfdenServis.Workflow1));

    [DesignerSerializationVisibilityAttribute(DesignerSerializationVisibility.Visible)]
    [BrowsableAttribute(true)]
    [CategoryAttribute("Parameters")]
    public Int32 sendActivity1_productId1
    {
        get
        {
            return
((int)(base.GetValue(WfdenServis.Workflow1.sendActivity1_productId1Property)));
        }
        set
        {
            base.SetValue(WfdenServis.Workflow1.sendActivity1_productId1Property,
value);
        }
    }
  }
}
```

Son olarak SendActivity bileşeninin **ChannelToken** özelliği ayarlanarak **EndPoint** bilgilerinin verilmesi ve WF' in hangi servis ile nasıl mesajlaşacağının ele alınması gerekmektedir. Bu bilgiler çok doğal olarak **Add Service Reference** işlemi sonrası gelen **App.config** dosyası içerisinde yer almaktadır. Bu amaçla **ChannelToken** özelliğine bir isim verildikten sonra gelecek olan **EndpointName**özelliğine **App.config** dosyası içerisindeki ilgili servis noktasının adı yazılmalıdır. Sonrasında ise **ChannelToken** nesnesinin kapsamı(Scope) belirlenir. Bu kapsam **OwnerActivityName**özelliği yardımıyla set edilmektedir. örnekte söz konusu özellikler aşağıdaki ekran görüntüsünde olduğu gibi belirlenebilir.



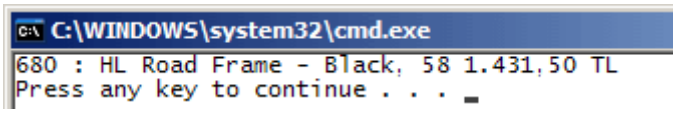
Artık test işlemleri başlatılabilir. **Console** formatında bir **Workflow** uygulaması geliştirildiğinden **Main** metodu içerisinde gerekli parametre tanımlama ve sonuç alma işlemleri kolaylıkla gerçekleştirilebilir. Bu amaçla **Main** metodu aşağıdaki gibi genişletilebilir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using System.Workflow.Runtime;
using System.Workflow.Runtime.Hosting;
using WfdenServis.ServiceReference1;

namespace WfdenServis
{
   class Program
   {
      static void Main(string[] args)
      {
         using(WorkflowRuntime workflowRuntime = new WorkflowRuntime())
         {
            AutoResetEvent waitHandle = new AutoResetEvent(false);
            workflowRuntime.WorkflowCompleted += delegate(object sender,
WorkflowCompletedEventArgs e)
               {
                  waitHandle.Set();
                  Product result = e.OutputParameters["sendActivity1__ReturnValue_1"] as
Product;
                  if(result!=null)
```

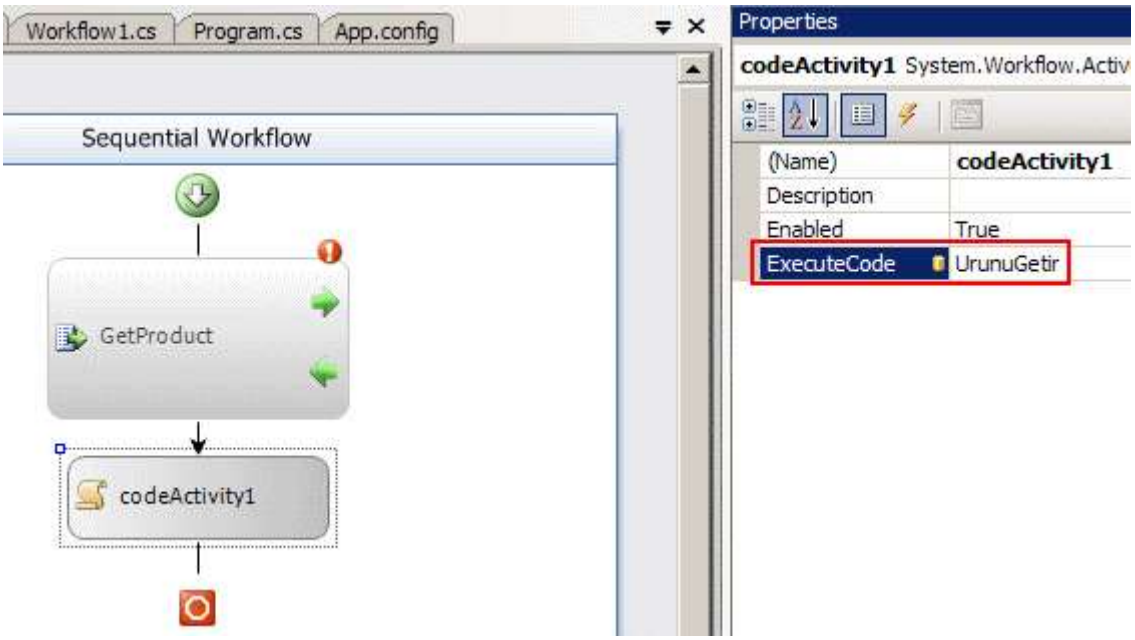
```
                    Console.WriteLine("{0} : {1} {2}", result.ProductId.ToString(),
result.Name, result.ListPrice.ToString("C2"));
                else
                    Console.WriteLine("ürün bulunamadı");
            };

          workflowRuntime.WorkflowTerminated += delegate(object sender,
WorkflowTerminatedEventArgs e)
            {
                Console.WriteLine(e.Exception.Message);
                waitHandle.Set();
            };

            Dictionary<string, object> parametreler = new Dictionary<string, object>() { {
"sendActivity1_productId1", 680 } };
            WorkflowInstance instance =
workflowRuntime.CreateWorkflow(typeof(WfdenServis.Workflow1),parametreler);

            instance.Start();

            waitHandle.WaitOne();
        }
    }
  }
}
```

Bilindiği gibi **WorkflowInstance** nesne örneği oluşturulurken **Dictionary<string,object>** tipinden olan ikinci parametre ile iş akışı içerisindeki özelliklere değer aktarımı gerçekleştirilebilmektedir. Bu sebepten örnekte **parametreler** isimli **Dictionary<string,object>** tipinden bir koleksiyon tanımlanmış ve **sendActivity1_productId1** ismi ile **680** değeri verilmiştir.**sendActivity1_productId1** isimli özellik hatırlanacağı gibi **SendActivity** bileşeninin bir parçasıdır ve **GetProduct** metodunun aldığı **productId** alanına eş düşmektedir. Bu sayede servis tarafındaki ilgili operasyona parametre gönderimi gerçekleştirilmektedir. Servis tarafındaki işlemler sonlandığında tasarlanan iş akışına göre otomatik olarak **WorkflowCompleted**olayı tetiklenmektedir. Bu olayın **WorkflowCompletedEventArgs** tipinden olan e parametresine ait **OutputParameters** özelliğinin işaret ettiği koleksiyon, iş akışına ait tüm özelliklere ulaşılabilmesini sağlamaktadır ki bunlardan biriside **GetProduct** isimli servis metodunun dönüş değerini işaret eden ve **SendActivity** bileşenine ait olan **sendActivity1_ReturnValue_1** isimli **anahtardır(key)**. Bu anahtarın sonucu **Product** tipinden olacağı için **as** anahtar kelimesi ile bir dönüştürme işlemi yapılmakta ve sonuç **null** değilse elde edilen değişkene ait bilgiler ekrana yazdırılmaktadır. Uygulamanın test edilebilmesi için **öncelikli olarak servis tarafının çalışıyor**

olması gerekmektedir. Buna göre iş akışı uygulamasının **680 productId** değeri için vereceği çıktı aşağıdaki gibi olacaktır.



Elbette tasarlanan akışın tipine bağlı olaraktan, servise ait operasyonların adımların arasında herhangibir yerde tetiklenmesi ve arkasından sonuçların alınması için başka aktivitelerin(örneğin CodeActivity) iş akışına eklenmeside söz konusudur. Söz gelimi yazılan son örnekte ek bir CodeActivity bileşeni kullanılabilir. Bunun için tasarım zamanında aşağıdaki gibi bir **CodeActivity** bileşeni eklendiğini ve **ExecuteCode** özelliğindede **UrunuGetir** isimli bir metod bildirimi yapıldığını varsayalım.



Buna göre **UrunuGetir** metodunun içeriği aşağıdaki gibi tasarlanıp **SendActivity** çağrısından sonra **GetProduct** operasyonunun sonuçlarının alınması sağlanabilir. *(Burada**sendActivity1_ReturnValue_1** değerine özelliğine doğrudan erişilmesi son derece doğaldır nitekim metodun tanımlandığı yer **Workflow1** sınıfının içidir.)*

```
private void UrunuGetir(object sender, EventArgs e)
{
   WfdenServis.ServiceReference1.Product prd = sendActivity1__ReturnValue_1 as
WfdenServis.ServiceReference1.Product;
   Console.WriteLine(prd.Name + " " + prd.ListPrice.ToString("C2"));
}
```

Görüldüğü gibi **SendActivity** bileşeni sayesinde bir iş akışı nesnesi içerisinden **WCF** tabanlı servis çağırılması, bu servisten sunulan operasyonların icra edilmesi, varsa operasyon sonuçlarının alınması gibi aksiyonlar kolay bir şekilde gerçekleştirilebilmektedir. Bu yazıda bir WCF servisinin, WF içerisinden nasıl çağırılabileceğinin temelleri üzerinde durulmuş ve basit bir örnek adım adım işlenmeye çalışılmıştır. Bir sonraki makalede ise bir WF uygulamasının servis olarak nasıl sunulacağı konularına değinilmeye çalışılacaktır. Böylece geldik bir makalemizin daha sonunda. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

WFileWCF.rar (77,62 kb)

LINQ Maceralarım (2008-04-10T01:36:00)

c# 3.0,linq,

Language INtegrated Query(LINQ) mimarisi sayesinde **CLR nesneleri(Common Language Runtime Objects)** üzerinden **SQL** tarzı sorgu ifadeleri yazılabilmektedir. Hatta **LINQ** mimarisi, **SQL** veritabanı(LINQ to SQL) ve **XML** (LINQ to XML) kaynakları üzerindede kullanılabilmektedir. özellikle **IEnumerable<T>** arayüzünü uyarlayan tiplere ait nesne örnekleri için, **Select**, **Where**, **GroupBy**,**Sum**, **Avg, Distinct** ve daha pek çok bilinen sorgulama metodu uygulanabilmektedir. **LINQ** içerisinde yer alan imkanlar göz önüne alındığında, **.Net Framework 1.1**, **2.0 ve 3.0** ile geliştirilmiş pek çok projenin **.Net 3.5**' e aktarılarak bu olanaklardan yararlanabilmeleri için gerekli geçiş hazırlıklarının ciddi anlamda düşünüldüğüde ortadadır. üstelik **Visual Studio 2008**, getirdiği çoklu framework desteği sayesinde **.Net Framework 2.0, 3.0** ve **3.5** arasındaki geçişlerin kolayca yapılabilmesini sağlamaktadır. Bu gibi konular göz önüne alındığında bir geliştirici olarak **LINQ**' in daha önceki kod parçalarında kullanılabileceği yeni yerlerde merak konusu haline gelmektedir. İşte bu makalemizde, **LINQ** sorgularını farklı kod parçalarında kullanmaya çalışıyor olacağız.

LINQ mimarisinin kullanılabileceği alanlar göz önüne alındığında, **Reflection(Yansıma), IO(Dosya giriş/çıkış), Bağlantısız Katman(Disconnected Layer)** sadece bir kaç basit alan olarak ön plana çıkmaktadır. Ancak bu alanlar pek çok uygulamada önemli görevler üstlenmektedir. Söz gelimi yansıma teknikleri ile **IDE** geliştirilmesi(Visual Studio benzeri), **Plug-In**tabanlı uygulamalar yazılması, **nitelik(Attribute)** bazlı olacak şekilde çalışma ortamının organize edilmesi(özellikle deklerafit programlama tekniklerinde) gibi işlevsellikler ön plana çıkmaktadır. Dosyalama işlemleri en basit anlamda **resim işleme programlarından**, **XML** ayrıştırma uygulamalarına kadar pek çok alanda kullanılmaktadır. çok doğal olarak **Ado.Net**mimarisine göre geliştirilen pek çok uygulamada **bağlantısız katman** nesneleri görülebilmektedir. Sadece bu konular bile göz önüne alındığında bazı kod ihtiyaçları için diziler, döngüler ve koşullu ifadelerin çok sık kullanıldığıda göze çarpmaktadır. Ancak **LINQ** sorguları sayesinde bu işlemler çok daha basit bir şekilde gerçekleştirilebilir. Elbetteki genişletme metodlarının üstlendiği yük çerçevesinde söz konusu döngülerin, koşullu ifadelerin ortadan kalkması gibi bir durum

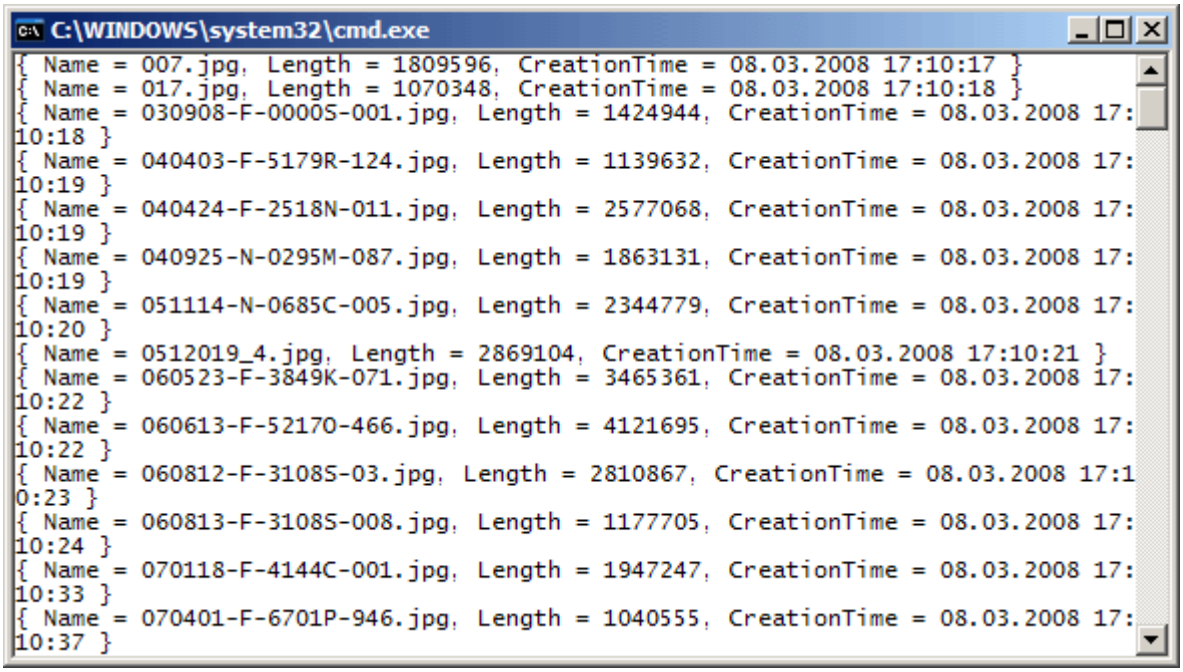
mümkün değildir. Ancak kodun çok daha etkin bir şekilde ve bir sorgulama diline yatkın olaraktan geliştirilmesi önemli bir avantajdır.

Not : *Bilindiği gibi* ***LINQ*** *sorgularında yer alan* ***anahtar kelimeler(keywords)*** *aslında arka planda birer* ***genişletme metoduna(Extension Methods)*** *karşılık gelmektedir. Bu metodlar söz gelimi basit bir arama işlemi için gereken* ***döngüsel*** *veya* ***koşullu ifadeleri*** *kapsülleyerek geliştiricinin üzerinden almaktadır. Bu sayede geliştirici SQL diline yatkın bir şekilde sorgular yazabilmekte ve kodun daha* ***etkili, ölçeklenebilir, anlaşılır*** *bir şekilde geliştirilmesine odaklanabilmektedir.*

Dilerseniz hiç vakit kaybetmeden örneklerimize başlayalım. İlk olarak dosyalama işlemlerini göz önüne alarak ilerleyebiliriz. Söz gelimi, herhangibir klasör içerisinde yer alan **Jpg** uzantılı dosyalardan boyutu **1000 kb** üzerinde olanların tespit edilmesini istediğimizi düşünelim. Bu işlemi **VS 2008** tabanlı bir **Console** Uygulamasında aşağıdaki kod parçası ile gerçekleştirebiliriz.

```
string klasorAdresi= @"C:\Documents and Settings\BurakSenyurt\My Documents\My
Pictures\Google Pictures\";
DirectoryInfo dInfo = new DirectoryInfo(klasorAdresi);
var resimDosyalari = from fInfo in dInfo.GetFiles()
                     where fInfo.Extension == ".jpg" && fInfo.Length >= 1000 * 1024
                        select new
                               {
                                  fInfo.Name
                                  ,fInfo.Length
                                  ,fInfo.CreationTime
                               };
foreach (var dosya in resimDosyalari)
   Console.WriteLine(dosya);
```

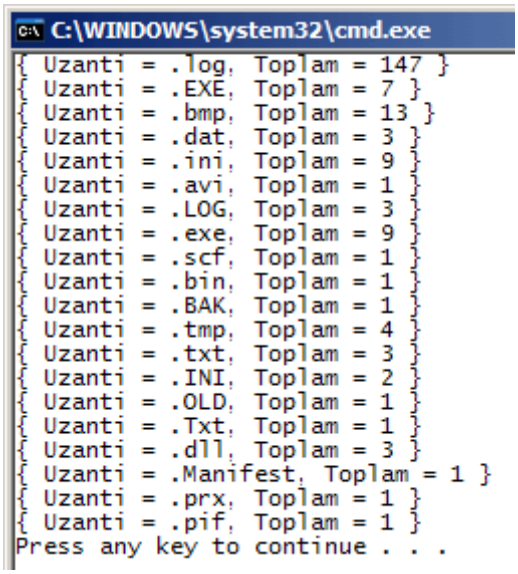
DirectoryInfo sınıfının **GetFiles** metodu **FileInfo** tipinden bir dizi döndürmektedir. Bu dizi bir **Array** tipi olduğu için **LINQ** ile birlikte gelen **genişletme metodlarını(Extension Methods)**kullanabilmektedir. Dolayısıyla **LINQ** ifadesi içerisinde **from**, **select**, **where** gibi anahtar kelimeler kolay bir şekilde ele alınabilmektedir. **FileInfo** dizisi üzerinden **dosya uzantısı(Extension) .jpg** ve**uzunluğu(Length) 1000 Kb üzerinde olanlar** tespit edilirken aynı zamanda **isimsiz bir tip(Anonymous Type)** üretimide gerçekleştirilmekte ve ilgili dosya için **ad(Name), uzunluk(Length)** ve**oluşturulma zamanı(CreationTime)** bilgilerinin yer aldığı yeni bir nesne örneği oluşturulmaktadır. Program kodunun çıktısı örnek klasör için aşağıdaki gibidir.



Eğlenceli değil mi? öyleyse devam edelim. Diyelimki dosyalama işlemleri ile ilgili olaraktan şöyle bir ihtiyacımız oldu;Bir klasör içerisindeki dosyaları **tiplerine göre gruplayıp, her grup içerisinde kaçar adet dosya** bulunduğunu öğrenmek istiyoruz. Bu kodun **LINQ** ifadesini yazmadan önce, **LINQ olmada**n nasıl geliştirilebileceğini düşünmenizi öneririm. LINQ ile bu sorgu aşağıdaki kod parçasında olduğu gibi gerçekleştirilebilir.

```
string adres = @"C:\Windows\";
DirectoryInfo dInfo = new DirectoryInfo(adres);
var dosyaGruplari = from fInfo in dInfo.GetFiles()
                    group fInfo by fInfo.Extension into grp
                       select new
                              {
                                 Uzanti = grp.Key,
                                 Toplam = grp.Count()
                              };

foreach (var dosyaGrubu in dosyaGruplari)
   Console.WriteLine(dosyaGrubu.ToString());
```

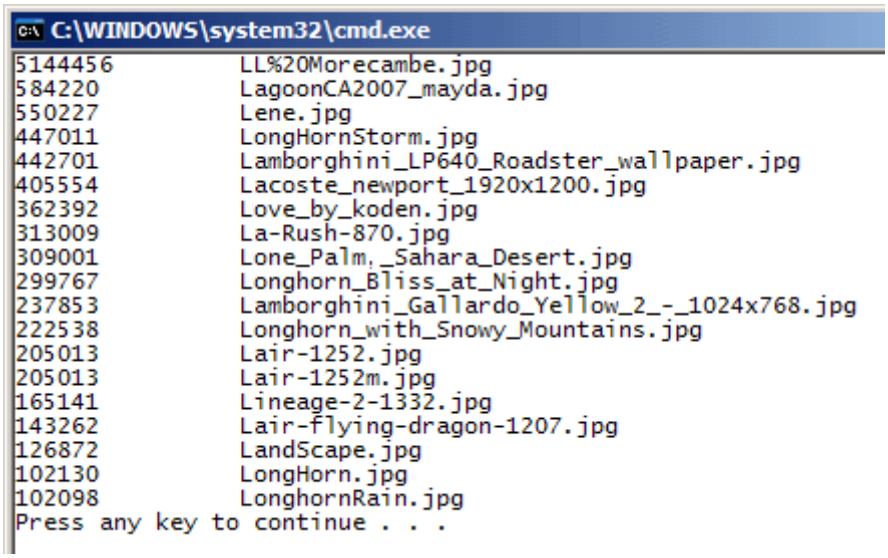
Bu seferki **LINQ** ifadesinde **group by** kullanımı söz konusudur. Group By sayesinde aynen **SQL**' de olduğu gibi gruplama işlemi nesneler üzerinde yapılabilmektedir. örnekte **Windows** klasörü altındaki dosyalar **FileInfo** tipinin **Extension** özelliğine göre gruplanmaktadır. Sonrasında ise gruplanan koleksiyon üzerinden **Count** genişletme metodu kullanılmakta ve her bir tip grubu için kaçar dosya olduğu hesaplanmaktadır. İlk örnekte olduğu gibi yine **isimsiz tip(Anonymous Type)** kullanılarak dosya grubuna ait uzantı ve toplam dosya sayısı bilgileri elde edilmektedir. Sonuç olarak uygulamanın ekran çıktısı aşağıdaki gibi olacaktır.



Şimdide herhangibir klasördeki **jpg** uzantılı dosyalardan **L harfi ile başlayanları boyutlarına** göre **tersten sıralayarak** elde etmek istediğimizi düşünelim. **LINQ** kullanmadığımız takdirde bize en çok sorun çıkartacak noktalardan biriside **tersten sıralama** işlemi olacaktır. Bunu sağlamak için doğal olarak **FileInfo** dizisi üzerinden ters sıralama algoritması uygulanması gerekir. Oysaki LINQ ifadeleri ile bu işlem için gerekli kod parçası aşağıdaki gibi kolayca geliştirilebilir.

```
string klasorAdresi = @"C:\Documents and Settings\BurakSenyurt\My Documents\My Pictures\Google Pictures\";
DirectoryInfo dInfo = new DirectoryInfo(klasorAdresi);

var dosyalar=from fInfo in dInfo.GetFiles()
                where fInfo.Extension==".jpg" && fInfo.Name[0]=='L'
                   orderby fInfo.Length descending
                      select new
                                {
                                   fInfo.Name,
                                   fInfo.Length
                                };

foreach (var dosya in dosyalar)
   Console.WriteLine("{0} \t{1}",dosya.Length,dosya.Name);
```

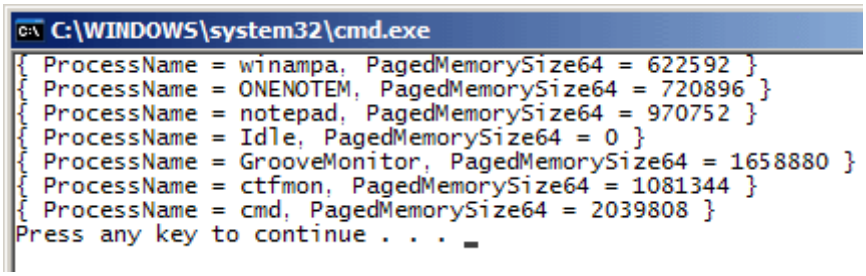
Bu kez **orderby** anahtar kelimesi(ki bu arka planda **OrderBy** genişletme metoduna dönüştürülmektedir) kullanılarak dosyaların boyutlarına göre tersten sıralanması sağlanmıştır. Sonuç olarak kodun ekran çıktısı aşağıdakine benzer olacaktır.



LINQ sorguları dosyalama işlemleri dışında özellikle **reflection(yansıma)** tarafındada etkili bir şekilde kullanılabilir. Yazımızın bundan sonraki kısmındada yansıma teknikleri içerisinde **LINQ** ifadelerini örnekler üzerinde ele almaya çalışacağız. öncelikli olarak **Process**' lerden başlamak taraftarıyım. Bilindiği üzere **.Net** uygulamaları sistem üzerinde açılan **Process**' ler içerisinde ayrı **uygulama alanları(Application Domains)** altına dahil edilirler. Hatta bu uygulama alanları kendi içlerinde, birden fazla*(en az bir tane olmak üzere)* **Thread**' ede sahip olabilirler. Sistem üzerinde **çalışan Process**' lerin yada o anda çalışmakta olan **güncel Process**' in bilgilerini almak için **Process** sınıfının farklı metodları bulunmaktadır. Bizimde aklımıza gelen soru şudur; acaba **sistem üzerinde çalışmakta olan Process' ler içerisinde** sadece **tek bir Thread' e sahip olanlar** hangileridir. Nitekim bilindiği üzere bazı Process' ler kendi içlerinde birden fazla Thread içermektedir. Bu amaçla aşağıdaki gibi bir kod parçası geliştirilebilir.

```
var processes = from prc in Process.GetProcesses()
                where prc.Threads.Count == 1
                   orderby prc.ProcessName descending
                      select new
                             {
                                prc.ProcessName
                                , prc.PagedMemorySize64
                             };
foreach (var process in processes)
   Console.WriteLine(process.ToString());
```

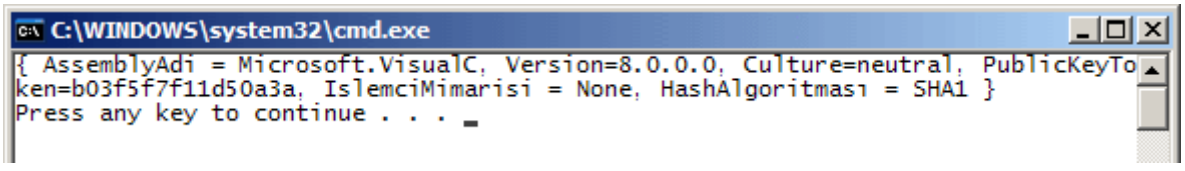
Process sınıfının **static GetProcesses** metodu ile o anda sistemde çalışmakta olan **Process**' ler elde edilmektedir. Sonrasında **where** anahtar kelimesi ile **Threads** özelliği üzerinden **Count**değeri kontrol edilir. **1 olanlar adlarına(ProcessName) göre orderby** anahtar kelimesinden yararlanılarak **tersten sıralanacak** şekilde yeni bir isimsiz tip içerisinde toplanırlar. Bu **isimsiz tip(Anonymous Type)** örnek olarak **Process**' in **adı(ProcessName)** ve **sayfalanmış bellek boyutu(PagedMemorySize64)** değerlerini içermektedir. Sonuç olarak kodun çıktısı, çalışılan sistem üzerinde aşağıdaki gibi olmuştur.



Reflection ile başlamışken hızımızı kesmeyelim ve yeni bir sorgu ile devam edelim. Bu kez şöyle bir ihtiyacımız var; bir assembly' ın **referans ettiği assmebly' lar** içerisinden **versiyonu .Net Framework 2.0 olmayanları** bulmak istiyoruz. Bu tip bir durumda var olan **Assembly**' ın yüklenmesi ve referans ettiği **Assembly**' ların **GetReferencedAssemblies** metodu ile çekilmesi gerekir. Ne tesadüftürki **GetReferencedAssemblies** metodu **AssemblyName** tipinden bir dizi döndürmektedir. Dolayısıyla bu dizi üzerinden **LINQ** ifadeleri kullanılabilmesi olasıdır. Söz konusu ihtiyaç için aşağıdaki gibi bir kod parçası düşünülebilir.

```
AssemblyName[] result1 =
Assembly.LoadFrom(@"C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.EnterpriseServices.dll")
.GetReferencedAssemblies();
var framework2Olmayanlar = from asmb in result1
                           where asmb.Version != new Version(2, 0, 0, 0)
                              select new
                                     {
                                        AssemblyAdi = asmb.FullName
                                        ,IslemciMimarisi = asmb.ProcessorArchitecture
                                        ,HashAlgoritması = asmb.HashAlgorithm
                                     };

foreach (var a in framework2Olmayanlar)
   Console.WriteLine(a.ToString());
```

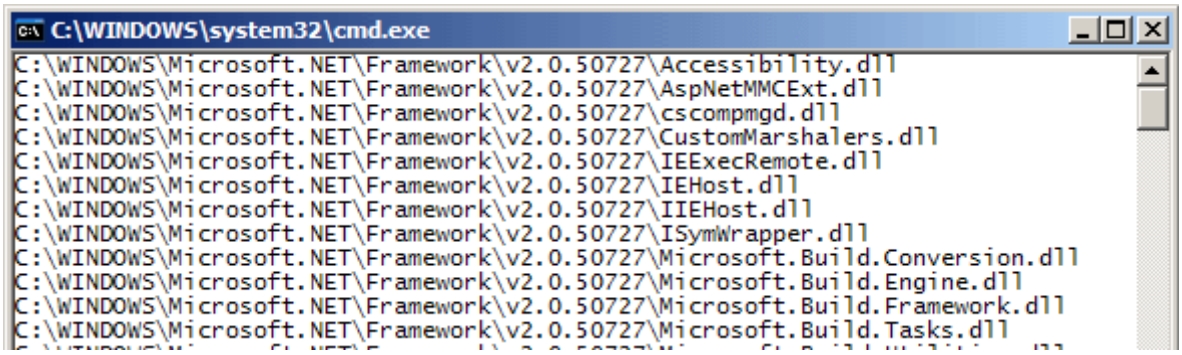
örnek olarak **System.EnterpriseServices.dll** assembly' ı kullanılmaktadır. Sorgu içerisinde dikkat edilecek olursa **GetReferencedAssemblies** metodu ile elde edilen sonuç kümesi üzerinden çekilen her bir **AssemblyName** nesnesinin **Version** özelliğine bakılmaktadır. Sonrasında ise yine bir **isimsiz tip** kullanılarak sadece Assembly' ın **adı(FullName)**, **işlemcimimarisi(ProcessorArchitecture)** ve **hash algoritması(HashAlgorithm)** değerleri toplanmaktadır. örneğin ekran çıktısı aşağıdaki gibi olacaktır.



LINQ sorguları içerisinde bazı yerlerde **harici metodlarında** çağırılması mümkündür. Söz gelimi bir koşul kontrolü için iterasyonun o andaki nesnesinin denetlenmesi gerektiği durumlarda harici metod çağrıları gerekebilir. örneğin **dll** uzantılı dosyalar ile dolu bir klasör içerisinde **.Net Assembly**' ı olarak **yüklenebilenlerin** tespit edilmesini istediğimiz düşünelim. Böyle bir senaryoda **Assembly** sınıfının **static LoadFrom** metodu oldukça işe yarayacaktır. Nitekim söz konusu **dll** herhangibir nedenle yüklenebilen bir **Assembly** değilse**çalışma zamanı istisnası(Runtime Exception)** oluşacaktır. Aşağıdaki kod parçası bu durumu analiz etmek için geliştirilmiştir.

```
class Program
{
   static void Main(string[] args)
   {
      string path = @"C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727";
      DirectoryInfo klasor = new DirectoryInfo(path);
      var assemblyOlanlar = from dosya in klasor.GetFiles("*.dll")
                            where Yuklenebildinmi(dosya.FullName)
                               select dosya;

      foreach (var asmb in assemblyOlanlar)
         Console.WriteLine(asmb.FullName);
   }

   static bool Yuklenebildinmi(string assemblyAdresi)
   {
      try
      {
         Assembly asmbly = Assembly.LoadFrom(assemblyAdresi);
         return true;
      }
      catch
      {
         return false;
      }
   }
}
```

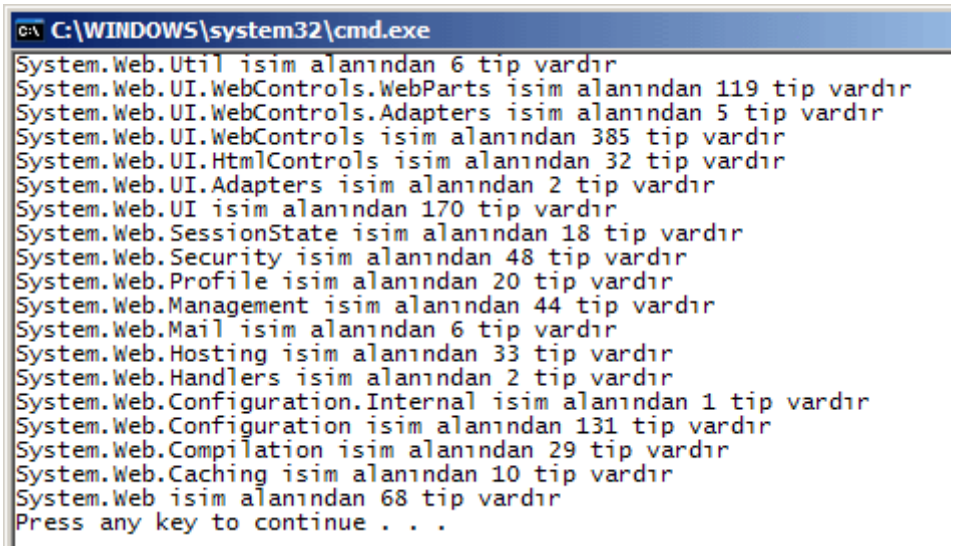
GetFiles metodu ile **dll** uzantılı **FileInfo** dizisi elde edildikten sonra her bir eleman için **Yuklenebildimi** isimli bir metod ile denetleme işlemi gerçekleştirilmektedir. Yuklenebildimi isimli fonksiyon,**Assembly.LoadFrom** metodu işe yarıyorsa **true** değerini, yaramıyorsa **false** değerini döndürmektedir. Buna göre **true** değeri dönen dosyaların yüklenebilen assembly' lar olduğu sonucuna varılmaktadır. Uygulamanın çalışma zamanındaki görüntüsü aşağıdakine benzer olacaktır.



Yine assembly' lar üzerinden **LINQ** sorguları yazmaya devam edelim. örneğin bir **assembly** içerisinden **dışarıya sunulan harici tipler** göz önüne alınsın. Burada işin içerisine gruplama fonksiyonelliğinide katarak, hangi **isim alanı(namespace)** içerisinden kaç adet tipin dışarıya sunulduğu bilgiside elde edilebilir. Bu işi gerçekleştirmek için örnek olarak aşağıdaki gibi bir kod parçası göz önüne alınabilir.

```
Assembly systemAsmb =
Assembly.LoadFrom(@"C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.
Web.dll");
var hariciTipler = from t in systemAsmb.GetExportedTypes()
                   group t by t.Namespace into ng
                      orderby ng.Key descending
                         select new
                                {
                                   IsimAlaniAdi = ng.Key,
                                   TipSayisi = ng.Count()
                                };
foreach (var hariciTip in hariciTipler)
   Console.WriteLine("{0} isim alanından {1} tip vardır", hariciTip.IsimAlaniAdi,
hariciTip.TipSayisi.ToString());
```

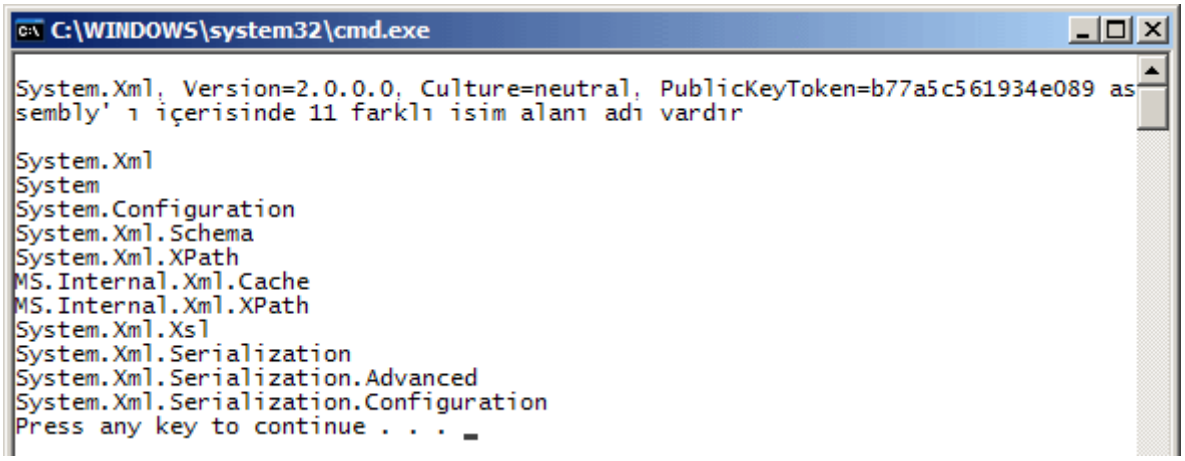
Bu **LINQ** sorgusunda **GetExportedTypes** metodu yardımıyla örnek olarak **System.Web.dll assembly'** ı içerisinden dışarıya sunulmakta olan harici tiplerin listesi **Type** türünden bir dizi olarak elde edilmektedir. Sonrasında ise her tip, **Namespace** özelliğinin değerine göre group anahtar kelimesi yardımıyla ng isimli değişken altında gruplanmaktadır. Gruplanan veriler sonucu elde edilen liste **Namespace** adlarına göre **tersten(descending)** sıralanmaktadır. Bu noktada devreye **orderby** anahtar kelimesi girmektedir. Elde edilen listeden isim alanı adları **Key** özelliği ile ve tip sayılarıda **Count** genişletme metodu ile çekilerek yeni bir **isimsiz tip** altında toplanmaktadır. Kod parçasının çalışmasının sonucu oluşan örnek ekran çıktısı ise aşağıdaki gibidir.



Peki herhangibir **assembly** içerisinde **kaç farklı isim alanı** olduğunu bulmak istersek. Normal şartlarda bu işlem için isim alanı adlarını çektikten sonra bir fonksiyonellik geliştirilmesi gerekmektedir. Oysaki **LINQ** ile birlikte genen **Distinct genişletme metodu** sayesinde söz konusu işlem aşağıdaki kod parçasında olduğu gibi kolayca gerçekleştirilebilir.

```
Assembly systemAsmb =
Assembly.LoadFrom(@"C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.Xml.dll");
var isimAlanlari = (from t in systemAsmb.GetTypes()
                    select t.Namespace).Distinct();
Console.WriteLine("\n{0} assembly' ı içerisinde {1} farklı isim alanı adı vardır",
systemAsmb.FullName,isimAlanlari.Count()-1);
foreach (var isimAlani in isimAlanlari)
   Console.WriteLine(isimAlani);
```

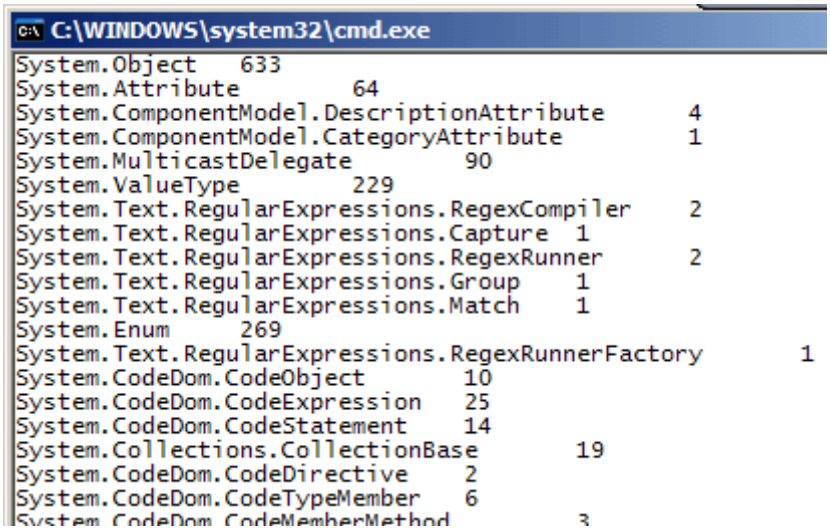
Burada dikkat edilmesi gereken noktalardan biriside **Distinct işlevselliğinin** bir metod olarak **select sorgusunun arkasından** kullanılmasıdır. Buna ek olarak **Count genişletme metodu** ilede **farklı isim alanlarının sayısı** çekilmektedir. örnekte yer alan **System.Xml.dll assembly' ı** için ilgili sonuçlar aşağıdaki gibi olacaktır.



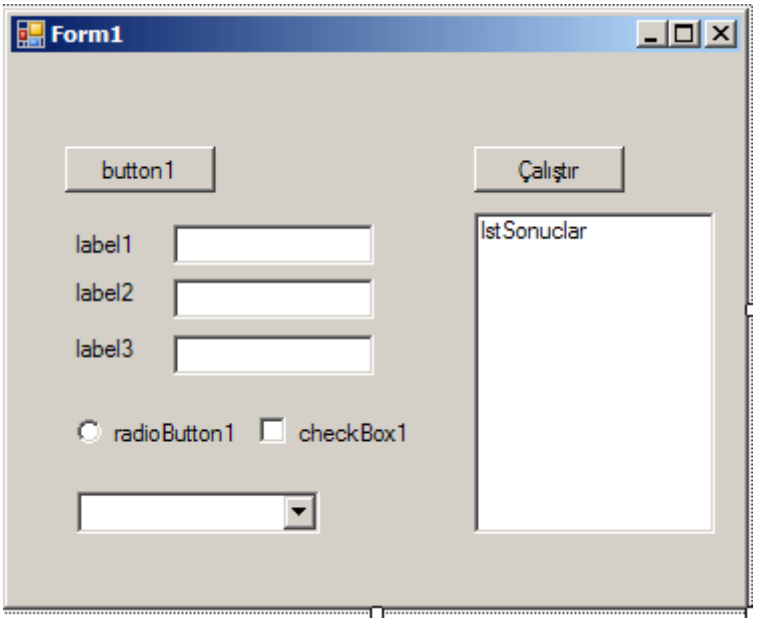
Reflection ile ilişkili olarak **LINQ** sorgularını kullanacağımız son bir örnek ile devam edelim. Bu sefer bir **assembly** içerisinde yer alan **tiplerin toplam sayılarını türedikleri base type' lara göre gruplayarak** elde etmeye çalışıyor olacağız. Bu amaçla, **Type** sınıfının **BaseType** özelliği gruplama işleminde kullanılabilir. Söz gelimi **System.dll assembly**' ı içerisindeki tipleri **BaseType** özelliklerinin değerlerine göre gruplamak istersek aşağıdaki kod parçası yeterli olacaktır.

```
Assembly systemAsmb =
Assembly.LoadFrom(@"C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\System.dl
l");
var tipler = from m in systemAsmb.GetTypes()
                group m by m.BaseType into grp
                   select new
                            {
                               grp.Key
                               ,Toplam = grp.Count()
                            };
foreach (var tip in tipler)
   Console.WriteLine("{0} \t{1}", tip.Key, tip.Toplam.ToString());
```

Bir önceki örnektekine benzer olacak şekilde yine **group by** fonksiyonu kullanılmaktadır. Sonuç olarak üretilen isimsiz tip içerisinde BaseType adı ve toplam tip sayısı değerleri yer almaktadır. örnek kodun çalışma zamanındaki ekran çıktısı aşağıdakine benzer olacaktır.



LINQ(Language INtegrated Query) ifadeleri pek çok dizi tipi ve koleksiyon üzerinde etkin bir şekilde kullanılabildiğinden, akla gelen konulardan biriside görsel uygulamalarda yer alan **Controls**koleksiyonlarıdır. Bir **windows** uygulamasında yada **web** uygulamasında **Container** görevi üstlenen ve bu sebepten **Controls** koleksiyonuna sahip olan nesnel topluluklar üzerinde de **LINQ** sorguları çalıştırılabilir. Bunu basit bir örnek üzerinden inceleyebiliriz. Söz gelimi aşağıdaki ekran görüntüsüde yer alan bir Windows Formumuz olduğunu düşünelim.

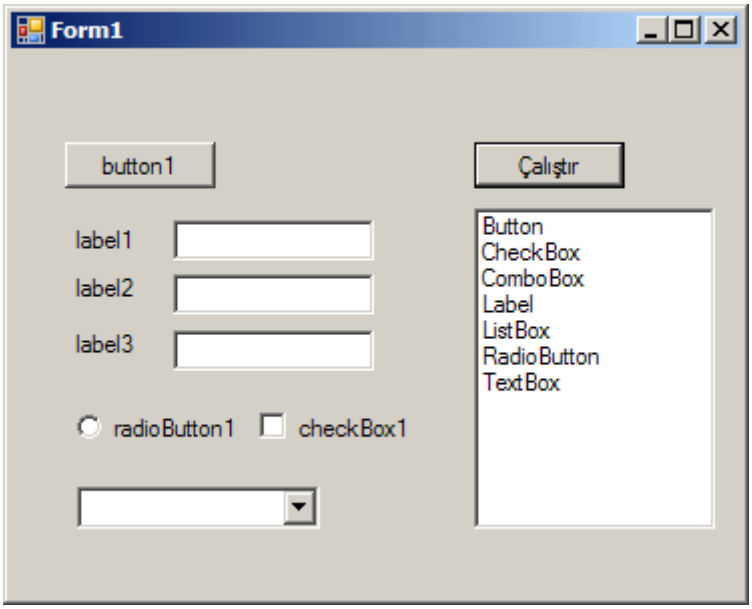


Amacımız şimdilik bu form üzerinde **hangi tipte kontroller** bulunduğunu göstermek. Bu amaçla basit olarak aşağıdaki gibi bir kod parçası yeterli olacaktır.

```
private void button2_Click(object sender, EventArgs e)
{
   lstSonuclar.Items.Clear();

   IEnumerable<Control> kontroller=Controls.Cast<Control>();
```

```
    var farkliTipler = (from kontrol in kontroller
                        select kontrol.GetType()).Distinct().OrderBy(k => k.Name);

    foreach (Type farkliTip in farkliTipler)
        lstSonuclar.Items.Add(farkliTip.Name);
}
```

Bu kod parçasıda belkide en önemli noktalardan biriside **Controls** özelliği üzerinden kullanılan **Cast<T> genişletme metodudur**. Cast<T> metodu kullanılmadığı takdirde **Controls**özelliği üzerinden **Select**, **Where**, **GroupBy** gibi **LINQ** sorgularında önem arz eden fonksiyonelliklere **erişilemediği** görülür. **Cast<T>** metodunun buradaki görevi **Controls** koleksiyonu içerisindeki bileşenleri, parametre olarak verilen generic tipe dönüştürerek **IEnumerable** arayüzünün taşıyabileceği bir nesne topluluğu referansı halinde üretmektir. Böylece LINQ sorguları için gerekli fonksiyonellikler elde edilebilmektedir. Windows formu üzerindeki görsel bileşenler **Control** sınıfından türemektedir. Bu sebepten Cast<T> metodunun generic parametresi**Control** tipindendir. Bu dönüştürme işleminin ardından **Distinct** ve **OrderBy** genişletme metodlarınında yer aldığı bir LINQ sorgusu çalıştırılması mümkün olmaktadır.**Select** sorgusunda, **GetType** metodunun kullanılmasının sebebi tiplerin benzersiz şekilde ele alınmak istemesidir. Sonuç olarak uygulamanın çalışma zamanındaki ekran çıktısı aşağıdakine benzer olacaktır.

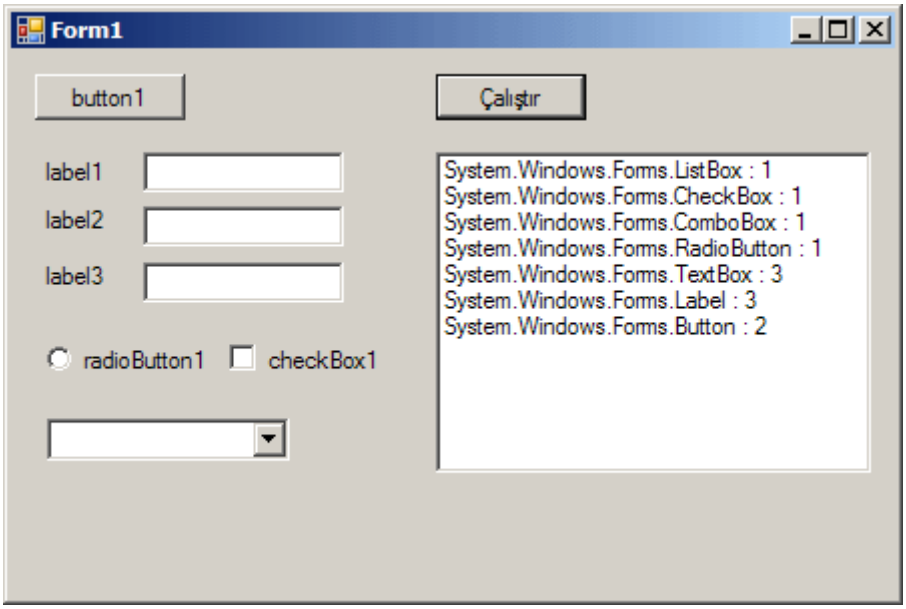


Görüldüğü gibi form üzerinde hangi tipten kontrollerin var olduğu listelenmektedir. Yeni bir sorgu ile devam edelim. Bu sefer form üzerindeki kontrolleri **tiplerine göre** gruplayıp **her bir tipten kaç adet** olduğunu bulmak istediğimizi düşünelim. Bu basit gruplama işleminin kodu aşağıdaki gibi geliştirilebilir.

```
private void button2_Click(object sender, EventArgs e)
{
    lstSonuclar.Items.Clear();
```

```
   IEnumerable<Control> kontroller=Controls.Cast<Control>();

   var farkliTipler = from kontrol in kontroller
                      group kontrol by kontrol.GetType() into grp
                         select new
                                {
                                   KontrolAdi=grp.Key
                                   ,Toplam=grp.Count()
                                };

   foreach (var farkliTip in farkliTipler)
      lstSonuclar.Items.Add(String.Format("{0} :
{1}",farkliTip.KontrolAdi,farkliTip.Toplam.ToString()));
}
```

Bu sefer gruplama işlemi **Control** tipinin **GetType** metoduna göre yapılmaktadır. Uygulama kodunun ekran çıktısı aşağıdaki gibi olacaktır.

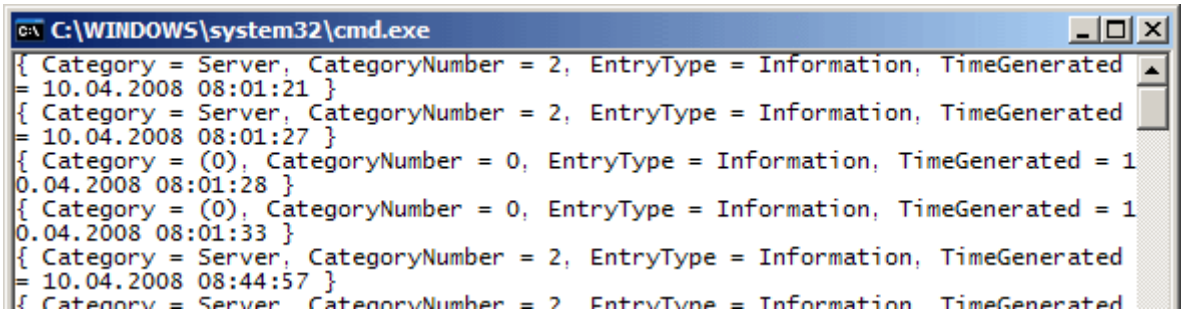


Cast<T> metodu doğrudan **LINQ genişletme metodlarının** kullanılamadığı pek çok senaryoda ele alınabilir. Söz gelimi aşağıdaki kod parçası çok basit olarak **Application Log**altındaki girişlerden **programın çalıştırıldığı gün içerisinde** üretilenlerin çekilmesini sağlamaktadır.

```
EventLog logs = new EventLog("Application", ".", "");
IEnumerable<EventLogEntry> entries = logs.Entries.Cast<EventLogEntry>();

var girisler = from entry in entries
               where entry.TimeGenerated.Day == DateTime.Now.Day
                  select new
                         {
```

```
                    entry.Category,
                    entry.CategoryNumber,
                    entry.EntryType,
                    entry.TimeGenerated
                };

foreach (var giris in girisler)
    Console.WriteLine(giris.ToString());
```

Bu kod parçasında kullanılan **Cast<T>** genişletme metodu geriye, **IEnumerable<EventLogEntry> arayüzü(interface)** tarafından taşınacak bir nesne topluluğu referansı döndürmektedir.**IEnumerable<T>** arayüzüne ulaşıldığı içinde LINQ sorgusu kolay bir şekilde ele alınmış ve aşağıdaki ekran çıktısının üretilmesi sağlanmıştır.



Cast<T> metodu ile benzer özelliğe sahip bir diğer önemli metodda **OfType<T> genişletme metodudur**. Bu metod bir nesne topluluğu üzerinde, **generic parametre tipine göre filtreleme** yapılabilmesini ve geriye LINQ sorgularının uygulanabileceği **bir IEnumerable<T>** referansı döndürülmesini sağlamaktadır.

***Not : Cast<T>** metodu ile **OfType<T>** metodu benzer işlevselliğe sahip görünmekle birlikte arada önemli farklar vardır. **OfType<T>** metodu temel olarak generic parametre tipine göre bir**filtreleme** yapmakta iken, **Cast<T>** metodu generic parametre tipine **dönüştürme** yapmaktadır. Bu sebepten dönüştürme yapılamayacağı durumlarda **Cast<T>** metodu, çalışma zamanında **InvalidCastException** istisnası üretilmesine neden olur. Oysaki **OfType<T>** metodu bu durumu tamamen görmezden gelir ve diğer nesneden devam eder. **OfType<T>** kendi içerisinde**is** anahtar kelimesini kullanarak tip kontrolü yapmaktayken, **Cast<T>** doğrudan dönüştürme adımını uygular.*

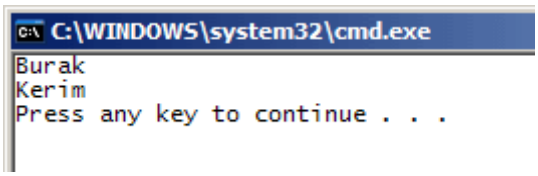
Şimdi **OfType<T>** metodunu örnek bir senaryo üzerinden ele almaya çalışalım. örneğin uygulamamızda kullandığımız .Net Framework 2.0 ile geliştirilmiş bir kütüphane olsun. Bu kütüphane içerisinde yer alan metodlardan bazılarınında **ArrayList** gibi tür güvenli olmayan koleksiyonlar döndürdüğünü düşünelim. Referansta bulunan uygulamanın .Net 3.5 tabanlı olduğu düşünülecek olursa, gelen koleksiyon nesneleri üzerinden **LINQ** sorguları çalıştırılması istenebilir. Bu noktada **OfType<T>** metodu oldukça işe yarayacaktır. Söz konusu senaryoyo ele almak için aşağıdaki tipi içeren bir **sınıf kütüphanesi(Class Library)** olduğunu düşünelim.

```
public class Yardimci
{
   public ArrayList ListeyiAl()
   {
      ArrayList liste = new ArrayList();
      liste.Add("Burak");
      liste.Add("Bili");
      liste.Add("Behçet");
      liste.Add("Necdet");
      liste.Add("Kerim");
      liste.Add("Mayk");
      liste.Add(19.90);
      liste.Add(10);
      liste.Add(true);
      liste.Add(false);
      liste.Add('C');
      return liste;
   }
}
```

Kod parçasında kasıtlı olarak **ArrayList** içerisine farklı tipte veriler atılmıştır.
Eğerki **LINQ** sorgusunda bu metoddan dönen değerler içerisinden sadece **string** tabanlı olanları ele almak istiyorsak, **OfType<T>** metodunu aşağıdaki kod parçasında olduğu gibi kullanabiliriz.

```
GenelIslemler.Yardimci yrdm = new GenelIslemler.Yardimci();

var besHarfliler = from nesne in yrdm.ListeyiAl().OfType<string>()
                   where nesne.Length == 5
                      select nesne;

foreach (string nesne in besHarfliler)
   Console.WriteLine(nesne);
```

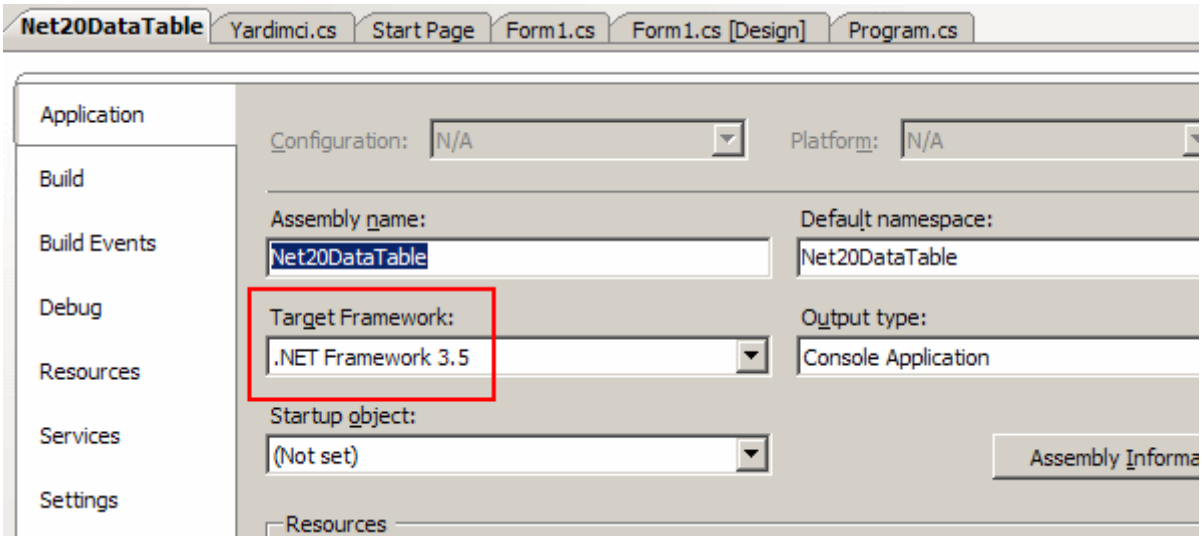
OfType<string> metodu buradaki kullanıma göre **ListeyiAl** fonksiyonundan gelen **ArrayList** içerisindeki tüm nesnelerde, **is** kontrolünü yaparak sadece **String** olanları geriye döndürmektedir. Sonrasında nesnelerin karakter uzunluğu kıyaslanarak 5 ise çekilmektedir. Program kodunun çıktısı aşağıdaki gibi olacaktır.



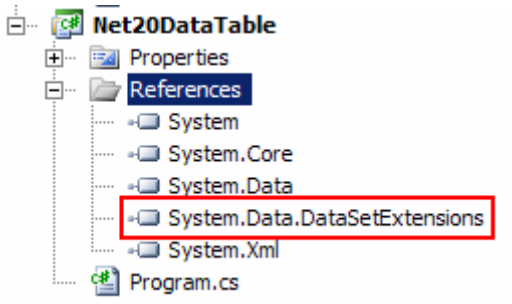
Yazımızda son olarak **.Net Framework 2.0** ile yazılmış ve **DataTable** nesnelerini kullanan bir uygulamayı **.Net 3.5**' e taşıyarak basit **LINQ** sorgularını nasıl ele alabileceğimizi incelemeye çalışacağız. *(LINQ sorgularının işlevselliğinin ön plana çıktığı vakalarda, var olan .Net uygulamaları **.Net 3.5** versiyonuna terfi edilmek durumdan kalabilir.)* Bu amaçla ilk olarak **.Net Framework 2.0** ile geliştirilmiş ve test amacıyla aşağıdaki kodlara sahip bir **Console** uygulamamız olduğunu düşünelim.

```
using System;
using System.Data;
using System.Data.SqlClient;

namespace Net20DataTable
{
   class Program
   {
      static void Main(string[] args)
      {
         DataTable tbl = null;

         using (SqlConnection conn = new SqlConnection("data
source=.;database=AdventureWorks;integrated security=SSPI"))
         {
            SqlDataAdapter adapter = new SqlDataAdapter("Select
ProductId,Name,ListPrice,Class,SellStartDate,ProductSubCategoryId From
Production.Product", conn);
            tbl = new DataTable("Products");
            adapter.Fill(tbl);
         }
      }
   }
}
```

Kod, **AdventureWorks** isimli **SQL Server 2005** veritabanına bağlanmakta ve **Production** şemasındaki **Product** tablosundan bir kaç alanı çekmektedir. çekilen veri kümesi işlenilmek üzere bir**DataTable** nesnesi içerisinde toplanmaktadır. çok doğal olarak uygulama **.Net Framework 2.0** tabanlı olduğundan, **LINQ** ifadelerinin **DataTable** üzerinden uygulanması*(veya başka bağlantısız katman nesneleri üzerinden)* mümkün değildir. Eğer elimizde **Visual Studio 2008** var ise yapılması gerekenler çok basittir. öncelikli olarak proje özelliklerinden(Properties) **Application**sekmesine geçilmeli ve **Target Framework** seçeneği **.Net Framework 3.5** olarak değiştirilmelidir.



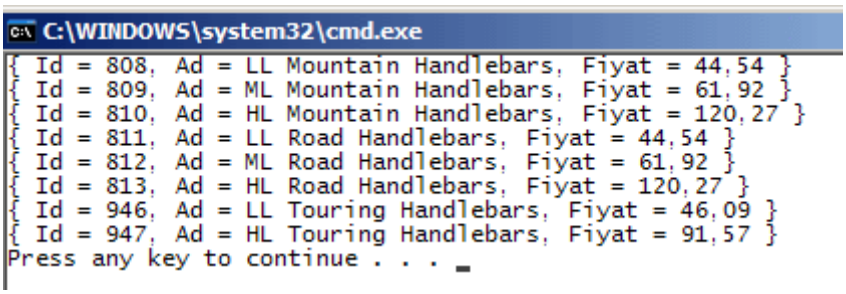
Bu işlemin ardından uygulamanın bir kere daha derlenmesinde yarar vardır. *(Söz konusu adımların ardından **System.Core.dll assembly**' ının projeye hemen referans edildiğide görülebilir.)* Artık **LINQ** sorgularının yazılmasına başlanabilir. **DataTable** için bu sorguların uygulanabilmesi için **AsEnumerable** metodunun erişilebilir olması gerekmektedir. Ancak bu genişletme metoduna şu anda erişilemediği görülmektedir. Bunun sebebi **System.Data.DataSetExtensions.dll assembly**' ının projeye referans edilmemiş olmasıdır. Dolayısıyla öncelikle bu assembly' ın referans edilmesi gerekmektedir.



Artık uygulamada yer alan DataTable üzerinde LINQ sorguları çalıştırılabilir. İşte bir örnek;

```
var altKategorisi4OlanUrunler = from row in tbl.AsEnumerable()
                                where row["ProductSubCategoryId"].ToString() == "4"
                                  select new
                                         {
                                           Id = Convert.ToInt16(row["ProductId"]),
                                           Ad = row["Name"].ToString(),
                                           Fiyat = Convert.ToDouble(row["ListPrice"])
                                         };

foreach (var urun in altKategorisi4OlanUrunler)
   Console.WriteLine(urun.ToString());
```

Bu kod parçasında görülen **LINQ** sorgusuna göre **DataTable** içerisinden **ProductSubCategoryId** alanının değeri **4** olanların **ProductId,Name,ListPrice** kolonlarının verilerinden oluşan yeni bir **isimsiz tip(Anonymous Type)** topluluğu elde edilmektedir. Kodun ekran çıktısı aşağıdakine benzer olacaktır.



Görüldüğü gibi **LINQ** sorguları **.Net Framework** içerisinde pek çok farklı alanda uygulanabilmektedir. **Reflection, IO, Windows Forms Controls, Application Log, DataTable,** eski bir uygulamadan gelen **ArrayList** bu yazıda ele alınan basit bir kaç alandır. LINQ sorguları **Office** ürünlerinde dahi kullanılabilmektedir. Söz gelimi **Outlook** içerisindeki kontaklar **LINQ** sorguları ile filtrelenebilir. örnekleri arttırmak ve yaymak mümkündür. Ancak unutulmaması gereken noktalardan biriside bu işlemlerin yapılması için **LINQ** sorgularının olmasının **zorunlu olmadığıdır**. öyleki LINQ sorgularıda özünde, **.Net Framework 3.5** ile gelen **genişletme metodlarını(Extension Methods)** yoğun bir şekilde ele almaktadır. Bir başka deyişle LINQ olmadanda metodlar yardımıyla bu istekler karşılanabilir. Diğer taraftan LINQ sorgularının getirdiği dil esnekliği, kullanım kolaylığı, anlaşılabilirlik göz ardı edilmemelidir. Her geliştirici kullandığı **programlama dili yardımıyla nesneler üzerinden SQL benzeri sorgu ifadeleri yazabilmek** ister. LINQ bu imkanı sağlayarak önemli bir açığı kapatmaktadır. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

LINQveReflectionArastirma.rar (97,49 kb)

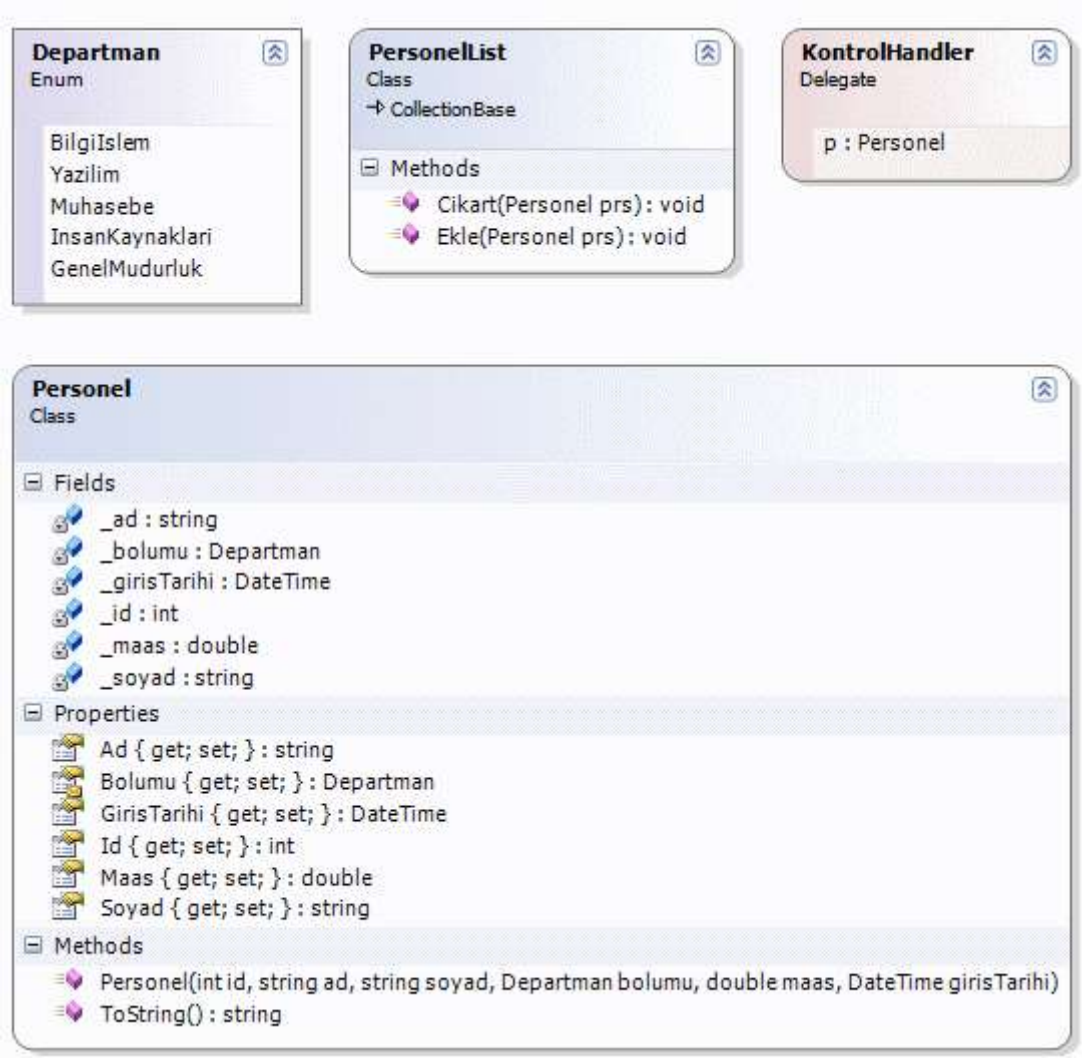
C# 3.0 - Derinlemesine Lambda İfadeleri (2008-03-31T21:28:00)

c# 3.0,

C# programlama dilinin 3ncü versiyonu ile birlikte gelen önemli yeniliklerden biriside **lambda(=>)** operatörüdür. Bu operatörün kullanıldığı ifadeler yardımıyla **temsilci(delegate)**oluşturulması, kod bloğunun yazılması, sonuçların alınması ve **tip tahmini(Type Inference)** gibi işlemlerin tek seferde gerçekleştirilmesi mümkündür. Bu sebepten dolayı**LINQ(Language INtegrated Query)** sorgularında yer alan **genişletme metodlarında(Extension Methods)** büyük öneme sahiptir. Ne varki lambda operatörünü kavramak için ona olan ihtiyacın nereden doğdunu bilmek ve nasıl bu operatöre ulaşıldığını anlamak gerekmektedir. En iyi başlangıç noktası elbetteki C# dilinin ilk versiyonudur. Bu

yazımızda lambda operatörünün getirdiği avantajları görmeye çalışırken derinlemesinede inceliyor olacağız.

Herşeyden önce **C# 1.0** versiyonunda kullanıcı tanımlı bir tipe ait koleksiyonlar üzerinde bazı sorgulamalar yapmak istediğimizi düşünelim. C# 1.0 versiyonunda **generic** mimari kavramı yoktur. Bu sebepten generic olarak türden bağımsız ve **.Net Framework** içerisinde önceden tanımlanmış olan koleksiyonlar bulunmamaktadır. Bunun yerine elemanları her zaman **object**türünden olan **ArrayList**, **Stack**, **Queue** gibi **Collection** bazlı koleksiyonlar ile **Hashtable** ve **SortedList** gibi **Dictionary** bazlı koleksiyonlar mevcuttur. Eğer sadece bizim istediğimiz tip ile çalışacak **kuvvetle türlendirilmiş bir koleksiyon(Strongly Typed Collection)** kullanmak istersek **CollectionBase** veya **DictionaryBase abstract** sınıflarından türetme yolunu tercih edebiliriz. Böylece sadece istenilen tipler ile çalışacak bir koleksiyonumuz olur.*(Lakin bu koleksiyon **tip güvenli-type safety** olmasına rağmen gereksiz **boxing ve unboxing** işlemlerini engellemez.)* Şu an için asıl amaç bu koleksiyon içerisinde yer alan tipler üzerinde farklı sorgular çalıştırabilmektir. Söz gelimi bir personelin bilgirini tanımlayan sınıfa ait bir koleksiyon içerisinden, çalışanın maaşına, adına, giriş tarihine göre sorgular yaparak alt koleksiyonların çekilmesini sağlayacak fonksiyonelliklerin olması istenebilir. Hatta bu metodarın sayısının arttırılmasınada olanak verecek şekilde esnek bir yapının geliştirilmesi istenebilir. Dolayısıyla alt koleksiyonları elde edebilmek için geliştirilen ortak bir metodun kullanacağı koşulsal fonksiyonelliklerin işaret edilebilmesi son derece yararlı olur. İşte bu noktada **temsilciler(Delegates)** devreye girmektedir. Bu cümleler ile tam olarak neye sebebiyet verdiğimizi görmek üzere **C# 1.0** dilinin yeteneklerinin kullanıldığı aşağıdaki program kodu göz önüne alınabilir.



Kod içeriği;

```
using System;
using System.Collections;

namespace DotNet1Deyken
{
   enum Departman
   {
      BilgiIslem
      ,Yazilim
      ,Muhasebe
      ,InsanKaynaklari
      ,GenelMudurluk
   }

   class Personel
   {
```

```
int _id;
string _ad;
string _soyad;
Departman _bolumu;
double _maas;
DateTime _girisTarihi;

public DateTime GirisTarihi
{
   get { return _girisTarihi; }
   set { _girisTarihi = value; }
}

public string Soyad
{
   get { return _soyad; }
   set { _soyad = value; }
}

public double Maas
{
   get { return _maas; }
   set { _maas = value; }
}

internal Departman Bolumu
{
   get { return _bolumu; }
   set { _bolumu = value; }
}

public string Ad
{
   get { return _ad; }
   set { _ad = value; }
}

public int Id
{
   get { return _id; }
   set { _id = value; }
}

public Personel(int id, string ad, string soyad, Departman bolumu, double
maas,DateTime girisTarihi)
```

```
        {
            Id = id;
            Ad = ad;
            Soyad = soyad;
            Bolumu = bolumu;
            Maas = maas;
            GirisTarihi = girisTarihi;
        }
        public override string ToString()
        {
            return String.Format("{0} {1} {2} {3} {4} {5}", Id.ToString(), Ad,
Soyad.ToUpper(), Bolumu.ToString(), Maas.ToString("C2"),
GirisTarihi.ToShortDateString());
        }
    }

    class PersonelList
        : CollectionBase
    {
        public void Ekle(Personel prs)
        {
            List.Add(prs);
        }
        public void Cikart(Personel prs)
        {
            List.Remove(prs);
        }
    }

    delegate bool KontrolHandler(Personel p);

    class Program
    {
        static bool DepartmaniIKmi(Personel p)
        {
            return p.Bolumu == Departman.GenelMudurluk;
        }
        static bool AdininBasHarfiBmi(Personel prs)
        {
            return prs.Ad[0] == 'B';
        }
        static bool Maas1000Uzerindemi(Personel prs)
        {
            return prs.Maas > 1000;
        }
```

```
    static PersonelList Bul(PersonelList liste,KontrolHandler handler)
    {
        PersonelList sonucListesi = new PersonelList();
        foreach (Personel prs in liste)
        {
            if (handler(prs))
                sonucListesi.Ekle(prs);
        }
        return sonucListesi;
    }

    static void Listele(PersonelList liste)
    {
        foreach (Personel prs in liste)
            Console.WriteLine(prs.ToString());
        Console.WriteLine("");
    }

    static void Main(string[] args)
    {
        PersonelList calisanlar = new PersonelList();

        calisanlar.Ekle(new Personel(1000,"Mayk","Hemır",
Departman.BilgiIslem,1050,new DateTime(1979,10,1)));
        calisanlar.Ekle(new Personel(1001,"Büyük","Başkan",
Departman.GenelMudurluk,53000,new DateTime(1989,2,3)));
        calisanlar.Ekle(new Personel(1002,"EmSi","Hemmır",
Departman.GenelMudurluk,13500,new DateTime(1990,2,4)));
        calisanlar.Ekle(new Personel(1003,"Tombul","Raydır",
Departman.InsanKaynaklari,2250,new DateTime(1994,8,5)));
        calisanlar.Ekle(new Personel(1008,"Şirine","Şirin", Departman.BilgiIslem,900,new
DateTime(1991,3,6)));
        calisanlar.Ekle(new Personel(1006,"Burak","Selim",
Departman.InsanKaynaklari,2250,new DateTime(1976,7,3)));
        calisanlar.Ekle(new Personel(1004,"Osvaldo","Nartayyo",
Departman.Muhasebe,3500,new DateTime(1975,6,3)));
        calisanlar.Ekle(new Personel(1005,"Higuin","Kim", Departman.Yazilim,1250,new
DateTime(1974,4,2)));
        calisanlar.Ekle(new Personel(1007,"Karim","Cabbar", Departman.Yazilim,750,new
DateTime(1975,2,7)));
        calisanlar.Ekle(new Personel(1011, "Billl", "Geytis", Departman.Yazilim, 650, new
DateTime(1976, 3, 8)));

        // Departmanı Insan Kaynakları olanların bulunması
        PersonelList sonuclar1=Bul(calisanlar, new KontrolHandler(DepartmaniIKmi));
```

```
            // İsminin baş harfi B olanların bulunması
            PersonelList sonuclar2 = Bul(calisanlar, new
KontrolHandler(AdininBasHarfiBmi));

            // Maaşı 1000 YTL üzerinde olanların bulunması
            PersonelList sonuclar3 = Bul(calisanlar, new
KontrolHandler(Maas1000Uzerindemi));

            Listele(sonuclar1);
            Listele(sonuclar2);
            Listele(sonuclar3);
        }
    }
}
```

öncelikli olarak bu uzun **Console** uygulaması kodlarında neler olduğuna bir bakalım. **Personel** isimli sınıf bir çalışanın Id' sini, adını, soyadını, işe giriş tarihini, maaşını, departmanını ve maaşını tutacak şekilde tanımlanmıştır. Bu sınıfa ait örneklerin içeriklerinin kolay bir şekilde **string** olarak alınabilmesi içinde **ToString** metodu Personel tipi içerisinde **ezilmiştir(override)**. Personelin bölümü, **Departman** isimli bir **enum** sabiti ile belirtilmektedir. **PersonelList** isimli sınıf **CollectionBase abstract** sınıfından türetilmektedir. Bu nedenle**Collection** tabanlı bir koleksiyondur. Konunun kolay anlaşılabilmesi için sadece **Ekle** ve **Cikart** isimli iki fonksiyonelliğe sahiptir. Bu metodlar sadece **Personel** tipinden parametreler almaktadır. Buda zaten**PersonelList** isimli koleksiyonun **tip güvenli(Type Safety)** olmasını sağlamaktadır. Dikkatimiz çeken tiplerden biriside **KontrolHandler** isimli **temsilcidir(delegate)**.

Not: Temsilcileri(delegates) *metodları işaret edebilecek şekilde kullanılabilen* ***.Net tipidir****. Bir temsilci işaret edebileceği metodun* ***parametrik yapısı*** *ile* ***dönüş tipinide*** *belirtmektedir.*

Söz konusu temsilci, **Personel** tipinden bir parametre alan ve geriye bool değer döndüren metodları işaret edecek şekilde tanımlanmıştır. Bu temsilcinin tek bir tasarım amacı vardır. Buna göre, bir Personel nesne örneğinin herhangibir şartı sağlayıp sağlamadığına dair **true** veya **false** değer döndürecek bir metodun işaret edilmesini sağlamaktadır. **Peki neden böyle bir temsilciye ihtiyacımız vardır?** Bu sorunun cevabını **Bul** isimli fonksiyon vermektedir.

```
static PersonelList Bul(PersonelList liste,KontrolHandler handler)
{
    PersonelList sonucListesi = new PersonelList();
    foreach (Personel prs in liste)
    {
        if (handler(prs))
            sonucListesi.Ekle(prs);
```

```
    }
    return sonucListesi;
}
```

Dikkat edilecek olursa **Bul** metodu geriye **PersonelList** tipinden bir nesne örneği döndürmektedir. Bu nesne örneği metod içerisinde oluşturulmaktadır. Oluşturulma işlemi sırasında ise belirli bir şarta bakılmaktadır. Nitekim bu şartın ne olduğu belli değildir. Ancak şartın sonucunun alınmasını sağlayan metodu işaret edebilecek **KontrolHandler** tipinden bir **temsilci,**fonksiyona parametre olarak gelmektedir. Temsilci nesne örneği **çalışma zamanında(runtime)** ilgili fonksiyonu işaret edeceğinden, **if** ifadesi içerisindeki çağrı aslında o andaki **Personel** nesne örneği için koşul metoduna doğru yapılan bir yürütmeden başka bir şey değildir.

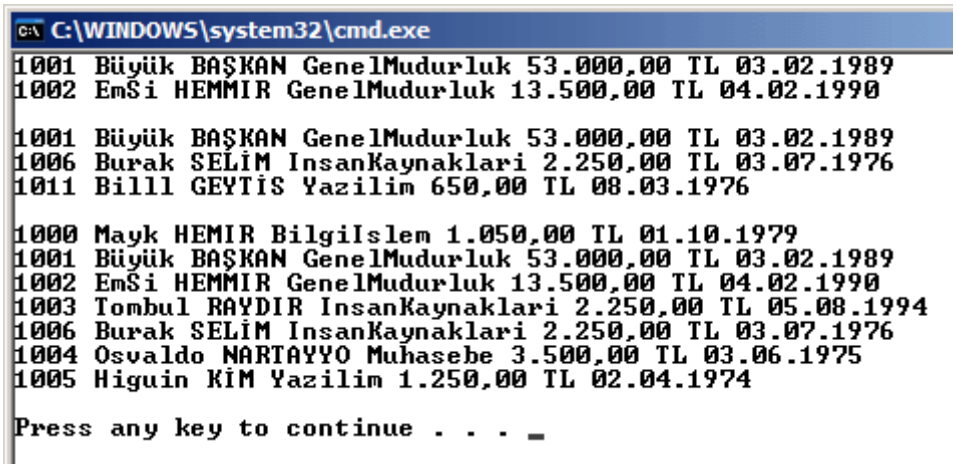
Artık tek yapılması gereken şartları sağlayacak metodların yazılması ve sonrasında ise Bul fonksiyonelliğinin kullanılarak ilgili sonuç kümesinin alınmasıdır. örneğin **IK departmanında çalışan personelin** bulunabilmesi için **DepartmanIKmi** isimli bir metod geliştirilmiştir.

```
static bool DepartmaniIKmi(Personel p)
{
    return p.Bolumu == Departman.GenelMudurluk;
}
```

Bu metod basitçe gelen **Personel** nesne örneğinin **Bolumu** özelliğine bakmakta ve geriye **true** yada **false** değerini döndürmektedir. Başka bir örnek olarak **maaşı 1000 YTL üzerinde olanların** elde edilmesi istenebilir. Bunun içinde **Maas1000Uzerindemi** isimli bir metod geliştirilmiştir.

```
static bool Maas1000Uzerindemi(Personel prs)
{
    return prs.Maas > 1000;
}
```

Tahmin edileceği üzere bu fonksiyonda, **KontrolHandler** temsilcisinin belirttiği yapıya uygun bir şekilde tasarlanmıştır. Bu yaklaşımlar göz önüne alındığında koleksiyon içerisinde istenildiği gibi filtreleme yapılabileceği görülmektedir. Tek şart temsilciye uygun tipte bir karşılaştırma fonksiyonelliğinin var olmasını sağlamaktır. Uygulamanın çalışma zamanındaki çıktısı aşağıdaki gibi olacaktır.



Görüldüğü gibi maaşı 1000 YTL üzerinde olanlar, IK departmanında çalışanlar ve isminin baş harfi B olanlar kolay bir şekilde elde edilmektedir.

Bu yaklaşım her ne kadar kolay ve anlaşılır olsada bazı sıkıntılar olduğu ortadadır. Herşeyden önce **Bul** fonksiyonunun parametresi olan temsilci tipinin işaret edeceği metod bloklarının ayrı ayrı yazılıyor olma şartı vardır. Diğer taraftan söz konusu mimari şuanda sadece **PersonelList** isimli koleksiyona uygulanabilecek şekilde tasarlanmıştır. Hatta**KontrolHandler** isimli temsilci dahi sadece **Personel** tipleri ile çalışabilecek şekilde ele alınabilmektedir. Oysaki bu yapının herhangibir koleksiyon tipi içerisinde ele alınabilmesi sağlanabilmelidir. Bu, ilgili yapının sadece uygulama bazlı değil Framework bazlı olacak şekilde genişletilebilmesini sağlayacaktır ki bu oldukça önemlidir. Elbette bu iş sanıldığı kadar kolay değildir. Nitekim türden bağımsız olacak şekilde fonksiyonel yapıların olması şarttır. Peki öyleyse olaya **C# 2.0** açısından bakmaya çalışalım. Bu sefer elimizde **isimsiz metodlar(Anonymous Methods)** ve **generic mimari** gibi oldukça güçlü kozlar yer almaktadır. Dolayısıyla yukarıdaki örnek mimari modeli **C# 2.0** içerisinde aşağıdaki şekilde ele alınabilir.

```
using System;
using System.Collections.Generic;

namespace DotNet2Deyken
{
   enum Departman
   {
      BilgiIslem
      ,Yazilim
      ,Muhasebe
      ,InsanKaynaklari
      ,GenelMudurluk
   }

   class Personel
   {
```

```
int _id;
string _ad;
string _soyad;
Departman _bolumu;
double _maas;
DateTime _girisTarihi;

public DateTime GirisTarihi
{
   get { return _girisTarihi; }
   set { _girisTarihi = value; }
}

public string Soyad
{
   get { return _soyad; }
   set { _soyad = value; }
}

public double Maas
{
   get { return _maas; }
   set { _maas = value; }
}

internal Departman Bolumu
{
   get { return _bolumu; }
   set { _bolumu = value; }
}

public string Ad
{
   get { return _ad; }
   set { _ad = value; }
}

public int Id
{
   get { return _id; }
   set { _id = value; }
}
```

public Personel(int id, string ad, string soyad, Departman bolumu, double maas, DateTime girisTarihi)

```
        {
            Id = id;
            Ad = ad;
            Soyad = soyad;
            Bolumu = bolumu;
            Maas = maas;
            GirisTarihi = girisTarihi;
        }
        public override string ToString()
        {
            return String.Format("{0} {1} {2} {3} {4} {5}", Id.ToString(), Ad,
Soyad.ToUpper(), Bolumu.ToString(), Maas.ToString("C2"),
GirisTarihi.ToShortDateString());
        }
    }

    // Koşul kontrolünü yapabilecek metodları içeren tür bağımsız temsilci(Generic
Delegate) tanımı
    delegate bool KontrolHandler<T>(T parametre);

    class Program
    {
        // generic tipten oluşan koleksiyon üzerinden alt küme çekme işlemini üstlenen metod
        static List<T> Bul<T>(List<T> liste,KontrolHandler<T> handler)
        {
            List<T> sonuclar = new List<T>();
            foreach (T eleman in liste)
                if (handler(eleman)) // Generic temsilcinin işaret edeceği karşılaştırma metodu
çağırılır.
                    sonuclar.Add(eleman);
            return sonuclar;
        }

        // Generic Listeleme fonksiyonu
        static void Listele<T>(List<T> liste)
        {
            foreach (T t in liste)
                Console.WriteLine(t.ToString());
            Console.WriteLine("");
        }

        static void Main(string[] args)
        {
            List<Personel> calisanlar = new List<Personel>();
```

```
        calisanlar.Add(new Personel(1000, "Mayk", "Hemır", Departman.BilgiIslem, 1050,
new DateTime(1979, 10, 1)));
        calisanlar.Add(new Personel(1001, "Büyük", "Başkan",
Departman.GenelMudurluk, 53000, new DateTime(1989, 2, 3)));
        calisanlar.Add(new Personel(1002, "EmSi", "Hemmır",
Departman.GenelMudurluk, 13500, new DateTime(1990, 2, 4)));
        calisanlar.Add(new Personel(1003, "Tombul", "Raydır",
Departman.InsanKaynaklari, 2250, new DateTime(1994, 8, 5)));
        calisanlar.Add(new Personel(1008, "Şirine", "Şirin", Departman.BilgiIslem, 900,
new DateTime(1991, 3, 6)));
        calisanlar.Add(new Personel(1006, "Burak", "Selim", Departman.InsanKaynaklari,
2250, new DateTime(1976, 7, 3)));
        calisanlar.Add(new Personel(1004, "Osvaldo", "Nartayyo", Departman.Muhasebe,
3500, new DateTime(1975, 6, 3)));
        calisanlar.Add(new Personel(1005, "Higuin", "Kim", Departman.Yazilim, 1250,
new DateTime(1974, 4, 2)));
        calisanlar.Add(new Personel(1007, "Karim", "Cabbar", Departman.Yazilim, 750,
new DateTime(1975, 2, 7)));
        calisanlar.Add(new Personel(1011, "Billl", "Geytis", Departman.Yazilim, 650, new
DateTime(1976, 3, 8)));

        // Anonymous Method yardımıyla arama işlemleri yapılır
        // Bul metodunun ikinci parametrelerinin nasıl verildiğine dikkat edelim

        // Insan Kaynakları departmanında çalışanların bulunması
        List<Personel> IKCalisanlari=Bul<Personel>(calisanlar,delegate(Personel p)
                                                   {
                                                      return p.Bolumu ==
Departman.Yazilim;
                                                   }
                                                  );

        // Şubat ayında işe girenlerin bulunması
        List<Personel> SubatAyindaBaslayanlar = Bul<Personel>(calisanlar,
delegate(Personel p)
                                                   {
                                                      return p.GirisTarihi.Month == 2;
                                                   }
                                                  );

        //Departmanı Yazilim olanlardan Maaşı 1000 YTL üzerinde olanların bulunması
        List<Personel> MaasiVeDepartmaninaGore = Bul<Personel>(calisanlar,
delegate(Personel p)
                                                   {
                                                      return (p.Maas >= 1000 &&
```

```
p.Bolumu == Departman.Yazilim);
                                                    }
                                                );
            Listele<Personel>(IKCalisanlari);
            Listele<Personel>(SubatAyindaBaslayanlar);
            Listele<Personel>(MaasiVeDepartmaninaGore);
        }
    }
}
```

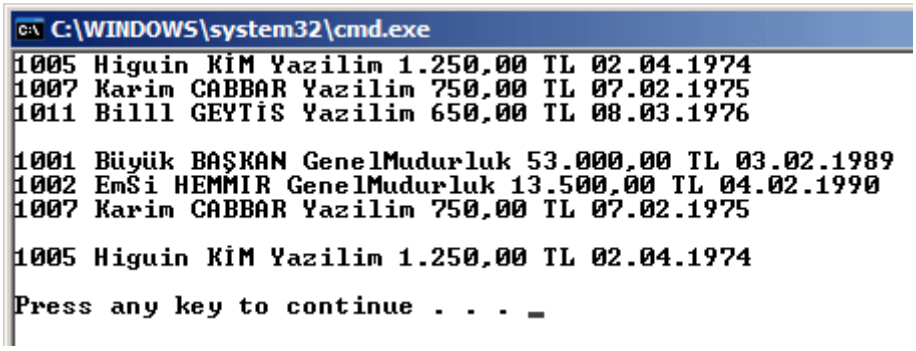
Bu uzun kod parçasında bir önceki versiyona göre en büyük farklılıklar **generic** koleksiyon ile **generic** ve **isimsiz metod(Anonymous Method)** kullanımlarıdır. Dikkat edilecek olursa herhangibir tipteki **List** koleksiyonu üzerinde arama işlemi yapılabilmesini sağlayacak şekilde **generic** bir **Bul** metodu yer almaktadır. Dahada önemlisi, koleksiyon içerisindeki elemanların kıyaslama işlemlerinin yapılacağı metodları işaret edebilecek olan temsilcide **generic** olarak tanımlanmıştır. Bu sayede **T** tipindeki bir **List** koleksiyonu içerisinde **Bul** metodunun kullanılabilmesi ve o tip için bir koşullandırma yapılabilmesi sağlanmaktadır. Fakat bütün bunlara rağmen en çok dikkate değer kısımlardan biriside, **isimsiz metodların** kullanımıdır. Bu sebepten dolayı bir önceki örnekte olduğu gibi, ayrı ayrı karşılaştırma metodlarının yazılmasına gerek kalmamaktadır. Tam aksine **Bul** metodunun kullanıldığı yerlerde ikinci parametrelerde**isimsiz metod** kullanılarak koşul deyimlerinin aynı ifade içerisinde tanımlanabilmeside sağlanmıştır. örneğin Şubat ayında işe giren personelin bulunabilme sürecini göz önüne alalım. Burada **Bul** metodu, **calisanlar** isimli **generic** koleksiyondaki **Personel** nesne örneklerini **tek tek dolaşmalı**, **GirisTarihi** özellikleri üzerinden **Month** değerlerinin **2** olup olmadığına bakmalı ve eğer öyleyse bunları yeni bir koleksiyonda birleştirerek geriye döndürmelidir. **İsimsiz metodlar** yardımıyla bu iş aşağıda görüldüğü gibi tek bir ifadede sağlanabilir.

```
List<Personel> SubatAyindaBaslayanlar = Bul<Personel>(calisanlar, delegate(Personel p)
                                                    {
                                                        return p.GirisTarihi.Month ==
2;
                                                    }
                                                );
```

Dikkat edilecek olursa **delegate** anahtar kelimesi burada **KontrolHandler** tipini işaret etmektedir. Dahası kod yazılırken generic mimarinin, **Visual Studio IDE**' si içerisinde aşağıdaki şekilde ele alındığı görülmektedir.

Görüldüğü gibi **Bul** metoduna generic parametre olarak **Personel** tipi verildiğinde, **liste** isimli **List<T>** koleksiyonu ve **handler** isimli **KontrolHandler** temsilcisi otomatik olarak bu tiple çalışacak hale gelmektedir. Buda Bul metodunun generic yapısından kaynaklanmaktadır.

Sonuç olarak program çıktısı aşağıdaki gibi olacaktır.

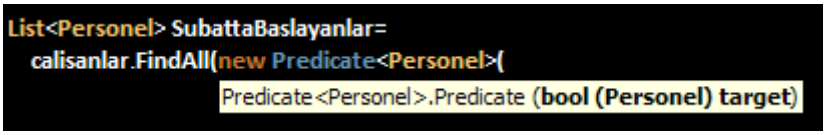


Elbette **Bul** fonksiyonu geliştirici tarafından yazılmış bir metoddur. Oysaki **.Net Framework 2.0** özellikle **List<T>** koleksiyonları üzerinde bu tip filtreleme ve arama işlemlerinin gerçekleştirilmesi amacıyla hazır **Predicate** temsilcisini kullanan **Find**, **FindAll, Exists, FindIndex, FindLast, FindLastIndex, RemoveAll** gibi metodlar içermektedir. *(Burada hazır bir temsilcinin olması geliştiricinin uygulamadan bağımsız olacak şekilde, Framework' ün kullanıldığı her yerde söz konusu koşullandırma metodlarını işaret ederek, başka hazır **CLR tipi** metodlarına parametre olarak verebileceği anlamına da gelmektedir.)* Dolayısıyla yukarıda geliştirilen örnek, **.Net Framework 2.0**' ın tipleri sayesinde aşağıdaki hale getirilebilir.

```
class Program
{
   static void Listele<T>(List<T> liste)
   {
      foreach (T t in liste)
         Console.WriteLine(t.ToString());
      Console.WriteLine("");
   }

   static void Main(string[] args)
   {
      List<Personel> calisanlar = new List<Personel>();

      #region Test Verileri

      // Test verilerinin girildiği kodlar

      #endregion
```

```
        List<Personel> BHarfliler =
                        calisanlar.FindAll(delegate(Personel p)
                                           {
                                              return p.Ad[0] == 'B';
                                           }
                                          );
        List<Personel> SubattaBaslayanlar=
                        calisanlar.FindAll(delegate(Personel p)
                                           {
                                              return p.GirisTarihi.Month == 2;
                                           }
                                          );
        List<Personel> GenelMudurlukCalisanlari =
                        calisanlar.FindAll(delegate(Personel p)
                                           {
                                              return p.Bolumu == Departman.GenelMudurluk;
                                           }
                                          );

        Listele<Personel>(BHarfliler);
        Listele<Personel>(SubattaBaslayanlar);
        Listele<Personel>(GenelMudurlukCalisanlari);
    }
}
```

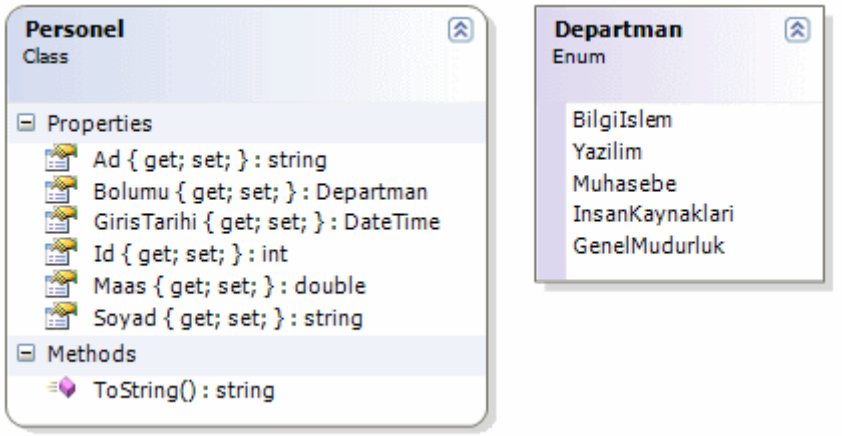
Bu kez **delegate** anahtar kelimesi **FindAll** metodunun istediği **Predicate<T>** temsilcisini işaret etmektedir. Kod yazımı sırasında **intellisense** ile bu açık bir şekilde görülmektedir.



Mimaride halen daha eksiklikler vardır. özellikle veri tabanı uygulamalarında yer alan sorgulama tekniklerinin, programatik tarafta ifade edilme zorlukları bilinmektedir. çok basit olarak düşünüldüğünde, bir **veritabanı tablosunun** program tarafında **Entity** olarak **sınıf bazlı** ifade edilmesi sonrasında geliştiricilerin beklentisi, veri sorgulama dili esnekliğinin nesnel olarakta sağlanabilmesidir. Bir başka deyişle bilinen **select**, **where**, **group by**, **distinct**, **sum** vb... sorgulama kelimelerinin, program tarafındaki nesnel varlıklar üzerinde de uygulanamabilmesi istenmektedir. İşte bu **LINQ** mimarisinin geliştirilmesinin en büyük nedenlerinden de birisidir. Peki elde bulunan imkanlar ile bu nasıl sağlanabilir? Yoksa dile yeni bir takım kolaylaştırıcı özelliklerin entegre edilmesimi gerekmektedir?

Herşeyden önce son örnekte yer alan **FindAll** metodu ile, bir koleksiyon üzerinde filtreleme yapılabildiği görülmektedir. Bu bir anlamda **Where** ve **Select** gibi ifadelerin bir

karşılığı olarak göz önüne alınabilir. Ancak bu yeterli değildir. Yeni tipler geliştirmeden, var olan **.Net Framework** tiplerine **FindAll** metoduna benzer fonksiyonel yenilikleri ilave edebilmek gerekmektedir. İşte bu noktada **Extension** metodlar devreye girerek özellikle **IEnumerable<T>** uyarlamalı tiplerin genişletme metodları ile **LINQ** mimarisine destek verebilmesi sağlanmaktadır. İşin içerisinde yine temsilci(delegate) tipleri rol almaktadır. **LINQ** mimarisine destek verebilmek için pek çok **temsilci** tipi geliştirilmiş ve **Framework** içerisine dahil edilmiştir. Bu kadar ilerlemeden önce, yazıya konu olan örneğin **C# 3.0** içerisindeki geliştirilme şekline bakmakta yarar vardır. Nitekim ilk hedef Lambda ifadelerinin rolünü kavramaktır. *(Bu seferki örnek Visual Studio 2008 üzerinde .Net Framework 3.5 seçilerek yapılmıştır.)*




```
using System;
using System.Collections.Generic;
using System.Linq;

namespace DotNet3Nokta5Deyken
{
   enum Departman
   {
      BilgiIslem
      ,Yazilim
      ,Muhasebe
      ,InsanKaynaklari
      ,GenelMudurluk
   }
   class Personel // Bu sınıfta otomatik özellikler(Automatic Property) kullanılmıştır.
   {
      public int Id { get; set; }
      public string Ad { get; set; }
      public string Soyad { get; set; }
      public Departman Bolumu { get; set; }
      public double Maas { get; set; }
      public DateTime GirisTarihi { get; set; }
```

```
        public override string ToString()
        {
            return String.Format("{0} {1} {2} {3} {4} {5}", Id.ToString(), Ad,
Soyad.ToUpper(), Bolumu.ToString(), Maas.ToString("C2"),
GirisTarihi.ToShortDateString());
        }
    }
    class Program
    {
        static void Main(string[] args)
        {
            // Object Initializers' dan faydalanılmıştır.
            List<Personel> calisanlar = new List<Personel>()
            {
                new Personel(){Id=1000, Ad="Mayk", Soyad="Hemır",
Bolumu=Departman.BilgiIslem, Maas=1050, GirisTarihi=new DateTime(1979, 10, 1)},
                new Personel(){Id=1001, Ad="Büyük", Soyad="Başkan",Bolumu=
Departman.GenelMudurluk,Maas= 53000,GirisTarihi= new DateTime(1989, 2, 3)},
                new Personel(){Id=1002, Ad="EmSi", Soyad="Hemmır",
Bolumu=Departman.GenelMudurluk, Maas=13500, GirisTarihi=new DateTime(1990, 2,
4)},
                new Personel(){Id=1003, Ad="Tombul", Soyad="Raydır",
Bolumu=Departman.InsanKaynaklari, Maas=2250, GirisTarihi=new DateTime(1994, 8,
5)},
                new Personel(){Id=1008, Ad="Şirine", Soyad="Şirin",Bolumu=
Departman.BilgiIslem, Maas=900, GirisTarihi=new DateTime(1991, 3, 6)},
                new Personel(){Id=1006, Ad="Burak", Soyad="Selim",
Bolumu=Departman.InsanKaynaklari,Maas= 2250, GirisTarihi=new DateTime(1976, 7,
3)},
                new Personel(){Id=1004, Ad="Osvaldo", Soyad="Nartayyo",
Bolumu=Departman.Muhasebe, Maas=3500,GirisTarihi= new DateTime(1975, 6, 3)},
                new Personel(){Id=1005, Ad="Higuin", Soyad="Kim",Bolumu=
Departman.Yazilim, Maas=1250,GirisTarihi= new DateTime(1974, 4, 2)},
                new Personel(){Id=1007, Ad="Karim", Soyad="Cabbar",
Bolumu=Departman.Yazilim, Maas=750, GirisTarihi=new DateTime(1975, 2, 7)},
                new Personel(){Id=1011, Ad="Billl", Soyad="Geytis",
Bolumu=Departman.Yazilim, Maas=650,GirisTarihi= new DateTime(1976, 3, 8)}
            };

            // B ile başlayanlar
            var AdiBIleBaslayanlar = calisanlar.FindAll((Personel p) => (p.Ad[0] == 'B'));

            // Departmanı Yazilim olanlar (Burada type inference söz konusu)
            var YazilimDepartmaniCalisanlari = calisanlar.FindAll(p => p.Bolumu ==
Departman.Yazilim);
```

```
        //Giris yılı 1976 öncesi olanlar çekilirken başka bir metod çağırılıyor.
        var GirisYili1976OncesiOlanlar = calisanlar.FindAll(
                                                p =>{
                                                   if (p.GirisTarihi.Year < 1976)
                                                   {
                                                      PrimArttir(p);
                                                      return true;
                                                   }
                                                   else
                                                      return false;
                                                }
                                            );

        Listele<Personel>(AdiBIleBaslayanlar);
        Listele<Personel>(YazilimDepartmaniCalisanlari);
        Listele<Personel>(GirisYili1976OncesiOlanlar);
    }

    private static void PrimArttir(Personel p)
    {
        Console.WriteLine("\t"+p.Ad+" "+p.Soyad.ToUpper()+" için prim arttırım talebi");
    }

    static void Listele<T>(IEnumerable<T> liste)
    {
        foreach (T t in liste)
            Console.WriteLine(t.ToString());
        Console.WriteLine("");
    }
  }
}
```

örnekte **C# 3.0** ile birlikte gelen pek çok yenilik kullanılmaya çalışılmıştır. **var** anahtar kelimesi, koleksiyon ve Personel nesnelerininin başlatılması**(object initialization)**, otomatik özellikler**(automatic properties)** gibi. Ancak yazımızda odaklanacağımız nokta => operatörü ve içerisinde yer aldığı ifadelerdir. Dikkat edilecek olursa **FindAll** metodlarının içerisinde kullanılan parametrelerde => operatörleri yer almaktadır. İlk metod çağrısı aşağıdaki gibidir.

```
var AdiBIleBaslayanlar = calisanlar.FindAll((Personel p) => (p.Ad[0] == 'B'));
```

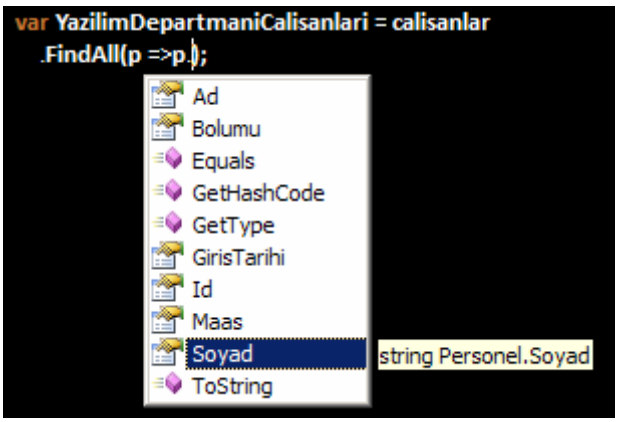
Burada => operatörünün sol tarafında **Personel** tipinden bir değişken tanmı yer almaktadır. Operatörün sağ tarafında ise yine parantezler içerisinde p değişkeninin Ad özelliğinin ilk karakterine bakılmaktadır. Daha önceki örneklerden hatırlanacağı gibi **FindAll** metodu **Predicate** temsilcisini parametre olarak almaktadır. Bu temsilci

geriye **bool** değer döndüren ve **generic** tipte parametre alan metodları taşıyabilemektedir. Bu sebepten **Lambda** operatörünün sağ tarafında yer alan kod parçasının **bool** tipinden bir değer döndürüyor olması şarttır.**Predicate** temsilcisinin işaret edeceği metodun alacağı parametre ise operatörün sol tarafında belirtilmektedir. Peki burada Lambda operatörü neyi sağlamaktadır? Nitekim aynı amaç için**isimsiz metod(anonymous method)** kullanımıda mümkündür. Hatta isimsiz metod kullanmadanda yapılabildiği görülmektedir. Ne farki fonksiyonel programlama ortamlarına bakıldığında bu tip ifadelerin yaygın bir şekilde ele alındığı görülmüştür. Bununla beraber => operatörü burada, temsilcinin örneklenmesi, işaret edeceği metoda parametre aktarılması, uygun tipte sonuç üreten bir kod bloğunun yazılması operasyonlarının tek bir ifade içerisinde gerçekleştirilmesini sağlamaktadır.

İkinci kullanım şekli ilkinden biraz daha farklıdır.

```
var YazilimDepartmaniCalisanlari = calisanlar.FindAll(p => p.Bolumu ==
Departman.Yazilim);
```

Bu sefer dikkat çekici nokta => operatörünün sol ve sağ tarafındaki deyimlerde **parantez** kullanılmayışı değildir. Dikkat edilmesi gereken nokta operatörün sol tarafında **sadece p** yazılmasıdır. Oysaki bir önceki kullanım şeklinde temsilcinin işaret ettiği metoda aktarılacak olan parametrenin tipi açık bir şekilde belirtilmiştir. Burada **tip tahmini(type inference)** kavramı devreye girmektedir. öyleki **FindAll** metodunun, List koleksiyonunun **generic** yapısına göre kullanacağı tipin **Personel** olma olasılığı muhtemeldir. Bu son derece doğaldır nitekim **calisanlar**değişkeni **List<Personel>** tipinden bir koleksiyonu taşımaktadır. Buna göre **compiler**, **p** değişkeninin **Personel** tipinden olacağını tahmin eder. Bu tahminin ne kadar tutarlı olduğu **Visual Studio IDE**' si içerisinde **intellisense** özelliği ile açık bir şekilde görülebilmektedir.



Görüldüğü gibi lambda operatörünün sağ tarafında p değişkeni kullanılmak istendikten sonra, tahmin edilen tipin üyeleri ekrana gelmektedir.

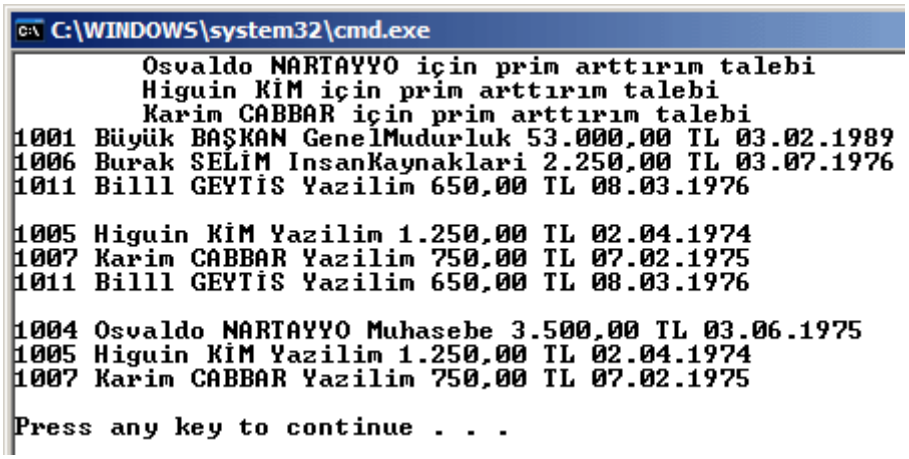
üçüncü kullanım şekli ise aşağıdaki gibidir.

```
var GirisYili1976OncesiOlanlar = calisanlar.FindAll(
                                                    p =>{
```

```
                if (p.GirisTarihi.Year < 1976)
                {
                    PrimArttir(p);
                    return true;
                }
                else
                    return false;
            }
        );
```

Burada ise tek fark lambda operatörünün sağ tarafında yer alan deyimlerde normal kod bloklarınında geliştirilebiliyor olmasıdır. Bir başka deyişle **Predicate** temsilcisinin istediği şekilde**bool** değer döndürmek dışında, metod çağrısı gibi farklı deyimlerde yapılabilmektedir.

Geliştirilen son örnek çalıştırıldığında aşağıdaki ekran görüntüsünde yer alan sonuçların alındığı görülmektedir.



Lambda operatörleri özellikle **LINQ** içerisinde yer alan genişletme metodlarında sıklıkla kullanılmaktadır. Bilindiği üzere **LINQ** sorgularının desteklenmesi için**Enumerable***(**System.Core.dll** assembly' ı içerisinde yer alan **System.Linq** isim alanında yer almaktadır)* isimli **static** sınıf içerisine çok sayıda **genişletme metodu(Extension Methods)** dahil edilmiştir. Bu sınıfın en büyük özelliklerinden biriside içerisinde yer alan metodlarının **IEnumerable<T>** türevli tipleri genişletmesidir. Diğer taraftan söz konusu sınıf içerisinde
çoğunlukla **Func** isimli **generic temsilci** kullanılmaktadır. Bu temsilcisinin farklı versiyonları aşağıdaki gibidir.

```
delegate TResult Func<TResult>(T arg)
delegate TResult Func<T, TResult>(T arg)
delegate TResult Func<T1,T2, TResult>(T1 arg1, T2 arg2)
delegate TResult Func<T1,T2,T3 TResult>(T1 arg1, T2 arg2, T3 arg3)
delegate TResult Func<T1,T2,T3,T4, TResult>(T1 arg1, T2 arg2, T3 arg3, T4 arg4)
```

Func bir temsilci olduğu için, kullanılacağı her yerde lambda operatörleri ele alınabilir. Buna çok basit olarak aşağıdaki kod parçasını örnek gösterebiliriz.

```
double sonuc = calisanlar
                    .Where<Personel>(p => p.Bolumu == Departman.Yazilim)
                      .Sum<Personel>(p => p.Maas);
Console.WriteLine(sonuc.ToString("C2"));

int sonuc2 = calisanlar.Aggregate(0,(toplam,p) => p.Maas>2000?toplam+=1:toplam);
Console.WriteLine(sonuc2.ToString());
```

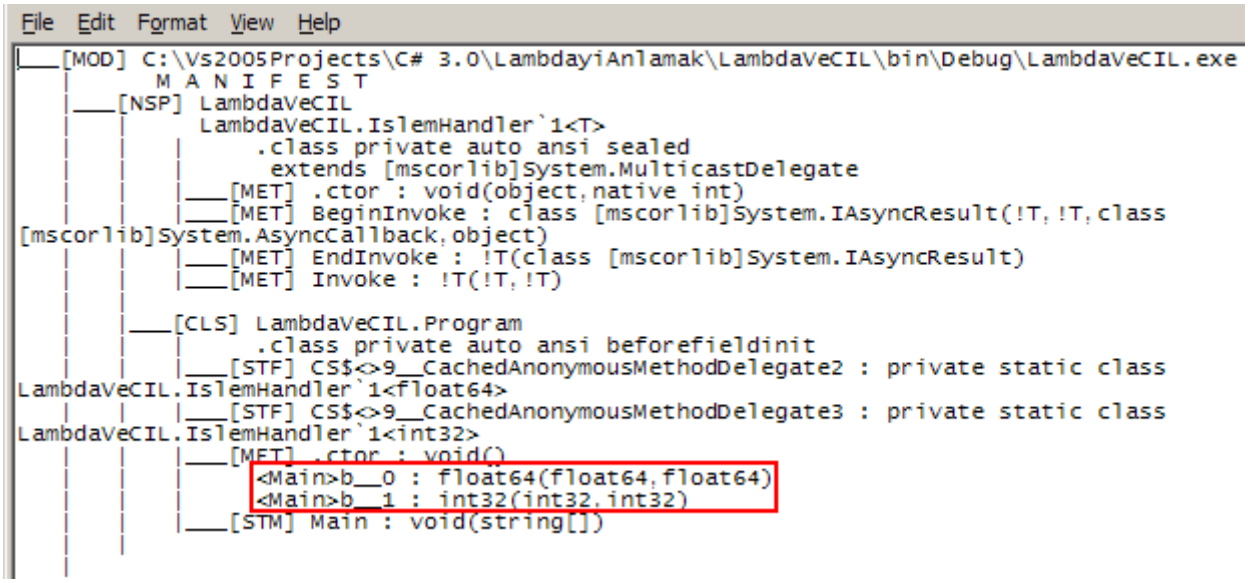
İlk kullanımda **departmanı yazılım olan personelin maaşlarının toplamı** bulunmuştur. İkinci kullanımda ise lambda operatörünün iki parametreyi birden temsilciye gönderdiği görülmektedir. Dikkat edilecek olursa operatörün sol tarafında **toplam** ve **p** isimli değişkenler tanımlanmaktadır.*(Bu iki parametre bildirimi eğer parantezler içersinde yazılmassa derleme zamanı hatası alınır. Dolayısıyla lambda operatörünün sol tarafında birden fazla parametre olacaksa bunların parantezler içerisine alınması gerekmektedir.)* Buradaki toplam değişkeni yine tahmin edilerek **int** tipinden belirlenmiştir. Bu tarz bir kullanım normaldir nitekim **Func** temsilcisinin bu şekilde iki parametre ile çalışan versiyonu mevcuttur. Buna göre **Aggregate**metodu, **personelin maaşı 2000 YTL üzerinde olanlar** var ise, toplam değişkeninin değerini **1 arttırarak** geriye döndürmektedir. *(Bu işlem için **Sum** metoduda kullanılabilir. Aggregate genişletme metodunun yazılmasının amacı **Sum**, **Count**, **Max**, **Min, Avg** gibi standart gruplama fonksiyonları dışındaki gereksinimlerin karşılanmasıdır.)*

Artık **Lambda** operatörünün kullanımı hakkında fikir sahibi olduğumuzu sanıyorum. Şimdi diğer noktalara değinmeye çalışalım. Söz gelimi **lambda** operatörünün yer aldığı ifadeler **IL(Intermediate Language)** tarafında nasıl yorumlanmaktadır? Bu noktada lambda operatörünün, isimsiz metod kullanımı ile aynı IL çıktısını verdiğini söyleyebiliriz. örnek olarak aşağıdaki kod parçasını göz önüne alalım.

```
using System;

namespace LambdaVeCIL
{
   delegate T IslemHandler<T>(T T1,T T2);

   class Program
   {
      static void Main(string[] args)
      {
         IslemHandler<double> hnd = (x, y) => x + y;

         IslemHandler<int> hnd2=
            delegate(int a,int b){
```

```
                return a + b;
            };
        }
    }
}
```

Yukarıdaki kod parçasında yer alan **IslemHandler** isimli **generic temsilci** tipi, **T** türünden iki **parametre** alan ve yine **T** türünden **sonuç** üreten metodları işaret edebilecek şekilde tasarlanmıştır. **hnd** değişkeni oluşturulurken **lambda** ifadesinden, **hnd2** oluşturulurkende **isimsiz metoddan(anonymous method)** yararlanılmıştır. Bu kodun **IL** çıktısına **ildasm**aracı ile bakıldığında ağaç yapısının aşağıdaki gibi olduğu görülür.*(Ağaç yapısının kolay bir şekilde elde edilmesi için Dump TreeView seçeneğinden yararlanılmıştır.)*



Program içerisinde **b__0** ve **b__1** adları ile tanımlanmış iki adet metod olduğu görülmektedir. Tahmin edileceği üzere bu iki üye, **lambda** ifadesi ve isimsiz metod kullanımı sonrası oluşturulmuş metodlardır. Bir başka deyişle lambda operatörü kullanıldığında aynen isimsiz metodlarda olduğu gibi **IL** tarafında iş yapan metod oluşturulmaktadır. Bu metodlar derleyici tarafından oluşturulan gizli metodlardır. **Compiler** tarafından oluşturuldukları için **CompilerGenerated niteliği(Attribute)** ile imzalanmışlardır.
Eğer **Main** metodunun **IL** çıktısına bakılırsa aşağıdaki kod parçalarının üretildiği görülür.

```
.method private hidebysig static void Main(string[] args) cil managed
{
  .entrypoint
  // Code size 66 (0x42)
  .maxstack 3
  .locals init ([0] class LambdaVeCIL.IslemHandler`1<float64> hnd,[1] class
LambdaVeCIL.IslemHandler`1<int32> hnd2)
```

```
    IL_0000: nop
    IL_0001: ldsfld class LambdaVeCIL.IslemHandler`1<float64>
LambdaVeCIL.Program::'CS$<>9__CachedAnonymousMethodDelegate2'
    IL_0006: brtrue.s IL_001b
    IL_0008: ldnull
    IL_0009: ldftn float64 LambdaVeCIL.Program::'<Main>b__0'(float64,float64)
    IL_000f: newobj instance void class
LambdaVeCIL.IslemHandler`1<float64>::.ctor(object, native int)
    IL_0014: stsfld class LambdaVeCIL.IslemHandler`1<float64>
LambdaVeCIL.Program::'CS$<>9__CachedAnonymousMethodDelegate2'
    IL_0019: br.s IL_001b
    IL_001b: ldsfld class LambdaVeCIL.IslemHandler`1<float64>
LambdaVeCIL.Program::'CS$<>9__CachedAnonymousMethodDelegate2'
    IL_0020: stloc.0
    IL_0021: ldsfld class LambdaVeCIL.IslemHandler`1<int32>
LambdaVeCIL.Program::'CS$<>9__CachedAnonymousMethodDelegate3'
    IL_0026: brtrue.s IL_003b
    IL_0028: ldnull
    IL_0029: ldftn int32 LambdaVeCIL.Program::'<Main>b__1'(int32,int32)
    IL_002f: newobj instance void class
LambdaVeCIL.IslemHandler`1<int32>::.ctor(object,native int)
    IL_0034: stsfld class LambdaVeCIL.IslemHandler`1<int32>
LambdaVeCIL.Program::'CS$<>9__CachedAnonymousMethodDelegate3'
    IL_0039: br.s IL_003b
    IL_003b: ldsfld class LambdaVeCIL.IslemHandler`1<int32>
LambdaVeCIL.Program::'CS$<>9__CachedAnonymousMethodDelegate3'
    IL_0040: stloc.1
    IL_0041: ret
} // end of method Program::Main
```

Her ne kadar **IL(Intermediate Language)** tarafı karışık görünsede dikkat edilmesi gereken noktalar **IL_0001 - IL_0020** aralığındaki yapının **IL_0020 - IL_0040** arasındaki ile aynı olmasıdır. Söz edilen ilk aralıkta **lambda** ifadesinin kullanıldığı satıra ait üretimler yer almaktadır. İkinci parçada ise **isimsiz metod** kullanımına ait üretimler bulunmaktadır. Yazımızın asıl amacı **IL** tarafındaki üretimleri kavramak değildir ancak sonuç itibariyle lambda ifadeleri, isimsiz metodlar ile **aynı IL çıktılarının** üretilmesini sağlamaktadır.

Son olarak lambda ifadeleri kullanılırken dikkat edilmesi gereken bazı durumları göz önüne alalım.

1 - Lambda ifadelerinde tanımlanan değişkenler diğer metodlar tarafından **kullanılamazlar**. Başka bir deyişle, değişkenlerin kapsamı lambda ifadesinin sınırlarıdır. Aşağıdaki ekran görüntüsünde bu durum ifade edilmektedir. Görüldüğü gibi ifade içerisinde tanımlanan **d** değişkenine kapsam dışından erişilememekte ve **derleme zamanı hatası(Compile-Time Error)** alınmaktadır.

```
static void Main(string[] args)
{
   IslemHandler<double> hnd = (x, y) => {
      int d=10;
      return x + y;
   };
   d = 1;
   The name 'd' does not exist in the current context
```

2 - Elbette lambda ifadesi dışında tanımlanmış olan bir değişkene, ifade içerisinden erişilebilmektedir. Söz gelimi aşağıdaki ekran çıktısındanda görüleceği gibi, **d**değişkeni lambda ifadesi dışında 10 olarak tanımlanmış ve metod çağrısından sonra 11 olarak değiştirilmiştir.

```
9    static void Main(string[] args)
10   {
11      double d = 10;
12      IslemHandler<double> hnd = (x, y) => {
13         d++;
14         re  d | 11.0
15      };
16      hnd(4, 5);
```

3 - Lambda ifadelerinin sol tarafında yer alan parametrelerde **ref** ve **out** anahtar kelimeleri kullanılamaz.

Bu yazımızda lambda ifadelerinin genel kullanımı üzerinde durulmaya çalışmıştır. Lambda ifadelerinin getirdiği kolaylığı görmek amacıyla C# 1.0 tarafından C# 3.0 tarafına doğru ilerlenmeye çalışılmıştır. Lambda ifadeleri ile ilişkili bir diğer önemli konuda **ifade ağaçlarıdır(Expression Trees)**. Bu konuyu ilerleyen yazılarımızda incelemeye çalışacağız. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

LambdayiAnlamak.rar (119,00 kb)

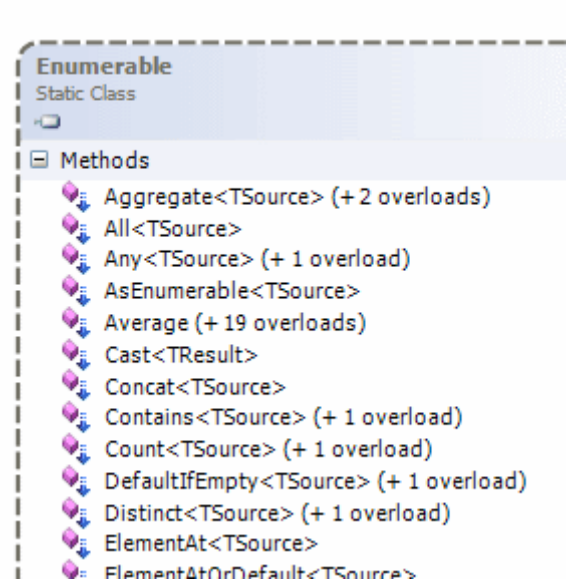
C# 3.0 - Derinlemesine Extension Method Kavramı (2008-03-14T20:23:00)

c# 3.0,

Bilindiği üzere Language INtegrated Query(LINQ) mimarisinin uygulanışında C# 3.0(Visual Basic 9.0) ile birlikte gelen yenilikler oldukça önemli bir yere sahiptir. Bu yeniliklerin çoğu var olan .Net Framework 2.0 yapısını bozmadan genişletebilmek amacıyla tasarlanmıştır. Genişletme Metodları(Extension Methods) bu yeniliklerden sadece bir tanesidir.(Object Initializers, Anonymous Types, Partial Methods, var anahtar kelimesi, auto-implemented property, => operatörü diğer C# 3.0 yenilikleri arasında sayılabilir) Söz konusu yeniliğin çıkış amacı genişletilemeyen tiplere yeni fonksiyonelliklerin eklenebilmesinin sağlanmasıdır. öyleki bu sayede koleksiyonlar(Collections), DataTable,

dizi(Array) gibi var olan CLR tipleri(Common Lanugage Runtime) üzerinde LINQ tarzı sorgu ifadelerinin yazılabilmesi olanaklı hale gelmiştir.

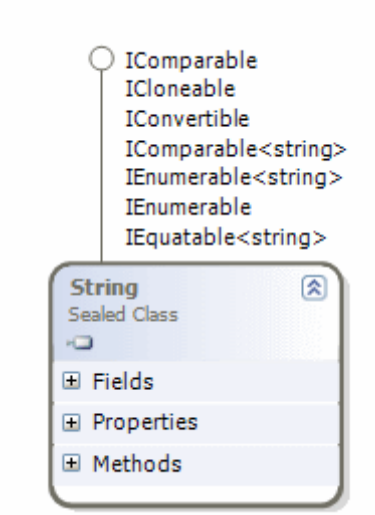
örneğin IEnumerable<T> arayüzüne(Interface) uygulanan genişletme metodları(Extension Methods) sayesinde T türünden koleksiyonlar üzerinde Sum, Count, Select, Average, OrderBy,Distinct gibi fonksiyonellikler uygulanabilmektedir. Bunun için System.Linq isim alanı(Namespace) altında Enumerable isimli static bir sınıf geliştirilmiş ve içerisine aşağıdaki sınıf diagramda(Class Diagram) bir kısmı görünen pek çok genişletme metodu ilave edilmiştir.



Bilindiği üzere SQL sorgularına benzeyen LINQ ifadeleri aslında arka planda metodlar yardımıyla işaret edilebilirler. Nitekim programatik ortamlar bu tarz bir yaklaşımı gerektirmektedir. üstelik bu işlemler yapılırken var olan tiplerin içeriklerine müdahale edilmemekte, sadece ek fonksiyonellikler katılmaktadır. Bu nedenle genişletme metodları LINQ ifadelerinin kullanılabilmesinde önemli bir role sahiptir. Bu makalemizde genişletme metodlarını derinlemesine incelemeye çalışacak ve ayrıntılara bakıyor olacağız.

Not : *Genişletme metodları var olan tiplere ek fonksiyonellikler kazandırılmasını sağlarken bunların orjinal yapısını asla bozmazlar. Tanımlandıkları programda, uygulandıkları tipin bir parçası olarak yaşar ama o tipin orjinalliğine etki etmeden ek işlevselliklerin kullanılabilmesini olanaklı kılarlar.*

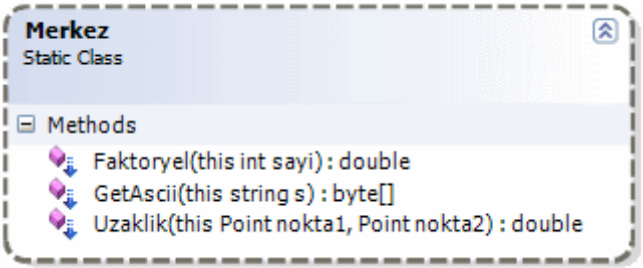
Herşeyden önce nesne tabanlı programlama dillerinde(Object Oriented Programming Language), kalıtım(Inheritance) sayesinde var olan tiplerin(Type) genişletilmesi mümkündür. Ancak türetilmesine izin verilmeyen tiplerde mevcuttur. Söz gelimi sealed anahtar kelimesi ile imzalanmış olan tipler türetme tekniği yardımıyla genişletilemez. üstelik .Net içerisinde bu şekilde tanımlanmış olan sayısız sınıf vardır. örneğin String sınıfı sealed olarak imzalanmış bir sınıf olduğundan kendisinden türetme yapılmasına izin verilmemektedir. (üstelik String parçalı bir sınıf(Partial Class)' da değildir.)



Bu nedenle bu sınıfa ek fonksiyonellikler ilave edilmesi mümkün değildir. Oysaki var olan .Net tiplerinin(yada kendi geliştirdiğimiz ama türetme yapılmasına izin verilmeyen tiplerin) yapısını bozmadan yeni fonksiyonelliklerin katılarak uygulamalar içerisinde ela alınması istendiği vakalar söz konusudur. Ki nesnelerin SQL tarzında sorgulanabilmeside buna bir örnek olarak verilebilir. Bu sebepten genişletme metodlarının önemi oldukça fazladır.

Not : *Genişletme metodları(Extension Methods), değer(value) ve referans(reference) türleri ile arayüzlere(Interface) uygulanabilir. Değer türü olarak yapılar(struct) göz önüne alınabilir. Nitekim yapılar açıkça belirtilmesede kendilerinden türetilme yapılmasına izin vermemektedir.*

Genişletme metodları ele alınırken gözden kaçırılmaması gereken bir nokta daha vardır. Genişletme metodları nesne yönelimli programlama jargonundaki kurallardan birisi değildir. Sadece .Net Framework mimarisine özgü bir kavramdır. Genişletme metodları bu anlamda bir tipin paylaşımlı fonksiyonları olarakda düşünülebilir. Hatta bu metodlar .Net Framework 3.5 öncesi sürümlerdeki tipler içinde uygulanabilirdir. Tabi öncelikli olarak genişletme metodlarının(Extension Methods) C# 3.0 içerisinde nasıl yazıldığına bakmata yarar vardır. Genişletme metodlarının yazılmasında üç basit kural vardır. Bu metodlar static bir sınıf içerisinde static olarak tanımlanmalı ve uygulanacakları tipi ilk parametrelerinde this anahtar kelimesi ile birlikte almalıdır.(Ki bunların bir takım sebepleri vardır) örnek olarak ayrı bir sınıf kütüphanesi içerisinde geliştirilmiş olan aşağıdaki sınıfı göz önüne alabiliriz.



Merkez isimli static sınıfın içeriği aşağıdaki gibidir.

```
using System;
using System.Drawing;

namespace Genisletmeler
{
   /// <summary>
   /// Genişletme fonksiyonelliklerini içeriri
   /// </summary>
   public static class Merkez
   {
      /// <summary>
      /// Bir string içerisindeki tüm karakterlerin Ascii değerlerini ele alıp byte dizisi şeklinde geriye döndürür. Eğer string null veya empty ise exception döndürür.
      /// </summary>
      /// <param name="s">Byte değerleri döndürülecek string parametre</param>
      /// <returns>Ascii değerleri</returns>
      public static byte[] GetAscii(this string s)
      {
         if (String.IsNullOrEmpty(s))
            throw new Exception("String veri olmalıdır.");
         byte[] result = new byte[s.Length];
         for (int i = 0; i < s.Length; i++)
         {
            result[i] = (byte)s[i];
         }
         return result;
      }

      /// <summary>
      /// Int32 tipinden bir sayının faktöryelinin bulunmasını sağlar
      /// </summary>
      /// <param name="sayi">Faktöryel değeri hesap edilecek değişken</param>
      /// <returns>Sayının faktöryeli</returns>
      public static double Faktoryel(this Int32 sayi)
      {
         if (sayi == 0
            || sayi == 1)
            return 1;
         else
            return  sayi*Faktoryel(sayi-1);
      }
```

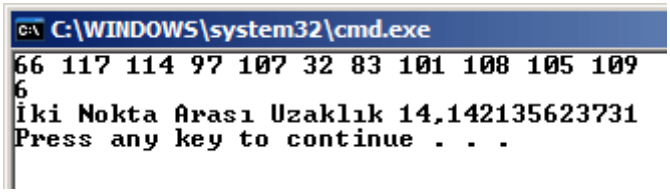
```
        /// <summary>
        /// İki Point arasındaki uzaklığın pisagor teoremine göre hesap edilmesini sağlar.
        /// </summary>
        /// <param name="nokta1">Birinci nokta</param>
        /// <param name="nokta2">İkinci nokta</param>
        /// <returns>Mesafe</returns>
        public static double Uzaklik(this Point nokta1,Point nokta2)
        {
            int xFarki = nokta1.X - nokta2.X;
            int yFarki = nokta2.X - nokta2.Y;
            return Math.Sqrt((xFarki * xFarki) + (yFarki * yFarki));
        }
    }
}
```

Merkez isimli static sınıf içerisinde 3 farklı metod yer almaktadır. Bu metodlardan GetAscii String sınıfına, Faktoryel Int32, Uzaklik ise Point yapılarına(Struct) uygulanmaktadır. GetAscii metodu yardımıyla string bir değişkenin karakterlerinin byte tipinden bir dizi olarak elde edilmesi sağlanmaktadır. Faktroyel metodu Int32 tipinden değişkenlere uygulanabilmekte olup, basit olarak sayının faktöryelini hesaplamaktadır. (üstelik yinelemeli-Recursive bir metod olarak tasarlanmıştır. Buna göre genişletme metodlarının recursive formasyonda kullanılabileceği söylenebilir.) Uzaklik isimli metod ise diğerlerinden farklı olarak birde ek parametre almaktadır. Buna göre Point tipinden bir değişkenin başka bir Point ile arasındaki uzaklığın bulunabilmesi sağlanmaktadır. (Yani kabaca iki nokta arasındaki mesafenin pisagor teoremi çerçevesinde hesaplanması gerçekleştirilmeltedir.) Bir başka deyişle genişletme metodları(Extension Methods) uygulanacakları tipi belirten ilk parametreden sonra ek parametrelerde alabilmektedir. Söz konusu sınıfın özellikle CIL(Common Intermediate Language) tarafına nasıl aktarıldığını incelemeden önce kullanımına bakılabilir. Bu amaçla Merkez isimli static sınıfı içeren kütüphaneyi(Class Library) referans eden basit bir Console uygulaması geliştirilip aşağıdaki kodlar test amacıyla kullanılabilir.

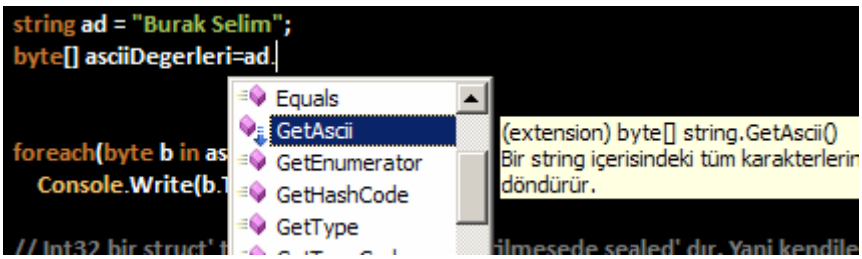
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using Genisletmeler;
using System.Drawing;

namespace DerinlemesinExtensionMethods
{
    class Program
    {
        static void Main(string[] args)
        {
```

```
        // String sınıfı sealed olarak imzalanmıştır bu nedenle kendisinden türetme yapılıp ek fonksiyonellikler katılamaz.
        string ad = "Burak Selim";
        byte[] asciiDegerleri=ad.GetAscii();
        foreach(byte b in asciiDegerleri)
           Console.Write(b.ToString()+" ");

        // Int32 bir struct' tır. Struct' lar açıkça belirtilmesede sealed' dır. Yani kendilerinden türetme yapılamaz. Ancak genişletme metodları yardımıyla bunlar ek fonksiyonellikler     katılabilir.
        int sayi = 3;
        Console.WriteLine(sayi.Faktoryel());

        // Point .Net Framework içerisinde System.Drawing isim alanında tanımlanmış olan struct tipidir. Kendisinden türetme yapılamaz. Ancak extension method sayesinde Uzaklik isimli bir metoda sahip olabilir
        Point pn = new Point(10, 20);
        Console.WriteLine("İki Nokta Arası Uzaklık {0}",pn.Uzaklik(new Point(20, 30)).ToString());
      }
   }
}
```

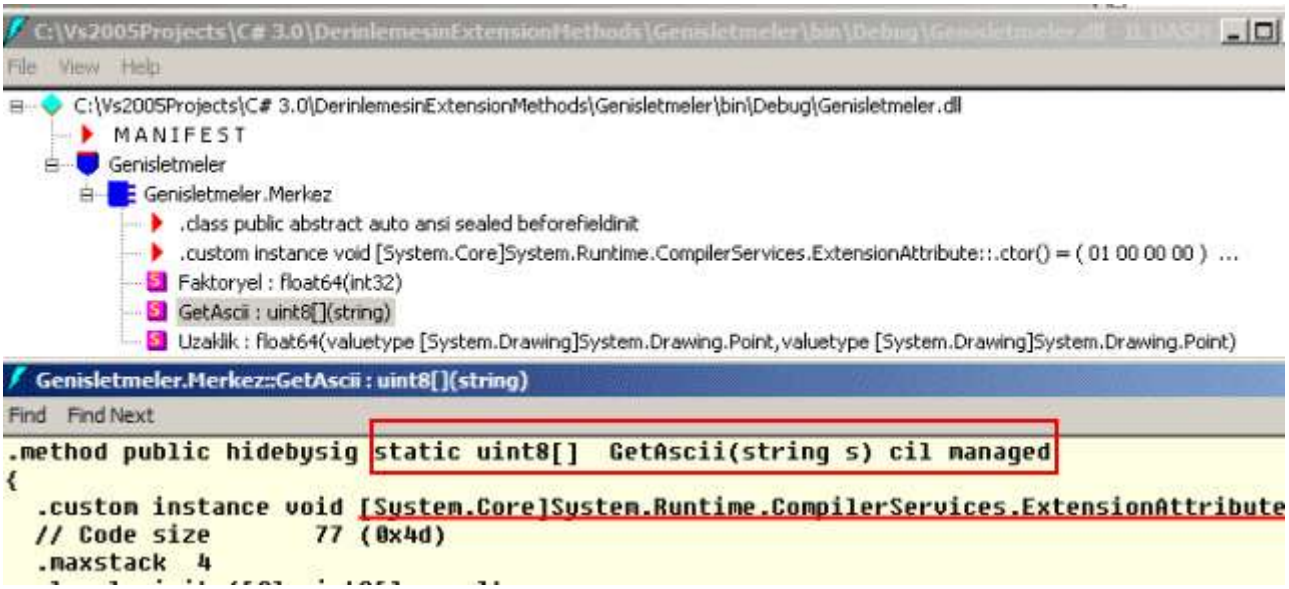
Uygulama test edildiğinde aşağıdakine benzer sonuçlar ile karşılaşılır.



Görüldüğü gibi string, int ve Point tipinden değişkenler üzerinden yeni fonksiyonellikler kullanılabilmektedir. Peki bu sistem alt tarafta nasıl yürümektedir? Sonuç itibariyle var olan bir CLR tipinin(Common Language Type) bozulmadan genişletilebilmesinin ancak çalışma zamanı motoru(Runtime Engine) tarafından önemli olduğu ortadadır. Hatta dikkat edileceği üzere Visual Studio geliştirme ortamı, eklenen genişletme metodlarını intellisense özelliğinde aynen aşağıdaki ekran görüntüsünde olduğu gibi gösterebilmektedir.



çalışma zamanında bazı bilgilere bakılması söz konusu ise eğer, niteliklerden(Attribute) yararlanılmaktadır. Bilindiği üzere nitelikler yardımıyla çalışma zamanına ekstra metadata bilgileri aktarılabilmektedir. Bu sebepten genişletme metodlarında da başrol oyuncusu olarak System.Core.dll assembly' ında System.Runtime.CompilerServices isim alanı(Namespace) altında bulunan Extension isimli bir nitelik(Attribute) görev almaktadır. Bu her ne kadar kod tarafında sadece Visual Basic 9.0 ile görülsede, IL tarafında rahat bir şekilde tespit edilebilmektedir.



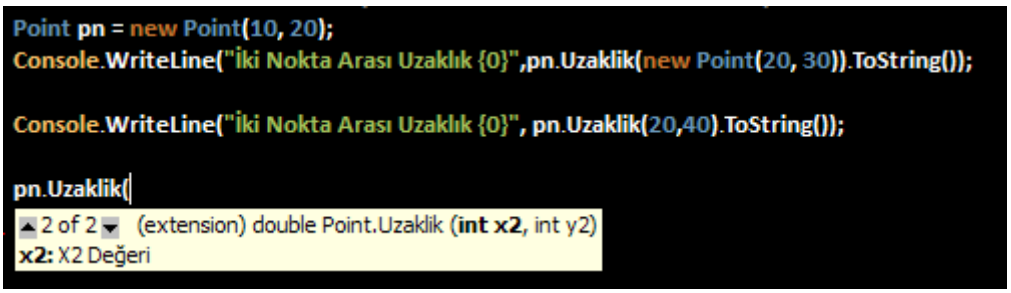
Yukarıdaki IL görüntüsündende dikkat edileceği üzere GetAscii isimli metod ExtensionAttribute niteliği ile imzalanmıştır. Bu son derece anlamlıdır nitekim hem compiler hemde çalışma zamanı(Runtime) için, takip eden metodun, ilk parametre ile belirtilen tip için bir genişletme olduğu belirtilmektedir. Bir başka deyişle söz konusu nitelik derleyiciye veya çalışma zamanına, ilk parametredeki tip için bazı ek bilgiler gönderir ve yeni fonksiyonelliği kazanmasını sağlar. Buraya kadar genişletme metodlarının ne olduğundan ve nasıl uygulandığından bahsetmeye çalıştık. Genişletme metodları ile ilişkili dikkat edilmesi gereken bazı noktalar da vardır. Dilerseniz yazımızın ilerleyen kısımlarında bu konulara değinelim.

1 - Genişletme metodları aşırı yüklenebilirler(Overloading)

Metodlar aynı isim altında birden fazla kez yazılabilirler(Buna şu an için verilebilecek en güzel örneklerden birisi WriteLine metodudur. Dikkat edileceği üzere bu metodun 19 farklı versiyonu bulunmaktadır.) Bu kısaca metodun aşırı yüklenmesi(Method Overloading) olarak adlandırılmaktadır. Metodun aşırı yüklenmesi sırasındaki önemli kriter, parametre tipleri ve sayılarının belirlediği imzalardır(Method Signature). çok doğal olarak genişletme metodlarıda aşırı yüklenebilirler. Söz gelimi Merkez sınıfı içerisinde iki nokta arasındaki uzaklığı bulmak için tasarlanmış olan Uzaklik metodunun farklı bir versiyonu aşağıdaki gibi yazılabilir.

```
/// <summary>
/// İki Point arasındaki uzaklığın pisagor teoremine göre hesap edilmesini sağlar
/// </summary>
/// <param name="nokta1">Metodun uygulanacağı Point tipinden değişken</param>
/// <param name="x2">X2 Değeri</param>
/// <param name="y2">Y2 Değeri</param>
/// <returns>Mesafe</returns>
public static double Uzaklik(this Point nokta1, int x2, int y2)
{
   int xFarki = nokta1.X - x2;
   int yFarki = nokta1.Y - y2;
   return Math.Sqrt((xFarki * xFarki) + (yFarki * yFarki));
}
```

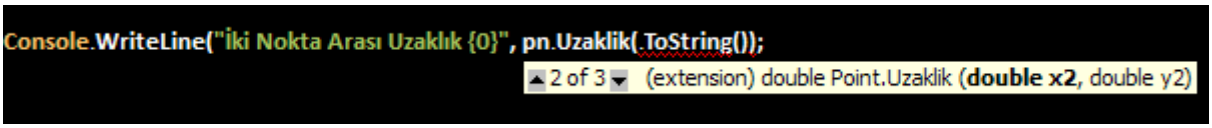
Bu versiyon diğerinden farklı olarak Point tipinden ikinci bir parametre almak yerine int tipinden iki ayrı parametre kullanmaktadır. Burada metodların hem parametre sayıları hemde tipleri farklılaşmayı sağlamaktadır. Merkez sınıfı bu haliyle örnek uygulamada kullanıldığında aşağıdaki ekran görüntüsünden de izlenebileceği gibi iki farklı Uzaklik metodunun çağırılabileceği görülür.



Burada akla hemen şu soru gelebilir. Merkez sınıfının haricinde başka bir static sınıf içerisinde, Point tipi için Uzaklik metodunun aşırı yüklenmiş başka bir versiyonu yazılabilir mi? Eğer yazılırsa söz konusu uygulamada bu versiyon kullanılabilir mi? Bu sorulara cevap verebilmek için Console uygulaması içerisinde aşağıdaki gibi bir static sınıf tanımlaması yapıldığını varsayalım.

```
static class Genisletme
{
   public static double Uzaklik(this Point nokta1, double x2, double y2)
   {
      double xFarki = nokta1.X - x2;
      double yFarki = nokta1.Y - y2;
      return Math.Sqrt((xFarki * xFarki) + (yFarki * yFarki));
   }
}
```

Merkez sınıfı Genisletmeler isim alanı altında ve üstelik farklı assembly içerisinde yer almaktadır. Genisletme isimli sınıf ise DerinlemesineExtensionMethods isimli Console uygulaması içerisinde tanımlanmıştır. Bu durumda Main metodu içerisinde Point tipinden bir değişken kullanılmak istendiğinde Uzaklik isimli fonksiyonun 3 farklı versiyonuna ulaşılabildiği görülecektir.



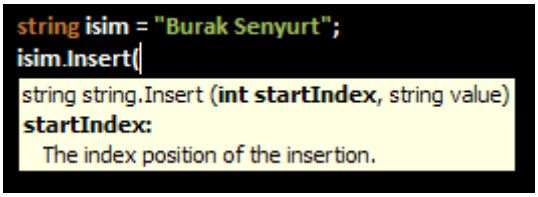
Bir başka deyişle farklı static sınıflar içerisindede olsalar genişletme metodları aşırı yüklenebilirler.

2 - CLR Tipi(Common Language Runtime Type) içerisinde tanımlı olan bir fonksiyonun aynısı extension method olarak yazılıp, örnek(Instance) tipe ait metod ezilebilir(Override) mi?

Bu bir anlamda orjinal CLR tiplerinin güvenliği ile ilişkilide bir konudur. Nitekim türetilmesine izin verilmeyen tiplerin asıl tasarım amaçlarından biriside içeriklerinin değiştirilmesinin engellenmesidir. Bu anlamda sealed olarak işaretlenmiş tiplerin aslında genişletme metodları yardımıyla ek fonksiyonelliklere sahip olabilmesi ve hatta aşırı yükleme yapılabilmesi orjinal tipte tanımlı metodların ezilip ezilemeyeceği vakasını ortaya çıkarmaktadır. Bu durumu analiz etmek için basit olarak String tipinde tanımlı olan bir metodun aynısını extension method olacak şekilde tanımlamaya çalışabiliriz.

```
public static string Insert(this string s,int siraNo, string metin)
{
   Console.WriteLine("Extension Method");
   return metin;
}
```

Burada String sınıfının Insert metodunun aynısı extension metod olarak yazılmaya çalışılmaktadır. Uygulama derlendiğinde herhangibir hata mesajı alınmaz. Ancak string bir değişken üzerinden Insert metodu çağırıldığında aşağıdaki ekran görüntüsünde olduğu gibi orjinal versiyonun kullanılabileceği görülür.



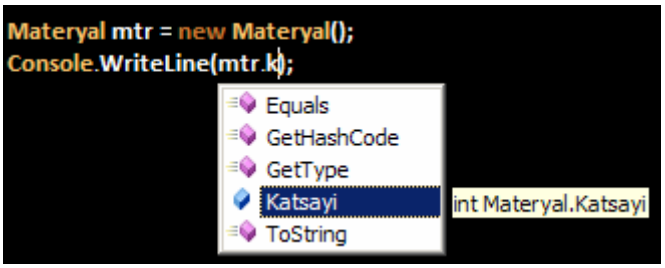
Buna göre derleyici açısından nesne örneği(Object Instance) metodunun daha öncelikli olduğu ortadadır. Bir başka deyişle genişletme metodları yardımıyla orjinal nesne örneğine ait metodlar ezilemezler.

3 - Bir tip içerisinde tanımlı özellik yada alan ile aynı isimde bir extension metod tanımlanırsa.

Bu durumu analiz edebilmek için aşağıdaki kod parçası göz önüne alınabilir.

```
sealed class Materyal
{
   public int Katsayi;
}
static class Genisletme
{
   public static void Katsayi(this Materyal mtr)
   {
      Console.WriteLine("Genişletme metodu");
   }
}
```

Burada tanımlanan Materyal isim sınıf sealed olarak imzalanmıştır ve içerisinde int tipinden Katsayi isimli bir alan(Field) içermektedir. Bu tip bir sınıfın başka bir nesne kullanıcısı(Object User) tarafından genişletilmek istendiği bir durumda, bilinçsiz olarak aynı isme sahip genişletme metodları eklenebilir. Bunu sembolize eden Genisletme sınıfı kendi içerisinde, Materyal sınıfındaki alanla aynı adda olan Katsayi isimli bir metod içermektedir. Ne varki kod tarafında Materyal sınıfına ait bir örnek oluşturulduğunda, Katsayi genişletme metoduna erişilemediği açık bir şekilde görülmektedir.



Dikkat edilecek olursa sadece Katsayi isimli nesne alanı(Field) görünmektedir. Fakat burada oldukça enteresan bir durumda söz konusudur. Eğer kodda ısrar edilir ve Katsayi genişletme metodu kullanılmak istenirse derleme zamanı hatası alınmadığı görülür. Hatta kod yürütüldüğünde, genişletme metodunun çalıştığı görülecektir. Bu durum aslında genişletme metodlarının isimlendirilmesinin önemli olduğunu göstermektedir.

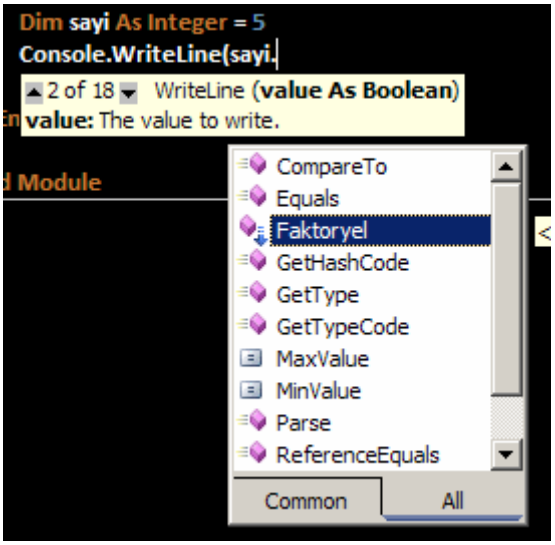
4 - Extension metodlar dilden bağımsızdır.

Genişletme metodları daha öncedende bahsedildiği gibi CIL(Common Intermediate Language) tarafında Extension niteliği ile imzalanırlar. Bu sebepten dolayıda .Net destekli diller tarafından kullanılabilirler. Söz gelimi C# kodlaması ile geliştirilmiş genişletme metodları, Visual Basic ile yazılmakta olan bir proje içerisinde kullanılabilir. Elbette tam tersi durumda geçerlidir. Konuyu daha kolay analiz etmek için Merkez isimli sınıfı içeren

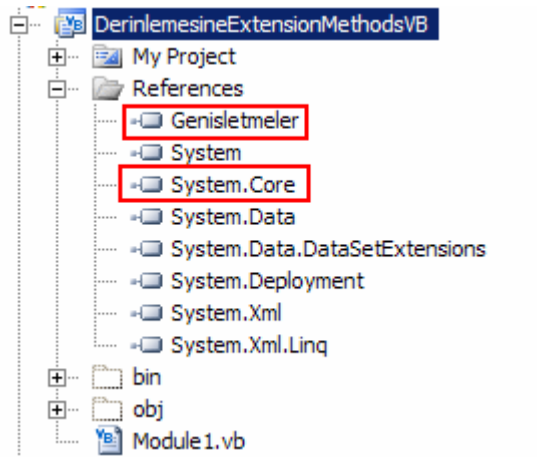
C# tabanlı kütüphaneyi basit bir Visual Basic Console uygulamasında aşağıdaki gibi deneyebiliriz.

```
Imports Genisletmeler
Imports System.Drawing

Module Module1

    Sub Main()

        Dim str As String = "Burak Selim Şenyurt"
        Dim dizi As Byte() = str.GetAscii()
        For i As Int32 = 0 To dizi.Length - 1
            Console.Write(dizi(i).ToString() + " ")
        Next

        Console.WriteLine()

        Dim sayi As Integer = 4
        Console.WriteLine(sayi.Faktoryel().ToString())

        Dim nokta1 As New Point(3, 4)
        Console.WriteLine(nokta1.Uzaklik(6, 8).ToString())

    End Sub

End Module
```

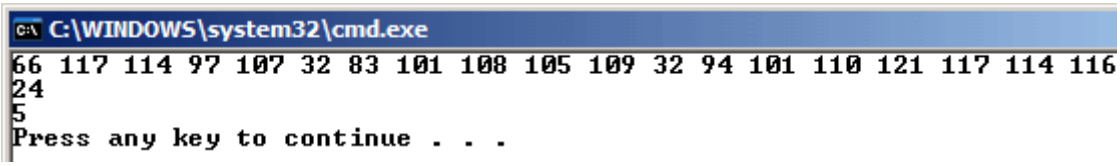
Visual Basic tarafında kod yazıyor olsakta, Visual Studio arabiriminin intellisense özelliği C# ile yazılmış genişletme metodlarını gösterecektir. Sonuç itibariyle burada yapılan farklı bir assembly içerisinde tip ve üyelerine erişmektir.



Elbetteki kodun çalışabilmesi için C# kütüpanesinin Visual Basic tabanlı projeye referans edilmesi gerekmektedir.



Bu işlemin ardından uygulama çalıştırılırsa genişletme metodlarının başarılı bir şekilde çalıştığı görülür.

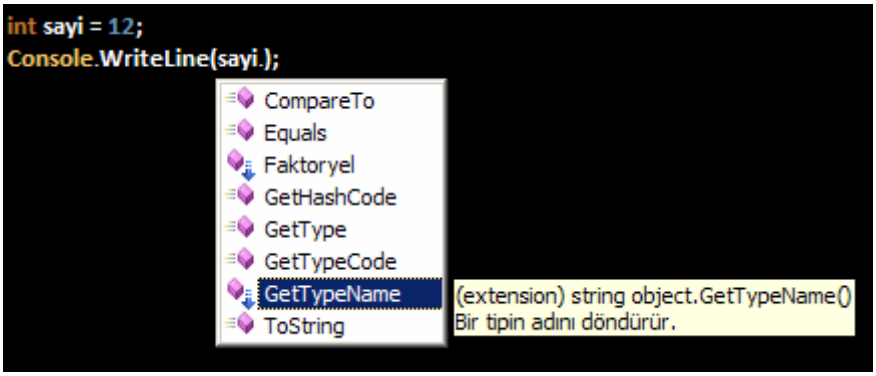


5 - Object tipinin genişletilmesi.

Object tipide genişletme metodlarına sahip olabilir. .Net Framework içerisinde yer alan tipler object türevli olduklarından çok doğal olarak tanımlanan genişletme metodlarını kullanabilirler. Bu durumu test edebilmek için sembolik olarak aşağıdaki genişletme metodunu eklediğimizi düşünelim.

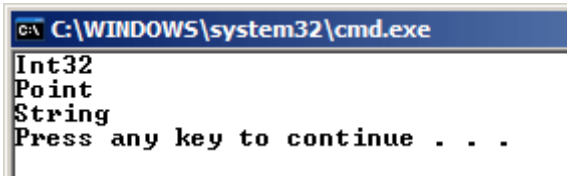
```
public static string GetTypeName(this object obj)
{
   return obj.GetType().Name;
}
```

Metod basitçe herhangibir nesnenin tip adını döndürmektedir. Metodun uygulanışına bakıldığında ise herhangibir tipteki değişkenden sonra çağırılabildiği görülecektir.



Aşağıda, örnek bir kod parçası kullanımı ve çalışma zamanı çıktısı yer almaktadır.

```
int puan = 51;
Console.WriteLine(puan.GetTypeName());

Point nokta3 = new Point(3, 4);
Console.WriteLine(nokta3.GetTypeName());

string firmaAdi = "FreeLancer";
Console.WriteLine(firmaAdi.GetTypeName());
```



Visual Basic 9.0' da özellikle Object tipinden bir değişkene atama yapıldığında, genişletme metodlarını çağırmak çalışma zamanı istisnasına(Run Time Exception) neden olmaktadır.

```
Dim obj As Object = 3.14F
Console.WriteLine(obj.GetTypeName())
```

Bu kullanım çalışma zamanında MissingMemberException istisnasının(Exception) fırlatılmasına neden olmaktadır. Sorun Object tipinin Late-Bound olmasından kaynaklanmaktadır. Bunun çözmek için type inference kavramından(C# karşılığı var anahtar kelimesi) yararlanılabilir. (Dim obj=3.14F)

Ne varki bu durum C# tarafında geçerli değildir. Bu nedenle C# tarafında aşağıdaki kod parçası sorunsuz olarak çalışmaktadır.

```
Object obj = 3.14f;
Console.WriteLine(obj.GetTypeName());
```

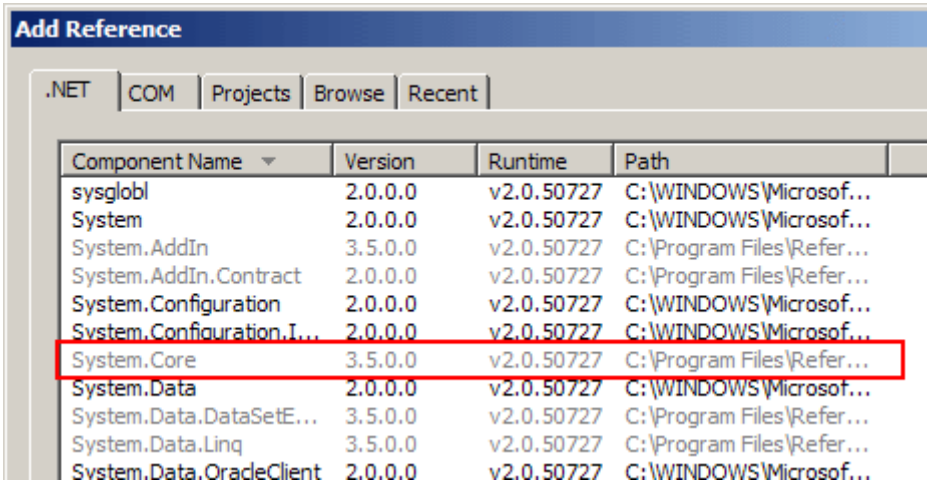
Object tipi için genişletme metodları var anahtar kelimesi ile birliktede kullanılabilirler. Aşağıdaki kod parçası bu durumu göstermektedir. Bu kod içerisinde var anahtar kelimesi ile tanımlanan nesne isimli değişken eşitliğin sol tarafı göz önüne alındığında float(Single

yapısı-struct) tipindendir. Bu sebepten nesne üzerinden çağırılan GetTypeName isimli genişletme metodu geriye Single değerini döndürecektir.

```
var nesne = 3.14f;
Console.WriteLine(nesne.GetTypeName());
```

6 - .Net Framework 2.0 hedefli bir uygulama içerisinde extension metodlar kullanılabilir mi?

Extension niteliği System.Core.dll assembly' içerisinde tanımlanmıştır ve System.Runtime.CompilerServices isim alanında bulunmaktadır. System.Core.dll' i .Net Framework 3.5 ile gelmekte olsada .Net Framework 2.0 motorunu kullanarak çalışmaktadır. Bu noktada .Net 2.0 ile geliştirilmiş bir uygulamada extension metod kullanımı söz konusu olabilir mi? Akla ilk gelen yöntem System.Core.dll assembly' ının ilgili projeye referans edilmesidir. Ancak Visual Studio 2008 içerisinde bu denendiğinde .Net 2.0 tabanlı projeye söz konusu referansların eklenemediği görülecektir.



Browse seçeneği ile ekleme yapılmaya çalışılsada durum değişmeyecektir. Ancak izlenecek basit bir yol ile .Net 2.0 tabanlı projede extension metod kullanımı sağlanabilir. Bunun için uyulamada System.Runtime.CompilerServices isimli bir namespace tanımlanır ve içerisine Extension isimli bir attribute sınıfı eklenir. Bu işlemin ardından extension metod yazılabildiği hatta kullanılabildiği görülecektir. Durumu daha iyi analiz etmek amacıyla .Net 2.0 tabanlı bir Console uygulamasına ait aşağıdaki kod parçası göz önüne alınabilir.

```
using System;
using System.Runtime.CompilerServices;

// 1nci : İlk olarak System.Runtime.CompilerServices adlı isim alanı içerisinde
ExtensionAttribute isimli bir nitelik tanımlanır
namespace System.Runtime.CompilerServices
{
    // 2nci: Nitelik assembly, sınıf ve metod seviyesinde uygulanabilir. Bir kere
```

kullanılabilir.

```
   [AttributeUsage(AttributeTargets.Assembly| AttributeTargets.Class|
AttributeTargets.Method,AllowMultiple=false,Inherited=false)]
   public class ExtensionAttribute :
      Attribute
   {
   }
}
namespace DerinlemesineExtensionMethods2
{
   static class ExtensionMethods
   {
      // Eğer ExtensionAttribute tanımlanmazsa this keyword kullanımı için derleme zamanı
hatası alınacaktır.
      public static string GetTypeName(this object obj)
      {
         return obj.GetType().Name;
      }
      public static double Faktoryel(this Int32 sayi)
      {
         if (sayi == 0
            || sayi == 1)
            return 1;
         else
            return sayi * Faktoryel(sayi - 1);
      }
   }
   class Program
   {
      static void Main(string[] args)
      {
         int puan = 12;
         Console.WriteLine(puan.GetTypeName()); // Extension metod kullanımı

         int sayi = 4;
         Console.WriteLine(sayi.Faktoryel().ToString());
      }
   }
}
```

Uygulama çalıştırıldığında genişletme metodlarının işe yaradığı görülebilir.

Tabi bu vaka Visual Studio 2008 üzerinde .Net 2.0 tabanlı bir proje şablonu için gerçeklenmektedir. Nitekim derleme aşamasında sadece C# 3.0 derleyicisi genişletme

metodunu değerlendirebilmektedir. Bir başka deyişle Visual Studio 2005 ortamında aynı örnek çalıştırılamayacaktır.

7 - Arayüzlere genişletme metodları eklenebilir.

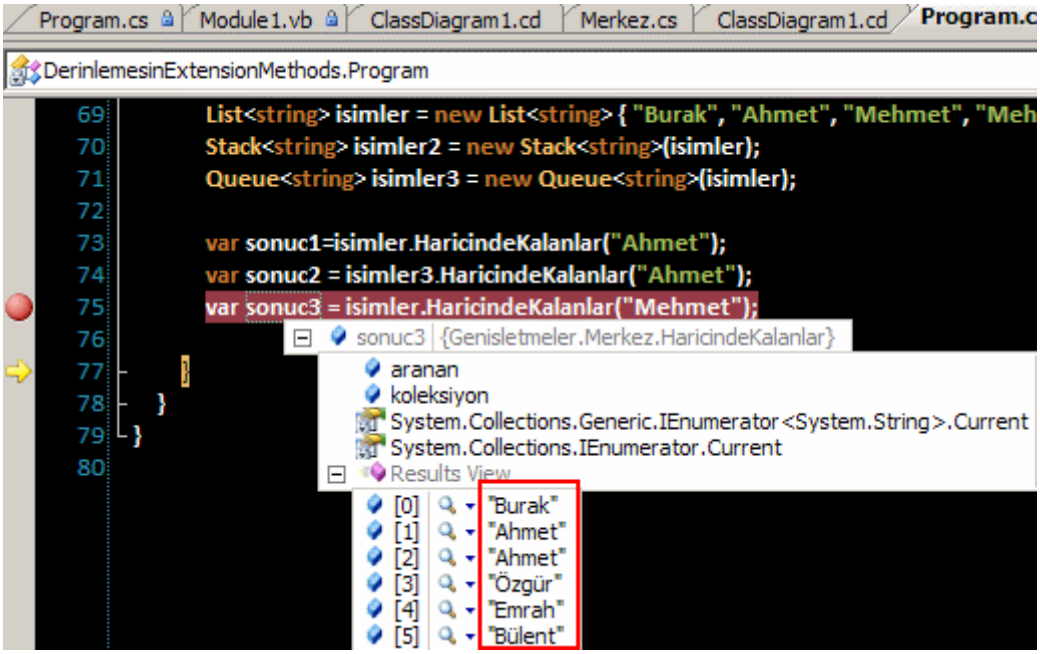
LINQ(LanguageINtegratedQuery) mimarisinin temelinde yatan genişletme metodlarının çoğu arayüzlere(Interface) uygulanmaktadır. Böylece, genişletme metodlarının uygulandığı arayüz tiplerinden türeyen türlerin tamamı, söz konusu ek fonksiyonellikleri kullanabilir duruma gelmektedir. Bu gerçektende önemli bir yetenektir. çok doğal olarak geliştirici tarafından yazılmış olan yada Framework içerisinde yer alan arayüz tiplerine genişletme metodları eklenebilir. Aşağıdaki örnek kod parçasında bu duruma örnek olacak bir metod içeriği yer almaktadır.

```
public static IEnumerable<string> HaricindeKalanlar(this IEnumerable<string>
koleksiyon,string aranan)
{
   foreach (string s in koleksiyon)
   {
      if (s != aranan)
         yield return s;
   }
}
```

HaricindeKalanlar isimli genişletme metodu, IEnumerable<string> tipinden türeyen generic koleksiyonlara uygulanabilmektedir. Görevi parametre olarak verilen string değer dışında kalan elemanları tespit ederek yeni bir IEnumerable<string> tipi içerisinde geriye döndürmektedir.(İşlerin kolaylaştırılmasında .Net 2.0 ile birlikte gelen yield anahtar kelimesinin önemli bir rolü vardır.) Buna göre IEnumerable<string> arayüzünden türeyen her tip, HaricindeKalanlar isimli genişletme metodunu kullanabilmektedir. Söz gelimi aşağıdaki kod parçasında List<string>, Stack<string>, Queue<string> tiplerine uygulanmaktadır.

```
List<string> isimler = new List<string> { "Burak", "Ahmet", "Mehmet",  "Mehmet",
"Ahmet", "özgür", "Emrah", "Bülent" };
Stack<string> isimler2 = new Stack<string>(isimler);
Queue<string> isimler3 = new Queue<string>(isimler);

var sonuc1=isimler.HaricindeKalanlar("Ahmet");
var sonuc2 = isimler3.HaricindeKalanlar("Ahmet");
var sonuc3 = isimler.HaricindeKalanlar("Mehmet");
```

çalışma zamanında örneğin sonuc3 değişkeninin içeriği aşağıdaki ekran görüntüsündeki gibi olacaktır. Dikkat edileceği üzere Mehmet ismi dışında kalanlar elde edilmektedir.



Buraya kadar bahsedilenler kısaca değerlendirilirse, genişletme metodlarının aşağıdaki avantajları sağladığından bahsedilebilir.

- Var olan tiplere(Types) yeni fonksiyonellikerin eklenebilmesi sağlanır. öyleki yazılmış olan uygulamaların çalışma sistemini bozmadan yeni fonksiyonellikler katarak genişlemelerine yardımcı olur.
- Tiplere yeni fonksiyonellikler eklenirken orjinal içeriklerine müdahale edilmesine gerek kalmaz.
- özellikle kaynak koda(Source Code) erişilemediği durumlarda ek işlevselliklerin katılabilmesinde önemli rol oynar.
- Tipleri türeterek genişletmek mümkündür, ancak türetilmelerine izin verilmeyen(Sealed Types) tipler söz konusu olduğunda çözüm genişletme metodlarıdır.

Yinede genişletme metodlarının nesne yönelimli programlama modeli nosyonunun bir parçası olmadığını düşünmekte yarar vardır. öyleki nesne yönelimli programlama nosyonu göz önüne alındığında, tip genişletmesi aslında türetme ile gerçeklenmektedir. Böylece geldik bir makalemizin daha sonuna. Bu makalemizde kısaca genişletme metodlarını(Extension Methods) derinlemesine incelemeye çalıştık. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

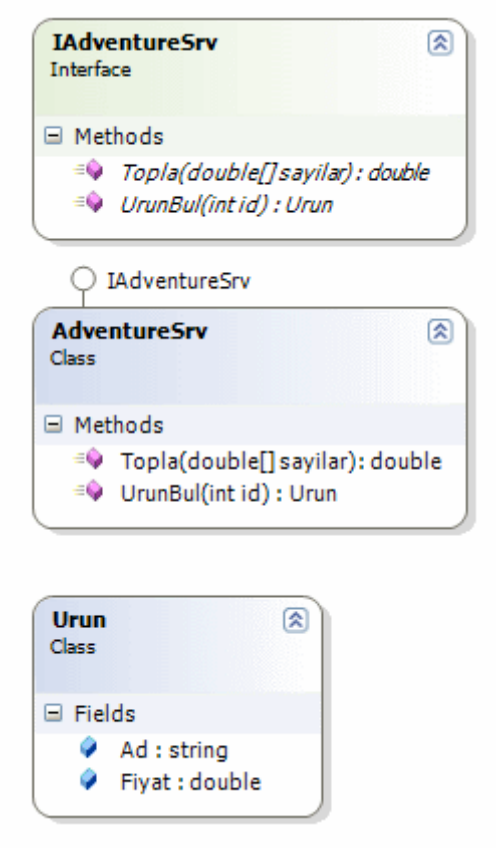
DerinlemesinExtensionMethods.rar (115,64 kb)

WCF - Visual Studio 2008 ile Gelen Yenilikler (2008-03-14T04:01:00)

wcf,

Yazılım dünyası çeşitli ürün gruplarını ve bunların üretimini içeren materyaller içermektedir. **Son kullanıcıya(End User)** veya **geliştiricilere(Developers)** yönelik olarak tasarlanan ürünlerin yazılmasında çeşitli program geliştirme arabirimleri kullanılmaktadır. Belkide bunlardan en popüler olanları Microsoft tarafından üretilen **Visual Studio** ailesidir. **Visual Studio.Net** ile başlayan serüvende kısa bir süre öncede **Visual Studio 2008** sürümü son haliyle yayınlanmıştır. Yeni sürüm özellikle **.Net Framework 2.0, 3.0** ve **3.5** için ortak ve tek bir geliştirme ortamı sunmasıyla hemen dikkati çekmektedir. Bu ve benzer özelliklerin yanında **Windows Communication Foundation** çözümleri içinde ek bir takım yenilikleri gelmektedir.

Göze çarpan yeniliklerden ilki **WcfSvcHost.exe** ve **WcfTestClient.exe** isimli yardımcı uygulamalardır. **Visual Studio 2008** kurulumundan sonra **C:\Program Files\Microsoft Visual Studio 9.0\Common7\IDE** klasörü altına eklenen bu programlar sayesinde herhangibir **WCF servis kütüphanesi(WCF Service Library)** **Host** ve **istemci** uygulamalara ihtiyaç duyulmadan test edilebilir. Genellikle bir servis kütüphanesi geliştirilirken ve test edilirken ekstra çaba sarfederek basit bir Host uygulama ve istemci(Client) yazılması gerekmektedir. Ancak **Visual Studio 2008** ile gelen yardımcı araçlar sayesinde buna gerek kalmadan basit testler yapılabilmektedir. üstelik **VS 2008** ile geliştirilen **WCF** servis kütüphaneleri, IDE içerisinden **Start(F5 veya Ctrl+F5-Start Without Debugging)**edildiklerinde otomatik olarak **WcfSvcHost.exe** ve **WcfTestClient.exe** araçları devreye girmektedir. Bir başka deyişle yazılan WCF servis kütüphaneleri anında çalıştırılıp test edilebilir. Söz konusu araçları komut satırındanda çalıştırmak ve kullanmak mümkündür. Yazıda ilk olarak bu araçlar tanınmaya çalışılacaktır. Elbette test amacıyla bir WCF servis kütüphanesine ihtiyaç vardır. Bu amaçla **VS 2008** ortamında içerikleri aşağıdaki gibi olan tipler geliştirilerek işe başlanabilir.



Servis sözleşmesinin içeriği;

```
[ServiceContract]
interface IAdventureSrv
{
   [OperationContract]
   Urun UrunBul(int id);

   [OperationContract]
   double Topla(double[] sayilar);
}
```

Sözleşmeyi uygulayan sınıfın içeriği;

```
class AdventureSrv
      :IAdventureSrv
{
   #region INorthwindSrv Members

   public Urun UrunBul(int id)
   {
      Urun urn = null;
      using (SqlConnection conn = new SqlConnection("data
```

```
source=.;database=AdventureWorks;integrated security=SSPI"))
      {
         SqlCommand cmd = new SqlCommand("Select Name,ListPrice From
Production.Product Where ProductId=@PrdId", conn);
         cmd.Parameters.AddWithValue("@PrdId", id);
         conn.Open();
         SqlDataReader reader = cmd.ExecuteReader();
         if (reader.Read())
            // C# 3.0 Object Initializers kullanılarak Urun nesnesi örneklenmektedir.
            urn = new Urun()
            {
               Ad=reader["Name"].ToString()
               ,Fiyat=Convert.ToDouble(reader["ListPrice"])
            };
         reader.Close();
      }
      return urn;
   }

   public double Topla(double[] sayilar)
   {
      return sayilar.Sum<double>(s => s); //C# 3.0 Extension Methods kavramı
kullanılmıştır.
   }

   #endregion
}
```

Urun sınıfının içeriği;

```
[DataContract]
class Urun
{
   [DataMember]
   public string Ad;

   [DataMember]
   public double Fiyat;
}
```

Servis sözleşmesi(Service Contract) basit olarak iki adet metod içermektedir. UrunBul isimli metod aynı zamanda Urun isimli sınıfa ait nesne örneği döndürmektedir. Diğer taraftan Urun sınıfı **veri sözleşmesi(Data Contract)** şeklinde tanımlanmıştır. Diğer taraftan metodun parmametrik yapısının test araçlarındaki kullanımını daha kolay irdelemek için Topla isimli fonksiyon, double tipinden bir dizi ile çalışmaktadır. Kod

içerisinde Urun sınıfına ait nesne örneklenirken **C# 3.0 Object Initializers** tekniği kullanılmaktadır. Topla metoduna gelen **double**tipinden dizinin içerisindeki sayıların toplamını bulmak içinse **Sum<T>** metodu(**C# 3.0 Extension Methods**) ele alınmaktadır. Geliştirilen servis kütüphanesinin özellikle **WcfSvcHost** için önemli olan kısmı konfigurasyon bilgileridir. Nitekim WcfSvcHost.exe uygulaması kod bazlı Host ayarlama işlemlerini ele alamaz. Bir başka deyişle test için kütüphanenin mutlaka **config** dosyasının yazılmış olması gerekir. İstemci tarafının, servis üzerinden **Host** edilecek tiplere ait servis metadata bilgilerini çekebilmek için **MEX(Metadata Exchange) EndPoint** tanımlanması yapılmamasında da yarar vardır. Bunlara göre örnek olarak bir **config** içeriği aşağıdaki gibi tasarlanabilir.

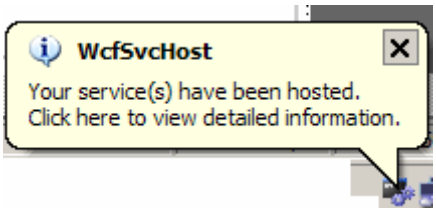
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service behaviorConfiguration="MexBehavior" name="AdventureSrv">
        <endpoint address="net.tcp://localhost:45001/AdventureSrv"
binding="netTcpBinding" name="AdvTcpEndPoint" contract="IAdventureSrv" />
        <endpoint address="http://localhost:45002/AdventureSrv"
binding="wsHttpBinding" name="AdvWsHttpEndPoint"
contract="IAdventureSrv" />
        <host>
          <baseAddresses>
            <add baseAddress="http://localhost:45000/" />
          </baseAddresses>
        </host>
      </service>
    </services>
    <behaviors>
      <serviceBehaviors>
        <behavior name="MexBehavior" >
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="true"/>
        </behavior>
      </serviceBehaviors>
    </behaviors>
  </system.serviceModel>
</configuration>
```

Konfigurasyon dosyasındanda görüldüğü gibi servis kütüphanesi **Tcp(NetTcpBinding)** ve **WsHttp(WsHttpBinding)** bazlı iki farklı **EndPoint** sunmaktadır. Bununla birlikte**http://localhost:45000** adresi üzerinden **Metadata** yayınlamasıda yapılmaktadır. Metadata bilgisinin çekilebiliyor olması, istemciler için gerekli olan **Proxy** sınıfının üretilmesinde önemli bir yere sahiptir.

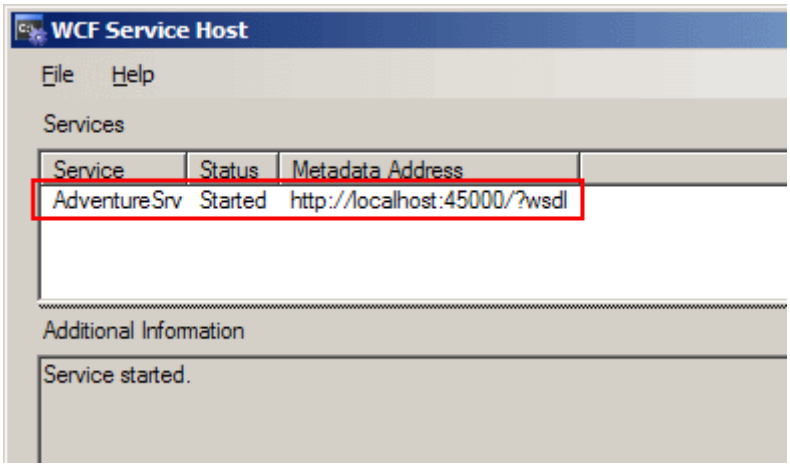
WcfSvcHost aracının kullanım amacı bir servis kütüphanesinde tanımlanan hizmetlerin **Host** edilmesini sağlayacak otomatik bir **Windows** uygulamasını başlatmaktır. Bu uygulama çalıştırıldığında, parametre olarak verilen servis kütüphanesi ve konfigurasyon dosyasını kullanarak **Host** işlemini gerçekleştirir. Aynı zamanda konfigurasyon dosyasında tanımlanan**EndPoint** noktalarında tanımlanan **servis sözleşmelerine(Service Contract)** ait metadata bilgilerinin yayınlanmasınıda sağlayabilir. Basit olarak yukarıdaki örnek kütüphaneyi Host etmek üzere WcfSvcHost aracı komut satırından aşağıdaki gibi kullanılabilir.

WcfSvcHost /service:GenelIslemler.dll /config:GenelIslemler.dll.config

Service ile tanımlanan parametreden sonra servis kütüphanesi ismi verilmektedir. **Config** parametresinde ise konfigurasyon dosyası işaret edilir. Bu işlemin ardından WcfSvcHost uygulamasının **Tray Icon**' a atıldığı görülür.

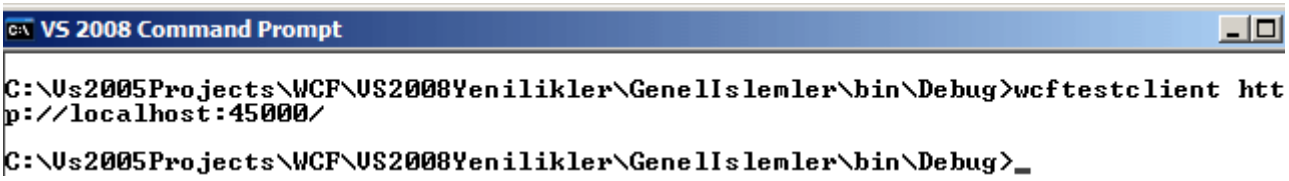


Bir başka deyişle **WcfSvcHost** uygulaması arka planda **Exit** seçeneği ile çıkılana kadar çalışmaya ve servisleri yayınlamaya devam edecektir.*(**Close** seçeneği yada **X** işareti kullanıldığında **WcfSvcHost** uygulaması **Tray Icon** olarak çalışmaya devam etmektedir.)* WcfSvcHost uygulamasının çalışma zamanındaki görüntüsü ise aşağıdaki gibidir.

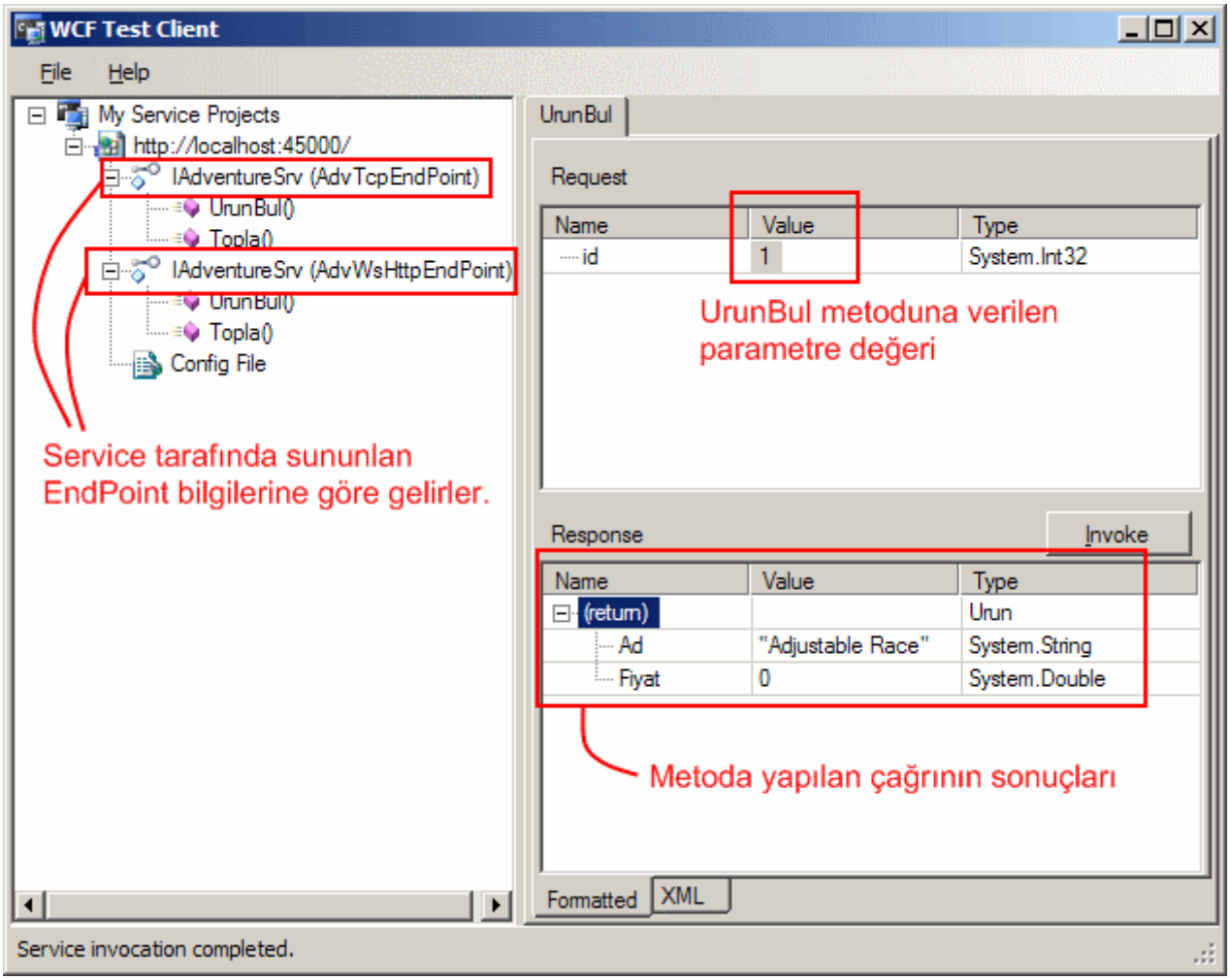


Dikkat edileceği üzere yayınlanan servisin adı, durumu ve Metadata içeriğinin alınabileceği URL adresi bilgileride gösterilmektedir. Servis uygulaması başarılı bir şekilde çalıştırıldıktan sonra istenirse **WcfTestClient** aracı yardımıyla istemcilerin denenmesine başlanabilir. **WcfTestClient** aracı aldığı parametrelere göre bir **Windows** uygulaması başlatır ve servisten aldığı **metadata** bilgilerine göre kullanılabilecek hizmetleri ve metodları gösterir. En basit kullanımında servis kütüphanesinde belirtilen **base address** bilgisi aşağıdaki gibi parametre olarak belirtilir.

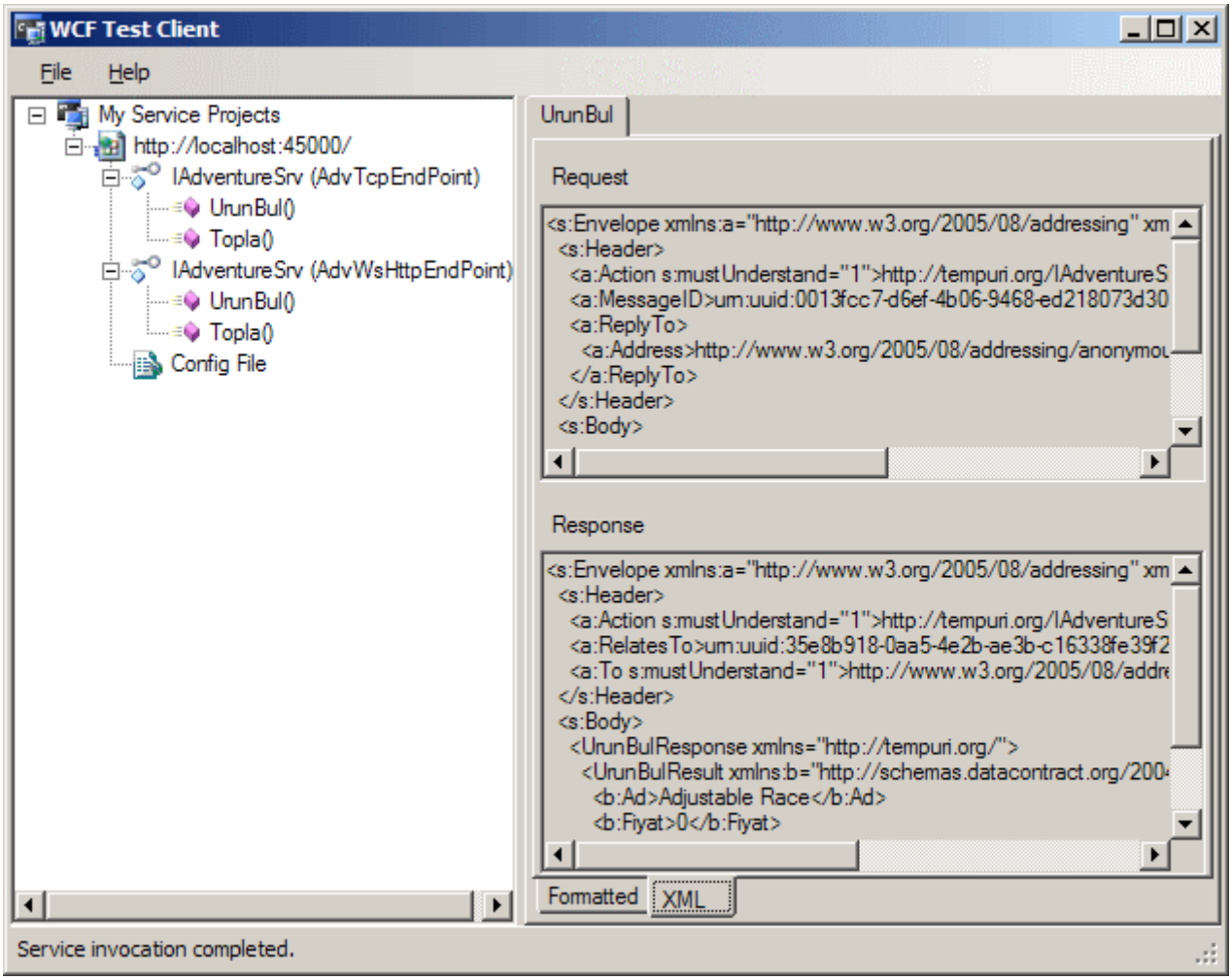
WcfTestClient http://localhost:45000/



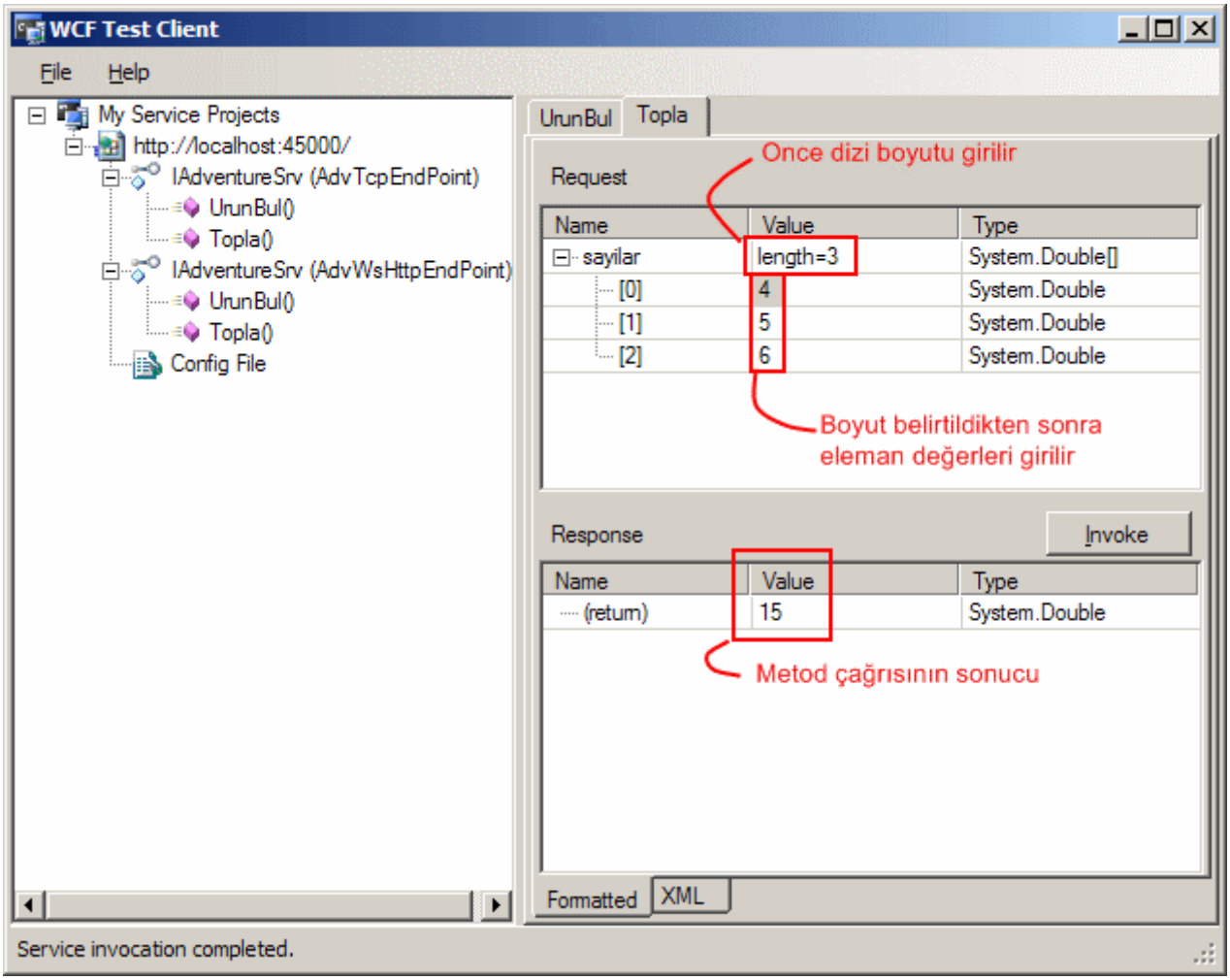
Bu işlemin arkasından **WcfTestClient** aracı bir **Windows** uygulaması çalıştıracak ve parametre olarak verilen adresin metadata bilgisinide kullanarak, çağırılabilecek hizmet noktalarını listeleyecektir. Arabirimde servis ile ilişkili metodlara çift tıklanarak çalıştırılmaları ve sonuçlarının görülmesi de sağlanabilir. Söz gelimi **UrunBul** isimli metod **1** değeri ile çalıştırıldığında aşağıdaki sonuçlar elde edilir.



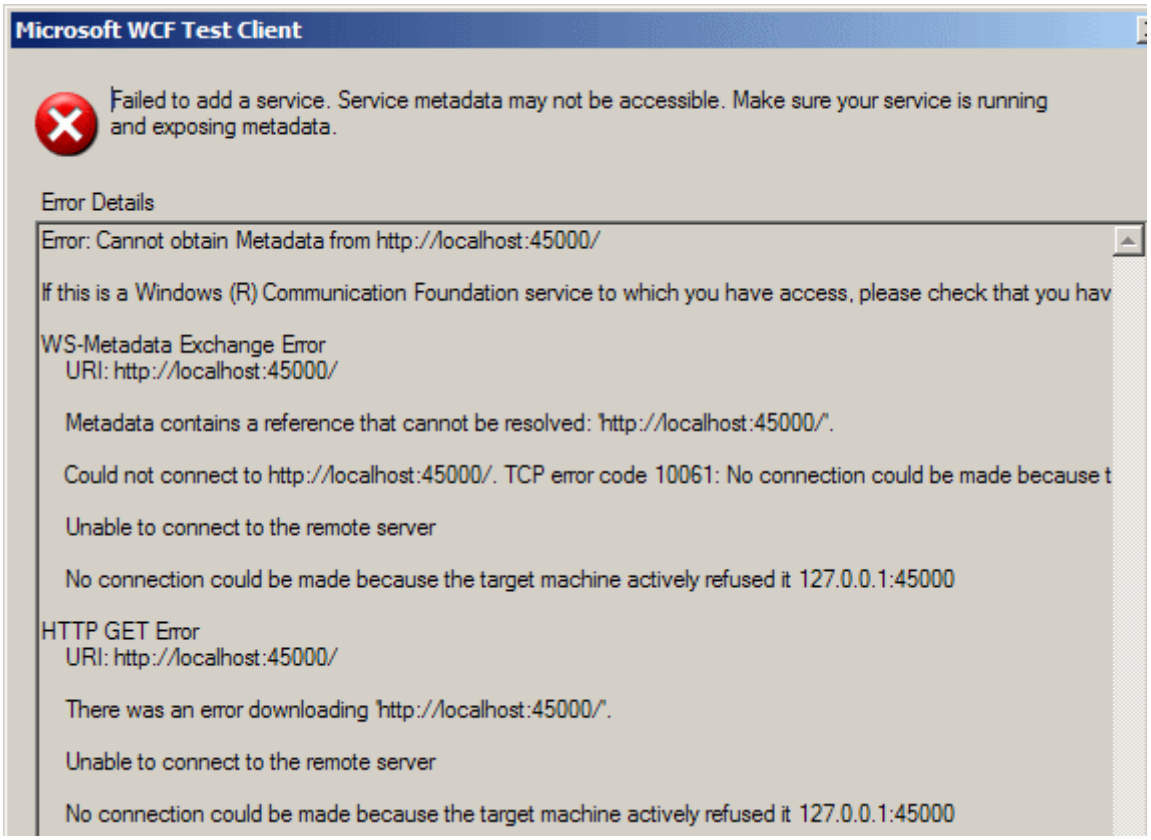
Invoke düğmesine basılması ile birlikte **UrunBul** metoduna bir çağrı gerçekleştirilir. Bu çağrının sonucunda elde edilen **Urun** nesne örneği ve veri içeriği **Response** sekmesinde görülmektedir. Burada **talep(Request)** ve **cevap(Response)** paketlerinin içeriklerinin **XML** tarafındaki hallerinede bakılabilir. Bunun için XML sekmesine geçilmesi yeterlidir**.** Bu durumda aşağıdaki çıktı elde edilir. *(Bu XML içeriklerinin daha önceden belirtilen paketlerin istemci tarafından manuel olarak hazırlanarak gönderilmesinde oldukça işe yarayacağı da göz önüne alınmalıdır.)*



Servis tarafında yer alan metodlardan **Topla** fonksiyonu parametre olarak **double** tipinden bir dizi almaktadır. Bu tip bir metodun **WcfTestClient** aracı ile çalıştırılması esnasında öncelikli olarak kaç adet değer gönderileceği(bir başka deyişle dizinin boyutu) belirlenir. Daha sonra ise değişken değerleri girilir ve fonksiyon çağırılır. Bu durum aşağıdaki şekilde görüldüğü gibi örneklenebilir.

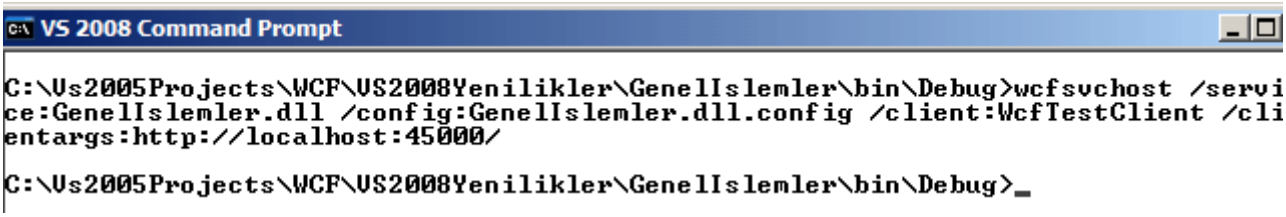


WcfTestClient aracının kullanımı sırasında eğer **Host** uygulama **(WcfSvcHost)** çalışmıyor yada herhangibir nedenle servislere ait **metadata** bilgileri çekilemiyorsa **çalışma zamanı istisnası(Runtime Exception)** alınır. örnekte WcfSvcHost uygulaması kapalıyken WcfTestClient çalıştırılmak istenmiş ve sonuç olarak aşağıdakine benzer bir hata ekranı alınmıştır.

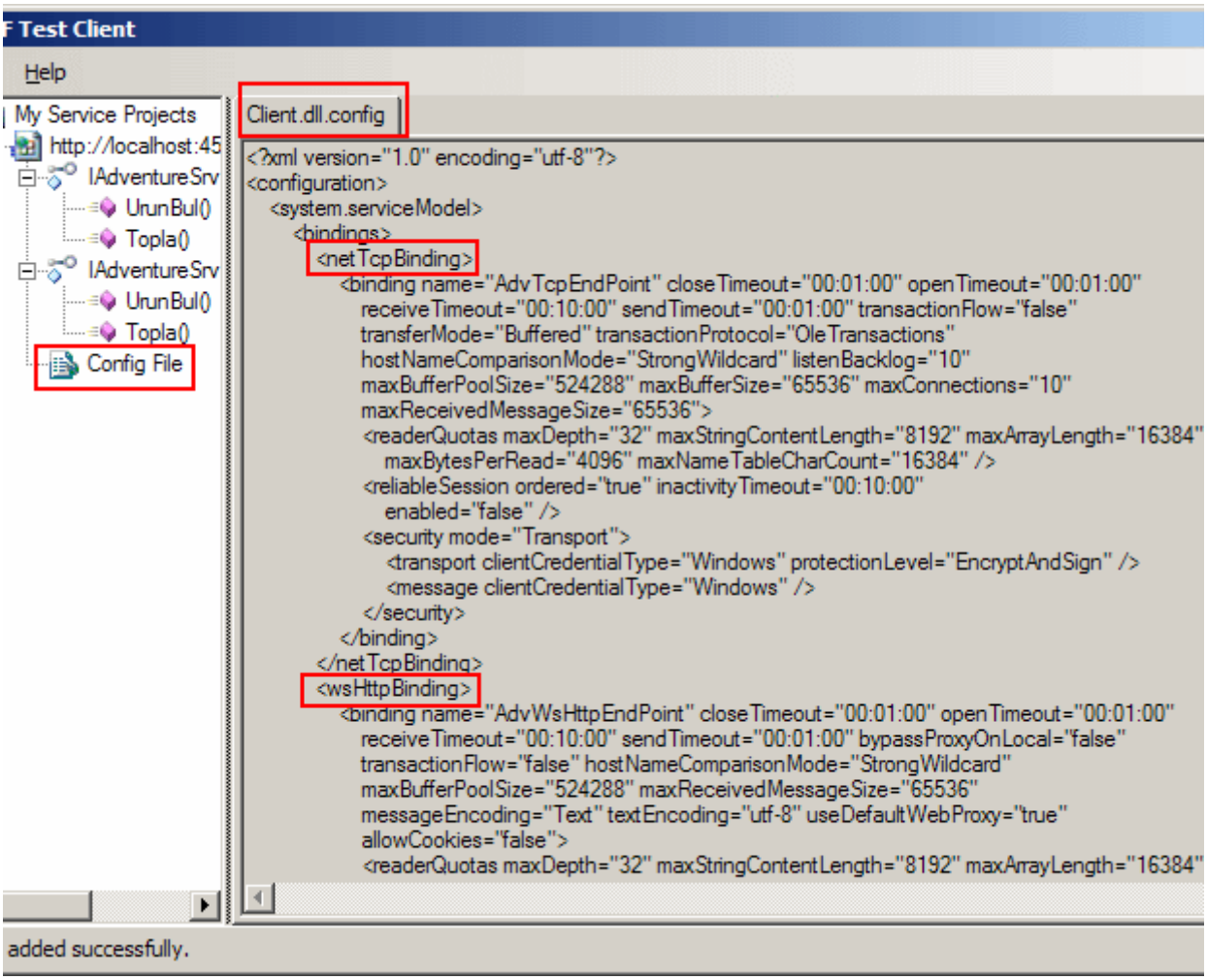


WcfSvcHost uygulamasının parametrik yapısı kullanılarak istenirse aynı anda **WcfTestClient** uygulamasınında çalıştırılması sağlanabilir ki **Visual Studio 2008** ortamının servis kütüphanesinin çalıştırılması sonrası gerçekleştirilen işlemde budur. Bunun için **WcfSvcHost** aracını aşağıdaki gibi kullanmak yeterlidir.

WcfSvcHost /service:GenelIslemler.dll /config:GenelIslemler.dll.config /client:WcfTestClient /clientargs:http://localhost:45000/

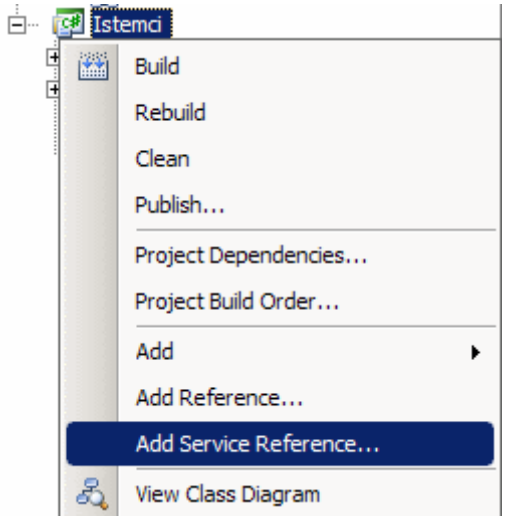


Client parametresinden sonra istemci uygulama olarak **WcfTestClient** işaret edilmektedir. Bununla birlikte WcfTestClient çalışırken gerekli olan **base address** bilgiside **clientargs** parametresinden sonra belirtilmektedir. Bu işlemin ardından önce **WcfSvcHost** uygulaması, sonrasında ise **WcfTestClient** uygulaması çalışacak ve hizmetler başlatılarak istemci tarafından kullanılabilir hale getirilecektir. WcfTestClient aracının yararlı özelliklerinden biriside istemci için üretilen konfigurasyon dosyasını gösteriyor olmasıdır. Dolayısıyla bu **config** dosyası istenirse gerçek istemci uygulamalar içinde kullanılabilir. örnekte bu içerik **config** isimli sekme altında yer almaktadır.

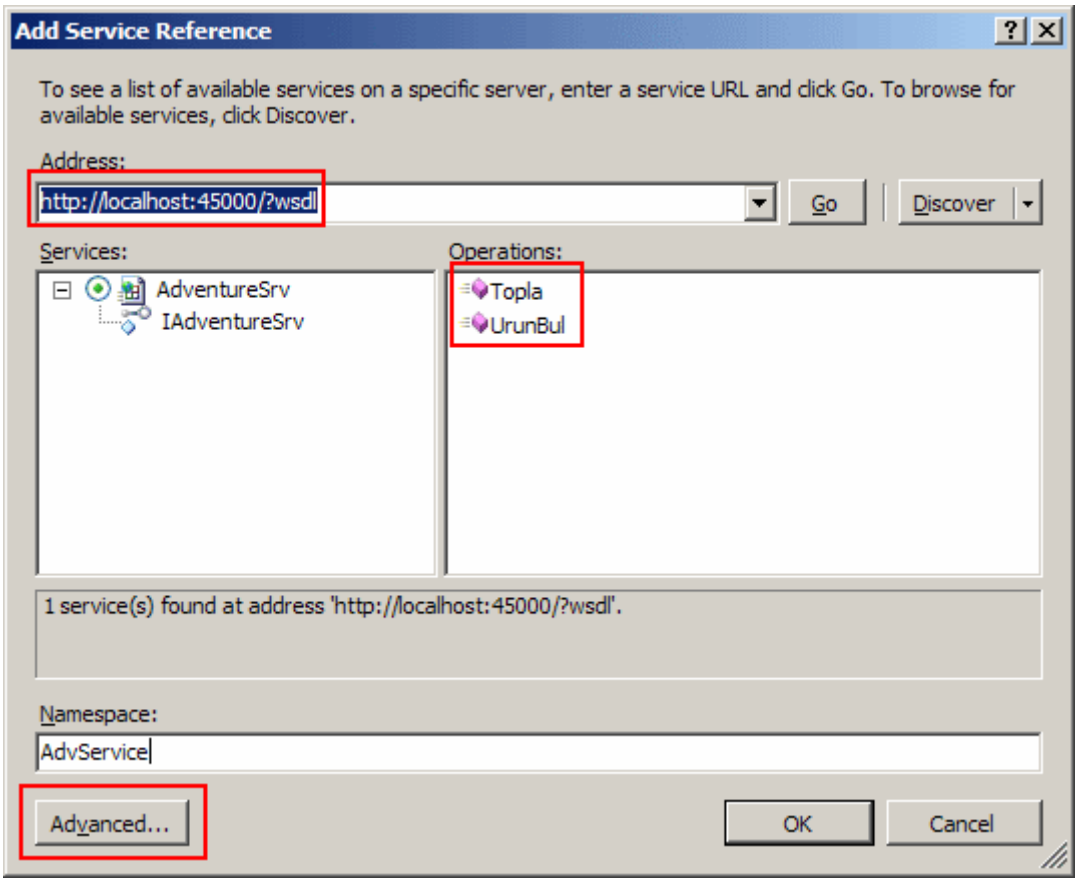


örnekte yer alan servis kütüphanesinde**(WCF Service Library),** hem **TCP** hemde **WS** bazlı **EndPoint** noktaları tanımlanmış olduğundan istemci için üretilen config dosyası içerisinde iki farklı **binding** elementinin yer aldığı açık bir şekilde görülmektedir.

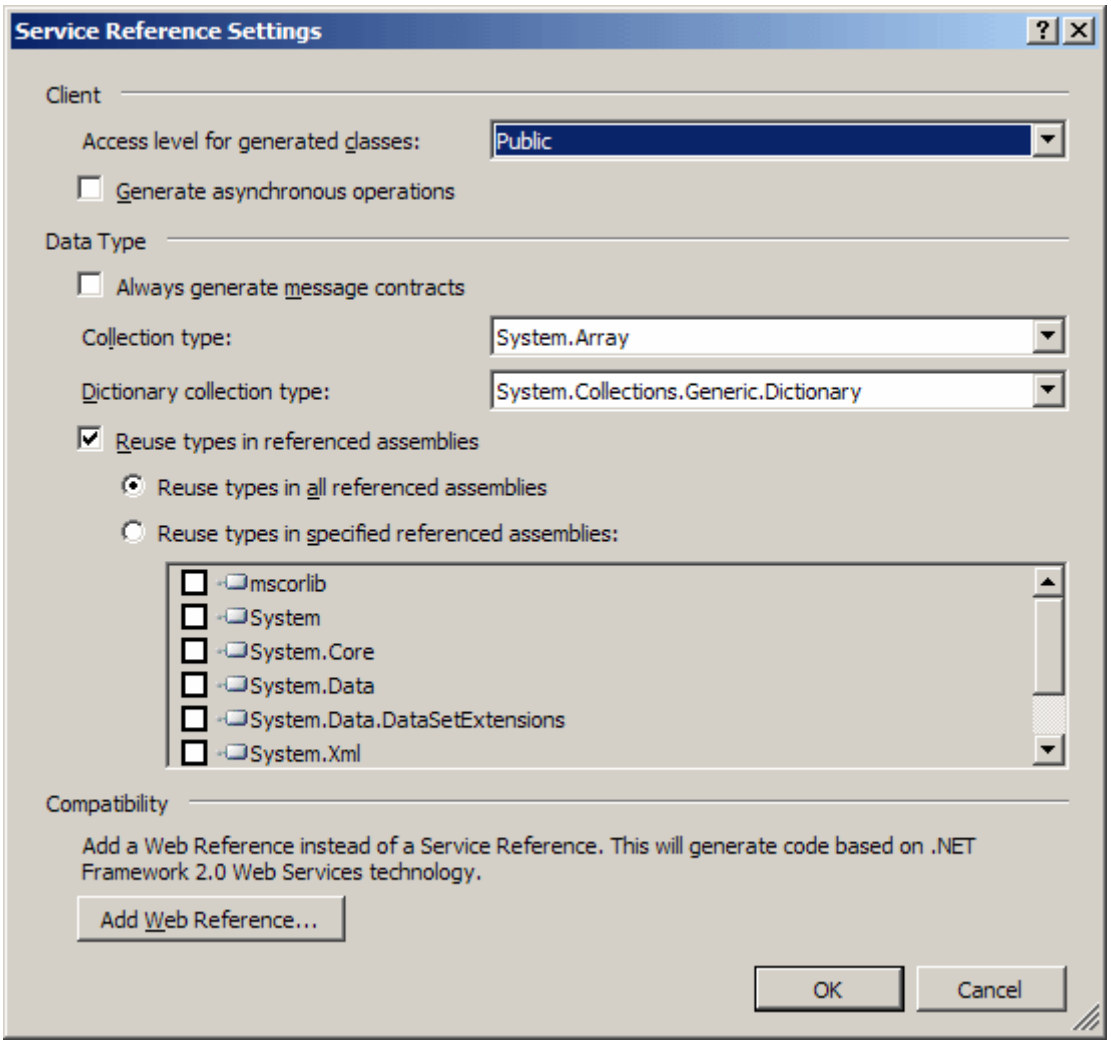
WCF istemcileri geliştirilirken önemli olan noktalardan biriside **Proxy** üretimidir. Bilindiği gibi Proxy nesneleri yardımıyla istemci tarafından servis operasyonlarına erişebilmek **nesne-metod(Object-Method)** ilişkisi çerçevesinde olabilmektedir. Proxy üretimi için **Visual Studio 2008** tarafında servis referanslarını eklemek gerekmektedir. Bu amaçla **Add Service Reference** seçeneği kullanılmaktadır. *(Bu seçenek zaten **WCF** servis eklentilerinin **Visual Studio 2005** yüklemesinden sonrada çıkmaktadır.)*



örnekte yer alan servis kütüphanesine ait servis referansını eklemek için öncelikli olarak **WcfSvcHost** uygulamasının çalıştırılması gerekmektedir. Bu işlemin ardından **base address** üzerinden **WSDL** içeriği talep edilebilir. Sonuç olarak basit bir **Console** uygulamasına **Add Service Reference** seçeneği yardımıyla, GenelIslemler.dll **WCF Service Library** projesi içerisinde tanımlanmış olan servis aşağıdaki ekran görüntüsünde olduğu gibi eklenebilir.



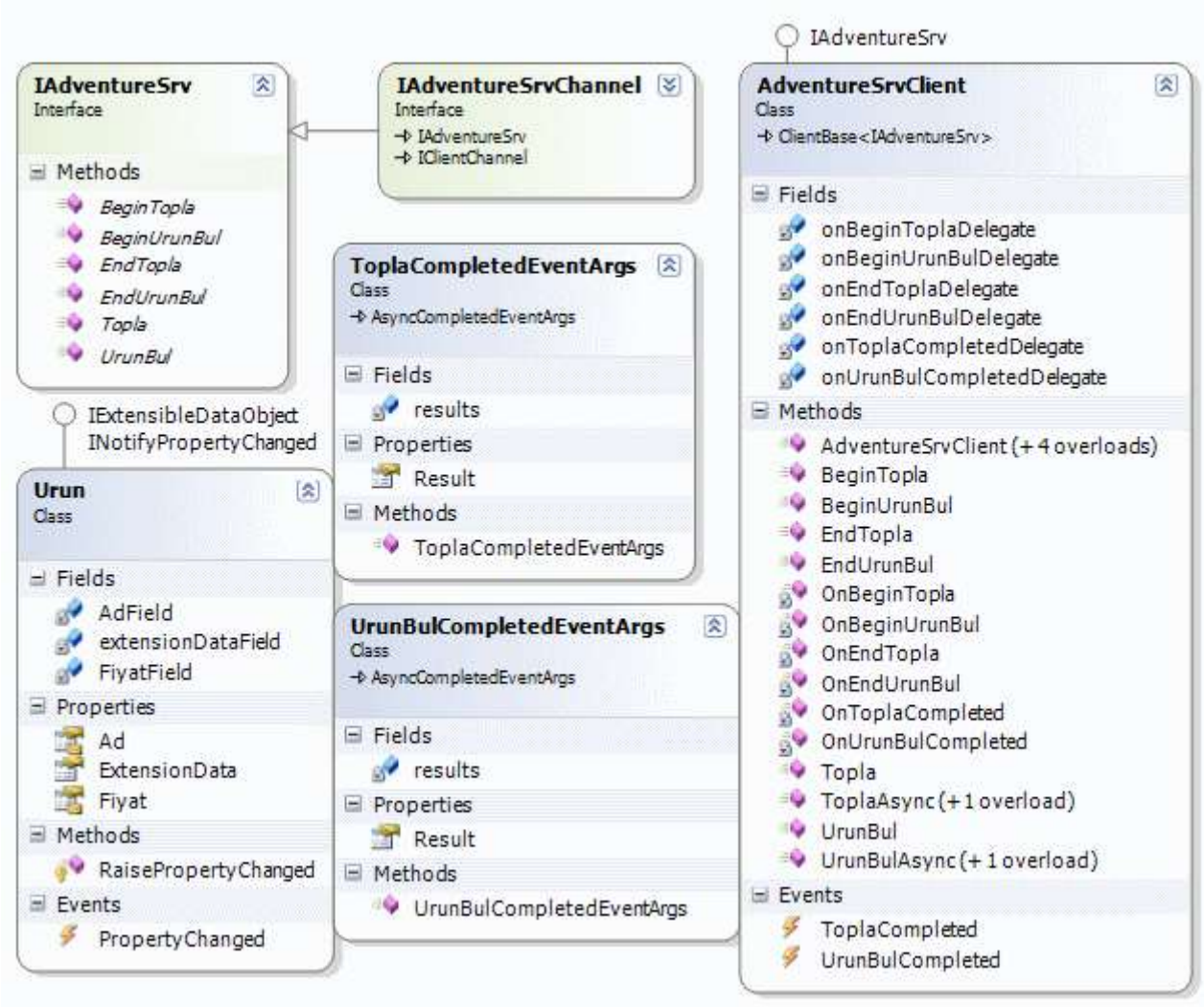
Görüldüğü gibi **WCFSvcHost** aracı ile çalıştırılan servise ait operasyonlar elde edilebilmektedir. Burada önemli olan fark **Advanced** sekmesine geçildiğinde servis ile ilişkili detaylı ayarlamaların yapılabileceği bir ekranla karşılaşılmasıdır.



Access level for generated classes seçeneği kullanıldığında, **proxy** içerisinde üretilecek olan tiplerin hangi **erişim belirleyicisi(Access Modifier)** ile oluşturulacağı belirtilmektedir. Tip bazında **Public** ve **Internal** olmak üzere iki farklı seçenek yer almaktadır. Bazı durumlarda **başka bir servis kütüphanesinin***(Söz gelimi başka bir WCF Servis kütüphanesi)* elde ettiği**proxy** tiplerinin sadece o **assembly** içerisinde geçerli olması istenebilir. Böyle bir vakada **Internal** seçeneğini işaretlemek gerekmektedir. İkinci olarak **Generate Asynchronous Operations** seçeneği işaretlendiğinde servis operasyonlarını asenkron olarak çağırabilmek için gerekli fonksiyonel alt yapının yüklenmesi sağlanmaktadır.

***Not :** Bazı durumlarda servis operasyonlarının uzun zaman alması söz konusudur. Bu gibi vakalarda istemcilerin operayon sonuçlarını beklemeden çalışmasına devam edebilmesi için bilinen asenkron çağırma tekniklerinden**(Polling, WaitHandle, Callbak, Even-Based)** yararlanılmaktadır.*

Yazının bu bölümünde, **Console** tipindeki istemci uygulamaya eklenecek olan servis referansı için **Internal** erişim belirleyicisi ve **asenkron** erişim seçeneği uygulanmaktadır. Buna göre istemci tarafına eklenen tiplerin **sınıf diagramındaki(Class Diagram)** görüntüsü ve içerikleri aşağıda olduğu gibidir.



Hatırlanacağı gibi **Xml Web Servisleri**, **Visual Studio 2005** sürümü ile birlikte **olay tabanlı asenkron çağrma(Event-Based Asynchronous Invoking)** modelini uygulamaya başlamıştır. Bu basit asenkron modelde, zaman alan servis operasyonları işlemlerini tamamladığında otomatik olarak tetiklenen bir **olay(Event)** sayesinde sonuçlar uygulama ortamına alınabilmektedir. Böylece istemci tarafında **BeginInvoke**, **EndInvoke** gibi metodların kullanıldığı **Polling**, **Callback**,**WaitHandle** gibi tekniklerde görece daha basit bir uygulama şekli ortaya çıkmaktaydı. Modelin uygulanması sırasında istemci tarafında bazı **olay(Event)** ve **temsilci(Delegate)** tanımlamaları oluşturulmaktadır. Aynı durum **Visual Studio 2008** sayesinde **WCF** servislerinede kolay bir şekilde uygulanabilir.

AdventureSrvClient isimli proxy sınıfına bakıldığında Topla ve UrunBul isimli metodlar için **Async** son eki ile biten versiyonlar yer almaktadır. Tahmin edileceği üzere bu metodlar yardımıyla ilgili servis operasyonlarına **asenkron** çağırlar gerçekleştirilebilir. Diğer

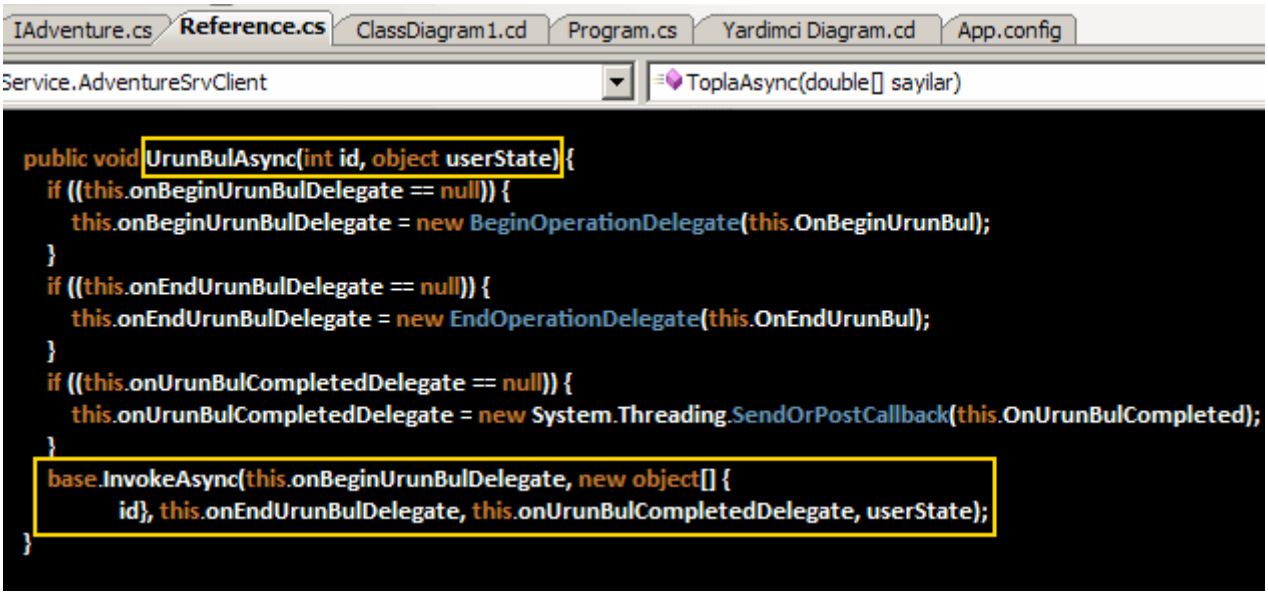
taraftan işlemler tamamlandıktan sonra devreye girecek olan olaylar ise **ToplaCompleted** ve **UrunBulCompleted** olarak eklenmiştir*(**Completed** kelimesi ile bittiklerine dikkat edilmelidir)*. Söz konusu olaylar devreye girdiğinde, operasyon sonuçlarını alabilmek içinse **CompletedEventArgs** kelimesi ile biten argüman sınıflarının oluşturulduğu görülebilir. İstemci tarafına yazılacak aşağıdaki kod parçası yardımıyla servis metodunun olay tabanlı(Event Based) olacak şekilde asenkron olarak çalıştırılması sağlanabilir.

```
using System;
using System.Threading;

namespace Istemci
{
   class Program
   {
      static void Main(string[] args)
      {
         // Proxy nesnesi AdvTcpEndPoint isimli EndPoint noktasını baz alacak şekilde oluşturulur.
         AdvService.AdventureSrvClient proxy = new Istemci.AdvService.AdventureSrvClient("AdvTcpEndPoint");
         // UrunBul metodu tamamlandıktan sonra devreye girecek olan UrunBulCompleted olayı yüklenir
         proxy.UrunBulCompleted += new EventHandler<Istemci.AdvService.UrunBulCompletedEventArgs>(proxy_UrunBulCompleted);
         proxy.UrunBulAsync(1); // UrunBul metoduna yapılan asenkron çağrı

         // Console tarafında sembolik olarak yazılmış kodlar
         // bu döngü devam ederken servis tarafındaki metodun sonuçlarının alınmasıyla birlikte proxy_UrunBulCompleted isimli olay metodu tetiklenmektedir.
         for (int i = 0; i < 10; i++)
         {
            Console.WriteLine("...");
            Thread.Sleep(1000); // Olayı daha iyi izlemek için ana thread 1 saniye duraksatılır
         }
      }

      // Asenkron olarak çağırılan servis metodu bittiğinde devreye girecek olan olay metodu
      static void proxy_UrunBulCompleted(object sender, Istemci.AdvService.UrunBulCompletedEventArgs e)
      {
         // UrunBul metodu geriye Urun tipinden nesne örneği döndürmektedir. Bu sebepten
```

UrunBulCompletedEventArgs sınıfının Result özelliği Urun tipinden bir değer döndürür. Bu sayede Ad ve Fiyat alanlarına erişilebilmektedir.

```
            Console.WriteLine(e.Result.Ad+" "+e.Result.Fiyat.ToString("C2"));
        }
    }
}
```

Elbette istenirse **Callback**, **WaitHandle** ve **Polling** modellerine göre asenkron erişimlerde gerçekleştirilebilir. Nitekim **Begin** ve **End** kelimeleri ile başlayan metodların eklenmelerinin nedenide budur. **Begin**ve **End** metodlarının oluşturduğu sıkıntılardan biriside özellikle **Windows** veya **WPF** uygulamalarında ortaya çıkartacağı thread senkronizasyon problemleridir. Meşhur **Illegal Cross Thread Exception** oluşumuna neden olabilecek bu durumda tedbir olarak metod invoker' lardan yararlanılmaktadır. Bununla birlikte proxy' nin türedigi **ClientBase<T> abstract sınıfı** kendi içerisinde potected erişim belirleyicisi ile işaretlenmiş **InvokeAsync** isimli bir metod içermektedir. Bu metod senkronize problemlerini ortadan kaldırmak üzere **proxy** sınıfı içerisindeki olay tabanlı asenkron fonksiyonlar tarafından kullanılmaktadır.

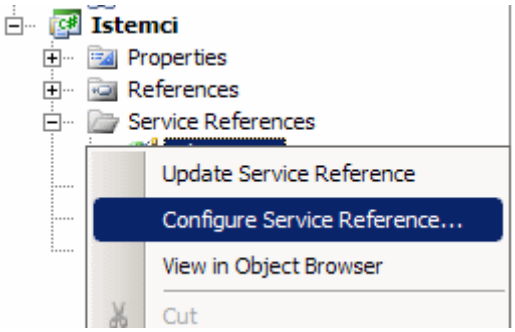


Dikkat edilecek olursa UrunBulAsync metodu son olarak **ClientBase<T>** içerisindeki **InvokeAsync** metodunu çalıştırmaktadır. Bu metodun parametrik yapısı aşağıdaki gibidir ve istemci tarafında **threadlerin** senkronize çalışmasını sağlayacak şekilde tasarlanmıştır.

```
protected void InvokeAsync(
    ClientBase<TChannel>.BeginOperationDelegate beginOperationDelegate
    , object[] inValues
    , ClientBase<TChannel>.EndOperationDelegate endOperationDelegate
    , SendOrPostCallback operationCompletedCallback
    , object userState
);
```

özellikle **SendOrPostCallback** isimli **temsilci(delegate)** senkronizasyon ile ilişkili geri bildirim metodunu işaret etmekle görevlidir.

Advanced sekmesinde dikkati çeken noktalardan bir diğeride servis operasyonlarında koleksiyon bazlı dönen türlerin istemci tarafında ele alınış şekillerinin değiştirilebilmesidir. Nitekim şu anda servis referansı eklenmiştir. Var olan referansa ait özellikle nasıl değiştirilebilir? Yine **Visual Studio 2008** ile gelen yeniliklerden biriside **Configure Service Reference** seçeneğidir.



Bu seçenek sayesinde, daha önceden eklenmiş olan bir servis referansına ait özellikler değiştirilebilmektedir. Koleksiyon döndüren opersayonların istemci tarafındaki durumunu daha net kavrayabilmek için servis tarafına eklenmiş aşağıdaki operasyon ve metodlar göz önüne alınabilir.

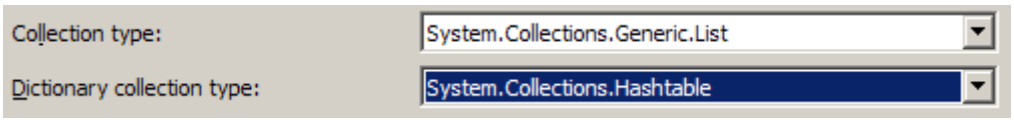
```
[ServiceContract]
interface IAdventureSrv
{
   // Diğer tanımlamalar
   [OperationContract]
   List<double> RastgeleSayilar(int baslangic, int bitis);

   [OperationContract]
   Hashtable Isimler();
}

class AdventureSrv
   :IAdventureSrv
{
   // Diğer metod uyarlamaları

   public List<double> RastgeleSayilar(int baslangic,int bitis)
   {
      List<double> liste = new List<double>();
      Random rnd=new Random();
      for (int i = baslangic; i < bitis; i++)
         liste.Add(rnd.NextDouble());
```

```
        return liste;
    }

    public Hashtable Isimler()
    {
        Hashtable dict = new Hashtable();
        dict.Add(1001, "Burak");
        dict.Add(1002, "Mayk");
        dict.Add(1005, "Conn");
        return dict;
    }
}
```

Servis tarafında **List<double>** ve **Hashtable** tiplerinden değer döndüren iki operasyon yer almaktadır. Normal şartlarda **WCF** servisine ait **proxy** sınıfı üretilirken **List<T>** gibi koleksiyonlar için**T** tipinden bir **dizi(Array)** baz alınmaktadır. **Hashtable**, **SortedList** gibi koleksiyonlar içinse **Dictionary<object,object>** tipi ele alınmaktadır. Bu nedenle istemci uygulamadaki servis içeriği **Update Service Reference** seçeneği ile güncelleştirilirse **Isimler** ve **RastgeleSayilar** adlı fonksiyonların aktarımı aşağıdaki gibi olacaktır.

```
public System.Collections.Generic.Dictionary<object, object> Isimler() {
    return base.Channel.Isimler();
}

public double[] RastgeleSayilar(int baslangic, int bitis) {
    return base.Channel.RastgeleSayilar(baslangic, bitis);
}
```

Ancak istenirse bu aktarım tipleri değiştirilebilir. Bunun için **Advanced** sekmesinde yer alan **Collection Type** ve **Dictionary Collection Type** değerlerini değiştirmek yeterli olacaktır. Aşağıdaki ekran görüntüsünde bu durum gösterilmektedir.



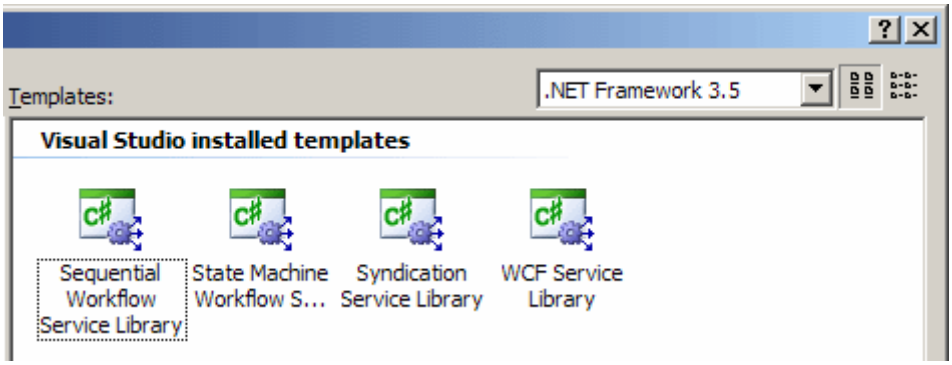
Bu işlemin ardından ilgili operasyonların **Proxy** sınıfı içerisindeki yapılarının aşağıdaki gibi değiştirildiği görülebilir.

```
public System.Collections.Generic.List<double> RastgeleSayilar(int baslangic, int bitis)
{
    return base.Channel.RastgeleSayilar(baslangic, bitis);
}
```

```
public System.Collections.Hashtable Isimler() {
   return base.Channel.Isimler();
}
```

Dikkat edileceği üzere, servis tarafında sunulan operasyonlardaki dönüş tiplerinin aynen istemci tarafındaki proxy sınıfınada aktarılması sağlanmaktadır. Tabi bu noktada istemci tarafındaki proxy sınıfının **.Net** bağımlı hale geldiğinede dikkat etmek gerekmektedir.

Visual Studio 2008 geliştirme ortamı **.Net Framework 3.5** şablonu altında birden fazla **WCF kütüphanesi** sunmaktadır.



WCF servisleri, **Workflow Foundation** içerisindede zaman zaman kullanılmaktadır. **Workflow Foundation** mimarisinde iki farklı iş akışı tipi vardır. **Sequential** ve **State Machine**. Bir WCF servisi bu tipteki iş akışları içerisinden kullanılabilir yada iş akışı bir WCF servisinin parçası haline gelebilir. Bu noktada **Sequential Workflow Service Library** ve S**tate Machine Workflow Service** Library şablonları kullanılabilir. Bilindiği gibi **.Net Framework 3.5** ile WCF tarafında **RSS**, **Atom** formatlı yayınlamalara destek verilmektedir. Bu tip bir proje şablonu için **Syndication Service Library** kullanılabilir.

Buraya kadar anlatılanlara göre Visual Studio 2008 ile birlikte gelen WCF yenilikler aşağıdaki tablo ile özetlenebilirler.

özellik	**Açıklama**
WcfSvcHost	Herhangibir **Host** uygulaması yazılmasına gerek kalmadan **WCF** servis kütüphaneleri test amaçlı olarak yayınlanabilmektedir. **Visual Studio 2008** varsayılan olarak **WCF Servis** kütüphaneleri için bu aracı kullanmaktadır.
WcfTestClient	Yayınlanan servislerin test edilmesi için kullanılan Windows uygulamasıdır. Servise ait operasyonların anında görülmesi, kullanılması, **talep(Request)** ve **cevap(Response)**paketlerinin **data** veya **XML** formatında okunabilmesi, farklı **EndPoint** noktalarının test edilebilmesi gibi imkanlar sunmaktadır. **Visual Studio 2008** varsayılan olarak WCF Servis kütüphanelerinin

	çalıştırılmasında **WcfSvcHost** uygulamasından sonra bu programı çalıştırırak anında testin yapılabilmesini sağlamaktadır.
Add Service Reference - Advanced Sekmesi	**Advanced** sekmesindeki ayarlar yardımıyla oluşturulacak **proxy** sınıfı ve ilişkili tiplerine ait pek çok detay ayarlanabilir.
Public/Internal	İstemciye eklenen servis referansı içerisindeki tiplerin erişim belirleyicileri**(Access Modifiers)** vakaya göre **public** yada **internal** olarak set edilebilir.
Koleksiyon Eşleştirme	**List<T>** gibi koleksiyonlar(IEnumerable<T>, IList<T> vb...) istemci tarafında **T[]** dizileri şeklinde ele alınırlarken, **Hashtable** gibi koleksiyonlar **Dictionary<object,object>** olarak yorumlanır. Bu eşleştirme Advanced sekmesindeki **Collection Type** ve **Dictionary Collection Type** seçenekleri yardımıyla değiştirilebilir.
Asenkron Proxy Metodların üretimi	Web servislerindeki hazır üretime benzer şekilde olay tabanlı asenkron (event based asynchronous) yürütme desenlerinin eklenmesi sağlanabilir. üstelik bu uyarlamada **thread**senkronizasyonunu etkili hale getirmeyi kolaylaştırıcı fonksiyonellikler vardır.
Configure Service Reference	İstemci tarafına eklenmiş olan bir servisin koleksiyon eşleştirme, erişim belirleyicisi gibi pek çok özelliği sonradan **Configure Service Reference** seçeneği yardımıyla değiştirilebilir.
Yeni Library Şablonları	özellikle **WWF(Windows Workflow Foundation)** tarafına ve **.Net Framework 3.5** ile **WCF**' e getirilen Web programlama modelinin bir ürünü olan **Syndication** yayınlama için ek proje şablonları gelmektedir.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde yakın zamanda son haliyle yayınlanan Visual Studio 2008 ürününün Windows Communication Foundation için getirdiği yenilikleri basit seviyede ele almaya çalıştık. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

VS2008Yenilikler.rar (70,17 kb)

WCF - Kod Tarafından Yönetmek (2008-03-07T04:13:00)

wcf,

Windows Communication Foundation ile geliştirilen **Servis Yönelimli Uygulama(SOA-Service Oriented Architecture)** çözümlerinde **konfigurasyon bazlı(Configuration Based)** geliştirme süreci oldukça yaygındır. Konfigurasyon dosyaları

içerisinde yer alan bilgiler yardımıyla WCF çalışma zamanı(WCF Runtime) ortamı otomatik olarak bazı işlemler gerçekleştirir. Söz gelimi **istemci(Client)** ve **servis(Service)** arasında taşınacak olan mesajların çözümlenmesi**(Encoding)**, bunların seçilen **bağlayıcı tipin(Binding Type)** belirlediği protokole göre aktarılması gibi **alt yapı(Infrastructure)** hazırlıkları otomatik olarak gerçekleştirilir. Hatta istemcinin servis üzerinden talep ettiği bir operasyon(Service Operation), servis tarafındaki konfigurasyon bilgilerinden yola çıkılarak hazırlanan çalışma zamanı sayesinde, anlamlı bir metod çağrısı haline dönüşür.

Ne varki bazı vakalarda, **WCF** altyapı hazırlıklarının konfigursayon bazlı olması tercih edilmez. Bunun en büyük nedenlerinden bir taneside, konfigurasyon bilgilerinin **XML(eXtensible Markup Lanuage)** bazlı açık **text** dosyalarında duruyor olmasıdır. Buda ilgili dosyanın dışarıdan değiştirilebileceği anlamına gelmektedir. Her ne kadar servis tarafının **Host** edileceği uygulama **administrator(veya bu gruba dahil kullanıcıların)** yetkisindede olsa, söz gelimi **HTML Metadata Publishing** özelliğinin konfigurasyon içerisinde, program koduna girmeye gerek kalmadan, yanlışlıklada olsa değiştirilmesi istenmeyebilir. çünkü bunun etkisi sonrasında istemcilerin servise ait metadata bilgisini çekememesi gibi durumlar ortaya çıkmaktadır. Bu gibi bazı sebeplerden dolayı, **WCF** alt yapı hazırlıklarının özellikle servis tarafında iken, **kod bazında(Code Based)** gerçekleştirilmesi tercih edilebilir. Bu yazıda ağırılıklı olarak kod tarafında gerekli hazırlıkların nasıl yapılabileceği gibi konulara değinilmektedir.

WCF mimarisi içerisinde çok sayıda **CLR tipi(Common Language Runtime-Type)** yer almaktadır. Bu tiplerin bazıları genişletilebilir ve **çalışma zamanında(Runtime)** kullanılabilir. Hatta var olan tiplerden yararlanarak özel durumlar için birleşik tipler tasarlanabilir. Söz gelimi var olan **bağlayıcı tiplerden(Binding Types)** bir kaçının bir arada kullanılacağı özelleştirilmiş bir bağlayıcı tip tasarlamak mümkündür. Burada **CustomBinding** isimli CLR tipi önemli rol oynamaktadır.

Bu bölümde çok göze batan WCF tipleri ele alınmaya çalışılmaktadır. Ancak herşeyden önce **WCF** çalışma zamanını kavramakta yarar vardır. Bu nedenle **Host** uygulamanın(servis tipinin yayınlayan programın) çalışmaya başladığı andan itibaren ilerlemek daha doğrudur. Host uygulama çalıştığında, servis için tanımlanan **EndPoint** bilgilerinden yararlanılarak birer **ChannelListener** nesnesi ve bir **kanal yığını(Channel Stack)** oluşturulur.

NOT : *Bilindiği gibi* ***EndPoint*** *içerisinde servisin* ***Address***, ***Binding*** *ve* ***Contract*** *bilgileri yer almaktadır. Bu bilgiler çalışma zamanı ortamının hazırlanmasında önemli değer sahiptir. Bu sebepten dolayı* ***kanal yığını(Channel Stack)*** *hazırlanırken* ***EndPoint*** *içerisindeki bilgilerden yararlanılır.*

ChannelListener nesneleri aslında **EndPoint** noktalarını ilgili kanallara bağlamakla görevlidir. öyleki, **URI(Uniform Resource Identifier)** üzerinden bir mesaj geldiğinde, **ChannelListener** nesne örneği ilgili mesajı kanal yığınının(Channel Stack) en altında yer alan **iletişim kanalına(Transport Channel)** aktarır. Gelen mesaj içeriğinin

iletişim kanalı açısından bir önemi yoktur. Nihayetinde bu bilgi **byte** tipinden bir **akımdır(Stream)**. Bununla birlikte iletişim kanalı gelen mesajı aldıktan sonra, **çözümleme kanalına(Encoding Channel)** iletir. çözümleme kanalının, gelen mesajlar içerisindeki operasyonel talepleri alıp nesne-metod ilişkisine dönüştürmek gibi önemli bir görevi vardır. çözümleme kanalı **SOAP(Simple Object Access Protocol), plain text, binary** data veya **MTOM(Message Transmision Optimization Mechanism)** gibi formatları kullanmaktadır. *(Hatta .Net Framework 3.5 ile gelen yenilikler ile birlikte JSON-JavaScript Object Notation formatının kullanılabilmeside olanaklı hale gelmiştir.)*

İletişim(Transport) ve **çözümleme(Encoding) kanalları bağlayıcıların(Bindings)** olmassa olmaz parçalarıdır. Bir başka deyişle **kanal yığını(Channel Stack)** içerisinde mutlaka ve mutlaka var olmaları gerekmektedir. Ancak duruma göre bu kanalların arkasına **güvenilir oturumların(Reliable Sessions)** sağlanması ve **Replay** saldırılarının engellenmesi, günveliğin sağlanması, transaction yönetiminin gerçekleştirilmesi gibi işlemler için ek kanallarda ilave edilebilir. Bu durumda zaten birden fazla kanalın bir arada ele alındığı özel bir kanal yığının oluşumu söz konusudur.

***Not : İkili çözümleme Kanalı(Binary Encoding Channel) WCF**' e özeldir. Bu nedenle **interoperability** desteği olmayan senaryolarda daha çok ele alınır. **MTOM(Messsage Transmision Optimization Mechanism)** çözümleme kanalı, büyük boyutlu verilerin **binary formatta** ve **interoperability**' nin önemli olduğu vakalarda ele alınır.*

*Bu açıdan bakıldığında, tüm iletişim kanalları varsayılan olarak bazı çözümleyicileri kullanırlar. Söz gelimi **HTTP/HTTPS** tabanlı kanallar varsayılan olarak **text**formatlı çözümleme kanallarını ele alırken, **TCP** binary formatlı çözümleme kanallarını kullanır. Elbette bu noktada geliştiriciler kanal ve çözümleme kanallarını istedikleri gibi özelleştirebilir ve farklı kombinasyonları ister konfigurasyon bazlı, ister kod bazlı olacak şekilde geliştirebilirler.*

Şu ana kadar anlatılanlar göz önüne alındığında kanal yığının içerisinde yer alan tiplerden geçilerek tepeye ulaşıldığı görülür. Kanal yığınının en tepe noktasına ulaşılmasının ardından ilgili mesaj bir **ChannelDispatcher** nesnesi tarafından karşılanır. **ChannelDispatcher** nesnesi ilgili mesajı alır ve bir **EndPointDispatcher** nesnesine aktarır.*(ChannelDispatcher arkasında mesajın iletilebileceği birden fazla EndPointDispatcher nesnesi olabilir)* **EndPointDispatcher** nesnesinin görevi, gelen mesajın içeriğine göre uygun olan servis metodunun çalıştırılmasıdır. Bununla birlikte **EndPointDispatcher** nesnesi gelen mesaj içerisinden servis tarafındaki metod için gerekli parametreleride almaktadır. **ChannelDispatcher** ve**EndPointDispatcher** nesneleri, **ServiceHost** nesnesi örneklendiğinde otomatik olarak üretilirler.

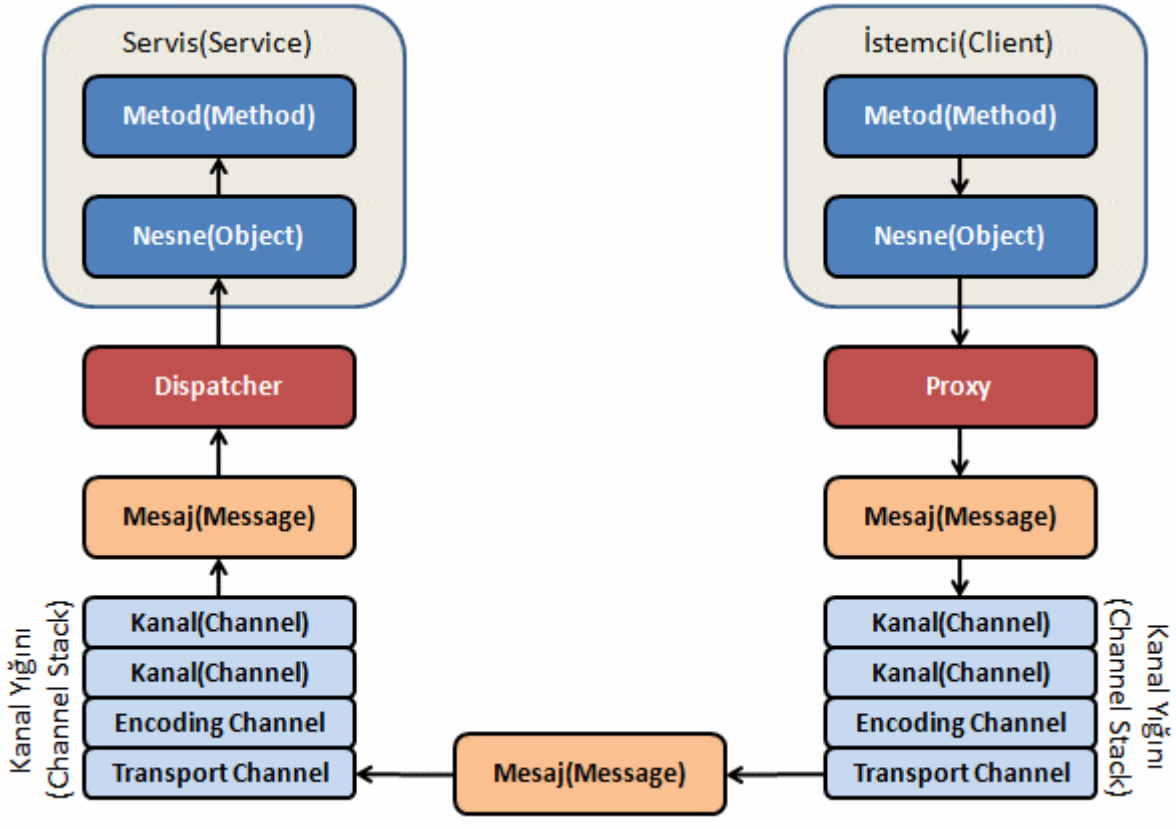
İstemcinin servis tarafından talep ettiği operasyon çağrısı kanal yığınından geçerek metod çağrısı haline geldikten sonra , metod üzerinden alınan sonuçlar aynı yolla geriye dönecektir. İstemci, servisten gelen mesajı ele alırken, **WCF** çalışma zamanının

ürettiği **ChannelFactory** nesnesinden yararlanır. Bununla birlikte istemci tarafında önem arz eden nesnelerden biriside **Proxy** bileşenidir. **Proxy** nesnesi ilgili metod çağrılarının **request** mesajlarına çevrilmesinden ve gelen cevaplarında(Responses) istemci programın anlayacağı kodlara dönüştürülmesinden sorumludur.

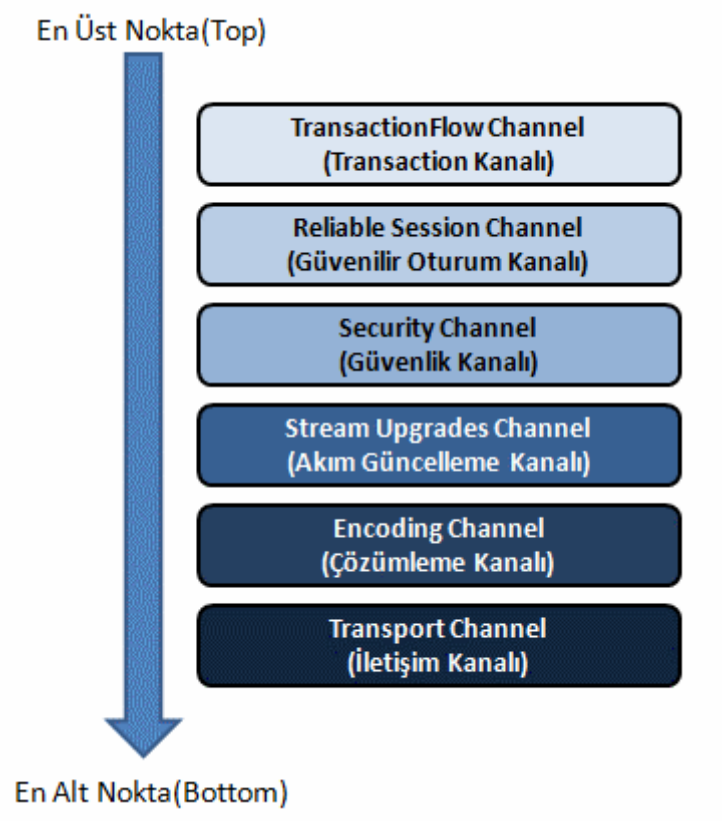
WCF çalışma ortamı ile ilişkili önemli noktalardan biriside **davranış(Behavior)** tanımlamalarıdır. Davranış tipleri sayesinde **WCF** altyapısı içerisindeki parçaların nasıl yürüyeceği belirlenebilir. **.Net Framework 3.0** ve **3.5** içerisinde tanımlanmış olan sayısız **davranış tipi(Behavior Type)** vardır. Bu tipler sayesinde örneğin, **transaction** içerisindeki operasyonların toplu olarak nasıl gerçekleştirileceği, verinin nasıl ve ne şekilde serileştirileceği, mesajların gönderilmesi yada alınması sırasında kullanılacak olan**ehliyetlerin(credentials)** ların tayini, servis üzerinde **debug** yapılıp yapılamayacağı, **HTTP** üzerinden **metadata publishing** aktarımına izin verilip verilmeyeceği, exception detaylarının istemci tarafınada fırlatılıp fırlatılmayacağı ve daha pek çok çalışma zamanı davranışı belirlenebilir. Davranışlar, WCF alt yapısı içerisinde yer alan tiplere veya üyelere(Members) uygulanabilirler. Söz gelimi kimi davranışlar sadece metodlara, kimileri bir servisin tamamına, kimileride **sözleşmelere(Contracts)** uygulanabilirler. Tabi istenirse özelleştirilmiş**davranış tipleri(Behavior Type)** tasarlanabilir. Davranışlar kod bazında **imperative** yada konfigurasyon dosyaları içerisinden **declerative** tarzda kullanılabilir.

Not : Davranışlar(Behaviors) *geliştiriciler tarafından yazılan bir servisin, daha sonra yöneticiler tarafından farklı şekillerde davranabilmesini sağlamak amacıyla geliştirilmiş yapılardır. Hazır olan davranış tipleri sayesinde, alt yapıya(Infrastructure) müdahale etmeye, ek kod yazmaya veya hazırlık yapmaya gerek kalmadan istenen değişimler gerçekleştirilebilmektedir.*

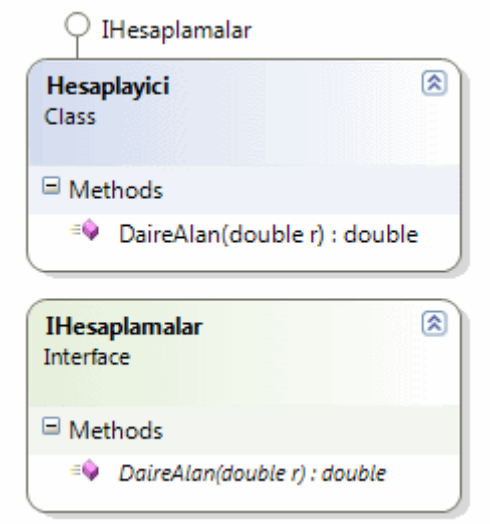
Buraya kadar anlatılanlar göz önüne alındığında **WCF(Windows Communication Foundation)** mimarisinin, servis ve istemci taraflarınıda içeren alt yapısı kabaca aşağıdaki şekilde görüldüğü gibi ele alınabilir.



Şekildende görüleceği üzere **kanal yığının(Channel Stack)** en alt noktasında bulunan **Transport Channel** ve **Encoding Channel** kanalarının ardından farklı tipte kanallar gelebilmektedir. Daha öncedende bahsedildiği gibi bu kanallar **Transaction**, **Security**, **Relaibility** gibi konularla ilgili olabilir. Burada söz konusu olan **kanal yığını(Channel Stack)** kombinasyonu kod tarafında **CustomBinding** tipi yardımıyla yada konfigurasyon bazında customBinding elementi ile karşılanabilmektedir. Ebeltte eklenecek olan kanalların mantıklı ve düzgün bir sırada olmaları şarttır. Burada söz konusu olan sıra için aşağıdaki şekil göz önüne alınmalıdır.



Bu kadar teorik bilgiden sonra adım adım basit bir örnek üzerinde ilerleyerek kod bazında **WCF** alt yapı hazırlıklarını nasıl yapılabileceği incelenmeye başlanabilir. öncelikli olarak bir**WCF servis kütüphanesi(WCF Service Library)** geliştirilmelidir. Söz konusu servis kütüphanesi içerisinde yer alacak olan tipler(sözleşme-Service Contract ve uygulayıcı sınıf) basit olarak aşağıda görüldüğü gibidir.

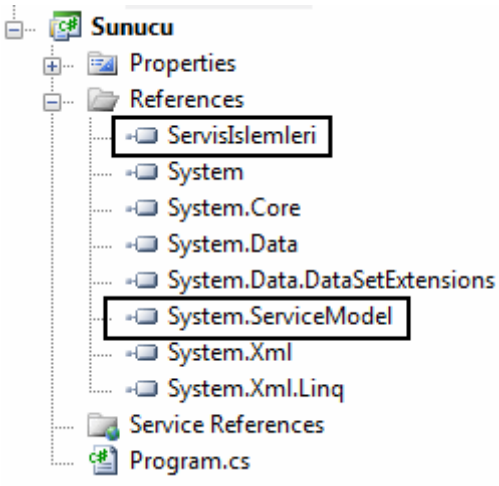


IHesaplamalar arayüzünün ve **Hesaplayici** sınıfının içerikleri aşağıdaki gibidir.

```
using System;
using System.ServiceModel;
```

```
namespace ServisIslemleri
{
   [ServiceContract]
   public interface IHesaplamalar
   {
      [OperationContract]
      double DaireAlan(double r);
   }

   public class Hesaplayici
      : IHesaplamalar
   {
      #region IHesaplamalar Members

      public double DaireAlan(double r)
      {
         return Math.PI * r * r;
      }

      #endregion
   }
}
```

Konunun basit olarak ele alınması amacıyla sözleşme ve uygulayıcı tipler yalın bir şekilde tasarlanmışlardır. önemli olan kısım servis uygulaması içerisinde programatik olarak yapılacak ayarlamalardır. Bu amaçla **Visual Studio 2008** ortamında geliştirilen **Console** uygulamasında **System.ServiceModel.dll** ve **servis kütüphanesi(WCF Service Library) assembly**' larının referanslarının mutlaka var olması gerekmektedir.

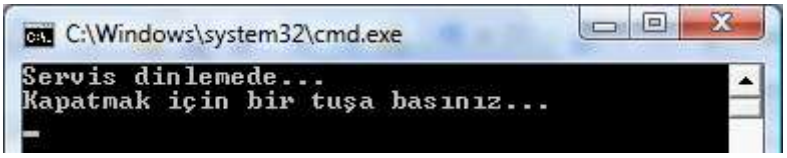


Servis uygulamasının kodları basit olarak aşağıdaki gibi tasarlanabilir.

```
using System;
using System.ServiceModel.Channels; // Kanal tipleri için eklenen isim alanıdır.
using System.ServiceModel;
using ServisIslemleri;

namespace Sunucu
{
   class Program
   {
      static void Main(string[] args)
      {
         CustomBinding binding = new CustomBinding(); // özel bir bağlayıcı tip sınıfı örneklenir. Dikkat edileceği üzere bilinen bir Binding tipi oluşturmaktan bir farkı yoktur.

         // İlk örnek olarak Reliable Session için bir Binding tipi oluşturulur ve eklenir.
         ReliableSessionBindingElement rBinding =
new ReliableSessionBindingElement(true); // Mesajların gönderildiği sırada iletilmesini sağlamak için yapıcı metoda true değeri verilmiştir. İtenirse Ordered özelliğine true değeri atanaraktanda bu işlem sağlanabilir.
         rBinding.FlowControlEnabled = true;
         rBinding.MaxRetryCount = 3; // Mesajların başarılı şekilde iletimi için maksimum tekrar sayısı belirlenir.
         binding.Elements.Add(rBinding); // Oluşturulan Bindin elementi eklenir.

         SecurityBindingElement sBinding=SecurityBindingElement.CreateSecureConversationBindingElement(SecurityBindingElement.CreateSspiNegotiationBindingElement());
         sBinding.LocalServiceSettings.DetectReplays = true; // Replay ataklarını kontrol et.
         binding.Elements.Add(sBinding); // Oluşturulan SecureBindingElement, CustomBinding nesnesinin ELements koleksiyonuna eklenir.

         // çözümleme kanalı için gerekli binding elementi oluşturulur
         TextMessageEncodingBindingElement tBinding =
new TextMessageEncodingBindingElement();
         binding.Elements.Add(tBinding); // CustomBinding nesne örneğinin elements koleksiyonuna eklenir.

         // İletişim kanalı için gerekli element oluşturulur
         TcpTransportBindingElement tcpBinding =
new TcpTransportBindingElement();
         tcpBinding.TransferMode = System.ServiceModel.TransferMode.Buffered; // Mesajların transfer modu belirlenir
         binding.Elements.Add(tcpBinding); // Elements koleksiyonuna eklenir.
```

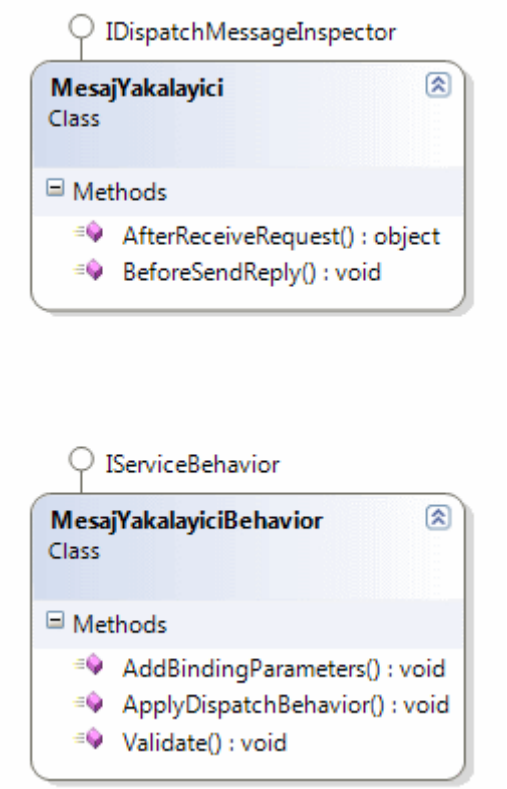
```
        /* ServiceHost nesnesi oluşturulur. Parametre olarak servis sözleşmesini uygulayan
tipin adı verilir. ServiceHost nesnesi örneklendiğinde, kanalların oluşturulmasınıda sağlar.
Aynı zamanda talep edilen servis örneklerinin yaşam sürelerini yönetir. Bunları
gerçekleştirirken ChannelListener, ChannelDispatcher, EndPointDispatcher tiplerinin
otomatik olarak örneklenmesinide sağlar.*/
        ServiceHost host = new ServiceHost(typeof(Hesaplayici));

        // EndPoint eklenir. İlk parametre sözleşme tipi(Contract), ikinci parametre
bağlayıcı(Binding Type), son parametre ise adres(Address) bilgisidir.
        host.AddServiceEndpoint("ServisIslemleri.IHesaplamalar", binding,
"net.tcp://localhost:45000/HesaplamaServisi");
        host.Open(); // host açılır. Open metodundan sonra ChannelListener gelen talepleri
dinlemeye başlar. Gelen mesajları kanal yığınına devreder. ChannelDispatcher nesnesi ise
bu mesajları kanal yığınının en üstünden alır ve uygun olan EndPointDispatcher nesnesine
devreder.
        Console.WriteLine("Servis dinlemede...\nKapatmak için bir tuşa basınız...");
        Console.ReadLine();
    }
  }
}
```

Burada sadece örnek olması açısından bazı kanallar için gerekli elementler oluşturulmakta ve **CustomBinding** nesne örneğinin **Elements** koleksiyonu içerisinde toplanmaktadır. Söz konusu element nesne örneklerinin pek çok özelliği ve metodu burada göz ardı edilmektedir. Amaç programatik olarak özel bir bağlayıcı tipin oluşturulması ve **ServiceHost** nesne örneği üzerinden kullanılabilmesidir. Söz konusu uygulama çalıştırıldığında aşağıdaki ekran görüntüsü ile karşılaşılır.



WCF alt yapısını programalamakla ilişkili olarak **Microsoft** kaynaklarının gösterdiği örneklerde yapılan çalışmalardan biriside istemci ve servis arasındaki mesajların yakalanmasıdır. *(Yakalama işlemini takiben mesaj içeriklerinin örneğin herhangibir algoritmaya göre sıkıştırılması, bazı doğrulama süreçlerinden geçirilmesi gibi ek fonksiyonellikler yapılabilmektedir.)* Bu işlem için **IDispatchMessageInspector arayüzünü(Interface)** uygulayan bir sınıfı yazmak yeterlidir. Daha sonra söz konusu sınıf örneği servis tarafında **davranış(Behavior)** olarak ele alınmaktadır. Söz konusu davranışın yazılması içinse **IServiceBehavior** arayüzünü uygulayan bir sınıfın geliştirilmesi ve içerisindeki uygun metodlarda mesaj yakalamak üzere tasarlanan tipin ele alınması gerekmektedir. Davranışın servis seviyesinde uygulanması halinde, gelen tüm mesajların yakalanması söz konusudur. Mesaj yakalama süreci ile ilişkili tipler istemci tarafınada uygulanabilir.

Yazının bu bölümünde mesaj yakalamak için gerekli işlemlerin programatik olarak nasıl gerçekleştirileceği ele alınmaktadır. Bu amaçla servis uygulamasına aşağıdaki sınıf diagramında(Class Diagram) görülen tiplerin eklenmesi yeterlidir.



MesajYakalayici isimli
sınıf **IDispacthMessageInspector arayüzünü(Interface)** uygulamaktadır. Söz konusu arayüz **AfterReceiveRequest** ve **BeforeSendReply** isimli iki metodun uygulanmasını beklemektedir. Tahmin edileceği üzere **AfterReceiveRequest** metodu ile, istemcinin talep ettiği operasyonun servis tarafında çalıştırılmasından hemen önce mesajın yakalanması mümkün olmaktadır. Hatta istenirse mesaj içeriği burada değiştirilebilir ki bu çok dikkatli bir şekilde ele alınması gereken bir durumdur. **BeforeSendReply** isimli metod ise, servis tarafından talep edilen metod tamamlandıktan sonra devreye girmektedir. örnekteki sınıfta sadece ve sadece gelen ve giden mesajlara ait bilgiler ele alınmaya çalışılmaktadır. MesajYakalayici sınıfına ait kod içerisi aşağıdaki gibidir.

```
using System;
// IDispatchMessageInspector arayünüzün yer aldığı isim alanıdır.
using System.ServiceModel.Dispatcher;
using System.ServiceModel.Channels;
using System.Xml;
using System.ServiceModel.Description;

namespace Sunucu
{
   class MesajYakalayici
```

```
    :IDispatchMessageInspector
  {
    #region IDispatchMessageInspector Members

    // Servis üzerinden talep edilen metod çalıştırılmadan hemen önce devreye giren metoddur.
    // ilk parametrenin ref olarak tanımlandığına dikkat edelim. Yani gelen mesajda yapılacak olan değişiklikler çalışma ortamını doğrudan etkileyecektir.
    public object AfterReceiveRequest(ref System.ServiceModel.Channels.Message request, System.ServiceModel.IClientChannel channel, System.ServiceModel.InstanceContext instanceContext)
    {
      // System.Runtime.Serialization referansı eklenmediği takdirde MessageId özelliği için derleme zamanı hatası alınmaktadır.
      Console.WriteLine("\n\n");
      Console.WriteLine("********Request Bilgisi*********");
      Console.WriteLine("Message Id : " + request.Headers.MessageId.ToString());
      Console.WriteLine("MessageVersion Addressing : " + request.Headers.MessageVersion.Addressing);
      Console.WriteLine("To : " + request.Headers.To.AbsolutePath);
      Console.WriteLine("Action : " + request.Headers.Action);
      Console.WriteLine("Encoder : "+request.Properties.Encoder);

      MessageBuffer buffer = request.CreateBufferedCopy(Int32.MaxValue);
      request = buffer.CreateMessage();
      Console.WriteLine("Mesaj Body :"+buffer.CreateMessage().GetBody<XmlElement>().InnerXml);

      return null;
    }

    // Servis metodu tamamlandığında devreye giren metoddur.
    public void BeforeSendReply(ref System.ServiceModel.Channels.Message reply, object correlationState)
    {
      Console.WriteLine("\n\n");
      Console.WriteLine("********Reply Bilgisi*********");
      Console.WriteLine("Action : " + reply.Headers.Action);
      Console.WriteLine(reply.ToString());
    }
    #endregion
  }
}
```

Yazılan bu mesaj yakalama sınıfının **EndPointDispatcher** nesnelerinin her birinde uygulanabilmesi için servis tarafında bir davranış belirtilmesi gereklidir. **MesajYakalayiciBehavior** sınıfının yazılmasının amaçlarından biriside budur. **EndPointDispatcher** noktlarına mesaj yakalama işlemini üstlenen sınıfların eklenmesi için **ApplyDispatcherBehavior** metodundan yaralanılmaktadır.

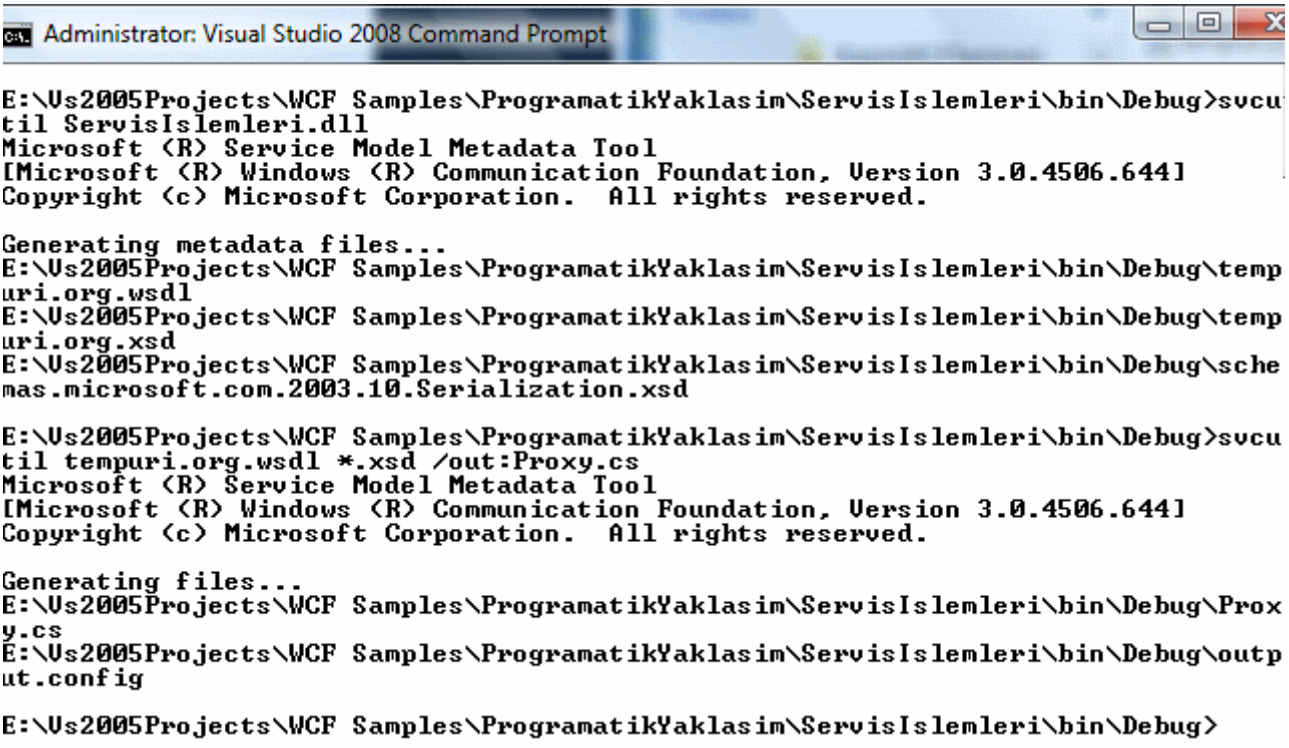
```
using System;
// IDispatchMessageInspector arayünüzün yer aldığı isim alanıdır.
using System.ServiceModel.Dispatcher;
using System.ServiceModel.Channels;
using System.Xml;
using System.ServiceModel.Description;

namespace Sunucu
{
   // Mesaj yakalama sınıfının uygulanması için bir davranış tipi (Behavior Type) geliştirilir
   public class MesajYakalayiciBehavior
      : IServiceBehavior // Davranış tipleri IServiceBehavior arayüzünden türerler.
   {
      #region IServiceBehavior Members

      /* Ek bağlayıcı parametrelerin ilave edilebilmesini sağlayan metoddur. Söz konusu dış ortam parametreleri metoda BindingParameterCollection tipinden aktarılır. WCF çalışma zamanı tarafından servisin dinlediği her bir URI için bir kere çağrılır.*/
      public void AddBindingParameters(ServiceDescription serviceDescription,
System.ServiceModel.ServiceHostBase serviceHostBase,
System.Collections.ObjectModel.Collection<ServiceEndpoint> endpoints,
BindingParameterCollection bindingParameters)
      {
      }

      /* ServisHost nesnesine behavior nesnesinin uygulandığı metoddur. örnekteki mesaj yakalayıcının uygulanacağı yerdir. Servis tarafından kullanılan her bir EndPointDispatcher nesnesi için birer mesaj yakalayıcı bu metod içerisinden eklenebilir. */
      public void ApplyDispatchBehavior(ServiceDescription serviceDescription,
System.ServiceModel.ServiceHostBase serviceHostBase)
      {
         /* Aşağıdaki kod parçası sayesinde, EndPointDispatcher' lara ulaşan her mesaj yada işletilen servis metodundan EndPointDispatcher' a dönen her mesaj MesajYakalayici nesne örneği üzerinden geçmek durumundadır. */
         // ChannelDispatchers içerisindeki her bir ChannelDispatcher nesnesini ele al
         foreach (ChannelDispatcher cd in serviceHostBase.ChannelDispatchers)
         {
            // O anki ChannelDispatcher içerisindeki her bir EndPointDispatcher nesnesini ele al
```

```
            foreach (EndpointDispatcher ed in cd.Endpoints)
            {
               // MesajYakalayici nesne örneğini MessageInspectors koleksiyonuna ekle
               ed.DispatchRuntime.MessageInspectors.Add(new MesajYakalayici());
            }
         }
      }

      /* Servisin çalışma zamanında gerekli özellikleri sağlayıp sağlamadığının
denetlenebileceği yerdir.Şartlara uymayan durumlarda sözleşmenin geri çevrilmesi
amacıyla metod içerisinde exception fırlatılması gibi işlemler yapılabilir */
      public void Validate(ServiceDescription serviceDescription,
System.ServiceModel.ServiceHostBase serviceHostBase)
      {
      }

      #endregion
   }
}
```

Artık servis tarafında **MesajYakalayıcıBehavior** isimli davranışın uygulanma işlemi gerçekleştirilebilir. Bunun için host isimli nesne örneği üzerinden **Open** metodu çağırılmadan hemen önce aşağıdaki kod parçasının yazılması yeterlidir.

```
host.Description.Behaviors.Add(new MesajYakalayiciBehavior());
```

Söz konusu davranışın eklenmesi için **ServiceHost** nesne örneğinin **Description** özelliği üzerinden ulaşılan **Behaviors** özelliğinin **Add** metodu kullanılmaktadır. Elbette sonuçları görebilmek için istemci tarafından **talep(Request)** gelmesi gerekmektedir. İstemci uygulama olayın kolay bir şekilde analiz edilebilmesi ve özellikle mesaj yakalayıcının etkilerinin görülebilmesi için basit bir **Console** projesi olarak geliştirilmektedir. Yazıdakine benzer senaryolarda çoğunlukla **svcutil** aracından yararlanılarak **istemci(Client)** için gerekli olan **proxy sınıfı(Class)** ve **config**dosyasının üretilmesi yolu tercih edilir. Yada **metadata publishing** seçeneği aktif bırakılarak istemcilerin servis referanslarını eklemeleri sağlanabilir. Bazen güvenlik nedeniyle metadata publishing seçeneğinin kapatıldığı durumlar söz konusudur. örnek senaryoda istemci tarafı için gerekli olan **proxy** sınıfı **svcutil** aracı yardımıyla örneklenmekte ve otomatik üretilen config dosyası kullanılmaktadır. Bu amaçla svcutil aracından aşağıdaki ekran görüntüsünde olduğu gibi yararlanılması yeterlidir.



Bu işlemin arkasından üretilen **Proxy.cs** ve **App.config** dosyası örnek bir **Client** uygulamasında ele alınabilir. üretilen config dosyası otomatik olarak **basicHttpBinding** bazlı bir **bağlayıcı tipi(Binding Type)** kullanmaktadır. Oysaki örnekte yer alan servis tarafı **CustomBinding** tipini ele almaktadır. Bu sebepten istemci tarafındaki config dosyasının içeriği aşağıdaki gibi değiştirilemelidir.

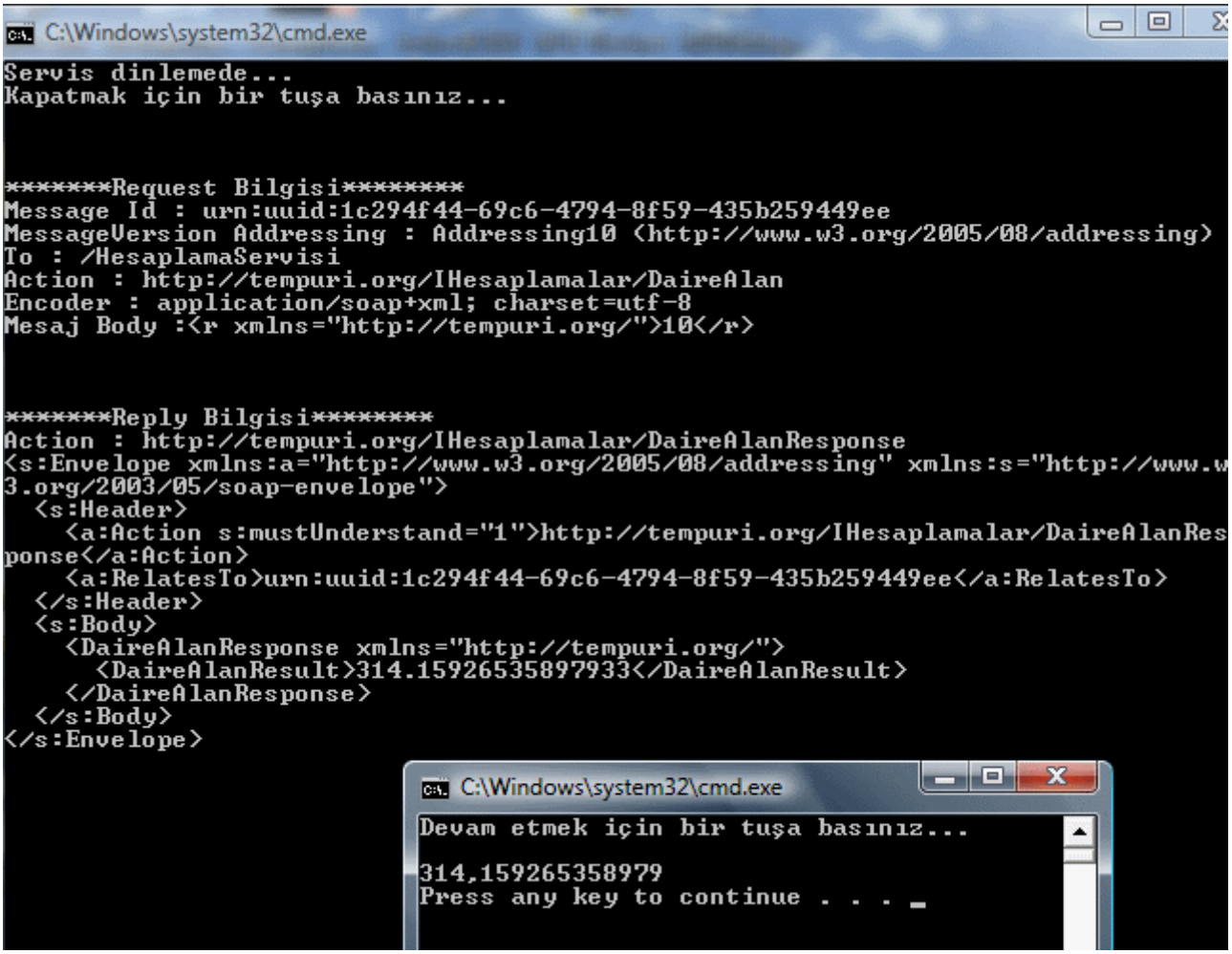
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <customBinding>
        <binding name="HesaplamaServisiBindingConfig">
          <reliableSession />
          <security authenticationMode="SecureConversation"
requireSignatureConfirmation="false">
            <localClientSettings/>
            <secureConversationBootstrap />
          </security>
          <textMessageEncoding />
          <tcpTransport />
        </binding>
      </customBinding>
    </bindings>
    <client>
      <endpoint
address="net.tcp://localhost:45000/HesaplamaServisi" binding="customBinding" bindin
```

```
gConfiguration="HesaplamaServisiBindingConfig" contract="IHesaplamalar"
name="HesaplamalarClientEndPoint" />
      </client>
   </system.serviceModel>
</configuration>
```

özel bağlayıcı tiplerin konfigurasyon dosyası içerisinde tanımlanması için **bindings** elementi altında yer alan **customBinding** alt elementinden yararlanılır. Config dosyası içerisinde dikkat edileceği üzere servis tarafındaki **CustomBinding** bildirimlerine uygun olacak şekilde bazı tanımlamalar yapılmaktadır. Buna göre güvenilir oturumlar için **relaibleSession** elementi, güvenlik için **security** elementi, text bazlı mesaj çözümlemesi için **textMessageEncoding** elementi ve son olarak **TCP** bazlı iletişimi işaret edecek şekilde **tcpTransport** elementi kullanılmaktadır. Elbetteki bu elementlerin doğru sırada yazılmalarıda oldukça önemlidir. **Console** uygulamasının kodları ise aşağıdaki gibi geliştirilebilir.

```
using System;

namespace Istemci
{
   class Program
   {
      static void Main(string[] args)
      {
         HesaplamalarClient proxy =new
HesaplamalarClient("HesaplamalarClientEndPoint");

         Console.WriteLine("Devam etmek için bir tuşa basınız...");
         Console.ReadLine();
         Console.WriteLine(proxy.DaireAlan(10).ToString());
      }
   }
}
```

Uygulama test edildiğinde servis ve istemci ekranlarında aşağıdaki sonuçların alındığı görülecektir.



Dikkat edileceği üzere servis tarafındaki mesaj dinleyici sınıf içerisindeki metodlar devreye girerekten, istemciden gelen ve istemciye gönderilen paket içeriklerinin yakalanması ve bunlar hakkında bilgi alınması işlemleri başarılı bir şekilde gerçekleştirilmiştir. Peki istemci açısından bakıldığında aynı işlemler kod tarafından nasıl yapılabilir. Bunun için öncelikli olarak istemci tarafında, servis operasyonlarının bildirimlerini içeren bir arayüz(Interface) tipinin tasarlanmış olması gerekmektedir. Bu tipin aşağıdaki gibi tasarlandığı düşünülsün.

```
using System;
using System.ServiceModel;

namespace Istemci
{
   [ServiceContract]
   public interface IHesaplamalarV2
   {
      [OperationContract(Action = "http://tempuri.org/IHesaplamalar/DaireAlan",
ReplyAction = "http://tempuri.org/IHesaplamalar/DaireAlanResponse")]
      double DaireAlan(double r);
```

```
    }
}
```

Bu arayüz tipinin görevi çalışma zamanı için gerekli **proxy** nesnesi üretildiğinde, ilgili nesne referansını taşıyabilmektir. Ayrıca, **DaireAlan** metoduna ait operasyonel bilgilerin,**OperationContract** niteliği içerisinde **Action** ve **ReplyAction** özellikleri ile nasıl tanımlandığına dikkat edilmelidir. Servis tarafındaki sözleşme içerisinde herhangibir **namespace** tanımlanmamış olduğundan, varsayılan olarak **http://tempuri.org** ön ifadesi tüm operasyon bilgilerinin başında yer almaktadır. İstemci tarafındaki kodlarda dikkat edilmesi gereken en önemli nokta **CustomBinding** tipinin servis tarafındakine uygun olacak şekilde tasarlanmasıdır. Bu amaçlar istemci tarafındaki kodların aşağıdaki gibi revize edilmesi yeterlidir.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Channels;

namespace Istemci
{
    class Program
    {
        static void Main(string[] args)
        {
            CustomBinding binding = new CustomBinding();

            ReliableSessionBindingElement rBinding = new
ReliableSessionBindingElement();
            binding.Elements.Add(rBinding);

            SecurityBindingElement sBinding =
SecurityBindingElement.CreateSecureConversationBindingElement(SecurityBindingElem
ent.CreateSspiNegotiationBindingElement());
            binding.Elements.Add(sBinding);

            TextMessageEncodingBindingElement eBinding = new
TextMessageEncodingBindingElement();
            binding.Elements.Add(eBinding);

            TcpTransportBindingElement tcpBinding = new
TcpTransportBindingElement();
            binding.Elements.Add(tcpBinding);

            IHesaplamalarV2 proxy =
ChannelFactory<IHesaplamalarV2>.CreateChannel(binding, new
EndpointAddress("net.tcp://localhost:45000/HesaplamaServisi"));
```

```
            Console.WriteLine("Devam etmek için bir tuşa basınız...");
            Console.ReadLine();
            Console.WriteLine(proxy.DaireAlan(10).ToString());
        }
    }
}
```

İstemcinin ilgili operasyonel çağrıları yapabilmesi için bir **proxy** nesne örneğine ihityaç vardır. Bu nesne örneğinin üretilmesi için **ChannelFactory** sınıfının **CreateChannel**metodundan yararlanılmaktadır. Dikkat edileceği üzere, bu metodun ilk parametresi bağlayıcı tipi ifade etmektedir. örnekteki bağlayıcı tipi **CustomBinding** türünden olup servis tarafındaki bildirime uygun olacak kanal bilgilerini içermektedir. Diğer taraftan ikinci parametre ile servisin adres bilgisi işaret edilir. Uygulama bu haliyle çalıştırıldığındada aynı sonuçların elde ediliği görülebilir.

WCF mimarisinde istenirse istemci tarafından servise gönderilecek olan mesajların manuel olarak hazırlanmasıda mümkündür. Bu Web servislerinde SOAP bazlı mesajların içeriklerinin hazırlanıp gönderilmesi ile benzer bir durumdur. özellikle WCF istemcilerinin, WCF harici servisler ile haberleşmesi gerektiği durumlarda mesajların manuel olarak hazırlanması söz konusu olabilir. Böyle bir durumda istemci tarafında herhangibir proxy nesnesi olmadan operasyon çağrıları yapılabilmektedir. Bu konuya ilerleyen yazılarda değinmeye çalışacağım. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

ProgramatikYaklasim.rar (68,06 kb)

WCF - Ajax ve Json Desteği (2008-02-25T04:19:00)

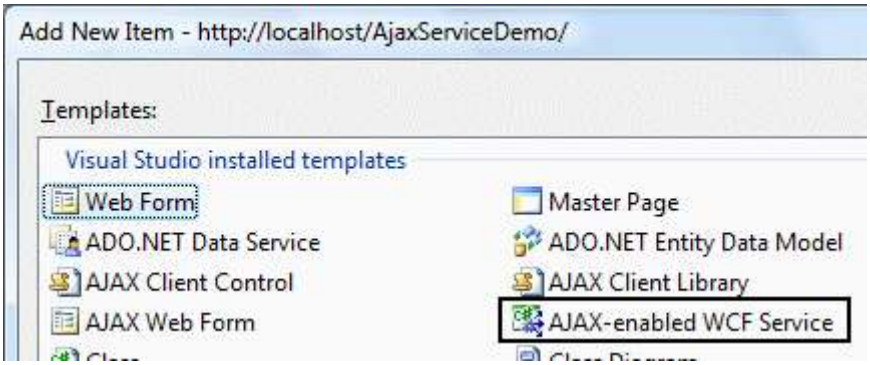
wcf,

Son yıllarda özellikle Web uygulamalarında **AJAX(Asynchronous Javascript And XML)** mimarisi oldukça yaygın bir şekilde kullanılmaktadır. özellikle **sunucu taraflı(Server-Side)** çalışan **Asp.Net** gibi web uygulaması geliştirme modellerinde **istemciler(Clients)** tarafından sunucuya(Server) doğru gerçekleştirilen **POST** işlemleri sırasında, sayfanın tamamının gönderilmesi söz konusudur. **AJAX** modeli sayesinde istemci tarafında yer alan sayfanın tüm içeriği yerine sadece değiştirilmesi istenen parçaların sunucuya gönderilmesi, işlenmesi ve cevapların alınarak **tarayıcı uygulama(Browser Application)** üzerinde gösterilmesi mümkün olmaktadır. Böylece sayfanın gerçektende değişmesi gereken içeriğinin istemci ve sunucu arasındaki hareketi söz konusudur. Bir başka deyişle gereksiz olan içeriğin sunucuya tekrar gönderilmesinin, işlenmesinin önüne geçilmesi sağlanmaktadır. Bu bir anlamda **son kullanıcı(End User)** için daha zengin etkileşime sahip ve performanslı bir web ortamı oluşturulması anlamına da gelir.

AJAX mimarisi, **Asp.Net AJAX** ile .NET platformu üzerinde çok daha kolay bir şekilde uygulanabilir hale getirilmiştir. Microsoft' un .Net Framework mimarisine getirdiği bu ilave yenilik hakkında söylenecek ve yazılacak çok şey vardır. Bu yazıdaki hedef ise **Windows Communication Foundation** servislerinin(WCF Services) AJAX bazlı istemcilere hizmete verebilecek şekilde nasıl geliştirileceklerinin öğrenilmesidir. Nitekim Web uygulamalarıda kendi içlerinde WCF servislerine erişebilir ve kullanabilirler. Bununla birlikte WCF mimarisi AJAX tipindeki istemcilere **JSON(JavaScript and Object Notation)** formatında veri içeriği sunabilme kapasitesine de sahip hale gelmiştir. İşe ilk olarak Asp.Net AJAX tipindeki istemcileri ele alarak başlamakta yarar vardır.

Asp.Net AJAX modeli aslında iki önemli parçadan oluşur. Bu parçalardan birisi **istemci betik kütüphanelerdir(Client Script Libraries)**. Diğer parça ise **sunucu taraflı betik kontrollerdir(Server Side Script Controls)**. Asp.Net AJAX sayfalarından bir WCF servisine ulaşmak son derece kolaydır. Bunun için öncelikli olarak servisin adres bilgisinin **ScriptManager** bloğu içerisinde belirtilmesi gerekir. Bu işlemden sonra istemci tarafından sanki bir **JavaScript** fonksiyonu çağırılıyormuş gibi WCF **servisine ait operasyonlar(Service Operations)** kullanılabilir. Elbette servis tarafından sunulan **EndPoint** noktasının AJAX tipinden istemcilere hizmet verecek şekilde ayarlanmış olması gerekir.

NOT : *AJAX istemcileri için WCF servisleri geliştirmek eğer* ***Visual Studio 2008*** *gibi bir geliştirme ortamı kullanılıyorsa çok kolaydır. Nitekim bir Web uygulamasına* ***Add New Item*** *ile****Ajax Enabled WCF Service*** *şablonun eklenmesi yeterlidir.*

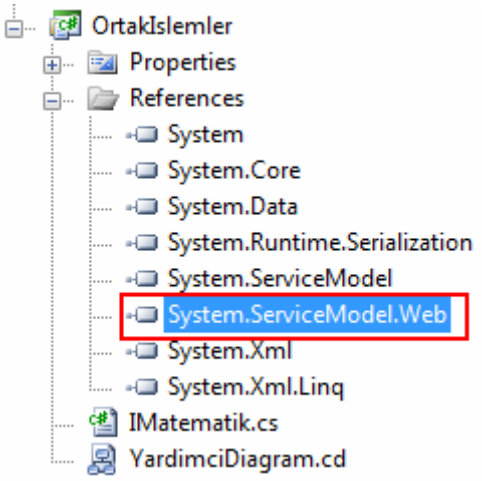


Ajax-enabled WCF Service seçeneğini kullanmadanda AJAX uyumlu WCF servisleri geliştirilebilir. Bunu yapmanın temel olarak iki farklı yolu vardır. Buna göre servisin kod tarafında yada konfigurasyon bazlı olacak şekilde geliştirilmesi mümkündür. Kod tarafında yapılan geliştirme çoğunlukla **Dinamik Host Aktivasyon(Dynamic Host Activation)** olarak bilinmektedir. Hangi model seçilirse seçilsin, AJAX istemcilere destek verecek **EndPoint** noktasına sahip olan WCF servisinin, **IIS(Internet Information Services)** üzerinde **Host** ediliyor olması şarttır. Dynamic Host Activation modeline göre gelen talepleri değerlendirmek üzere **WebServiceScriptHostFactory** isimli bir **CLR tipi (Common Language Runtime Type)** devreye girmektedir. Bilindiği üzere IIS üzerinden yayınlanan WCF servislerinde **svc** uzantılı bir dosya bulunmaktadır. Söz konusu dosyada yer alan **Service direktifi(Directive)** ne ait **Factory niteliğinde(Attribute)**

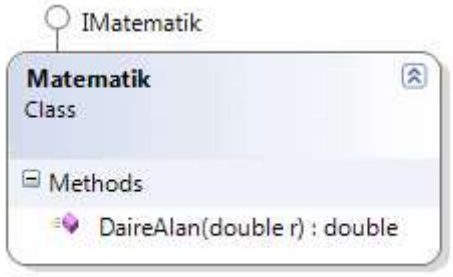
WebScriptServiceHostFactoryataması yapılarak, çalışma zamanında gelen **taleplerin(Requests)** dinamik olarak üretilen bir **EndPoint** tarafından karşılanması sağlanmaktadır. Bir başka deyişle AJAX istemcilerin gönderecekleri talepere göre EndPoint dinamik olarak uygun bir şekilde **WebScriptServiceHostFactory** tipi tarafından üretilir.

NOT : *WebScriptServiceFactory sınıfı, çalışma zamanında herhangibir konfigurasyon bilgisine ihtiyaç duymadan servise,* ***Asp.Net Ajax EndPoint*** *noktası eklenmesini sağlar. Bu sınıfın üretimi hem* ***IIS(Internet Information Services)*** *hemde* ***(WAS)Windows Process Activation Services*** *ortamları tarafından desteklenmektedir. WebScriptServiceFactory tipi,****ServiceHostFactoryBase abstract*** *sınıfından türeyen* ***ServiceHostFactory*** *sınıfından kalıtılmıştır.*

Konuyu daha net kavrayabilmek adına basit bir örnek ile devam edilmesinde yarar vardır. örnekte, IIS üzerinden Ajax istemcileri için **Dynamic Host Activation** mantığına uygun olacak şekilde bir servis geliştirilmektedir. Her zamanki gibi işe **servis kütüphanesinin(WCF Service Library)** tasarlanmasıyla başlanılmasında yarar vardır. Servis kütüphanesi içerisinde yer alacak olan tipler dışında Web üzerinde AJAX istemcilere hizmet verileceği için ilgili operasyonların **WebGet** yada **WebInvoke nitelikleri(attribute)** ile imzalanması gerekmektedir. Bu nedenle WCF servis kütüphanesinin **System.ServiceModel.Web.dll** isimli assembly referansına sahip olması gerekmektedir.

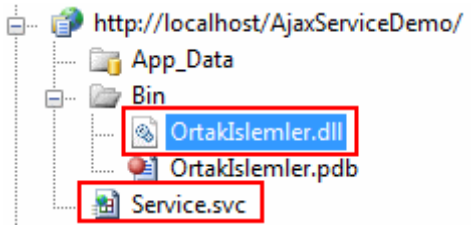


Servis tarafı şimdilik geriye ilkel tip döndüren tek bir metoda sahiptir.




```
using System;
using System.ServiceModel;
using System.ServiceModel.Web;

namespace OrtakIslemler
{
   [ServiceContract(Namespace="OrtakServis")]
   public interface IMatematik
   {
      [OperationContract]
      [WebGet]
      double DaireAlan(double r);
   }

   public class Matematik : IMatematik
   {
      #region IMatematik Members

      public double DaireAlan(double r)
      {
         return Math.PI * r * r;
      }

      #endregion
   }
}
```

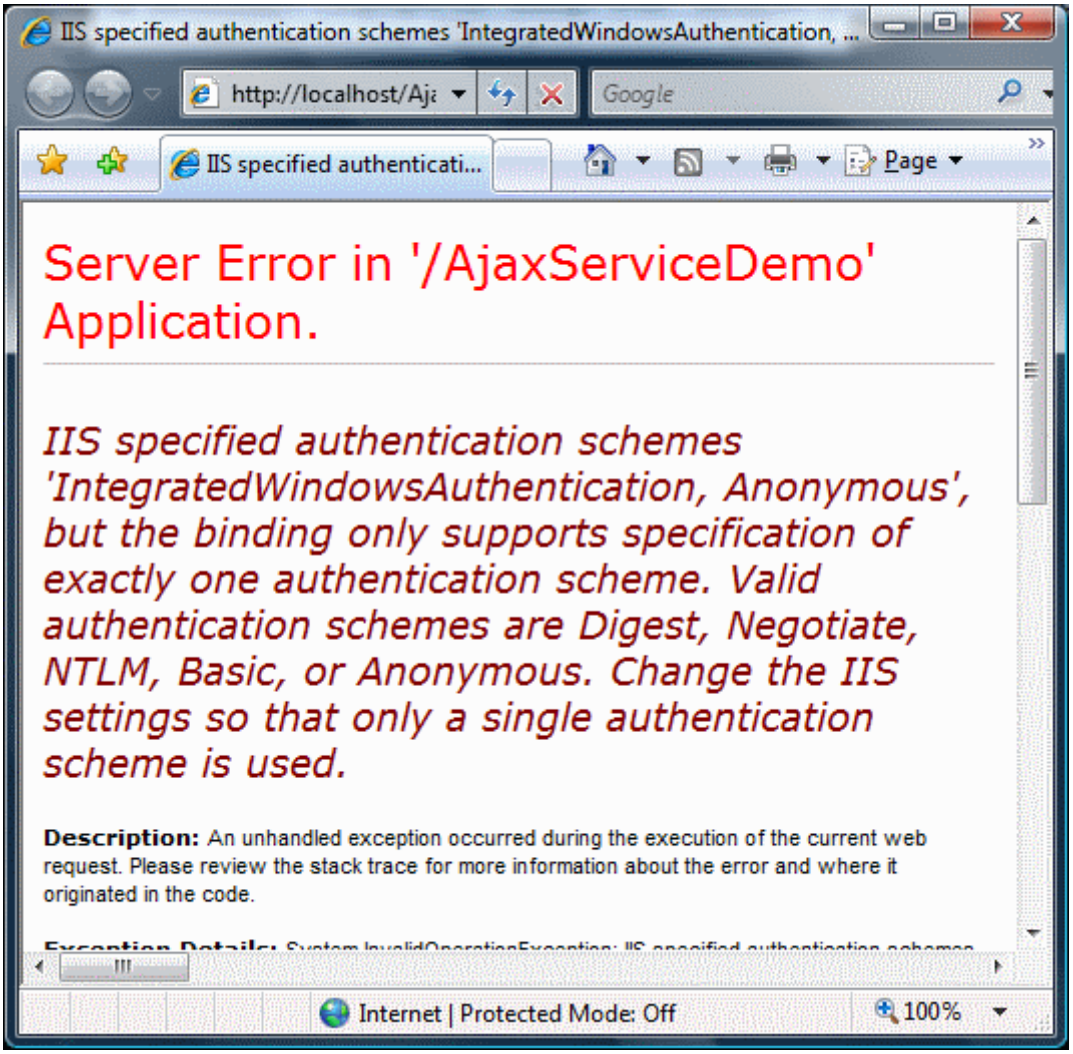
IMatematik isimli **servis sözleşme arayüzü(Interface)** geriye **double** değer döndüren ve **double** tipinden parametre alan **DaireAlan** isimli bir operasyon tanımlamaktadır. İlgili operasyonun**WebGet** niteliği ile imzalandığına dikkat edilmelidir. Servis sözleşmesini uygulayan **Matematik** isimli sınıf ise **DaireAlan** metodunu uygulamaktadır. Servis uygulaması **IIS** üzerinden yayınlama yapması gerektiğinden **Visual Studio 2008** ortamında **Add New Web Site** ile **WCF Service** şablonundan bir proje oluşturulabilir. Söz konusu uygulamada şu an için **web.config** dosyasının olmasına gerek yoktur. Nitekim ilk amaç **Dynamic Host Activation** modelini uygulamaktır. Diğer taraftan servis kütüphanesinin ilgili web uygulamasına referans edilmesi gerekmektedir. Sonuç itibariyle AjaxServiceDemo isimli WCF Service uygulamasının Solution Explorer üzerinden görülen ilk hali aşağıdaki gibi olmalıdır.



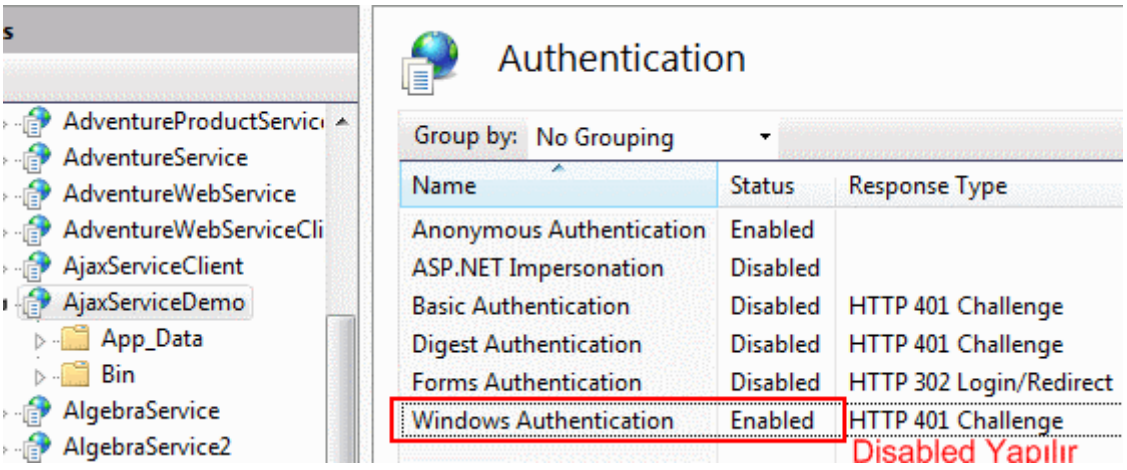
Artık Service.svc dosyasının içeriği aşağıdaki gibi geliştirilebilir.

<%@ ServiceHost Language="C#" Debug="true" **Service="OrtakIslemler.Matematik" Factory=System.ServiceModel.Activation.WebScriptServiceHostFactory** %>

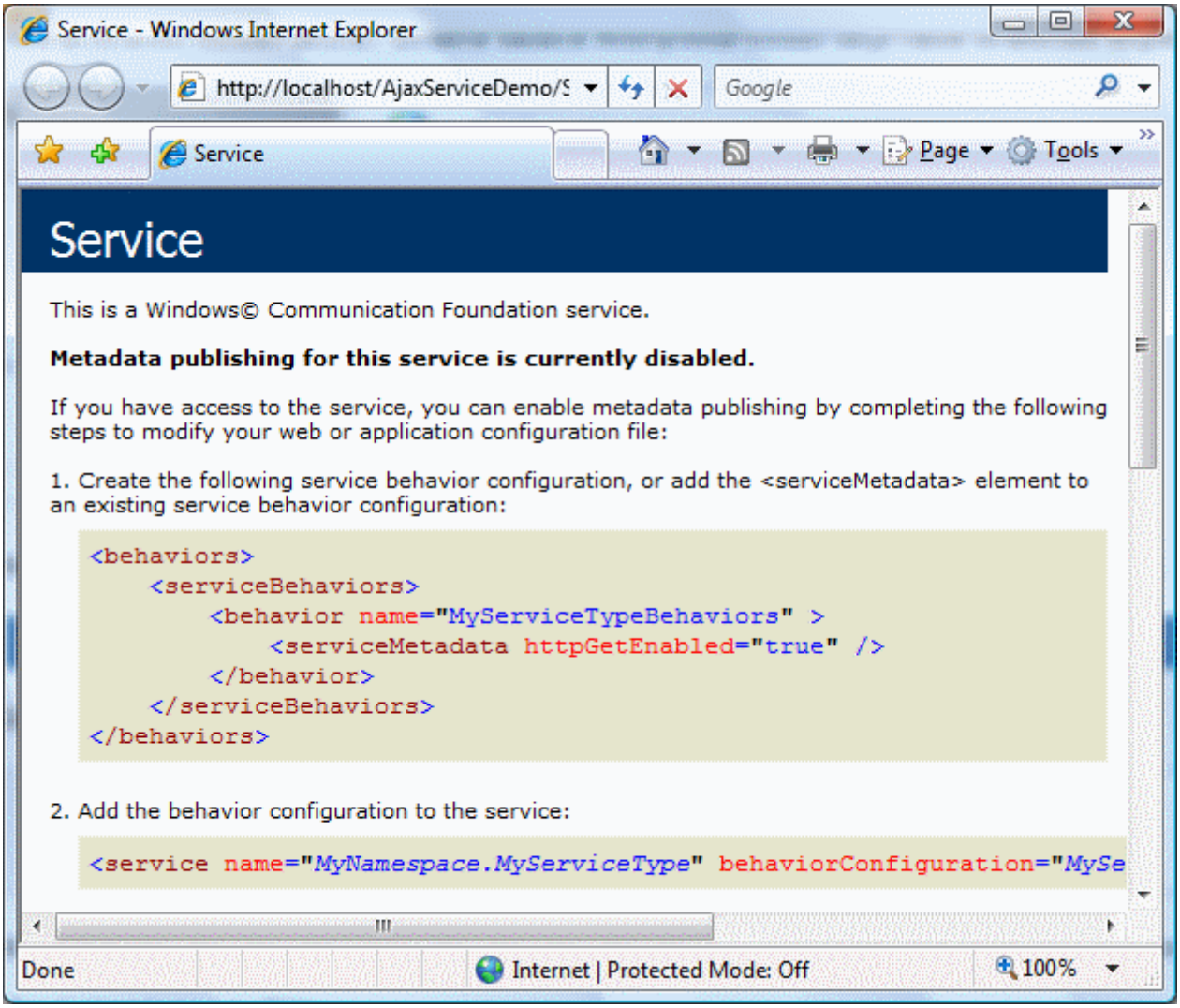
Service niteliğinde standart olarak servis nesnesine ait tipin bildirimi yapılmaktadır. Bununla birlikte Ajax istemcilerine hizmet verilmesini sağlayabilmek içinde **Factory** niteliğine**System.ServiceModel.Activation.WebScriptServiceHostFactory** bilgisi atanmıştır. Bu noktadan sonra **svc** dosyası herhangibir tarayıcı penceresinden talep edilebilir. Yanlız dikkat edilemesi gereken bir nokta vardır. Yazının hazırlandığın tarih itibariyle Vista işletim sistemi üzerinde yer alan **IIS 7.0** sürümünde aşağıdaki ekran görüntüsünde yer alan hata mesajı ile karşılaşılmaktadır.



Bu sorunun şimdilik çözümü için IIS 7.0 üzerinde ilgili **WCF** servisine ait **Authentication** bilgisinden sadece tek bir modelin seçili olmasını sağlamaktır. Nitekim ilgili WCF servisi için hem**Anonymous** hemde **Windows Authentication** modları aktiftir. örnekte sadece Anonymous modun seçil halde bırakılması sağlanılmalıdır.



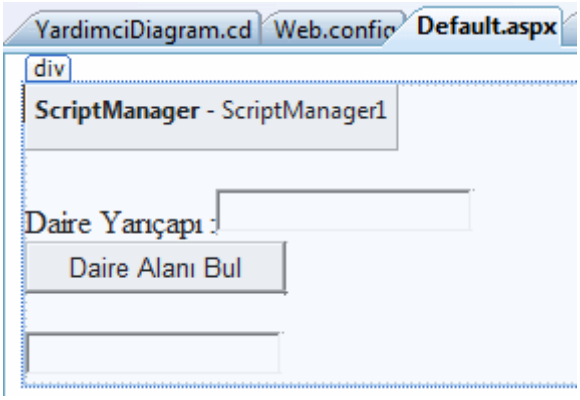
Bu düzeltmeden sonra service.svc yeniden talep edilirse aşağıdaki ekran görüntüsü elde edilir.



Görüldüğü gibi standart olarak bir **WCF** servisine **HTTP** üzerinden yapılan talep sonrası karşılaşılan ekran üretilmiştir. Elbette burada herhangibir şekilde konfigurasyon ayarı yapılmadığından yada ilgili nitelikler ile servise veya EndPoint noktasına bir **davranış(Behavior)** belirtilmediğinden HTTP üzerinden metadata bilgisi çekilmesine izin verilmemektedir.

NOT : *İster* ***Dynamic Host Activation*** *modeli ister konfigurasyon bazlı modele göre AJAX istemcilere hizmet verecek şekilde tasarlanmış olsun, bir WCF servisi ek* ***EndPoint*** *noktaları içerebilir. Söz gelimi hem AJAX istemcilere hizmet veren hemde* ***SOAP*** *protokolü üzerinden hizmet veren* ***EndPoint*** *noktalarına sahip bir WCF servisinin tasarlanması mümkündür.*

Artık AJAX uyumlu istemcinin yazılmasına başlanabilir. Söz konusu istemci **Visual Studio 2008** ortamında geliştirilen bir **Asp.Net** uygulamasıdır. Diğer taraftan **.Net Framework 3.5** şablonu seçileceği için doğrudan AJAX desteğide otomatik olarak gelmektedir. Sayfanın en önemli olan noktalarından birisi **ScriptManager** içeriğidir. Nitekim ScriptManager elementi içerisinde kullanılmak istenen servisin adresi mutlaka belirtilmelidir. örnek aspx sayfasının içeriği aşağıdaki gibidir.

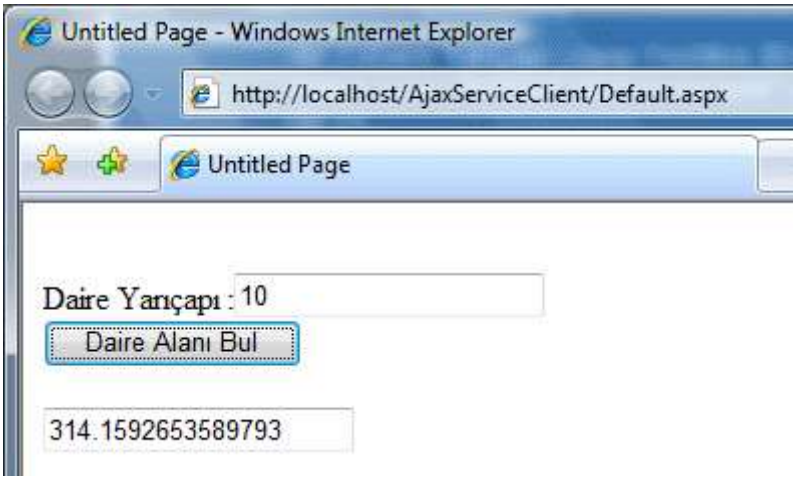



```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script type="text/javascript">

   function ServisCagir()
   {
      var proxy = new OrtakServis.IMatematik();
      var r=parseFloat(document.getElementById("txtYaricap").value);
      proxy.DaireAlan(r,Basarili,Basarisiz,null);
   }

   function Basarili(sonuc)
   {
      document.getElementById("lblSonuc").value=sonuc;
   }
   function Basarisiz()
   {
      document.getElementById("lblSonuc").value="Bir sorun var.";
   }
</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
   <form id="form1" runat="server">
      <div>

         <asp:ScriptManager ID="ScriptManager1" runat="server">
```

```
            <Services>
               <asp:ServiceReference
Path="http://localhost/AjaxServiceDemo/Service.svc" />
            </Services>
         </asp:ScriptManager>

         <br />Daire Yarıçapı :<asp:TextBox ID="txtYaricap"
runat="server"></asp:TextBox><br />
         <input type="button" id="btnHesapla" value="Daire Alanı
Bul" onclick="ServisCagir()" /><br />
         <br />
         <input type="text" id="lblSonuc" />
      </div>
   </form>
</body>
</html>
```

ScriptManager kontrolü kendi içerisinde birden fazla servis noktası tanımlanabilmesini sağlayacak şekilde **Services** isimli bir alt elemente sahiptir. Bu element içerisinde **ServiceReference** isimli bileşenler kullanılır. ServiceReference bileşeninin **Path** özelliğine verilen değer kullanılmak istenen WCF servisine ait **URL** bilgisini taşımaktadır. Diğer taraftan **javascript** kodlarında dikkat edileceği üzere **btnHesapla** isimli **input** kontrolüne basıldığında çalışan **ServisCagir** isimli bir metod yer almaktadır. Bu metod içerisinde önce bir proxy nesnesi oluşturulmaktadır. Bu nesne oluşturulurken **OrtakServis.IMatematik** isimli tipten yararlanılmaktadır. **OrtakServis** adı, **ServiceContract** niteliğinde belirtilen **Namespace** değerinden gelmektedir. Diğer taraftan IMatematik ismi ise, **servis sözleşmesinin(Service Contract)** adıdır. Buna göre üretilen **proxy** nesnesi üzerinden **DaireAlan** metodu çağırılarbilir.

Metod çağrısı gerçekleştirilirken ilk parametre olarak, **DaireAlan** metodunun beklediği **yarıçap** değeri verilmektedir. İkinci parametre söz konusu operasyon başarılı bir şekilde tamamlanırsa devreye girecek olan ve sonuçların alınabileceği geri bildirim fonksiyonunun adıdır. Bu metod içerisinde çağırılan servis operasyonunun sonucu alınması gerektiğinden ilgili metod, DaireAlan operasyonunun dönüş değerini parametre olarak almaktadır. üçüncü parametre ile belirtilen metod ise, servis üzerindeki operasyonun çağırılması esnasında bir hata oluşması halinde devreye girecek olan metoddur. Son parametre **null** olarak geçilmekle birlikte çoğunlukla **HttpContext** tipinin kullanılması gerektiği durumlarda ele alınmaktadır. Artık istemci uygulama çalıştırılıp test edilebilir. Eğer düğme tıklanırsa sayfanın tamamının **Post** edilmeden, sadece servis operasyonun çağırıldığı ve sonucun ekrandaki ilgili**TextBox** kontrolüne alındığı görülebilir.



Şimdi **WCF** servisinin **Dynamic Host Activation** yerine konfigurasyon bazlı olarak nasıl inşa edileceğine bakılabilir. Konfigurasyon dosyası kullanıldığında **WebHttpBinding bağlayıcı tipinin(Binding Type)** ve **EnableWebScript davranışının(Behavior)** kullanılması gerekir. Söz gelimi az önce geliştirilen **WCF servis** uygulamasının konfigurasyon tabanlı olacak şekilde çalıştırılması için **web.config** dosyasında yer alan **system.serviceModel** elementinin içeriğinin aşağıdaki gibi tasarlanması yeterlidir.

```
<system.serviceModel>
   <behaviors>
      <endpointBehaviors>
         <behavior name="AjaxEndPointBehavior">
            <enableWebScript />
         </behavior>
      </endpointBehaviors>
   </behaviors>
   <services>
      <service name="OrtakIslemler.Matematik">
         <endpoint address="http://localhost/AjaxServiceDemo/Service.svc" behaviorC
onfiguration="AjaxEndPointBehavior" binding="webHttpBinding"
name="AjaxEndPoint"contract="OrtakIslemler.IMatematik" />
      </service>
   </services>
</system.serviceModel>
```

Dikkat edileceği üzere **AjaxEndPoint** isimli **EndPoint** tanımlanırken **webHttpBinding bağlayıcı tipinin(Binding Type)** kullanılacağı belirtilmiştir. Ayrıca **EndPoint** davranışı içerisinde **enableWebScript** elementi kullanılmıştır. Bu durumda istemci uygulama test edildiğinde yine WCF servisinin başarılı bir şekilde çalıştığı görülebilir.

AJAX uyumlu WCF servislerinde önem arz eden konulardan biriside, istemcilerin servis operasyonlarına yaptıkları talep sonrası dönecek olan verinin formatıdır. AJAX destekli

WCF servisleri**XML(eXtensible Markup Language)** tipinde veri formatını kullanmakla birlikte **.Net Framework 3.5** ile **JSON(JavaScript Object Notation)** desteğinede sahip olmuşlardır ki çoğu AJAX servisi varsayılan olarak **JSON** bazlı yayınlama yapmaktadır. Daha önceki bölümlerden de hatırlanacağı gibi, bir servis operasyonuna uygulanan **WebGet** ve **WebInvoke niteliklerine(attribute)** ait**ResponseFormat** özellikler inin değerleri kullanılarak **JSON** formatında veri dönüştürüleceği belirtilebilmektedir. Diğer taraftan **WCF AJAX EndPoint** noktaları hem **JSON** hemde **XML**formatındaki taleplere cevap verebilmektedir. **application/json** tipindeki **talepler(requests)** tahmin edileceği üzere **JSON** formatı ile alakalıdır. Bununla birlikte **text/xml** formatındaki taleplerde **XML**formatı ile alakalıdır. Tabi Ajax destekli servislerde sorun oluşturan önemli noktalardan biriside, .NET tiplerinin JSON formatına **serileştirme(Serialization)** işlemleridir. Bu nedenle ilk olarak JSON serileştirmesine bir göz atmakta yarar vardır.

NOT : JSON (JavaScript Object Notation) *özellikle* ***AJAX uyumlu servisler(Ajax Enabled WCF Services/Web Services)*** *ile istemciler arasında hızlı veri değiş tokuşu yapılmasına olanak tanıyan önemli* ***veri formatı(Data Format)*** *standartlarındadır.*

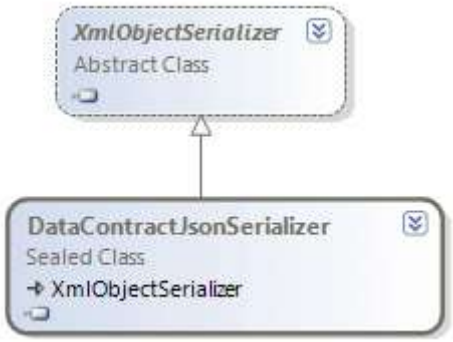
JSON serileştirmede de servis tarafından istemciye yayınlanan veri tiplerinin önemi büyüktür. Nitekim **WCF** gibi .Net tabanlı bir ortamda, **CLR** tiplerinin **JSON** karşılıklarının bilinmesinde yarar vardır. Bu amaçla öncelikli olarak aşağıdaki tablonun göz önüne alınması yararlı olabilir. Hemen hemen tüm **CLR tipleri(Common Language Runtime Types)** uygun JSON tiplerine dönüştürülmektedir.

.Net Tipi	**JSON Karşılığı**
Int16, Int32, Double, Decimal gibi sayısal tiplerin tamamı.	Number
Boolean	Boolean
String, Char	String
Timespan, Guid, Uri	String
XmlElement, XmlNode gibi Xml tipleri	String
ISerializable, DataSet gibi tipler	String
Enum	Number
Byte Dizisi	Number Dizisi
DateTime	DateTime veya String
Collections(Koleksiyonlar), Dictionary Tipleri ve Arrays (Diziler)	Array

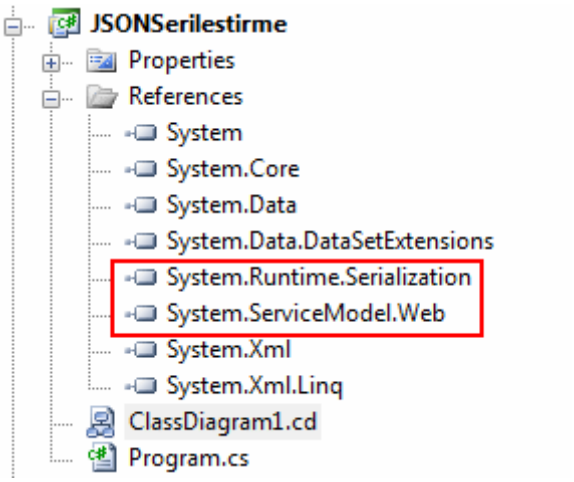
Herhangibir tipin Null değeri	Null
DataContract niteliğini(Attribute) uygulamış tipler	Complex Type
ISerializable arayüzünü(Interface) uygulamış tipler	Complex Type
DBNull	Empty Complex Type
XmlQualifiedName	String

Buradaki tablo **.Net** ve **JSON** tipleri arasındaki eşleştirmelerin basit bir özetidir. Tip dönüşümleri sırasında göz önüne alınması gereken oldukça fazla kural ve vaka bulunmaktadır.*(Söz konusu durumlar makalemizin konusunu aşmaktadır. Ancak detaylı bilgi için **MSDN** kaynaklarındaki ilgili bağlantıya bakılabilir.)*

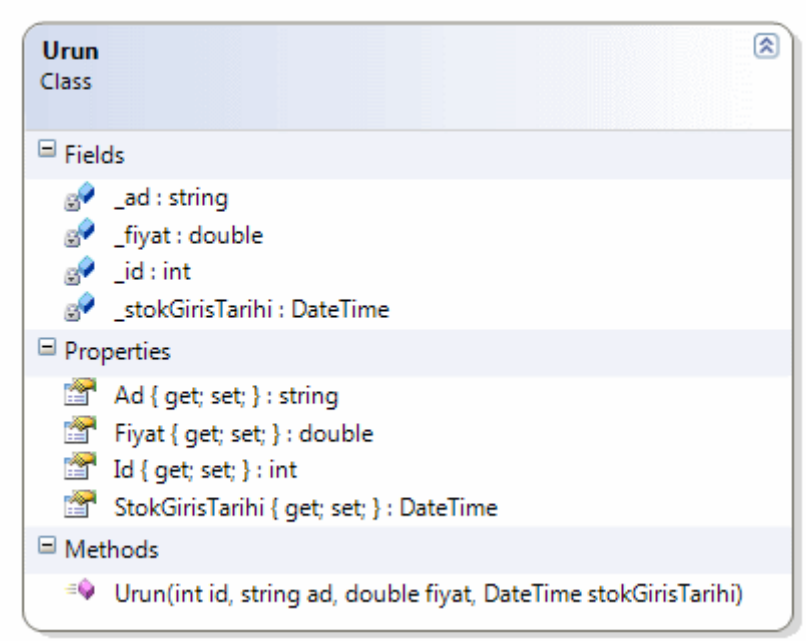
Normal şartlarda WCF servisleri **JSON** serileştirmesini otomatik olarak gerçekleştirmektedir. özellikle kullanıcı **tanımlı tiplerde (User Defined Types)** veri sözleşmesi tanımlanarak**(DataContract** ve **DataMember nitelikleri-attribute** yardımıyla) serileştirme işlemi otomatik hale getirilmektedir. Yinede bazı durumlarda **JSON serileştirilme(Serialization)** ve **ters-Serileştirme(DeSerialization)** işlemlerinin elle yapılması gerekebilir. Bu amaçla **.Net Framework 3.5** ile birlikte gelen **DataContractJsonSerializer** tipinden yararlanılabilir.



System.ServiceModel.Web.dll assembly' ındaki **System.Runtime.Serializetion.Json isim alanında(Namespace)** bulunan **DataContractJsonSerializer**, **XmlObjectSerializer abstr act** sınıfından türemiş **selaed** olarak imzalanmış bir **CLR** tipidir. Sealed olarak işaretlenmiş olması nedeniyle kendisinden **türetme(Inherit)** yapılamamaktadır. Temel görevi tipleri(Types) **JSON** formatında veri olarak serileştirmek veya tam tersini yapmaktır. JSON serileştirmeyi daha iyi kavramak için bir örnek ile devam edilmesinde yarar vardır. Bu amaçla **Visual Studio 2008** ortamında **.Net Framework 3.5** tabanlı bir **Console** uygulaması geliştirildiği göz önüne alınsın. öncelikli olarak **System.ServiceModel.Web.dll** ' inin projeye refarans edilmiş olması gerekmektedir. Ayrıca**DataContract** ve **DataMember** niteliklerini kullanabilmek içinde **System.Runtime.Serialization.dll** assembly' ının referans edilmesi şarttır.



örnek olarak **class diagram** görüntüsü aşağıdaki gibi olan Urun isimli bir sınıf(Class) tasarlanabilir.




```
using System;
using System.Runtime.Serialization;

namespace JSONSerilestirme
{
   [DataContract]
   class Urun
   {
      private int _id;
      private string _ad;
      private double _fiyat;
      private DateTime _stokGirisTarihi;
```

```
        [DataMember]
        public DateTime StokGirisTarihi
        {
           get { return _stokGirisTarihi; }
           set { _stokGirisTarihi = value; }
        }

        [DataMember]
        public double Fiyat
        {
           get { return _fiyat; }
           set { _fiyat = value; }
        }

        [DataMember]
        public string Ad
        {
           get { return _ad; }
           set { _ad = value; }
        }

        [DataMember]
        public int Id
        {
           get { return _id; }
           set { _id = value; }
        }

        public Urun(int id, string ad, double fiyat, DateTime stokGirisTarihi)
        {
           Id = id;
           Ad = ad;
           Fiyat = fiyat;
           StokGirisTarihi = stokGirisTarihi;
        }
     }
}
```

Urun sınıfı **DataContract niteliği(Attribute)** ile imzalanmıştır. Bununla birlikte serileştirmeye tabi tutulacak olan Id,Ad,Fiyat,StokGirisTarihi **özellikleride(Properties) DataMember** nitelikleri ile işaretlenmişlerdir. Serileştirme ve ters-Serileştirme için örnek olarak aşağıdaki gibi bir kod parçası göz önüne alınabilir.

```
using System;
using System.Runtime.Serialization.Json;
```

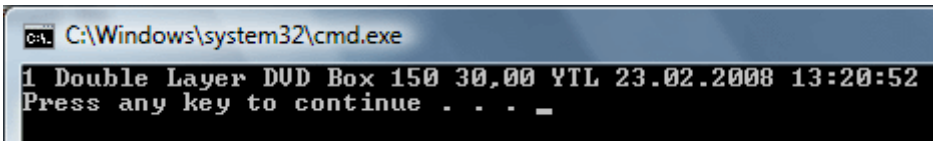
```
using System.IO;

namespace JSONSerilestirme
{
   class Program
   {
      static void Main(string[] args)
      {
         // Urun sınıfına ait bir nesne örneklenir.
         Urun dvd = new Urun(1, "Double Layer DVD Box 150", 30, DateTime.Now);

         #region Kullanıcı tanımlı bir tipi JSON Serileştirme

         // Json serileştirme işlemleri için DataContractSerializer tipi örneklenir. örnekleme işlemi sırasında parametre olarak serileştirilecek olan tip belirtilir.
         DataContractJsonSerializer serializer = new DataContractJsonSerializer(typeof(Urun));
         // Serileştirme örnek olarak bir dosyaya doğru yapılacaktır.
         using (FileStream stream = new FileStream("UrunJason.xml", FileMode.Create, FileAccess.Write))
         {
            // WriteObject metodu ile ilk parametrede belirtilen stream üzerine, ikinci parametrede belirtilen nesne örneğinin verisi serileştirilir
            serializer.WriteObject(stream, dvd);
         }

         #endregion

         #region JSON datasını DeSerialize edip Object haline getirme

         // Dosyadaki veriden nesne elde edileceği için FileStream oluşturulur.
         using (FileStream stream = new FileStream("UrunJason.xml", FileMode.Open, FileAccess.Read))
         {
            // ReadObject metodu ile stream ile belirtilen dosya içerisindeki JSON formatlı veri okunur Object olarak elde edilir. Sonrasında ise kullanılabilmesi için Urun tipine cast edilir.
            Urun okunanDvd = (Urun)serializer.ReadObject(stream);
            Console.WriteLine(okunanDvd.Id+" "+okunanDvd.Ad+" "+okunanDvd.Fiyat.ToString("C2")+" "+okunanDvd.StokGirisTarihi.ToString());
         }

         #endregion
      }
```

```
    }
}
```

Serileştirme işlemi sonrasında oluşan UrunJason.xml dosyasının içeriği aşağıdaki gibi olmaktadır.

```
{"Ad":"Double Layer DVD Box
150","Fiyat":30,"Id":1,"StokGirisTarihi":"\/Date(1203765080248+0200)\/"}
```

Ayrıca ters serileştirme(DeSerializetion) işlemi sonrasında ise programın ekran çıktısı aşağıdaki gibidir. Görüldüğü gibi dosyada duran **JSON** formatlı veri içeriğinden Urun nesne örneği elde edilmiştir.



Elbette Urun tipinden nesne örneklerini bünyesinde barındıran bir dizide JSON formatında serileştirilebilir ve hatta okunabilir. Aşağıdaki kod parçasında bu durum örneklenmeye çalışılmaktadır.

```
Urun[] urunler ={
                    new Urun(2,"Urun X",1.45,new DateTime(2007,12,2))
                    ,new Urun(3,"Urun Y",2.34,new DateTime(2008,2,3))
                    ,new Urun(4,"Z Urun",34.56,new DateTime(2006,6,7))
                    };
```

```
DataContractJsonSerializer arraySerializer = new
DataContractJsonSerializer(typeof(Urun[]));
using (FileStream stream = new FileStream("Urunler.json", FileMode.Create,
FileAccess.Write))
{
   arraySerializer.WriteObject(stream, urunler);
}

using (FileStream stream = new FileStream("Urunler.json", FileMode.Open,
FileAccess.Read))
{
   Urun[] gelenUrunler=(Urun[])arraySerializer.ReadObject(stream);
   foreach(Urun urn in gelenUrunler)
      Console.WriteLine(urn.Ad);
}
```

Bu kez serileştirme işleminde Urun tipinden bir **dizi(Array)** söz konusudur. Bu sebepten **DataContractJsonSerializer** sınıfına ait nesne örneklenirken parametre

olarak **typeof(Urun[])** bilgisi verilmektedir. Sonuç olarak üretilen **JSON** formatlı veri içeriği aşağıdaki gibi olacaktır.

[{"Ad":"Urun X","Fiyat":1.45,"Id":2,"StokGirisTarihi":"\/Date(1196546400000+0200)\/"},{"Ad":"Urun Y","Fiyat":2.34,"Id":3,"StokGirisTarihi":"\/Date(1201989600000+0200)\/"},{"Ad":"Z Urun","Fiyat":34.56,"Id":4,"StokGirisTarihi":"\/Date(1149627600000+0300)\/"}]

Hemen yeni bir kod parçası ile bir **DataSet** nesne örneğinin **JSON** formatında nasıl serileşeceğine bakmakta yarar vardır. Nitekim veri tabanı uygulamalarının çok sık kullanılması nedeni ile WCF servislerinden özellikle **bağlantısız katmana(Disconnected Layer)** ait tiplerin döndürülmesi sık rastlanan bir durumdur. Bu amaçla Console uygulamasına aşağıdaki kod parçasının eklenmesi yeterlidir.

```
using (SqlConnection conn = new SqlConnection("data source=.;database=AdventureWorks;integrated security=SSPI"))
{
   SqlDataAdapter adapter = new SqlDataAdapter("Select Top 5 ProductId,Name,ListPrice From Production.Product", conn);
   DataSet set = new DataSet();
   adapter.Fill(set);

   DataContractJsonSerializer dataSetSerializer = new DataContractJsonSerializer(typeof(DataSet));
   using (FileStream stream = new FileStream("Products.json", FileMode.Create, FileAccess.Write))
   {
      dataSetSerializer.WriteObject(stream, set);
   }
}
```

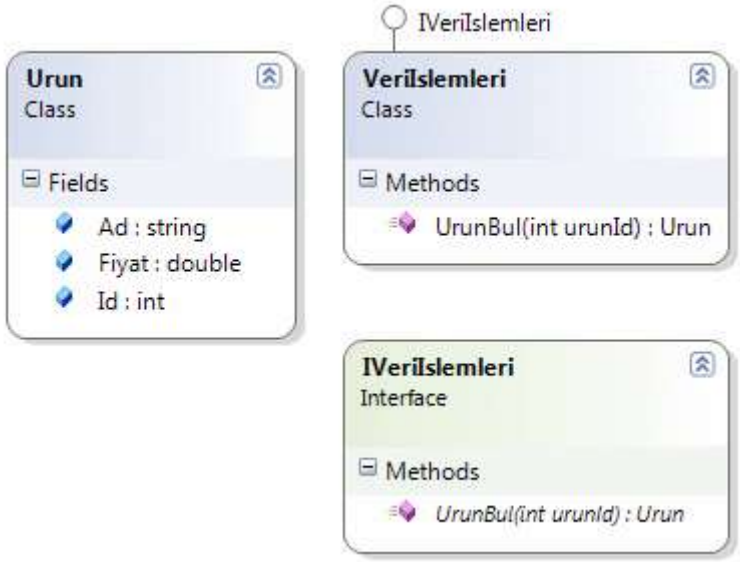
Bu kodun çalışma sonrasın oluşan Products.json dosyasının içeriği aşağıdaki gibi olur.

"<DataSet><xs:schema id=\"NewDataSet\" xmlns:xs=\"http:\/\/www.w3.org\/2001\/XMLSchema\" xmlns:msdata=\"urn:schemas-microsoft-com:xml-msdata\"> <xs:element name=\"NewDataSet\" msdata:IsDataSet=\"true\" msdata:UseCurrentLocale=\"true\"> <xs:complexType> <xs:choice minOccurs=\"0\" maxOccurs=\"unbounded\"><xs:element name=\"Table\"> <xs:complexType><xs:sequence><xs:element name=\"ProductId\" type=\"xs:int\" minOccurs=\"0\"\/> <xs:element name=\"Name\" type=\"xs:string\" minOccurs=\"0\"\/><xs:element name=\"ListPrice\" type=\"xs:decimal\" minOccurs=\"0\"\/><\/xs:sequence> <\/xs:complexType><\/xs:element><\/xs:choice> <\/xs:complexType> <\/xs:element><\/xs:schema><diffgr:diffgram xmlns:diffgr=\"urn:schemas-microsoft-com:xml-diffgram-v1\" xmlns:msdata=\"urn:schemas-microsoft-com:xml-msdata\"><NewDataSet> <Table

diffgr:id=\"Table1\" msdata:rowOrder=\"0\"><ProductId>1<\/ProductId> <Name>Adjustable Race<\/Name><ListPrice>0.0000<\/ListPrice><\/Table><Table diffgr:id=\"Table2\" msdata:rowOrder=\"1\"><ProductId>2<\/ProductId> <Name>Bearing Ball<\/Name><ListPrice>0.0000<\/ListPrice><\/Table><Table diffgr:id=\"Table3\" msdata:rowOrder=\"2\"><ProductId>3<\/ProductId><Name>BB Ball Bearing<\/Name><ListPrice>0.0000<\/ListPrice><\/Table><Table diffgr:id=\"Table4\" msdata:rowOrder=\"3\"><ProductId>4<\/ProductId><Name>Headset Ball Bearings<\/Name><ListPrice>0.0000<\/ListPrice><\/Table><Table diffgr:id=\"Table5\" msdata:rowOrder=\"4\"><ProductId>316<\/ProductId><Name>Blade<\/Name> <ListPrice>0.0000<\/ListPrice><\/Table><\/NewDataSet><\/diffgr:diffgram><\/DataSet>"

Daha öncedende belirtildiği gibi **DataSet** tipinin **JSON** karşılığı **String** olarak ifade edilmektedir. Bu nedenle üretilen çıktı çift tırnaklar içerisinde yer almaktadır.

Bu bölümde son olarak **kullanıcı tanımlı bir tipin(User Defined Type) WCF** servisinden **JSON** formatında döndürüldüğü ve **AJAX** uyumlu bir istemci tarafından ele alındığı örnek geliştirilmeye çalışılmaktadır. Burada servis operasyonunun **JSON** formatında **cevap(Response)** vermesi ve **POST** metoduna göre çalışması için **WebInvoke niteliği(attribute)**ile imzalanmış olması gerekmektedir. İstemci tarafında ise gelen cevabın ayrıştırılması ve kullanılması ele alınmaktadır. İlk olarak servis tarafında kullanılacak olan standart WCF servis kütüphanesi geliştirilerek işe başlanabilir. örnek kütüphane içerisinde yer alacak tipler aşağıdaki gibidir.



Servis sözleşmesi(Service Contract), uygulayıcı sınıf ve **veri sözleşmesi(Data Contract)** tiplerin içeriği ise aşağıdaki gibidir.

Urun Sınıf;

using System;
using System.Runtime.Serialization;

```
namespace VeriServisKutuphanesi
{
   [DataContract]
   public class Urun
   {
      [DataMember]
      public int Id;
      [DataMember]
      public string Ad;
      [DataMember]
      public double Fiyat;
   }
}
```

Servis Sözleşmesi ve Uygulayıcı Tip;

```
using System;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Data.SqlClient;

namespace VeriServisKutuphanesi
{
   [ServiceContract(Namespace="AdventureVeriHizmeti")]
   public interface IVeriIslemleri
   {
      [OperationContract]
      [WebInvoke(ResponseFormat = WebMessageFormat.Json)]
      Urun UrunBul(int urunId);
   }

   public class VeriIslemleri
      : IVeriIslemleri
   {
      #region IVeriIslemleri Members

      public Urun UrunBul(int urunId)
      {
         Urun u = null;
         using (SqlConnection conn = new SqlConnection("data
source=.;database=AdventureWorks;integrated security=SSPI"))
         {
            SqlCommand cmd = new SqlCommand("Select ProductID,Name,ListPrice From
Production.Product Where ProductID=@ID", conn);
```

```
            cmd.Parameters.AddWithValue("@ID", urunId);
            conn.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            if (reader.Read())
            {
               u = new Urun()
                           {
                              Id=Convert.ToInt32(reader["ProductID"]),
                              Ad=reader["Name"].ToString(),
                              Fiyat=Convert.ToDouble(reader["ListPrice"])
                           };
            }
         }
         return u;
      }
      #endregion
   }
}
```

UrunBul isimli metoda **WebInvoke** niteliği uygulanmıştır. Dikkat edileceği üzere **ResponseFormat** özelliğine **WebMessageFormat.Json** değeri atanmıştır. Bu atama, metodun çıktısının istemcilere **JSON** formatında gönderileceğini belirtmektedir. **UrunBul** isimli metod parametre olarak aldığı **urunId** değerine göre **Product** tablosundan çektiği satırı baz alarak **Urun**tipinde bir nesne örneği oluşturup döndürmektedir. **Servis** tarafı yine **IIS(Internet Information Services)** üzerinde konuşlandırılmış olarak tasarlanmalıdır. Bu amaçla **WCF Service**şablonunda bir uygulama açılarak devam edilebilir. Söz konusu uygulama çok doğal olarak **VeriServisKutuphanesi.dll**' ini referans etmelidir. Bununla birlikte **Service.svc** dosyasının içeriği aşağıdaki gibi tasarlanabilir.

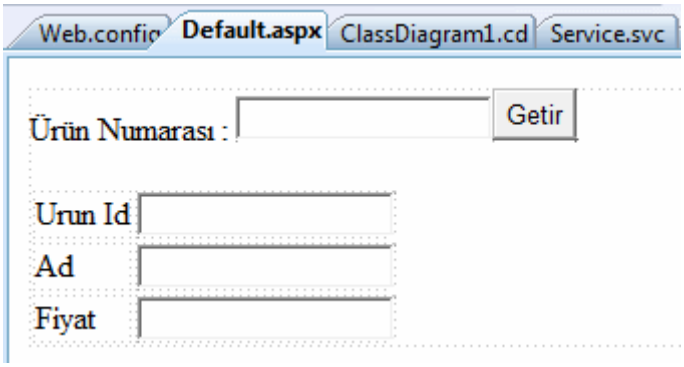
```
<%@ ServiceHost Language="C#"
Debug="true" Service="VeriServisKutuphanesi.VeriIslemleri" %>
```

örnekte **AJAX** uyumlu **EndPoint** noktası **Web.config** dosyası içerisinde tanımlanmaktadır. Web.config dosyasında yer alan **serviceModel** elementinin içeriği aşağıdaki gibidir.

```
<system.serviceModel>
   <behaviors>
      <endpointBehaviors>
         <behavior name="AjaxEndPointBehavior">
            <enableWebScript />
         </behavior>
      </endpointBehaviors>
   </behaviors>
   <services>
```

```
      <service name="VeriServisKutuphanesi.VeriIslemleri">
        <endpoint
address="http://localhost/AjaxServiceDemo2/Service.svc" behaviorConfiguration="A
jaxEndPointBehavior" binding="webHttpBinding" bindingConfiguration=""
name="AjaxEndPoint" contract="VeriServisKutuphanesi.IVeriIslemleri" />
      </service>
    </services>
</system.serviceModel>
```

Gelelim istemci tarafındaki uygulamamıza. Bu sefer ilk örnekten farklı olarak **ScriptManager** kullanmadan **WCF Servis** operasyonunu çağırılmaktadır. Bu amaçla geliştirien Asp.Net Web uygulamasında yer alacak olan aspx sayfasının içeriği aşağıdaki gibi geliştirilebilir.

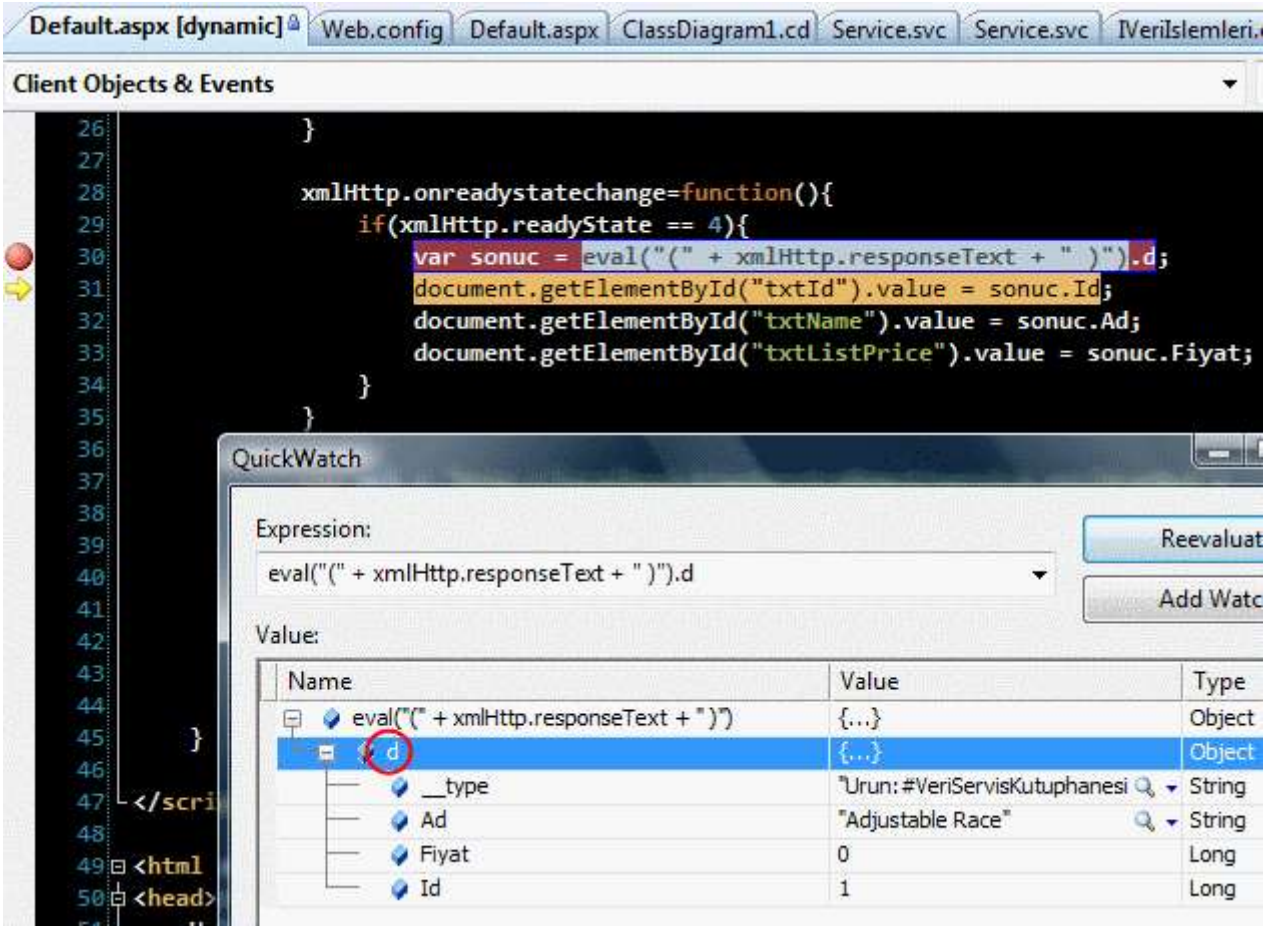



```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script type="text/javascript">

  function Getir()
  {
    var productId = document.getElementById("txtProductId").value;

    if(productId)
    {
      var xmlHttp;
      try
      {
        xmlHttp = new XMLHttpRequest();
      }
      catch (e)
      {
```

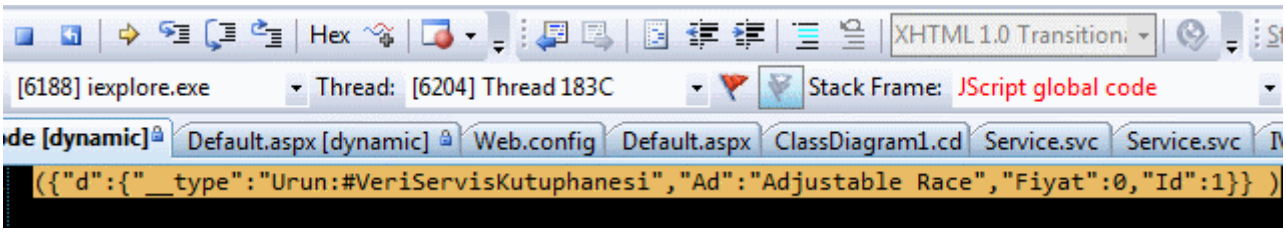
```
            try
            {
               xmlHttp = new ActiveXObject("Msxml2.XMLHTTP");
            }
            catch (e)
            {
               try
               {
                  xmlHttp = new ActiveXObject("Microsoft.XMLHTTP");
               }
               catch (e)
               {
                  alert("Tarayıcıda AJAX desteği yok.");
                  return false;
               }
            }
         }

         xmlHttp.onreadystatechange=function()
            {
               if(xmlHttp.readyState == 4)
               {
                  var sonuc = eval("(" + xmlHttp.responseText + " )").d;
                  document.getElementById("txtId").value = sonuc.Id;
                  document.getElementById("txtName").value = sonuc.Ad;
                  document.getElementById("txtListPrice").value = sonuc.Fiyat;
               }
            }

         var url = "http://localhost/AjaxServiceDemo2/service.svc/UrunBul";

         var mesajGovde = '{"urunId":'+
document.getElementById("txtProductId").value + '}';

         xmlHttp.open("POST", url, true);
         xmlHttp.setRequestHeader("Content-type", "application/json");
         xmlHttp.send(mesajGovde);
      }
   }

</script>

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title>Untitled Page</title>
```

```
</head>
<body>
   <form id="form1" runat="server">
   <div>
      ürün Numarası : <asp:TextBox ID="txtProductId" runat="server" />
      <input type="button" onclick="return Getir();" value="Getir" />
      <br />
      <br />
      <table>
         <tr>
            <td>Urun Id</td>
            <td><input id="txtId" type="text" /></td>
         </tr>
         <tr>
            <td>Ad</td>
            <td><input id="txtName" type="text" /></td>
         </tr>
         <tr>
            <td>Fiyat</td>
            <td><input id="txtListPrice" type="text" /></td>
         </tr>
      </table>
   </div>
   </form>
</body>
</html>
```

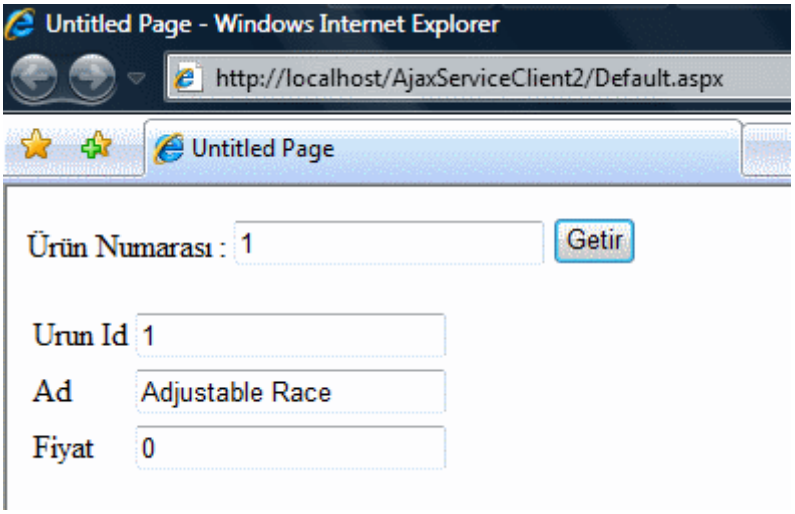
İstemci tarafında klasik olarak **AJAX** uyumlu **Javascript** kodları yer almaktadır. önemli olan noktalardan bir tanesi, **xmlHttp** isimli nesnesinin örneklenmesinden sonra **open** metodu ile **WCF** servisine doğru yapılan çağrıdır. Dikkat edilecek olursa **WebInvoke** niteliği nedeni ile **http://localhost/AjaxServiceDemo2/Service.svc/UrunBul** isimli bir **URL** bilgisi kullanılmaktadır. Diğer taraftan istemciden servis operasyonuna doğru gönderilecek olan **talebin(Request)** başlık kısmının içeriğinin **JSON** olacağı **setRequestHeader** metodu ile belirlenmektedir. Sonrasında **send** metodu ile ilgili paket **WCF** servisine gönderilmektedir. İşlem tamamlandığında devreye giren fonksiyon içerisinde **responseText** özelliğinden de yararlanılarak dönen cevap alınmakta ve sayfa üzerindeki ilgili bileşenlere gelen değerler aktarılmaktadır. Burada **d** isimli özellik yardımıyla aslında Urun tipinin verisine ulaşılabilmektedir. Bunu daha iyi görebilmek için ilgili kodlar debug edilerek**QuickWatch** çıktısına bakılabilir.



Dikkat edilecek olursa dönen cevap içerisinde Urun nesne örneği d isimli bir değişken olarak gelmektedir. Bu değişken üzerinden Ad, Fiyat ve Id isimli alanlarada erişilebilmektedir. **çalışma zamanında(Runtime) F11** ile **step into** modunda hareket edildiğinde istemci tarafına aşağıdaki ekran görüntüsünde yer alan bir içeriğin geldiği görülür.



İşte bu bilgiden yararlanılarak geliştirilen istemcide Getir başlıklı düğmeye basıldığında aşağıdaki ekran görüntüsü elde edilecektir.



Elbette burada gözden kaçırılmaması gereken bazı noktalar vardır. öncelikli olarak veri içeriğin istemciye **Null** gelme olasılığı vardır. Bu da çok doğal olarak çalışma zamanı hatalarının oluşması anlamına gelmektedir. Bu gibi noktalar elbetteki gerçek bir uygulamada mutlaka ele alınmalıdır.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde kısaca **WCF** servis uygulamalarının **AJAX** destekli olacak şekilde nasıl geliştirilebildiklerini incelemeye çalıştık. Burada önemli olan **JSON** veri formatı ile ilişkin olaraktanda **.Net Framework 3.5** ile gelen **DataContractJsonSerializer** tipini inceleme fırsatı bulduk. Son olarakta **JSON** formatında içerik sunan bir **WCF** servisinin**AJAX** tabanlı bir istemci ile nasıl çağırılabileceğini inceledik. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

AjaxDestegi.rar (70,13 kb)

WCF - Web Bazlı Programlama Modeli (2008-02-14T04:29:00)

wcf,

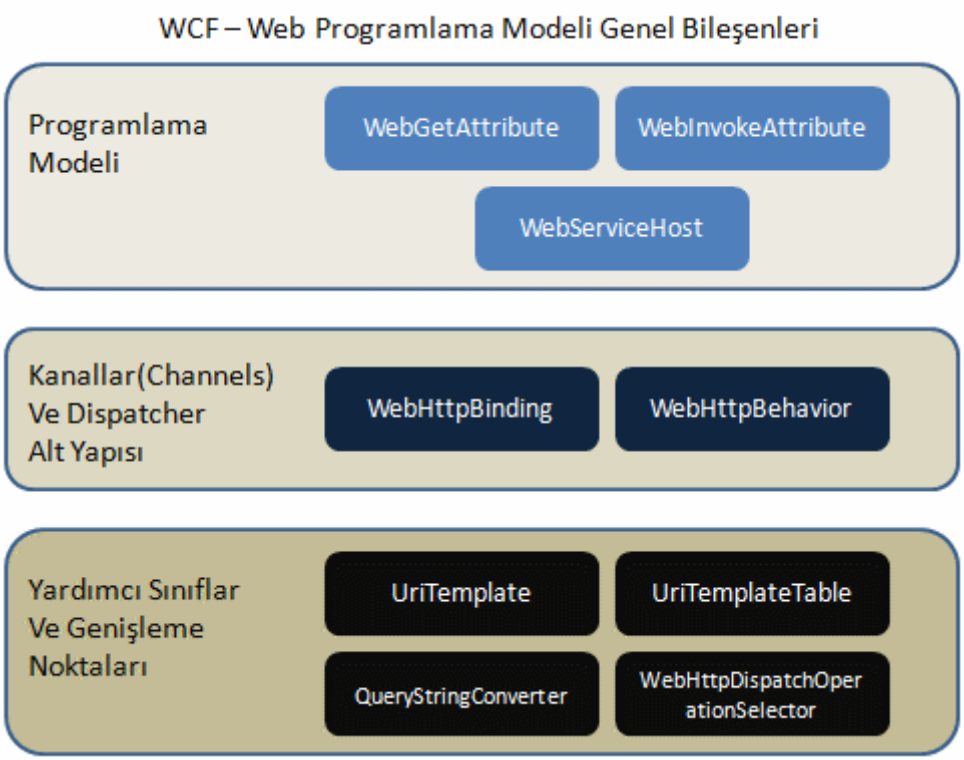
Web programlama modelinin en büyük avantajlarından biriside istemci(Client) tarafındaki uygulamaları düşünmeye gerek kalmadan **istemci-sunucu(Client/Server)** mimarisine uygun sistemler geliştirilebilmesidir. Basit olarak **HTTP** protokolünün farklı metodlarına göre işleyen bu sistemde, istemcilerin farklı tipte olabilecek **tarayıcı programlar(Browsers)** üzerinden talepte bulunmaları söz konusudur. özellikle **servis yönelimi mimari(Serivce Oriented Architecture)** yaklaşımlarına bakıldığında örneğin Xml Web Servislerinde**(Xml Web Services) HTTP**protokolünün basit **GET** metoduna göre taleplerde(Requests) bulunulabilmektedir. Bu istemci tarafına bir **proxy** nesnesi koymadan servis fonksiyonelliklerini **HTTP** protokolünün basit bir metoduna göre çağırabilme anlamınada gelmektedir. *(Bununla birlikte Web servislerinde **SOAP(Simple Object Access Protocol)** protokolüne uygun olacak şekilde proxy kullanmadan talepte bulunulup cevap alınabileceğide bilinmektedir.)*

Hal böyle olunca **WCF(Windows Communication Foundation)** gibi gelişmiş bir **dağıtık mimari(Distributed Architecture)** modelinde Web bazlı servis desteği olmaması düşünülemez. Ne varki WCF, özellikle **.Net Framework 3.5** ile gelen yeni tipler sayesinde **Web bazlı programlama modeli(Web-Based Programming Model)** yeteneklerine kavuşmuştur. Bu model temel olarak **SOAP**bazlı olmayan **EndPoint** noktalarının tasarlanabilmesini sağlamaktadır. Bu sayede bir WCF servisi istemci tarafını programlamaya gerek kalmadan, basit tarayıcı arayüzleri sayesinde hizmet verebilmektedir.*(Tabi burada POST benzeri metodlarda istisnai bazı durumlar olabileceğide göz ardı edilmemelidir).* üstelik web programlama modelinin doğası gereği istemciler servis üzerindeki fonksiyonellikleri kullanırken **queryString** tarzında parametrik **taleplerde(Requests)** de bulunabilirler. Web programlama modelinde sağlanması gereken önemli bazı hususlar vardır. Buna göre;

- Modelin **URI(Uniform Resource Identifier)** desteği olmalıdır. Bu destek söz konusu servis operasyonlarının istemci tarafına parametrik olabilecek **şablonlara(template)** uyan adresler ile sunulabilmesi için gereklidir.
- **HTTP** protokolünün farklı metodları için destek olmalıdır. Günümüzde en çok kullanılanları **GET** ve **POST** metodlarıdır. Sadece istemci tarafına veri alma gibi işlemler söz konusu olduğunda GET metodu, tam tersine servis tarafına parametreler gönderip düzenleme, veri aktarma yada operasyon çağırma gibi işlemler söz konusu olduğunda POST,**PUT** gibi metodlar kullanılır.
- Farklı tipte veri formatlarına destek olmalıdır**(Multiple Data Format Support)**. Bu destek sayesinde **XML(eXtensible Markup Lanugage)**, **JSON(JavaScript Object Notation)**, **binary içerikli stream (video, resim, ses dosyası gibi), düz metin(Plain Text)** bazlı veriler kullanılabilmektedir.

NOT : *WCF Web programlama modeli, servislerin **REST(REpresentational State Transefer)** tipinde geliştirilebilmesini sağlamaktadır. **REST**, **www(World Wide Web)** gibi sistemlerin prensiplerini referans etmektedir. Bu prensipler basit olarak bir **network** mimarisinin kaynakları nasıl adresleyeceği ve ne şekilde tanımlayacağını belirtmektedir.* ***Detaylı bilgi için Wikipedia***

Tahmin edileceği üzere **WCF** mimarisi **.Net Framework 3.5** ile gelen tipler sayesinde bu üç temel özelliğede destek verecek şekilde genişletilmiştir. WCF mimarisindeki bu yeni genişlemede rol alan başlıca **tipler(Types)** aşağıdaki şekilde görüldüğü gibi ele alınabilirler.



HTTP üzerinden **GET**, **POST, PUT** gibi metodlar ile gelen çağrıları **yönetimli kod(Managed Code)** tarafında ele almak için **WebGetAttribute** ve **WebInvokeAttribute nitelik(attribute)** sınıflarından yararlanılmaktadır. Bu nitelikler temel olarak, operasyonların **URI** bilgilerine nasıl bağlanacağını ve hangi **HTTP** metodları ile eşleştirileceğini belirlemekte kullanılırlar. WebGetAttribute niteliği **GET** operasyonları için ele alınırken, **WebInvokeAttribute POST, DELETE, PUT** gibi operasyonların kullanımını sağlamaktadır.

__NOT : HTTP__ üzerinden __POST__, __GET__, __PUT__, __DELETE__ metodları kullanılır. Bu metodların işleyiş şekli çoğu zaman __CRUD__ tablosunda aşağıdaki gibi ifade edilir.

HTTP	**CRUD**
POST	Create, Update, Delete
GET	Read
DELETE	Delete
PUT	Create, Update

WebGetAttribute sınının dört
önemli **özelliği(property)** vardır. **WebMessageBodyStyle** enum sabiti tipinden olan **BodyStyle** özelliği yardımıyla parametre ve dönüş değerlerinin**XML** elementleri içeriğine **sarmalanıp(wrap)** sarmalanmayacağına karar

verilir. **WebMessageBodyStyle** enum sabitinin içerdiği değerler **Bare**, **WrappedRequest**, **WrappedResponse** ve**Wrapped**' dır. Bare değerine göre **talep(request)** ve **cevap(response)** mesajlarındaki veriler sarmalanmazlar. **Wrapped** değerine göre her iki haldede sarmalanırlar. **WrappedRequest** değerine göre sadece taleplerin sarmalanması, **WrappedResponse** değerine göre ise sadece cevapların sarmalanması söz konusudur. **WebGetAttribute** sınıfının diğer özellikleri **RequestFormat**,**ResponseFormat** ve **UriTemplate** üyeleridir.

RequestFormat ve **ResponseFormat** özellikleri **WebMessageFormat enum** sabiti tipinden değerler almaktadır. WebMessageFormat enum sabitinin değerleri **XML** veya **JSON(JavaScript Object Notation)** olabilir. Bir başka deyişle cevapların ve taleplerin hangi formatta oluşturulacağına karar verilir. **WebGetAttribute** sınıfının belkide en önemli üyeside **UriTemplate** özelliğidir. Bu özelliğe atanan değer ile **GET** çağrımı için bir **URI şablonu(template)** belirlenmiş olur. **WebInvokeAttribute** sınıfıda **WebGetAttribute** sınıfı ile aynı özelliklere sahiptir. Nitekim birde ek özelliği vardır ki buda **Method** isimli üyedir. Method özelliği **string** bazlıdır ve **POST**, **DELETE** yada **PUT** gibi **HTTP** metodlarını göstermektedir. Bu özelliğin varsayılan değeri **POST** olarak belirlenmiştir.

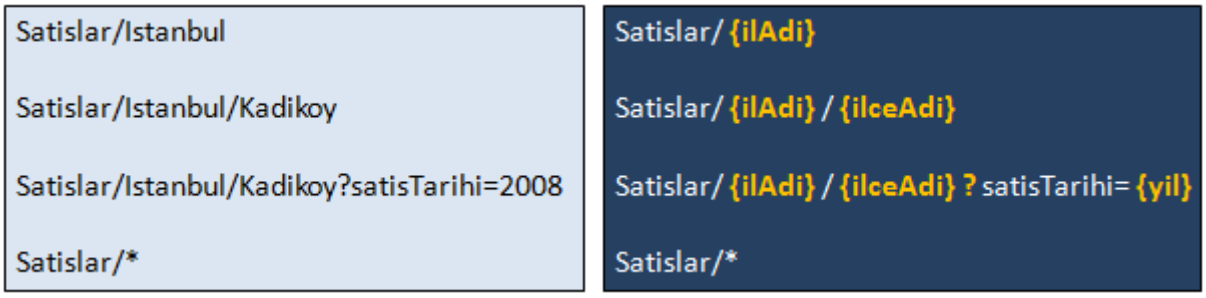
Web programlama modelinde yer alan önemli tiplerden bir diğeride **WebHttpBinding** sınıfıdır. Bu **bağlayıcı tip(Binding Type)** ile servis **EndPoint** noktalarının, **SOAP** bazlı mesajlaşma yerine **POX(Plain Old Xml)** bazlı mesajlaşmaya izin verecek şekilde çalışması sağlanmaktadır. Buna göre **Web bazlı EndPoint(Web-Based EndPoint)** noktalarının **POX(Plain Old XML)** bazlı mesajlaşma yaptığı söylenebilir. Diğer taraftan bu bağlayıcı tip **XML**,**JSON** ve anlaşılması güç **binary** tipteki(çoğunlukla video, resim gibi) verilerin okunması ve yazılması amacıyla kullanılmaktadır.

NOT : *WCF Web programlama modeli* ***SOAP(Simple Object Access Protocol)*** *bazlı mesajlaşmayı kullanmadığından* ***WS-**** *protokollerini desteklemez. Ancak WCF mimarisinin aynı servis üzerinde birden fazla* ***EndPoint*** *konuşlandırabilme özelliği nedeni ile* ***SOAP*** *destekli olan ve olmayan* ***EndPoint*** *noktalarının bir arada kullanılması mümkündür. Bir başka deyişle bir servisin ilgili operasyonları* ***WS-**** *destekli olacak şekilde bir* ***EndPoint*** *üzerinden sunulurken, aynı servis üzerindeki diğer bir* ***EndPoint*** *Web programlama modelini ele alabilir.*

WebServiceHost sınıfı **SOAP** bazlı olmayan web stilindeki servislerin host edilmesi için **ServiceHost** tipinden türetilmiştir. Eğer servisi host eden uygulamada çalıştırılan **WebServiceHost**nesne örneği, Web bazlı herhangibir **EndPoint** bulamassa otomatik olarak bir tane üretecektir. Bu üretim sırasında **servis adresini(Base Address)** baz alan bir **EndPoint** noktası oluşturulur.

Web programlama modelinde yer alan yardımcı tiplerden **UriTemplate** ve **UriTemplateTable** son derece önemli görevler üstlenmektedir.

Herşeyden önce **HTTP** bazlı talepler servis operasyonlarına ulaştıklarında bir **URI** tarafından karşılanmalıdır. Bu URI bilgilerinin **yönetimli kod (Managed Code)** tarafında ifade edilmesinde söz konusu tipler görev almaktadır. **URI şablonları(URI Templates)** temelde **path** ve **query** olmak üzere iki temel parçadan oluşurlar. Söz gelimi aşağıdaki şekilde bazı örnek URI şablonları yer almaktadır.



Burada sol taraftaki kutucukta gerçek kullanımlar yer alırken sağ taraftada karşılık gelen **URI** şablonları görülmektedir. Süslü parantezler içerisinde yer alan kısımlar değişken olmakla birlikte diğer kısımlar sabittir. Elbette bu URI' lerin başında birde **base URI address** bilgisi gelmektedir. **UriTemplate** sayesinde bir **URI** şablonunun kolay bir şekilde oluşturulması, var olan bir gerçek **URL** adresi ile **eşleştirilmesi(Match)** gibi işlemler yapılabilir. **UriTemplate** tipi çoğunlukla **WebGetAttribute** yada **WebInvokeAttribute** nitelikleri ile kullanılır. çok doğal olarak bir servisin sunabileceği birden fazla URI şablonu ve dolayısıyla **UriTemplate** nesnesi söz konusu olabilir. Birden fazla UriTemplate tipinin bir arada daha kolay yönetebilmek içinde**UriTemplateTable** isimli sınıf kullanılmaktadır. Bu iki tipin kullanımı önem arz ettiği için basit bir **Console** uygulması üzerinde fonksiyonelliklerini incelemekte yarar vardır. Bu amaçla **Visual Studio 2008** ortamında .**Net Framework 3.5** şablonunda bir **Console** uygulaması açtığımızı ve **System.ServiceModel.Web.dll assembly**' ını ilgili projeye referans ettiğimizi düşünelim. Bu assembly tahmin edileceği üzere Web programlama modeli için gerekli temel tipleri bünyesinde barındırmaktadır. Aşağıdaki kod parçası basit olarak **UriTemplate** tipinin kullanımını örneklemektedir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel.Web;
using System.Collections.Specialized;

namespace YardimciTipler
{
   class Program
   {
      static void Main(string[] args)
```

```
    {
        #region UriTemplate Kullanımı

        // Yeni bir URI şablonu oluşturulurken süslü parantezler içerisinde yer alan verilen değişken diğerleri ise sabittir.
        UriTemplate uriTemp = new UriTemplate("satislar/{sehirAdi}/{ilceAdi}?tarih={satisTarihi}");

        // değişkenlerin pozisyonuna göre bağlama
        // Bu metod ile süslü parantezler içerisine gelecek veriler sıralı olarak eklenirler
        Uri uri1 = uriTemp.BindByPosition(new Uri("http://localhost"), "Istanbul", "Kadıköy", "2007");

        // değişkenlerin adlarına göre bağlama
        // Bu metodda parametre adı - değeri eşleştirmesi bilgilerini taşıyan NameValueCollection koleksiyonu kullanılır.
        NameValueCollection values = new NameValueCollection()
            {
               {"sehirAdi","Ankara"}
               ,{"ilceAdi","Esenboğa"}
               ,{"satisTarihi","2007"}
            };
        Uri uri2 = uriTemp.BindByName(new Uri("http://www.bsenyurt.com/servisler"), values);

        // Eşleştirme yapmak ve doğruluğunu tespit etmek
        Uri uri3 = new Uri("http://localhost/satislar/Istanbul/Besiktas?tarih=2008");
        // Eğer Match metodu geriye null değer döndürmemişse URI içeriği şablonda belirtilene uygundur.
        UriTemplateMatch match=uriTemp.Match(new Uri("http://localhost/"), uri3);
        if (match != null)
           Console.WriteLine("Match geçerlidir");

        #endregion
    }
  }
}
```

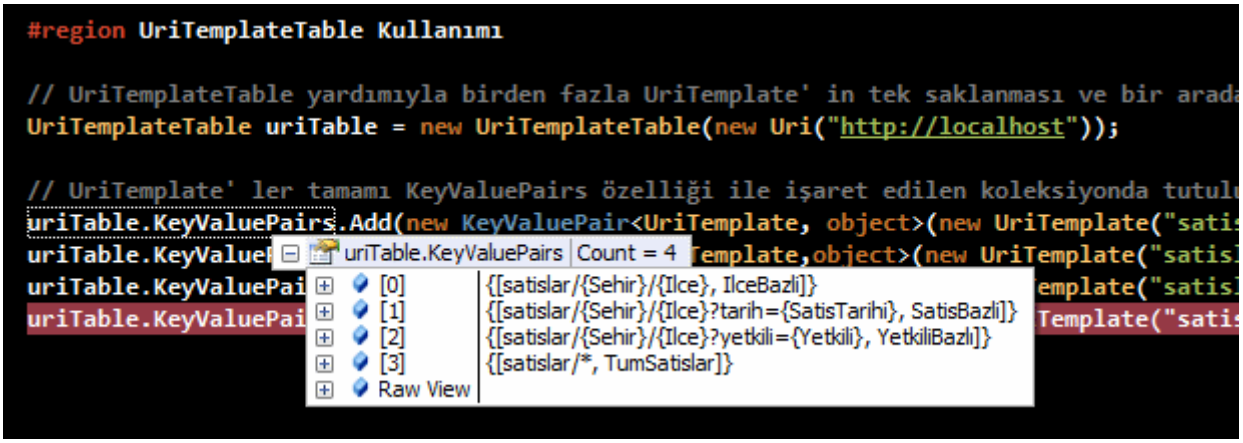
örnektende görüldüğü üzere bir **UriTemplate** nesnesi örneklendiğinde çoğunlukla bir **URI** şablonuda tanımlar. Sonrasında bu şablonda yer alan parametrelere değer aktarımı için **BindByPosition** yada **BindByName** gibi metodlar kullanılabilir. Diğer taraftan bir **URI** bilgisinin, **UriTemplate** ile belirtilen şablona uygunluğunu denetlemek için **Match** metodundan yararlanılabilir.*(**BindByName** metodunun uygulanışı sırasında **C#***

3.0 object initializers *özelliğinden yararlanıldığınada dikkat edelim.)* Diğer taraftan aşağıdaki kod parçasında da basit olarak **UriTemplateTable** kullanımı örneklenmektedir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel.Web;
using System.Collections.Specialized;

namespace YardimciTipler
{
   class Program
   {
      static void Main(string[] args)
      {
         #region UriTemplateTable Kullanımı

         // UriTemplateTable yardımıyla birden fazla UriTemplate' in tek saklanması ve bir arada tutulması sağlanabilir.
         UriTemplateTable uriTable = new UriTemplateTable(new Uri("http://localhost"));

         // UriTemplate' ler tamamı KeyValuePairs özelliği ile işaret edilen koleksiyonda tutulurlar.
         uriTable.KeyValuePairs.Add(new KeyValuePair<UriTemplate, object>(new UriTemplate("satislar/{Sehir}/{Ilce}"), "IlceBazli"));
         uriTable.KeyValuePairs.Add(new KeyValuePair<UriTemplate,object>(new UriTemplate("satislar/{Sehir}/{Ilce}?tarih={SatisTarihi}"),"SatisBazli"));
         uriTable.KeyValuePairs.Add(new KeyValuePair<UriTemplate,object>(new UriTemplate("satislar/{Sehir}/{Ilce}?yetkili={Yetkili}"),"YetkiliBazlı"));
         uriTable.KeyValuePairs.Add(new KeyValuePair<UriTemplate, object>(new UriTemplate("satislar/*"), "TumSatislar"));

         foreach (KeyValuePair<UriTemplate, object> kv in uriTable.KeyValuePairs)
         {
            Console.WriteLine("{0} : \t {1}", kv.Value, kv.Key);
         }
         #endregion
      }
   }
}
```

UriTemplateTable sınıfı **UriTemplate** örneklerini **KeyValuePairs** isimli özelliğin işaret ettiği koleksiyonda saklamaktadır. Yukarıdaki kod parçasına çalışma

zamanında **debug**modda bakıldığında **uriTable** isimli değişkenin içeriğinin aşağıdaki gibi olduğu görülebilir.



Web stilinde tasarlanan servisler bir tarayıcı uygulama yardımıyla **URL** bazında çağırılabilirler. URL üzerinden servis tarafına gelecek olan web bazlı taleplerde kullanılan parametrelerinin veri formatlarıda önemlidir. Bunlar aynı zamanda çağırılan operasyonların geri döndürdüğü değer tipleri içinde önemlidir. Kullanılabilecek veri formatları

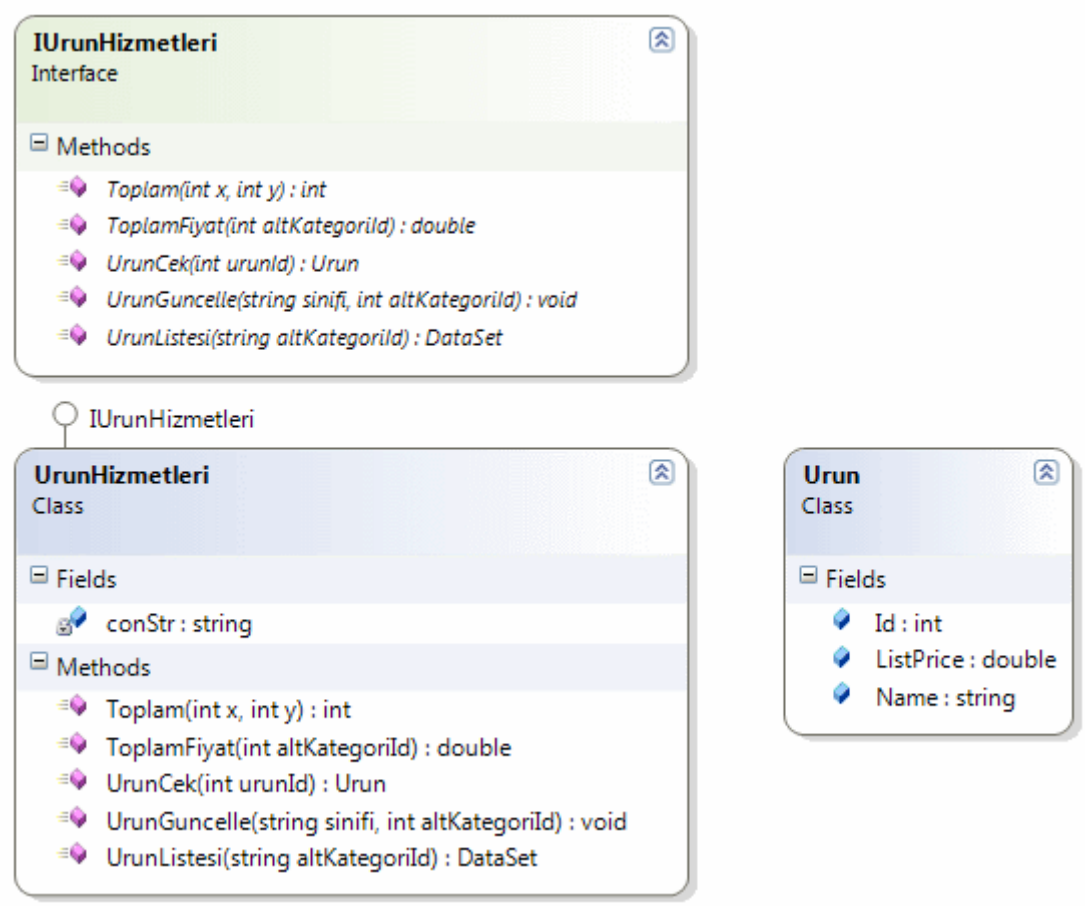
- **Byte**
- **SByte**
- **Int16**
- **Int32**
- **Int64**
- **UInt16**
- **UInt32**
- **UInt64**
- **Single**
- **Double**
- **Char**
- **Decimal**
- **Boolean**
- **String**
- **DateTime**
- **TimeSpan**
- **Guid**
- **DateTimeOffset**
- **Enums**
- **TypeConverterAttribute** niteliğini uygulayan

tipler olabilir.

NOT : WCF Web programlama modelinde *dikkat edilmesi gereken noktalardan biriside* ***güvenlik(Security)*** *dir. Bu modele göre güvenlik sadece* ***HTTPS*** *şeklinde yani* ***iletişim(transport)****seviyesinde sağlanabilir. Bunun nedeni Web tabanlı WCF*

*modelinin **SOAP zarflarını(Envelope)** kullanmayışıdır. Normal şartlarda **mesaj seviyesinde(Message Level Security)** güvenlik uygulandığında güvenlik ile ilişkili bazı bilgiler **SOAP** zarflarında yer alan **başlık(Header)** kısmına yazılır. Oysaki web bazlı modelde bu tip bir alan yoktur. Dolayısıyla sadece iletişim seviyesinde güvenlik uygulanabilmektedir.*

Artık Web bazlı örnek bir servis geliştirerek devam edebiliriz. İlk olarak **servis sözleşmesini(Service Contract)** ve uygulayıcı tipi barındıracak olan **WCF servis kütüphanesini(WCF Service Library)** geliştirmekte yarar vardır. Söz konusu kütüphanenin Web tabanlı model tiplerini kullanabilmesi için doğal olarak **System.ServiceModel.Web.dll assembly**' ını referans etmesi gerekmektedir. Söz konusu kütüphanedeki tiplerin aşağıdaki gibi tasarlandığı varsayılabilir.



IUrunHizmetleri isimli servis sözleşmesi **int**, **double**, **DataSet**, **Urun** tipinden değerler döndüren operasyonlar tanımlamaktadır. Aynı zamanda **WebInvoke niteliğinin(attribute)** kullanımını örneklemek amacıyla değer dönürmeyen bir operasyon daha tanımlamaktadır. **UrunCek** isimli operasyon geriye **Urun** tipinden bir nesne örneği döndürmektedir. Urun sınıfının içeriği aşağıdaki gibidir.

```
[DataContract]
public class Urun
```

```
{
   [DataMember]
   public int Id;
   [DataMember]
   public string Name;
   [DataMember]
   public double ListPrice;
}
```

Urun sınıfı basit olarak Product tablsoundaki belirli bir satırın ProductId,Name ve ListPrice alanlarının değerlerini taşımak üzere tasarlanmıştır. çok doğal olarak Urun nesne örneği **serileşme(Serialization)** işlemine tabi tutulacağından **DataContract** ve **DataMember** nitelikleri(attributes) ile imzalanmıştır. Bir başka deyişle bir **veri sözleşmesi(Data Contract)**tanımlanmaktadır. **Servis sözleşmesini(Service Contract)** taşıyan IUrunHizmetleri **arayüzünün(Interface)** içeriği ise aşağıdaki gibidir.

```
[ServiceContract]
public interface IUrunHizmetleri
{
   [OperationContract]
   [WebGet(UriTemplate="urunler/{altkategoriId}")]
   DataSet UrunListesi(string altKategoriId);

   [OperationContract]
   [WebGet]
   Urun UrunCek(int urunId);

   [OperationContract]
   [WebInvoke(UriTemplate="guncelle?sinifi={sinifi}&altKategori={altKategoriId}")
]
   void UrunGuncelle(string sinifi, int altKategoriId);

   [OperationContract]
   [WebGet(UriTemplate="toplamaIslemi?sayi1={x}&sayi2={y}")]
   int Toplam(int x, int y);

   [OperationContract]
   [WebGet(UriTemplate="toplam?kategori={altKategoriId}")]
   double ToplamFiyat(int altKategoriId);
}
```

UrunListesi isimli operasyon geriye **DataSet** döndürmektedir. Bu DataSet içeriği ürünlerin alt kategori değerine göre elde edilmektedir. Söz konusu operasyon **HTTP GET** metoduna göre talep edilebilir. Dikkat edileceği üzere **WebGet** niteliğinin **UriTemplate** özelliğinde

süslü parantezler içerisinde metod parametresi ile aynı isimde olacak şekilde bir tanımlama yapılmaktadır. UrunCek operasyonu geriye Urun tipinden bir nesne örneği döndürmekle birlikte urunId değeri queryString üzerinden alınmaktadır. **WebGet** niteliğinde **UriTemplate** kullanılmaması nedeniyle talepte **UrunCek?urunId=1** gibi bir adres kullanılmalıdır. Bir başka deyişle **WebGet** niteliğinde **UriTemplate** kullanılmadığı durumlarda, **MetodAdı?parametreAdı1=değeri1&parametreAdı2=değeri2** tarzı adresler ile talepte bulunulabilinir. UrunGuncelle isimli operasyon, içerisinde sinifi ve altKategori queryString parametrelerini bulunduran adres taleplerine **HTTP POST** metoduna göre cevap verecek şekilde tanımlanmaktadır. Bu sebepten dolayı **URL** satırından bir bilgi gönderilmesine gerek yoktur. Toplam operasyonu **URI** bilgisi içerisinde birden fazla**queryString** parametresini ele alıp basit bir veri tipini geriye döndürecek şekilde tasarlanmıştır. Benzer şekilde ToplamFiyat operasyonuda belirli bir alt kategerideki ürünlerin toplam liste fiyatı değerini bulacak bir fonksiyonelliği tanımlamaktadır. Söz konusu **servis sözleşmesini(Service Contract)** uygulayan sınıfın içeriği ise aşağıdaki gibidir.

```
public class UrunHizmetleri
    : IUrunHizmetleri
{
   string conStr = "data source=.;database=AdventureWorks;integrated security=SSPI";
   #region IUrunHizmetleri Members

   public DataSet UrunListesi(string altKategoriId)
   {
      DataSet set = null;
      using (SqlConnection conn = new SqlConnection(conStr))
      {
         SqlCommand cmd = new SqlCommand("Select
ProductId,Name,ListPrice,ProductSubCategoryId,Class,SellStartDate From
Production.Product Where ProductSubCategoryId=@SubCatId",     conn);
         cmd.Parameters.AddWithValue("@SubCatId", altKategoriId);
         SqlDataAdapter adapter = new SqlDataAdapter(cmd);
         set = new DataSet();
         adapter.Fill(set);
      }
      return set;
   }

   public Urun UrunCek(int urunId)
   {
      Urun u = null;
      using (SqlConnection conn = new SqlConnection(conStr))
      {
         SqlCommand cmd = new SqlCommand("Select ProductId,Name,ListPrice From
```

```
Production.Product Where ProductId=@PrdId", conn);
            cmd.Parameters.AddWithValue("@PrdId", urunId);
            conn.Open();
            SqlDataReader reader = cmd.ExecuteReader();
            if (reader.Read())
            {
               u = new Urun()
                     {
                        Id=urunId
                        ,Name=reader["Name"].ToString()
                        , ListPrice=Convert.ToDouble(reader["ListPrice"])
                     };
            }
            reader.Close();
         }
         return u;
      }

      public void UrunGuncelle(string sinifi, int altKategoriId)
      {
         using (SqlConnection conn = new SqlConnection(conStr))
         {
            SqlCommand cmd = new SqlCommand("Update Production.Product Set
ListPrice=ListPrice+1 Where Class=@Class and ProductSubCategoryId=@SubCatId",
conn);
            cmd.Parameters.AddWithValue("@Class", sinifi);
            cmd.Parameters.AddWithValue("@SubCatId", altKategoriId);
            conn.Open();
            cmd.ExecuteNonQuery();
         }
      }

      public int Toplam(int x, int y)
      {
         return x + y;
      }

      public double ToplamFiyat(int altKategoriId)
      {
         double result = 0;
         using (SqlConnection conn = new SqlConnection(conStr))
         {
            SqlCommand cmd = new SqlCommand("Select Sum(ListPrice) From
Production.Product Where ProductSubCategoryId=@SubCatId", conn);
            cmd.Parameters.AddWithValue("@SubCatId", altKategoriId);
```

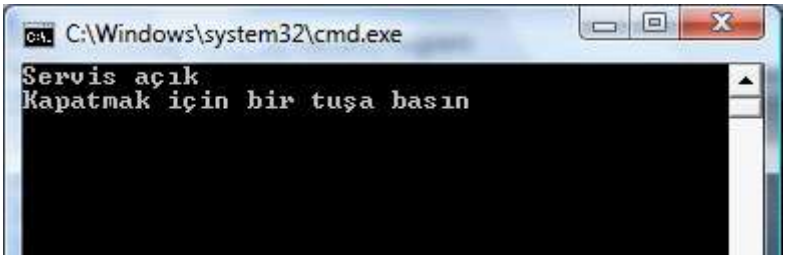
```
            conn.Open();
            result =Convert.ToDouble(cmd.ExecuteScalar());
        }
        return result;
    }

    #endregion
}
```

UrunHizmetleri isimli sınıf içerisinde yer alan operasyon uyarlamalarının çoğu **AdventureWorks** veritabanında yer alan **Production.Product** tablosu üzerinden çalışmaktadır. Dönüş tipleri için önemli olan serileşebilmeleridir. Şimdi bu servis kütüphanesini sunacak olan basit bir **host** uygulama yazılabilir. Bu amaçla bir **Console** projesi göz önüne alınabilir. Sunucu uygulamanın**System.ServiceModel.Web.dll**, **System.ServiceModel.dll assembly**' ları haricinde **servis sözleşmesini(Service Contract)** ve uygulayıcı tipi barındıran kütüphaneyide**(WCF Service Library)**referans etmesi gerekmektedir. Bu işlemlerin ardından Host uygulama kodları aşağıdaki gibi geliştirilebilir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using UrunLib;
using System.ServiceModel.Web;
using System.ServiceModel;
using System.ServiceModel.Description;

namespace Sunucu
{
    class Program
    {
        static void Main(string[] args)
        {
            WebServiceHost host = new WebServiceHost(typeof(UrunHizmetleri), new
Uri("http://localhost:60001/"));
            ServiceEndpoint ep=host.AddServiceEndpoint(typeof(IUrunHizmetleri), new
WebHttpBinding(), "");

            host.Opened += delegate(object sender, EventArgs e)
                        {
                            Console.WriteLine("Servis açık");
                        };
            host.Closed += delegate(object sender, EventArgs e)
                        {
```

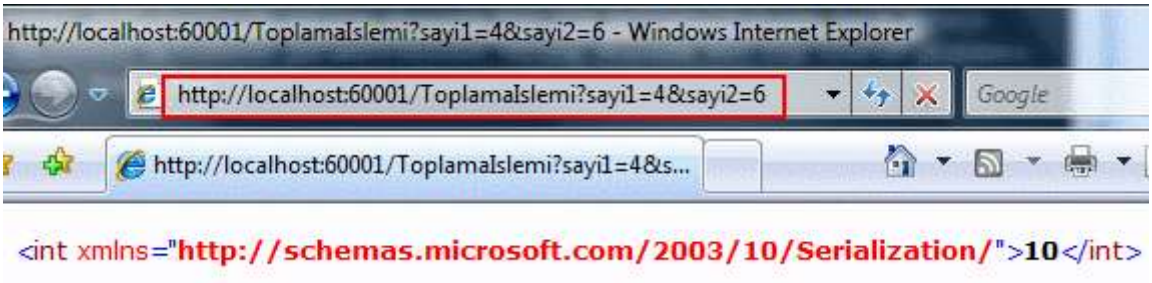
```
                    Console.WriteLine("Servis kapatıldı");
                };

        host.Open();

        Console.WriteLine("Kapatmak için bir tuşa basın");
        Console.ReadLine();
        host.Close();
    }
  }
}
```

İlk olarak bir adet **WebServiceHost** nesnesi örneklenmektedir. **WebServiceHost** sınıfının **yapıcı metoduna(Constructor)** parametre olarak sözleşmeyi uygulayan **sınıf tipi(Class Type)** ve yayınlanacağı adres bilgisi verilmektedir. Sonrasında zorunlu olmamakla birlikte **WebHttpBinding** bağlayıcı tipini barındıran bir **EndPoint** host isimli WebServiceHost nesne örneğine eklenmektedir. Diğer **EndPoint** tanımlamalarında olduğu gibi önce **servis sözleşmesi(Service Contract)**, sonrasında **bağlayıcı tip(Binding Type)** bildirilir. **Servis adresi(Service Address)**ise zaten host nesnesi örneklenirken tanımlandığı için boş geçilebilir. Bu işlemlerin ardından servis **Open** metodu ile açılır. Eğer yazılan kodlarda herhangibir aksilik yoksa uygulamanın çalışma zamanı görüntüsü aşağıdakine benzer olmalıdır.

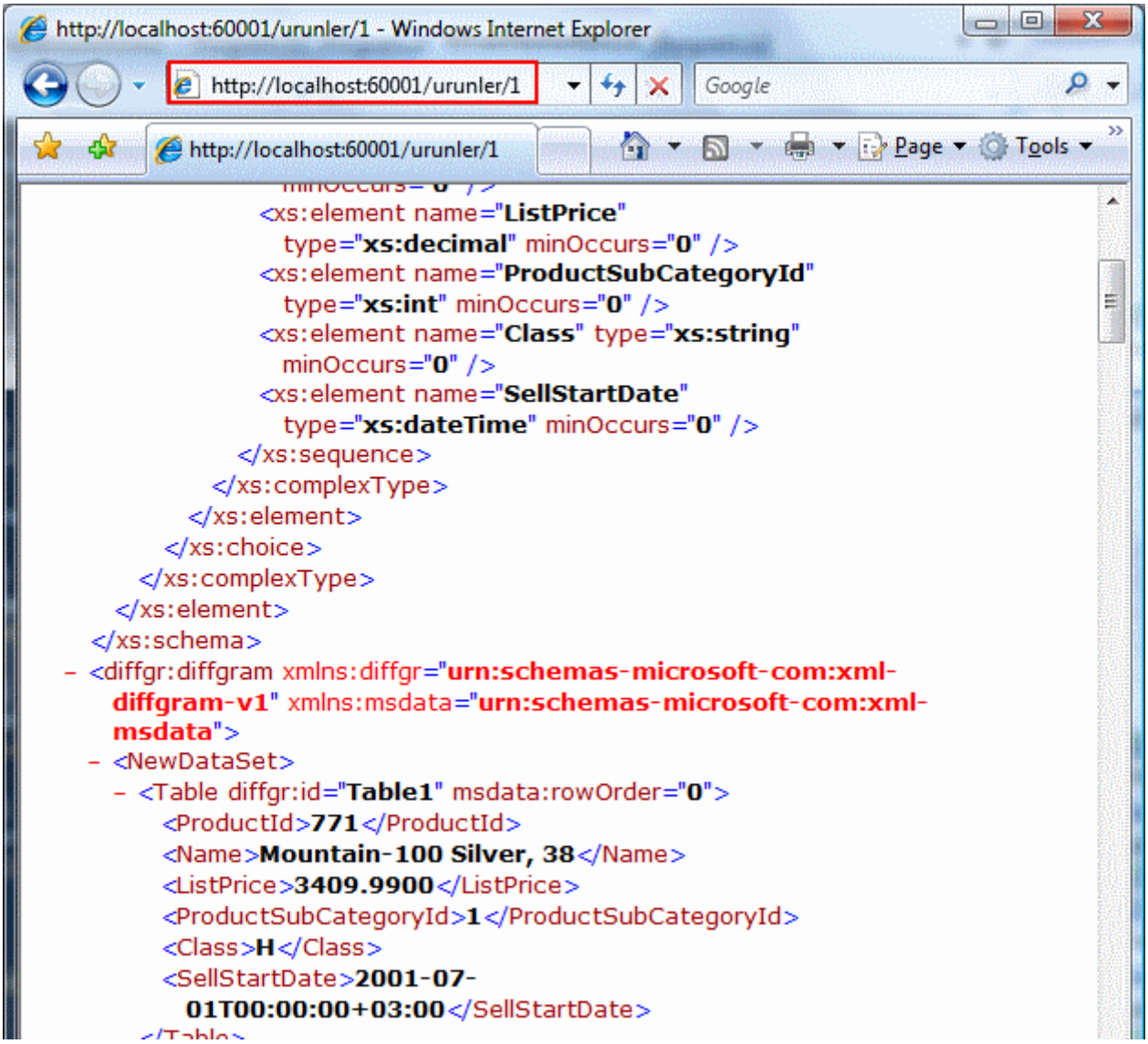


Artık **HTTP GET** metodu bazlı taleplerde bulunulabiliriz. Söz konusu **talepleri(Request)** basit bir **tarayıcı(Browser)** uygulaması yardımıyla gerçekleştirmek mümkündür. Elbette istemcilerin **cevap(Response)** alabilmeleri için Host uygulamanında çalışıyor olması şarttır. Aşağıda söz konuzu operasyon çağrıları ve ekran görüntüleri yer almaktadır.

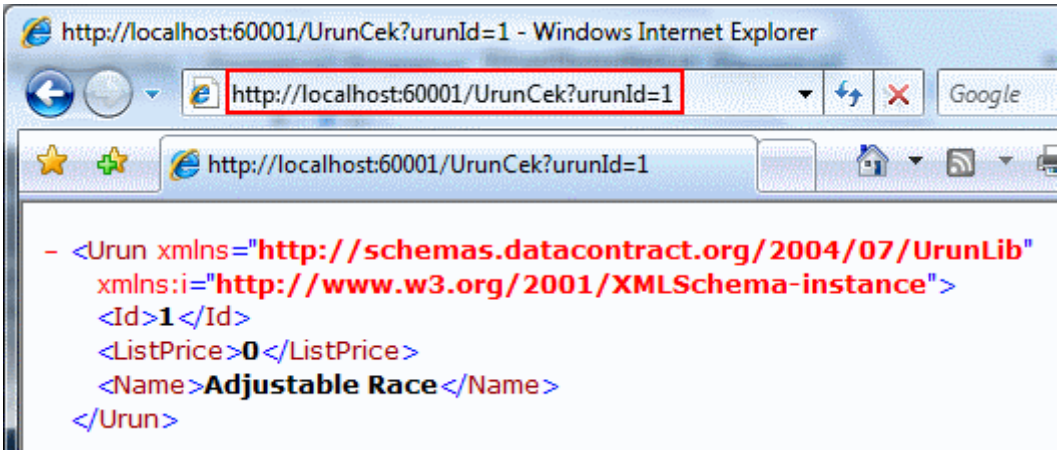
Toplama işlemi için yapılan çağrı ;
http://localhost:60001/ToplamaIslemi?sayi1=4&sayi2=6 *(4 ve 6 sayılarının toplamı geriye döndürülmektedir)*



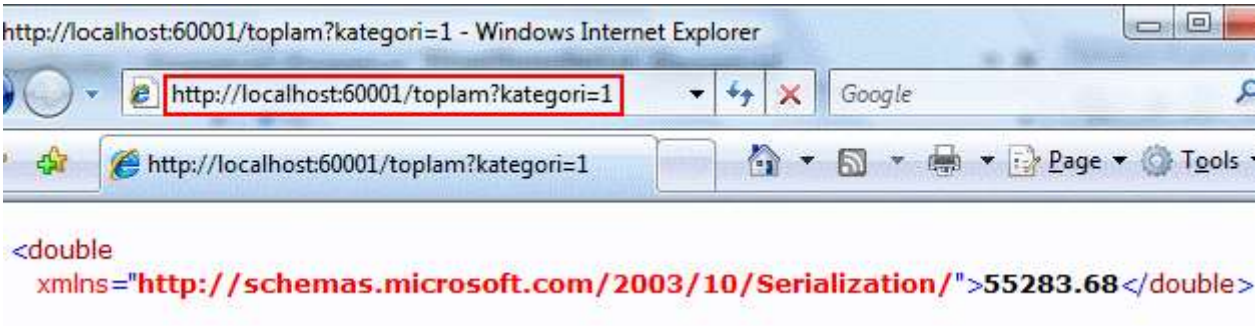
UrunListesi operasyonu için yapılan çağrı ; http://localhost:60001/urunler/1 *(1 numaralı ProductSubCategoryId değerine sahip ürünlerin **DataSet** içerisinde toplanmış listesini verir)*



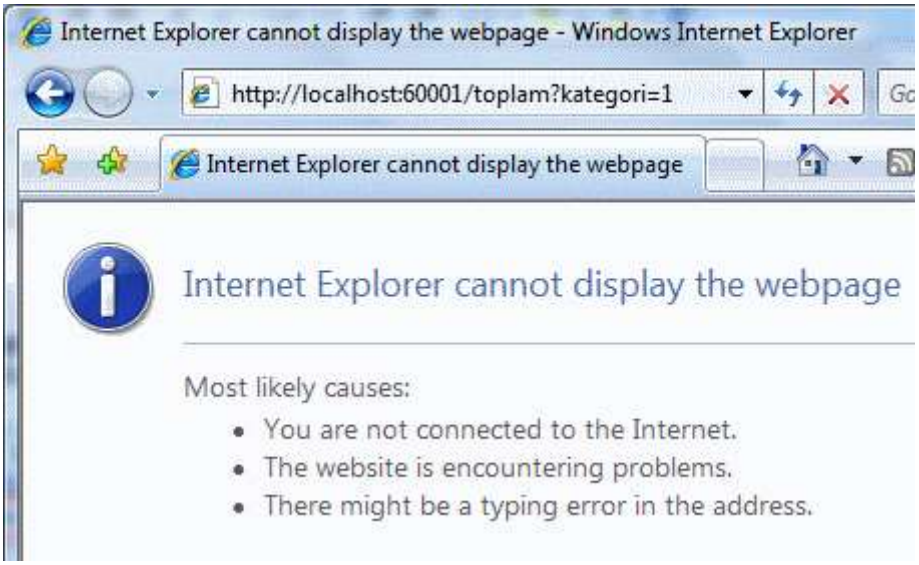
UrunCek isimli metoda yapılan çağrı ; http://localhost:60001/UrunCek?urunId=1 *(ProductId alanı değeri 1 olan Product tablosu satırının içeriği Urun tipine göre çekilir)*



ToplamFiyat operasyonu için yapılan çağrı ; http://localhost:60001/toplam?kategori=1 *(ProductSubCategoryId değeri 1 olan ürünlerin ListPrice alanlarının toplamı elde edilir.)*



Tabi sunucunun çalışmaması halinde gelecek olan istemci taleplerinde aşağıdaki gibi ekran görüntüleri ile karşılaşılabilir.



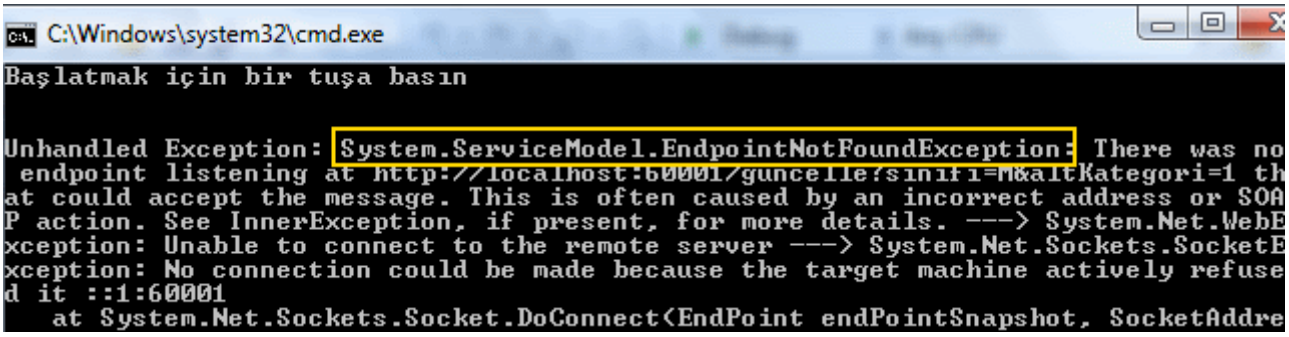
Görüldüğü gibi **URL** bazlı olacak şekilde servis operasyonları çağırılabilmekte ve sonuçları alınabilmektedir. Gelelim UrunGuncelle isimli metodun test edilmesine. Bu operasyon bir istemci uygulama üzerinden test edilebilir. **POST** metoduna göre bir talepte

bulunulacağından istemcinin **servis sözleşmesini(Service Contract)** varsa **veri sözleşmesini(Data Contract)** bilmesi gerekmektedir. Bunların dışından istemci uygulamanın **System.ServiceModel.dll, System.Runtime.Serialization.dll(3.0 versiyonu)** ve **System.ServiceModel.Web.dll assembly**' larını referans etmesi gerekmektedir. Bu amaçla hazırlanan örnek istemci bir **Console** uygulaması olarak tasarlanabilir. Söz konusu uygulamanın kodları ise aşağıdaki gibidir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;
using System.ServiceModel.Description;
using System.Data;
using System.ServiceModel.Web;
using System.Runtime.Serialization;

namespace Istemci
{
   [DataContract]
   public class Urun
   {
      [DataMember]
      public int Id;
      [DataMember]
      public string Name;
      [DataMember]
      public double ListPrice;
   }

   [ServiceContract]
   public interface IUrunHizmetleri
   {
      [OperationContract]
      [WebGet(UriTemplate = "urunler/{altkategoriId}")]
      DataSet UrunListesi(string altKategoriId);

      [OperationContract]
      [WebGet]
      Urun UrunCek(int urunId);

      [OperationContract]
      [WebInvoke(UriTemplate = "guncelle?sinifi={sinifi}&altKategori={altKategoriId}")]
      void UrunGuncelle(string sinifi, int altKategoriId);
```

```
        [OperationContract]
        [WebGet(UriTemplate = "toplamaIslemi?sayi1={x}&sayi2={y}")]
        int Toplam(int x, int y);

        [OperationContract]
        [WebGet(UriTemplate = "toplam?kategori={altKategoriId}")]
        double ToplamFiyat(int altKategoriId);
    }

    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Başlatmak için bir tuşa basın");
            Console.ReadLine();
            using (WebChannelFactory<IUrunHizmetleri> cf = new
WebChannelFactory<IUrunHizmetleri>(new Uri("http://localhost:60001")))
            {
                IUrunHizmetleri hiz=cf.CreateChannel();
                hiz.UrunGuncelle("M", 1);
            }
        }
    }
}
```

İlk olarak **WebChannelFactory generic** tipinden bir nesne örneklenmektedir.
Burada **generic** tip olarak servis sözleşmesi kullanılmalıdır. Dikkat edilecek olursa parametre olarak servisin host edildiği**URL** adresi verilmekedir. Sonrasında yapılan metod çağrısı **HTTP POST** tekniğine göre sunucuya ulaşacak ve operasyon sonuçlandırılacaktır. önce sunucu sonrasında ise istemci çalıştırılırsa istemci uygulama başarılı bir şekilde yürütülecek ve **Class** değeri **M**, **ProductSubCategoryID** değeri **1** olan **Product** tablosu satırlarının **ListPrice** değerlerinin 1 birim arttırıldığı görülecektir. Tahmin edileceği gibi istemci tarafında yapılan bu metod çağırısı,
sunucuya **HTTP** protokolünün **POST** metoduna göre ulaşacaktır. Metod içerisinde kullanılan parametre değerleride servis tarafına vardıklarında aynen alınıp değerlendirilebilecektir. Eğer sunucu çalışmıyorken istemci çalıştırılırsa çalışma zamanında **EndPointNotFoundException** sınıfı tipinden bir **istisna(Exception)** alındığı görülebilir.



Buraya kadar işlenenler göz önüne alındığında, istemcilerin **HTTP** üzerinden **GET,POST, PUT, DELETE** gibi metodları
kullanarak **WCF** operasyonlarını **talep(Request)** edebileceği görülmüştür. **Host** uygulama test olması açısından **Console** olarak tasarlanmış olmakla birlikte **IIS(Internet Information Service)** üzerindede konuşlandırılabilir. Bu tip bir durumda**svc** uzantılı servis dosyasında yer alan **Service** direktifinde **Factory** isimli **niteliği(attribute)** kullanmak ve **System.ServiceModel.Activation.WebServiceFactory** değerini atamak yeterlidir. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

WebStyleServices.rar (81,33 kb)

WCF - RSS, Atom Formatlı İçerik Paylaşımı(Syndication) (2008-02-08T04:56:00)

wcf,

Windows Communication Foundation, **Net Framework 3.5** ile gelen bazı yeni **CLR(Common Language Runtime)** tipleri sayesinde **RSS 2.0** ve **Atom 1.0** formatlarında yada diğer özel formatlarda **içerik paylaşımı(Syndication)** yapılmasına izin veren özelliklere sahip olmuştur. Bu tipler sayesinde bir WCF servisi(Service) üzerinden özellikle **HTTP** protokolünün **GET**, **POST**, **HEAD** ve benzeri metodlarına göre **talep-cevap(Request-Response)** işlemleri gerçekleştirilebilmektedir. Bir başka deyişle **EndPoint** noktaları üzerinden farklı tipte verilerin, dünya standartlarına uygun olacak şekilde yayınlanması mümkündür. İşte bu makalemizde bir **WCF** servisi üzerinden en basit haliyle **RSS** veya **Atom** formatında içerik paylaşımlarının nasıl yapılabileceğini incelemeye çalışacağız.

RSS veya **Atom** gibi formatların ortak noktası **platform bağımsız(Interoperability)** veri içerikleri sunulabilmesi için gerekli standartları içeriyor olmalarıdır. Bu sayede yayınlanan veriyi alacak olan istemcilerin(Clients) farklı özelliklerinin düşünülmesine gerek kalmamaktadır. Burada söz konusu olan platform bağımsız yayınlanabilen veriler genellikle **Feed** olarak adlandırılırlar. Feed yapısı kendi içerisinde, içerik yayınlaması ile ilgili olaraktan **yazar(author), başlık(title), adres(url)** ve bunlar ile ilişkili olan **metadata** bilgilerini barındırır. Ayrıca kendi içerisinde birden fazla **öğe(Item)** barındırabilir. Bu öğelerin her biride kendi içerisinde **başlık(title),**

adres(url), oluşturulma tarihi(creation date), açıklama(description), kategori(category) gibi bilgileri barındırmaktadır. Bu içeriklerin şu anda popüler olan iki farklı sunuş şekli **RSS(Really Simple Syndication)** ve **Atom** tekniklerdir. Her ikiside **XML(eXtensible Markup Languge)**tabanlı olacak şekilde içerik paylaşımı standartları sunarlar.

***NOT : RSS/Atom** gibi içerik yayınlama formatlarının ortak özellikleri, platform bağımsız bir içerik için gerekli olan **metadata** standartlarını sağlıyor olmalarıdır.*

Söz konusu formatların her ikisde **Windows Communication Foundation** tarafından desteklenmektedir. WCF, söz konusu **Feed** ve **Feed Item**' lar ile çalışılabilmesini kolaylaştırmak adına**SyndicationFeed**, **SyndicationItem**, **SyndicationLink**, **SyndicationPerson** gibi pek çok **CLR** tipi içermektedir. Tahmin edileceği üzere bu tiplerin isimlendirilmeleri, içerik paylaşım formatlarında kullanılan element adları ile benzerdir. Ancak en önemli avantaj, **RSS** yada **Atom** gibi formatlara **yönetimli kod tarafından(Managed Code)** doğrudan destek veriliyor olmasıdır. Şu anda WCF içerisinde farklı içerik paylaşımı formatlarına destek verebilmek amacıyla **Atom10FeedFormatter**, **RSS20FeedFormatter** vb tipler yer almaktadır.

Peki söz konusu **RSS** veya **Atom** formatlı içerikler hangi amaçlarla kullanılmaktadırlar? Söz gelimi haber sitelerinin hemen hepsi güncel başlıkları ortak bir standartta yayınlayabilmek için **XML**tabanlı olan **RSS** veya **Atom** formatlı içerikler sunarlar. Aynı sistem blog siteleri içinde geçerlidir. Pek çok blog sitesinde en güncel **girişler(Entry) RSS** veya **Atom** formatında(yada iki formatta birden olacak şekilde) yayınlanmaktadır. Elbetteki bu yayınlanan içerikler **XML** formatlı olduğundan, başka sistemler tarafından alınıp yorumlanabilirler. Söz gelimi haber başlıkları yada blog girişleri yada bir topluluk sitesinde yayınlanan son makalelerin listesi **XML** içeriklerinden alınıp işlenebilirler. İşlenen bu veriler uygulama bağımsız olacak şekilde değişik kontroller ile son kullanıcılara sunulabilirler. örneğin farklı haber sitelerinin **RSS/Atom** içeriklerini kullanarak istemcilere birer özet şeklinde sunan intranet tabanlı web siteleri son derece yaygındır. Bu tip bir sistemde istemciler doğrudan internete çıkamasalarda, **intranet** üzerinde erişebildikleri ortak bir **portal** üzerinden çeşitli haber sitelerinin güncel konu başlıklarına bakabilir ve bilgi alabilirler. Tabiki burada bahsetmiş olduğumuz senaryolar en yaygın kullanılanlarıdır. **Geliştirici(Developer)** olarak baktığımızda paylaşılabilen herhangibir veri topluluğunu **RSS/Atom**formatlarında yayınlayabileceğimiz sonucu ortaya çıkmaktadır.

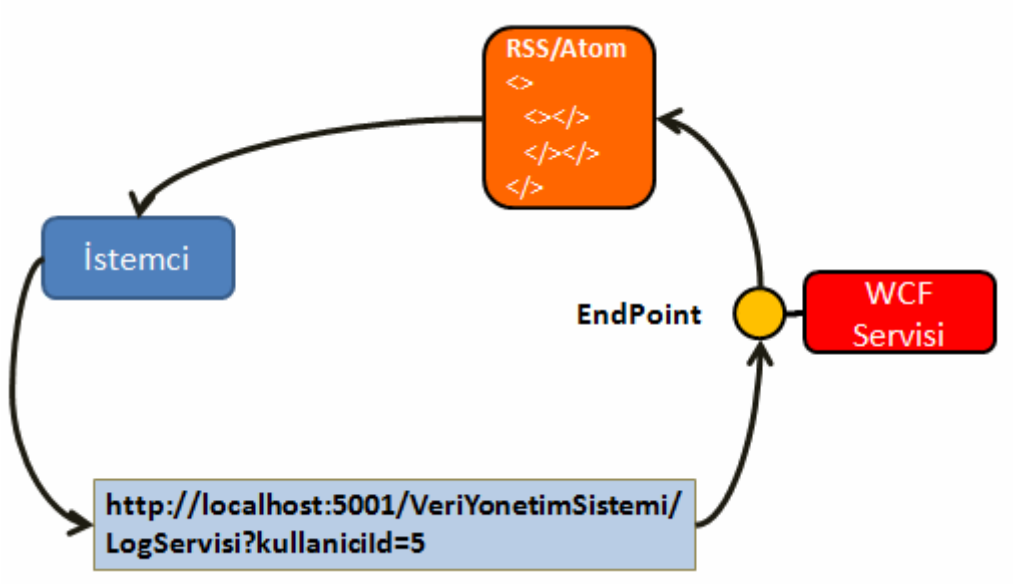
***NOT : RSS/Atom** gibi formatlarda sunulan içerikler standartlaştırılmış **XML** verileridir. Bu veriler versiyonlara göre farklılık gösterebilir. Ancak her haldede, XML içeriklerinin **ayrıştırılıp(Parse)** kullanılması mümkündür. **.Net** içerisinde ezelden beri gelen XML tipleri ile bu işlemler gerçekleştirilebilir. Ancak WCF açısından olaya bakıldığında göze çarpan noktalar şunlardır;*

- ***RSS** veya **Atom** formatlı verilerin **yönetimli kod(Managed Code)** tarafında kolayca ele alınmasını sağlayan tipler **.Net Framework 3.5** içerisinde gelmektedir.*

*Böylece **RSS** veya **Atom**formatlı içeriklerin oluşturulması veya okunması(ayrıştırılması) dahada kolaylaşmaktadır.*

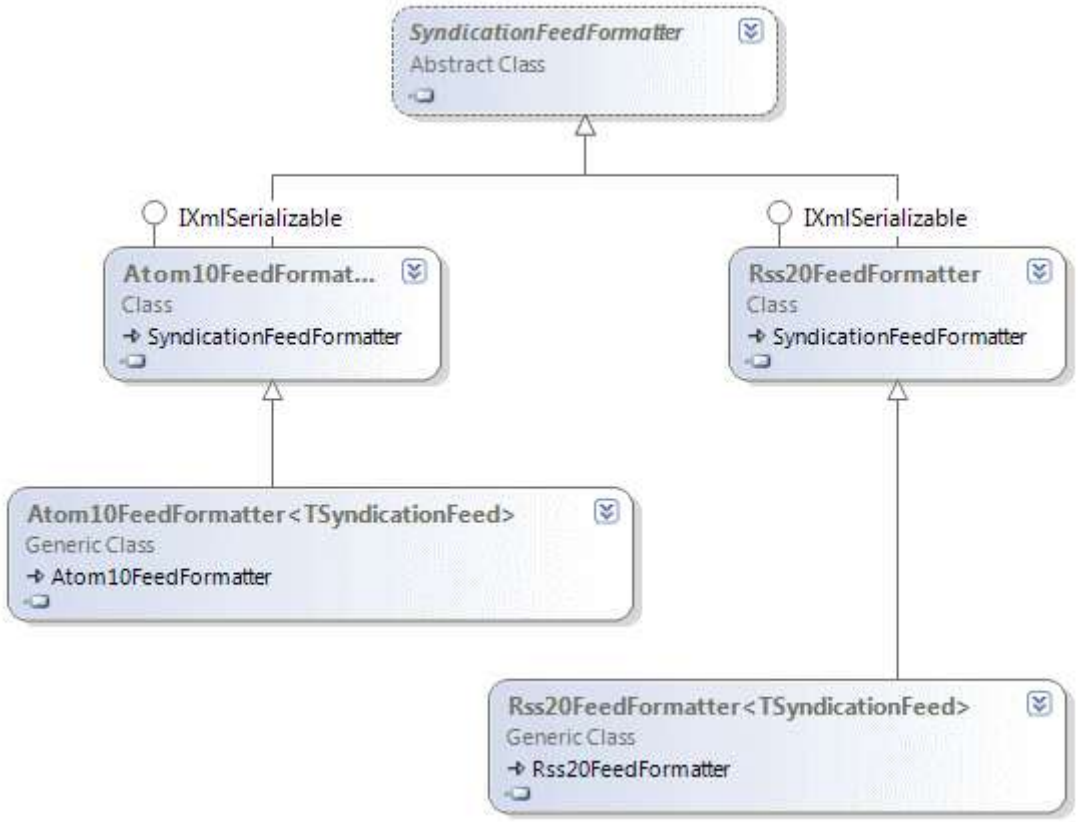
- ***WCF**, **.Net 3.5** içerisinde gelen destekler ile **HTTP Get** gibi bir metod yardımıyla **EndPoint**' ler üzerinde **RSS/Atom** desteği verebilecek şekilde kullanılabilmektedir.*

Söz gelimi bir veri yönetim sistemi üzerinde çalışan bir **WCF** servisi, log bilgilerini yetkili kişilere **RSS/Atom** formatında sunacak şekilde **URL** desteği verebilir. Burada URL desteğinden kasıt**http://localhost:5001/VeriYonetimSistemi/LogServisi?kullaniciId=5** gibi bir adrestir. Dikkat edilecek olursa URL üzerinden yapılacak olan bu **talep(request)** sonrasında, WCF servisi **kullanıcıID**değeri **5** olan kişiyi bulup, log bilgilerini **RSS/Atom** formatında hazırlayarak email olarak gönderebilir. Hemen bu noktada aşağıdaki şekil ile olayı daha net kavrayabiliriz.



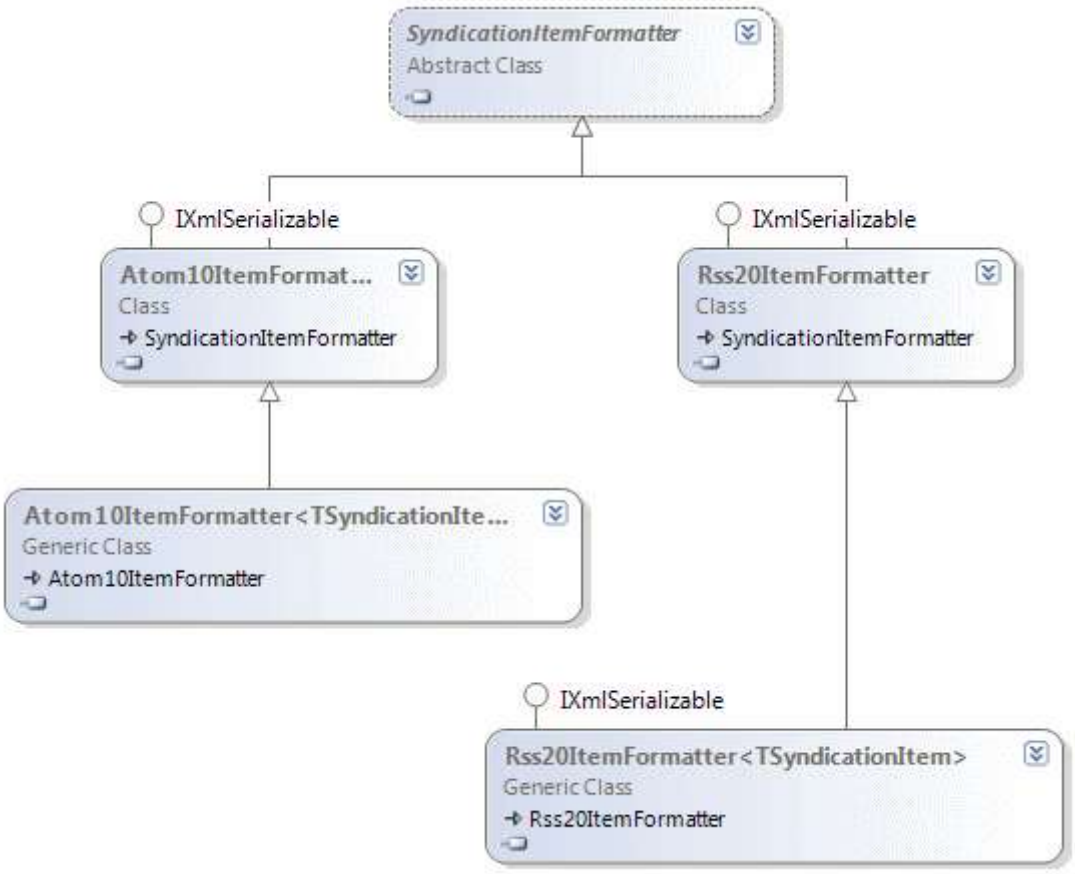
Şekilden de anlaşılacağı üzere servisin **HTTP Get** metoduna göre **RSS/Atom** desteği bulunmaktadır. Bu sistemin gerçekleştirilebilmesi için **.Net Framework 3.5** içerisinde **WebHttpBinding**ve **WebHttpBehavior** isimli yeni tipler yer almaktadır. Tahmin edileceği üzere **WebHttpBinding** yeni **bağlayıcı tiplerdendir(Binding Type)** ve **HTTP** protokolünün **Get** gibi metodlarına**EndPoint** üzerinden destek verilmesini sağlamak amacıyla geliştirilmiştir. Burada önemli olan noktalardan biriside servisin **RSS** yada **Atom** formatında veri içeriklerini nasıl hazırlayacağıdır. Yine daha öncedende belirtildiği gibi bu aslında **XML** formatlı bir metin içeriğinin hazırlanmasından başka bir şey değildir. Ne varki **.Net Framework 3.5** içerisinde gelen yardımcı tipler sayesinde bu işlemlerin **yönetimli kod(Managed Code)** tarafında yapılması mümkündür. Bu amaçla **SyndicationFeedFormatter** ve **SyndicationItemFormatter abstract** tiplerinden türemiş olan çeşitli sınıflar bulunmaktadır. Aşağıdaki sınıf diagramlarında temel olarak kullanılabilecek formatlama tipleri gösterilmektedir.

Feed formatlama için kullanılan tipler;



Görüldüğü gibi **Atom 1.0** ve **RSS 2.0** tarzındaki **Feed** formatları için ikişer tip yer almaktadır. Söz konusu sınıfların **generic** versiyonları olduğunada dikkat edelim.

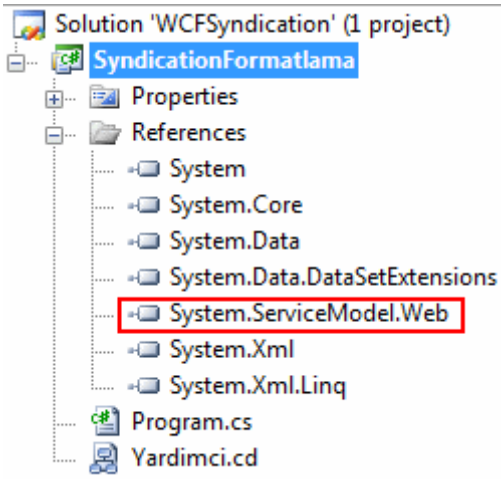
Item formatlama için kullanılan tipler;



Görüldüğü gibi **Atom 1.0** ve **RSS 2.0** standartlarına uygun olacak şekilde **Item** formatlaması için kullanılan ikişer farklı tip vardır. Bu tiplerden hepsi **SyndicationItemFormatter abstract**sınıfından türemekte ve **XML** serileştirmesi için gerekli **IXmlSerializable arayüzünü(Interface)** uygulamaktadır.Bu bilgiler ışığında kendi formatlama modellerimizide geliştirebileceğimizi söyleyebiliriz. Bu tarz bir işlem için **Feed** veya **Item** formatlamasının farklı bir versiyonunu yazmak istiyorsak **SyndicationItemFormatter**, **SyndicationFeedFormatter abstract** sınıfları ile **IXmlSerializable** arayüzünü göz önüne almamız yeterlidir.

.Net 3.5 içerisindeki nesne modeline bakıldığında **SyndicationFeed**, **SyndicationItem**, **SyndicationCategory**, **SyndicationPerson**, **SyndicationContent** gibi pek çok **CLR tipinin(Common Language Runtime Type)** yer aldığı görülür. Bu tiplerden belkide en çok kullanılanları **SyndicationFeed** ve **SyndicationItem** sınıflarıdır. Söz konusu sınıflar**System.ServiceModel.Web.dll assembly** içerisinde yer alan **System.ServiceModel.Syndication isim alanında(Namespace)** bulunmaktadır.

Bu kadar teorik bilgiden sonra bir kaç örnek ile konuyu genişletmeye çalışalım. İlk olarak basit bir **Console** uygulaması geliştirecek ve **RSS 2.0/ Atom 1.0** formatlarında içeriklerin nasıl hazırlanabileceğini incelemeye çalışacağız. Bu amaçla **Visual Studio 2008** üzerinden **.Net 3.5** modeline uygun olacak şekilde bir **Console** uygulaması açmamız ve**System.ServiceModel.Web.dll**' ini projeye referans etmemiz yeterlidir.



Bu işlemin ardından kodları aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.ServiceModel.Syndication;
using System.Collections.ObjectModel;
using System.Xml;

namespace SyndicationFormatlama
{
   class Program
   {
      static void Main(string[] args)
      {
         // Bir Feed oluşturulur. Parametreler title, description ve Uri bilgileridir.
         SyndicationFeed feed=new SyndicationFeed("Makaleler","Burak Senyurt
.Net Makaleleri",new Uri("http://www.bsenyurt.com"));
         // Feed yazarı tanımlanır. Yazarlar SyndicationPerson tipi ile temsil edilebilirler.
         feed.Authors.Add(new SyndicationPerson("selim@bsenyurt.com"));
         // Feed için bir kategori tanımalası yapılır. Bu kategori tanımlaması
SyndicationCategory tipi ile temsil edilebilir.
         feed.Categories.Add(new SyndicationCategory(".Net Teknolojileri"));
         // Feed içeriğinin dili belirtilir.
         feed.Language = "TR-TR";
         // Son güncelleme tarihi atanır.
         feed.LastUpdatedTime = DateTime.Now;

         // Feed içerisinde yer alacak öğelerin her biri SyndicationItem tipindendir.
         // SyndicationFeed tipinin Items özelliği bu nesne örneklerini barındırır.
         // Items özelliği Collection<SyndicationItem> tipinden koleksiyonları kullanır.
         // C# 3.0 Object Initializers yardımıyla koleksiyon oluşturulur ve örnek öğeler
eklenir.
         Collection<SyndicationItem> items = new Collection<SyndicationItem>()
```

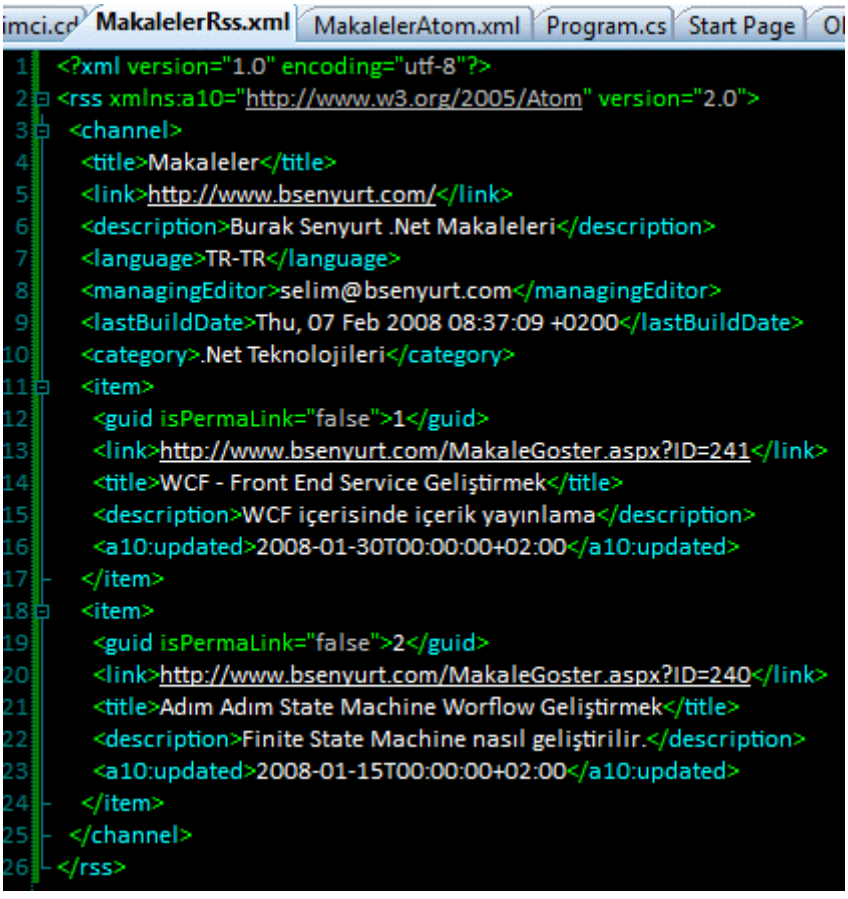
```
                    {
                        // Feed içerisindeki öğeler(Items) SyndicationItem tipi ile temsil
edilirler.
                        // Parametreler title,content,uri,id,lastUpdatedTime
                        new SyndicationItem("WCF - Front End Service
Geliştirmek","WCF içerisinde içerik yayınlama",new
Uri("http://www.bsenyurt.com/MakaleGoster.aspx?ID=241"),"1",new
DateTime(2008,1,30))
                        ,new SyndicationItem("Adım Adım State Machine Worflow
Geliştirmek","Finite State Machine nasıl geliştirilir.",new
Uri("http://www.bsenyurt.com/MakaleGoster.aspx?ID=240"),"2",new
DateTime(2008,1,15))
                    };

        feed.Items = items; // oluşturulan öğelere ait koleksiyon Feed için set edilir.

        // Atom 1.0 notasyonunda formatlama için SyndicationFeed nesne örneğinin
GetAtom10Formatter metodu ile Atom10FeedFormatter nesnesi örneklenir.
        Atom10FeedFormatter atom10Formatter = feed.GetAtom10Formatter();

        // Rss 2.0 notasyonunda formatlama için SyndicationFeed nesne örneğinin
GetRss20Formatter metodu ile Rss20FeedFormatter nesnesi örneklenir.
        Rss20FeedFormatter rss20Formatter = feed.GetRss20Formatter();

        // Atom içeriğinin kaydedileceği Xml dosyası XmlWriter tipi ile oluşturulur
        XmlWriter atom10writer = XmlWriter.Create("MakalelerAtom.xml");
        // Formatter nesnesinin WriteTo metodu ile Atom 1.0 notasyonunda formatlanan
veri içeriği XmlWriter nesnesinin işaret ettiği fiziki dosyaya yazılır.
        atom10Formatter.WriteTo(atom10writer);
        // XmlWriter nesnesi kapatılır
        atom10writer.Close();

        // Atom 1.0 için yapılan formatlama işlemi Rss 2.0 için benzer şekilde yapılır.
        XmlWriter rss20writer = XmlWriter.Create("MakalelerRss.xml");
        rss20Formatter.WriteTo(rss20writer);
        rss20writer.Close();
    }
  }
}
```

Yukarıdaki kod parçasında örnek olarak oluşturulan **Feed** içerikleri **Atom 1.0** ve **RSS 2.0** formatlarına uygun olacak şekilde fiziki **XML** dosyalarına yazdırılmaktadır. **Feed** oluşturulması için **SyndicationFeed** sınıfına ait nesne örnekleri kullanılır. Bununla birlikte içerik paylaşımı ile ilgili ekstra metadata bilgileri söz konusu nesne örneğinin çeşitli özellikleri yardımıyla belirlenebilir. örneğin içeriği paylaşan

yazar **Author** özelliği ile belirlenebilir. Yazar gibi bilgiler içerisinde mail adresi tarzında ek verilerde olabileceğinden **SyndicationPerson** sınıfına ait nesne örnekleri ile **author** elementinin yönetimli kod tarafında ele alınması sağlanabilmektedir. Benzer durum kategori bilgileri içinde geçerlidir. Kategori için **SyndicatioCategory** sınıfından yararlanılmaktadır. **Feed** içerisinde yer alan bilgilendirici **öğeler(Items), SyndicationFeed** sınıfı içerisinde yer alan **Items** özelliği(Property) ile tutulmaktadır. Items özelliği her bir elemanı**SyndicationItem** sınıfından olan **generic Collection** tipi ile ele alınabilir.

Feed ve içeriği oluşturulduktan sonra bu verinin **Atom 1.0** veya **RSS 2.0 f**ormatında üretilmesi için **Atom10FeedFormatter** ve **Rss20FeedFormatter** sınıflarına ait nesne örneklerinden yararlanılmaktadır. Bu nesne örneklerinin üretimi için **SyndicationFeed** sınıfına ait **GetAtom10Formatter** ve **GetRss20Formatter** metodları kullanılır. Veri içerikleri **XML** formatlı olarak yazılabileceklerinden fiziki kaynağı işaret etmek adına **XmlWriter** tipinden yararlanılır. Söz konusu uygulama çalıştırıldığında **MakalelerRss.xml** ve **MakalelerAtom.xml** dosyalarının içeriklerinin aşağıdaki gibi oluştuğu görülür.

MakalelerRss.xml içeriği;

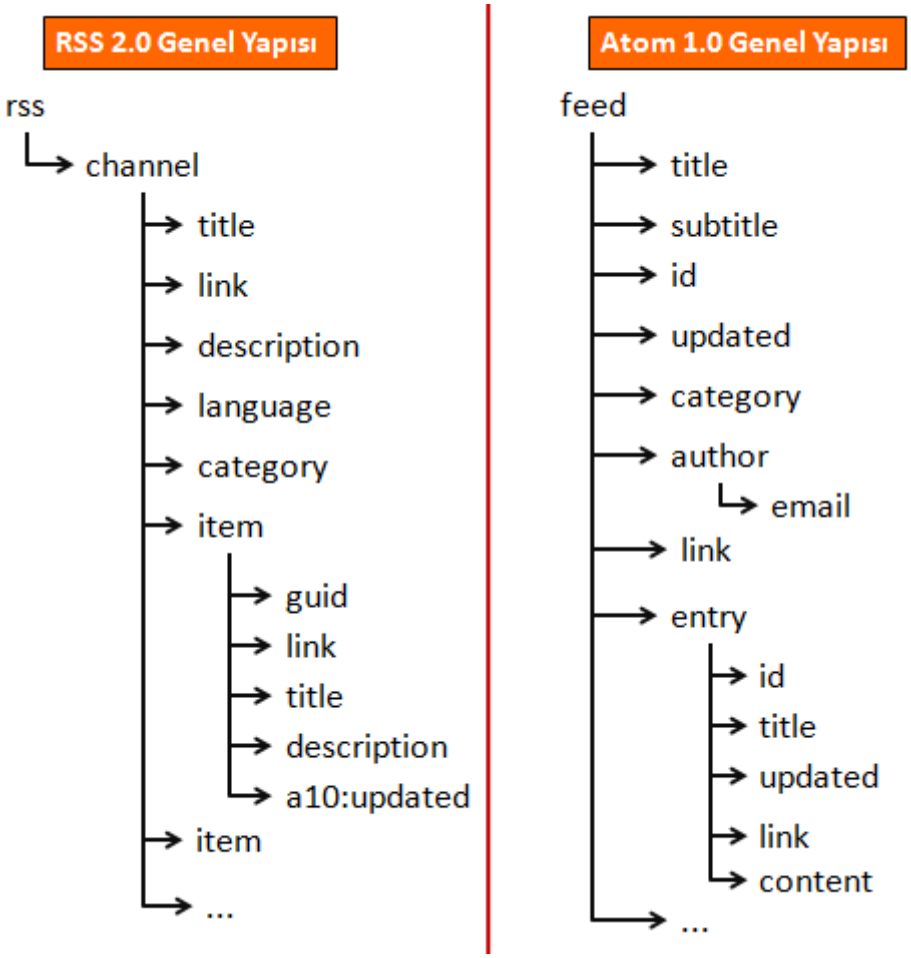


MakalelerAtom.xml içeriği;

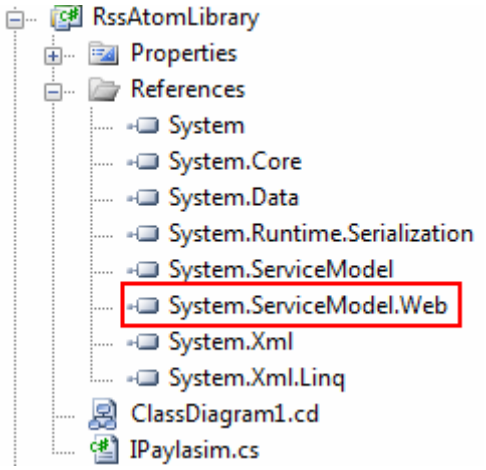
dimci.cd | MakalelerRss.xml | **MakalelerAtom.xml** | Program.cs | Start Page | Object Browser

```xml
<?xml version="1.0" encoding="utf-8"?>
<feed xml:lang="TR-TR" xmlns="http://www.w3.org/2005/Atom">
  <title type="text">Makaleler</title>
  <subtitle type="text">Burak Senyurt .Net Makaleleri</subtitle>
  <id>uuid:775d92c6-31db-4700-b186-ebec1ee7846c;id=1</id>
  <updated>2008-02-07T08:37:09+02:00</updated>
  <category term=".Net Teknolojileri" />
  <author>
    <email>selim@bsenyurt.com</email>
  </author>
  <link rel="alternate" href="http://www.bsenyurt.com/" />
  <entry>
    <id>1</id>
    <title type="text">WCF - Front End Service Geliştirmek</title>
    <updated>2008-01-30T00:00:00+02:00</updated>
    <link rel="alternate" href="http://www.bsenyurt.com/MakaleGoster.aspx?ID=241" />
    <content type="text">WCF içerisinde içerik yayınlama</content>
  </entry>
  <entry>
    <id>2</id>
    <title type="text">Adım Adım State Machine Worflow Geliştirmek</title>
    <updated>2008-01-15T00:00:00+02:00</updated>
    <link rel="alternate" href="http://www.bsenyurt.com/MakaleGoster.aspx?ID=240" />
    <content type="text">Finite State Machine nasıl geliştirilir.</content>
  </entry>
</feed>
```

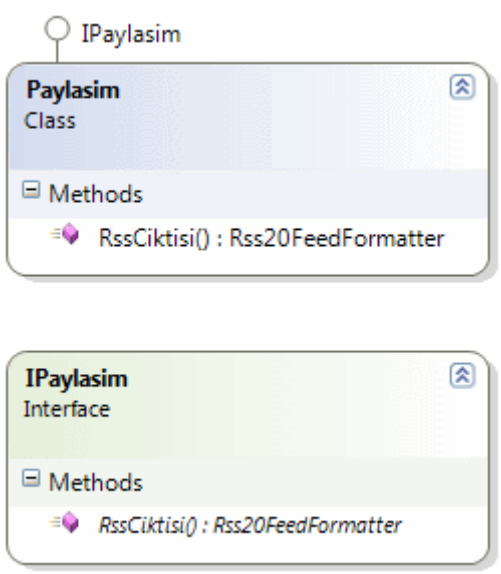
Her iki yapının içeriğini aşağıdaki grafik ile daha kolay bir şekilde de karşılaştırabiliriz.



Görüldüğü üzere arada bazı farklılıklar mevcuttur. Ancak her iki formatta genel standarttır. Gelelim bu içerikleri bir **WCF servisi** üzerinden nasıl yayınlayabileceğimize. öncelikli olarak**RSS** formatında yayınlama yapılmasına izin veren bir **WCF Servis Kütüphanesi(WCF Service Library)** geliştiriyor olacağız. Bu amaçla **Visual Studio 2008** ortamında **.Net Framework 3.5**şablonları içerisinde yer alan **WCF Service Library** proje tipi seçilerek işe başlanabilir. Elbette **Synidaction** için gerekli tipleri kullanacağımızdan servis kütüphanesinin**System.ServiceModel.Web.dll assembly**' ının referans etmesi gerekmektedir.



Servis kütüphanesi içerisinde yer alacak olan **sözleşme(Contract)** ve uygulayıcı sınıfa ait sınıf diyagramı(Class Diagram) ile içerikleri ise aşağıdaki gibidir.



IPaylasim **arayüzünün(Interface)** içeriği;

```
using System;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Collections.Generic;
using System.ServiceModel.Syndication;

namespace RssAtomLibrary
{
   [ServiceContract]
   public interface IPaylasim
   {
      [OperationContract]
      [WebGet]
      Rss20FeedFormatter RssCiktisi();
   }
}
```

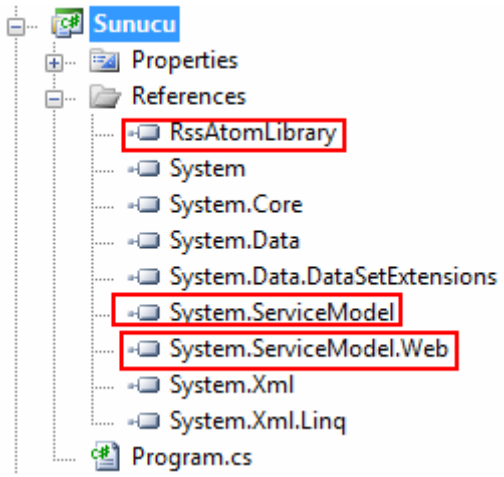
Arayüz(Interface) tanımlamasında belkide dikkati çeken en önemli noktalardan biriside **WebGet** isimli **niteliğin(attribute)** kullanıllmasıdır. Bu nitelik söz konusu servise **HTTP Get** metoduna göre talepte(Request) bulunabileceğini göstermektedir.

Paylasim **sınıfının(Class)** içeriği;

```
using System;
using System.Linq;
using System.ServiceModel;
using System.ServiceModel.Web;
using System.Collections.Generic;
using System.ServiceModel.Syndication;

namespace RssAtomLibrary
{

   public class Paylasim : IPaylasim
   {
      #region IPaylasim Members

      public Rss20FeedFormatter RssCiktisi()
      {
         // öncelikle bir Feed oluşturulur
         SyndicationFeed feed = new SyndicationFeed();

         // İçerik paylaşımı yapan yazarlar SyndicationPerson sınıfı yardımıyla tanımlanırlar ve Authors koleksiyonuna dahil edilirler.
         feed.Authors.Add(new SyndicationPerson("selim@bsenyurt.com", "Burak Selim Senyurt", "http://www.bsenyurt.com"));
         feed.Authors.Add(new SyndicationPerson("kariim@bsenyurt.com", "Kariim Abdul Cabbar", "http://www.bsenyurt.com"));

         // İçeriğin ilgili olduğu kategoriler SyndicationCategory sınıf yardımıyla örneklenir ve Categories koleksiyonuna dahil edilir.
         feed.Categories.Add(new SyndicationCategory(".Net"));
         feed.Categories.Add(new SyndicationCategory("C#"));

         // İçerik il ilgili açıklama TextSyndicationContent sınıfı yardımıyla eklenir
         feed.Description = new TextSyndicationContent(".Net ve C# ağırılıklı makaleler", TextSyndicationContentKind.Html);
         feed.Language = "Tr-Tr"; // İçerik dili belirtilir
         feed.LastUpdatedTime = DateTime.Now; // Son güncelleme tarihi belirtilir
         feed.Title=new TextSyndicationContent(".Net ile ilgili Herşey"); // İçeriğin başlığı TextSyndicationContent sınıfı yardımıyla belirtilir.

         // Ogeler eklenir
         List<SyndicationItem> items = new List<SyndicationItem>()
               {
                  // Feed içerisindeki öğeler(Items) SyndicationItem tipi ile temsil edilirler.
                  // Parametreler title,content,uri,id,lastUpdatedTime
                  new SyndicationItem("WCF - Front End Service Geliştirmek","WCF
```

```
içerisinde içerik yayınlama",new
Uri("http://www.bsenyurt.com/MakaleGoster.aspx?ID=241"),"1",new
DateTime(2008,1,30))
                    ,new SyndicationItem("Adım Adım State Machine Worflow
Geliştirmek","Finite State Machine nasıl geliştirilir.",new
Uri("http://www.bsenyurt.com/MakaleGoster.aspx?ID=240"),"2",new
DateTime(2008,1,15))
                };

            feed.Items = items;
          return new Rss20FeedFormatter(feed);
      }

      #endregion
   }
}
```

Paylasim sınıfının **RssCiktisi** isimli metodu **Rss20FeedFormatter** tipinden bir nesne örneğini döndürmektedir. Bu nesne örneklenirken parametre olarak **SyndicationFeed** tipinden oluşturulan nesne örneği parametre olarak verilmektedir. **Servis kütüphanesinin(Service Library)** bu şekilde hazırlanmasının ardından artık **Host** uygulamanın yazılmasına geçilebilir. **Host**uygulama servis kütüphanesini, **System.ServiceMode.dll** ve yine **System.ServiceModel.Web.dll assembly**' larını referans etmelidir.

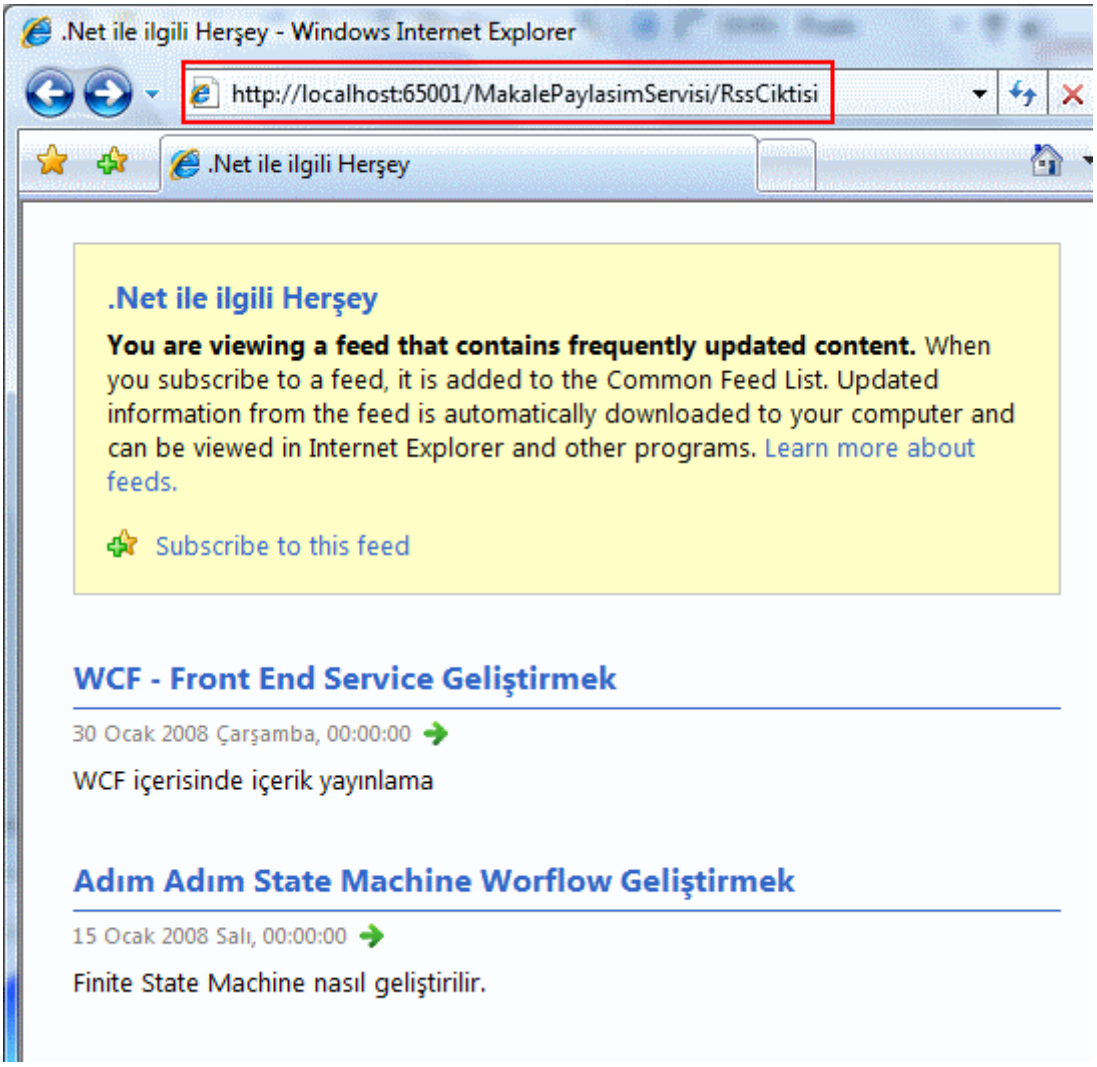


Host uygulama basit olarak bir **Console** programı şeklinde tasarlanabilir.

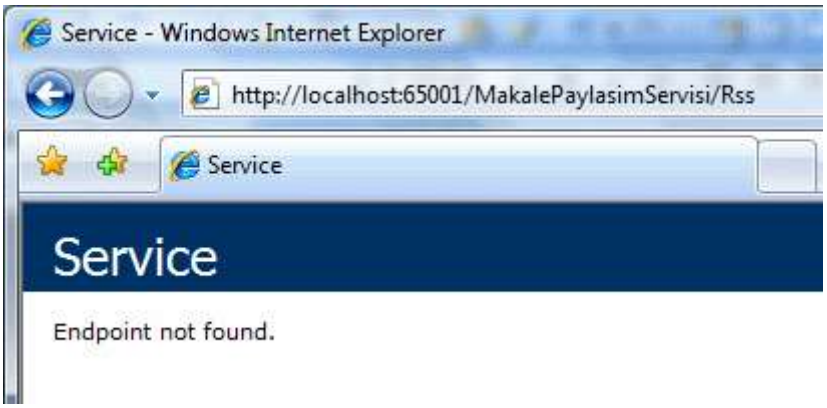
```
using System;
using System.ServiceModel;
using System.ServiceModel.Web;
using RssAtomLibrary;
using System.ServiceModel.Syndication;
using System.Xml;
```

```
namespace Sunucu
{
   class Program
   {
      static void Main(string[] args)
      {
         WebServiceHost host = new WebServiceHost(typeof(Paylasim), new
Uri("http://localhost:65001/MakalePaylasimServisi"));

         host.Open();

         Console.WriteLine("Servis durumu {0} ", host.State);
         Console.WriteLine("Host dinlemede...çıkmak için bir tuşa basınız...");
         Console.ReadLine();

         host.Close();
      }
   }
}
```

Host uygulamada ilk dikkati çeken noktalardan birisi **WebServiceHost** sınıfına ait bir nesne örneğinin kullanılmasıdır. Bilindiği gibi normal şartlarda **ServiceHost** sınıfından yararlanılmaktadır. **WebServiceHost** nesnesi örneklenirken ilk parametre olarak **servis sözleşmesini(Service Contract)** uygulayan sınıfın tipini almaktadır. Sonraki parametrede ise servis adresi belirlenmektedir. Host' un açılması için yine **Open** metoduna başvurulmaktadır. Benzer şekilde kapatma işlemi içinde **Close** fonksiyonundan yararlanılır. Bu işlemlerin ardından servis uygulaması çalıştırılabilir. Servis uygulaması çalışıyorken herhangibir tarayıcı
penceresinden **http://localhost:65001/MakalePaylasimServisi/RssCiktisi** adresi talep edilirse aşağıdaki ekran çıktısında yer alan görüntü elde edilecektir.

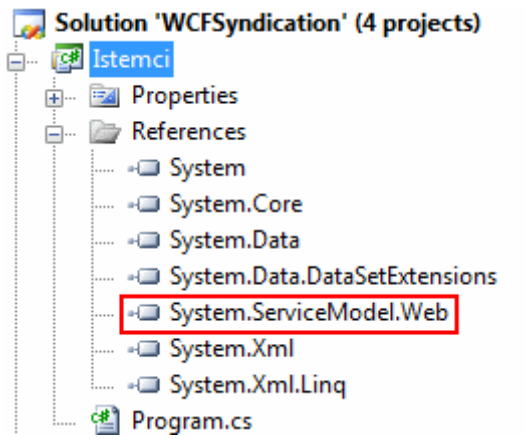


Görüldüğü gibi servisteki **RssCiktisi** isimli metodun sonucu **HTTP** üzerinden elde edilebilmektedir. Burada dikkat edilmesi gereken önemli bir nokta vardır. **URL** adresinin sonu, **RSS** çıktısı veren metodun adı ile aynıdır. Eğer farklı bir adres girilirse **Service EndPoint** bulunamayacak ve aşağıdakine benzer bir ekran ile karşılaşılacaktır.



Bu durum istemci tarafınada bir **çalışam zamanı istisnası(Runtime Exception)** olarak yansıyacaktır. Peki istemci kod tarafından bu tip bir içeriği nasıl ele alabilir? Bu amaçla

basit bir istemci uygulamayı **Console** projesi olacak şekilde geliştirip devam edelim. Istemci tarafında standart **XmlReader** veya **XmlDocument** nesneleri yardımıyla **HTTP Get** ile talep edilen servis içeriği çekilebilir. Ancak **.Net Framework 3.5** içerisinde gelen **Syndication** tipleri yardımıyla bu işlemi gerçekleştirmek çok daha kolaydır. Bu nedenle istemci tarafındaki uygulamaya**System.ServiceModel.Web.dll assembly**' ının referans edilmesi gerekmektedir.

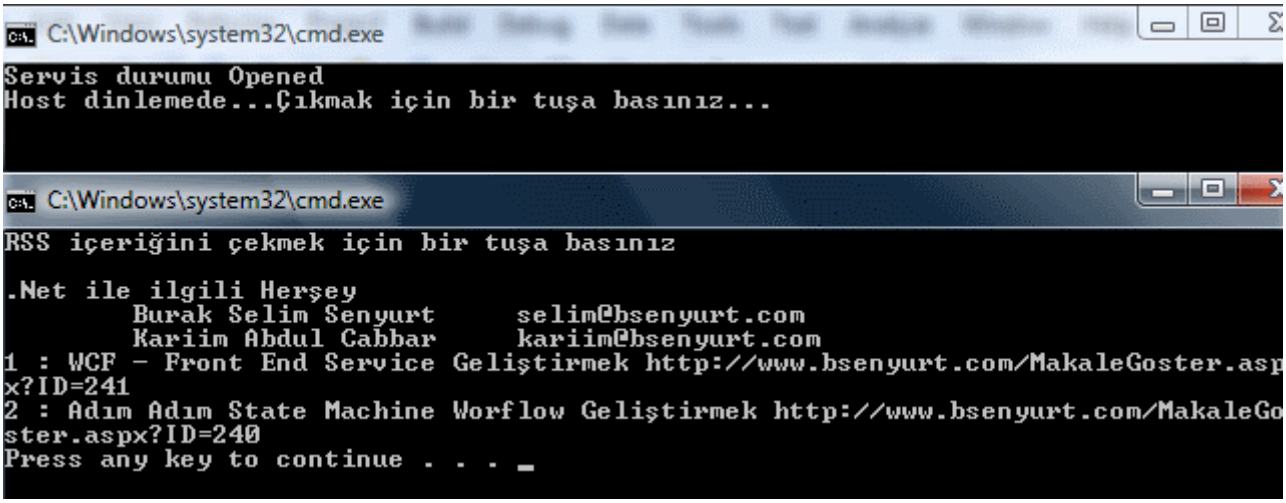


Dikkat edileceği üzere herhangibir şekilde **servis referansı** eklenmemiş yada **proxy** sınıfı oluşturulmamıştır. Nitekim servise gönderilecek olan **talep(request)** aslında **EndPoint**davranışı sergileyen bir metoda doğru **HTTP Get** üzerinden sağlanacaktır. Bu sebeplerden istemci tarafındaki kodlar çok basit olarak aşağıdaki gibi geliştirilebilirler.

```
using System;
using System.Xml;
using System.ServiceModel.Syndication;

namespace Istemci
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("RSS içeriğini çekmek için bir tuşa basınız");
         Console.ReadLine();

         XmlReader reader =
XmlReader.Create("http://localhost:65001/MakalePaylasimServisi/RssCiktisi");
         SyndicationFeed feed = SyndicationFeed.Load(reader);

         Console.WriteLine(feed.Title.Text);
         foreach (SyndicationPerson yazar in feed.Authors)
         {
            Console.WriteLine("\t{0} \t {1}",yazar.Name,yazar.Email);
```

```
        }

        foreach (SyndicationItem item in feed.Items)
        {
           Console.WriteLine(String.Format("{0} : {1}
{2}",item.Id,item.Title.Text,item.Links[0].Uri));
        }
     }
   }
}
```

İlk olarak **XmlReader** sınıfından yararlanılarak **http://localhost:65001/MakalePaylasimServisi/RssCiktisi** adresinden talepte bulunulmaktadır. Bu talep sonrası elde edilen **XML** çıktısının **SyndicationFeed** sınıfı tarafından daha kolay bir şekilde ele alınabilmesini sağlamak amacıyla **static Load** metoduna **XmlReader** nesne örneği parametre olarak verilir. Bu işlemin ardından **Feed** ile ilgili olarak **başlık(title)** bilgisi elde edilir. Bununla birlikte **SyndicationPerson** sınıfı ve **Authors** özelliklerinden yararlanılarak yazarlara ait **isim(Name)** ve **elektronik posta(e mail)**bilgileri çekilir. Son olarakta **Items** koleksiyonu dolaşılarak var olan tüm öğeler **SyndicationItem** sınıfına ait nesne örnekleri yardımıyla ele alınır. örnek olarak öğelerin **Id**, **başlık(Title)** ve **url** bilgileri verilir. İstemcilerin talepte bulunabilmesi ve **Feed** içeriklerini çekebilmesi için çok doğal olarak servis tarafınının çalışıyor olması gerekir. Eğer bu şart sağlanırsa aşağıdakine benzer bir ekran görüntüsü ile karşılaşılacaktır.



Görüldüğü gibi **Feed** içeriği istemci tarafına başarılı bir şekilde aktarılmıştır.

İçerik paylaşımı(Syndication) ile ilişkili bir diğer önemli konuda bir servisin yeri geldiğinde **RSS 2.0** formatında yeri geldiğinde de **Atom 1.0** formatında içerik paylaşımına izin vermesidir. Bu amaçla servis tarafında bazı değişiklier yapmak gerekecektir. Bu amaçla az önce geliştirilen servis kütüphanesindeki(WCF Service Library) arayüzü(Interface) aşağıdaki gibi değiştirerek işe başlayalım.

```
[ServiceContract]
[ServiceKnownType(typeof(Rss20FeedFormatter))]
[ServiceKnownType(typeof(Atom10FeedFormatter))]
public interface IPaylasim
{
   [OperationContract]
   [WebGet]
   Rss20FeedFormatter RssCiktisi();

   [OperationContract]
   [WebGet(UriTemplate="IcerikOzeti?icerikTipi={icerikTipi}")]
   SyndicationFeedFormatter IcerikOzeti(string icerikTipi);
}
```

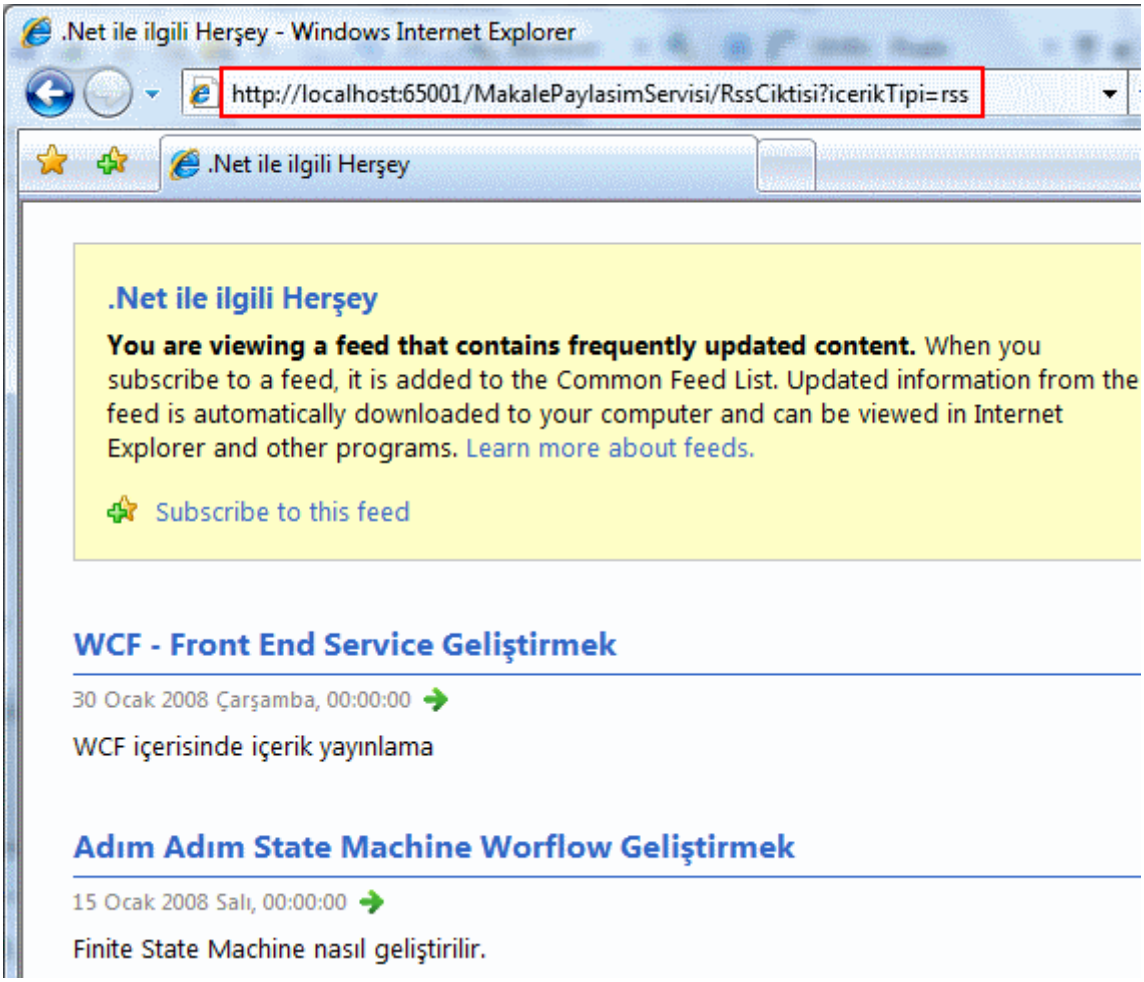
öncelikli olarak **ServiceKnownType** niteliği(attribute) ile servisin **Rss20** ve **Atom10** formatlarından nesne örnekleri döndürebileceği belirtilmektedir. Nitekim içerik paylaşım modelinde ilgili servis metodlarının **SyndicationFeedFormatter** sınıfı ile taşınabilecek tipler döndürmeleri şarttır. Diğer taraftan **IcerikOzeti** metodunda kullanılan **WebGet** niteliğinde bir **querystring**bildirimi yapılmaktadır. Bu querystring bildirimi metoda **HTTP Get** üzerinden gelecek olan **parametre** bilgisinin şablonunu tanımlamaktadır. **Arayüz sözleşmesini(Interface Contract)** uygulayan sınıf içerisindeki metodun ise aşağıdaki gibi geliştirilmesi yeterlidir.

```
public SyndicationFeedFormatter IcerikOzeti(string icerikTipi)
{
   SyndicationFeed feed = new SyndicationFeed();

   feed.Authors.Add(new SyndicationPerson("selim@bsenyurt.com", "Burak Selim
Senyurt", "http://www.bsenyurt.com"));
   feed.Authors.Add(new SyndicationPerson("kariim@bsenyurt.com", "Kariim Abdul
Cabbar", "http://www.bsenyurt.com"));

   feed.Categories.Add(new SyndicationCategory(".Net"));
   feed.Categories.Add(new SyndicationCategory("C#"));

   feed.Description = new TextSyndicationContent(".Net ve C# ağırılıklı makaleler",
TextSyndicationContentKind.Html);
   feed.Language = "Tr-Tr";
   feed.LastUpdatedTime = DateTime.Now;
   feed.Title = new TextSyndicationContent(".Net ile ilgili Herşey");

   List<SyndicationItem> items = new List<SyndicationItem>()
      {
         new SyndicationItem("WCF - Front End Service Geliştirmek","WCF içerisinde
```

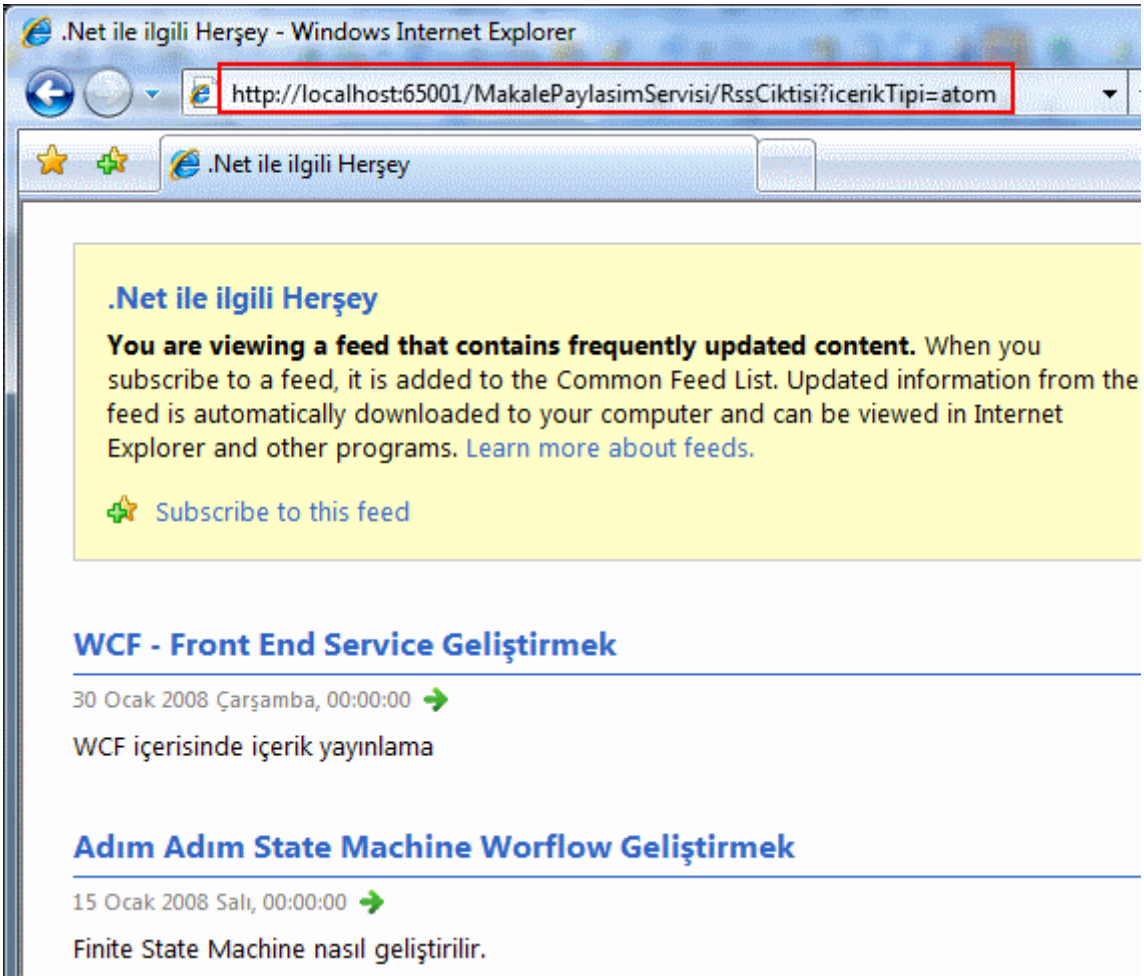
```
içerik yayınlama",new
Uri("http://www.bsenyurt.com/MakaleGoster.aspx?ID=241"),"1",new
DateTime(2008,1,30))
          ,new SyndicationItem("Adım Adım State Machine Worflow Geliştirmek","Finite
State Machine nasıl geliştirilir.",new
Uri("http://www.bsenyurt.com/MakaleGoster.aspx?ID=240"),"2",new
DateTime(2008,1,15))
       };
   feed.Items = items;

   if (icerikTipi == "atom")
      return new Atom10FeedFormatter(feed);
   else if (icerikTipi == "rss")
      return new Rss20FeedFormatter(feed);
   else
      return null;
}
```

Bu metod içerisinde dikkat edilmesi gereken en önemli nokta icerikTipi değerine göre **Atom10FeedFormatter** veya **Rss20FeedFormatter** tipinden nesne örnekleri döndürülmesidir. Bu sebepten dolayıda metodun dönüş tipi **SyndicationFeedFormatter** tipindendir. Artık istemciler servise talepte bulunurlarken **rss** veya **atom** formatında içerik isteyebilirler. Aşağıdaki ekran çıktılarında bu durum açık bir şekilde görülmektedir.

Rss talebi;


.Net ile ilgili Herşey - Windows Internet Explorer
http://localhost:65001/MakalePaylasimServisi/RssCiktisi?icerikTipi=rss
.Net ile ilgili Herşey
.Net ile ilgili Herşey
You are viewing a feed that contains frequently updated content. When you subscribe to a feed, it is added to the Common Feed List. Updated information from the feed is automatically downloaded to your computer and can be viewed in Internet Explorer and other programs. Learn more about feeds.
Subscribe to this feed
WCF - Front End Service Geliştirmek
30 Ocak 2008 Çarşamba, 00:00:00
WCF içerisinde içerik yayınlama
Adım Adım State Machine Worflow Geliştirmek
15 Ocak 2008 Salı, 00:00:00
Finite State Machine nasıl geliştirilir.

Atom talebi;



Böylece geldik bir makalemizin daha sonuna. Bu makalemizde basit olarak **.Net Framework 3.5** ile gelen tipler sayesinde bir **WCF(Windows Communication Foundation)** servisi üzerinden **RSS 2.0** veya **Atom 1.0**formatında içerik paylaşımının nasıl yapılabileceğini incelemeye çalıştık. İlerleyen makalelerimizde WCF in **.Net Framework 3.5** ile gelen yeniliklerini incelemeye devam edeceğiz. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

WCFSyndication.rar (78,48 kb)

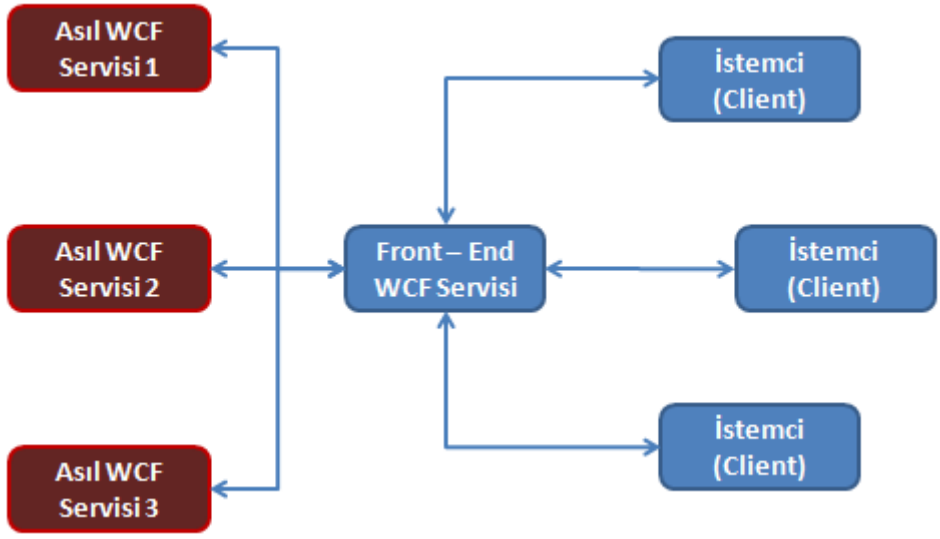
WCF - Front-End Service Geliştirmek (2008-01-30T05:17:00)

wcf,

Windows Communication Foundation(WCF) mimarisinde belkide en kritik unsurlardan birisi **EndPoint** kavramıdır. EndPoint, **Servis Yönelimli Mimari(Service Oriented Architecture - SOA)** uygulamaları geliştirmek için kullanılan WCF modelinde, istemciler(Clients) ile servis(Service) arasındaki haberleşmede yer alan kritik bir parçadır. WCF' in temellerini incelediğimiz daha önceki yazılarımızda, EndPoint kavramının aslında WCF mimarisinin **ABC**' si olduğundan bahsetmiştik. ABC bilindiği üzere **adres(Address), bağlayıcı(Binding)** ve **sözleşme(Contract)** bilgilerinden oluşmaktadır. Buna göre bir

EndPoint yardımıyla, servisin istemcilere hangi adresten, hangi protokolle, hangi kurallara göre neyi sunacağı bilgisi aktarılabilir. Bununla birlikte, EndPoint' ler istemci tarafından gelecek olan taleplerin karşılanmasında da büyük öneme sahiptir.

Bir WCF servisi, kendi üzerinde birden fazla **EndPoint** bilgisi taşıyabilir. üstelik bu EndPoint' ler aynı **sözleşme(Contract)** veya farklı sözleşmeler(Contract) içinde tanımlanmış olabilirler. Bu noktada WCF servisi aldığı mesajı hangi EndPoint bileşenine ileteceğinede karar vermektedir. çok doğal olarak geliştiriciler bu karar aşamasına müdahale edebilir ve EndPoint yönlendirmelerini programlayabilirler. İşte bu da **Front-End Service** adı verilen WCF servislerinin geliştirilebilmesine neden olmaktadır ki makalemizin konusuda budur. Bazı durumlarda bir WCF servisi, istemcilerden(Clients) gelecek olan mesajları asıl servislere yönlendirmek için kullanılabilir. Aşağıdaki şekilde bu durum analiz edilmeye çalışılmaktadır.



Bu grafiğe göre istemcilerin **talepleri(Requests) Front-End Service** tarafından karşılanmakta, sonrasında ise bu talepler(Request) ilgili olan asıl WCF servislerine yönlendirilmektedir. Hatta yönlendirme işlemi Front-End Service' in arkasında duran asıl WCF servisi içerisindeki farklı EndPoint noktalarına doğruda olabilir. Bu tarz bir yönlendirme için pek çok sebep vardır. özellikle istemcilerin kullanmak istedikleri servislere doğrudan erişemediği durumlar göz önüne alınabilir. örneğin arka tarafta duran WCF servisleri ile i**stemciler(Clients)** farklı ve birbirlerini görmeye**ağlarda(Network)** bulunabilirler. Diğer yandan, **Front-End Service**' lerin bazı avantajlarıda vardır. örneğin **yük dengelemesini (Load Balancing)** daha iyi yapabilirler. Bu performans ve dengeli yönetim açısından önemli bir unsurdur.

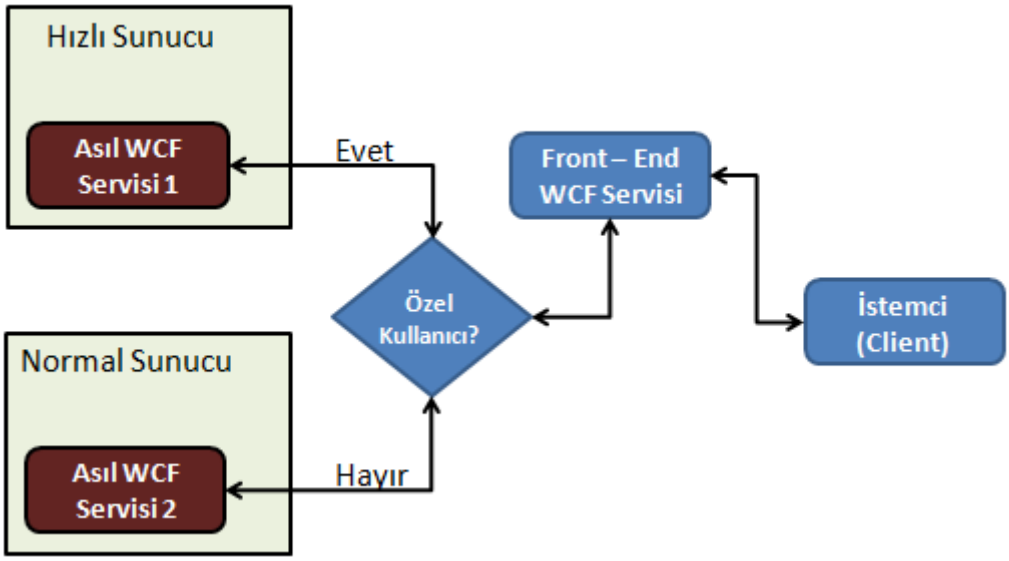
NOT : *Load Balancing modelinde, Front-End Service genellikle istemcilere(Clients) tek bir adres üzerinden hizmet vermektedir. İstemcilerden gelen talepler arka servislere eşit şekilde dağıtılmaktadır. Bir başka deyişle istemciler hep aynı adrese talepte bulunurlarken, Front-End Service arka tarafta duran farklı port numaralarına sahip EndPoint noktalarına yükü eşit şekilde yaymaktadır. Bu sayede arka tarafta daha dengeli çalışan bir servis*

topluluğu tasarlanması mümkün olmaktadır.Tabi bunun için yazılan Front-End Service içerisinde özel kodlama yapılması gerekmektedir.

Ayrıca, kötü niyetli mesajların sızmasının engellemesi içinde tek bir merkez olarak kullanılabilirler. Elbetteki arka tarafta yönlendirilme yapılacak servis ve EndPoint sayısı çok fazla olabileceğinden Front-End Service yönetimi biraz daha zordur.

***NOT : Front-End WCF Service**' ler istemciden gelen **talepleri(Request)** değerlendirerek uygun olan gerçek arka **WCF Service**' lerine yönlendirirler.*

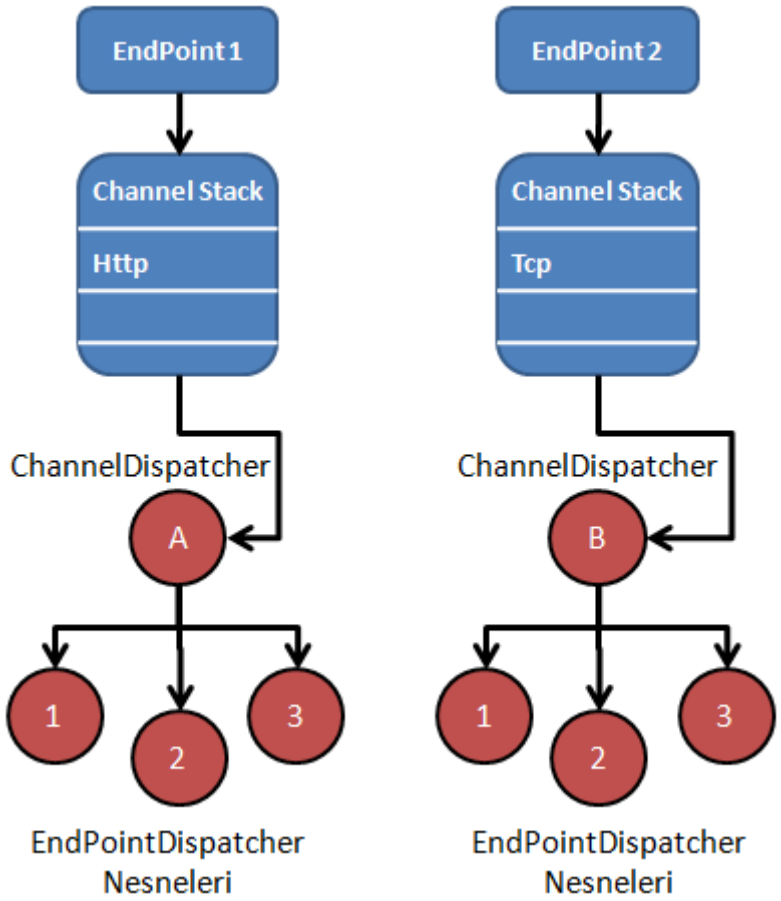
Front-End Service' ler çoğunlukla iki ana kategoride ele alınmaktadır. **Adres tabanlı yönlendirme(Address-Based Routing)** yapanlar ve **içerik tabanlı yönlendirme(Content-Based Routing)**yapanlar. Adres tabanlı yönlendirme sisteminde, istemcilerin hangi EndPoint noktalarına talep (Request) gönderdikleri önemlidir. Ancak içerik tabanlı yönlendirme sisteminde istemcilerin talep olarak gönderikleri mesaj içeriklerine bakılır. Mesaj içeriklerinden örneğin kullanıcıya ait **kimlik bilgileri(Identity Informations)**, **transaction id** değerleri vs... elde edilebilir. Bunların durumuna göre uygun olan WCF servislerine yönlendirme işlevini **Front-End Service** üstlenir. örneğin bir servise talepte bulunan kullanıcıların iki farklı profilde olduklarını göz önüne alalım. Aşağıdaki şekilde bu durum gösterilmeye çalışılmaktadır.



Burada özel kullanıcılar taleplerine hizmet alırlarken, Front-End Service tarafından daha hızlı bir sistem üzerinde konuşlandırılmış bir WCF servisine yönlendirilmektedirler. Diğer taraftan normal kullanıcılar için bu tip bir yönlendirme daha farklı yapılmaktadır. İşte bu tam anlamıyla Front-End Service' e gelen kullanıcıların **kimlikleri(Identity)** ile alakalıdır ve**içerik tabanlı yönlendirme(Content Based Routing)** sisteminin bir örneğidir. Nitekim kimlik bilgileri çoğunlukla mesaj içeriklerinden elde edilebilmektedir.

WCF mimarisi içerisinde **Front-End Service**' lerin nasıl yazılacağını incelemeden önce, bir WCF servisinin gelen mesajları kabaca nasıl ele aldığını anlamakta yarar vardır.

özellikle kod tarafında gelen taleplerin ele alınmasında önemli rol oynayan **CLR Tipleri(Common Language Runtime Types)** bulunmaktadır. Herşeyden önce **WCF çalışma zamanı motoru(WCF Runtime Engine)** gelen talepleri değerlendirmek için **kanal yığınlarını(Channel Stacks)** kullanmaktadır. Bu kanal yığınları **ChannelDispatcher** ve **EndPointDispatcher** adı verilen tiplere ait nesne örnekleri ile sıkı bir ilişkidedir. En basit anlamda **ChannelDispatcher** nesne örnekleri, gelen taleplerin doğru **EndPointDispatcher** bileşenlerine aktarılmasından sorumludur. Bu anlamda bir ChannelDispatcher nesnesi, birden fazla EndPointDispatcher bileşenini ele alabilmektedir. **EndPointDispatcher** nesnesleri, gelen mesajları çözümlemek ve servis içerisindeki uygun yerlere iletmekten sorumludur. EndPointDispatcher nesneleri çoğunlukla servis içerisinden talepte bulunulan fonksiyonelliğe ait metod çağrılarını gerçekleştirmektedir. Bu son derece basit ve yüzeysel bir bakış açısıdır. Nitekim bu işlemler sırasında çok sayıda ek yardımcı **CLR(Common Language Runtime)** nesneside devreye girmektedir. Aslında çalışma zamanındaki durum az çok aşağıdaki şekilde görüldüğü gibidir.

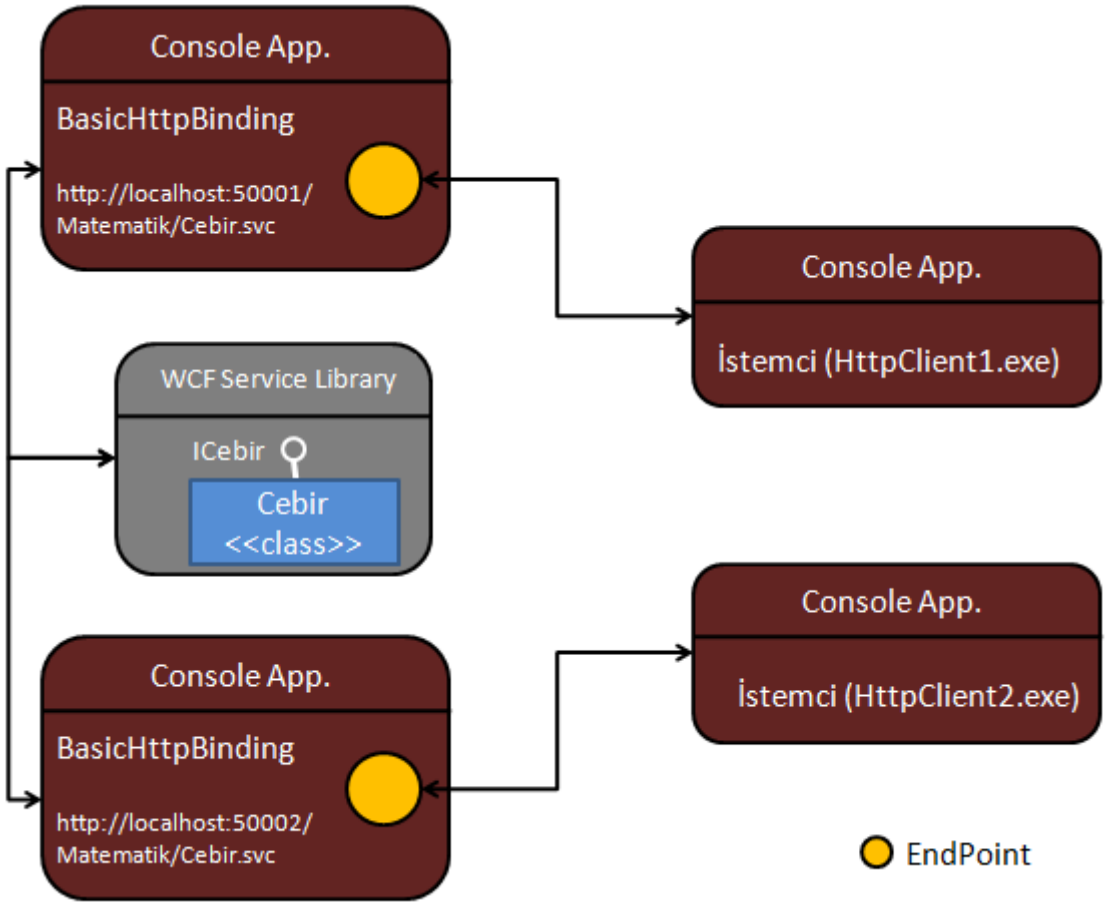


Görüldüğü gibi istemcilerden EndPoint noktalarına doğru gelen mesajlar **kanal yığınları(Channel Stack)** üzerinden **ChannelDispatcher** nesne örneklerine ulaşmaktadır. Sonrasında ise talepler(Requests) uygun olan EndPointDispatcher bileşenleri tarafından ele alınmaktadır. Bu noktada dikkat edilmesi gereken bir husus vardır. Gelen mesaj herhangibir EndPointDispatcher tarafından

karşılanmassa **ServiceHost** nesnesinin **UnknownMessageReceived** olayı tetiklenir. çok doğal olarak bu olayın kontrolü geliştirici(Developer) tarafından yapılabilir.

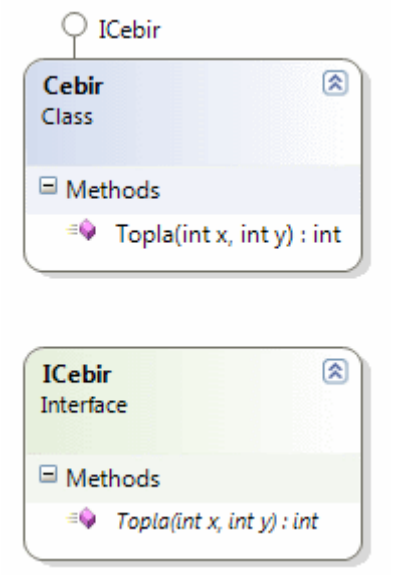
Front-End Service' ler içerisinde, **istemciden(Clients)** gelen talepler gerçek WCF servislerine yönlendirilmeden önce pek çok işlemde gerçekleştirilebilmektedir. örneğin kimlik doğrulama yada arka servislere parametre aktarma gibi işlemler söz konusu olabilir. Diğer yandan **Front-End Service**' lerin tasarlanması sırasında dikkat edilmesi gereken bazı noktalar vardır. Yönlendirici servisin, hedef servis sözleşmelerini uyguluyor olma zorunluluğu nedeni ile yönetilebilirlik zorlaşmaktadır. Buna bağlı olarak **serileştirilebilir(Serializable)** verilere ait sözleşmelerinde yönlendirici servis tarafından bilinme zorunluluğu söz konusudurki bu da yönetilebilirliği(Management) zorlaştırmaktadır.

Bu kısa bilgilerden sonra adım adım bir **Front-End WCF** Servisinin nasıl oluşturulacağı incelenmeye başlanabilir. öncelikli olarak iki adet servis geliştirilecek ve bu servisler birer **EndPoint** noktası barındıracak şekilde tasarlanacaktır. Bu noktada Front-End Service, gelen **talepleri(Request)** doğrudan arka taraftaki uygun servis ve EndPoint noktalarına yönlendirmekle görevli olacaktır. Senaryo kabaca aşağıdaki şekilde görüldüğü gibidir.



Yönlendirme işlemlerini kolay bir şekilde anlayabilmek için basit olarak **BasicHttpBinding bağlayıcı tipinden(Binding Type)** yararlanılmaktadır. **Back-End** servisler iki adettir ve her biri farklı**port** numaraları üzerinden **HTTP** bazlı olacak

şekilde yayınlama yapmaktadır. Her iki **Back-End Service** uygulamasıda **Console** olacak şekilde tasarlanmaktadır. Elbetteki gerçek hayat senaryolarında bu servisler **IIS(Internet Information Service)** üzerinden yada bir **Windows Servisi** içerisinde gömülü olacak şekildede çalışabilirler. Şu durumda her iki istemcide bu farklı**HTTP** servislerine talepte bulunmaktadır. Bizim amacımız bir **Front-End Service** yazmak ve üzerinden **Load-Balancing** yaparak istemcilerden gelecek olan talep yükünü servislere eşit şekilde dağıtmaya çalışmaktır. Front-End Service geliştirilmeye başlamadan önce ilk olarak **servis sözleşmesinin(Service Contract)** yer aldığı **WCF sınıf kütüphanesini(WCF Service Library)** tasarlayarak işe başlanmalıdır. Bu kütüphanenin içeriği konunun anlaşılır olması açısından mümkün olduğu kadar basit tutulmuştur.



ICebir **arayüzüne(Interface)** ait kod içeriği aşağıdaki gibidir.

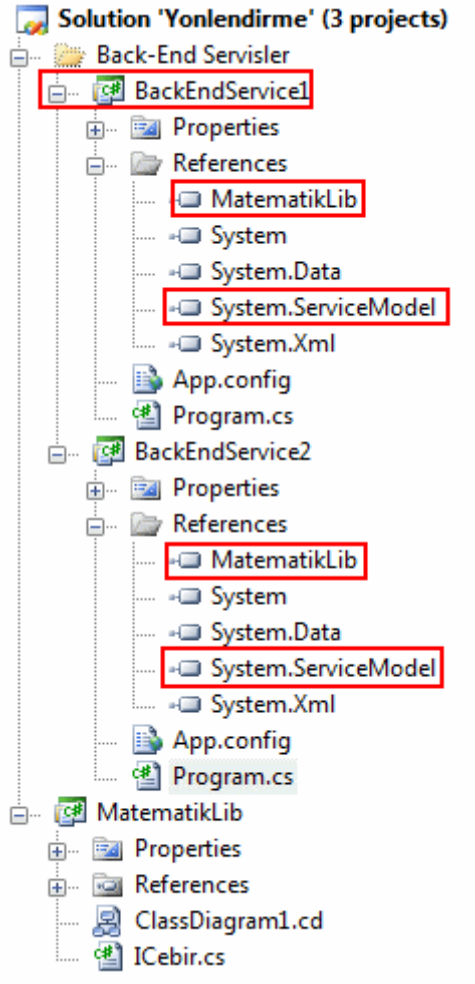
```
[ServiceContract]
public interface ICebir
{
   [OperationContract]
   int Topla(int x, int y);
}
```

Cebir **sınıfına(Class)** ait kod içeriği ise aşağıdaki gibidir.

```
[ServiceBehavior(InstanceContextMode= InstanceContextMode.PerCall)]
public class Cebir : ICebir
{
   #region ICebir Members

   public int Topla(int x, int y)
   {
      return x + y;
```

```
    }

    #endregion
}
```

Bu işlemin ardından **Back-End Service**' lerin tasarlanmasına başlanabilir. Her iki serviste **Console** uygulaması üzerinden **host** edilmektedir. Aralarındaki tek fark yayınlama adreslerindeki **port**numaralarının farklı olmasıdır. Sembolik olarak bu port numaraları 50001 ve 50002 olarak set edilmektedir. Söz konusu Console uygulamaları çok doğal olarak MatematikLib isimli **WCF servis kütüphanesini(WCF Service Library)** ve **System**.**ServiceModel**.**dll assembly**' ını referans etmelidir.



Bununla birlikte konfigurasyon dosyasının**(App.config)** içeriği aşağıdaki gibidir.

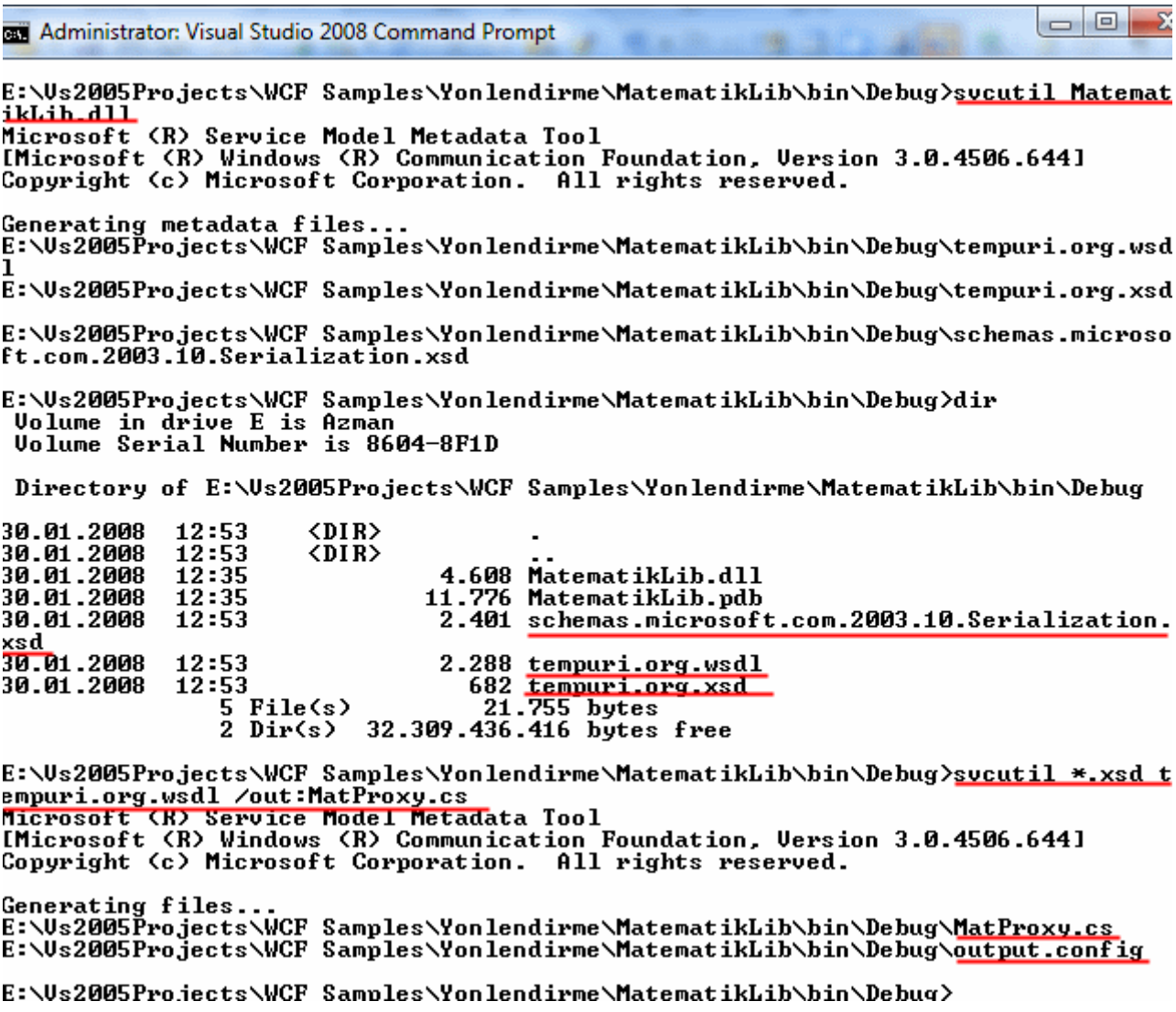
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.serviceModel>
        <behaviors>
            <serviceBehaviors>
                <behavior name="CebirServiceBehavior">
```

```
            <serviceDebug includeExceptionDetailInFaults="true" />
          </behavior>
        </serviceBehaviors>
      </behaviors>
      <services>
        <service behaviorConfiguration="CebirServiceBehavior"
name="MatematikLib.Cebir">
          <endpoint address="http://localhost:50001/Matematik/Cebir.svc"
binding="basicHttpBinding" bindingConfiguration="" name="HttpEndPoint"
contract="MatematikLib.ICebir" />
        </service>
      </services>
  </system.serviceModel>
</configuration>
```

EndPoint tanımlamaları **BasicHttpBinding** bağlayıcı tipini kullanmaktadır. Bununla birlikte her iki serviste aynı **WCF** sözleşmesini(ICebir) sunmaktadır. Yukarıdaki konfigurasyon dosyasının sahibi olan sunucu uygulama **HTTP** protokolüne göre **50001** numaralı port üzerinden yayınlama yaparken diğer **Console** uygulamasıda **50002** numaralı port üzerinden hizmet vermek üzere ayarlanmıştır. özellikle gerçek hayat vakalarında, aynı servis içerisinde yer alan birden fazla farklı **EndPoint** tanımlaması da söz konusu olabilir. Yada bu farklı tipteki **EndPoint** bileşenleri farklı servisler üzerine yayılmış olabilir. Servis uygulamalarına ait kod içerikleri aşağıdaki gibi tasarlanabilir.

```
using System;
using System.ServiceModel;
using MatematikLib;

namespace BackEndService1
{
  class Program
  {
    static void Main(string[] args)
    {
      ServiceHost host = new ServiceHost(typeof(Cebir));
      host.Open();
      Console.WriteLine("Host dinlemede\nKapatmak için bir tuşa basınız");
      Console.ReadLine();
      host.Close();
    }
  }
}
```

Standart olarak servis uygulaması, **Cebir** tipi üzerinden bir **ServiceHost** nesnesi örneklemekte ve **Open** metodu ile hizmete başlamaktadır. Uygulama kapatılırkende **Close** metodu yardımıyla hizmet sonlandırılmaktadır. Bu basit servisleri kullanacak olan istemcilerde birer **Console** uygulaması olarak tasarlanabilirler. Söz konusu istemci uygulamalar için gerekli **proxy** sınıflarının elbette üretilmesi şarttır. Bu amaçla **svcutil** aracı kullanılarak aşağıdaki ekran görüntüsünde olduğu gibi istemciler için gerekli proxy sınıfları üretilebilir. *(HTTP üzerinden yayınlama yapılmasına rağmen servis uygulamaları IIS üzerinden host edilmediklerinden Visual Studio IDE' si içerisinde Add Service Reference seçeneği kullanılamamaktadır. Bu nedenle svcutil aracından yardım alınmaktadır.)*



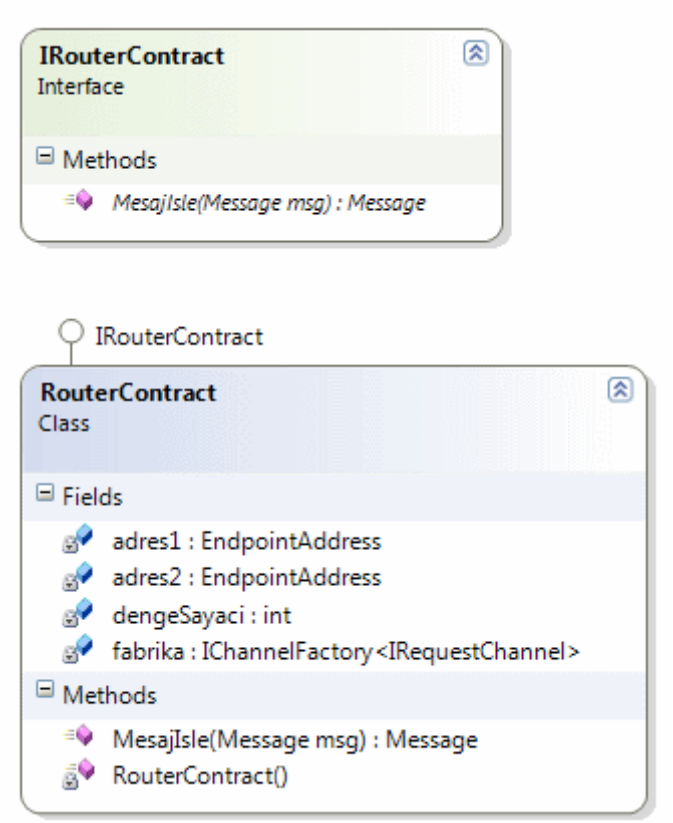
Bu işlemlerin ardından ilgili **proxy** sınıfı ve konfigurasyon dosyası istemci uygulamalara eklenmelidir. İstemci uygulamalar için oluşturuluan **output.config** dosyasının içeriği biraz daha sadeleştirilerek aşağıdaki hale getirilebilir. Bu örnekte **IIS** üzerinden bir hosting yapılmadığından konfigurasyon dosyasında yer alan **EndPoint** elementi içerisine **address niteliği(Attribute)**atılmayacaktır. Bu nedenle adres bilgisininde açık bir şekilde girilmesi şarttır. *(Bununla birlikte Output.config adının App.config olarak değiştirilmesi önerilir)*

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <system.serviceModel>
      <client>
         <endpoint binding="basicHttpBinding"
address="http://localhost:50001/Matematik/Cebir.svc" contract="ICebir"
name="CebirClientEndPoint" />
      </client>
   </system.serviceModel>
</configuration>
```

İstemci uygulamanın kod içeriği aşağıdaki gibi tasarlanabilir.

```
using System;
using System.ServiceModel;

namespace HttpClient1
{
   class Program
   {
      static void Main(string[] args)
      {
         CebirClient client = new CebirClient();
         double toplam=client.Topla(4, 5);
         Console.WriteLine(toplam.ToString());
         Console.WriteLine("çıkmak için bir tuşa basınız.");
         Console.ReadLine();
      }
   }
}
```

İkinci istemci uygulamada aynı türden olmakla birlikte sadece **50002** numaralı **port** adresi üzerinden talepte bulunacak şekilde ayarlanmalıdır. Böylece elimizde hazır bir sistem mevcuttur. Artık**Front-End Service**' in yazılmasına başlanabilir. **Front-End Service** uygulamasıda aslında bir servis uygulaması olduğundan kendi içerisinden istemcilere bir **sözleşme(Contract)** sunmak durumundadır. Ne varki burada istemcilerden gelen taleplerin, dengeli bir şekilde arka servislere(Back-End Services) yönlendirilmesi söz konusudur. Ayrıca gelen **SOAP** paketlerinin ayrıştırılarak uygun olan arka servislere aktarılmasıda gereklidir. Dolayısıyla **Front-End Service** kendi içerisinde özel bir servis sözleşmesi sunmalıdır. Bu nedenle ilk olarak içeriği aşağıdaki gibi olan bir **WCF Servis kütüphanesinin(WCF Service Library)** geliştirilmesi gereklidir.



IRouterContract arayüzüne(interface) ait kod içeriği aşağıdaki gibidir.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Channels;

namespace RouterLib
{
   [ServiceContract]
   public interface IRouterContract
   {
      [OperationContract(Action="*",ReplyAction="*")]
      Message MesajIsle(Message msg);
   }
}
```

Arayüz içerisinde dikkat çekici noktalardan birisi **OperationContract niteliği(Attribute)** içerisinde uygulanan **Action** ve **ReplyAction** özellikleridir. Her iki özelliğe ***** değerinin verilmesi ile, talep edilen operasyon istekleri ne olursa olsun işlem yapılacağı belirtilmetkedir. Buradaki operasyon isimleri bilindiği üzere **WSDL(Web Service Description Language)** dökümanınca belirtilen adlardır. Servis tarafında her ne kadar tek bir fonksiyonellik söz konusu olsada birden fazla işlevin olduğu senaryolarda kabul edilen ve cevaplanan operasyon adlarının belirtilmesi yerine ***** seçeneği sıklıkla kullanılmaktadır.

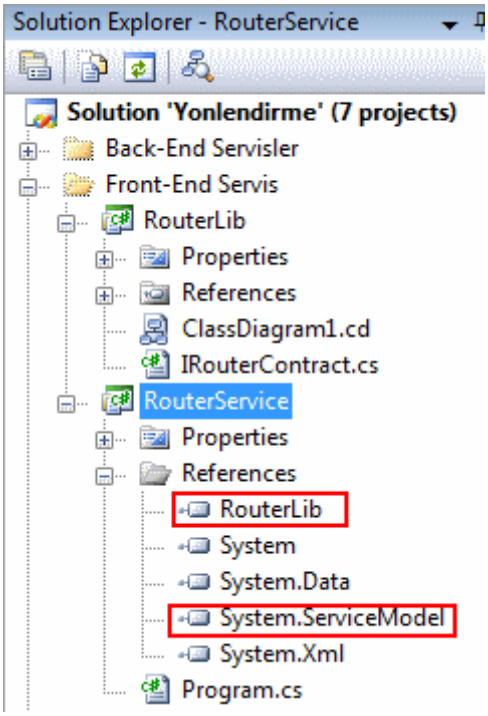
Bunun yanında arayüz(Interface) içerisindeki metod **System.ServiceModel.Channels** isim alanında yer alan **Message** tipinden bir parametre almakta ve aynı tipten bir örnek döndürmektedir. Bu metodun amacı, istemciden gelen mesajların arka servislere yönlendirilmesini sağlamaktır. Ayrıca arka servislerden gelen mesajlarında istemcilere ulaştırılmasında önemli bir role sahiptir. Gerçektende **Message** sınıfının asli görevi **EndPoint** noktaları arasındaki iletişimi sağlamaktır. Geliştirilen bu arayüzü uygulayan **RouterContract** isimli sınıfa ait kod içeriği ise aşağıdaki gibidir.

```
using System;
using System.ServiceModel;
using System.ServiceModel.Channels;

namespace RouterLib
{
   [ServiceBehavior(ValidateMustUnderstand=false,InstanceContextMode=
InstanceContextMode.PerCall)]
   public class RouterContract : IRouterContract
   {
      private static IChannelFactory<IRequestChannel> fabrika = null;
      private EndpointAddress adres1 = new
EndpointAddress("http://localhost:50001/Matematik/Cebir.svc");
      private EndpointAddress adres2 = new
EndpointAddress("http://localhost:50002/Matematik/Cebir.svc");
      private static int dengeSayaci = 1;

      static RouterContract()
      {
         BasicHttpBinding binding = new BasicHttpBinding();
         fabrika = binding.BuildChannelFactory<IRequestChannel>();
         fabrika.Open();
      }

      #region IRouterContract Members

      public Message MesajIsle(Message msg)
      {
        IRequestChannel kanal = null;
         Message cevap = null;
         try
         {
            if (dengeSayaci % 2 == 0)
               kanal = fabrika.CreateChannel(adres1);
            else
               kanal = fabrika.CreateChannel(adres2);
```

```
            dengeSayaci++;
            kanal.Open();
            cevap = kanal.Request(msg);
            kanal.Close();
        }
        catch (Exception exp)
        {
        }
        return cevap;
    }
    #endregion
  }
}
```

öncelikli olarak sınıf içerisinde iki adet **EndPointAddress** değişkeni tanımlandığına dikkat edelim. Bu değişkenler tahmin edileceği üzere **Back-End Service**' lere erişim adreslerini taşımaktadırlar. Diğer taraftan yönlendirici serviste **PerCall** modelinde tasarlanmıştır. Buda istemcilerden gelecek her çağrıda bir **servis örneği(Service Instance)** oluşturulacağı anlamına gelmektedir. Burada sadece tek bir servis referansı olmasını sağlamak için **static yapıcı metod(Static Constructor)** içerisinde bazı kodlamalar yapılmaktadır. **MesajIsle** metodu içerisinde öncelikli olarak **IRequestChannel** arayüzüne ait bir değişken tanımlanmaktadır. Bu değişkenin üretilmesi için generic **IChannelFactory<T>** tipinden olan fabrika isimli değişkenin **CreateChannel** metodu kullanılmaktadır. Bir başka deyişle istemciden gelen talepler sonrasında **dengeSayaci** değişkeninin içeriğine göre uygun olan **kanal nesnesi(Channel Object)** oluşturulmakta ve bu kanal açılarak **talebin(Request)** aktarılması ve sonucunun alınarak geriye döndürülmesi sağlanmaktadır.

Request metodu, MesajIsle metoduna gelen Message tipinden parametre değerini kullanarak, oluşturulan kanal nesnesine bir talepte bulunmaktadır. Bu talep **Front-End Service**' inarkasında yer alan **Back-End**servislerden birisine doğru gerçekleştirilir. **Request** metodu arka servisten gelen cevabı yine bir **Message** değişkeni tipinden alarak sonucun elde edilmesini sağlamaktadır. *(Servis tarafı ile manuel olarak konuşma tekniklerini ilerleyen makalelerimizde incelemeye çalışacağız).* Artık yönlendirici servis tasarlanabilir. Bunun için yine bir **Console** uygulaması göz önüne alınabilir. Console uygulaması bu kez **RouterLib WCF servis kütüphanesini(WCF Service Library)** referans etmelidir.



Söz konusu **Front-End Service** uygulamasının konfigurasyon içeriği aşağıdaki gibi tasarlanabilir.

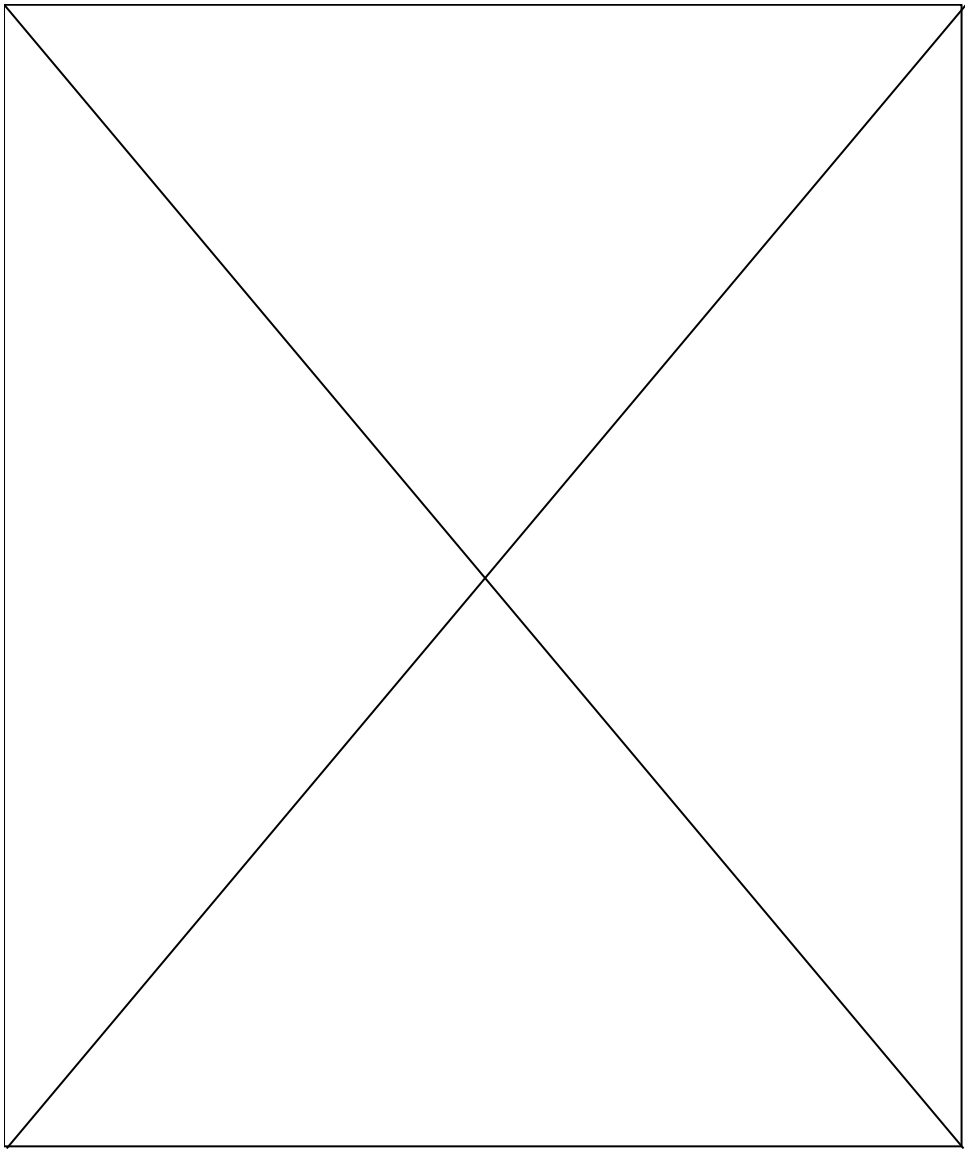
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <behaviors>
         <serviceBehaviors>
            <behavior name="RouterServiceBehavior">
               <serviceDebug includeExceptionDetailInFaults="true" />
            </behavior>
         </serviceBehaviors>
      </behaviors>
      <services>
         <service behaviorConfiguration="RouterServiceBehavior"
name="RouterLib.RouterContract">
            <endpoint address="http://localhost:50003/Matematik/Cebir.svc" binding="
basicHttpBinding" bindingConfiguration="" name="RouterEndPoint"
contract="RouterLib.IRouterContract" />
         </service>
      </services>
   </system.serviceModel>
</configuration>
```

Bu konfigurasyon dosyası içerisinde dikkat edilmesi gereken en önemli nokta adres bilgisidir. Dikkat edileceği üzere **50003** numaralı porttan hizmet verecek bir **EndPoint** tanımlaması yapılmaktadır. Buna göre istemciler **50001** veya **50002** için ayrı

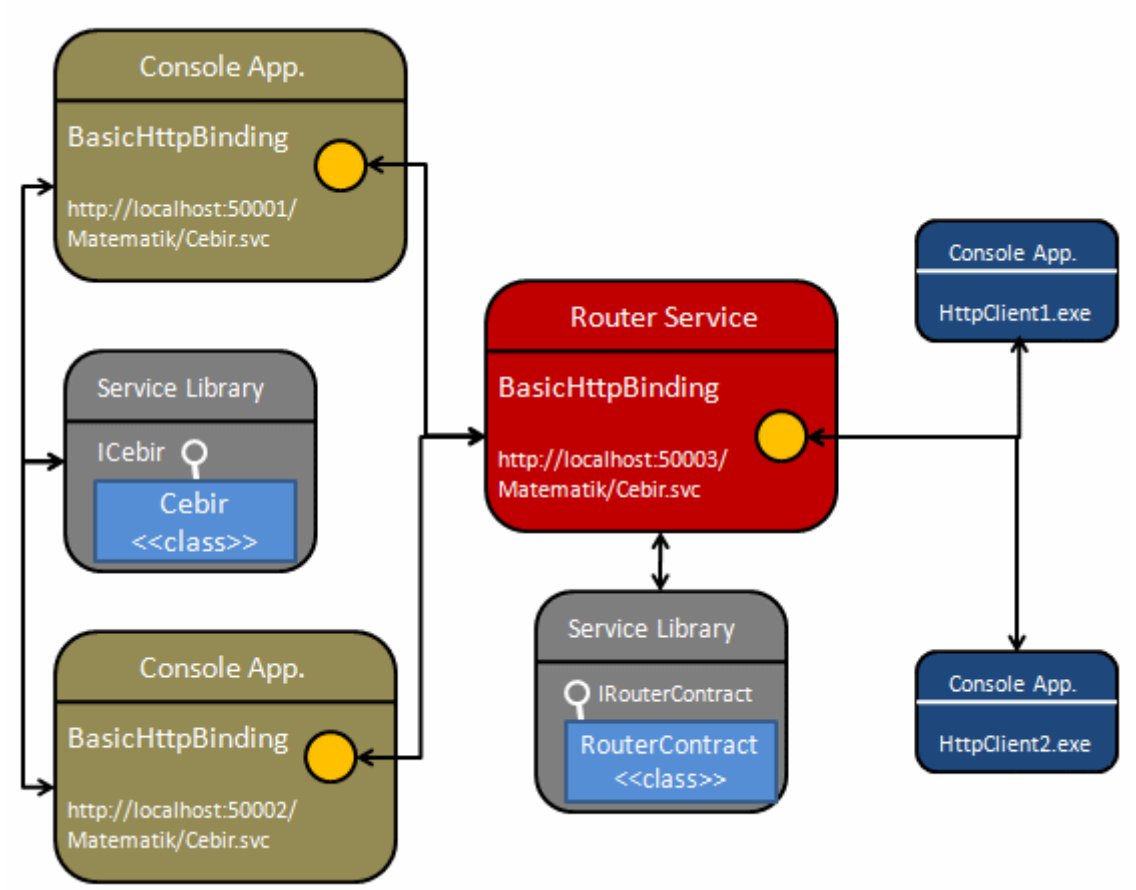
ayrı talepte bulunmaktansa, sadece **50003**' e istekte bulunacaklardır. Onları karşılayan **Front-End Service**' te **Load Balancing** algoritmasına göre arka servisler arasında talepleri dengeli bir şekilde paylaştıracaktır. **RouterService** isimli **Front-End Service** uygulamasının kod içeriği ise aşağıdaki gibidir.

```
using System;
using System.ServiceModel;
using RouterLib;

namespace RouterService
{
   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(RouterContract));
         host.Open();
         Console.WriteLine("Yönlendirici servis dinlemede\nçıkmak için bir tuşa basınız");
         Console.ReadLine();
         host.Close();
      }
   }
}
```

Bu işlemin arkasından tek yapılması gereken istemci uygulamalardaki konfigurasyon dosyalarında yer alan adres bilgisini **http://localhost:50003/Matematik/Cebir.svc** olacak şekilde değiştirmektir. Test aşamasında önce **Back-End Service'** lerin çalıştırılması, sonrasında **Front-End Service**' in yürütülmesi gerekmektedir. Bu servislerin tamamı çalıştığı sürece istemciler hizmet alabilirler. Uygulama eğer **debug** edilerek çalıştırılırsa MesajIsle metodu içerisinde aşağıdaki Flash görselinde yer alan durumun oluştuğu görülür.



Burada dikkat edileceği üzere istemcilerden gelen iki talep sonrasında, if döngüsü farklı arka servis erişimleri gerçekleştirmektedir. Buda zaten kurulan **Load-Balancing** algoritmasının bir sonucudur. Sonuç olarak **Front-End Service**' in eklenmesi ile birlikte sistem aşağıdaki şekilde görülen hale gelmiştir.



Gerçek hayat senaryolarında **Back-End Service** içerisinde yer alan **EndPoint** noktaları farklı **bağlayıcı tipleri(Binding Type)** kullanıyor olabilirler. Bu durumda yönlendirici servislerin tasarlanması biraz daha zorlaşmaktadır. örneğin **WS** standartlarına uygun bağlayıcı tiplerin(örneğin WsHttpBinding gibi) kullanıldığı durumlarda güvenlik(Security) ile ilişkili ayarlamaların mutlaka yapılması gerekmektedir.
Buda **mesaj(Message)** yada **iletişim(Transport)** güvenliği için ek ayarlamaların hem istemciler hemde servisler üzerinde gerçekleştirilmesi anlamına gelmektedir. Bu makalemizde bu tip konular göz ardı edilerek basit anlamda **Load-Balancing** yapan bir yönlendirici servisin nasıl tasarlanabileceği incelenmeye çalışılmıştır. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

Yonlendirme.rar (132,49 kb)

Adım Adım State Machine Workflow Geliştirmek (2008-01-15T05:27:00)

wf,

öyle iş akışları vardırki, **süreç(Process)** içerisinde yer alan adımlar arasındaki geçişler herhangibir zamanda ve herhangibir olayın meydana gelmesi sonrasında mümkün olur. çoğunlukla terminolojide **Sonlu Durum Makinesi(Finite State Machine)** olarak geçen bu yaklaşıma göre, herhangibir nesnel varlığın zaman içerisinde sahip olabileceği durumlar

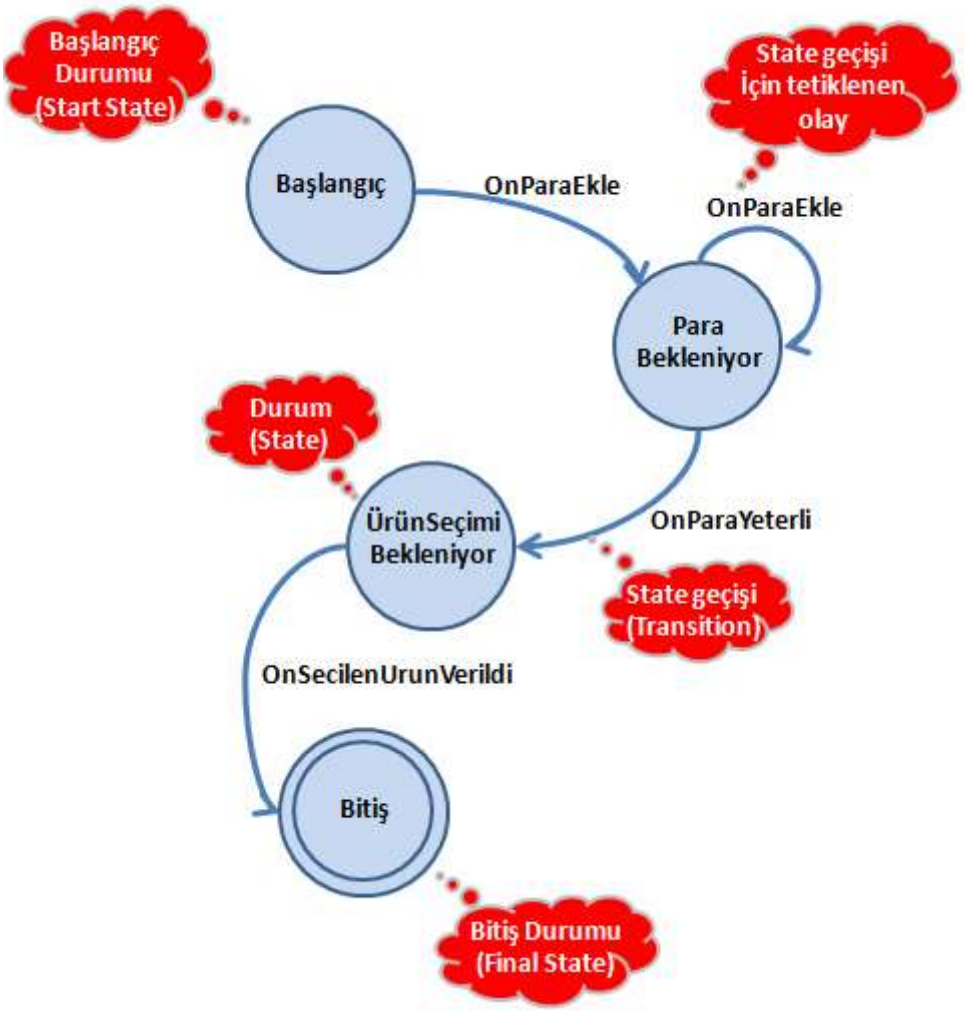
işaret edilmektedir. çok doğal olarak bu durum, programatik ortamda yer alan iş problemlerinin çözümündede göz önüne alınmaktadır. İşte bu makalemizde **Sonlu Durum Makinesi(Finite State Machine)** kavramını irdelemeye ve **Windows Workflow Foundation** içerisindeki kullanımını araştırmaya çalışacağız. Başlamadan önce Sonlu Durum Makinesi(Finite State Machine) kavramını anlamaya çalışamakta yarar vardır.

öncelikli olarak sonlu kelimesinin kullanılmasının sebebi söz konusu nesnel varlığın sahip olabileceği **durumların(State)** sayılı olmasıdır. Bir başka deyişle bu yaklaşıma göre bir makinenin sahip olabileceği durumların sayısı bellidir. Diğer taraftan makinenin zaman içerisinde sahip olabileceği haller onun durumlarını(States) ifade etmektedir. Makine doğal olarak bu durumlara sahip olan nesnel yapıyı temsil etmektedir. **Sonlu durum makinelerinde(Finite State Machine)** durumlar arasındaki geçişler bir aksiyon sonucu gerçekleşir. Tahmin edileceği üzere söz konusu aksiyonlar çoğunlukla bir olaydır ve genellikle insan etkileşimi sonrası gerçekleşmektedir. Ne varki burada insan etkileşimi sonucu olay tetiklenmesi zorunluluk değildir. İnsan etkileşimi olması nedeniyle, olaylar herhangibir zaman dilimi içerisinde meydana gelebilir. Bu tip vakalara mühendislik eğitimlerinde verilmekte olan iki basit örnek vardır. Hepimizin yakından tanıdığı, metro istasyonlarında, duraklarda, okullarda veya bazı kafelerde gördüğümüz otomat makineleri ile herkese açık olan çamaşırhanelerde yer alan ve jeton ile çalışan yıkama makineleri bu iki örneği oluşturmaktadır.

***NOT : Finite State Machine**' lerde insan ile olan etkileşim ön planda olup, **durumlar(States)** arası geçişler çoğunlukla insanlar tarafından tetiklenen **olaylara(Events)** bağlıdır.*

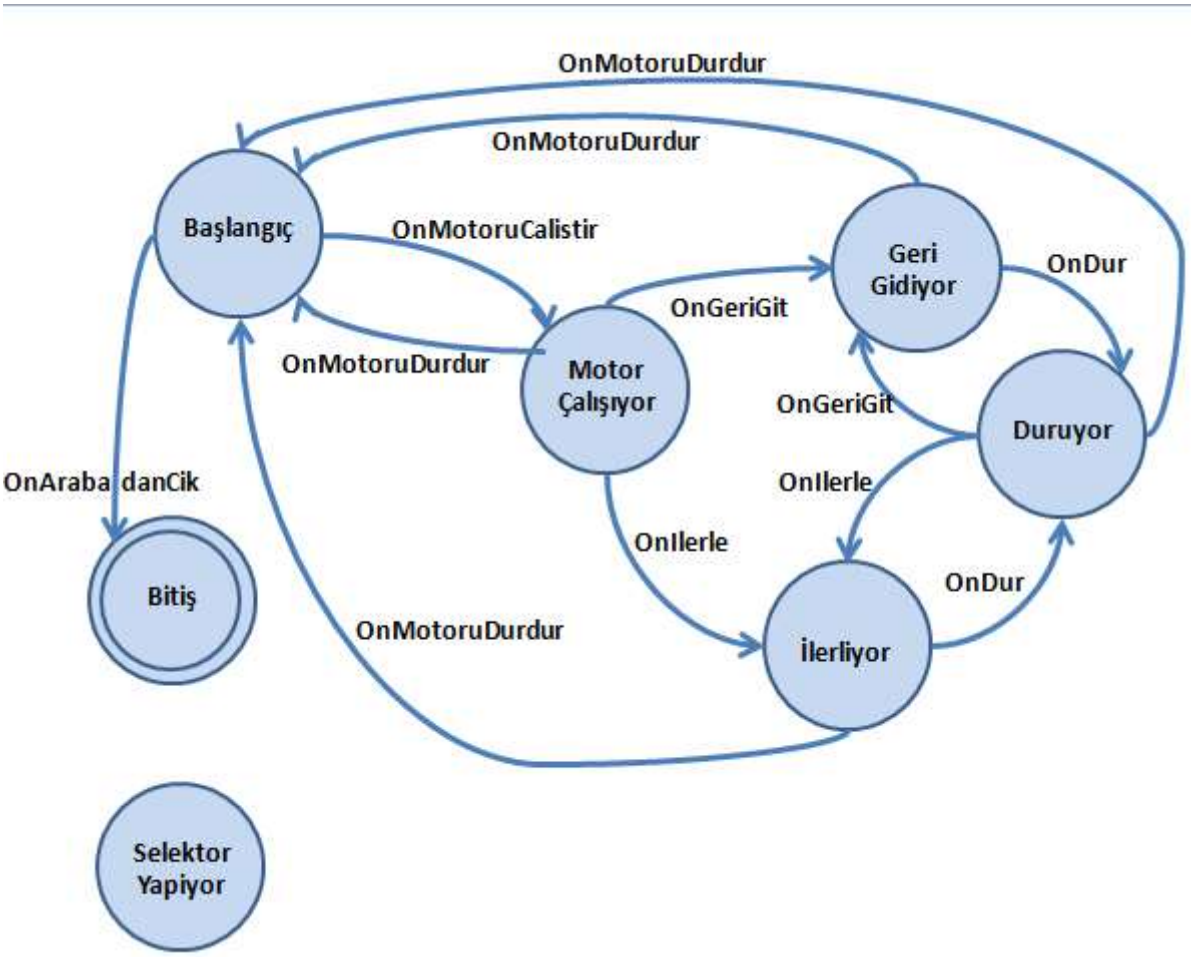
Otomat makinesinin kendisi göz önüne alındığında zaman içerisinde sahip olabileceği bazı **durumlar(State)** vardır. örneğin makine çalışmıyordur, çalışıyordur, para veya jeton bekliyordur, seçilen ürünü veriyordur yada tadilattadır. Burada bahsedilen hallerin tamamı birer **durum(State)** olarak irdelenir. Bu durumlar arasındaki geçişler için çoğunlukla makinenin bazı araçları insanlar tarafından kullanılır. Dikkat edilecek olursa makine zaman içerisinde herhangibir anda herhangibir durumuna geçebilir. Tabiki bazı durumlara geçişler sırasında bazı koşulların sağlanması gerekebilir. Benzer senaryo yıkama makinesi örneği içinde geçerlidir. *(İlerlemeden önce para veya jeton ile çalışan bir çamaşır makinesinin zaman içinde sahip olabileceği durumları ve bu durumlar arasındaki geçişler için gereken olayların neler olabileceğini kağıt üzerinde tasarlamaya çalışmanızı öneririm.)*

Finite State Machine' ler otomat, çamaşır makinesi gibi gerçek hayat örnekleri dışında üretim hattında yer alan sanayi makinelerinin otomasyon süreçlerinde, oyun programlamada yer alan karakterlerin zaman içerisindeki hareket dağılımlarında, **transaction** içerisinde çalışan bir para aktarma veya kredilendirme sürecinde ve benzer senaryolarda göz önüne alınabilir. Modelin böylesine popüler olması, özellikler programatik ortamlarda kolay bir şekile anlaşılabilmelerini sağlamak amacıyla **UML** formasyonunda da ifade edilmesini gerektirmiştir. Söz gelimi aşağıdaki ekran görüntüsünde otomat makinesinin Finite State Machine diagramı yer almaktadır.



Normal şartlarda bu diagramlarda ekranda görülen kırmızı baloncuklar elbetteki yer almaz. Herşeyden önce Finite State Machine içerisinde yer alan makinenin mutlaka bir **başlangıç durumu(Initial State)** ve **son durumu(Finalization State veya Terminal State** olarak adlandırılır- iç içe iki yuvarlağın olduğu parça ile ifade edilir) vardır. Başlangıç durumunda makinenin ilk konumdaki hali göz önüne alınır. Son durumda ise makinenin var olan durumlar sona erdikten sonraki hali ele alınmaktadır. Şekilde başlangıç durumundan, Para Bekleniyor isimli duruma geçiş yapılabilmektedir. Bu geçiş için gereken, makineyi kullanan kişinin para atmasıdır. Para Bekleniyor durumuda, para yeterli oluncaya kadar kendisine geçiş yapılmasına sağlayacak şekilde bir olaya sahiptir. Para yeterli olduktan sonra ise ürün Seçimi Bekleniyor isimli duruma geçiş yapılabilir. Burada ise kişi ürünü seçmekte ve sonrasında makine seçilen ürünü hazırlayarak Bitiş durumuna girmektedir. Bu senaryoda pek çok durum göz ardı edilmiştir. örneğin seçim yapıldıktan sonra makinenin arıza yapıp ürünü vermemesi halinde girilecek durum veya para üstü verme durumları gibi.

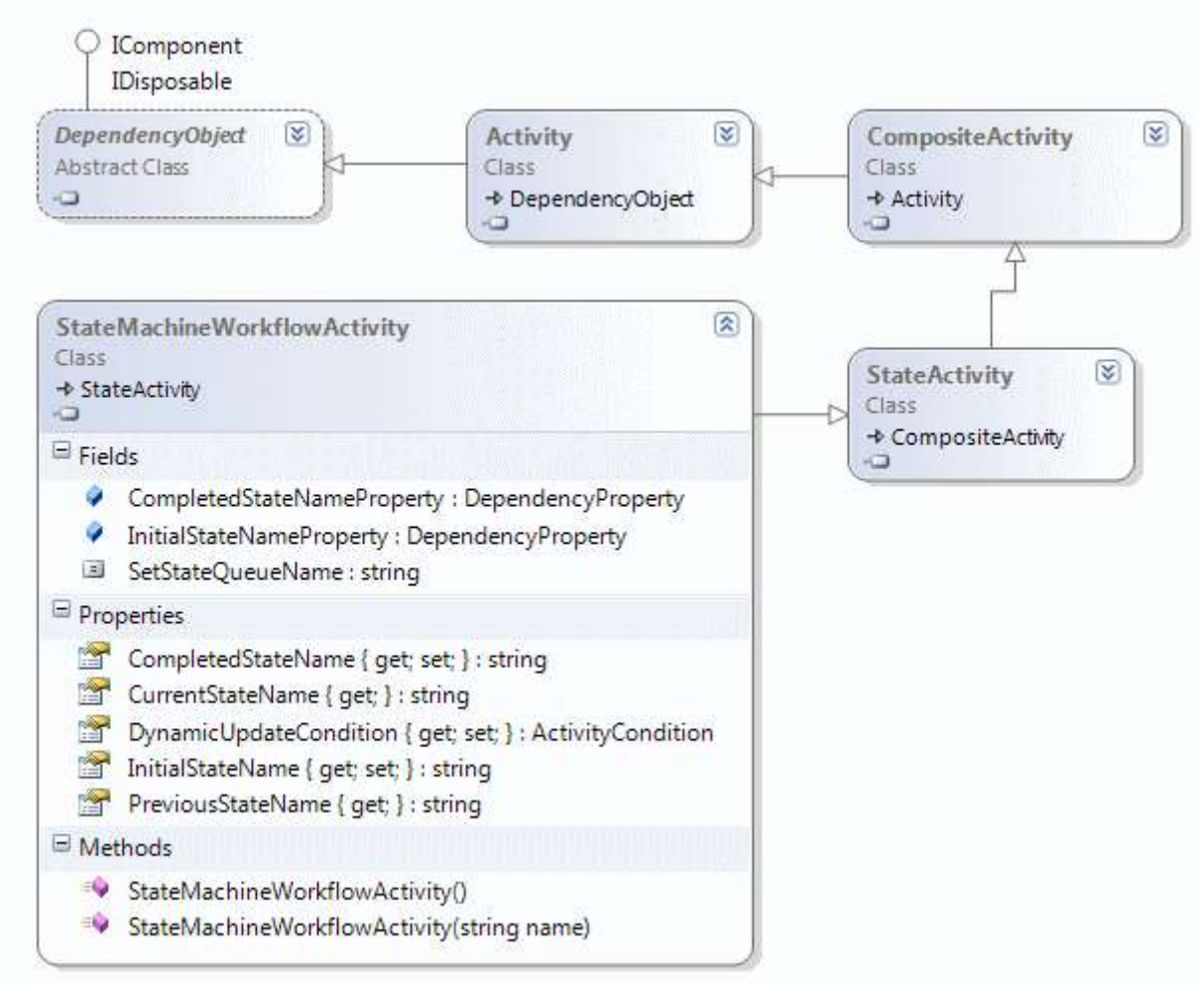
Hemen ikinci bir örnek ile devam edelim. Söz gelimi bir **Windows** uygulaması üzerinden kontrol edilen uzaktan kumandalı bir otomobilin sahip olabileceği durumlara ait bir **Finite State Machine** diagramı söz konusu olabilir. Bu diagram aşağıdakine benzer bir şekilde ele alınabilir.



Burada otomobilin kendisi programatik ortamda bir nesne ile ifade edilebilir. **Durumlar(States)** arasındaki geçişleri sağlayan ise **Windows** uygulamasından tetiklenen nesne **olayları(Events)** olabilir. Arabanın şekle göre sahip olabileceği durumlar belirlidir. Bu durumlarda söz konusu olabilecek olaylarda aynı şekilde belirli ve sayılıdır. Bir durumdan başka bir durumua geçiş veya geçişlerde birden fazla sayıda olay söz konusu olabilir. Söz gelimi Motor çalışıyor durumundayken, **OnIlerle**, **OnMotoruDurdur** veya **OnGeriGit** gibi olaylar tetiklenerek üç farklı duruma geçiş yapılması sağlanabilir. Buda bize durumlar arasındaki geçişlerde birden fazla olayın söz konusu olabileceğini göstermektedir. Hatta bazı noktalarda ortak olaylarda söz konusudur. örneğin araba ileri veya geri giderken **OnDur** olayı tetiklenerek durması sağlanabilir. OnDur olay hem Geri Gidiyor hemde Ilerliyor **durumları(States)** için geçerli ortak bir olaydır.

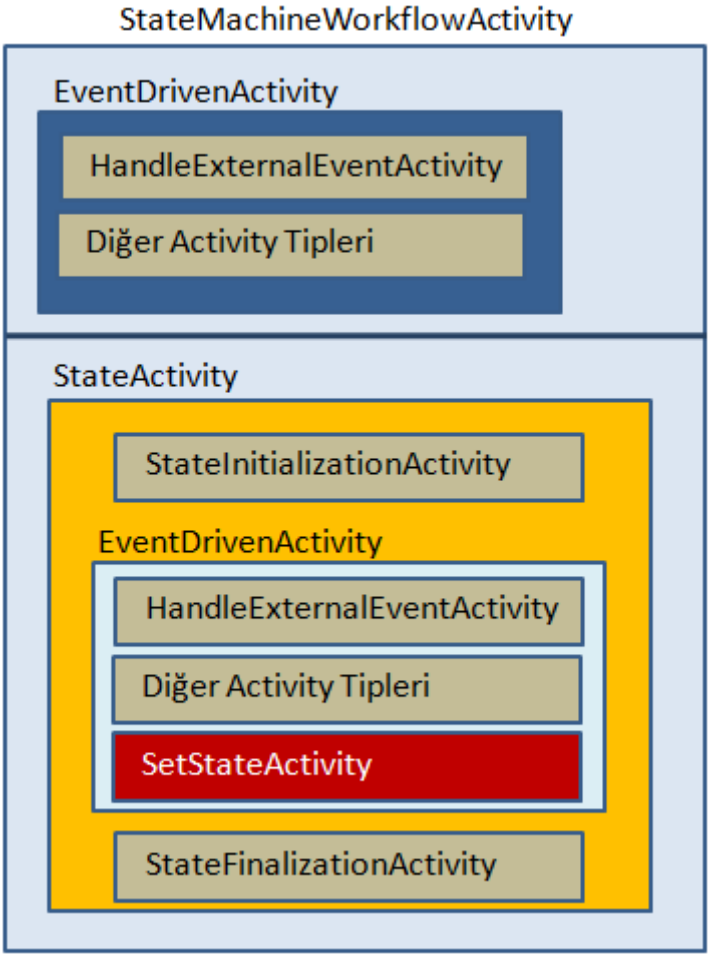
Sanıyorumki buraya kadar anlatılanlar sayesinde **Finite State Machine** yaklaşımı hakkında biraz fikir sahibi olunmuştur. Bundan sonraki kısımlarımızda ise **.Net Framework 3.0** ile gelen**Windows Workflow Foundation** açısından **Finite State Machine** modeline bakıyor olacağız. Nitekim söz konusu model gerçek hayat örneklerine benzer olacak şekilde programatik ortamdaki nesnel yapılar içinde söz konusu olabilmektedir. Bu noktada WWF bize kolaylaştırıcı bir yaklaşım sunmakta ve State Machine tarzı iş süreçlerinin .Net uygulamalarında rahatça ele alınabilmelerine olanak tanımaktadır.

Windows Workflow Foundation(WWF) iki temel iş akışı modelini ele alır. **Sequential Workflows** ve **State Machine Workflows**. Daha öncedende bahsedildiği gibi Sequential Workflows tipinden olan iş akışlarında adımlar yada aktiviteler arasındaki geçişlerin nasıl ve ne zaman olacağı bellidir. Hatta bu geçişler sırasında koşulların kullanılması çok sık rastlanan bir durumdur. Stata Machine Workflow tipindeki iş akışlarında ise daha öncedende değinildiği gibi adımlar veya durumlar arasındaki geçişler dış olayların tetiklenmesine bağlıdır. State Machine Workflow akışlarından, aktivitenin kendisi **StateMachineWorkflowActivity** sınıfından örneklenmektedir. *(Workflow mimarisindeki herşeyin birer **aktivite(Activity)** tipi olduğunu hatırlayalım)* **StateMachineWorkflowActivity** tipinin .**Net** içerisindeki yerine bakıldığında aşağıdaki **sınıf diagramında(Class Diagram)** yer alan hiyerarşide olduğu görülür.



Dikkat edileceği üzere **StateMachineWorkflowActivite** sınıfıda eninde sonunda bir **Activity** tipidir. Kendi içerisinde tanımlanmış olan üyelerden bir kaçını açıklayarak devam edelim.**CompletedStateName** özelliği ile, **bitiş durumu(Finalization State)** ifade edilir. Bu özellik önemlidir nitekim **Workflow**' un hangi durumdan sonra sonlanacağını belirtmektedir. Ancak yazılmadığı vakalarda vardır. Benzer şekilde **InitialStateName** özelliği başlangıçtaki durum aktivitesini işaret etmektedir. çalışma zamanında istenirse o anda makinenin bulunduğu durum

adı**CurrentStateName** özelliği ile elde edilebilir. Benzer şekilde **PreviousStateName** özelliği ile o anda bulunulan durumdan bir önceki durum adı elde edilebilir ki bu hangi durumdan gelindiğini öğrenmek için kullanılabilir. Buradaki örnek **özellikler(Properties)** dışında üst sınıflardan gelen pek çok **üye(Member)** yer almaktadır. Amacımız şu an için bu tipin tüm üyelerini öğrenmek değildir. Bunun yanında bir **StateMachineWorkflowActivity** tasarlanırken içerisinde çoğunlukla aşağıdaki şekilde yer alan tipler kullanılır.

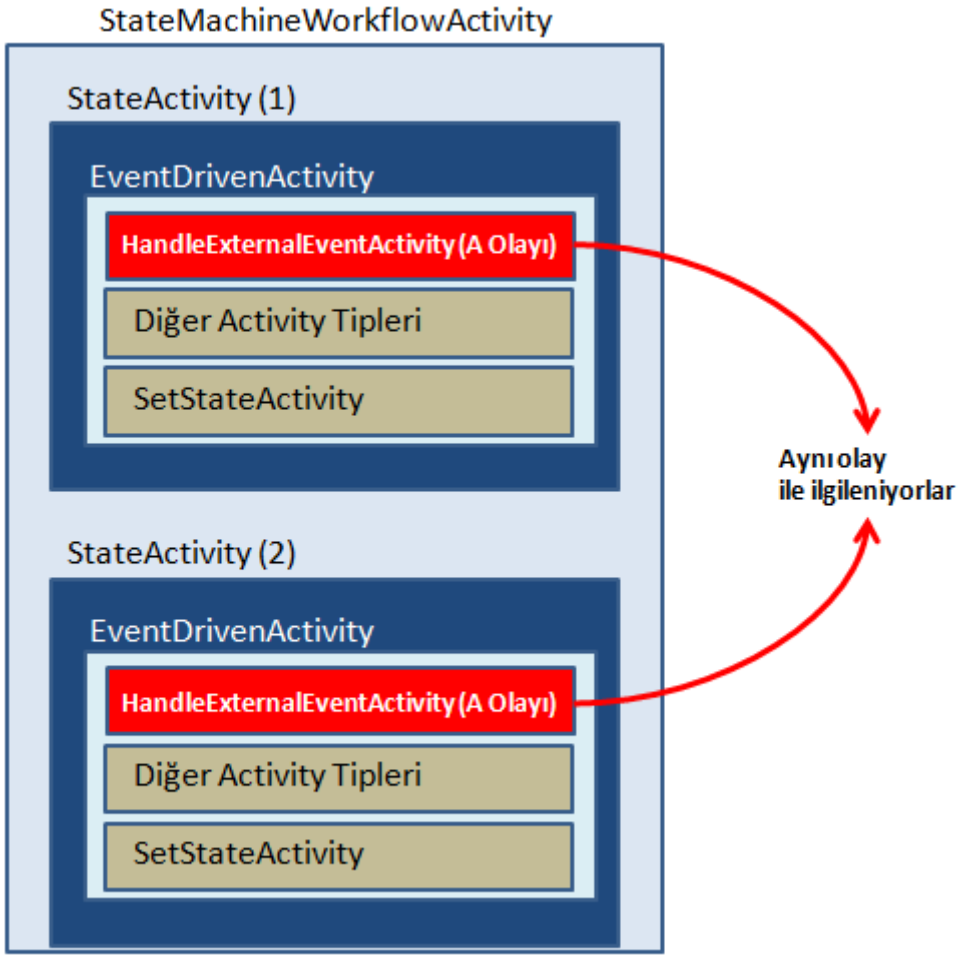


Burada belkide en kritik ve değerli tip **StateActivity** sınıfıdır. StateActivity, aslında makinenin içerisinde bulunacağı durumları işaret etmektedir. çok doğal olarak bir StateActivity içerisinde **StateInitializationActivity** yada **StateFinalizationActivity** tanımlanabilir. Ancak bir StateActivity içerisinde bunlardan sadece bir tane bulunabilir. öte yandan StateActivity içerisinde,**StateInitializationActivity** veya **StateFinalizationActivity** tiplerinin tanımlanması zorunlu değildir. Bunlar opsiyonel olarak ele alınmaktadır. **Durumlar(States)** arasındaki geçişler için**EventDrivenActivity** tipi kullanılmaktadır. StateActivity içerisinde birden fazla **EventDrivenActivity** nesnesi tanımlanabilir. Nitekim daha öncedende bahsettiğimiz gibi, bir durumda(State) söz konu olabilecek birden fazla **olay(Event)** olabilir. Her EventDrivenActivity mutlaka olayları

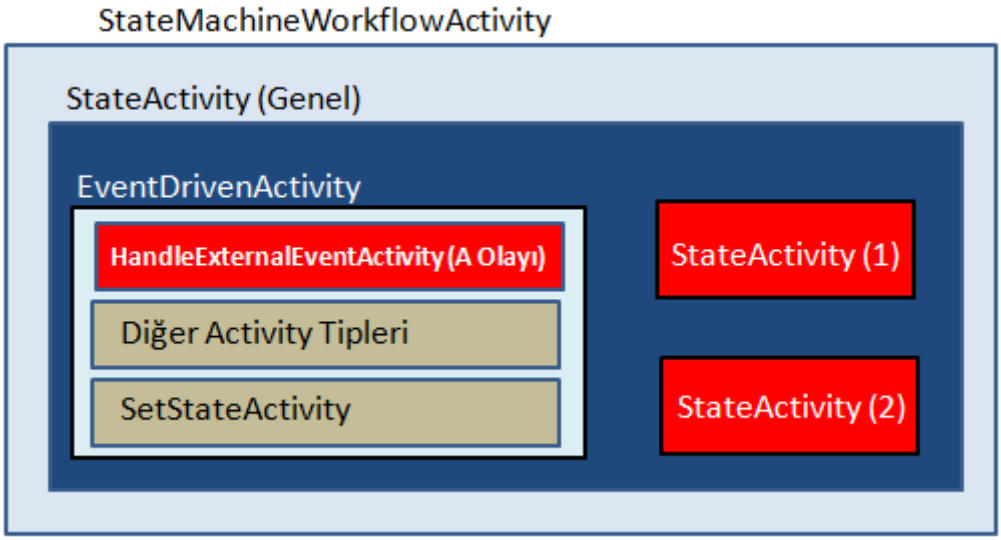
alabilecek bir aktivite tipi içerir. Bunu sağlayan **HandleExternalEventActivity** tipidir. Söz konusu aktivite tipini takiben herhangibir başka aktivitede gelebilir. örneğin bir olayın tetiklenmesinin ardından host uygulama üzerinden çağırılabilecek harici metodların ele alınması, kod işletilmesi gibi işlemler yapılabilmektedir. Şekilde dikkat edileceği üzere **EventDrivenActivity** içerisinde son olarak **SetStateActivity** kullanılmaktadır. Bu tip sayesinde bulunulan durumdan diğer bir duruma geçilmesi sağlanmaktadır.

Şekilde dikkat edilmesi gereken noktalardan biriside **StateActivity** tipleri dışında ve **StateMachineWorkflowActivity** içerisinde kalan alanda **EventDrivenActivity** bileşenlerinin tanımlanabilmesidir. Bazı hallerde durumlar(States) arasındaki geçişlerde kullanılmayan olaylar(Events) söz konusu olabilir. Söz gelimi otomobilin selektör yapması herhangibir anda herhangibir duruma geçiş yapılmasını gerektirmeyecek bir vaka olarak ele alınır. Bu sebepten bu vakaya ilişkin olayı ele alacak **EventDrivenActivity** nesnesinin bir **StateActivity** içerisinde tanımlanmasına da gerek yoktur.

Bazı durumlarda **StateActivity** bileşenleri kendi içlerindede birden fazla **StateActivity** içerebilir. Bu genellikle içerideki aktivitelerin aynı olayları ele aldığı durumlarda söz konusudur. Bu tip aktiviteler **Recursive Compositon Activities** olarakda adlandırılmaktadır. Aşağıda şekilde bu durum ele alınmaya çalışılmaktadır.

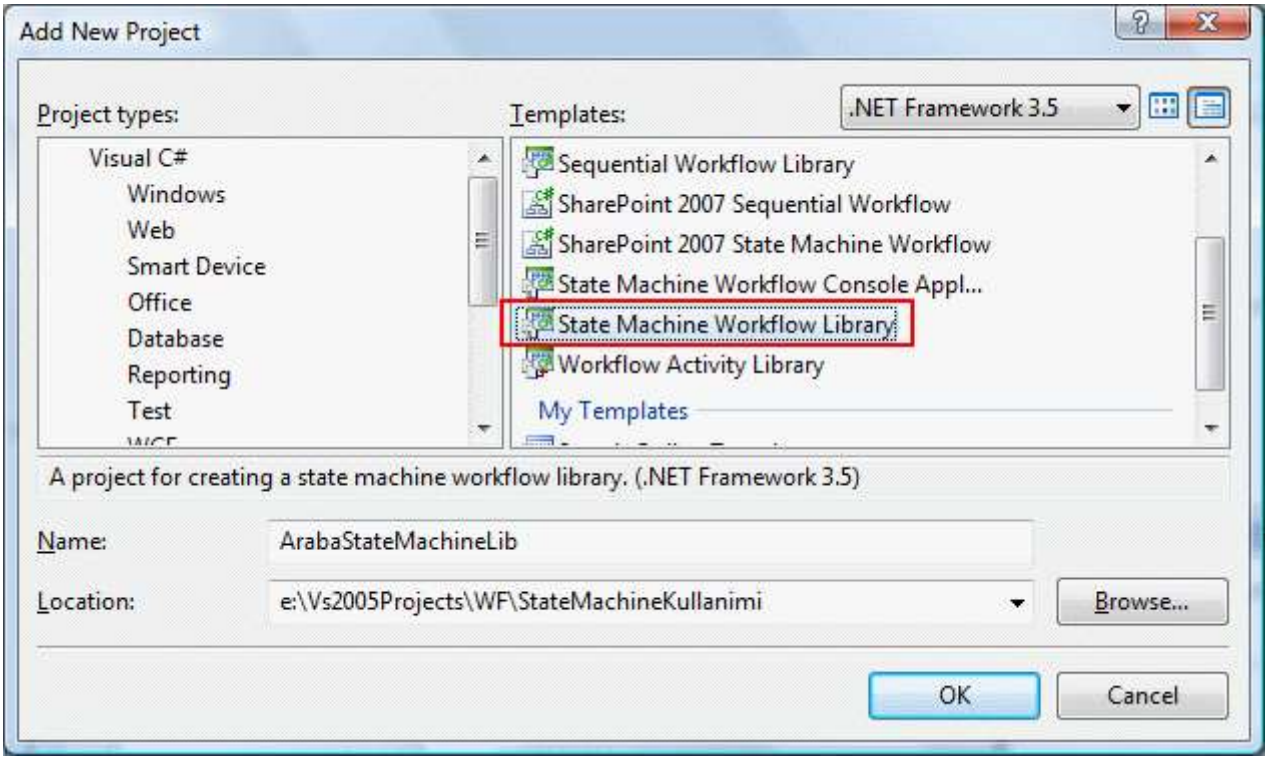


Burada her iki **StateActivity** tipi içerisinde yer alan **HandleExternalEventActivity** nesneleri aynı olay ile ilgilenmektedir. Böyle bir durumda söz konusu **StateActivity** aşağıdaki şekilde görüldüğü gibide tasarlanabilir. *(Otomobil örneği göz önüne alındığında Geri gitme veya ileri gitme durumları içerisinden Durma durumuna geçilmesi için aynı olaylar ele alınmaktadır.)*

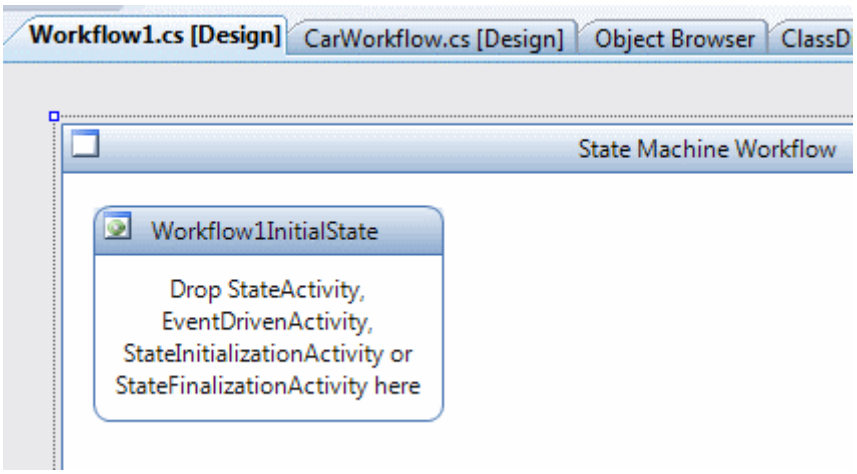


Görüldüğü gibi **StateActivity1** ve **StateActivity2** nesne örnekleri genel bir **StateActivity** nesnesi içerisine alınmıştır. Bununla birlikte her ikisinin **EventDrivenActivity** nesneleri dışarı alınarak tek bir noktada toplanmıştır. Nitekim her iki alt aktivitede aynı **EventDrivenActivity** nesnelerini ele almaktadır.

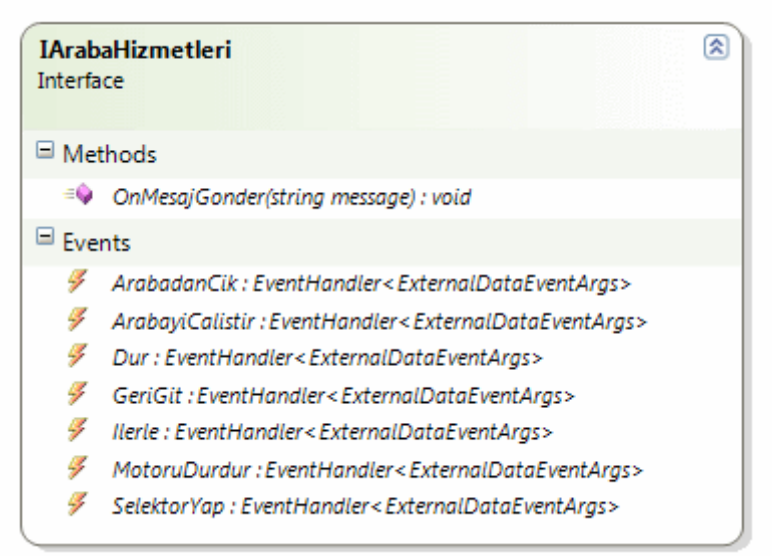
Bu kadar teorik bilgiden sonra bir örnek yaparak devam etmekte yarar vardır. Makale yazılmadan önce yapılmış olan araştırmalarda **Microsoft**' un **Otomat** makinesi örneğini kullandığı,**APress**' in ise bir arabanın durumlarını ele aldığı gözlenmiştir. Bizde senaryo olarak **APress** tarafından ele alınan Otomobil örneğini kendimize göre adım adım geliştirmeye ve anlamaya çalışacağız. İşe ilk olarak yeni bir **State Machine Workflow Library** projesi açarak başlayalım. Bunun için **Visual Studio 2008** ortamında **New Project->WF** sekmesinden ilgili**proje şablonunu(Project Template)** seçmemiz yeterlidir.



Proje oluşturulduğunda otomatik olarak **Workflow1**.cs dosyası tasarım penceresinde açılacak ve aşağıdaki ekran görüntüsü oluşacaktır.



Burada görüldüğü gibi **Workflow1**.cs içerisinde varsayılan olarak bir **InitialState** bileşeni bulunmaktadır. Şimdi örneğin temasını oluşturan arabanın zaman içerisindeki durumları göz önüne alınabilir. Bu durumları listeledikten sonra ise gerekli tiplerin hazırlanmasına başlanabilir. Herşeyden önce arabanın zaman içerisindeki durumları arasındaki geçişleri sağlayacak olan olayların veri değişimi sağlayacak şekilde tasarlanmış bir **arayüz(Interface)** içerisinde yer alması sağlanmalıdır. Bu amaçla projeye aşağıda sınıf diagramı(Class Diagram) ve kod çıktısı yer alan **arayüz(Interface)** eklenir.




```
[ExternalDataExchange]
public interface IArabaHizmetleri
{
   event EventHandler<ExternalDataEventArgs> ArabayiCalistir;
   event EventHandler<ExternalDataEventArgs> MotoruDurdur;
   event EventHandler<ExternalDataEventArgs> Dur;
   event EventHandler<ExternalDataEventArgs> Ilerle;
   event EventHandler<ExternalDataEventArgs> GeriGit;
   event EventHandler<ExternalDataEventArgs> ArabadanCik;
   event EventHandler<ExternalDataEventArgs> SelektorYap;

   void OnMesajGonder(string message);
}
```

Bu arayüz(Interface) basit olarak iş akışına **yerel bir servis(Local Service)** üzerinden sunulabilecek üye bildirilmlerini içermektedir ki bunlar çoğunlukla olay ve metod tanımlamalarıdır. Diğer taraftan arayüz tipi **ExternalDataExchange** niteliği(attribute) ile işaretlenmiştir. Bu **niteliğin(Attribute)** uygulanması sayesinde arayüz tipi, iş akışları tarafından **yerel bir servis(Local Service)** olarak kullanılabilir hale gelir. Arayüz(Interface) içerisinde durum geçişleri(State Transitions) için gerekli temel olay tanımlamaları yer almaktadır. örneğin arabanın ilerlemesi için **Ilerle** yada geriye gitmesi için **GeriGit** olaylarına ait bildirimler bulunmaktadır. Bununla birlikte arayüz, **OnMesajGonder** isimli bir metod bildirimi de içermektedir. Bu metod iş akışı(Workflow) tarafından, **host** uygulamaya mesaj göndermek amacıyla kullanılacaktır.

NOT : *Bilindiği gibi* ***arayüzler(Interface),*** *sadece üye bildirimleri içeren tiplerdir.* ***Polimorfik*** *yapıları vardır ve* ***çoklu kalıtma(Multi Inheritance)*** *destek verirler. çoğunlukla****türetme(Inheritance)*** *için kullanılır ve türeyen üyelerin mutlaka uyması gereken kuralları bildirirler.* ***Plug-In*** *tabanlı programlamada, tip genişletmelerinde, ortak*

*sözleşmelerin sunulmasında(Söz gelimi **WCF** gibi **SOA-Service Oriented Architecture** mimarilerinde) vb... gibi senaryolarda sıklıkla kullanılırlar.*

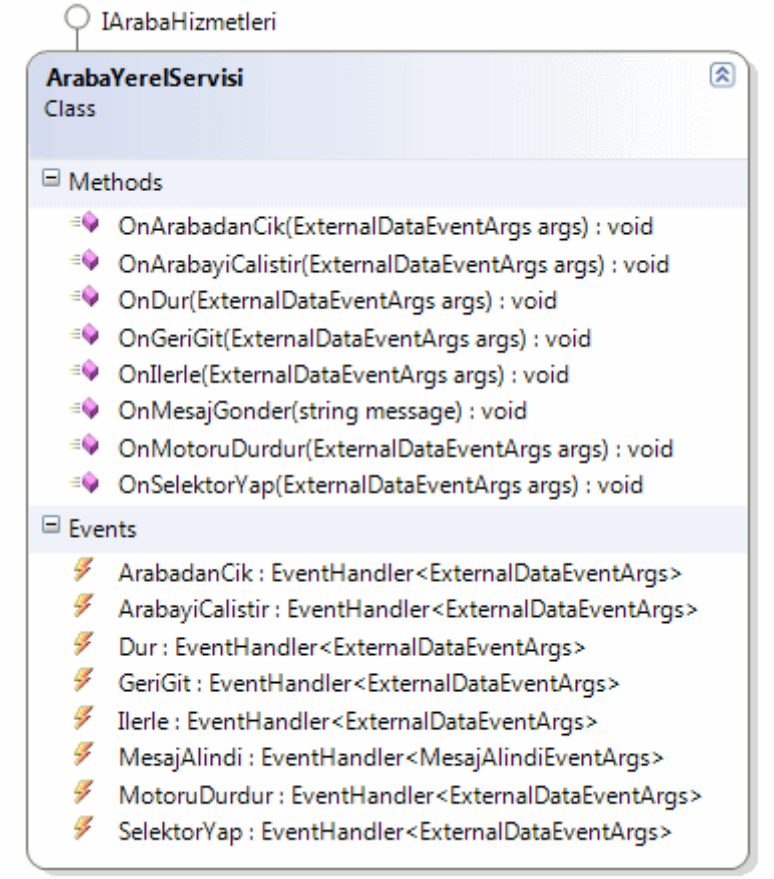
Arayüzün tanımlanmasından sonra bunu uygulayan sınıfın tasarlanması gerekmektedir. Bu **sınıf(Class)** aynı zamanda yerel bir **servis(Local Service)** olacaktır. Söz konusu sınıf ve**MesajGonder** için kullanılan yardımcı olay parametresinin içeriği aşağıdaki gibidir.

MesajAlindiEventArgs sınıfı;


```
[Serializable]
public class MesajAlindiEventArgs : ExternalDataEventArgs
{
   private string _bilgi;

   public string Bilgi
   {
      get { return _bilgi; }
      set { _bilgi = value; }
   }
   public MesajAlindiEventArgs(Guid ornekId, string bilgi)
         : base(ornekId)
   {
      _bilgi = bilgi;
   }
}
```

ExternalDataEventArgs sınıfının tüm yapıcı metod(Constructor Method) versiyonları **Guid** tipinden bir ilk parametre alırlar. Bu sebepten **base** anahtar kelimesi kullanılarak**MesajAlindiEventArgs** sınıfına gelen **Guid** değerinin üst sınıf örneğine gönderilmesi sağlanmaktadır.

ArabaYerelServisi sınıfı;




```
public class ArabaYerelServisi :IArabaHizmetleri
{
   #region IArabaHizmetleri Members

   public event EventHandler<ExternalDataEventArgs> ArabayiCalistir;
   public event EventHandler<ExternalDataEventArgs> MotoruDurdur;
   public event EventHandler<ExternalDataEventArgs> Dur;
   public event EventHandler<ExternalDataEventArgs> Ilerle;
   public event EventHandler<ExternalDataEventArgs> GeriGit;
   public event EventHandler<ExternalDataEventArgs> ArabadanCik;
   public event EventHandler<ExternalDataEventArgs> SelektorYap;

   public void OnMesajGonder(string message)
   {
      if (MesajAlindi != null)
      {
         MesajAlindiEventArgs args = new
MesajAlindiEventArgs(WorkflowEnvironment.WorkflowInstanceId, message);
         MesajAlindi(this, args);
      }
   }
```

```
#endregion

#region Host uygulama tarafından kullanılan üyeler

public event EventHandler<MesajAlindiEventArgs> MesajAlindi;

public void OnArabayiCalistir(ExternalDataEventArgs args)
{
   if (ArabayiCalistir != null)
      ArabayiCalistir(null, args);
}

public void OnMotoruDurdur(ExternalDataEventArgs args)
{
   if (MotoruDurdur != null)
      MotoruDurdur(null, args);
}

public void OnDur(ExternalDataEventArgs args)
{
   if (Dur != null)
      Dur(null, args);
}

public void OnIlerle(ExternalDataEventArgs args)
{
   if (Ilerle != null)
      Ilerle(null, args);
}

public void OnGeriGit(ExternalDataEventArgs args)
{
   if (GeriGit != null)
      GeriGit(null, args);
}

public void OnSelektorYap(ExternalDataEventArgs args)
{
   if (SelektorYap != null)
      SelektorYap(null, args);
}

public void OnArabadanCik(ExternalDataEventArgs args)
{
   if (ArabadanCik != null)
```

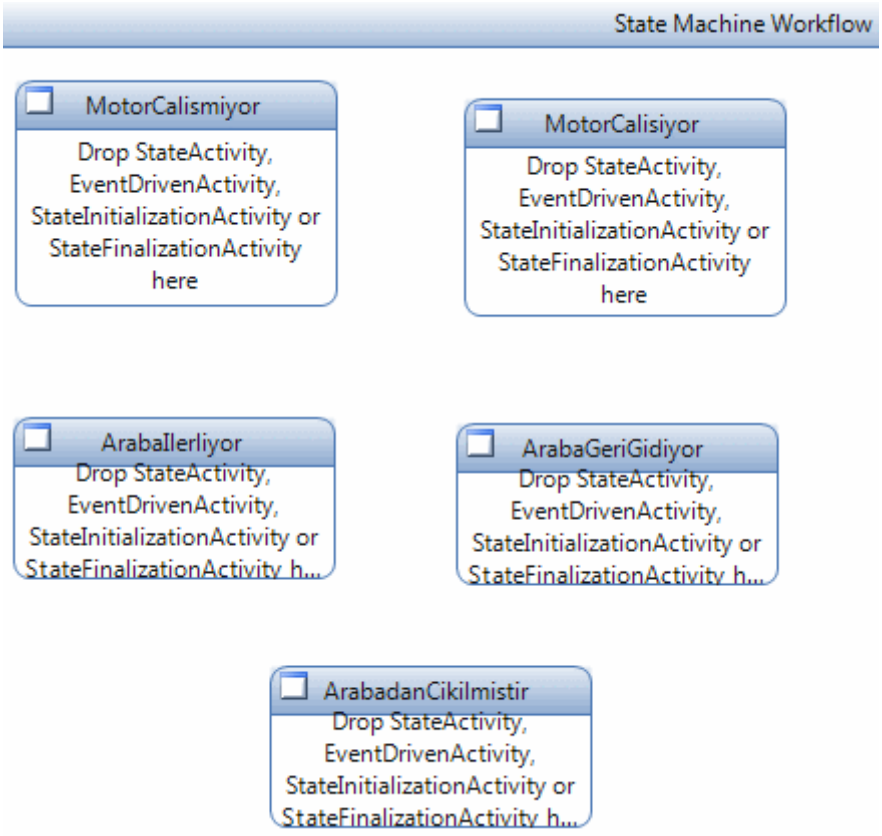
```
        ArabadanCik(null, args);
    }

    #endregion
}
```

ArabaYerelSinifi isimli sınıf(Class), ilgili **arayüzü(Interface)** uygulamak dışında **Host** uygulama tarafından tetiklenebilecek metodlarda içermektedir. Bu metodlar kendi içlerindende akışa ait durum geçişleri için gerekli olayların tetiklenmesinde kullanılmaktadır. Sınıf içerisinde yer alan **OnMesajGonder** isimli metod parametre olarak **string** tipinden bir değişken almaktadır. Bu parametre değeri **iş akışından(Workflow)** gelmekte olup **Host** uygulamaya iletilmektedir. Bu iletim sırasında **MesajAlindi** isimli olay devreye girmektedir ki bu olay sadece iş akışını barındıran Host uygulama tarafından ele alınabilir. Tahmin edileceği üzere **OnMesajGonder** metodunun parametre değeri iş akışı tasarlanırken belirlenecektir. **MesajAlindi** olayı tetiklenirken parametre olarak **ExternalDataEventArgs** sınıfından türemiş olan **MesajAlindiEventArgs** sınıfı kullanılmaktadır.

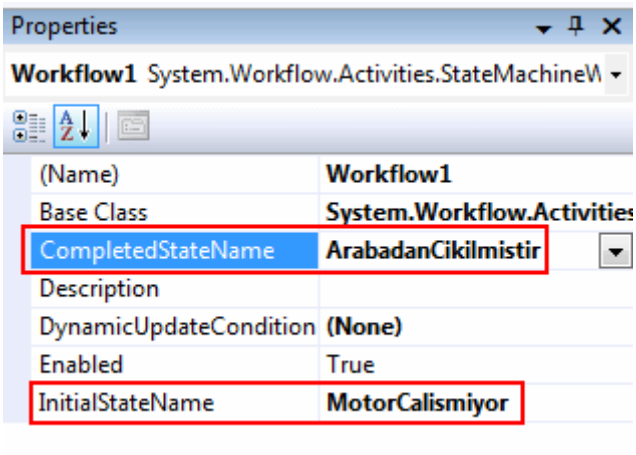
Bu işlemlerin tamamlanmasının ardından **Durum Makinesinin(State Machine)** tasarlanmasına başlanabilir. Bu amaçla **Workflow1.cs** üzerinden gerekli düzenlemelerin yapılması gerekmektedir. İlk olarak arabanın sahip olabileceği tüm durumlar **StateActivity** bileşenleri yardımıyla iş akışı üzerine alınırlar. Başlangıçta **StateActivity** bileşenlerinin **Name**özelliklerinin değerlerinin aşağıdaki tabloda yer aldığı gibi değiştirildiğini düşünelim.

StateActivity Bileşeni Name özelliği Değeri	**Kısa Bilgi**
MotorCalismiyor	Arabanın motorunun çalışmadığı durumu işaret eder.
MotorCalisiyor	Arabanın motorunun çalışmak olduğu durumu işaret eder.
ArabaIlerliyor	Arabanın ileri doğru hareket ettiği durumu işaret eder.
ArabaGeriGidiyor	Arabanın geriye doğru hareket ettiği durumu işaret eder.
ArabadanCikilmistir	Arabadan inildikten sonraki durumu işaret eder.

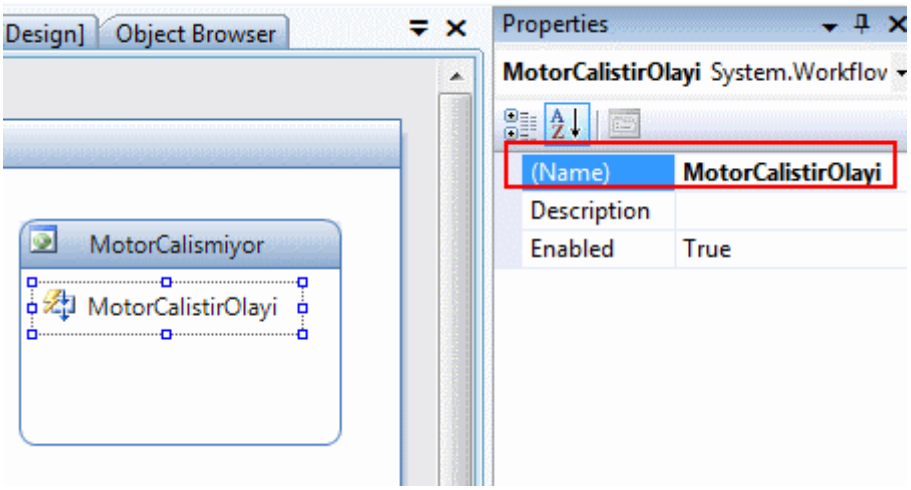
Bunun sonrasında iş akışına(Workflow) ait ekran görüntüsü aşağıdaki gibi olacaktır.



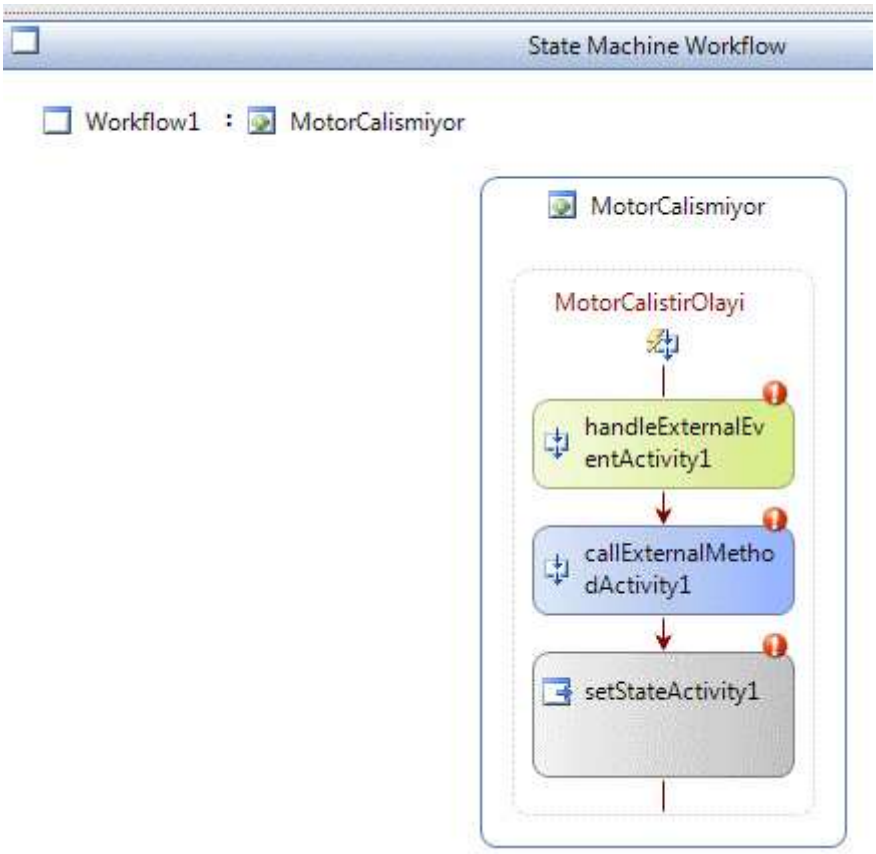
Son işlemleri takiben iş akışının başlangıç ve bitiş durumları belirlenebilir. Bunun için **Workflow1**' in özellikler(Properties) penceresinden **InitialStateName** ve **CompletedStateName**özelliklerine ilgili değerlerin verilmesi gerekmektedir. Senaryo gereği **MotorCalismiyor** başlangıç ve **ArabadanCikilmistir** bitiş **durumlarını(State)** bildirmektedir.



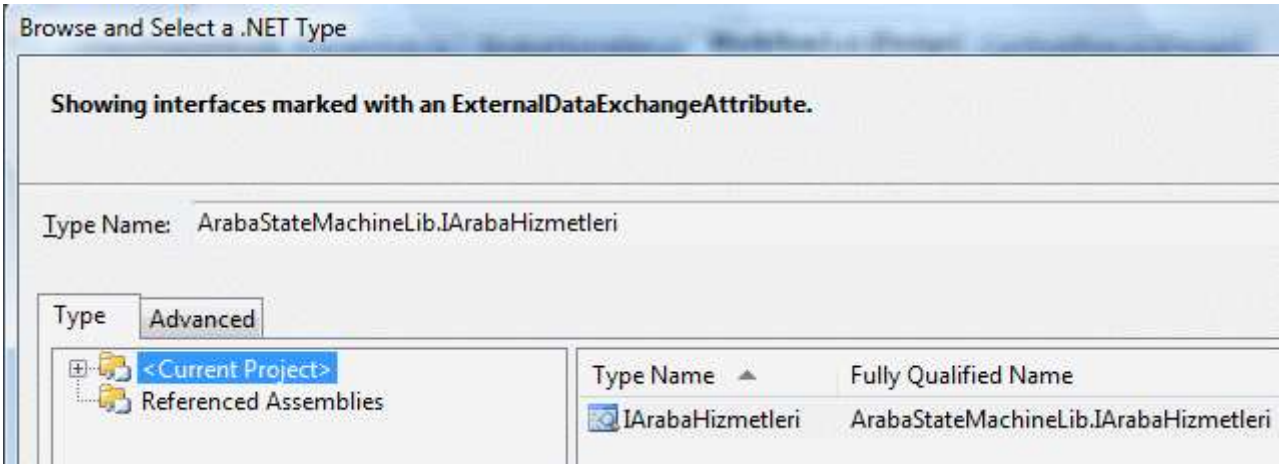
Artık ilk durumdan diğerine geçisi sağlayacak olan olay aktivitesi tanımlanabilir. Bu amaçla **MotorCalismiyor** isimli **StateActivity** içerisine bir adet **EventDrivenActivity** bileşeni sürüklenir.**EventDrivenActivity** bileşeninin **Name** özelliğine **MotorCalistirOlayi** adı verilebilir. Sonuç olarak ekran görüntüsü aşağıdaki gibi olacaktır.



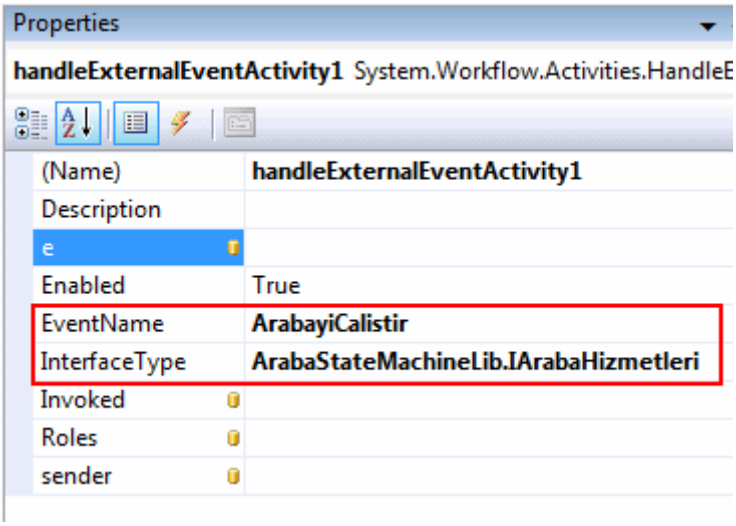
Daha öncedende bahsettiğimiz gibi **EventDrivenActivity** içerisinde genel olarak 3 farklı aktivite kullanılır. Bu senaryoda söz konusu olayın tetiklenmesi için **HandleExternalEventActivity** bileşeni ele alınmalıdır. Diğer taraftan yerel servis üzerinden harici metod çağrısı için **CallExternalMethodActivity** bileşeni değerlendirilir. Son olarak olayın tetiklenmesi sonrası geçilecek olan durumu işaret etmek için **SetStateActivity** bileşeni kullanılmalıdır. **EventDrivenActivity** bileşeni içerisine bahsedilen kontrolleri oluşturmak için **MotorCalistirOlayi** üzerinde çift tıklanması yeterlidir. Sonuç olarak ilk durum için tasarlanan **EventDrivenActivity** bileşeninin içeriği aşağıdaki ekran görüntüsünde yer aldığı gibi olacaktır.



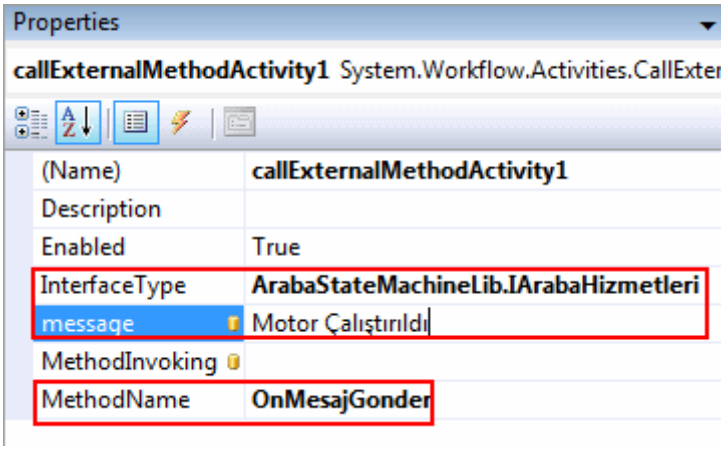
İlk olarak **HandleExternalActivity** bileşeni ile başlayalım. Bu bileşenin **InterfaceType** özelliğine tetiklenecek olan olayın bildirimini içeren arayüz adı verilmelidir. Bu amaçla üç nokta düğmesine basıldığında aşağıdakine benzer bir arabirim ile karşılaşılır. Bu arabirimden aynı proje içerisindeki veya farklı bir projedeki **ExternalDataExchange** niteliğini uygulayan arayüzler görülebilir. örneğimizdede **IArabaHizmetleri** arayüzü aktif olarak gelmektedir.



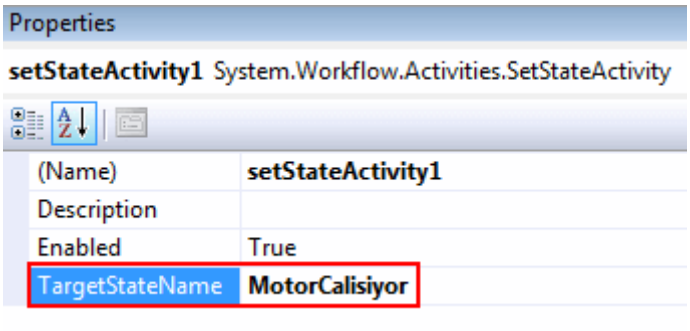
Arayüz seçimi yapıldıktan sonra **EventName** özelliğinde ele alınabilecek olan, bir başka deyişle **interface** tipi içerisinde bildirilmiş olan olayların listesi gelecektir. örneğimizdeki ilk durum için **ArabayiCalistir** olayı seçilmelidir. **EventDrivenActivity** için söz konusu olan durum aşağıdaki gibidir.



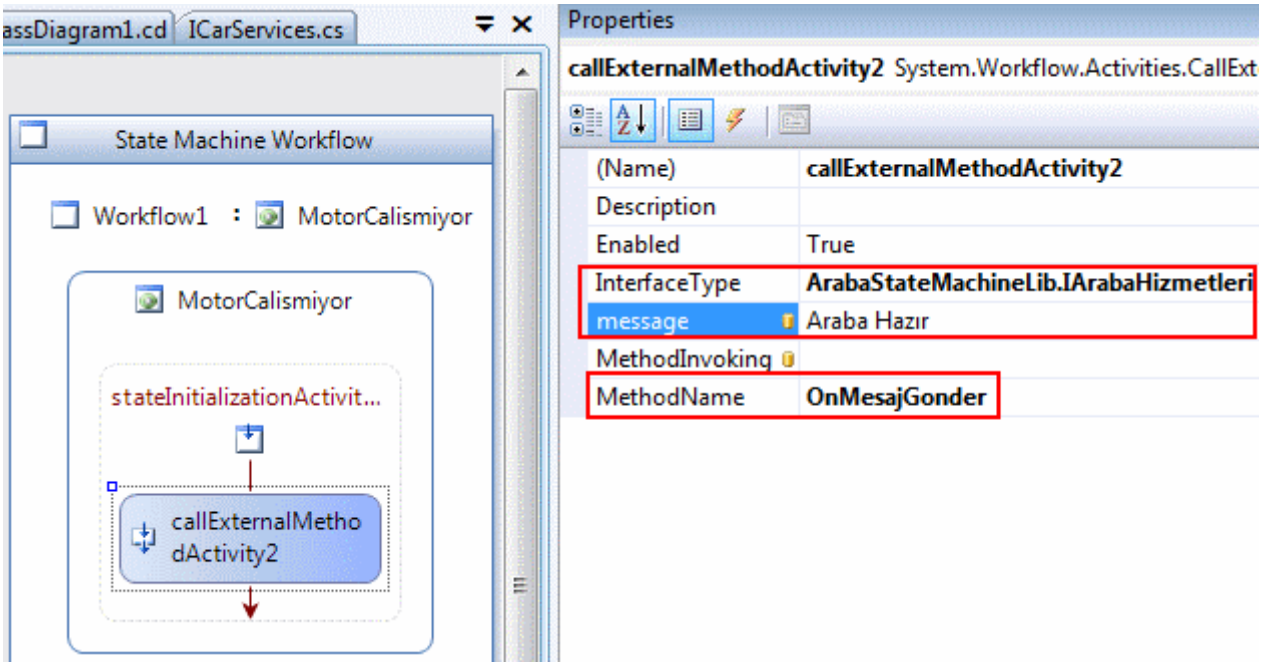
Gelelim **CallExternalMethodActivity** bileşenine. Bu bileşen içinde yine arayüz(Interface) seçimi yapılmalıdır. **InterfaceType** özelliğine yapılan atamanın ardından çalıştırılacak olan harici metodun adı **MethodName** özelliğinden seçilir. örnekte harici metodun aldığı **string** bir parametrede söz konusudur. Bu parametrede **message** isimli özelliğe atanan değer ile belirtilir.



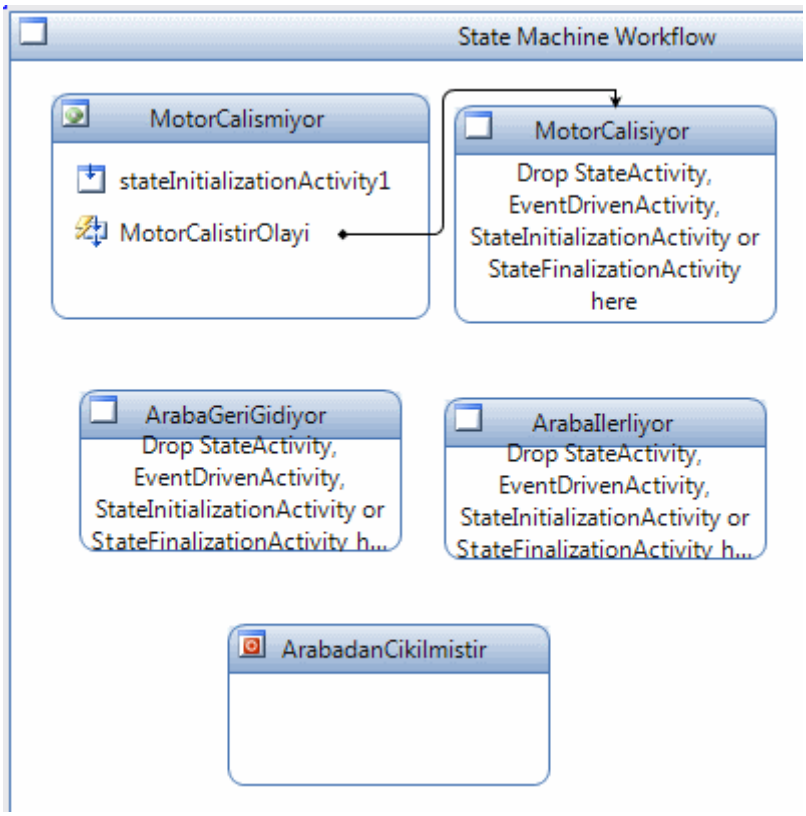
Burada dikkat edilmesi gereken noktalardan biriside, **OnMesajGonder** metodunun parametrik yapısına uygun olacak şekilde bir özelliğin **IDE**' deki **Properties** penceresine eklenmiş olmasıdır. örnekte message isimli olan parametre, özellik penceresine birer bir aynı olacak şekilde gelmiştir. Harici metod çağırılmasınıda tamamladıktan sonra geçilecek olan durum bileşenini belirlemek gerekmektedir. Bunun içinde **SetStateActivity** bileşeninin **TargetStateName** özelliğine **StateActivity** adının atanması yeterlidir. İlk durumda aracın motoru çalıştırıldıktan sonra MotorCalisiyor durumuna(State) geçilmektedir.



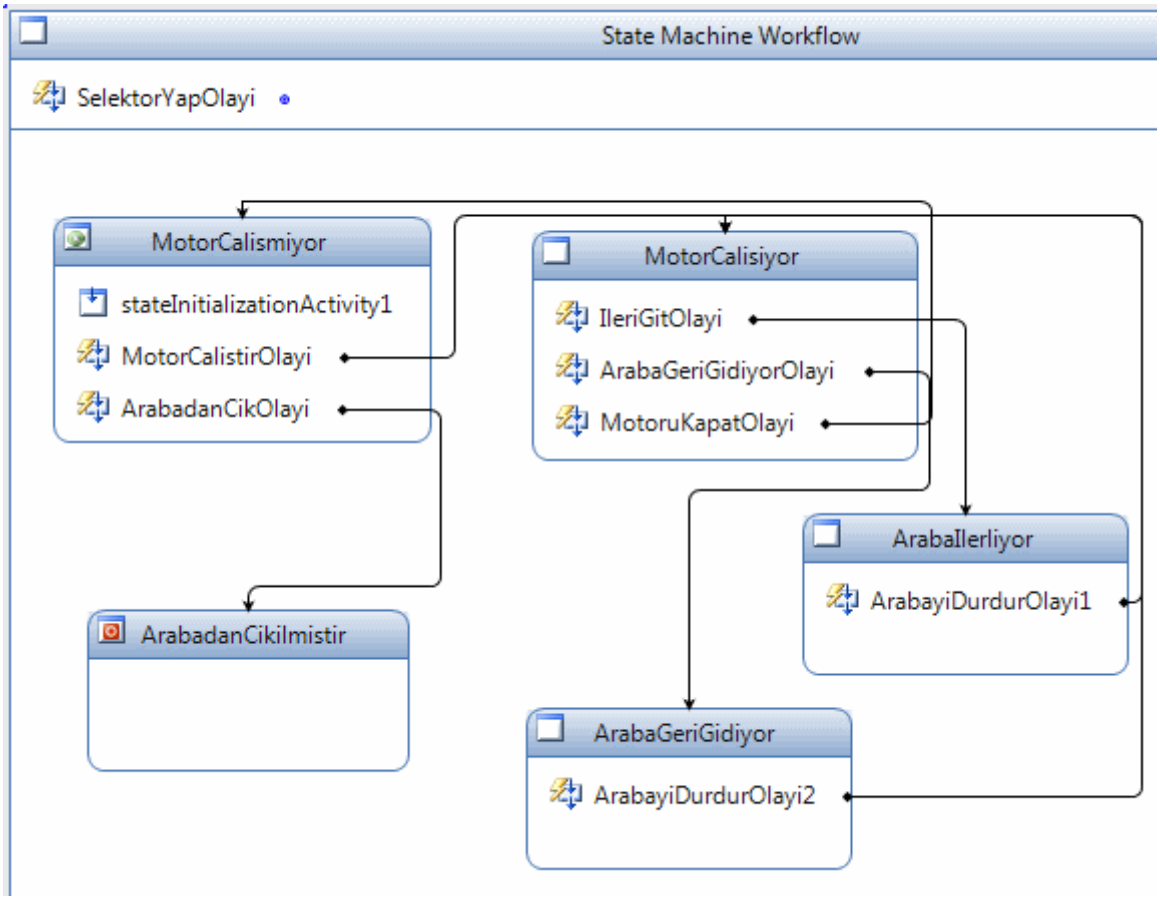
İstenirse ilk durum için **StateInitializationActivity** bileşeni de eklenebilir. Böylece makinenin ilk konumdaki durumu için gerekli hazırlıkların yapılması sağlanabilir. Söz gelimi örnek senaryoda**OnMesajGonder** metodunun harici olarak çağırılması ve mesaj olarakta "Araba hazır" denilmesi sağlanabilir. Bunun için **MotorCalismiyor** aktivitesi içerisine bir adet **StateInitializationActivity**bileşeni atanması ve bu bileşenin içerisinede bir adet **CallExternalMethodActivity** bileşeni eklenerek **InterfaceType**, **MethodName** ve **message** özelliklerinin değerlerinin belirlenmesi yeterlidir.



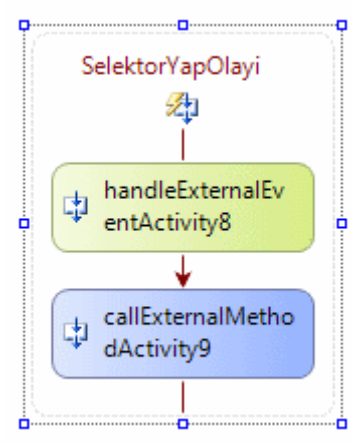
Böylece ilk durum tamamıyle hazırdır. Akışın şu andaki görüntüsü aşağıdaki gibi olacaktır.



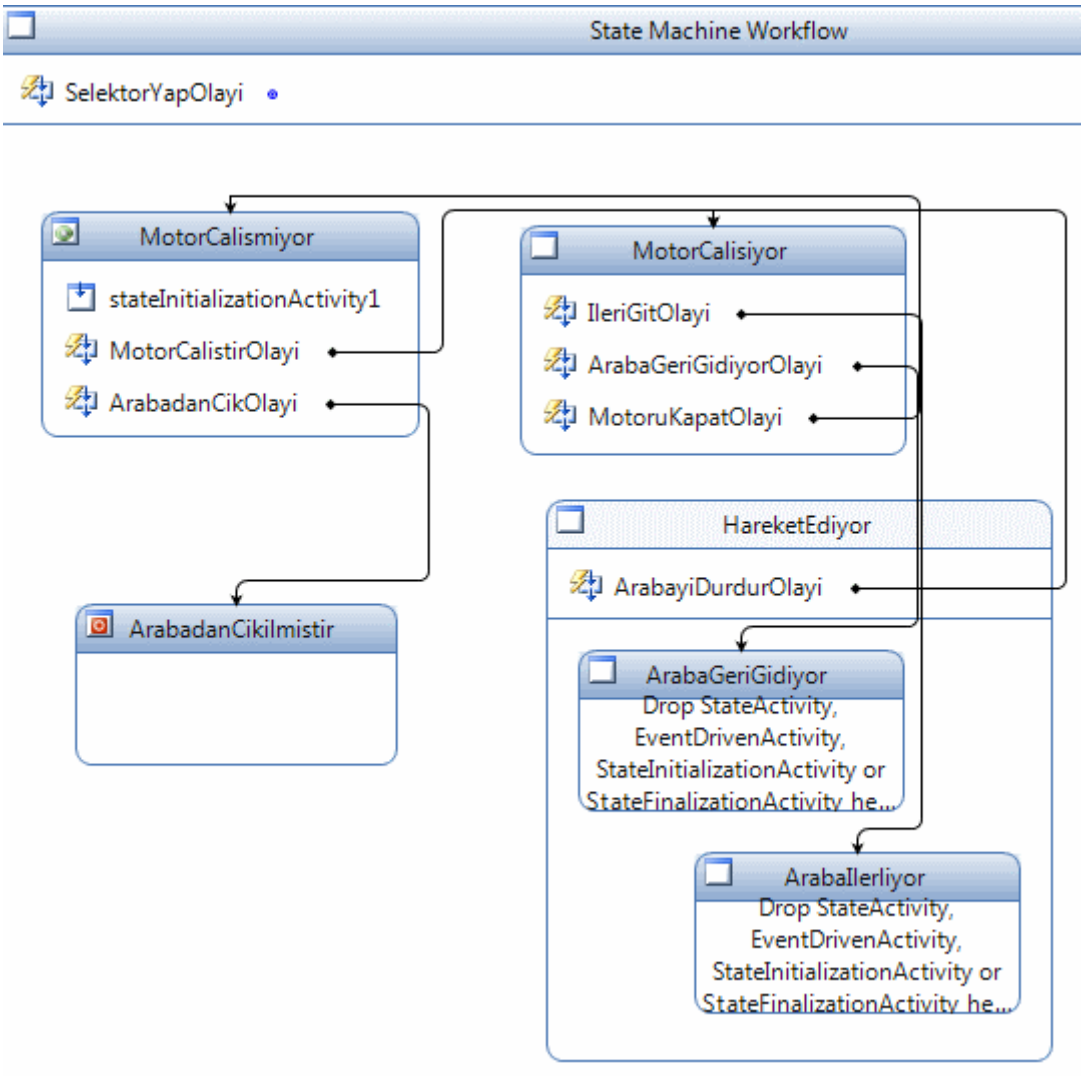
Burada yapmış olduğumuz adımların aynılarını diğer durumlar içinde gerçekleştirmeliyiz. Makalemizin dahada uzamaması için buradaki adımların gösterilmesini atlıyoruz. Gereken ayarlamalar yapıldıktan sonra iş akışının son hali aşağıdaki ekran görüntüsündeki gibi olmalıdır. Dikkat edileceği üzere olası durum geçişleri ince ok çizgiler ile daha belirgin haldedir.



Burada ekstradan selektör yapma durumununda, **State Machine** üzerinde ayrı bir **EventDrivenActivity** olarak tanımlanması gereklidir. Bu aktivite içerisinde sadece **HandleExternalEventActivity** ve**CallExternalMethodActivity** bileşenlerinin kullanılması yeterlidir. Dikkat edileceği üzere **SetStateActivity** kontrolünün kullanılması gerekli değildir. Nitekim selektör yapma olayının arkasından geçilecek herhangibir durum göz önüne alınmamaktadır. Bu nedenle **SelektorYapOlayı** isimli **EventDrivenActivity** içerisinde aşağıdaki bileşenlerin tasarlanması yeterlidir.



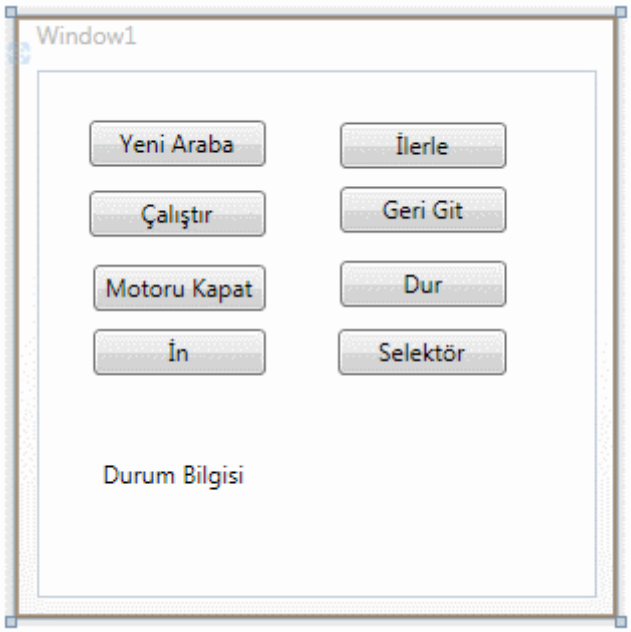
Host uygulamaya geçmeden önce şu durumda göz önüne alınmalıdır. Tasarım penceresine bakıldığında **ArabaGeriGidiyor** ve **ArabaIlerliyor StateActivity** bileşenleri içerisinde aynı **EventDrivenActivity** nesnelerinin kullanıldığı görülmektedir ki buda aracı durdurma olayını işaret etmektedir. Bu nedenle makalemizin başındada belirttiğimiz gibi bu **StateActivity**bileşenlerinin ortak bir **StateActivity** içerisine alınması düşünülebilir. Bu sebepten tasarım ekranına ortak bir **StateActivity** bileşeni sürüklenip, diğerlerini içine alması sağlanmalıdır. Aşağıdaki ekran görüntüsü bu durumu açık bir şekilde ifade etmektedir.



öncelikli
olarak **HareketEdiyor** isimli **StateActivity** içerisine **ArabaGeriGidiyor** ve **ArabaIlerliyo r** isimli **StateActivity** nesneleri sürüklenmiştir. Sonrasında ise bunlardan herhangibirisinde yer
alan**ArabayiDurdurOlayi** isimli **EventDrivenActivity** bileşeni **HareketEdiyor** isimli **Sta teActivity** içerisine çıkartılmış ve diğerininki silinmiştir.

Artık host uygulamanın yazılmasına başlanabilir. örneğimizde **host** uygulaması basit bir **WPF(Windows Presentation Foundation)** programı olarak tasarlanacaktır. Söz

konusu WPF uygulamasının **State Machine Workflow kütüphanesi(Library)** dışında, **System.Workflow.Activities, System.Workflow.Components** ve **System.Workflow.Hosting assembly**' larınıda referans etmesi gerekmektedir. YarisPisti isimli **WPF** uygulamamızın **Window1** penceresine ait ekran görüntüsü ve **XAML(eXtensible Application Markup Language)** içeriği ise aşağıdaki gibidir.



XAML içeriği;

```
<Window x:Class="YarisPisti.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Window1"
Height="300" Width="300">
   <Grid>
      <Label Height="42" Margin="28,0,22,32" Name="lblGelenMesaj"
VerticalAlignment="Bottom" Content="Durum Bilgisi"></Label>
      <Button Height="23" HorizontalAlignment="Left"
Margin="26,25,0,0" Name="btnYeniAraba" VerticalAlignment="Top"
Width="88" Click="btnYeniAraba_Click">Yeni Araba</Button>
      <Button Height="23" Margin="26,60,0,0" Name="btnMotoruCalistir"
VerticalAlignment="Top" Click="btnMotoruCalistir_Click" HorizontalAlignment="Lef
t" Width="88">çalıştır</Button>
      <Button Height="23" HorizontalAlignment="Left" Margin="28,97,0,0"
Name="btnMotoruKapat" VerticalAlignment="Top"
Width="86" Click="btnMotoruKapat_Click"> Motoru Kapat</Button>
      <Button HorizontalAlignment="Right" Margin="0,129,44,110"
Name="btnSelektorYap"
Width="84" Click="btnSelektorYap_Click">Selektör</Button>
```

```
        <Button Height="23" HorizontalAlignment="Right" Margin="0,26,44,0"
Name="btnIlerle" VerticalAlignment="Top"
Width="83" Click="btnIlerle_Click">İlerle</Button>
        <Button Height="23" HorizontalAlignment="Right" Margin="0,58,44,0"
Name="btnGeriGit" VerticalAlignment="Top"
Width="83" Click="btnGeriGit_Click">Geri Git</Button>
        <Button Height="23" HorizontalAlignment="Right" Margin="0,95,44,0"
Name="btnDur" VerticalAlignment="Top"
Width="83" Click="btnDur_Click">Dur</Button>
        <Button HorizontalAlignment="Left" Margin="28,129,0,110"
Name="btnArabadanIn" Width="86" Click="btnArabadanIn_Click">İn</Button>
    </Grid>
</Window>
```

Window1 penceremizin kod içeriği ise aşağıdaki gibidir.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using ArabaStateMachineLib;
using System.Workflow.Runtime;
using System.Workflow.Activities;

namespace YarisPisti
{
    public partial class Window1 : Window
    {
        // Workflow çalışma ortamı için gerekli nesne tanımlanır
        WorkflowRuntime _wf;
        ArabaYerelServisi _arabaServisi; // Yerel Servis nesnesi tanımlanır
        Guid ornekId = Guid.Empty; //ExternalDataEventArgs sınıfı parametre olarak Guid
almaktadır. Bu sebepten ornekId isimli bir değişlen tanımlanmıştır.

        public Window1()
        {
```

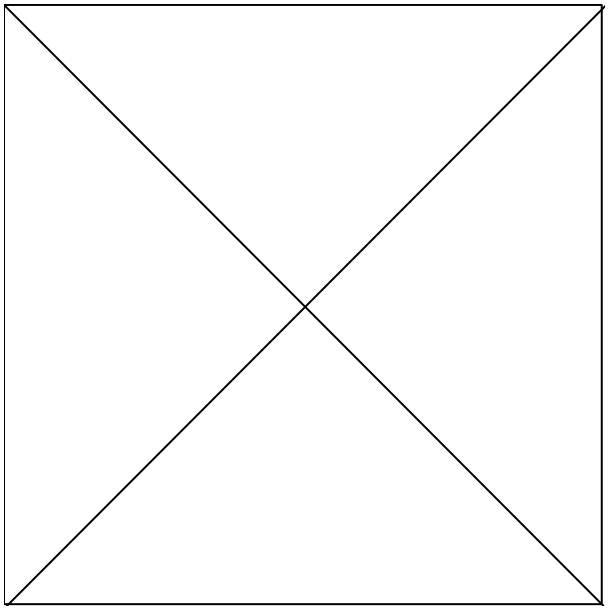
```
        InitializeComponent();

        _wf = new WorkflowRuntime(); // çalışma zamanı oluşturulur
        _wf.StartRuntime(); // WF çalışma zamanı başlatılır
           ExternalDataExchangeService excSrv = new
ExternalDataExchangeService(); // Bir adet ExternalDataExchangeService servisi
oluşturulur ve şu anki WF çalışma zamanına AddService metodu ile eklenir.
        _wf.AddService(excSrv);

        // Yerel Servis(Local Servis) nesnesi örneklenir.
        _arabaServisi = new ArabaYerelServisi();
        // Host üzerinden ele alınacak MesajAlindi olayı yüklenir.
        _arabaServisi.MesajAlindi+=new
EventHandler<MesajAlindiEventArgs>(_arabaServisi_MesajAlindi);
        // Yerel servis ExternalDataExchangeService örneğine eklenir
        excSrv.AddService(_arabaServisi);
    }

    // Label içeriğinin güncellenmesi için aşağıdaki gibi Invoker kullanımı gereklidir. Aksi
takdirde çalışma zamanında istisna alınır.
    private delegate void GuncellemeTemsilcisi();
    void _arabaServisi_MesajAlindi(object sender, MesajAlindiEventArgs e)
    {
        ornekId = e.InstanceId;
        GuncellemeTemsilcisi dlg = delegate()
        {
            lblGelenMesaj.Content = e.Bilgi.ToString();
        };
        // Normal Windows uygulamalarında this.Invoke ile çağırabilmemiz mümkünken
WPF uygulamalarında Dispatcher nesnesinden yararlanılmaktadır
        Dispatcher.Invoke(System.Windows.Threading.DispatcherPriority.Normal,
dlg);
    }

    // Yeni araba aslında Workflow1 tipinden yeni bir State Machine Workflow örneği
oluşturulmasını sağlamaktadır.
    private void btnYeniAraba_Click(object sender, RoutedEventArgs e)
    {
        WorkflowInstance wfOrnegi = _wf.CreateWorkflow(typeof(Workflow1), null);
        wfOrnegi.Start(); // State Machine Workflow başlatılır
        ornekId = wfOrnegi.InstanceId; // ExternalDataEventArgs' ta kullanılmak üzere
InstanceId değeri alını ve GUID tipinden olan ornekId değişkenine atanır.
    }

    // Olaylar için gerekli argümanların alınması sağlanan metod
```

```
    private ExternalDataEventArgs ArgumanAl()
    {
       ExternalDataEventArgs args = new ExternalDataEventArgs(ornekId);
       args.WaitForIdle = true;
       return args;
    }

    private void btnIlerle_Click(object sender, RoutedEventArgs e)
    {
       try
       {
          _arabaServisi.OnIlerle(ArgumanAl()); // Servis üzerinden ilgili olay metodu
tetiklenir
       }
       catch (Exception ex)
       {
          MessageBox.Show(ex.Message);
       }
    }

    private void btnMotoruCalistir_Click(object sender, RoutedEventArgs e)
    {
       try
       {
          _arabaServisi.OnArabayiCalistir(ArgumanAl());
       }
       catch (Exception ex)
       {
          MessageBox.Show(ex.Message);
       }
    }

    private void btnGeriGit_Click(object sender, RoutedEventArgs e)
    {
       try
       {
          _arabaServisi.OnGeriGit(ArgumanAl());
       }
       catch (Exception ex)
       {
          MessageBox.Show(ex.Message);
       }
    }

    private void btnMotoruKapat_Click(object sender, RoutedEventArgs e)
```

```
{
    try
    {
        _arabaServisi.OnMotoruDurdur(ArgumanAl());
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void btnDur_Click(object sender, RoutedEventArgs e)
{
    try
    {
        _arabaServisi.OnDur(ArgumanAl());
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void btnArabadanIn_Click(object sender, RoutedEventArgs e)
{
    try
    {
        _arabaServisi.OnArabadanCik(ArgumanAl());
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
}

private void btnSelektorYap_Click(object sender, RoutedEventArgs e)
{
    try
    {
        _arabaServisi.OnSelektorYap(ArgumanAl());
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message);
    }
```

```
        }
    }
}
```

Uygulamamızı çalıştırdığımızda aşağıdaki video görüntüsündeki gibi **State Machine Workflow** başarılı bir şekilde yüklendiği ve çalıştığı görülmektedir. Tabiki geçiş yapılamayan durumlarda söz konusudur. örneğin araba ileri gidiyorken geri gidiyor durumuna geçilmesi söz konusu değildir. Bu tip hallerde, **try...catch** blokları devreye girerek üretilen i**stisna(Exception)** mesajları MessageBox içerisinde görülecektir.



Böylece geldik bir uzun makalemizin daha sonuna. Bu makalemizde **State Machine Workflow** tipinden iş akışlarını kısaca ne olduklarını, nasıl tasarlandıklarını incelemeye çalıştık. Bunu yaparken StateActivity, EventDrivenActivity, HandleExternalEventActivity, SetStateActivity, CallExternalMethodActivity vb aktivite tiplerinden bir kaçına değinme fırsatımızda oldu. Ayrıca**durumlar(States)** arası geçişleri sağlayan olayların **yerel bir servis(Local Service)** içerisinde nasıl geliştirilebileceğini gördük. Son olarakta geliştirilen **Workflow** projesini bir **WPF** host uygulamasında yürütmeyi inceledik. örnek senaryo olarak **APress** yayınlarından olan ve **Bruce Bukovics** tarafından yazılan **Pro WF** kitabında yer alan **CarService** iş akışının daha basit bir versiyonunu adım adım açıklamalı olarak örneklemeye çalıştık. özel olarak host uygulamayı WPF üzerinde geliştirdik. Böylece **Window** uygulamalarında kullandığımız **method invoker** kavramının burada **Dispatcher** özelliği üzerinden ele alınabileceğini görme fırsatını elde ettik.*(Dispatcher kavramına ilerleyen makalelerimizde değinmeye çalışıyor olacağım)* **State Machine Workflow** tipinden iş akışlarının, bu senaryo dışında gerçek .**Net** nesneleri içerisindeki kullanımını araştırmanızı ve örneklemeye çalışmanızı şiddetle tavsiye ederim. İlerleyen makalalerimizde **Windows Workflow Foundation** ile ilgili farklı konulara değinmeye devam ediyor olacağız. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

StateMachineOrnegi.rar (90,12 kb)

İlk Bakışta Windows Workflow Foundation (2008-01-01T16:39:00)

wf,

Gerçek dünyada pek çok iş probleminin çözümünde **iş akışlarından(Workflow)** yararlanılır. Temelde bir iş probleminin çözümünde veya amacının gerçekleştirilmesinde izlenen yol birdir. önce problem yönetilebilir küçük parçalara bölünür. Bu parçalar, gerçekleştirilmesi gereken **görevler(Tasks)** olarak düşünülebilir. Her bir görevin(Task) içerisinde ona ait gerçekleştirilmesi gereken ne varsa **adımlar(Steps)** halinde tasarlanır. Bu adımlar dahil oldukları görevin tamamlanmasında rol oynarlar. Adımlar arasındaki geçişler basit olabileceği gibi çeşitli çevresel koşul veya faktörlerede bağımlı olabilir. Bir başka deyişle adımlar arası geçişlerde **koşullar(Conditions)** söz konusu olabilir. Adımlar düzenli bir sırada olup aralarındaki geçişler önceden tanımlanmış ve belirli olabileceği gibi, çeşitli olaylara göre farklı şekillerde ele alınabilirlerde. Sonuç olarak ortaya iş probleminin çözümü için tasarlanmış bir **süreç(Process)** ve **kontrollü bir akış(Control Flow)** çıkar.

İş akışları(Workflow) sayesinde, iş problemlerinin çözümlenmesi, istenirse genişletilebilmesi son derece kolay bir şekilde gerçekleştirilebilmektedir. Bir iş akışına pek çok yerde kolaylıkla rastlayabiliriz. Bunların bir kısmı için hiç bir geliştirme yapılamamakla birlikte çoğu için bilgisayar teknolojisinden yararlanılmaktadır. Bir başka deyişle bazı iş çevrelerinde izlenen süreçler bilinçsiz olarak kendiliğinden bir akışa sahip olabilmektedir. Ancak bilgisayar teknolojisinin hızla yaygınlaşması ve verimliliği arttırması nedeni ile, iş akışları yazılımsal ve donanımsal faktörler üzerinden ele alınmaktadır. Bu anlamda **iş akışlarının(Workflow)** ilk uygulamaları dökümanların bir noktadan başka bir noktaya taşınması olmuştur. Zaten buda **SharePoint Server** gibi gelişmiş bir sistemde iş akışlarının neden kullanıldığını açıklamaktadır. Nitekim portal tarzı uygulamalarda**döküman yönetimi(Document Management)** ve paylaşımı esastır. Bu noktada, dökümanların **kullanıcılar(Users)** ve **sistemler(Systems)** arasındaki hareketinde çeşitli onay mekanizmalarının devreye girmesi muhtemeldir. Buda çok doğal olarak bir iş akışı ile ifade edilip tanımlanabilir.

Bir kaç yıl önce çalıştığım özel bir yazılım firmasında şirketlerin **iş akışlarının(Workflow)** tasarlanabildiği bir yazılım projesinde görev almıştım. İş çevrelerinin çözüm bekleyen çok fazla sayıda problemi vardır. Söz gelimi bir elemanın işe alım yada işten çıkartılma süreçleri, şirketin mali yapısını gösteren raporların onay mekanizmaları, üretim hattına ait süreçler ve daha pek çoğu. Bilgisayar ve yazılımlar sayesinde bu örnek süreç ve benzelerine ait iş akışlarının kullanılması son derece kolaylaşmaktadır. örneğin **işe alım sürecini** ele alalım. Elemanın CV' sinin insan kaynakları departmanına verilmesi ve bilgisayar ortamına alınması, departman içerisinde CV' nin ilgili mercilere gönderilerek onaylarının alınması, onay verilmesi halinde işe alınmak istenen personelin görüşmeye çağırılması, görüşme sonrası tüm bilgilerin kayıt altına alınarak sürece dahil olan diğer kişilerede gönderilmesi, gönderme işlemlerinde mail sisteminden yararlanılması gibi işlemler söz konusu olacaktır. Bu işlemlerin her biri arasındaki geçişler karar yapıları ve onay mekanizmaları ile gerçekleşmektedir. Söz gelimi

CV' nin ilk değerlendirmesinde en üst mercinin red etmesi halinde iş başvurusunda bulunan kişiye olumsuz cevap verilmesi ve süreç içerisindeki ilgili kişilerede bu durumun mail, sms gibi yollarla aktarılması için bir onay mekanizması ve koşullandırmanın olması gerekmektedir.

Dikkat edilecek olursa iş akışılarının yukarıdaki gibi cümlesel olarak ifadesi anlaşılmasını zorlaştırmaktadır. Hatta bu sürecin kapsadığı **alana(Domain)** dahil olan kişilerin bilgisayarlarında adımları kolayca izleyebilmesi ve kimin üstüne hangi **görev(Task)** düşüyorsa bunu yapabilmesi demek aslında bir yazılım sisteminin geliştirilip kurulması anlamına gelmektedir. Buda doğal olarak yukarıdakinden daha uzun ve karmaşık olan iş akışı anlatımlarının önce kağıt üzerinde grafiksel olarak tasarlanması ve sonrasında gerekli yazılımın hazırlanması anlamına gelmektedir. **Geliştiriciler(Developers)** bu tip iş akışlarını sistemlere uygularken **görsel tasarlayıcılarıda(Visual Designer)** ele alırlar. Bir başka deyişle iş akışlarının görsel olarak tasarlanabilmeside önemlidir. Bu amaçla geliştirilmiş pek çok **yazılım(Software)** söz konusudur.

NOT : *Karmaşık iş kurallarını içeren akışların görsel olarak ele alınması, akışın içerisinde yer alan* ***adım(Step)*** *ve* ***kuralların(Rule)*** *kolayca tasarlanabilmesi hatta kodlanabilmesi demektir. Bu bir iş akışının sonradan kolayca değiştirilebilmesi bir başka deyişle genişletilebilmesininde kolaylaştırılması anlamına gelmektedir.*

Bu yazılımların genel amacı, pek çok iş akışının çoğunlukla birden fazla bilgisayarın olduğu sistemlerde kurulması ve kullanılabilmesidir. Elbette birden fazla bilgisayar olması şart değildir. Bazı durumlarda tek bir bilgisayar üzerindeki programlar için söz konusu olabilecek iş akışlarıda var olabilir. *(Aslında* ***Windows Workflow Foundation*** *mimarisinin bu modeldeki iş akışlarına daha yakın olduğunu düşünebiliriz.)*

Bu kısa bilgilerden sonra bir iş akışını tanımlamak çok daha kolaylaşmaktadır. Bir **iş akışı(Workflow)** herhangibir iş probleminin çözümü için gereken **adımları(Steps),** onay mekanizmalarını ve **karar yapılarını(Condition)** içeren bir model sunmaktadır. Bir başka deyişle bir iş akışı, belirli **kurallar(Rules)** üzerine sıralanmış adımlar topluluğu olarakta düşünülebilir. özellikle bilgisayar teknolojileri üzerinden baktığımızda bir iş akışının aslında farklı bir programlama modeli sunduğuda göz önüne alınabilir. Bu açılardan bakıldığında iş akışı denildiğine akla gelen pek çok kavramda bulunmaktadır. Bu kavramlar aşağıdaki maddelerde belirtildiği gibidir;

- Görsel Şemalar
- Kurallar(Rules)
- Politikalar(Policies)
- Sisteme giren(Input) ve çıkan(Output) veriler
- Kişiler(Users)
- Organizasyonlar(Organizations)
- Yordamlar(Procedures)
- Temel Görevle(Tasks)

- Adımlar(Steps)
- Aktiviteler(Activities)

Peki **Windows Workflow Foundation** ile kastedilen nedir? **Microsoft** bu Foundation ile iş akışlarının tasarlandığı bir programmı üretmiştir? Aslında Windows Workflow Foundation **tek bir iş alanı(Single Domain)** içerisinde yer alan **tek bir uygulamayı(Single Application)** hedeflemektedir. Bir başka deyişle Windows Workflow Foundation **.Net** uygulamalarının kullanabileceği**iş akışlarının(Workflow)** tasarlanması için gerekli altyapıyı sunan bir **Framework 3.0** yaklaşımıdır.

NOT : *İş akışları çoğunlukla* ***BizTalk Server****' un sundukları ile karşılaştırılır. BizTalk ile özellikle elektronik ticarete uygun olacak şekilde farklı platformlar üzerinde yer alan sistemlere ait iş süreçleri başarılı bir şekilde ele alınabilmektedir. Oysaki* ***Windows Workflow Foundation*** *sadece* ***işletim sistemi seviyesinde(Operating System Level)****düşünülmüştür.*

Bu anlamda Microsoft otoriteleri ***BizTalk****' un* ***interapplication(Birden fazla uygulama-Mutliple Applications)*** *olarak ele alınması gerektiğini,* ***Windows Workflow Foundation****' ın ise****intraapplication(Tek bir uygulama-Single Application)*** *şeklinde düşünülmesi gerektiğini vurgulamaktadır. Ancak bu bir kısıt değildir. Nitekim* ***WWF*** *içerisinde* ***Web Servisleri*** *gibi****SOA(Service Oriented Architecture)*** *modelleri sayesinde dış platformlara çıkılması ve iş sürecinin bu şekilde genişletilmeside mümkündür.*

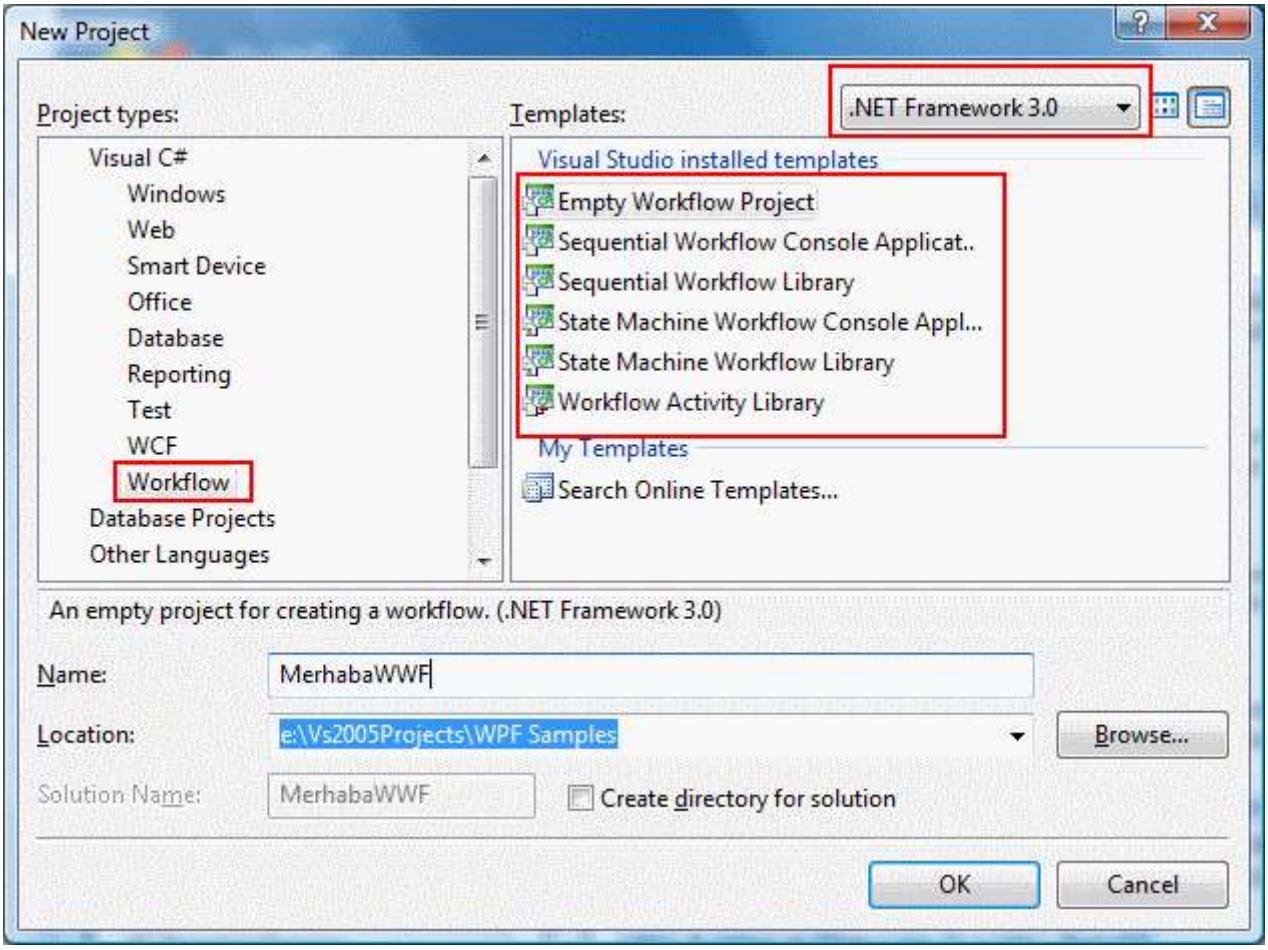
Windows Workflow Foundation mimarisi sayesinde iş akışları görsel olarak tasarlanıp kodlanabilirler. Ancak tasarlanan bu iş akışlarının işe yarayabilmesi için bir uygulama tarafından ele alınmaları şarttır. Söz konusu uygulamalar host görevini üstlenmekte olup bir veya daha fazla iş akışını barındırıp kullanabilirler. Windows Workflow Foundation mimarisi pek çok fayda sağlamaktadır. Söz konusu faydalar aşağıdaki maddeler ile özetlenebilir.

- İş akışlarının kolayca ve etkili bir biçimde tasarlanabilmesi için **görsel tasarımcı(Visual Designer)** sunar.
- Sistemin insan ile etkileşimde olduğu modellerde **aktiviteler(adımlar)** arasındaki süre farkları olabilir. Bu nedenle iş akışının güncel durumlarının kaydedilebiliyor ve daha sonra sonra başka bir zaman diliminde tekrar yüklenebiliyor olması gerekir. WWF bunu sağlamaktadır.
- İş akışları için gerekli **çalışma zamanı(Run-Time)** ortamı dışında, pek çok tipte farklı **aktivite(Activity)** için destek, aktivitelerin **denetlenmesi(Monitoring)**,**izlenmesi(Tracing)** sağlanmaktadır.
- **Sequential** iş akışı desteği vardır. Bu akışlar çoğunlukla sistem etkileşimi olan bir başka deyişle insan faktörü fazla bulunmayan durumlarda söz konusudur. Bu tip akışlarda adımların sırası ve düzeneği bellidir.
- **State Machine** iş akışı desteği vardır. Bu tip akışlarda insan etkileşimi söz konusudur. çoğunlukla aktivitiler arasındaki geçişlerin bazı olayların tetiklenmesine bağlı olduğu durumlarda kullanılır.

Windows Workflow Foundation, iş akışlarını esas alan bir programlama modeli sunmaktadır. Söz konusu programlama modeli **deklaratif(declarative)** yaklaşımı ele almaktadır. Buna göre iş mantığı ayrık bileşenler içerisinde **kapsüllenir(encapsulation)**. Nitekim bileşenler arasında akışların nasıl yölendirileceğini belirten kurallar deklaratif olarak tanımlanabilirler.

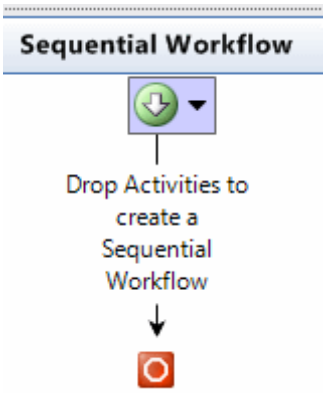
Windows Workflow Foundation mimarisinin nasıl kullanıldığını daha net kavrayabilmek için basit örnekler üzerinden devam etmekte yarar var. İlk örnekte son derece basit bir iş akışını tasarlayıp bunu kullanacak olan bir **Console** uygulaması geliştiriyor olacağız. Sonrasında ise iş akışını farklı **.Net** uygulamalarında da kullanabilmek adına bir **sınıf kütüphanesi(Class Library)** içerisinde tasarlayıp örnek bir Windows uygulaması içerisinden çalıştıracağız. Daha öncedende belirtildiği gibi **WWF** içerisinde tasarlanmış bir **iş akışının(Workflow)** işe yarayabilmesi için bir **host** uygulama tarafından ele alınması gerekmektedir. İlk örneğimizde **Host** program **Console** uygulaması olarak tasarlanacaktır. örneklerimizi **Visual Studio 2008 RTM** sürümü üzerinde tasarlıyor olacağız. Ancak istenirse gerekli genişletmeler yüklenerek **Visual Studio 2005** ilede WWF geliştirmeleri yapılabilir. Elbette**.Net Framework 3.0**' ın yüklü olması şarttır.

İlk olarak aşağıdaki ekran görüntüsünde olduğu gibi **Workflow** sekmesinde yer alan **proje şablonlarından(Project Templates)** birisinin seçilmesi gerekmektedir.

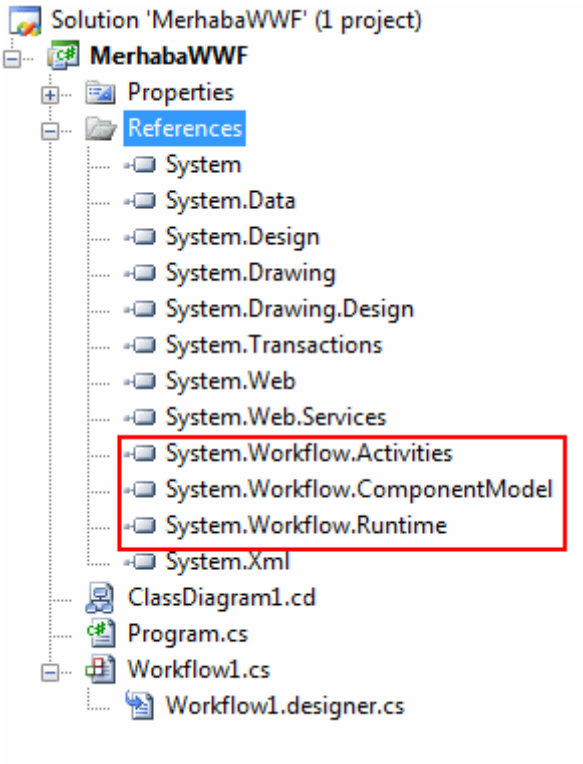


Dikkat edilecek olursa **Sequential** ve **State Machine iş akışlarının(Workflow)** geliştirilmesi için gerekli **proje şablonları(Project Template)** bulunmaktadır. Her iki tip içinde birer **sınıf kütüphanesi(Class Library)** ve **Console** uygulaması şablonu yer almaktadır. çok doğal olarak geliştirilen iş akışlarının farklı tipte **.Net** uygulamalarında kullanılacağı durumlar söz konusu olduğunda **Workflow Library** şablonlarını uygulamak daha doğru bir yaklaşım olacaktır. Windows Workflow Foundation ile geliştirilen iş akışlarında**aktivitelerin(Activity)** büyük önemi vardır. Geliştiriciler isterlerse kendi özel aktivite tiplerinide**(Custom Activity Types)** yazabilirler. Bunun içinde **Workflow Activity Library** şablonu kullanılır.

Makalemizdeki ilk örneğimizde sonuçları hemen irdeleyebilmek adına **Sequential Workflow Console Application** tipinden bir uygulama yazıyor olacağız. Buna ilişkin proje şablonu seçildikten sonra uygulama içerisine aşağıdaki ekran görüntüsünde olduğu gibi **Workflow1** isimli bir iş akışı tipinin eklendiği görülür.

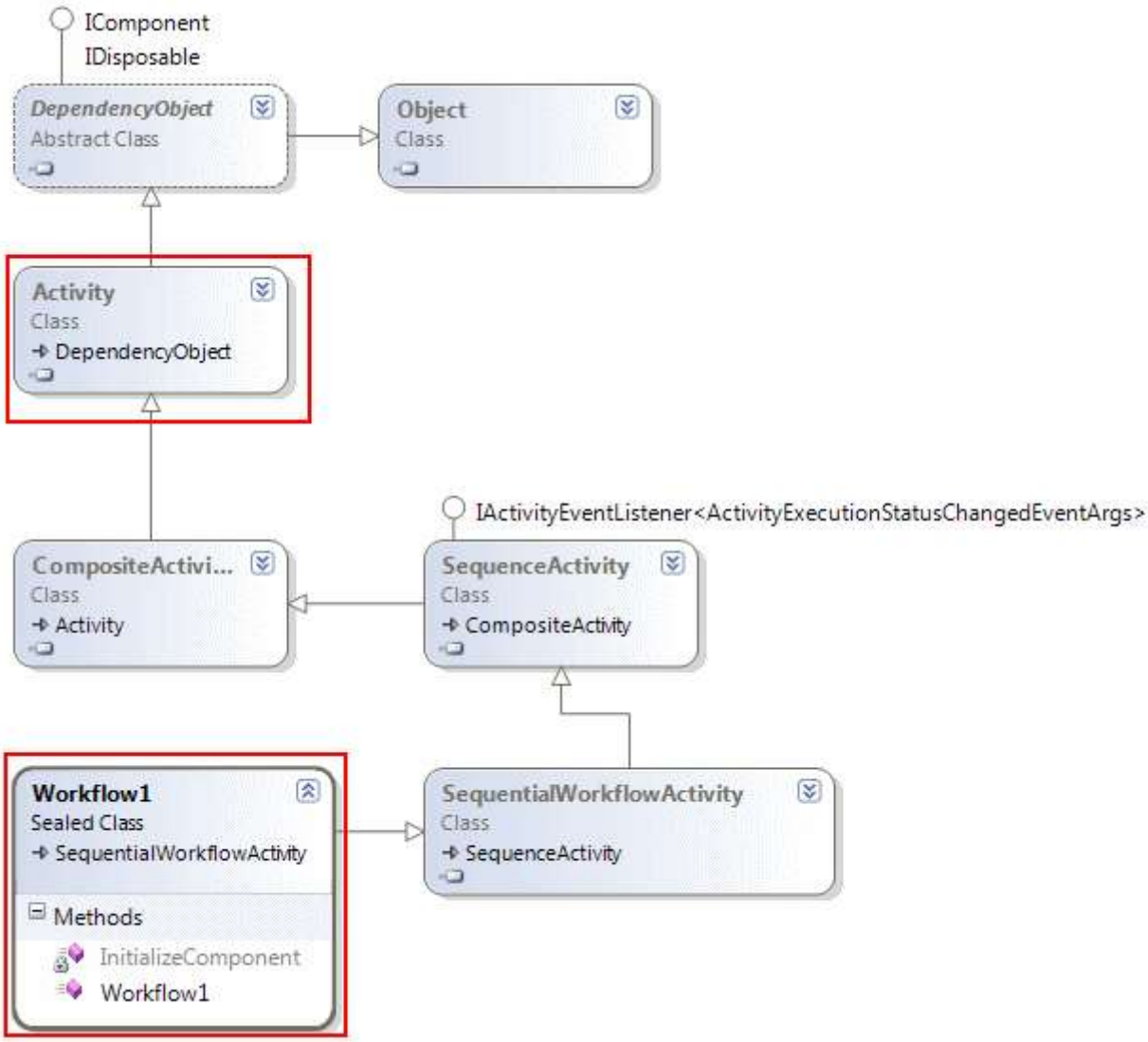


İş akışı içerisine örnek aktiviteleri eklemeden önce proje içerisinde oluşturulan tiplerden bahsetmekte yarar vardır. Herşeyden önce bir iş akışı projesi oluşturulduğunda**System.Workflow.Activities**, **System.Workflow.ComponentModel** ve **System.Workflow.Runtime** isimli **assembly**' ların referans edildiği görülür.



Tahmin edileceği gibi bu **assembly**' lar içerisinde iş akışlarının geliştirilmesi, çalıştırılması, denetlenmesi ve izlenmesi için gerekli temel tipler yer almaktadır. Bunların dışında**console** uygulamasına dahil edilmiş farklı **assembly** referanslarıda bulunmaktadır. Söz gelimi iş akışı içerisinde **transaction** desteğini sağlamak için **System.Transactions**, akış içerisinden web servisleri ile iletişime geçebilmek için **System.Web**, **System.Web.Services** gibi **assembly**' ları örnek olarak gösterebiliriz.

Proje içerisine varsayılan olarak atılan **Workflow1** isimli **sınıfın(Class)** hiyerarşik yapısı ise aşağıdaki **sınıf diyagramında(Class Diagram)** olduğu gibidir.



örnek **Sequential Workflow Console Application** olduğundan, içeride kullanılacak varsayılan iş akışıda **Sequential Workflow** olarak ele alınmaktadır. Bu sebepten dolayı**sealed(kendisinden türetme yapılamaz)** olarak tanımlanmış **Workflow1** isimli sınıf ilk etapta **SequentialWorkflowActivity** sınıfından türemektedir. Ancak enteresan olan bir nokta vardır. örnek geliştirilirken kullanılacak olan standart pek çok akitivite bileşeninin **Activity** isimli sınıfdan türediği görülecektir. Dikkate değer olan, **aktiviteleri(adımları-Steps)** içeren**Workflow** tipinin kendisininde aslında bir aktivite nesnesi olmasıdır. *(Sınıf diyagramında pek çok tip yer almaktadır. Bu tiplerin detaylarını ilerleyen makalelerimizde inceleme fırsatı bulacağız.)*

örneğimizdeki iş akışının içeriğini geliştirmeden önce **Program.cs** dosyası içerisindeki kod parçalarına kısaca bakalım. Şu aşamada **Workflow1** tipine ait **iş akışını(Workflow)** veya başkalarını çalıştıracak ve yürütücek olan kısım **Main** metodunun içeriğidir.

```
using System;
using System.Collections.Generic;
using System.Text;
```

```
using System.Threading;
using System.Workflow.Runtime;
using System.Workflow.Runtime.Hosting;

namespace MerhabaWWF
{
   class Program
   {
      static void Main(string[] args)
      {
         using(WorkflowRuntime workflowRuntime = new WorkflowRuntime())
         {
            AutoResetEvent waitHandle = new AutoResetEvent(false);
            workflowRuntime.WorkflowCompleted += delegate(object
sender, WorkflowCompletedEventArgs e) {waitHandle.Set();};
            workflowRuntime.WorkflowTerminated += delegate(object
sender, WorkflowTerminatedEventArgs e)
                                                  {
                                                     Console.WriteLine(e.Exception.Messa
ge);
                                                     waitHandle.Set();
                                                  };

            WorkflowInstance instance =
workflowRuntime.CreateWorkflow(typeof(MerhabaWWF.Workflow1));
            instance.Start();

            waitHandle.WaitOne();
         }
      }
   }
}
```

Şimdi **Main** metodu içerisinde neler yapıldığına kısaca bir bakalım. **WorkflowRuntime** sınıfı iş akışlarının devreye sokulması, bunların çalışması için gerekli ortamın hazırlanması, çalışma zamanı iş akışlarının izlenmesi, denetlenmesi gibi işlemleri üstlenen önemli bir sınıftır. Pek çok önemli **olayı(Event)** vardır. Bunlardan örnekte hazır olarak sunulan **WorkflowCompleted** olayı iş akışının tamamlanması sonrasında devreye girmektedir. Bir iş akışı işlemleri tamamlandıktan sonra geriye değer döndürebilir. Bu sebepten dolayı **WorkflowCompletedEventArgs** tipinden olan olay parametresinin **OutputParameters** özelliğinden yararlanılarak WorkflowCompleted olay metodu içerisinde sonuç alınması sağlanabilir.Tahmin edileceği gibi **WorkflowTerminated** olayı, herhangibir hata nedeni ile(çoğunlukla bir **istisna-exception**) iş akışı tamamlanamadığında devreye girmektedir. Bu olayla ilişkili olan **WorkflowTerminatedEventArgs** sınıfı üzerinden hareket edilerek

oluşan **istisna(Exception)** referansı **çalışma zamanında(runtime)** yakalanabilir. Dikkat edilecek olursa **Visual Studio** tarafından otomatik olarak üretilen bu kod parçasında olaylara ait fonksiyonelliklerin hazırlanmasında **isimsiz metodlardan(anonymous methods)** yararlanılmaktadır.

Using bloğu içerisinde oluşturulan örneklerden bir diğeride **AutoResetEvent** sınıfına ait nesnedir. Bu sınıf temel olarak **thread** senkronizasyonu yönetimi amacıyla tasarlanmıştır. Host uygulamanın kendi içerisinde bir veya daha çok iş akışını tetiklemesi sonrasında askıda kalması istenen bir durum değildir. Nitekim bir **iş akışı(Workflow)** çalıştırıldığında **Workflow çalışma zamanı(Runtime)** bunu **host** uygulamadaki ana **thread**' in dışında ayrı bir thread içerisine alacaktır. **AutoResetEvent** sınıfının **Set** metodu bu tip bir durumda kalındığında bekleyen thread' in serbest bırakılmasını sağlamaktadır. Dikkat edilecek olursa **Set** metodu hem **WorkflowCompleted** hemde **WorkflowTerminated** olay metoduları içerisinde çağırılmakta ve bekleyen **thread**' in serbest bırakılması sağlanmaktadır.

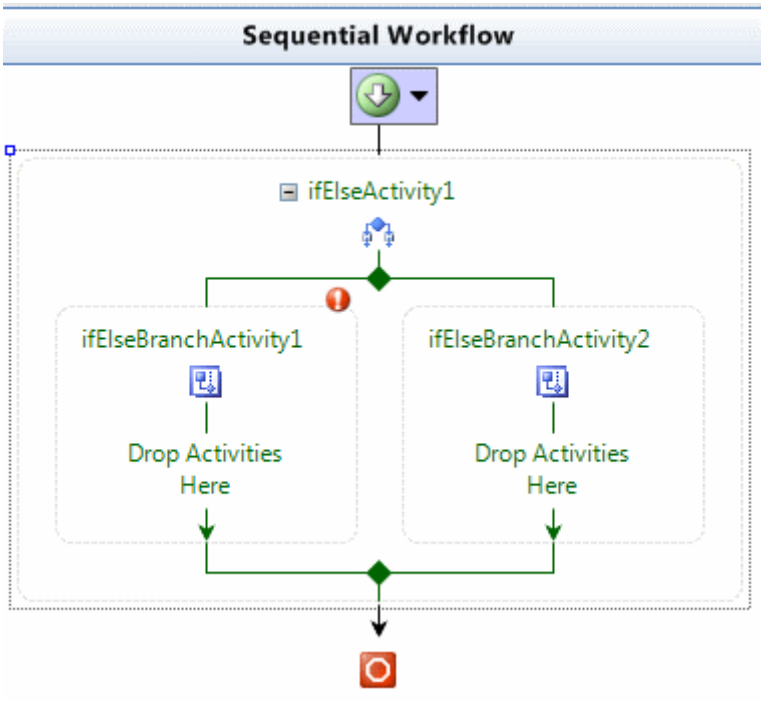
Main metodu içerisinde örneklenen diğer bir nesne sınıfıda **WorkflowInstance**' dır. Bu sınıfta **sealed** olarak tanımlanmıştır. Bir başka deyişle kendisinden türetme yapılamamaktadır. Aynı zamanda **yapıcı metodları(Constructor)** erişimide yoktur. Kendisi ancak **WorkflowRuntime** sınıfının static **CreateWorkflow** metodu ile örneklenebilmektedir. Bu metodun **aşırı yüklenmiş(Overload)** farklı versiyonları bulunmaktadır. özellikle iş akışına dışarıdan parametre değerleri aktarılabilmesini sağlayan versiyonu vardır ki bu önemlidir. Nitekim pek çok iş akışının başlangıcında bir takım dış parametre bilgilerine ihtiyaç vardır. Diğer taraftan bir **XOML(eXtensible Object Markup Language)** dosyasında tanımlı herhangibir iş akışının yüklenmesini sağlayan versiyonuda bulunmaktadır.

Main metodunun sonunda **AutoResetEvent** sınıfına ait nesne örneği üzerinden **WaitOne** fonksiyonu çağırılır. Bu çağrı, çalışan ana **thread**' in bekleyen iş akışları tamamlanıncaya kadar duraksatılması için önemlidir. Nitekim iş akışının içerisindeki adımların gerçekleştirilme sürelerinin belirsiz olması ihtimali vardır. **WaitOne** metodu ilerlenip ilerlenmeyeceğine, diğer **thread**' lerden gelen sinyallere göre karar vermektedir. Tahmin edileceği gibi örnekte yer alan söz konusu sinyal **Set** metodu ile yayınlanmaktadır.

Artık iş akışı içerisine örnek bir **adım(Step)** ekleyerek devam edebiliriz. Daha önceden de belirtildiği gibi adımlar aslında birer **aktivite(Activity)** olarak düşünülebilirler. **WWF**mimarisi çok sayıda hazır aktivite sunmaktadır. Bu aktivite bileşenlerine iş akışına ait tasarım penceresindeylen **ToolBox** kısmından da erişilebilir. Var olan tüm aktivite bileşenleri**System.Workflow.ComponentModel isim alanında(Namespace)** yer alan **Activity** sınıfından(Class) türemektedir. Hazır aktivite bileşenlerine örnek olarak **IfElseActivity**, **CodeActivity**,**CallExternalMethodActivity**, **DelayActivity**, **EventDrivenActivity**, **ReplicatorActivity**, **ParallelActivity**, **TerminateActivity** ve daha pek çoğu verilebilir.

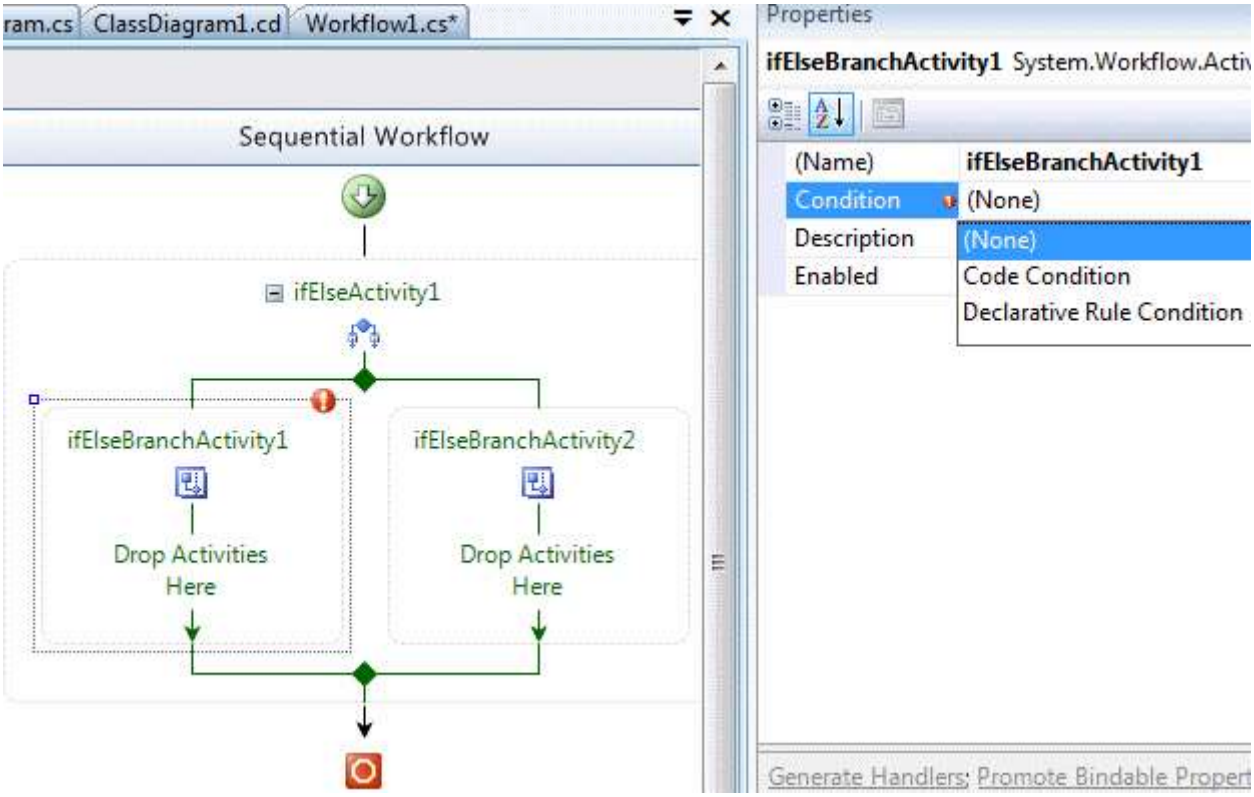
Biz örneğimizde basit olması açısından **CodeActivity** ve **IfElseActivity** bileşenlerini kullanıyor olacağız. CodeActivity bileşeni sayesinde iş akışının herhangibir adımında çalıştırılması istenen kodlar ele alınabilmektedir. Temel olarak bileşenin **ExecuteCode** özelliğine atanan değerin işaret ettiği metod, aktiviteye gelindiğinde çalıştırılacak olan kodları içermektedir. **IfElseActivity** bileşeni yardımıyla bir iş akışının herhangibir noktasına karar yapıları eklenebilmektedir. IfElseActivity bileşeni kendi içerisinde birden fazla**IfElseBranchActivity** örneği içerebilmektedir. Bu IfElseBranchActivity örneklerinin her biri karar yapısı içerisindeki farklı dallanmaları ifade etmektedir.

Dilerseniz örnek üzerinden devam ederek iş akışını tamamlamaya çalışalım. örnek senaryoda bir ürünün stoktaki miktarına göre çalışacak bir iş akışı tasarlayacağız. Söz gelimi stok miktarının belirli bir değerin altına inmesi halinde çalışacak bir iş akışı söz konusu olabilir. Hatta farklı aralık değerlerine göre farklı dallanmalar yapılmasıda sağlanabilir. *(Bu noktada iş akışının ne kadar anlamlı olduğu çok önemli değildir. Nitekim hedeflenen, temel bir iş akışının* **WWF** *altında geliştirilmesi ve kullanılmasıdır.)* İlk olarak tasarım zamanında iken **Workflow1** üzerine bir **IfElseActivity** bileşenini **ToolBox**' tan sürükleyerek bırakalım. Bırakılma işleminden sonra ilk etapta aşağıdaki görüntü ortaya çıkacaktır.

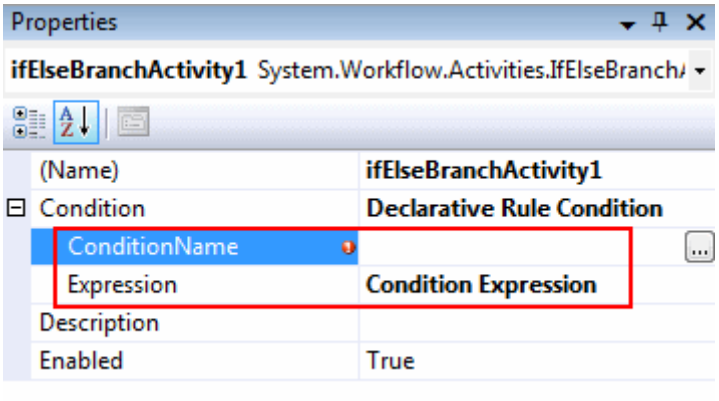


Dikkat edileceği gibi **IfElseActivity** içerisine varsayılan olarak iki adet **IfElseBranchActivity** bileşeni daha eklenmiştir. Bu noktada soldaki IfElseBranchActivity bileşeninin belirtilen koşul **true**olduğunda çalıştırılması, diğerinin ise **false** durumuna karşılık olarak değerlendirilmesi doğru bir yaklaşımdır. Her bir IfElseBranchActivity kendi içerisinde başka aktivitelerin tetiklenmesinede neden olmaktadır. **Drop Activites Here** kısmına bu amaçla çeşitli **aktivite bileşenleri(Activity**

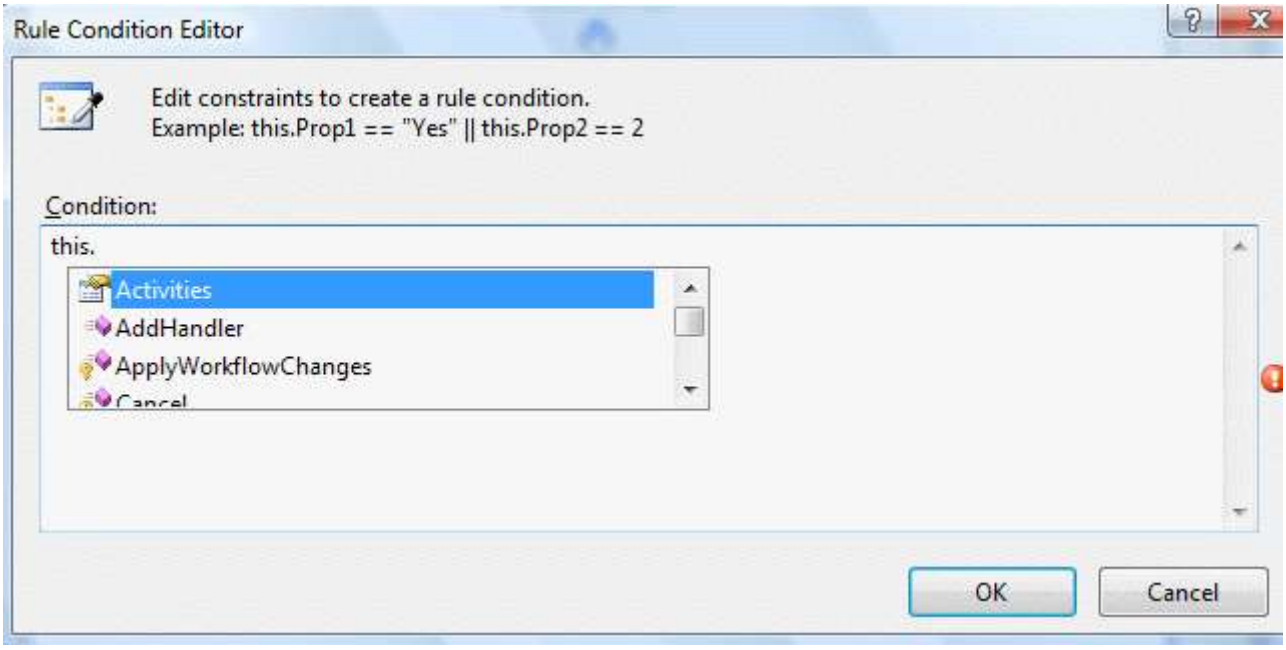
Components) bırakılabilir. IfElseBranchActivity' lerin en önemli üyesi aşağıdaki ekran görüntüsündende görülebileceği gibi **Condition özelliğidir(Property)**.



Condition özelliğine **Code Condition** ve **Declarative Rule Condition** olmak üzere iki farklı değer verilebilir. Peki bunlar ne anlama gelmektedir? Code Condition değerinin atanması halinde koşulun bir metod ile değerlendirileceği belirtilir. Söz konusu metod **bool** bir değerlendirme yapmalıdır. **Declarative Rule Condition** seçeneği sayesinde ise, koşul kodun dışında ayrı bir şekilde ele alınır. Bu çeşit bir kullanım **çalışma zamanında(Run-Time)** yeniden derleme gerektirmeden koşul değişikliği yapma imkanı sunmaktadır. örneğimizde ilk olarak **Declarative Rule Condition** seçeneğini inceliyor olacağız. Bu kriter seçildikten sonra **Condition** özelliği altına iki alt üyenin daha eklendiği görülecektir.



Bunlardan **ConditionName** koşulun adını ifade ederken, **Expression** ise koşula ait ifadeyi içermektedir. ConditionName özelliğine sembolik olarak **Miktar50Altinda** verdiğimizi düşünelim. Bundan sonra **Expression** özelliği yanındaki üç nokta düğmesine tıklarsak aşağıdaki arabirim ile karşılaşırız.



Bu ekranda geriye **true** veya **false** döndürebilecek şekilde bir koşul tanımlaması yapılmaktadır. Dikkat edilecek olursa metin kutusu içeriğinde **intellisense** desteği vardır. Ancak bu noktada iş akışının doğurduğu önemli bir ihtiyaçta ortaya çıkmaktadır. Bir şekilde iş akışına, ürüne ait stok miktarı değerinin aktarılması gerekmektedir. Lakin koşulun değerlendirmesi gereken durum stok miktarının belirli bir değerin altında olması halinde yapılacak işlemler ile ilgilidir. Dolayısıyla iş akışına dışarıdan paramete aktarılması gerekmektedir.

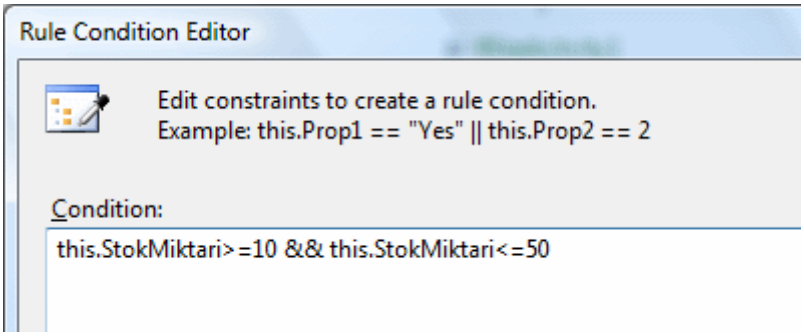
NOT : *Bir iş akışına aktarılacak olan parametreler herhangibir* ***.Net CLR(Common Language Runtime)*** *tipi olabilir. Bu önemli bir avantajdır, nitekim bir* ***iş akışı(Workflow)****önceden tanımlı veya geliştirici tarafından yazılmış herhangibir tip verisi ile başlatılabilir. İş akışlarının herhangibir* ***.Net CLR*** *tipini parametre olarak alabilmesinde* ***object****tipi etkin bir rol oynar. Elbetteki bir iş akışına birden fazla parametrede gönderilebilmektedir.*

Bunun yapmanın yolu ise son derece basittir. Nitekim Workflow tipi aslında bir sınıftır. Dolayısıyla **Workflow** tipine **özellikler(Property)** ekleyerek dış ortamdan parametre aktarımı sağlanabilir. Bu nedenle örnekte yer alan Workflow1 sınıfına aşağıdaki gibi bir özelliğin ilave edilmesi yeterlidir.

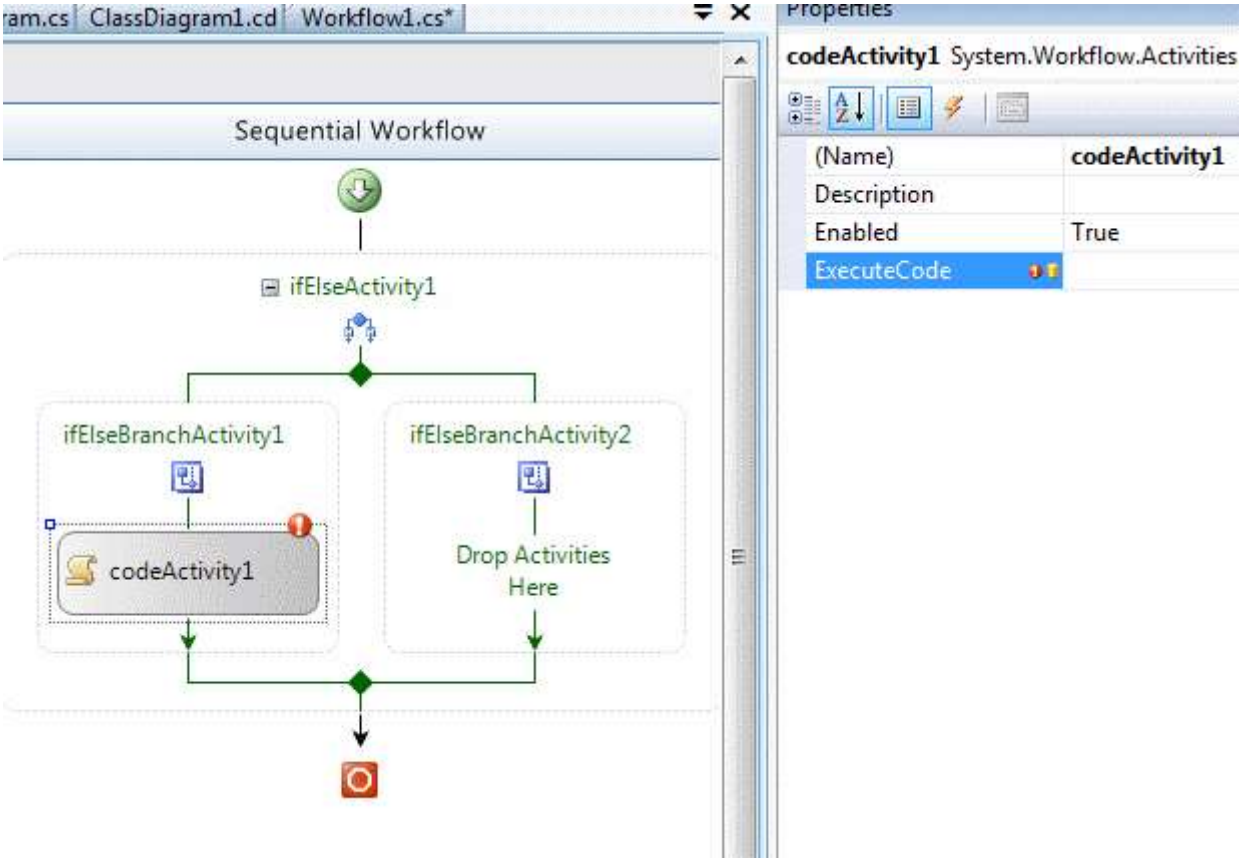
```
public sealed partial class Workflow1: SequentialWorkflowActivity
{
   private int _stokMiktari;
```

```
    public int StokMiktari
    {
        get { return _stokMiktari; }
        set { _stokMiktari = value; }
    }

    public Workflow1()
    {
        InitializeComponent();
    }
}
```

Artık tasarım tarafına dönülerek ilk **IfElseBranchActivity** için gerekli koşul aşağıdaki ekran görüntüsündeki gibi oluşturulabilir.



Buna göre **StokMiktari özelliğinin(Property)** değerinin 10 ile 50 arasındaki olması halinde, bu **IfElseBranchActitiy** bileşenin arkasından gelecek olan aktivite çalıştırılacaktır. Bu noktada ne yapılmak istendiği önemlidir. Söz gelimi bu koşulun sağlanması halinde üreticiye yeni ürün talepleri için **mail** veya **Sms** gönderilmesi gibi işlemler yaptırılabilir. Yine çok basit olarak düşünerek hareket edelim ve mail gönderme işleminin yapılacağı kod bloğunu işaret edecek bir **CodeActivity** bileşenini aşağıdaki ekran görüntüsünde olduğu gibi tasarım ortamına atalım.



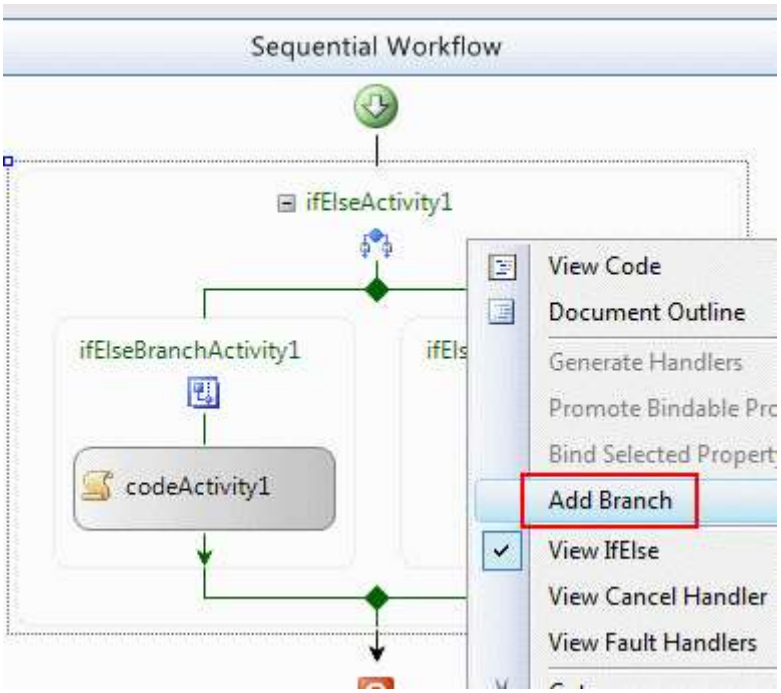
CodeActivity bileşeninin en önemli üyesi **ExecuteCode** özelliğidir. Bu özelliğe atanacak olan isim, bu adımda çalıştırılacak olan metodun adıdır. Bu olay metodunun otomatik olarak oluşturulması sağlanabilir. Bunun için **ExecuteCode** özelliğine bir metod adı yazılması ve **enter**' a basılması yeterlidir.
örnekteki **ExecuteCode** özelliğine **MailGonder** ismini yazıp**enter** tuşuna bastığımızda **Workflow1** sınıfına aşağıdaki metodun otomatik olarak eklendiği görülecektir.

```
private void MailGonder(object sender, EventArgs e)
{
}
```

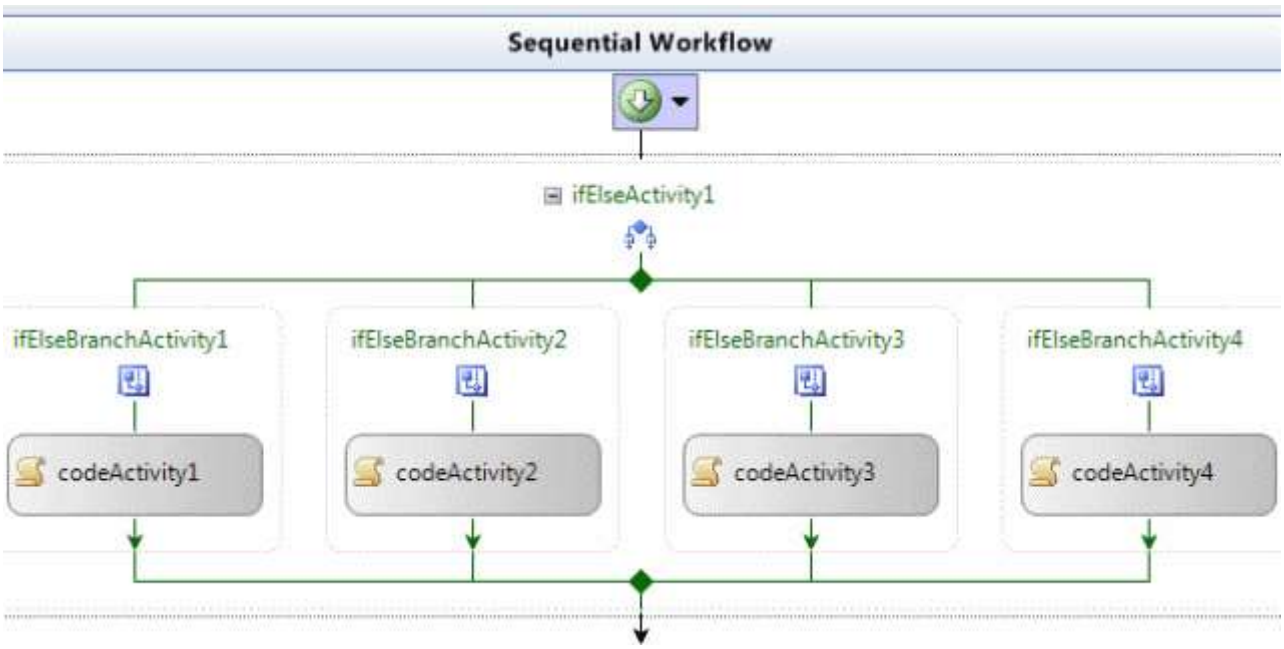
Görüldüğü gibi üretilen metod stadart bir olay metodu yapısındadır. Geriye değer döndürmemekte ve iki adet parametre almaktadır. İş mantığında bu adımda işletilmesi gereken kodlar nelerse bu metod içerisine yazılmalıdır. Söz gelimi bu adımda mail gönderme işlemi yaptırılabilir.

IfElseActivity içerisinde sağ tarafta kalan ikinci bir **IfElseBranchActivity** bileşeni daha vardır. Şu anki senaryoda bu bileşen **else** olma durumunda ne olacağını belirtmektedir. Diğer taraftan örnek senaryoda başka **IfElseBranchActivity** bileşenlerinin eklenmeside mümkündür. örneğin Stok miktarının herhangibir nedenle 0 ve altında olması hali ve Stok Miktarının 50' nin üzerinde olması hali gibi. Bu durumların her biri için birer IfElseBranchActivity kontrolü eklenip gerekli aksiyonların gerçekleştirilmesi sağlanabilir.

***NOT : IfElseActivity** bileşenleri içerisine **IfElseBranchActivity** bileşenlerini eklemek için sağ tıklayıp **Add Branch** demek yeterlidir.*



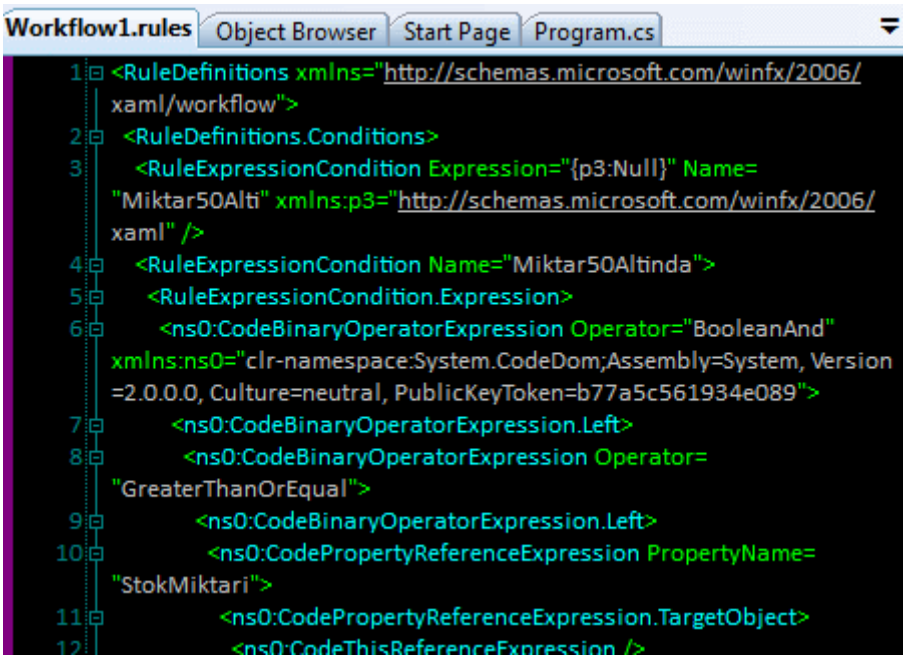
Bu amaçla örneğimize aşağıdaki şekildede görüldüğü gibi
başka **IfElseBranchActivity** bileşenleri daha eklediğimizi düşünelim.



Burada ikinci **IfElseBranchActivity** içerisinde **StokMiktari** özelliğinin değerinin **0**' ın altında ve eşit olduğu durumda çalıştırılacak bir kod aktivitesi yer almaktadır. 3ncü IfElseBranchActivity içerisinde **StokMiktari** özelliğinin değerinin **50** ile **500** arasında olduğu durumda çalışacak bir aktivite bulunur. Son IfElseBranchActivity parçasında ise var olan koşulların dışındaki durum ele alınmaktadır. Genellikle birden fazla

IfElseBranchActivity içeren durumlarda tüm ihtimallerin dışında kalabilecek bir seçeneğide ele almak için **Condition** özelliği herhangibir şekilde atanmamış boş bir **IfElseBranchActivity** parçası kullanmakta yarar vardır. örnekte bu tarz bir durum için yine kod aktivitesi yürütülmektedir. Elbette her **CodeActivity**bileşenin **ExecuteCode** özelliğine ilgili değerlerin atanması ve oluşan olay metodlarının kodlanması gerekmektedir. Şimdilik bu kısım bizim açımızdan önemli değildir. Nitekim örneğe son olarak yapılan eklemelerde kavranması gereken birden fazla **IfElseBranchActivity** bileşenin bir arada ele alınabilmesidir.

***NOT: IfElseBranchActivity** işlemlerinde karar mekanizması olarak **Declarative Rule Condition** kullanılması halinde kuralların **rules** uzantılı bir **XML** dosyası içerisine yazıldığı görülür. Aşağıdaki ekran görüntüsünde örnekte kullanılan **IfElseBranchActivity**' ler için oluşturulan **Workflow1.rules** dosyasının sadece bir kısmı görülmektedir.*



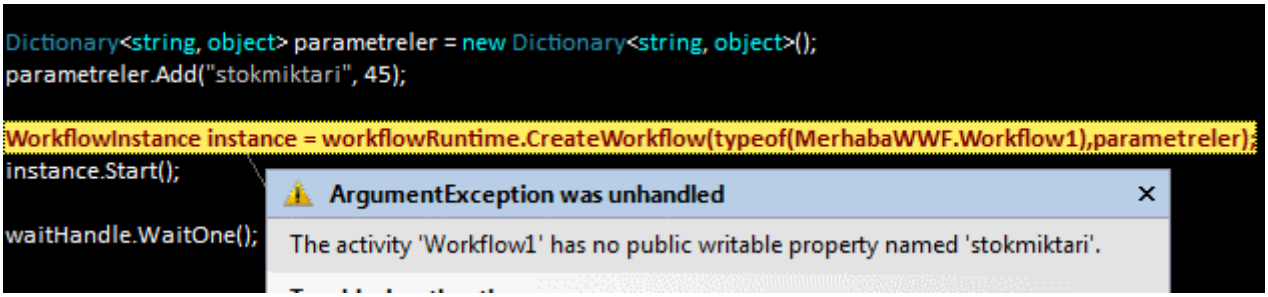
Şimdi iş akışı için daha fazla önem arz eden bir konu üzerinde durulmalıdır. İş akışına ilgili parametreler nasıl aktarılacaktır? Bu amaçla **Main** metodu içerisinde yer alan kod parçalarında bazı değişiklikler yapılması gerekmektedir. Daha öncedende belirtildiği gibi **WorkflowRuntime** sınıfına ait **CreateInstance** metodunun ikinci parametresi iş akışlarına değer göndermek için kullanılmaktadır. İş akışları herhangibir **.Net CLR** tipini parametre olarak aldığından ve dış ortamdan iş akışına gönderilen değerin hangi özelliğe aktarıldığının bilinmesi gerektiğinden **generic Dictionary<string,object>** koleksiyonu kullanılmaktadır. Böylece ilgili iş akışının hangi özelliğine, hangi değerin aktarılacağı belirtilebilir. Bu aynı zamanda iş akışına birden fazla parametre değeri gönderilebilmesi anlamınada gelmektedir.

örnekte kullanılan **StokMiktari özelliğinin(Property)** değeri iş akışına dış ortamdan, örneğin Host uygulama içerisinden gelmektedir. Bu nedenle **Main** metodu içerisinde aşağıdaki değişikliklerin yapılması gerekir.

Dictionary<string, object> parametreler = new **Dictionary<string, object>()**;
parametreler.**Add**("StokMiktari", 45);

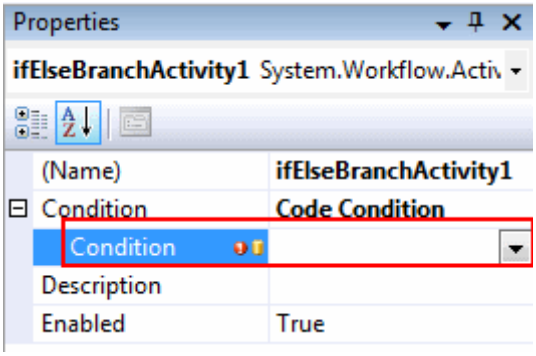
WorkflowInstance instance =
workflowRuntime.CreateWorkflow(typeof(MerhabaWWF.Workflow1)**,parametreler**);

İlk olarak parametre veya parametreleri taşıyacak
olan **Dictionary<string,object>** koleksiyonu örneklenir.
Bu **generic** koleksiyonun **anahtarları(Keys) string, değerleri(Values)** ise **object** türünden olmalıdır. Daha sonra **Add** metodu ile parametre ekleme işlemi gerçekleştirilir. örnekte StokMiktari isimi özellik için 45 değerinin verileceği belirtilmektedir. Son olarak **WorkflowInstance** örneği oluşturulurken **CreateWorkflow** metodunun ikinci parametresine, koleksiyona ait nesne örneği atanır. Burada dikkat edilmesi gereken bazı noktalarda vardır. Herşeyden önce **Dictionary** koleksiyonuna eklenen parametre adının, **iş akışı(Workflow)** sınıfı içerisinde tanımlanan **özellik adı(Property Name)** ile bire bir uygun olması gerekmektedir. Söz gelimi **Add** metodunda StokMiktari yerine stokmiktari yazılırsa aşağıdaki ekran görüntüsünde olduğu gibi çalışma zamanında **ArgumentException istisnası(Exception)** alınır.



Bu noktadan sonra iş akışı başarılı bir şekilde çalışacaktır. örnek olarak tasarlanan iş akışının herhangibir amacı ve başarısı yoktur ancak temel kavramların nasıl uygulanacağını göstermektedir.

Makalemize **IfElseBranchActivity** bileşenlerinde **Condition** özelliğinde **Code Condition** seçeneğini nasıl kullanacağımızı inceleyerek devam edelim. Bu amaçla örnek olarak herhangibir**IfElseBranchActivity**' nin **Condition** özelliğine **Code Condition** değerini atmamız yeterli olacaktır.



Burada **Condition** altında yer alan **Condition** özelliğinede bir metod adı verilmesi gerekmektedir. örneğin **Stok10ile50Arasindami** ismini verip enter tuşuna bastığımızı düşünelim. Bunun sonucu olarak **workflow1.cs** içerisine aşağıdaki metodun eklendiği görülecektir.

```
private void Stok10ile50Arasindami(object sender, ConditionalEventArgs e)
{

}
```

Metodun **bool** tipinde bir değerlendirme yapması gerekmektedir. Buna karşın dikkat edileceği üzere **void** tipinde oluşturulmuştur. İşte bu noktada koşulun sonucunu geriye döndürmek için**ConditionalEventArgs** tipinden olan parametrenin, **Result** özelliğinden yararlanılır. **Result** özelliği **bool** tipindendir ve koşula göre **true** veya **false** değerini almalıdır. Buna göre kod aşağıdaki şekilde düzenlenebilir.

```
private void Stok10ile50Arasindami(object sender, ConditionalEventArgs e)
{
   if (StokMiktari >= 10 && StokMiktari <= 50)
      e.Result = true;
   else
      e.Result = false;
}
```

Eğer StokMiktari özelliğinin değeri 10 ile 50 arasında ise bu koşul sağlanmış demektir. Bu durumda **Result** özelliğine **true** değeri atanmalıdır. Eğer true değeri atanırsa**IfElseBranchActivity**' den sonra gelen aktivitenin çalıştırılması da sağlanmış olur. Elbetteki buradaki kod parçasında, koşulun sağlanmış olma veya olmama haline göre sıradaki aktiviteye geçilmeden önce farklı işlemler yapıtırılmasıda sağlanabilir.

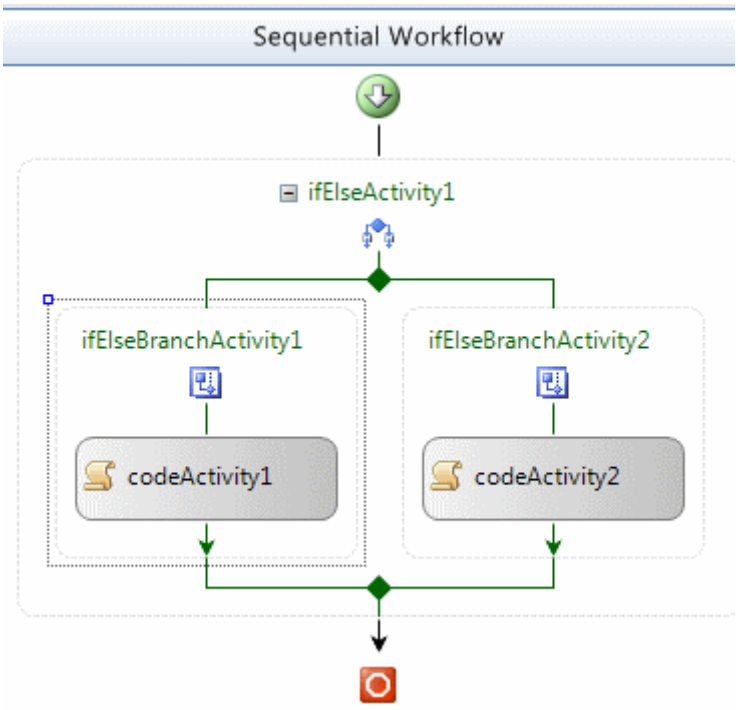
Makalemizin son bölümünde **iş akışını(Workflow)** bir **sınıf kütüphanesi(Class Library)** olarak nasıl tasarlayabileceğimizi ve bunu örneğin bir **windows** uygulamasında nasıl**host** edebileceğimizi incelemeye çalışacağız. Bu sefer proje şablonu olarak **Sequenatial Workflow Library** modelini seçmemiz gerekiyor. Oluşan sınıf kütüphanesi(Class Library) içerisinde yine standart olarak **Workflow1** isimli bir iş akışı nesnesi bulunur. örnek olarak bir metin dosyasının içerisinde parametre olarak verilen bir

kelimenin bulunması veya bulunmaması halinde yapılabilecek bazı işlemler olduğunu ve bunun bir iş süreci olarak göz önüne alındığını düşünelim. Söz konusu sürecin birden fazla **.Net** uygulaması içerisinde ele alınabileceğini düşünürsek iş akışının sınıf kütüphanesi olarak tasarlanması son derece mantıklıdır. İş akışının bu anlamda dışarıdan alması gereken iki adet parametre bulunmaktadır. Bunlardan birisi dosya adresini, diğeri ise aranacak bilgiyi tutmalıdır. Diğer taraftan iş akışından **Host** eden uygulamayada bilgi gönderilmesi istenebilir. Söz gelimi arama sonuçlarına dair bir string bilgi söz konusu olabilir. Doğal olarak 3ncü bir özelliğe daha gerek vardır. Bu nedenle **workflow1** sınıfı içerisine aşağıdaki kod parçasında yer alan **özelliklerin(Properties)** eklenmesi gerekmektedir.

```
public sealed partial class Workflow1: SequentialWorkflowActivity
{
   private string _dosyaAdresi;
   private string _arananKelime;
    private string _aramaSonucu;

   public string AramaSonucu
   {
      get { return _aramaSonucu; }
      set { _aramaSonucu = value; }
   }

   public string ArananKelime
   {
      get { return _arananKelime; }
      set { _arananKelime = value; }
   }

   public string DosyaAdresi
   {
      get { return _dosyaAdresi; }
      set { _dosyaAdresi = value; }
   }

   public Workflow1()
   {
      InitializeComponent();
   }
}
```

İlk örneğimizdeki gibi bir **IfElseActivity** ve iki adet **IfElseBranchActivity** ekliyoruz. Burada dosya içerisinde bir bilgi arama işlemi yapılmak istendiğinden koşulun kod yardımıyla kontrol edilmesi, bu nedenle **Code Condition** seçeneğinin kullanılması daha mantıklıdır. Hatalara çok fazla takılmamak adına sadece **txt** uzantılı dosyaları ele aldığımızı düşünelim.**IfElseBranchActivity1** bileşeninin **Condition** özelliğinin

değerini **Code Condition** olarak ayarladıktan sonra alt Condition özelliğinede AranankelimeVarmi bilgisini yazalim. Bu **IfElseBranchActivity1**için çalışacak koşul kontrol metodunun adı olacaktır. **IfElseBranchActivity2** bileşeni için herhangibir **Condition** ataması yapılmasına gerek yoktur. Nitekim otomatik olarak else durumunun değerlendirileceği yerdir. Her iki aktivitenin arkasından çalıştırılmak istenen kodların yer aldığı **CodeActivity** bileşenlerinide ekleyelim. Sonuçta iş akışının tasarım zamanındaki görüntüsü aşağıdaki gibi olacaktır.



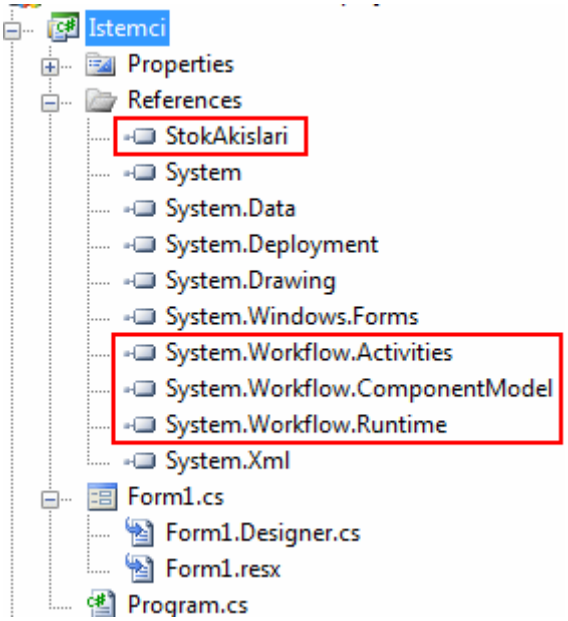
CodeActivity1 için **ArananBilgiVar** isimli bir metod, **CodeActivity2** bileşeni içinde **ArananBilgiYok** isimli bir metod devreye girecektir. Buna göre **Workflow1** sınıfının başlangıçtaki içeriği aşağıdaki gibidir.

```
using System;
using System.ComponentModel;
using System.ComponentModel.Design;
using System.Collections;
using System.Drawing;
using System.Workflow.ComponentModel.Compiler;
using System.Workflow.ComponentModel.Serialization;
using System.Workflow.ComponentModel;
using System.Workflow.ComponentModel.Design;
using System.Workflow.Runtime;
using System.Workflow.Activities;
using System.Workflow.Activities.Rules;
using System.IO;

namespace StokAkislari
```

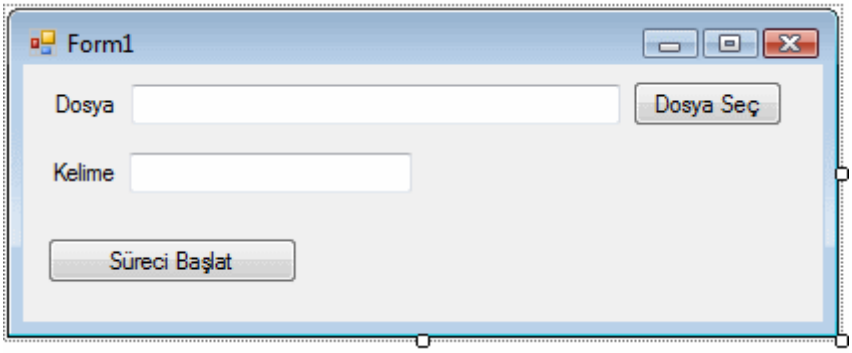
```
{
   public sealed partial class Workflow1: SequentialWorkflowActivity
   {
      private string _dosyaAdresi;
      private string _arananKelime;
      private string _aramaSonucu;

      public string AramaSonucu
      {
         get { return _aramaSonucu; }
         set { _aramaSonucu = value; }
      }

      public string ArananKelime
      {
         get { return _arananKelime; }
         set { _arananKelime = value; }
      }

      public string DosyaAdresi
      {
         get { return _dosyaAdresi; }
         set { _dosyaAdresi = value; }
      }

      public Workflow1()
      {
         InitializeComponent();
      }

      // IfElseBranchActivity1 bileşenine ait koşul kontrol metodu
      private void ArananKelimeVarmi(object sender, ConditionalEventArgs e)
      {
         if (File.Exists(DosyaAdresi))
         {
            StreamReader reader = new StreamReader(DosyaAdresi);
            e.Result = reader.ReadToEnd().Contains(ArananKelime);
         }
         else
            e.Result = false;
      }

      // CodeActivity1 için çalışacak olay metodu
      private void ArananBilgiVar(object sender, EventArgs e)
      {
```

```
            // Aranan bilgi bulunduğunda yapılacak işlemler
            AramaSonucu = AranankElime + " " + DosyaAdresi + " içinde bulunmuştur";
        }

        // CodeActivity2 için çalışacak olay metodu
        private void AranankBilgiYok(object sender, EventArgs e)
        {
            // Aranan bilgi bulunamadığında yapılacak işlemler
            AramaSonucu = AranankElime + " " + DosyaAdresi + " içinde bulunamamıştır";
        }
    }
}
```

Amacımız her zamanki gibi konuyu basitçe kavramak olduğundan **CodeActivity** bileşenleri ile ilişkili kod parçaları şimdilik göz ardı edilmiştir. Artık **host** uygulamayı yazarak devam edebiliriz. Bu amaçla basit bir **Windows** uygulaması tasarlayacağız. Uygulamamız **text** tabanlı dosyaların seçimine ve aranacak kelimenin girilmesine izin verecek bir arabirime sahip olacaktır. Buradaki asıl hedefimiz ise bir .**Net** uygulaması içerisinden herhangibir **iş akışının(Workflow)** nasıl çalıştırılabileceğini görmektir. Windows uygulamasında tahim edileceği gibi bazı**Workflow assembly**' larının ve iş akışlarını taşıyan kütüphanenin eklenmiş olması gerekmektedir. Sonuç itibariyle **windows** uygulamasında aşağıdaki şekilde görülen referansların dahil edilmesiyle işe başlanmalıdır.



Bu noktadan sonra Windows uygulamasının aşağıdaki gibi tasarlandığını düşünebiliriz. Kullanıcı Dosya Seç başlıklı düğmeye bastığında açılacak iletişim kutusu ile txt uzantılı dosya seçebilecektir. Aranacak kelime bilgiside girildikten sonra sürecin başlatılması için tek yapılması gereken Süreci Başlat başlıklı düğmeye basmak olacaktır.



Dosya seçme işlemi için **OpenFileDialog** bileşeni kullanılamktadır. Sadece **Text** tabanlı dosyalar ele alınmak istendiğinden **Filter** özelliğine **Text Files|*.txt** değeri atanmıştır. Uygulamanın kod içeriği ise aşağıdaki gibidir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.Workflow.Runtime;
using System.Threading;
using StokAkislari;

namespace Istemci
{
   public partial class Form1 : Form
   {
      private WorkflowRuntime _wfRunTime;
      private WorkflowInstance _wfInstance;
      // Workflow thread' lerinin yönetimi için AutoResetEvent nesnesi kullanılır
      private AutoResetEvent _arEvent = new AutoResetEvent(false);

      public Form1()
      {
         InitializeComponent();

         // Workflow çalışma zamanı nesnesi örneklenir
         _wfRunTime = new WorkflowRuntime();

         // Workflow tamamlandığında devreye girecek olay metodu yüklenir
         _wfRunTime.WorkflowCompleted += delegate(object sender,
WorkflowCompletedEventArgs e)
                                         {
```

```
                                        MessageBox.Show(e.OutputParameters["A
ramaSonucu"].ToString());
                                        _arEvent.Set();
                                    };

        // Workflow' un çalışması sırasında bir istisna oluştuğunda devreye girecek olay
metodu yüklenir.
        _wfRunTime.WorkflowTerminated += delegate(object sender,
WorkflowTerminatedEventArgs e)
                                    {
                                        MessageBox.Show(e.Exception.Message);
                                        _arEvent.Set();
                                    };
    }

    private void btnDosyaSec_Click(object sender, EventArgs e)
    {
        if (openFileDialog1.ShowDialog() == DialogResult.OK)
        {
            txtDosyaAdresi.Text = openFileDialog1.FileName;
        }
    }

    private void btnSureciBaslat_Click(object sender, EventArgs e)
    {
        try
        {
            if (!String.IsNullOrEmpty(txtArananKelime.Text)
                && !String.IsNullOrEmpty(txtDosyaAdresi.Text))
            {
                // parametrelerin gönderilmesi için Dictionary koleksiyonu örneklenir
                Dictionary<string, object> parametreler = new Dictionary<string,
object>();

                // Workflow1 içerisindeki özelliklerin alacağı değerler set edilir.
                parametreler.Add("DosyaAdresi", txtDosyaAdresi.Text);
                parametreler.Add("ArananKelime", txtArananKelime.Text);

                // Workflow1 için bir örnek oluşturulur.
                _wfInstance = _wfRunTime.CreateWorkflow(typeof(Workflow1),
parametreler);

                // Workflow1 başlatılır.
                _wfInstance.Start();
```

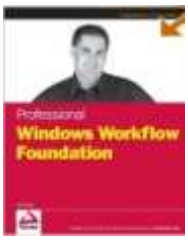
```
                _arEvent.WaitOne();
            }
            else
                MessageBox.Show("Verilerde eksik var");
        }
        catch (Exception exp)
        {
            MessageBox.Show(exp.Message);
        }
    }
  }
}
```

Geliştirilen örnekte **form** nesnesi üretilirken **iş akışı çalışma ortamı(Workflow Run-Time)** için gerekli ayarlar yapılmaktadır. Bu amaçla **WorkflowRuntime** nesnesi örneklenmekte,**WorkflowCompleted**, **WorkflowTerminated** gibi olaylar **isimsiz metodlar(Anonymous Methods)** aracılığıyla yüklenmektedir. Bir önceki örnekten farklı olarak dikkat edilmesi gereken nokta WorkflowCompleted olay metodu içerisinde **WorkflowCompletedEventArgs** sınıfına ait **OutputParameters** özelliğinin kullanılışıdır. Bu özellikte **Dictionary<string,object>** tipinden**generic** bir koleksiyonu ele almaktadır. özelliğin amacı, iş akışından uygulama ortamına değer aktarımını sağlamaktır. Bu amaçla **indeksleyici(Indexer)** operatörü içerisinde, iş akışında tanımlanan **AramaSonucu** özelliğinin adı verilmektedir.

Doğal olarak sürecin bir şekilde başlatılması gerekmektedir. Bu amaçla **WorflowInstance** örneği oluşturulduktan sonra, **Start** metodu çağırılmaktadır. Tüm bu başlatma işlemi ise **Form** üzerindeki bir **Button** kontrolüne ait **Click** olay metodu içerisinde gerçekleştirilmektedir. örnek test edildiğinde bir döküman içerisinde aranan bilginin var olup olmadığına dair sonuçların alındığı görülecektir. *(Tahmin edileceği üzere bu tarz bir ihtiyaç normal bir sınıf kütüphanesi içerisine alınacak bir tip ilede, iş akışlarına gerek olmadan tasarlanabilir. Ancak gerçek iş problemleri göz önüne alındığında tek başına yeterli bir çözüm olmayacaktır.)*

Buraya kadar yazdıklarımız ile **iş akışı(Workflow)** kavramını, **Windows Workflow Foundation** yaklaşımını incelemeye çalıştık. Giriş niteliğindeki bu makalemizde, bir iş akışı için gerçek hayat senaryoları kullanmamış olsakta, **WWF** ile nasıl geliştirilebileceklerini, herhangibir **.Net** uygulamasından nasıl kullanılabileceklerini gördük. Bundan sonraki makalelerimizde **WWF** mimarisinin başka konularınıda incelemeye çalışıyor olacağız. Makalemize son vermeden önce Windows Workflow Foundation ile ilgili kaynak **kitaplar** hakkında bilgi vermek isterim. Söz konusu kitaplar ve bunlara ait özet bilgiler aşağıdaki tabloda yer aldığı gibidir.

Kitap	**özet Bilgi**

	**Microsoft Windows Workflow Foundation Step by Step** **Mart 2007** tarihinde çıkan **Microsoft Press**' e ait bu kitap içerisindeki bilgiler ile V **Communication Foundation Step by Step** kitabında olduğu gibi oldukça iyi bir a seviyede öğrenmek mümkün. Toplam 19 bölümden oluşan kitap içerisinde iş akışla **tasarlanması** gibi ileri seviye konularda yer almakta.
	**Pro WF: Windows Workflow in .NET 3.0 (Expert's Voice in .Net)** **Şubat 2007** tarihinde **APress** tarafından yayınlanan bu kitap **744 sayfalık** bir içeriğ temelden ileri seviyeye doğru giden ve en anlaşılır olanlarından bir tanesi. İleri sevi serileştirme, iş akışlarının izlenmesi gibi konu başlıklarıda yer almaktadır.
	**Professional Windows Workflow Foundation** **Wrox** yayınlarından **Mart 2007** tarihinde çıkartılan bu kitap **410 sayfalık** mütevaz Foundation mimarisini anlamak ve etkin bir şekilde kullanabilmek mümkün. üstelik güncelleştirme, ofis(Office) sistemleri ile entegrasyon gibi enteresan kısımlarda ye

Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

WWFGiris.rar (83,28 kb)

LINQ to SQL : Arka Planda Neler Oluyor? (2007-12-19T16:43:00)

linq,linq to sql,

Veritabanı(Database) nesnelerinin programatik ortamda sınıf gibi tipler(Type) ve metod benzeri üyeler(Members) ile ifade ediliyor olması, bu tiplere ait nesne örnekleri üzerinden sorgulalamalar yapılabilmesi ihtiyacınıda ortaya çıkartmıştır. Bir veritabanı nesnesinin programatik taraftaki karşılığının nesne yönelimli(Object Oriented) bir dilde geliştirilmesi son derece kolaydır. örneğin bir **tablo(Table)** göz önüne alındığında, bu tablonun kendisi bir **sınıf(Class)** olarak tasarlanabilir. Benzer şekilde, tablo içerisindeki **alanlar(Fields)** sınıf içinde yer alan birer**özellik(Property)** olarak düşünülebilir.
Basit **CRUD(CreateRetrieveUpdateDelete)** işlemleri, **varlık sınıfı(Entity Class)** diyebileceğimiz tipin birer üye metodu olarak düşünülebilir. Tabloda yer alan kolonların bazı niceleyicileri(örneğin Null değer içerip içermedikleri, primary key olup olmadıkları vb...) sınıfın kendisi ve üyeleri için birer**nitelik(Attribute)** olarak tasarlanabilir. Ne varki bu eşleştirme kolaylığı dışında, programatik tarafta yer alan nesnel yapılar üzerinde, SQL cümlelerine benzer ifadeler ile sorgulamalar yapmak kolay değildir. Nitekim, programatik tarafın SQL benzeri cümelere karşılık gelen fonksiyonellikleri ele alıyor olması gerekmektedir. **LINQ(Language Integrated Query)** mimariside, temel

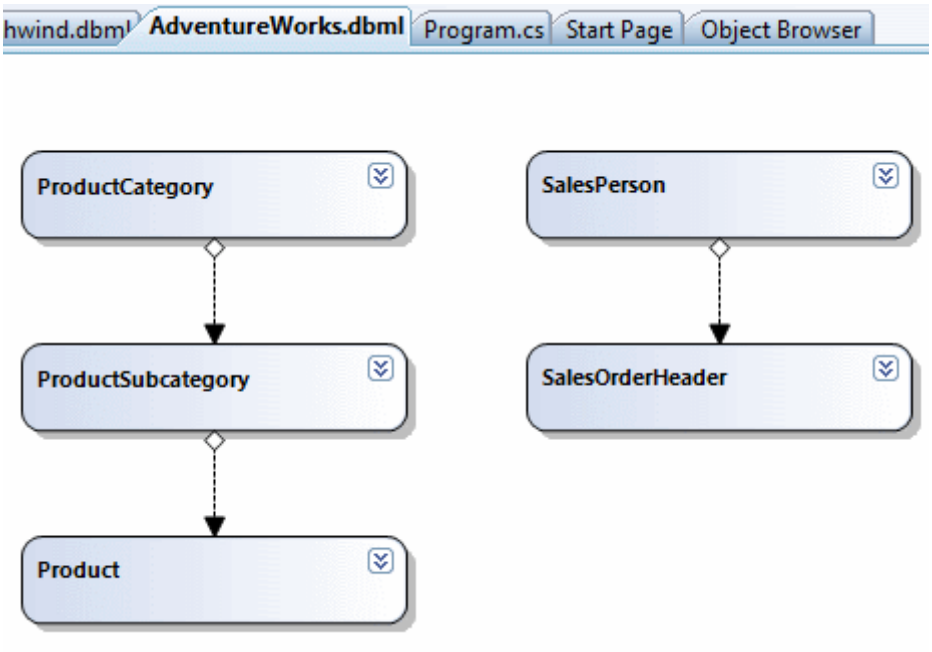
anlamda programatik tarafta yazılan ifadeleri arka planda metodlar, **temsilciler(Delegates)** yardımıyla kurduğu bir modele dönüştürmektedir. **LINQ**' in kullanıldığı alanlar göz önüne alındığında en popüler seçeneklerden biriside **LINQ to SQL** mimarisidir.

Language INtegrated Query to SQL mimarisi ile, **varlık tipleri(Entity Types)** üzerinden sorgular çalıştırılabilir. Basit anlamda, **nesneler(Objects)** üzerinde uygulanabilen LINQ sorguları, SQL tarafına ulaştıklarında ise bildiğimiz **sorgu ifadelerine(Query Expressions)** dönüşmektedir. Bilinen LINQ operatörlerinin veya metodlarının tamamının SQL tarafına uygulanamadığı veya henüz uygulanamadığı bir gerçektir. Nitekim programatik ortamın esnekliği nedeni ile, örneğin bir dizinin ters çevrilerek elemanları üzerinde döngüsel anlamda ilerlenebilmesinin, **SQL** tarafında karşılığının bulunması zordur(ki buda **LINQ** metodlarından olan **Reverse** fonksiyonelliğinin neden **LINQ to SQL** üzerinde kullanılamadığınıda açıklamaktadır).

Bu ana fikirlerden yola çıkarak makalemizdeki ana temamızın, SQL ifadelerine çevrilebilen **LINQ** operatörlerinin veya fonksiyonelliklerinin, arka planda ne şekilde tasarlandıklarını inceleyebilmektir. Bir başka deyişle basit ve karmaşık LINQ cümlelerinin, SQL tarafında ele alınabilen karşılıklarının ne olduklarını tespit edebilmektir. Bu araştırmadaki en büyük yardımcılarımız ise **SQL Server Profiler** ve **Estimated Execution Plan araçları(Tools)** olacaktır. **SQL Server Profiler** aracı kullanılarak, **varlık(Entity)** nesneleri üzerinde çalıştırılan**LINQ** ifadelerinin karşılığı olan SQL sorgu cümlelerini görmek mümkün olabilmektedir. Diğer taraftan **Estimated Execution Plan** aracı sayesinde, LINQ için arka tarafta çalıştırılan bir sorgu cümlesinin icra planının görülmesi ve alternatif ifadeler ile aralarındaki farklar tespit edilerek daha optimal yolların göz önüne alınması sağlanabilir.

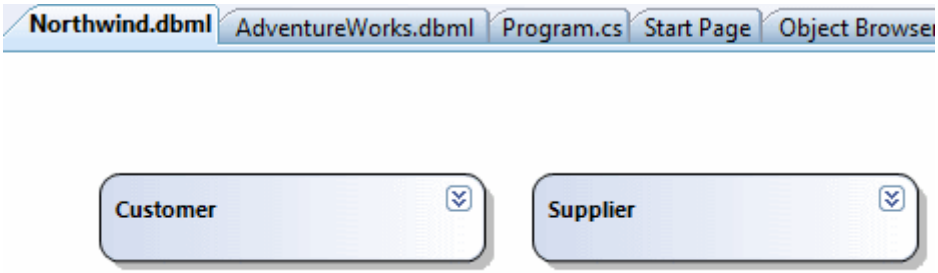
Dilerseniz hiç vakit kaybetmeden örneklerimize geçerek devam edelim. Her zamanki gibi **Visual Studio 2008 RTM** üzerinden örnek kod parçalarımızı çalıştırıyor olacağız. Basit bir**Console** uygulaması üzerinden ilerlerken **AdventureWorks** ve **Northwind veritabanlarındaki(Database)** bazı tabloları kullanıyor olacağız. Bu anlamda **AdventureWorks.dmbl** ve**Northwind.dmbl** isimli **DataBase Markup Language** içeriklerimiz aşağıdaki ekran görüntülerinde yer aldığı gibi olacaktır.

AdventureWorks.dbml;



AdventureWorks veritabanından örnek sorgulamalar için **ProductCategory**, **ProductSubCategory**, **Product**, **SalesPerson** ve **SalesOrderHeader** tabloları ele alınmaktadır.

Northwind.dmbl;



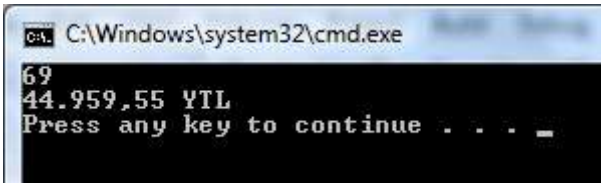
Northwind veritabanından ise **Customer** ve **Supplier** tabloları ele alınmaktadır.

İlk olarak aşağıdaki gibi basit bir **LINQ** ifadesi ile başlayalım.

```
AdventureWorksDataContext adw = new AdventureWorksDataContext();

var mClassProducts = from prd in adw.Products
                     where prd.Class == "M"
                        select prd;

int mCount = mClassProducts.Count();
double mSumListPrice = mClassProducts.Sum<Product>(prd =>
(double)prd.ListPrice);
```

```
Console.WriteLine(mCount.ToString());
Console.WriteLine(mSumListPrice.ToString("C2"));
```

İlk olarak AdventureWorksDataContext nesnesi örneklenmektedir. Sonrasında **mClassProducts** isimli alana atanan ifadede **AdventureWorksDataContext** içerisinde yer alan **Products** özelliğinin karşılığı olan generic **Table<Product>** tipi ele alınmaktadır. Buna göre **Product** nesne örneklerinden, **Class** özelliklerinin(Properties) değerleri **M** olanlar seçilmektedir. Daha önceki makalalerdede belirttiğimiz gibi **var** anahtar kelimesi ile tanımlanmış olan değişkene atanan bu ifade çalışma zamanında hemen yürütülmemektedir. İcra işlemi için bir **for** iterasyonu olması yada ifade üzerinden örnekteki gibi **Aggregate** benzeri fonksiyonelliklerin çalıştırılması gerekmektedir. **mCount** alanının değeri, hazırlanan **LINQ** ifadesinde **Count** metodu uygulanarak elde edilmektedir. Bir başka ifadeyle, sınıfı **M** olan ürünlerin toplam sayısı bulunmaktadır. **mSumListPrice** alanına atanan değer ilede, sınıfı M olan ürünlerin **liste fiyatlarının(ListPrice)** toplamı elde edilir. Sonrasında ise bu sonuçlar ekrana yazdırılır. **çalışma zamanında(run time)** uygulamanın çıktısı aşağıdaki gibi olacaktır.



Gelelim arka tarafta çalıştırılan SQL sorgu cümlelerine. **SQL Server Profiler** aracı kullanılarak yapılan çalışmada **Count** ve **Sum aggregate** metodlarının aşağıdaki gibi icra edildiği görülmektedir. Count fonksiyonunun çağırılması sonucu çalışan sorgu şu şekildedir;

```
exec sp_executesql N'SELECT COUNT(*) AS [value]
                      FROM [Production].[Product] AS [t0]
                        WHERE [t0].[Class] = @p0',N'@p0 nvarchar(1)',@p0=N'M'
```

Burada dikkat edilmesi gereken noktalardan birisi **Count(*)** ifadesidir. Normal şartlar altında tavsiye edilen yöntemlerden birisi **Count(ProductID)** tarzında bir kullanım yapılması yönündedir. Bu tarz bir kullanımın performans yönünde avantaj sağladığı bilinmektedir. Nitekim LINQ tarafından gelen ifadeye göre Count(*) şeklinde bir **SQL** fonksiyonu kullanılmıştır. Diğer tarafan**LINQ** sorgusunun aşağıdaki gibi değiştirilmesi düşünülebilir.

```
var mClassProducts = from prd in adw.Products
                     where prd.Class == "M"
                       select prd.ProductID;
```

Dikkat edileceği üzere burada sadece **ProductID** alanları seçilmektedir. Bu ifade üzerinden **Count** metodu kullanılırsa **SQL** tarafında icra edilen operasyonun değişmediği,

bir başka deyişle**Count(*)** çağrısı yapıldığı görülür. **Sum** fonksiyonunun çağırılması sonucu çalışan sorgu ise aşağıdaki gibidir.

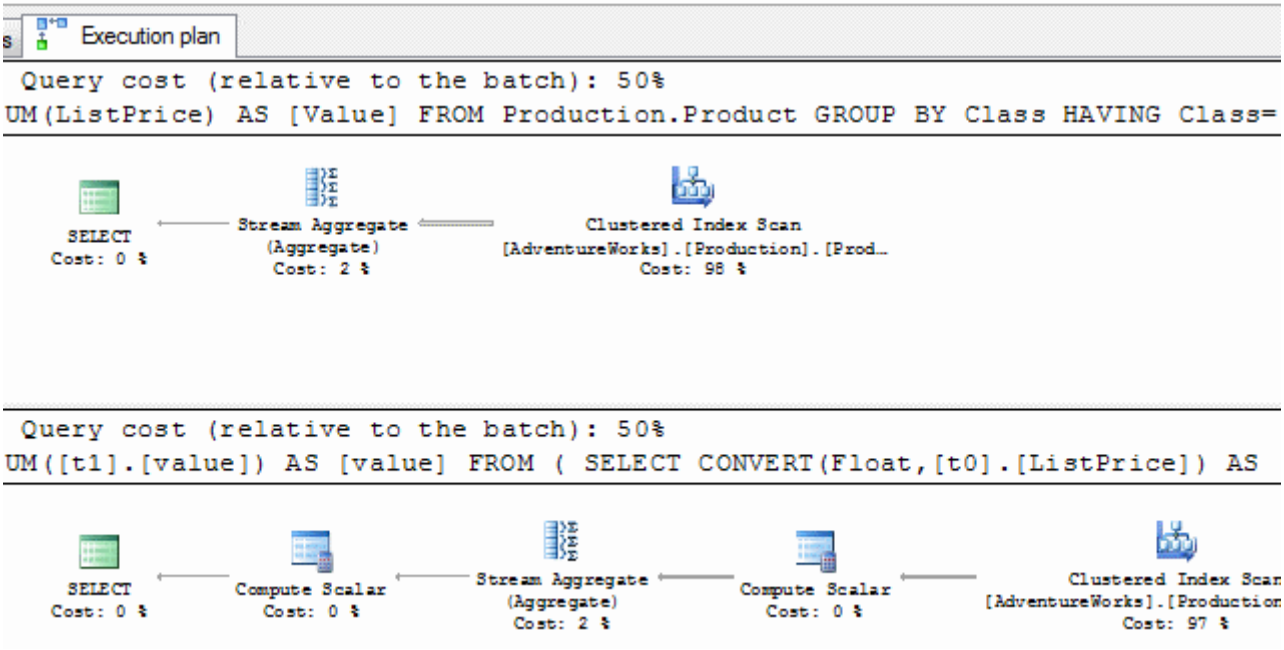
```
exec sp_executesql N'SELECT SUM([t1].[value]) AS [value]
FROM (
   SELECT CONVERT(Float,[t0].[ListPrice]) AS [value], [t0].[Class]
      FROM [Production].[Product] AS [t0]
   ) AS [t1]
WHERE [t1].[Class] = @p0',N'@p0 nvarchar(1)',@p0=N'M'
```

Dikkat edilecek olursa burada **Sum** işlemi için bir iç Select sorgusu daha yürütülmektedir. Aynı amaca yönelik olaraktan aşağıdaki gibi bir sorguda göz önüne alınabilir.

```
SELECT SUM(ListPrice) AS [Value]
FROM Production.Product
GROUP BY Class
HAVING Class='M'
```

Burada **Group By** kullanımı gerçekleştirilmektedir. **LINQ**' in **Sum** metodunu neden farklı bir şekilde yorumladığı tartışılabilir. Sonuç itibariyle her iki sorgu cümlesininde **beklenen icra planlarına(Esitamted Execution Plan)** bakıldığında **Products** gibi sadece 504 satıra sahip olan küçük bir tabloda çok fazla bir fark olmadığı görülmektedir. Ancak tablo boyutunun artması halinde bu durumun belirgin performans farklılıklarına yol açıp açmayacağına bakılmalıdır. Aşağıdaki ekran görüntülerinde her iki sorgunun icra planlarına ait icra maliyetleri daha net bir şekilde görülmektedir.

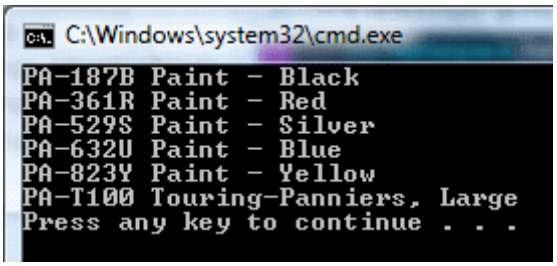
```
--ORNEK 1
SELECT SUM(ListPrice) AS [Value] FROM Production.Product
GROUP BY Class
HAVING Class='M'


SELECT SUM([t1].[value]) AS [value]
FROM (
    SELECT CONVERT(Float,[t0].[ListPrice]) AS [value], [t0].[Class]
    FROM [Production].[Product] AS [t0]
    ) AS [t1]
WHERE [t1].[Class] ='M'
```



Diğer **LINQ to SQL** operatörlerini inceleyerek devam edelim.
Sıradaki **LINQ** ifadesinde, **Contains** metodu ele alınmaktadır. Contains metodu **String** sınıfına ait bir fonksiyondur. Aşağıdaki kod parçasında yer alan ifadeye göre **Contains** metodunun görevi, **ProductNumber** alanında **PA** hecesi olan **Product** nesne örneklerinin tespit edilmesinin sağlanmasıdır.

```
var allProducts = from prd in adw.Products
                  where prd.Class == null && prd.ProductNumber.Contains("PA")
                    select prd;

foreach (Product p in allProducts)
{
   Console.WriteLine(p.ProductNumber + " " + p.Name);
}
```

Sorgu ifadesi **null** değere sahip olan sınıflara ait ürünlerden, ürün numarasında **PA** hecesi geçenleri elde etmektedir. Yukarıdaki kod parçasını içeren **Console** uygulaması çalıştırıldığında aşağıdaki sonuçlar alınacaktır.



Bizim için asıl önem arz eden konu **LINQ to SQL** ifadesinin **SQL** sunucusuna nasıl gönderildiğidir. Hemen **SQL Server Profiler** aracına bakılırsa aşağıdaki sorgu cümlesinin çalıştırıldığı görülebilir.

```
exec sp_executesql N'SELECT
   [t0].[ProductID], [t0].[Name]
   , [t0].[ProductNumber], [t0].[MakeFlag]
   , [t0].[FinishedGoodsFlag], [t0].[Color]
   , [t0].[SafetyStockLevel], [t0].[ReorderPoint]
   , [t0].[StandardCost], [t0].[ListPrice]
   , [t0].[Size], [t0].[SizeUnitMeasureCode]
   , [t0].[WeightUnitMeasureCode], [t0].[Weight]
   , [t0].[DaysToManufacture], [t0].[ProductLine]
   , [t0].[Class], [t0].[Style], [t0].[ProductSubcategoryID]
   , [t0].[ProductModelID], [t0].[SellStartDate]
   , [t0].[SellEndDate], [t0].[DiscontinuedDate]
   , [t0].[rowguid], [t0].[ModifiedDate]
FROM [Production].[Product] AS [t0]
WHERE
   ([t0].[Class] IS NULL) AND ([t0].[ProductNumber] LIKE @p0)'
,N'@p0 nvarchar(4)',@p0=N'%PA%'
```

Dikkat edileceği üzere programatik tarafta yapılan **==null** kontrolü **SQL** tarafında **Is Null** olarak, **Contains** metodu ise **Like** olarak çevrilmiştir. Buradaki Like ifadesine gönderilen **%PA%** değeri, içerisinde PA hecesi geçenleri ifade etmektedir.
öyleyse **Contains** metodu yerine örneğin **StartsWith** fonksiyonu kullanılırsa ne olacağına bakılmalıdır. Bu amaçla **LINQ to SQL** ifadesini aşağıdaki gibi değiştirdiğimizi düşünelim.

```
var allProducts = from prd in adw.Products
                  where prd.Class == null && prd.ProductNumber.StartsWith("PA")
                    select prd;
```

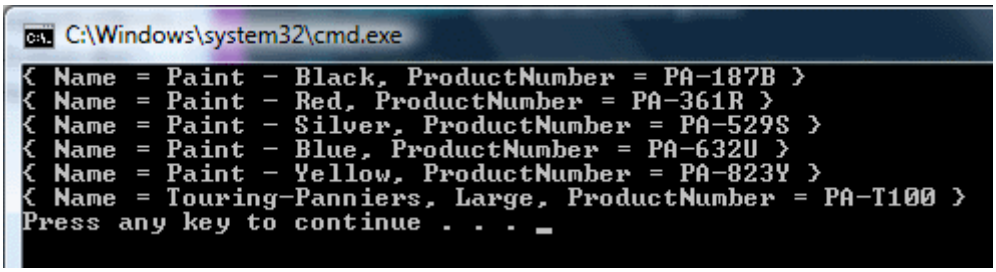
Bu durumda SQL tarafına gönderilen sorgu cümlesinin içeriğine bakıldığında **LIKE** anahtar kelimesine atanan parametrenin **PA%** şeklinde olduğu

görülmektedir. Dolayısıyla beklenildiği gibi **PA** hecesi ile başlayanların tedarik edilmesi için sorguda gerekli düzenleme yapılmıştır.

Bir önceki **LINQ to SQL** ifadesinde, SQL sunucusu üzerinde çalışan sorguya bakıldığında **Product** tablosundaki tüm alanların çekildiği görülmektedir. Oysaki çoğu durumda elde edilip veri kümesi **Entity** üzerine alındığında yanlızca bir kaç alan üzerinde işlem yapılmaktadır. Söz gelimi elde edilen listenin bir **GridView** üzerinde gösterilmesi istendiğinde tüm alanlar yerine gerekli olanların gösterilmesi tercih edilir. İşte bu noktada **isimsiz tiplerin(Anonymous Types)** faydası ortaya çıkmaktadır. Buna göre aşağıdaki LINQ to SQL ifadesini ele alalım.

```
var allProducts = from prd in adw.Products
                  where prd.Class == null && prd.ProductNumber.Contains("PA")
                     select new
                        {
                           prd.Name
                           , prd.ProductNumber
                        };

foreach (var p in allProducts)
{
   Console.WriteLine(p.ToString());
}
```

Bu örnek kod parçasında **PA** hecesini içeren ve **Class** alanının değeri **null** olan **Product** tiplerindeki **Name** ve **ProductNumber** özellikleri kullanılarak yeni bir tip elde edilmektedir. Burada ihtiyaçlar dahilinde **isimsiz tipin(Anonymous Type)** hangi özellikleri(Properties) içereceği belirlenebilir. Bu örnek kod parçasının çalışma zamanında üreteceği çıktı aşağıdaki ekran görüntüsündekine benzer olacaktır.



SQL sunucusu tarafına gönderilen sorgu cümlesinin içeriği ise aşağıdaki gibidir.

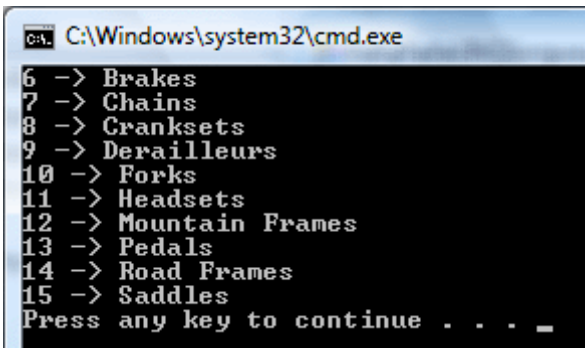
```
exec sp_executesql N'SELECT [t0].[Name], [t0].[ProductNumber]
FROM [Production].[Product] AS [t0]
WHERE
   ([t0].[Class] IS NULL) AND ([t0].[ProductNumber] LIKE @p0)'
,N'@p0 nvarchar(4)',@p0=N'%PA%'
```

Görüldüğü gibi sadece istenen alanların çekilmesi sağlanmaktadır. Buda isimsiz tiplerin **LINQ to SQL** tarafında oldukça önemli bir rol oynadığını göstermektedir.

LINQ to SQL tarafında kullanılan ilginç fonksiyonelliklerden ikiside **Skip** ve **Take** metodlarıdır. Skip metodu parametre olarak verilen değer kadar atlanılmasını, Take metodu ise atlanılan satırdan itibaren kaç satır alınacağını belirtmektedir. Buda basit olarak bir sayfalamanın(Paging) yapılabilmesine olanak sağlamaktadır. çok doğal olarak metodlar yardımıyla programatik ortamda kolayca uygulanabilen bu tekniğin SQL tarafına aktarılmasında **SQL 2005** ile birlikte gelen **Row_Number** fonksiyonunun önemli bir rolü vardır. Bu metodları daha iyi analiz etmek için aşağıdaki kod parçasını ele aldığımızı düşünelim.

```
var tenCtg = (from cat in adw.ProductSubcategories select cat)
                    .Skip<ProductSubcategory>(5)
                      .Take<ProductSubcategory>(10);

foreach (ProductSubcategory c in tenCtg)
{
   Console.WriteLine("{0} -> {1} ", c.ProductSubcategoryID.ToString(), c.Name);
}
```

Yukarıdaki kod parçasında yer alan **LINQ to SQL** ifadesine göre, **ProductSubCategories** koleksiyonu üzerinden ilk satır atlanarak 6ncı satırdan itibaren 10 satırlık bir **ProductSubCategory**nesne topluluğunun çekilmesi amaçlanmaktadır. Söz konusu kod parçasının çalıştırılmasının sonucu oluşan program çıktısına ait ekran görüntüsü aşağıdaki gibidir.



örnek kod parçasında kullanılan **LINQ to SQL** ifadesi için **SQL** sunucusu üzerine gönderilen sorgu cümlesi ise aşağıdaki gibi olacaktır.

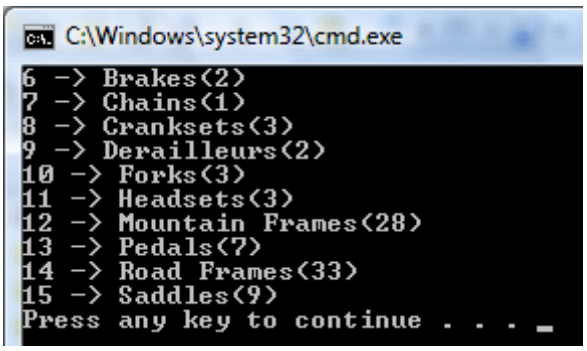
```
exec sp_executesql N'SELECT [t1].[ProductSubcategoryID], [t1].[ProductCategoryID],
[t1].[Name], [t1].[rowguid], [t1].[ModifiedDate]
FROM (
   SELECT ROW_NUMBER() OVER
      (ORDER BY
         [t0].[ProductSubcategoryID], [t0].[ProductCategoryID],
```

```
          [t0].[Name], [t0].[rowguid]
         , [t0].[ModifiedDate])
      AS [ROW_NUMBER], [t0].[ProductSubcategoryID], [t0].[ProductCategoryID],
[t0].[Name], [t0].[rowguid], [t0].[ModifiedDate]
   FROM [Production].[ProductSubcategory] AS [t0]
) AS [t1]
WHERE [t1].[ROW_NUMBER] BETWEEN @p0 + 1 AND @p0 + @p1
ORDER BY [t1].[ROW_NUMBER]',N'@p0 int,@p1 int',@p0=5,@p1=10
```

Dikkat edileceği üzere **Row_Number** fonksiyonu burada önemli bir rolü üstlenmektedir. Şimdi son kod parçasında aşağıdaki gibi bir ekleme daha yaptığımızı düşünelim.

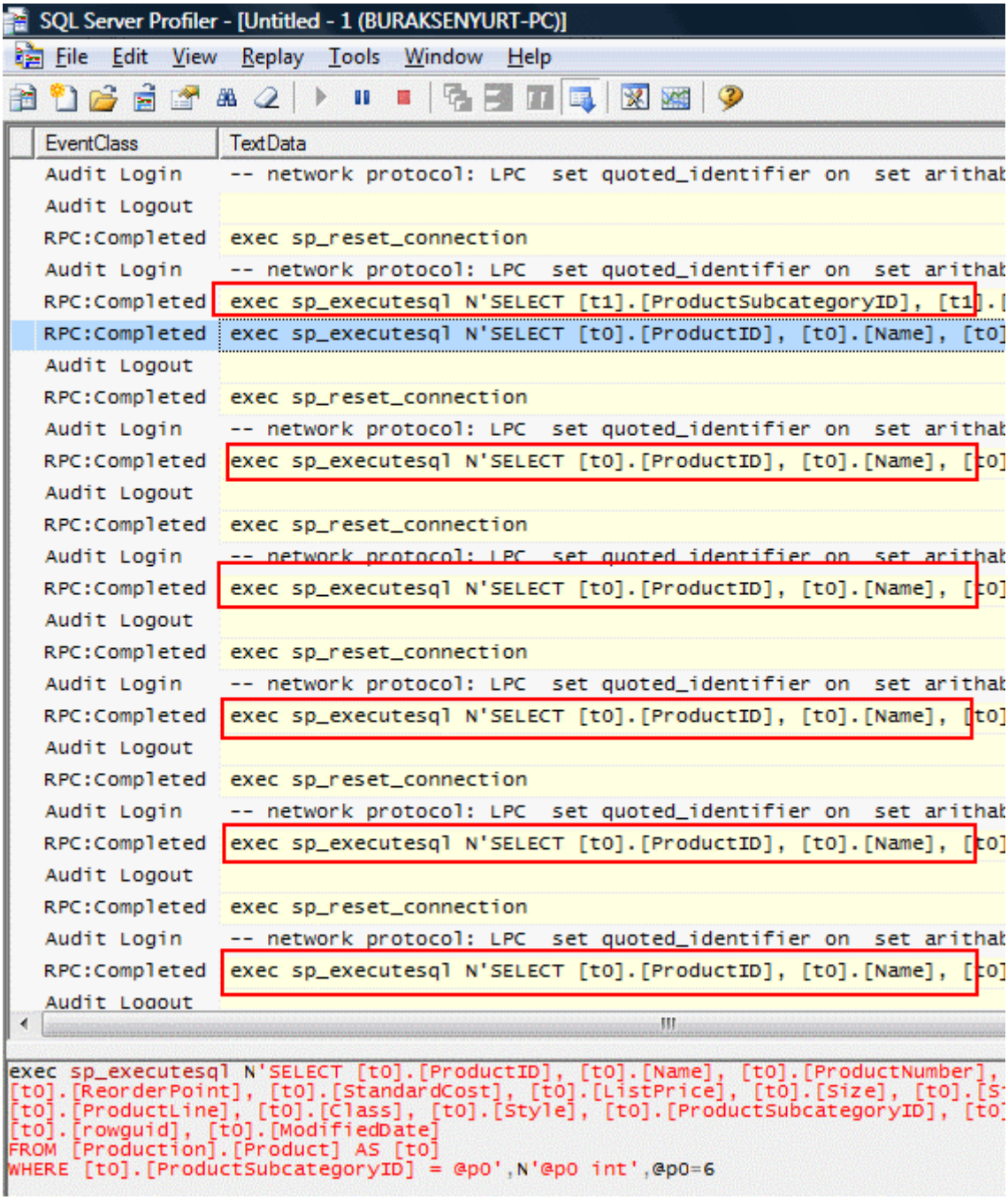
```
var tenCtg = (from cat in adw.ProductSubcategories select cat)
                    .Skip<ProductSubcategory>(5)
                      .Take<ProductSubcategory>(10);

foreach (ProductSubcategory c in tenCtg)
{
   Console.WriteLine("{0} -> {1}({2})
",c.ProductSubcategoryID.ToString(),c.Name,c.Products.Count().ToString());
}
```

Yapılan değişikliğe göre, 5nci satırdan itibaren alınan 10 **ProductSubCategory** nesnesinin her biri için **Products** özelliğinden yola çıkılarak **Count** değerleride hesaplanmaktadır. Bir başka deyişle her bir alt kategorideki ürünlerin toplam sayılarıda elde edilmektedir. Bu kod parçasının **çalışma zamanı(run time)** görüntüsü ise aşağıdaki gibidir.



ProductSubCategory ve **Product** sınıfları arasında **bire çok(one to many)** bir ilişki mevcuttur. Bu ilişki programatik taraftada ilgili **varlık sınıflarına(Entity Class)** yansıtılmaktadır. Bu nedenle bir ProductSubCategory nesne örneği üzerinden Products özelliği ile bağlı olunan Product nesne topluluğuna geçiş yapmak son derece kolaydır. Bu da gönül rahatlığı ile**Count** gibi metodları kullanıp istediğimiz tarzda sonuçları alabilmemizi olanaklı kılmaktadır. Ne varki **Count** çağrısı **for** döngüsü içerisinde, bir önceki sorgudan elde edilen her bir ProductSubCategory nesne örneği için ayrı ayrı yapılmaktadır. Bunun **SQL** sunucusu üzerinde oluşturacağı sonuç ise şudur; elde edilen her bir ProductSubCategory için, buna bağlı toplam ürün sayısını döndüren bir sorgu

cümlesi çalışmaktadır. Aşağıdaki ekran görüntüsünde bu durumun bir kısmı ifade edilmektedir.



EventClass	TextData
Audit Login	-- network protocol: LPC set quoted_identifier on set arithab
Audit Logout	
RPC:Completed	exec sp_reset_connection
Audit Login	-- network protocol: LPC set quoted_identifier on set arithab
RPC:Completed	exec sp_executesql N'SELECT [t1].[ProductSubcategoryID], [t1].[
RPC:Completed	exec sp_executesql N'SELECT [t0].[ProductID], [t0].[Name], [t0]
Audit Logout	
RPC:Completed	exec sp_reset_connection
Audit Login	-- network protocol: LPC set quoted_identifier on set arithab
RPC:Completed	exec sp_executesql N'SELECT [t0].[ProductID], [t0].[Name], [t0]
Audit Logout	
RPC:Completed	exec sp_reset_connection
Audit Login	-- network protocol: LPC set quoted_identifier on set arithab
RPC:Completed	exec sp_executesql N'SELECT [t0].[ProductID], [t0].[Name], [t0]
Audit Logout	
RPC:Completed	exec sp_reset_connection
Audit Login	-- network protocol: LPC set quoted_identifier on set arithab
RPC:Completed	exec sp_executesql N'SELECT [t0].[ProductID], [t0].[Name], [t0]
Audit Logout	
RPC:Completed	exec sp_reset_connection
Audit Login	-- network protocol: LPC set quoted_identifier on set arithab
RPC:Completed	exec sp_executesql N'SELECT [t0].[ProductID], [t0].[Name], [t0]
Audit Logout	
RPC:Completed	exec sp_reset_connection
Audit Login	-- network protocol: LPC set quoted_identifier on set arithab
RPC:Completed	exec sp_executesql N'SELECT [t0].[ProductID], [t0].[Name], [t0]
Audit Logout	

```
exec sp_executesql N'SELECT [t0].[ProductID], [t0].[Name], [t0].[ProductNumber],
[t0].[ReorderPoint], [t0].[StandardCost], [t0].[ListPrice], [t0].[Size], [t0].[S
[t0].[ProductLine], [t0].[Class], [t0].[Style], [t0].[ProductSubcategoryID], [t0]
[t0].[rowguid], [t0].[ModifiedDate]
FROM [Production].[Product] AS [t0]
WHERE [t0].[ProductSubcategoryID] = @p0',N'@p0 int',@p0=6
```

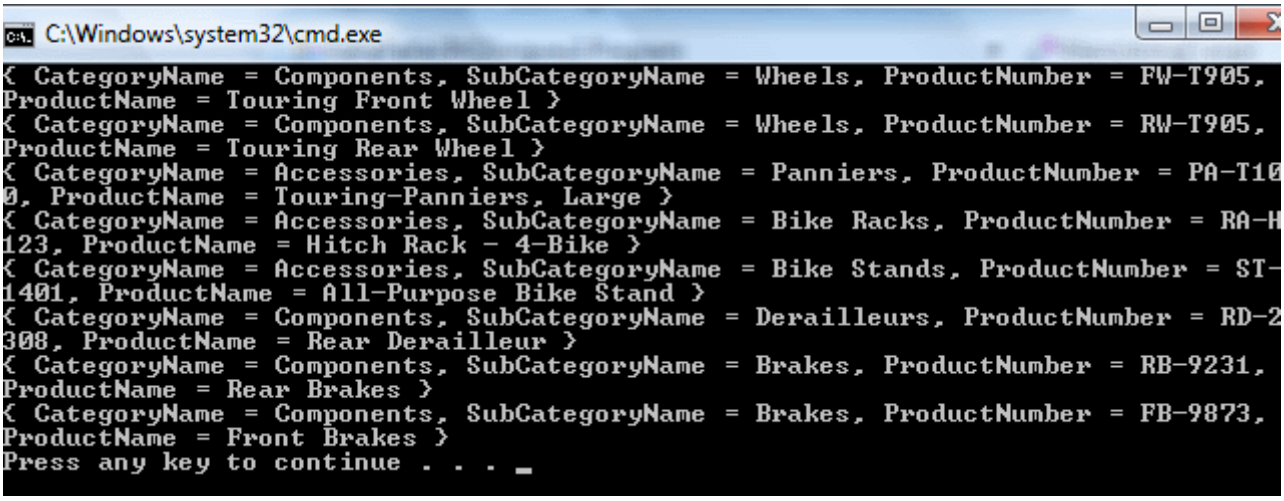
Sorgu ifadelerine dikkat edilecek olursa **Count** metodu çağırıldığında aslında SQL tarafında bir **Count** hesabı yapılmamaktadır. **Products** özelliğine geçildiğinden, o anki**ProductSubCategoryID** değerine bağlı Product satırları, programatik taraftaki nesne topluluğuna yüklenmektedir. Bu nesneler yüklendikten sonra bildiğimiz koleksiyonlara ait olan **Count** metodu çalışmakta ve toplam ürün sayıları bu şekilde elde edilmektedir. Hiç bir durumda bu tarz bir yol ile toplam sayıların elde edilmesi tercih edilmemelidir. Görüldüğü gibi gayet masumane olan ama çok işe yaradığı düşünülen basit bir kod parçası

arka tarafta son derece fazla sayıda ve yoğun sorgu cümlelerinin çalışmasına neden olmuştur.

SQL tarafında birden fazla tablo üzerinde bir arada işlem yapılması gerektiği durumlarda çoğunlukla **join** yapılarından yararlanılmaktadır. Aynı yapı bildiğiniz gibi **LINQ** ile nesneler üzerindede gerçekleştirilebilmektedir. Sıradaki örnekte LINQ to SQL için **join** kullanımına bakılmaktadır. Bu amaçla aşağıdaki gibi bir kod parçası geliştirdiğimizi düşünelim.

```
var allList = from pc in adw.ProductCategories
              join psc in adw.ProductSubcategories
                on pc.ProductCategoryID equals psc.ProductCategoryID
                  join p in adw.Products
                    on psc.ProductSubcategoryID equals p.ProductSubcategoryID
                      where p.Class == null && p.ListPrice > 100
                        select new
                        {
                          CategoryName = pc.Name
                          , SubCategoryName = psc.Name
                          , ProductNumber = p.ProductNumber
                          , ProductName = p.Name
                        };

foreach (var prd in allList)
{
   Console.WriteLine(prd.ToString());
}
```

Buradaki kod parçasına göre **ProductCategories**, **ProductSubCategories** ve **Products** özelliklerine bağlı generic **Table<T>** koleksiyonları join anahtar kelimesi yardımıyla anahtar özellikler üzerinden birleştirilmekte ve yeni bir **isimsiz tipe(Anonymous Type)** ait nesne topluluğu elde edilmektedir. Bu nesne topluluğu elde edilirken **Class** değeri **null** olan ve ürünlerin **ListPrice** değeri 100' den büyük olanların elde edilmesi sağlanmaktadır. Buna göre uygulamanın ekran çıktısı aşağıdaki gibi olacaktır.



Söz konusu LINQ to SQL ifadesinin SQL tarafındaki karşılığı ise aşağıdaki gibidir.

```
exec sp_executesql N'SELECT [t0].[Name] AS [CategoryName], [t1].[Name] AS
[SubCategoryName], [t2].[ProductNumber], [t2].[Name] AS [ProductName]
FROM [Production].[ProductCategory] AS [t0]
   INNER JOIN [Production].[ProductSubcategory] AS [t1] ON
[t0].[ProductCategoryID] = [t1].[ProductCategoryID]
      INNER JOIN [Production].[Product] AS [t2] ON ([t1].[ProductSubcategoryID]) =
[t2].[ProductSubcategoryID]
WHERE ([t2].[Class] IS NULL) AND ([t2].[ListPrice] > @p0)',N'@p0
decimal(33,4)',@p0=100.0000
```

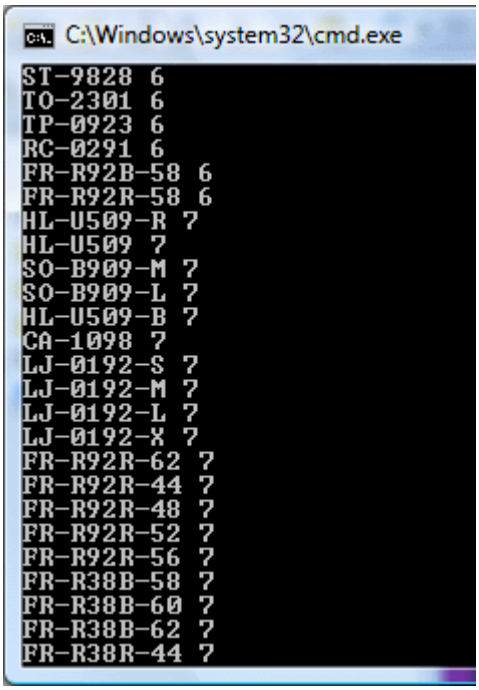
Görüldüğü gibi join kelimeleri SQL tarafında standart **inner join** muamelesi görmektedir.

Sıradaki **LINQ to SQL** ifadesinde **DateTime yapısının(struct)** parçalarından yararlanılmakta olup, bu ayrıştırmanın SQL tarafına nasıl yansıtıldığı incelenmeye çalışılmaktadır. Bu amaçla uygulamaya aşağıdaki kod parçasını eklediğimizi düşünelim.

```
var result = from p in adw.Products
             where p.SellStartDate.Month >= 6 && p.SellStartDate.Month <= 12
                select new
                       {
                          p.ProductNumber
                          , p.Name
                          , p.ListPrice
                          , p.SellStartDate
                          , p.SellEndDate
                       };

foreach (var p in result)
{
```

```
    Console.WriteLine(p.ProductNumber + " " + p.SellStartDate.Month.ToString() + " ");
}
```

Yukarıdaki kod parçasında yer alan **LINQ** ifadesinde, her bir **Product** nesne örneğinin **SellStartDate** özellikleri üzerinden hareket edilerek **Month** değerlerine bakılmakta ve 6ncı ila 12nci ay arasında olanlar değerlendirilerek yeni bir isimsiz tip(Anonymous Type) içerisinde toplanmaktadır. örneğe ait çalışma zamanı ekran çıktısı aşağıdaki gibidir.



Burada merak edilen konu, **Month** değerlerinin **SQL** tarafında nasıl ele alınacağıdır. Bu amaçla örnek çalıştırıldıktan sonra **SQL Server Profiler** aracına bakılırsa, **DatePartSQL** fonksiyonunun kullanıldığı açık bir şekilde görülebilir.

```
exec sp_executesql N'SELECT [t0].[ProductNumber], [t0].[Name], [t0].[ListPrice],
[t0].[SellStartDate], [t0].[SellEndDate]
FROM [Production].[Product] AS [t0]
WHERE
   (DATEPART(Month, [t0].[SellStartDate]) >= @p0)
      AND (DATEPART(Month, [t0].[SellStartDate]) <= @p1)'
,N'@p0 int,@p1 int',@p0=6,@p1=12
```

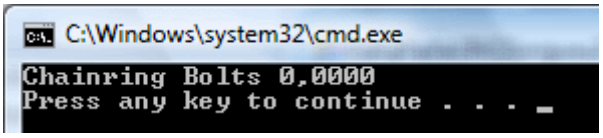
Böylece **SellStartDate** alanlarının içeriği **DatePart** fonksiyonu kullanılaraktan ayrıştırılmakta ve elde edilen **Month** kısmına görede değer aralığı kontrolü yapılmaktadır.

Bazı durumlarda sorgulanan verinin **string** bazlı olması halinde karakter tabanlı kontroller yapılmak istenebilir. Söz gelimi B harfi ile başlayan ürünlerin elde edilmesi gibi. Bu gibi durumlarda **string**bazlı verilerin karakter katarı olduğu programatik tarafta ele alınması

gereken bir durumdur. Aşağıdaki kod parçasında hem bu durum ele alınmakta hemde **First** metodunun kullanımı incelenmektedir.

```
Product result = (from p in adw.Products select p)
                 .First<Product>(prd => prd.Name[0] == 'C');

Console.WriteLine(result.Name + " " + result.ListPrice);
```

Buradaki ifadeye göre, her bir **Product** nesne örneğinin **Name** özelliklerinin ilk harflerine bakılmaktadır. İlk harfi C olan ürünlerden ise sadece ilki First metodu yardımıyla elde edilmektedir. Bu işlevselliği gerçekleştirmek için **First** metodu içerisinde **lambda(=>)** operatörü kullanılmaktadır. Lambda operatörü sayesinde eşitliğin sol tarafından sağ tarafına o anki Product nesne örneği geçirilmektedir. Eşitliğin sağ tarafında ise o anki Product nesne örneğinin **Name** özelliğinin ilk harfine bakılmaktadır. Eğer ilk harf **C** ise o anda üzerinde durulan **Product** nesne örneği eşitliğin sağından soluna doğru geri döndürülmektedir. örneğin çalışma zamanındaki ekran çıktısı aşağıdaki gibi olacaktır.



Söz konusu **LINQ to SQL** ifadesinin SQL tarafına aktarılan sorgu cümlesi ise aşağıdaki gibidir.

```
exec sp_executesql N'SELECT TOP (1)
      [t0].[ProductID], [t0].[Name], [t0].[ProductNumber]
      , [t0].[MakeFlag], [t0].[FinishedGoodsFlag], [t0].[Color]
      , [t0].[SafetyStockLevel], [t0].[ReorderPoint], [t0].[StandardCost]
      , [t0].[ListPrice], [t0].[Size], [t0].[SizeUnitMeasureCode]
      , [t0].[WeightUnitMeasureCode], [t0].[Weight], [t0].[DaysToManufacture]
      , [t0].[ProductLine], [t0].[Class], [t0].[Style], [t0].[ProductSubcategoryID]
      , [t0].[ProductModelID], [t0].[SellStartDate], [t0].[SellEndDate]
   , [t0].[DiscontinuedDate], [t0].[rowguid], [t0].[ModifiedDate]
FROM [Production].[Product] AS [t0]
WHERE
   UNICODE(CONVERT(NChar(1),SUBSTRING([t0].[Name], @p0 + 1, 1))) =
@p1',N'@p0 int,@p1 int',@p0=0,@p1=67
```

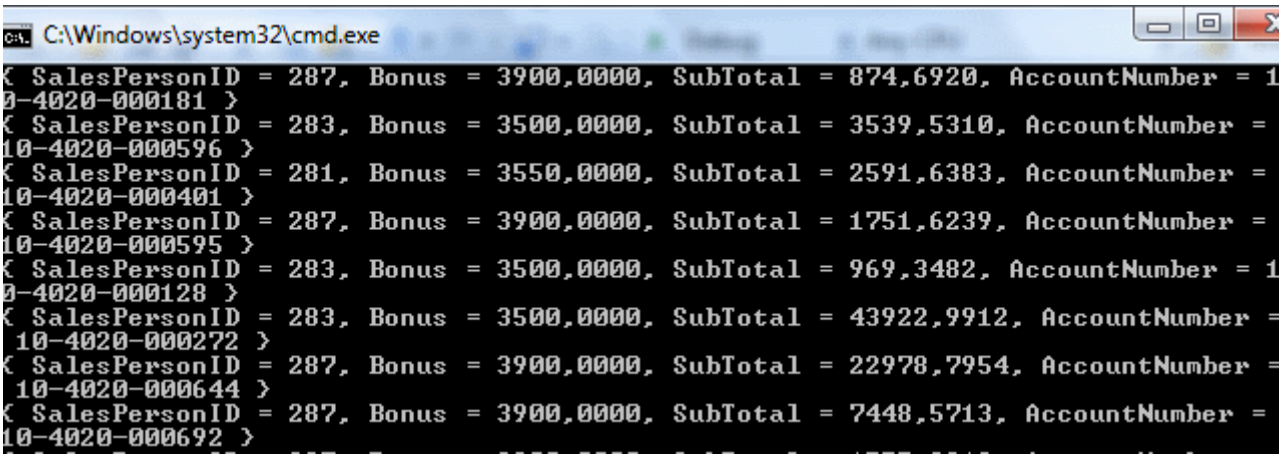
Herşeyden önce **First** metodunun tam karşılığı olarak **TOP (1)** söz dizimi kullanılmaktadır. Diğer taraftan programatik ortamda **[] indeksleyici** operatörünü kullanarak **string** veri tipinin ilk karakterine geçmemiz ve C harfini kontrol etmemizin karşılığı **Unicode**, **Convert**, **SubString SQL** fonksiyonları olmuştur. Burada 67 değerinin C harfine karşılık geldiğini hatırlayalım. Elbette çekilen veri bir **Product** tipi olduğundan, **Product** tablsoundaki tüm alanların **Select** ifadesine alındığı görülmektedir.

Daha öncedende belirtildiği gibi, sadece gerekli alanların çekilmesi adına kod tarafında **isimsiz metod(Anonymous Method)** kullanımına gidilebilir.

LINQ tarafında **çoklu seçimlerde(Select Many)** yapılabilmektedir. Bu tarz bir kullanıma örnek olarak aşağıdaki kod parçası ele alınabilir.

```
var result = from p in adw.SalesPersons
             where p.Bonus >= 1000
               from h in p.SalesOrderHeaders
                 where h.TerritoryID == 1
                   select new
                          {
                              p.SalesPersonID
                            , p.Bonus
                            , h.SubTotal
                            , h.AccountNumber
                          };

foreach (var r in result)
{
   Console.WriteLine(r.ToString());
}
```

Buradaki sorgu ifadesine göre, **SalesPersons** koleksiyonunda tutulan **SalesPerson** nesne örneklerinden **Bonus** özelliklerinin değeri 1000' in üzerinde olanlar alınmaktadır. Sonrasında ise elde edilen kümedeki her bir **SalesOrder** üzerinden **SalesOrderHeaders** koleksiyonuna gidilmekte ve bölge değeri 1 olanlar çekilmektedir. Bir başka deyişle Bonus' u 1000' in üzerinde ve sipariş kalemleri 1 numaralı bölgeye doğru yapılmış olan satış personelinin elde edilmesi söz konusudur. Elde edilen veri kümesi değerlendirilerek yeni bir **isimsiz tip(Anonymous Type)** içerisinde birleştirilmeleri sağlanmaktadır. örnek kodun çalışma zamanındaki çıktısı aşağıdaki gibi olacaktır.



Şu aşamada bizim ilgilendiğimiz kısım **SQL** tarafına gönderilen sorgu cümlesidir. Bu cümlede aşağıdaki şekildedir.

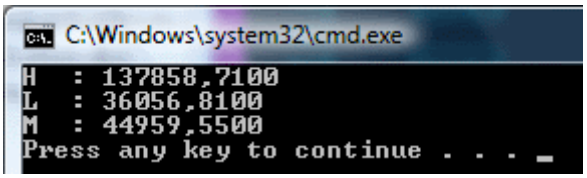
```
exec sp_executesql N'SELECT [t0].[SalesPersonID], [t0].[Bonus], [t1].[SubTotal],
[t1].[AccountNumber]
   FROM [Sales].[SalesPerson] AS [t0], [Sales].[SalesOrderHeader] AS [t1]
      WHERE ([t1].[TerritoryID] = @p0) AND ([t0].[Bonus] >= @p1)
                AND ([t1].[SalesPersonID] = [t0].[SalesPersonID])'
,N'@p0 int,@p1 decimal(33,4)',@p0=1,@p1=1000.0000
```

Burada **From** kelimesinden sonraki kısma bakıldığında **SalesPerson** ve **SalesOrderHeader** tablolarının birlikte ele alındıkları görülmektedir.

Gelelim gruplama fonksiyonelliklerinin **SQL** tarafına nasıl yansıtıldığında. Bu amaçla aşağıdaki örnek kod parçasını göz önüne alıyor olacağız.

```
var result = from p in adw.Products
             where p.Class != null
               group p by p.Class into g
                 select new
                          {
                            ClassName = g.Key
                            ,TotalListPrice = g.Sum<Product>(p => p.ListPrice)
                          };

foreach (var r in result)
{
   Console.WriteLine("{0} : {1}", r.ClassName, r.TotalListPrice);
}
```

örnekteki ifadede, **Products** koleksiyonundaki her bir **Product** nesnesinin **Class** özelliklerine göre gruplara ayrılması ve her bir gruba ait **ListPrice** özelliklerinin toplam değerlerinin bulunması sağlanmaktadır. Bir başka deyişle sınıfları olan ürünlerin sınıflara göre gruplandıklarında, toplam liste fiyatı değerlerinin ne olduğu elde edilmektedir. Bu kod parçasının icra edilmesi halinde, çalışma zamanında aşağıdakine benzer sonuç ortaya çıkmaktadır.



Görüldüğü gibi ürünler sınıflara göre gruplanmış ve toplam ürün fiyatlarının değerleri elde edilmiştir. Burada çalışan **LINQ to SQL** ifadesinin **SQL** tarafına gönderilen karşılığı ise aşağıdaki gibi olacaktır.

```
SELECT SUM([t0].[ListPrice]) AS [TotalListPrice], [t0].[Class] AS [ClassName]
FROM [Production].[Product] AS [t0]
WHERE [t0].[Class] IS NOT NULL
GROUP BY [t0].[Class]
```

Aslında üretilen **SQL** cümlesi tam olarak düşündüğümüz şekildedir. Bununla birlikte dikkat edilmesi gereken bir husus vardır. Buda **LINQ** sorgusundaki **where** kelimesinin kullanıldığı yerdir. örnekte, **where** ifadesi ile seçilen küme üzerinde gruplama yapılmaktadır. Bu nedenle **group by** kelimesinden önce where kullanılmaktadır. Ancak aynı ifade aşağıdaki haliylede geliştirilebilir.

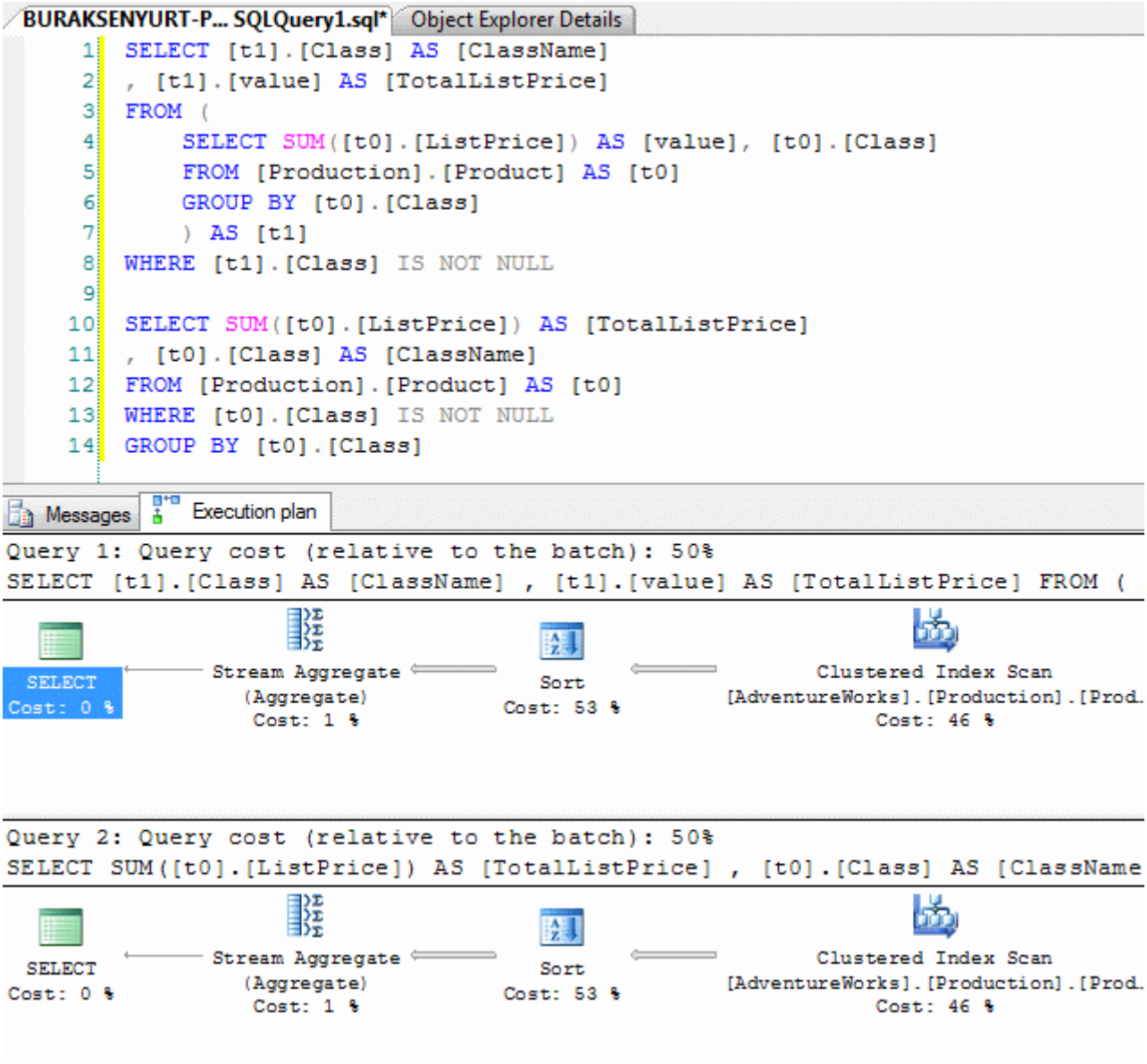
```
var result = from p in adw.Products
             group p by p.Class into g
               where g.Key!=null
                 select new
                        {
                          ClassName = g.Key
                          ,TotalListPrice = g.Sum<Product>(p => p.ListPrice)
                        };

foreach (var r in result)
{
  Console.WriteLine("{0} : {1}", r.ClassName, r.TotalListPrice);
}
```

Bu sefer gruplanan nesneye ait **Key** özelliğinin **null** olup olmadığına bakılmaktadır. Kod bu haliyle çalıştırıldığında da bir önceki ile aynı sonuçların elde edildiği görülebilir. Ne varki **SQL** tarafına gönderilen ifadeye bakıldığında aşağıdaki sonuçlar ortaya çıkmaktadır.

```
SELECT [t1].[Class] AS [ClassName], [t1].[value] AS [TotalListPrice]
FROM (
        SELECT SUM([t0].[ListPrice]) AS [value], [t0].[Class]
        FROM [Production].[Product] AS [t0]
        GROUP BY [t0].[Class]
) AS [t1]
WHERE [t1].[Class] IS NOT NULL
```

Sonuç bir öncekinden oldukça farklıdır. Bu kez devreye ek bir alt sorgu cümlesi daha girmektedir. önce sınıflara göre gruplanmış ürünlerin **ListPrice** değelerinin toplamları ve sınıf adlarının olduğu küme elde edilmektedir. Sonrasında ise bu küme üzerinden **Class** değerleri **null** olmayanlar çekilmektedir. Bu noktada **where** kelimesinin

kod tarafından yerinin değiştirilmesinin önemli olup olmadığına karar vermek gerekebilir. Ancak geliştirilen örneğe ait oluşturulan sorguların **icra planlarına(Execution Plan)** bakıldığında bir fark olmadığı açıkça görülmektedir.



LINQ tarafında yer alan enteresan metodlardan biriside **Except** metodudur. Bu metoddan yararlanılarak belirli bir şartın dışında kalan nesnel kümelerin elde edilmesi sağlanabilir. örnek olarak aşağıdaki gibi bir kod parçası geliştirdiğimizi düşünelim.

```
var result = (from c in north.Customers select c.City)
              .Except(from s in north.Suppliers select s.City);

foreach (var r in result)
{
   Console.WriteLine(r);
}
```

Bu örnekte **NorthwindDataContext** kullanılmaktadır. Buna göre **LINQ** ifadesinde ilk parantez içerisinde kalan kısımda **Customers** koleksiyonunda duran **Customer** nesne örneklerinden **City**özellikleri çekilmektedir. **Except** metodunda yazılan ifadede **Suppliers** tablosunda yer alan **Supplier** nesne örneklerinden **City** özelliklerini çekmektedir. Her iki küme bir arada düşünüldüğünde ortaya çıkan sonuç şudur; Customer nesne örneklerinde olup, Supplier nesne örneklerinde bulunmayan **City** özellikleri elde edilmektedir. Daha düzgün bir ifadeyle, bir başka deyişle **SQL**' ce düşünüşdüğünde, müşterilerin yaşayıpta tedarikçilerinin bulunmadığı şehir adlarının elde edildiğini söyleyebiliriz. Programın çalışma zamanındaki çıktısı aşağıdaki gibidir.

Bu tarz bir ihtiyacı SQL tarafında karşılamak için **Not In** kullanımı tercih edilebilir. **LINQ** tarafında metod bazlı yazılan bu örnek ise, **SQL** tarafına aşağıdaki şekilde aktarılmaktadır.

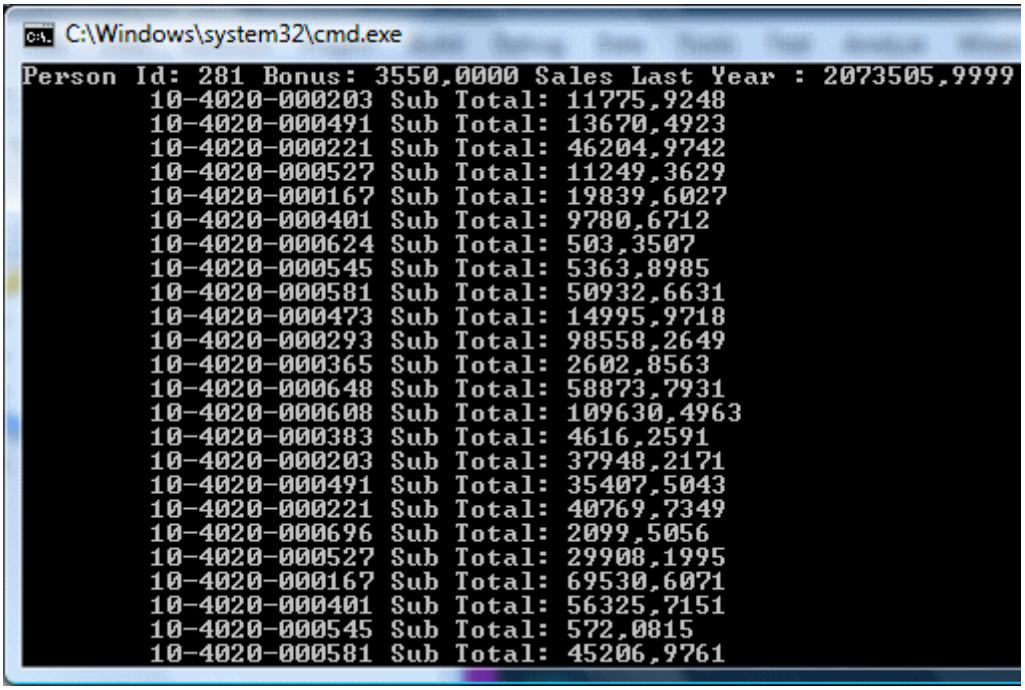
```
SELECT DISTINCT [t0].[City]
FROM [dbo].[Customers] AS [t0]
WHERE
   NOT (EXISTS
        (
      SELECT NULL AS [EMPTY]
         FROM [dbo].[Suppliers] AS [t1]
            WHERE (([t0].[City] IS NULL)
               AND ([t1].[City] IS NULL))
                  OR (([t0].[City] IS NOT NULL)
                     AND ([t1].[City] IS NOT NULL)
                        AND ([t0].[City] = [t1].[City]))
        )
      )
```

Burada önemli olan **Exists SQL** fonksiyonu ile gereken işlevselliğin sağlanmış olmasıdır. **Not** konulmasının sebebi, **Exists** ile belirtilen alt sorgudaki koşula uyanların dışarıda bırakılmasını sağlamaktır. Nitekim t0 ve t1 tablolarındaki City değerlerine bakılarak eşit olanların elde edilmesi sağlanırken **Except** metodu kullanılması nedeniyle bunların dışarıda tutulmasını ancak **Not** anahtar kelimesi sağlayabilmektedir. Ayrıca hem **Suppliers** hemde **Customers** tablosundaki **City** alanları için detaylı bir **Null** kontrolü yapılmaktadır.

Makalemize yine enteresan **LINQ** fonksiyonları ile devam edelim. Bu kez **Any** ve **All** isimli metodları incelemeye çalışıyor olacağız. Bu amaçla ilk olarak Any metodunun kullanımına kısaca bakılım.

```
 var result = from p in adw.SalesPersons
                  where p.SalesOrderHeaders.Any(soh => soh.SubTotal >= 224356)
                    select p;

foreach (SalesPerson person in result)
{
   Console.WriteLine("Person Id: " + person.SalesPersonID + " Bonus: " + person.Bonus +
" Sales Last Year : " + person.SalesLastYear);
   foreach (SalesOrderHeader header in person.SalesOrderHeaders)
      Console.WriteLine("\t" + header.AccountNumber + " Sub Total: " +
header.SubTotal);
}
```

LINQ to SQL ifadesinde **SalesPersons** koleksiyonundaki her bir **SalesPerson** çekilmektedir. Bunlara bağlı olan **SalesOrderHeaders** koleksiyonundaki **SalesOrderHeader** nesne örneklerinin ise **SubTotal**değerlerine bakılarak seçim işlemi koşullandırılmaktadır. Any metodunun buradaki görevi ise şudur; **SubTotal** değerlerinden herhangibiri 224356' nın üzerinde olan satırlar elde edilebilmektedir. Yani, satış personelinin siparişlerine ait SubTotal değerlerinden herhangibiri **224356**' nın üzerinde olanların elde edilmesi sağlanmaktadır. örnek kod parçasının çalışma zamanındaki çıktısı aşağıdaki gibi olacaktır.



Bu tarz bir işleyiş için **SQL** tarafı göz önüne alındığında ortaya karmaşık bir sorgu çıkacağı düşünülebilir. Nitekim **LINQ** to SQL ifadesinin SQL tarafındaki karşılığı aşağıdaki gibidir.

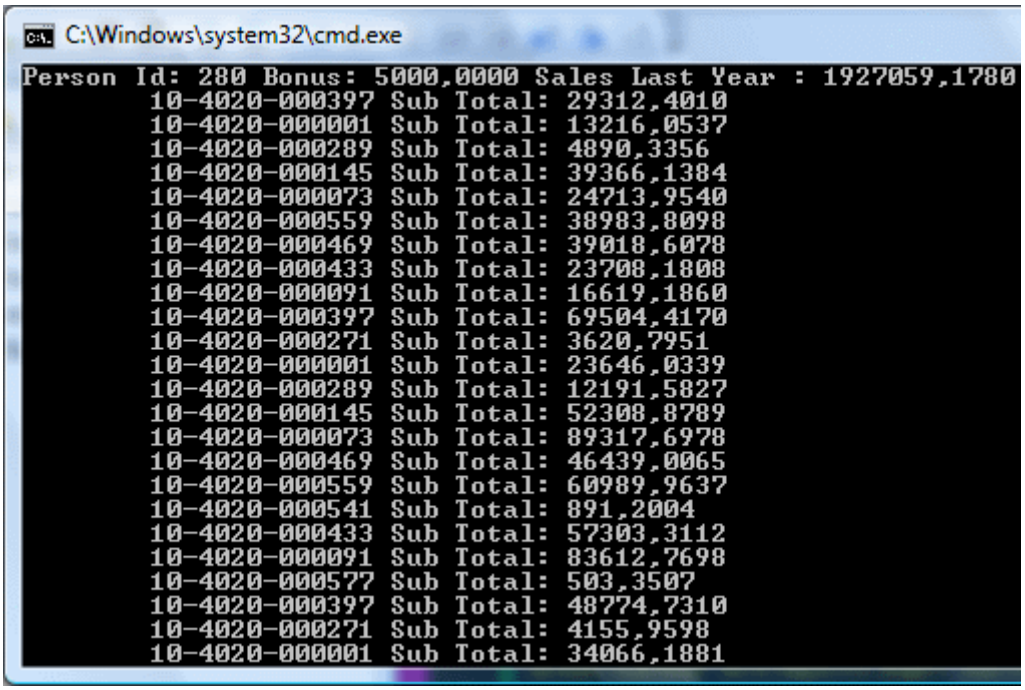
```
exec sp_executesql N'
SELECT
   [t0].[SalesPersonID], [t0].[TerritoryID], [t0].[SalesQuota]
   , [t0].[Bonus], [t0].[CommissionPct], [t0].[SalesYTD]
   , [t0].[SalesLastYear], [t0].[rowguid], [t0].[ModifiedDate]
FROM [Sales].[SalesPerson] AS [t0]
   WHERE EXISTS(
                        SELECT NULL AS [EMPTY]
                        FROM [Sales].[SalesOrderHeader] AS [t1]
                        WHERE ([t1].[SubTotal] >= @p0) AND ([t1].[SalesPersonID] =
[t0].[SalesPersonID])
   )'
,N'@p0 decimal(33,4)',@p0=224356.0000
```

Dikkat edileceği üzere **Exists** anahtar kelimesi kullanılarak **SubTotal** değeri ele alınmakta ve buna uyanların **SalesPerson** tablosundan çekilmesi sağlanmaktadır.
Gelelim **All** metoduna. Bu sefer **Any**' den farklı olarak bağlı olunan kümedeki her bir eleman için belirtilen koşulun sağlanmış olma şartı aranmaktadır. Bunu daha net kavrayabilmek için örneğimizi aşağıdaki gibi değiştirelim.

```
var result = from p in adw.SalesPersons
             where p.SalesOrderHeaders.All(soh => soh.SubTotal >= 80)
               select p;

foreach (SalesPerson person in result)
```

```
{
   Console.WriteLine("Person Id: " + person.SalesPersonID + " Bonus: " + person.Bonus +
" Sales Last Year : " + person.SalesLastYear);
   foreach (SalesOrderHeader header in person.SalesOrderHeaders)
      Console.WriteLine("\t" + header.AccountNumber + " Sub Total: " +
header.SubTotal);
}
```

Bu kodun çalışma zamanı çıktısı ise aşağıdaki gibi olacaktır.



Bu sefer bir **SalesOrder**' ın bağlı olduğu **SalesOrderHeaders** koleksiyonundaki her bir **SalesOrderHeader** nesne örneğinin **SubTotal** değerlerinin her biri 80' in üzerinde olanların elde edilmesi sağlanmaktadır. Bir başka deyişle bir **SalesOrder** üzerinden ulaşılan nesne topluluğunda n tane **SalesOrderHeader** olduğu düşünülecek olursa, bunların her birine ait **SubTotal**özelliklerinin değerlerinin 80 ve üzerinde olma şartı konulmaktadır. Söz konusu **All** metodu için SQL tarafında üretilen çıktı ise aşağıdaki gibidir.

```
exec sp_executesql N'
SELECT
   [t0].[SalesPersonID], [t0].[TerritoryID], [t0].[SalesQuota]
   , [t0].[Bonus], [t0].[CommissionPct], [t0].[SalesYTD]
   , [t0].[SalesLastYear], [t0].[rowguid], [t0].[ModifiedDate]
FROM [Sales].[SalesPerson] AS [t0]
WHERE NOT (
                EXISTS(
                        SELECT NULL AS [EMPTY]
                        FROM [Sales].[SalesOrderHeader] AS [t1]
```

```
                    WHERE ((
                                (CASE
                                   WHEN [t1].[SubTotal] >= @p0 THEN 1 ELSE
0
                                END)) = 0)
                                AND ([t1].[SalesPersonID] = [t0].[SalesPersonID])
))'
,N'@p0 decimal(33,4)',@p0=80.0000
```

Bu kez koşulun kontrolü için **Case** ifadesinden yararlanılmakta ve **SubTotal** 80' üzerinde ise 1, değilse 0 değeri **Where** ifadesine katılarak 0 olanların çekilmesi sağlanmaktadır. Yanlız burada yine Not Exist kullanıldığında dikkat etmekte yarar vardır. Buna görede bir **Where** ifadesinin **SalesPerson** tablosu için üretilmesi sağlanmaktadır.

örneklerimize **Concat** metodu ile devam edelim. Concat metodunu daha çok iki farklı sonuç kümesindeki belirli özelliklerin bir arada ele alınmasını istediğimiz durumlarda göz önüne alabiliriz. Bu bir anlamda iki **string**' in birleştirilemesine benzer bir durumdur. Tabi şu anda söz konusu olan string değil **IEnumerable** gibi referanslardır. **Concat** metodunun SQL tarafında ürettiği çıktıya bakmak için aşağıdaki örnek kod parçasını geliştirdiğimizi düşünelim.

```
var result = (from cust in north.Customers select new { cust.Country, cust.City })
                .Concat(from supl in north.Suppliers select new { supl.Country, supl.City });
foreach (var r in result)
{
   Console.WriteLine(r.Country + ":" + r.City);
}
```

Bu örnekte **NorthwindDataContext** tipi kullanılmakta
olup **Customers** ve **Suppliers** koleksiyonlarındaki nesnelerde **Country** ve **City** değerleri birleştirilip çekilmektedir. Sonuçta kodun çalışma zamanı çıktısı aşağıdaki gibi olacaktır.

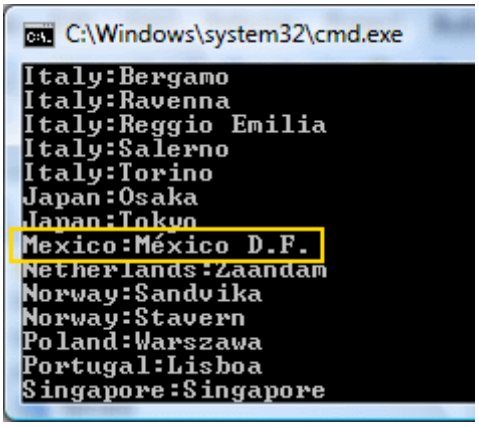
Elbetteki SQL tarafına bakıldığında **Concat** metodunun aşağıdakine benzer bir dönüşüme uğradığı görülmektedir.

```
SELECT [t2].[Country], [t2].[City]
FROM (
        SELECT [t0].[Country], [t0].[City]
        FROM [dbo].[Customers] AS [t0]
           UNION ALL
        SELECT [t1].[Country], [t1].[City]
        FROM [dbo].[Suppliers] AS [t1]
) AS [t2]
```

Açıkça **Union All** kullanılaraktan iki **Select** ifadesinin birleştirildiği ve elde edilen küme üzerinden **Country** ile **City** alanlarına ait değerlerin çekildiği söylenebilir. örnekte dikkati çeken noktalardan biriside tekrarlı alanların olmasıdır. örneğin ekran çıktısının üst taraflarına bakıldığında iki adet Mexico kentinin olduğu London' un iki kere geçtiği rahat bir şekilde görülebilir. çok doğal olarak tekrarsız bir listenin elde edilmesi istendiğinde kod tarafında **Distinct** metodunun kullanılıyor olması yeterli olacaktır. Yani kod parçasında aşağıdaki değişikliğin yapılması yeterlidir.

```
var result = (from cust in north.Customers select new { cust.Country, cust.City
}).Concat(from supl in north.Suppliers select new { supl.Country, supl.City }).Distinct();
```

Bu durumda programın çıktısı aşağıdaki gibi olacaktır.



Diğer taraftan **Distinct** metodunun kullanılması sonrasında **SQL** tarafına gönderilen sorgu cümlesinde ise **Distinct** anahtar kelimesinin kullanıldığıda aşikardır.

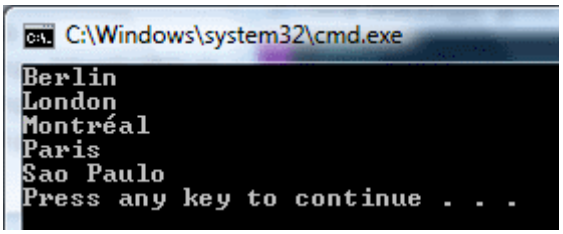
```
SELECT DISTINCT [t3].[Country], [t3].[City]
FROM (
        SELECT [t2].[Country], [t2].[City]
        FROM (
                SELECT [t0].[Country], [t0].[City]
                FROM [dbo].[Customers] AS [t0]
                   UNION ALL
                SELECT [t1].[Country], [t1].[City]
                FROM [dbo].[Suppliers] AS [t1]
        ) AS [t2]
) AS [t3]
```

Ancak burada bir öncekinden farklı bir sorgunun oluştuğuda gözlerden kaçmamalıdır. Bu kez iç içe alınmış **Select** sorgusu söz konusudur. Oysaki en dışta yer alan **Select** kullanımına gerek yoktur. çünkü **Distinct** anahtar kelimesi içerideki sorgu cümlesine eklenerekte aynı sonuçların alınması sağlanabilir. Ne varki sorgu cümlesini bu şekilde oluşturup SQL tarafına gönderen **LINQ to SQL** mimarisidir.

Yine ilginç bir LINQ metodu ve SQL karşılığı ile devam edelim. Bu kez iki farklı veri kümesinin kesişimlerinin elde edilmesinde kullanılabilen **Intersect** metodu üzerinde duracağız. Bu metodun analizi için aşağıdaki gibi bir kod parçası geliştirdiğimizi düşünelim.

```
var result = (from c in north.Customers select c.City)
                .Intersect(from s in north.Suppliers select s.City);

foreach (var r in result)
{
   Console.WriteLine(r);
}
```

öncelikli olarak ilk parantezler arasında **Customers** koleksiyonundaki her bir **Customer** nesnesinin **City** değerleri çekilmektedir. İkinci parantez içerisinde yapılanda benzerdir. Tek farkı **Suppliers**koleksiyonu için çalışmakta olmasıdır. **Intersect** metodunun burada getridiği kolaylık ise şudur. **Customers** ve **Suppliers** koleksiyonlarında bulundan ortak **City** özelliklerinin elde edilmesini sağlamaktadır. Bir başka deyişle yine SQL' ciler gibi konuşacak olursak, müşterilerin ve tedarikçilerin bir arada bulunduğu şehirlerin elde edilmesi sağlanmaktadır . Buna göre örneğin çalışma zamanındaki ekran çıktısı aşağıdaki gibi olacaktır.



Bu sonucun elde edilmesi için arka planda çalıştırılan SQL cümlesi ise aşağıdaki gibidir.

```
SELECT DISTINCT [t0].[City]
FROM [dbo].[Customers] AS [t0]
WHERE EXISTS(
                    SELECT NULL AS [EMPTY]
                    FROM [dbo].[Suppliers] AS [t1]
                    WHERE
                       (([t0].[City] IS NULL)
                          AND ([t1].[City] IS NULL))
                             OR (([t0].[City] IS NOT NULL)
                                AND ([t1].[City] IS NOT NULL)
                                   AND ([t0].[City] = [t1].[City]))
)
```

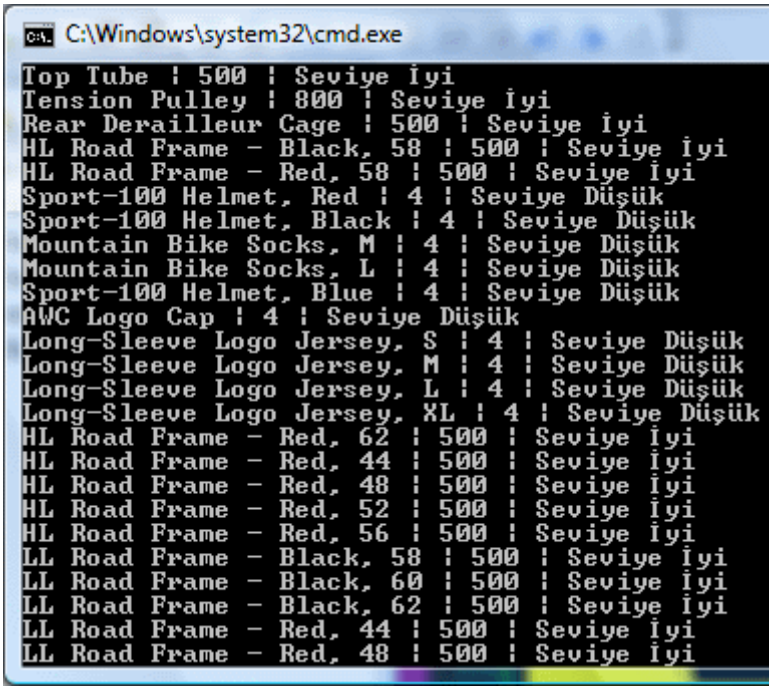
çalıştırılan SQL sorgusu, LINQ tarafında **Except** metodu kullanıldığı zamankine benzerdir. Tek fark burada kesişim kümesinin bulunması gerektiğinden **Not Exists** kullanılmamış olmasıdır.

Makalemizde son olarak **?:** operatörünün kullanıldığı bir durumu ele almaya çalışıyor olacağız. Bu operatör şu aşamada LINQ' e bağımlı olmayan **C#** programlama dilinin ilk versiyonundan beri var olan bir araçtır. Bu tip bir operatörün **LINQ** ifadesi içerisinde kullanılması haline **SQL** tarafında oluşacak olan cümlelere bakmaya çalışıyor olacağız. Bu amaçla aşağıdaki kod parçasını geliştirdiğimizi düşünelim.

```
 var result = from prd in adw.Products
                 select new
                          {
                             prd.Name
                             ,prd.SafetyStockLevel
```

```
                                ,LevelOk = prd.SafetyStockLevel >= 50 ? "Seviye İyi" :
"Seviye Düşük"
                            };

foreach (var p in result)
{
   Console.WriteLine(p.Name + " | " + p.SafetyStockLevel + " | " + p.LevelOk);
}
```

Bu kod parçasında kullanılan LINQ ifadesine bakıldığında, **LevelOk** isimli isimsiz tip özelliğinin değerinin **SafetyStockLevel** özelliğinin değerine göre belirlendiği görülmektedir.**SafetyStockLevel** özelliğinin değerinin 50 ve üzerinde olması halinde LevelOk özelliğine Seviye İyi değeri atanmaktadır. Aksi durumda ise Seviye Düşük değeri atanmaktadır. Kodun çalışma zamanındaki çıktısı aşağıdaki gibi olacaktır.



SQL tarafına baktığımızda ise aşağıdaki sorgu cümlesinin çalıştırıldığı görülmektedir.

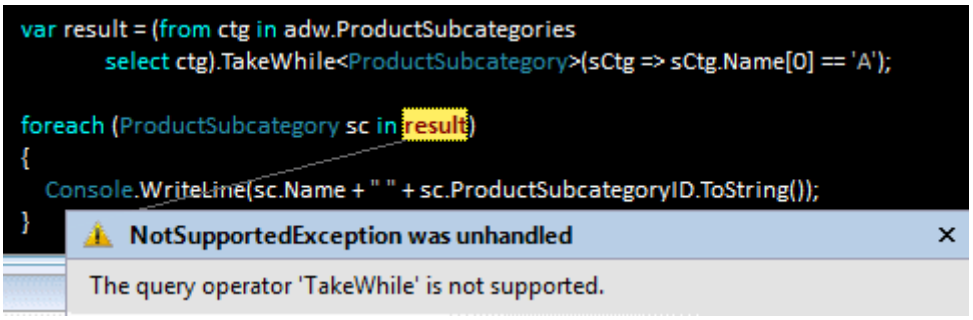
```
exec sp_executesql N'SELECT [t0].[Name], [t0].[SafetyStockLevel],
   (CASE
      WHEN [t0].[SafetyStockLevel] >= @p0 THEN CONVERT(NVarChar(12),@p1)
ELSE @p2 END
   )
   AS [LevelOk]
FROM [Production].[Product] AS [t0]'
   ,N'@p0 int,@p1 nvarchar(10),@p2 nvarchar(12)',@p0=50,@p1=N'Seviye
İyi',@p2=N'Seviye Düşük'
```

Dikkat edileceği üzere, **LevelOk** alanının elde edilmesi sırasında **Case When SQL** ifadesi kullanılmaktadır.

Buraya kadar anlatılan örneklerde LINQ operatörlerinden veya metodlarından bir kısmının SQL tarafına nasıl aktarıldıkları incelenmeye çalışılmıştır. Diğer taraftan makalemizin başındada belirtildiği üzere programatik tarafta kullanılan her tür LINQ operatörü veya fonksiyonunun SQL tarafına aktarılmasıda mümkün değildir. Söz gelimi aşağıdaki kod parçasını ele aldığımızı düşünelim.

```
 var result = (from ctg in adw.ProductSubcategories select ctg)
                .TakeWhile<ProductSubcategory>(sCtg => sCtg.Name[0] == 'A');

foreach (ProductSubcategory sc in result)
{
   Console.WriteLine(sc.Name + " " + sc.ProductSubcategoryID.ToString());
}
```

Bu kod parçası yürütülmek istendiğinde çalışma zamanında(run time), aşağıdaki ekran görüntüsündende izlenebileceği gibi **NotSupportedException** tipinden bir**istisna(Exception)** alınmaktadır.



Nitekim TakeWhile metodunun SQL tarafında bir karşılığı yoktur. **TakeWhile** gibi **SkipWhile**, **Last**, **ElementAt**, **Reverse** gibi pek çok metodunda SQL tarafında karşılığı bulunmadığından desteklenmemektedirler.

Sonuç olarak programatik tarafta **varlık katmanı(Entity Layer)** üzerinde işlemlerimizi oldukça kolaylaştıran ve nesneler üzerinde sorgular çalıştırabilmemizi sağlayan **LINQ to SQL**' in gücü ortadadır. Ne varki performansın öne geçmesi gereken durumlarda, yazılan LINQ ifadelerinin arka planda oluşturduğu SQL çıktıları değerlendirilmeli ve en doğru şekilde kullanılmalarına gayret edilmelidir. Zaten zaman içerisinde benzer vakalar için en uygun LINQ söz dizimlerinin ne olacağı daha net bir şekilde ortaya çıkacaktır. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

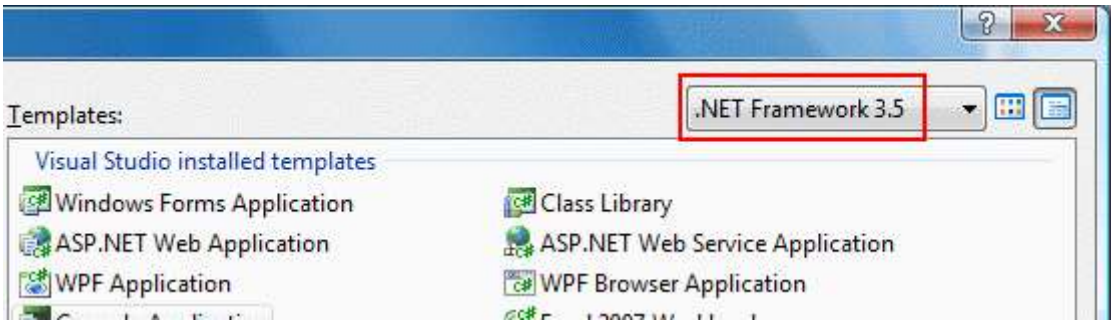
DahaFazlaLINQSorgusu2.rar (104,12 kb)

LINQ to SQL ile CRUD İşlemleri (2007-12-15T04:04:00)

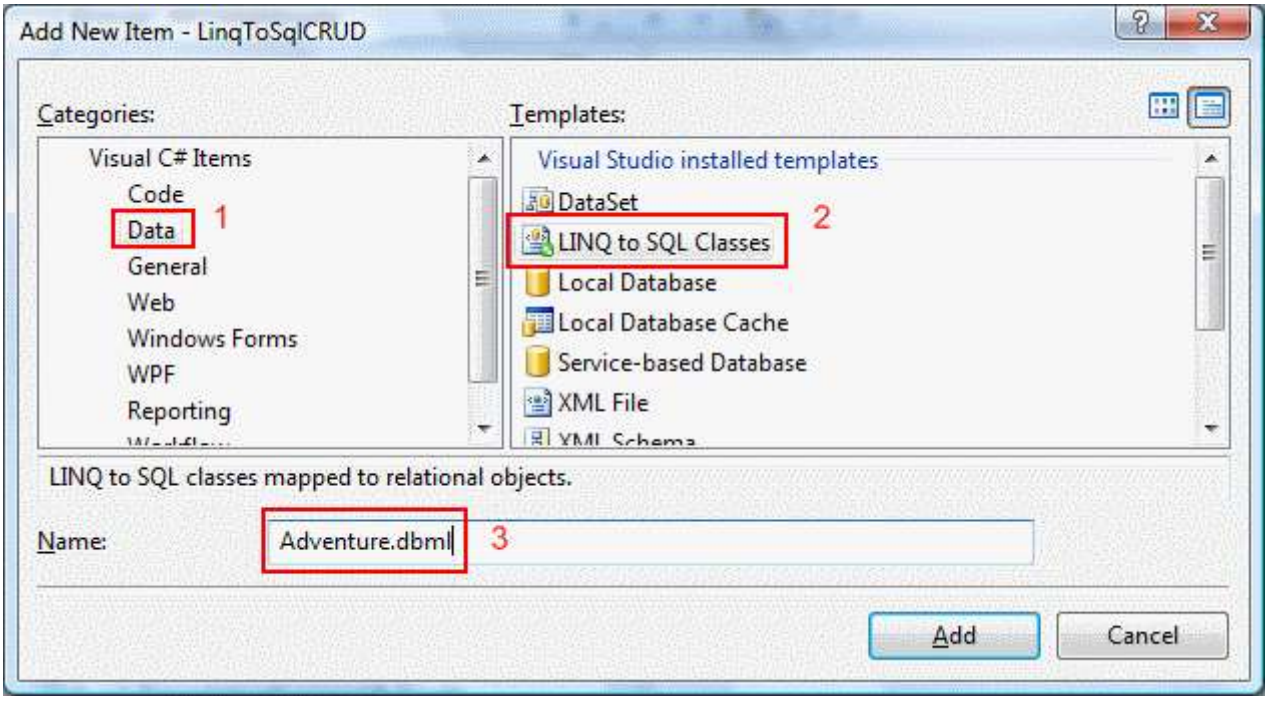
linq to sql,linq,c# 3.0,

Language Integrated Query(LINQ) mimarisi özellikle programatik ortamlarda tasarlanan nesneler üzerinde, **SQL** cümlelerine benzer ifadeler ile sorgulamalar yapılmasına izin vermektedir. çok doğal olarak **veritabanı(database)** tarafında yer alan **tablo(Table), saklı yordam(Stored Procedure), görünüm(View), fonksiyon(Function)** gibi unsurlarında programatik tarafta birer **varlık(Entity)**olarak ifade edilebilmesi, LINQ kurallarının SQL üzerindede gerçekleştirilebilmesini sağlamaktadır. Burada **varlık katmanı(Entity Layer)** olarakda düşünebileceğimiz yapı üzerinde yer alan nesneler, veritabanından çekilen sonuçları saklayabilmektedir. Bunun yanında programatik ortamdaki varlıklar üzerinde yeni varlık oluşturma, güncelleme, silme gibi operasyonlarda yapılabilmektedir. İşte bu makalemizde çoğunlukla **CreateRetrieveUpdateDelete (CRUD)** işlemleri olarak belirtilen bu operasyonları nasıl yapabileceğimizi, adım adım basit örnekler üzerinden incelemeye çalışıyor olacağız. *(Bu makalede geliştirilmekte olan örnek kod parçaları **Visual Studio 2008 RTM** ortamında yazılmıştır.)*

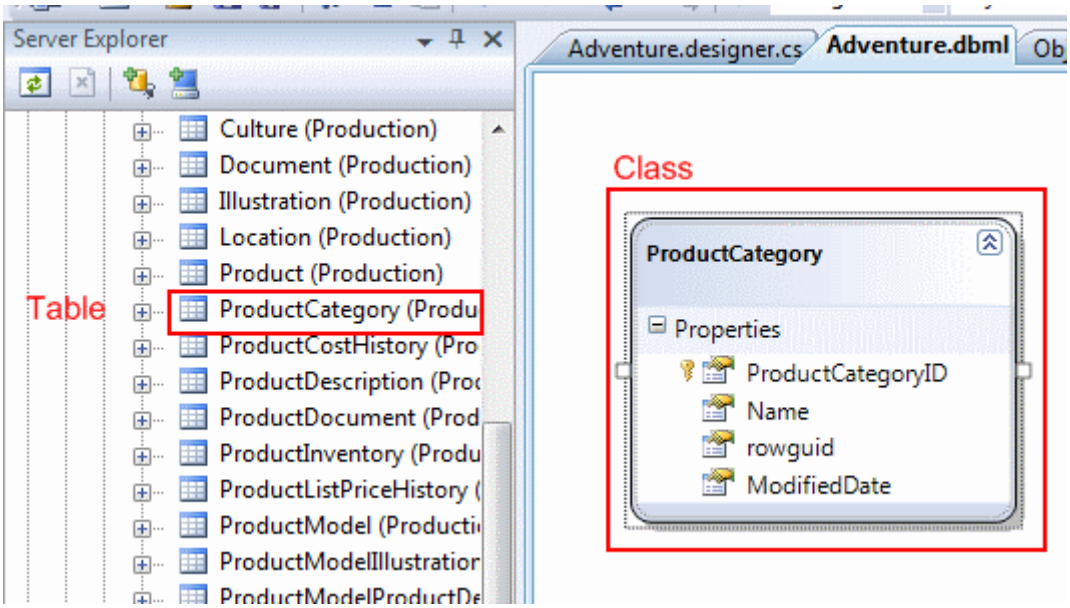
İlk olarak örnek bir **SQL** veritabanındaki **tablo(table)** yapılarını programatik ortamda taşıyacak olan sınıfların üretilmesi gerekmektedir. Bu amaçla **Visual Studio 2008** üzerinde basit bir**Console** uygulaması açarak ilerleyebiliriz. Burada dikkat edilmesi gereken önemli noktalardan birisi **New Project** seçimi sonrası karşımıza çıkacak olan iletişim penceresinden **.Net Framework 3.5** versiyonunun işaretlenmiş olmasıdır.



Bu şart olmamakla birlikte, **LINQ** için gerekli olan **assembly**'
ların(örneğin **System.Data.DataSetExtensions** gibi) otomatik olarak referans edilmesini sağlamaktadır. Bu adımdan sonra **entity** sınıflarının kolay bir şekilde oluşturulmasını sağlayan **LINQ To SQL Class** öğesini projemize eklememiz gerekmektedir.*(LINQ to SQL sınıflarının oluşturulması ile ilişkili detaylı bilgiyi C#Nedir? yer alan görsel dersten öğrenebilirsiniz.)*

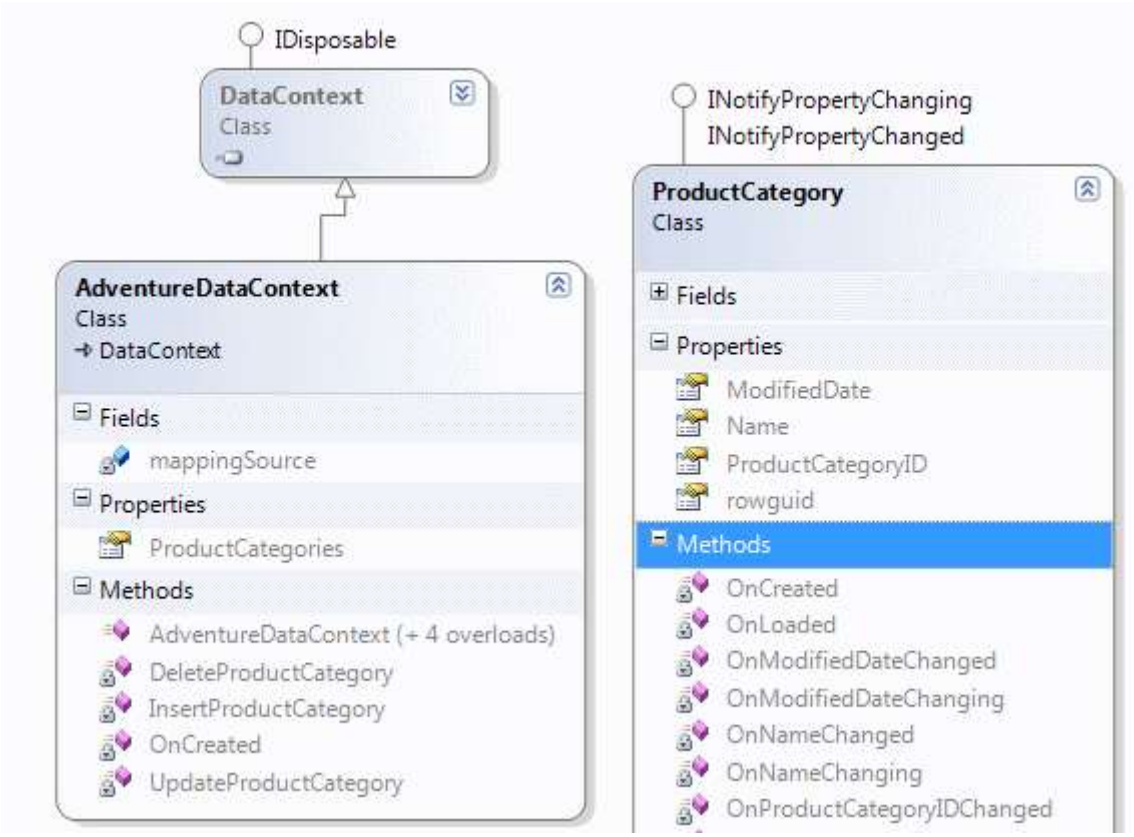


Böylece veritabanı üzerindeki nesnel yapıları programatik ortamda ifade edebileceğimiz **Database Markup Language(dbml)** dosyası otomatik olarak oluşturulmaktadır. **Adventure.dbml** dosyasının kod tarafına bakıldığında **DataContext** tipinden türetilmiş olan bir sınıfın yazıldığı görülmektedir. Şimdi yapmamız gereken, üzerinde işlemler gerçekleştirilecek olan veritabanı nesnelerini tasarım ortamına sürükleyip bırakmaktır. İlk etapta örneğin basit olması açısından sadece **Production** şemasında(schema) yer alan **ProductCategory** isimli tablo tasarım ortamına, **Server Explorer** pencersinden sürüklenmiştir.



Bu basit operasyonun ardından aşağıdaki **sınıf diagramında(Class Diagram)** olduğu gibi **ProductCategory** için bir tipin yazıldığı ve bununla ilişkili olaraktanda, **DataContext** tipinden türeyen **AdventureDataContext** sınıfı

içerisine **ProductCategories** isimli bir özelliğin(Properties) atıldığı görülmektedir. ProductCategories özelliği generic Table<ProductCategory> tipinden bir nesne referansını işaret etmektedir. Bu generic tip tahmin edileceği üzere ProductCategory tabosundaki tüm içeriği taşıyan sınıftır.



Artık bu noktadan sonra **AdventureDataContext** sınıfına ait nesne örneklerini kullanarak kategorilerin elde edilmesi, sorgulanması, yeni kategorilerin oluşturulması(Insert), var olanlardan bir veya bir kaçının silinmesi(Delete) yada güncellenmesi(Update) gibi işlemler kolaylıkla yapılabilir. İlk olarak yeni bir kategoriyi nasıl ekleyebileceğimizi örnek bir kod parçası üzerinden incelemeye çalışalım. Bu amaçla **Main** metodu içerisine aşağıdaki kod satırlarını eklediğimizi düşünelim.

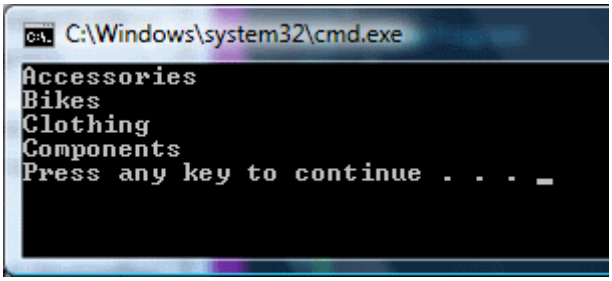
```
AdventureDataContext adwContext = new AdventureDataContext();

ProductCategory tools = new ProductCategory()
                                                {
                                                   Name = "Tools"
                                                   , ModifiedDate=new DateTime(2001,1,1)
                                                };

adwContext.ProductCategories.InsertOnSubmit(tools);
```

İlk olarak **AdventureDataContext** tipine ait bir nesne örneği oluşturulmaktadır. Bu işlemin arkadasındanda eklenmek istenen **ProductCategory** nesne örneği **C# 3.0** ile

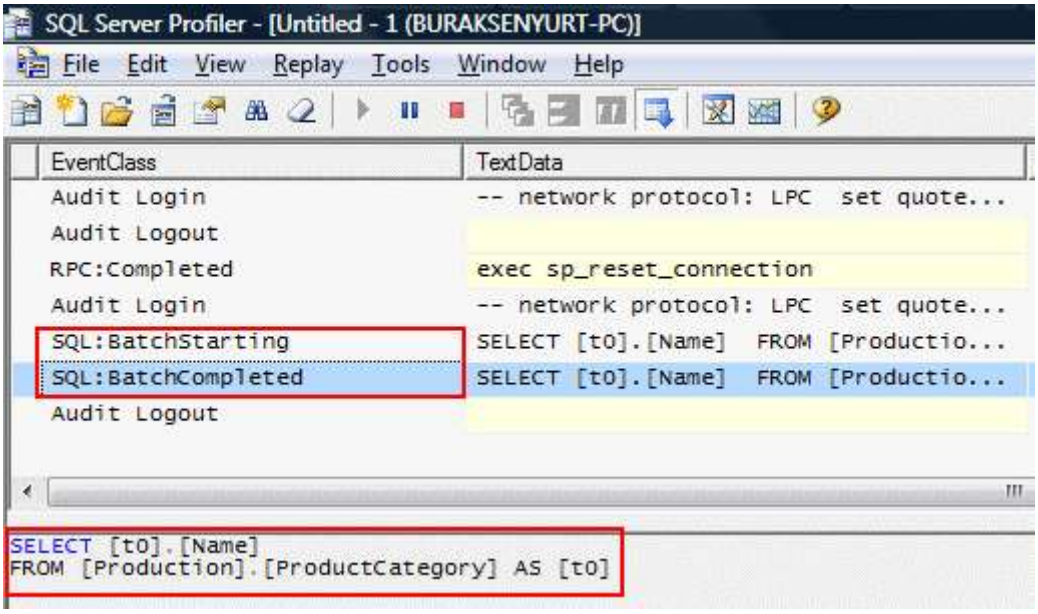
birlikte gelen **nesne başlatıcılarından(Object Initializers)** yararlanılarak örneklenmektedir. Bu noktada **Table<T>** generic tipinden olan **ProductCategories** sınıfının **InsertOnSubmit** metodu, koleksiyonuna ilave edilmek üzere yeni bir ProductCategory nesne örneğinin eklenmesi için kullanılmaktadır. Burada üzerinde durulması gereken önemli bir nokta vardır. **InsertOnSubmit** metodu sadece Table<T> koleksiyonuna ilave edilmek üzere bir nesne eklemektedir. Bir başka deyişle ProductCategories koleksiyonu üzerinde yapılacak bir for döngüsünde eklenen nesne(ler) görülmeyecektir. Bunu test etmek için aşağıdaki gibi bir kod parçasını ele alabiliriz.

```
var categories = from category in adwContext.ProductCategories
                 select category.Name;

foreach(string ctgr in categories)
   Console.WriteLine(ctgr);
```

Bunun sonucu olarak çalışma zamanında(run time) aşağıdaki ekran görüntüsünü elde ederiz.

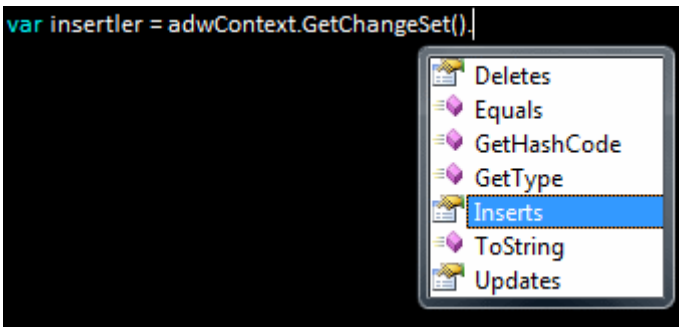


Burada sebep son derece açıktır. **var** anahtar kelimesinden(keyword) sonra **foreach** döngüsünün iterasyona başlamasıyla birlikte **SQL** sunucusu üzerinde bir **select** sorgusu çalışamaktadır. Bu sorgu **SQL Profiler** yardımıyla kolay bir şekilde yakalanabilir. Aşağıdaki ekran görüntüsünde bu durum görülmektedir.

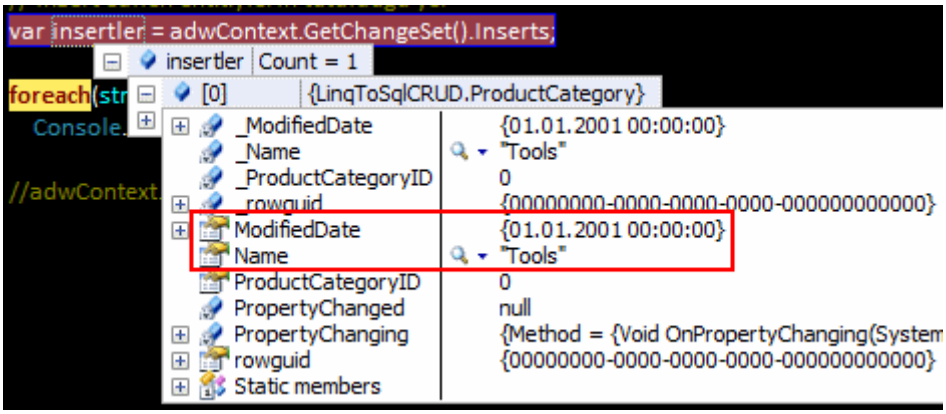


Dolayısıyla **eklenmek(Insert)** üzere ilave edilen ProductCategory nesne örneği henüz veritabanına doğru gönderilmemiştir. Bu işlemin nasıl yapıldığını görmeden önce koleksiyona eklenen ve insert işlemi için sırada bekleyen nesne örneklerini nasıl elde edebileceğimize bakalım. Burada **DataContext** sınıfına ait **GetChangeSet** metodundan yararlanılmaktadır. Bu metod ile elde edilen **ChangedSet** nesne örneği üzerinden **Inserts,Deletes** veya **Updates özellikleri(Properties)** kullanılarak eklenen, silinen veya güncellenen örnekler yakalanabilir.

***NOT : Inserts**, **Deletes** ve **Updates** özellikleri geriye **IList<T>** tipinden bir referans döndürmektedir. Bu referans bilgisinden yararlanılarak eklenen, silinen veya güncellenen tüm nesnelerin yakalanması mümkündür. **IList<T> arayüzü(Interface) .Net 2.0** versiyonundan beri mevcuttur. Ayrıca **IEnumerable<T>** arayüzünden türemiş olduğundan, elde edilen referans üzerinde **LINQ** sorgularıda yazılabilir. Bu sayede eklenen, silinen veya güncellenen nesnelerin veritabanına yazılmadan önce sorgulanmalarıda mümkün olmaktadır.*



örnek uygulamada bu kod parçası denenirse eklenen tool isimli yeni kategorininde elde edildiği görülecektir.



çok doğal olarak **ProductCategoryID** gibi alanlar veritabanı üzerindeki tabloda otomatik olarak üretildiklerinden ve kod içerisinde herhangibir değer atanmadığından henüz oluşturulmamıştır. Yapılan bu ekleme(Insert) işleminin veritabanına gönderilmesi için tek yapılması gereken ise **DataContext** tipinin **SubmitChanges** metodunu çalıştırmaktır.

adwContext.**SubmitChanges()**;

SubmitChanges metodu çalıştırıldığında **Insert**, **Update** veya **Delete** kuyruğunda bekleyen tüm nesneler için gerekli **SQL sorguları(Queries)** yürütülmektedir. Söz gelimi yukarıdaki örneğe göre **SQL Profiler** ile arkada çalışan kodlar izlenirse aşağıdaki sonuçların elde edildiği görülecektir.

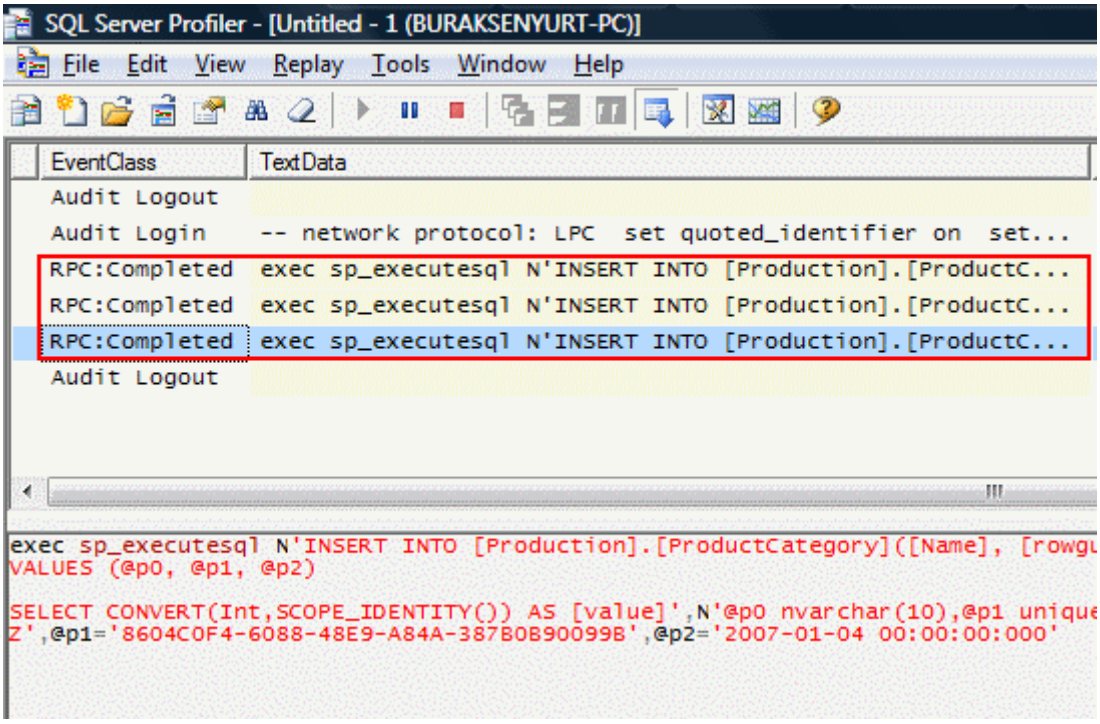
```
exec sp_executesql N'INSERT INTO [Production].[ProductCategory]([Name], [rowguid],
[ModifiedDate])
VALUES (@p0, @p1, @p2)

SELECT CONVERT(Int,SCOPE_IDENTITY()) AS [value]',N'@p0 nvarchar(5),@p1
uniqueidentifier,@p2 datetime',@p0=N'Tools',@p1='00000000-0000-0000-0000-
000000000000',@p2='2001-01-01 00:00:00:000'
```

Bu çalışan sorgu(Query) basit olarak üretilen varlık nesnesinin(Entity Object) veritabanına doğru yazılmasını sağlamaktadır. İstenirse toplu olarak veri ekleme işlemleride gerçekleştirilebilir. Bunun için tek yapılması gereken **InsertAllOnSubmit** metodunu kullanmaktadır. Aşağıdaki örnek kod parçasında toplu bir ekleme işleminin nasıl yapılabileceği gösterilmektedir.

```
adwContext.ProductCategories.InsertAllOnSubmit(
                                        new List<ProductCategory>()
                                           {
                                              new ProductCategory(){Name="Kategori X",
ModifiedDate=new DateTime(2006,12,3),rowguid=Guid.NewGuid()}
                                              ,new ProductCategory(){Name="Kategori
Y",ModifiedDate=new DateTime(2006,5,6),rowguid=Guid.NewGuid()}
                                              ,new ProductCategory(){Name="Kategori
Z",ModifiedDate=new DateTime(2007,1,4),rowguid=Guid.NewGuid()}
                                           }
                                        );

var eklenenler = adwContext.GetChangeSet().Inserts;

foreach (ProductCategory eklenen in eklenenler)
   Console.WriteLine(eklenen.Name);

adwContext.SubmitChanges();
```

Yine nesne başlatıcılarından(Object Initializers) yararlanılarak, **InsertAllOnSubmit** metodu içerisinde **ProductCategory** tipinden örnekler alabilecek generic List koleksiyonu oluşturulmuş ve 3 örnek ProductCategory eklenmiştir. İlave edilen satırları elde edebilmek için yine **GetChangeSet** metodu üzerinden **Inserts** özelliği kullanılmaktadır. **DataContext** tipi üzerinden**SubmitChanges** metodunun çalıştırılması ile birlikte, **varlık(Entity)** tiplerine eklenen 3 ProductCategory nesnesi içinde birer **Insert** sorgusu **SQL** sunucusu üzerinde

yürütülmektedir. Bu durumu daha iyi analiz etmek için **SQL Profiler** aracı kullanıldığında aşağıdakine benzer sonuçlar alındığı görülür.



Dikkat edileceği üzere her bir varlık nesnesi(Entity Object) için birer **Insert** sorgusu yürütülmektedir.

Gelelim silme işlemlerine. Silme(Delete) operasyonlarındada ekleme işlemlerine benzer şekilde **DeleteOnSubmit** ve **DeleteAllOnSubmit** gibi metodlar yer almaktadır. Herzamanki gibi varlık nesneleri üzerinden yapılan silme işlemleri **GetChangeSet** metodu üzerinden ulaşılan **Deletes** özelliği yardımıyla elde edilebilir. Aşağıdaki örnek kod parçasında tek bir satırın silme işleminin nasıl yapılabileceği gösterilmektedir.

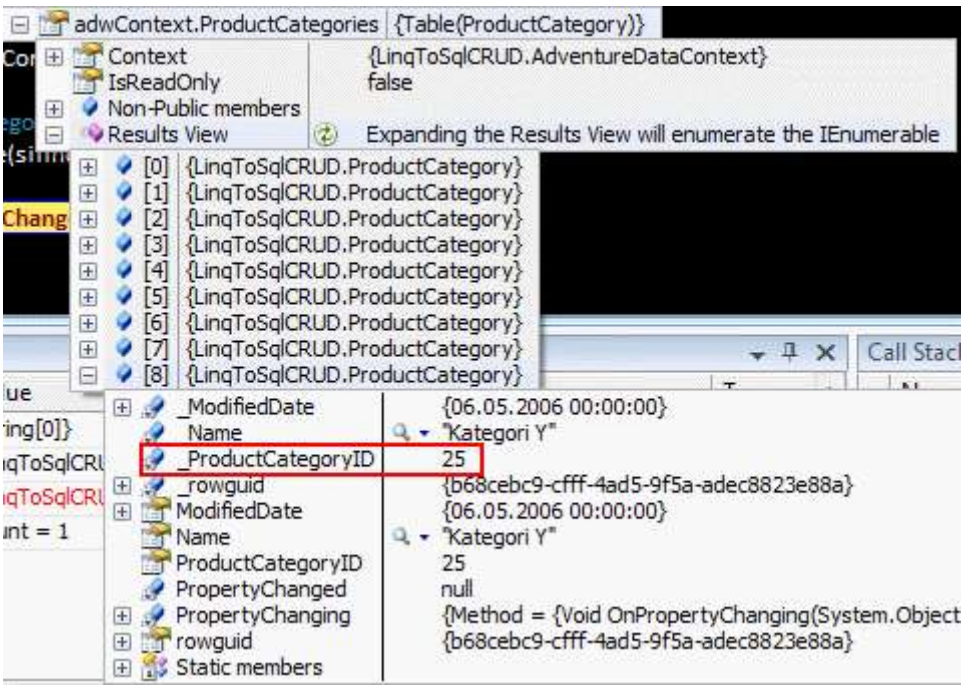
```
ProductCategory silinecekVeri = (from cat in adwContext.ProductCategories
                                 where cat.ProductCategoryID == 25
                                 select cat).Single<ProductCategory>();

adwContext.ProductCategories.DeleteOnSubmit(silinecekVeri);

var silinenler = adwContext.GetChangeSet().Deletes;

foreach(ProductCategory silinen in silinenler)
   Console.WriteLine(silinen.Name);

adwContext.SubmitChanges();
```

Burada dikkat edilmesi gerken bazı noktalar vardır. Silinmek istenilen satır veya satırların öncelikli olarak bulunması gerekir. Bu çok doğal olarak **Table<T>** tipi üzerinden

bir **LINQ** sorgusu ile mümkün olabilir. Dolayısıyla yukarıdaki kod parçasına göre, silinecekVeri isimli koleksiyon elde edilirken arka planda aşağıdaki sorgu cümlesi çalışacaktır.

exec sp_executesql N'SELECT TOP (1) [t0].[ProductCategoryID], [t0].[Name], [t0].[rowguid], [t0].[ModifiedDate]
FROM [Production].[ProductCategory] AS [t0]
WHERE [t0].[ProductCategoryID] = @p0',N'@p0 int',@p0=25

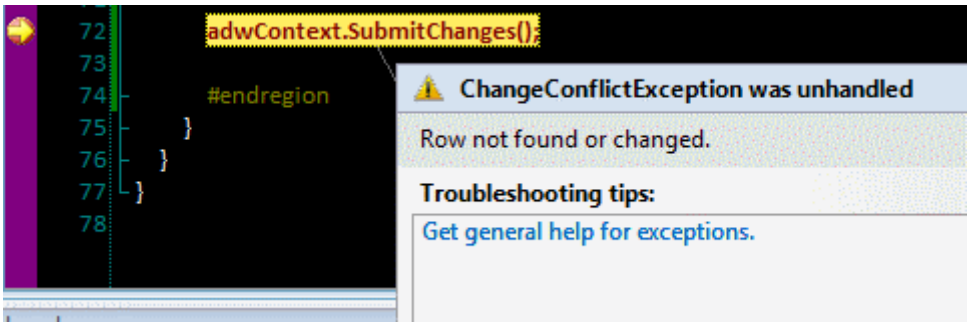
Bu adımdam sonra **DeleteOnSubmit** metoduna, silinmek istenen **varlık(Entity)** örneği parametre olarak verilmektedir. Bu işlem sadece silinecek olan veriler için kuyruğa bir ekleme yapmaktadır. öyleki bu metod çağırıldıktan
sonra **ProductCategories** koleksiyonuna bakılırsa, 25 numaralı **ProductCategory** tipinin halen daha mevcut olduğu görülebilir.



Silinmek istenen verinin programatik ortamda elde edilmesi için tek yapılması gereken **GetChangeSet** metodu üzerinden **Deletes** özelliğine ulaşmaktır. çalışma zamanı **SubmitChanges**metodunu yürüttüğünde ise silinmek istenen **entity** nesnesi ile ilgili olaraktan aşağıdaki sorgu cümlesi **SQL** tarafında işletilecektir.

exec sp_executesql N'DELETE FROM [Production].[ProductCategory]
WHERE
([ProductCategoryID] = @p0) AND ([Name] = @p1) AND ([rowguid] = @p2) AND ([ModifiedDate] = @p3)',N'@p0 int,@p1 nvarchar(10),@p2 uniqueidentifier,@p3 datetime',@p0=25,@p1=N'Kategori Y',@p2='B68CEBC9-CFFF-4AD5-9F5A-ADEC8823E88A',@p3='2006-06-05 00:00:00:000'

Sorgu cümlesindende dikkat edileceği üzere **Where** ifadesinden sonra tüm alanlar hesaba katılmaktadır. Bunun sebebi **LINQ to SQL** mimarisinin **Optimistic(İyimser) Concurrency** modelini kullanmasıdır. Bu modellde bilindiği üzere kontrol edilebilir tüm alanlar **Where** ifadesinden sonra hesaba katılmaktadır. Bir başka deyişle model, veriyi başkasının silip silmediğini, güncelleştirip güncelleştirmediğini araştırmaktadır. örnekte 25 numaralı kategoriye ait satırı silmeden önce başka birisi değiştirmişse eğer, çalışma zamanında **ChangeConflictException**istisnası alınır. Bu durumu daha iyi analiz etmek için 25 numaralı satırı silmek istediğimizi göz önüne alalım. **SubmitChanges** metodu çağırılmadan öncede veritabanından manuel olarak yada başka bir program üzerinden 25 numaralı satırın **Name** alanının değerini Kategori X' den Kategori XL' ye değiştirelim. Bunu kolay bir şekilde gerçekleştirmek için**SubmitChanges** satırına **breakpoint** koyup ilerlemeden önce tabloda değişiklik yapmak yeterli olacaktır. Böyle bir durumda çalışma zamanında(run time) aşağıdaki gibi bir durum olaşacaktır.



Nitekim biz silmek istediğimiz veriyi çektikten sonra **Name** alanının değeri Kategori X iken, **SubmitChanges**' den önce Kategori XL' ye değiştirilmiştir. Buda doğal olarak **Where** ifadesinin geçersiz olması anlamına gelmektedir. Ancak bu durum istenirse değiştirilebilir. öyleki, **varlık(Entity)** sınıflarında yer alan **özelliklerin(Properties) Column** isimli **niteliklerine(Attribute)** ait**ColumnChange** özelliğinin değeri **Never** yapıldığında söz konusu özelliklerin Where ifadelerinden sonrasına katılmadığı görülmektedir. Söz gelimi örnekte yer alan ProductCategory sınıfının **Name**, **rowguid** ve **ModifiedDate** özelliklerinde yer alan **Column** niteliğinde aşağıdaki gibi bir değişiklik yaptığımızı düşünelim.

```
[Column(Storage="_Name", DbType="NVarChar(50) NOT NULL",
CanBeNull=false,UpdateCheck=UpdateCheck.Never)]
public string Name
```

UpdateCheck değerine **Never** atanması sonucu söz konusu özelliğin değeri **WHERE** ifadesine parametre olarak dahil edilmeyecektir. Bu işlemin ardından örnek olarak başka bir satırı daha silmek istersek arka tarafta çalışan **Delete** sorgusunun aşağıdaki gibi oluşturulduğunu görebiliriz.

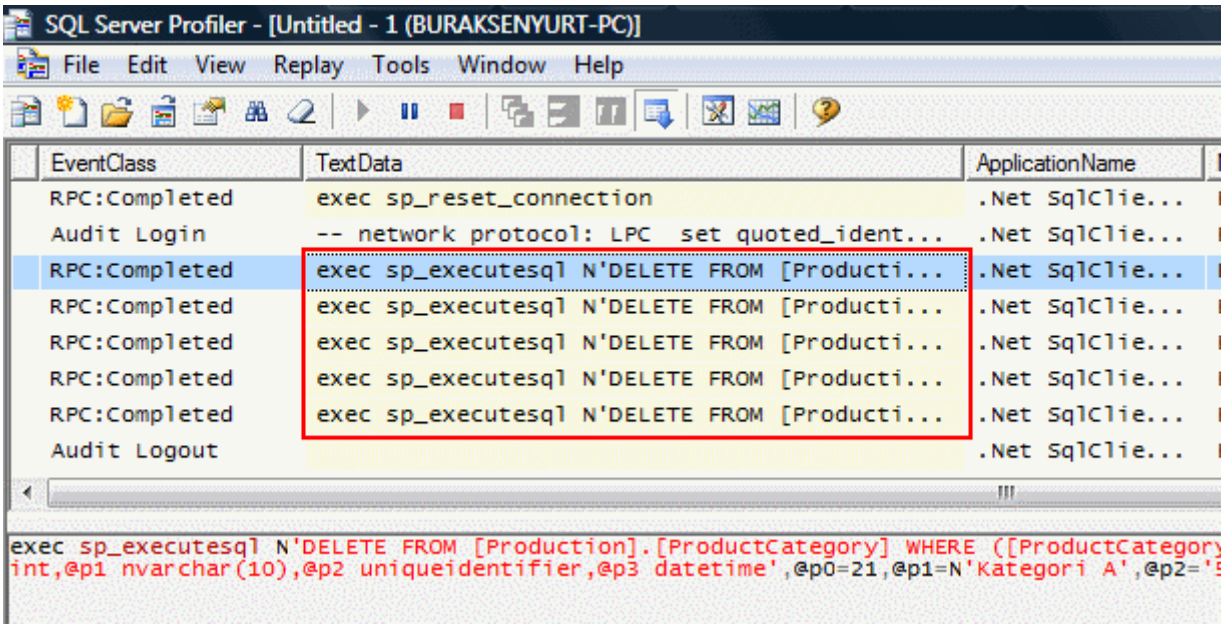
```
exec sp_executesql N'DELETE FROM [Production].[ProductCategory]
    WHERE [ProductCategoryID] = @p0',N'@p0 int',@p0=29
```

Görüldüğü gibi sadece **ProductCategoryID** alanı **WHERE** ifadesinden sonrasına katılmıştır. *(Kodun bundan sonraki kısımlarında Optimistic Concurrency modeline göre ilerleneceğinden UpdateCheck değişiklikleri geri alınmıştır.)*

İstenirse toplu olarak silme işlemleride gerçekleştirilebilir. örneğin Name özelliğinin içerisinde Kategori kelimesi geçen satırları silmek istediğimizi düşünelim. Bu amaçla aşağıdaki gibi bir kod parçası göz önüne alınabilir. öncelikli olarak Contains metodu ile Kategori kelimesi geçen ProductCategory nesnelerinin bir listesinin elde edilemsi gerekmektedir. LINQ sorgusu buna göre düzenlenmiştir.

```
var kategoriGecenler = from k in adwContext.ProductCategories
                       where k.Name.Contains("Kategori")
                         select k;

adwContext.ProductCategories.DeleteAllOnSubmit<ProductCategory>(kategoriGecenler);

adwContext.SubmitChanges();
```

Söz konusu kod içerisinde **SubmitChanges** metodu çalıştırıldığında **SQL** tarafında, silinmek istenen her satır için bir **Delete** sorgu ifadesinin yürütüldüğü görülecektir. örnekte bu kategoriye uyan 5 satır bulunmaktadır.

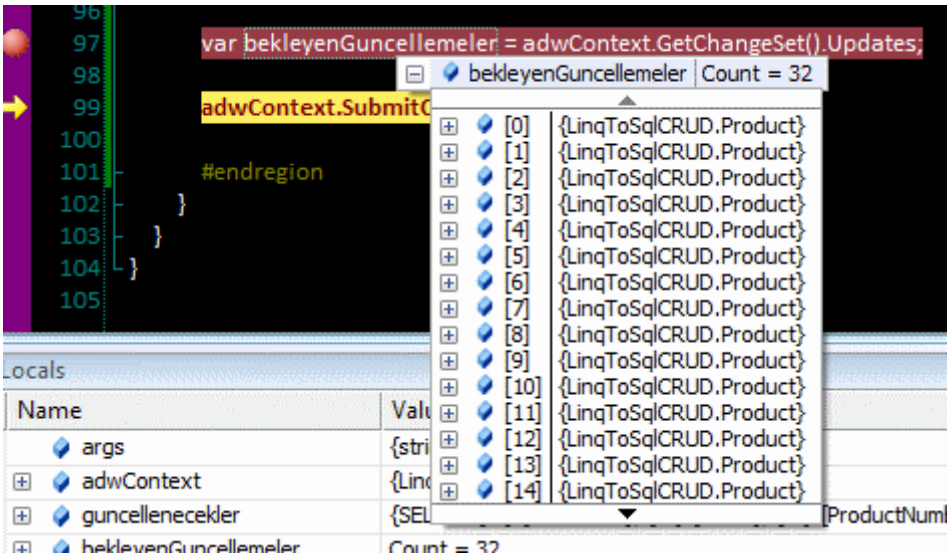


Gelelim **güncelleme(Update)** işlemlerine. Güncelleme süreçlerinde, **Insert** ve **Delete** işlemlerindeki gibi metodlar söz konusu değildir. Nitekim güncelleme işlemi aslında varlık nesnesinin herhangibir özelliğinin(özelliklerinin) değerinin değiştirilmesinden başka bir şey değildir. Dolayısıyla tek yapılması gereken değişiklikler tamamlandıktan sonra **SubmitChanges**metodunu çağırmaktır. Aşağıdaki örnek kod parçasında örnek olarak **Product** tablosuna ait bir **varlık sınıfı(Entity**

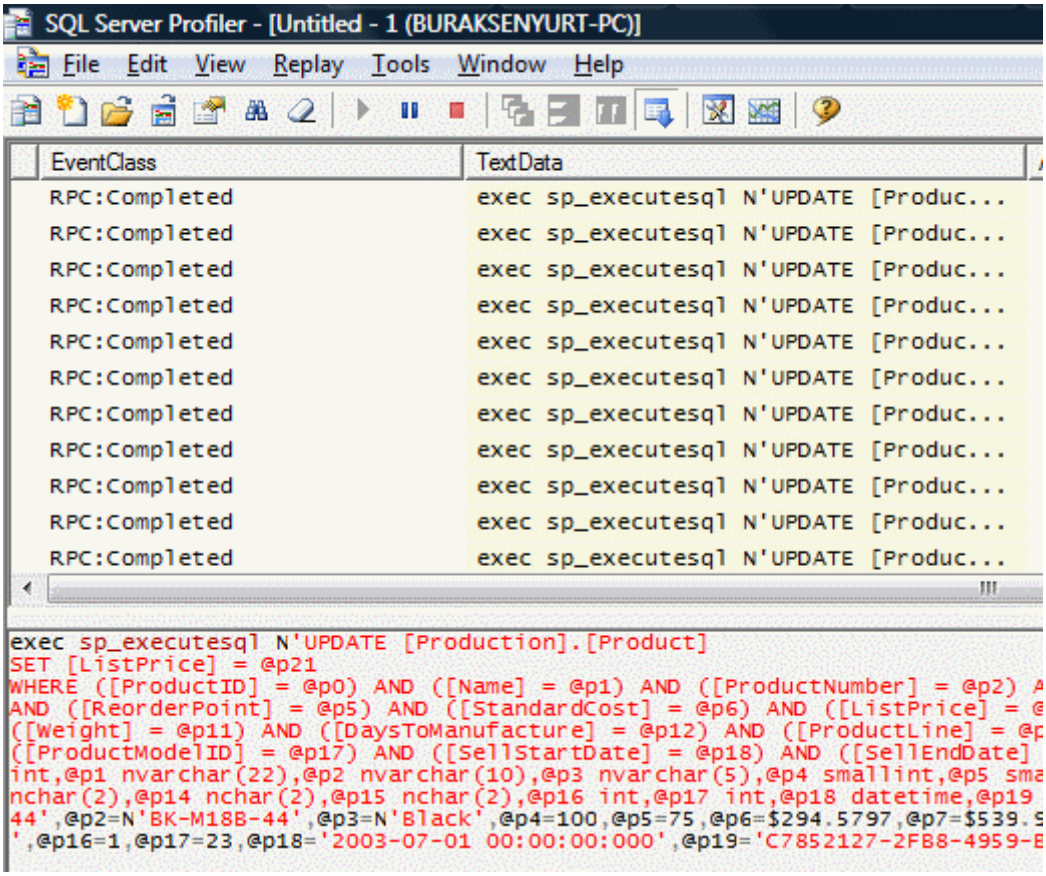
Class) kullanılmaktadır. Bu sınıfı oluşturmak için tek yapılması gereken tahmin edileceği üzere **Server Explorer**' dan **Product** tablosunu **Adventure.dbml** üzerine tasarım zamanında sürükleyip bırakmaktır.

```
var guncellenecekler = from p in adwContext.Products
                       where p.ProductSubcategoryID == 1
                          select p;

foreach (Product prd in guncellenecekler)
   prd.ListPrice += 10;

var bekleyenGuncellemeler = adwContext.GetChangeSet().Updates;

adwContext.SubmitChanges();
```

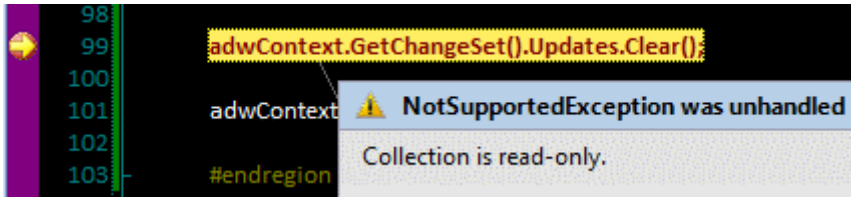
Bu kod parçasında **ProductSubCategoryId** değeri 1 olan nesneler elde edilmekte ve herbirinin **ListPrice** değeri 10 birim arttırılmaktadır. çok doğal olarak yapılan bu güncellemelerde **Updates**kuyruğuna atılacaktır. Söz gelimi örneğe göre varsayılan olarak kuyruğa 32 Product nesnesi atılmaktadır. Elbette koşula uyan kategorilerin her birinin ListPrice özelliklerinin değerleri değiştirildiğinde bu güncellemeler **Products** koleksiyonunada yansıyacaktır.



SubmitChanges metodunun işletilmesi ile birlikte güncelleme kuyruğunda bekleyen 32 adet **Product** nesne örneği için veritabanında **Update** sorguları çalıştırılacaktır. Aşağıdaki**SQL Profiler** ekran görüntüsünde bu sorguların bir kısmı yer almaktadır.



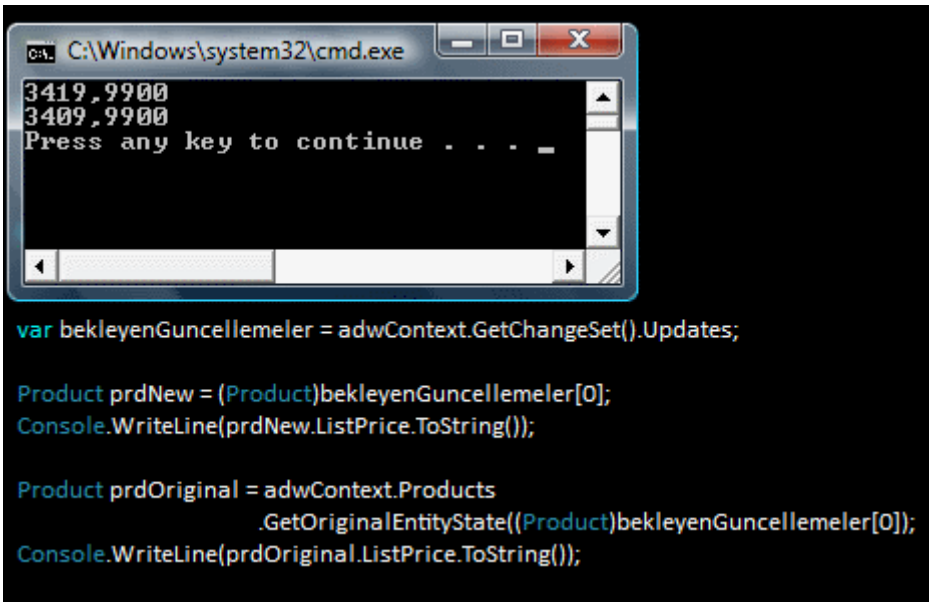
Buraya kadarki kısımda basit olarak **ekleme(Insert), silme(Delete) ve güncelleme(Update)** işlemlerini nasıl yapabileceğimizi görmeye çalıştık. önemli olan noktalardan biriside değişikliklerden vazgeçersek ne olacağıdır. Bu önemli bir sıkıntıdır. Nitekim **GetChangeSet** metodu üzerinden elde edilen **ChangeSet** tipinin sunduğu **Deletes**, **Inserts**, **Updates**özelliklerinin döndürdğü **IList<T>** koleksiyonları çalışma zamanında **yanlız okunabilir(read-only)** şekilde ele alınabilmektedir. Bu nedenle **Clear**, **Remove**, **RemoveAt** gibi metod çağrıları hatta **null** değer atanması sonrası çalışma zamanı istisnaları alınmaktadır. Söz gelimi son örnek koddaki güncelleştirmeleri onaylamak istemediğimizi düşünelim. Bu amaçla **Clear** metoduna başvurulması düşünülebilir. Ancak bu durumda çalışma zamanında aşağıdaki ekran görüntüsünde yer alan **NotSupportedException** istisnası alınacaktır.



Bu noktada **Table<T>** tipi üzerinden ulaşılabilen **GetOriginalEntityState** metodu alternatif bir yol olarak ele alınabilir. Nitekim bu metod, parametre olarak verilen **varlık nesnesinin(Entity Object)**değiştirilmeden önceki halini elde etmemizi sağlamaktadır. örneğin aşağıdaki kod parçasını ele alalım.

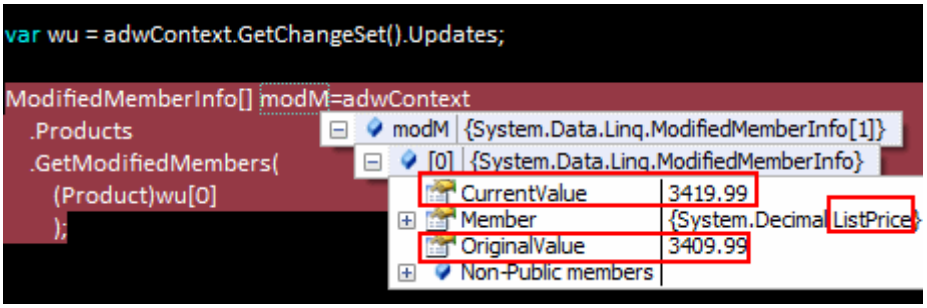
```
var bekleyenGuncellemeler = adwContext.GetChangeSet().Updates;

Product prdNew = (Product)bekleyenGuncellemeler[0];
Console.WriteLine(prdNew.ListPrice.ToString());

Product prdOriginal = adwContext.Products
                      .GetOriginalEntityState((Product)bekleyenGuncellemeler[0]);
Console.WriteLine(prdOriginal.ListPrice.ToString());
```

Burada görüldüğü gibi prdNew nesne örneğinin ListPrice değeri 3419.99 iken prdOriginal' in değeri 3409.00 dur. Aşağıdaki çalışma zamanı görüntüsünde bu durum net bir şekilde görülebilmektedir.



Ancak bu teknik yardımıyla güncellenmiş olan tüm satırların geri alınması oldukça zordur. Nitekim **Table<T>** tipinin **yapıcı metodunun(Constructor)** kullanılamadığı, bu yüzden **new** ile üretilemediği ortadadır. Ayrıca var olan **DataContext** nesnesinin Table<T> tipinden özellikleri **ReadOnly**' dir. Bir başka deyişle bu özelliklere doğrudan atama da yapılamamaktadır. Sonuç olarak **DataContext** tipinin yeniden örneklenmesi sorunu çözmek için yeterli olacaktır.

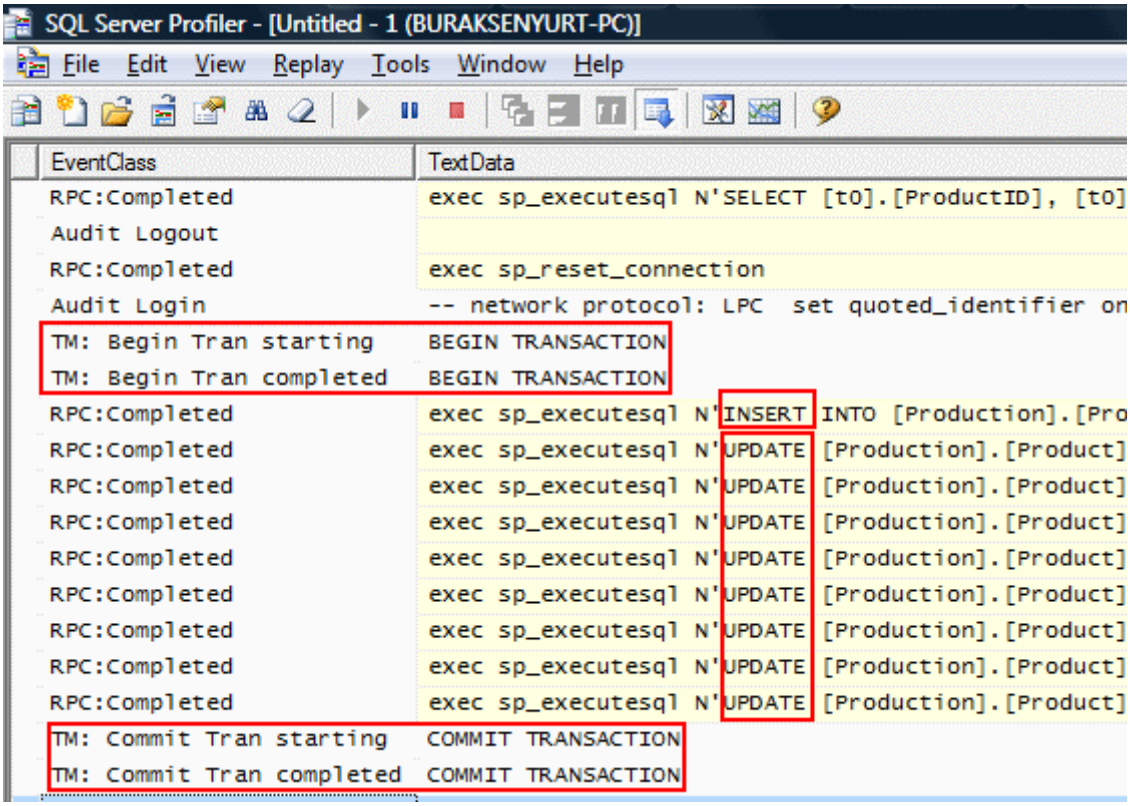
***NOT : Table<T>** generic tipi üzerinden kullanılan **GetModifiedMembers** metodu ile, parametre olarak verilen **varlık(entity)** nesne örneğinin değişikliğe uğrayan değerlerinin**orjinal(OriginalValue)** ve **anlık(CurrentValue)** hallerinin elde edilmesi sağlanabilmektedir. GetModifiedMembers metodu geriye **ModifiedMemberInfo** tipinden bir dizi döndürmektedir. Aşağıdaki şekilde örnek olarak güncellenen satırlardan ilki için orjinal ve güncel **ListPrice** değerlerinin elde edilişi gösterilmektedir.*



Normal şartlarda **SubmitChanges** metodunun çağırılmasından sonra güncelleme, ekleme ve silme işlemleri otomatik olarak transaction içerisinde çalıştırılırlar. Bir başka deyişle SubmitChanges metodu, veritabanı üzerinde yapılacak işlemlerin, biz söylemeden otomatik olarak bir **transaction** içerisinde olmasını sağlamaktadır. Söz gelimi aşağıdaki kod parçasını ele alalım. Bu kod parçasında güncelleme(Update) ve yeni ürün ekleme(Insert) işlemleri söz konusudur.

```
var guncellenecekler = from p in adwContext.Products
                       where p.Class == "M" && p.ProductSubcategoryID==1
                       select p;

foreach (Product prd in guncellenecekler)
   prd.ListPrice += 10;

Product newProduct = new Product()
   {
      Name = "Yeni Urun"
      , ProductSubcategoryID = 1, Color = "Red"
      , Class = "M", ListPrice = 100
      , ProductNumber = "PRD-1204", ReorderPoint = 10
      , StandardCost = 90, ProductModelID = 123
      , SafetyStockLevel = 45,SellStartDate=new DateTime(2007,1,1)
      , SellEndDate=new DateTime(2008,1,1), DiscontinuedDate=new DateTime(2006,6,6)
      , ModifiedDate=DateTime.Now
   };

adwContext.Products.InsertOnSubmit(newProduct);

adwContext.SubmitChanges();
```

İlk olarak Class değeri M ve ProductSubCategoryID değeri 1 olan Product tiplerine ait bir koleksiyon çekilmektedir. Daha sonra bu koleksiyon üzerinde dönülerek ListPrice değerleri sembolik olarak 10' ar birim arttırılmaktadır. Hemen arkasıdan örnek bir Product nesnesi oluşturulmakta ve InsertOnSubmit metodu ile eklenecekler listesine aktarılmaktadır. Uygulama çalıştırıldığında**SQL Profiler** aracılığıyla arkadaki işlemler incelenecek olursa aşağıdaki ekran görüntüsünde yer alan sonuçların elde edildiği görülür.

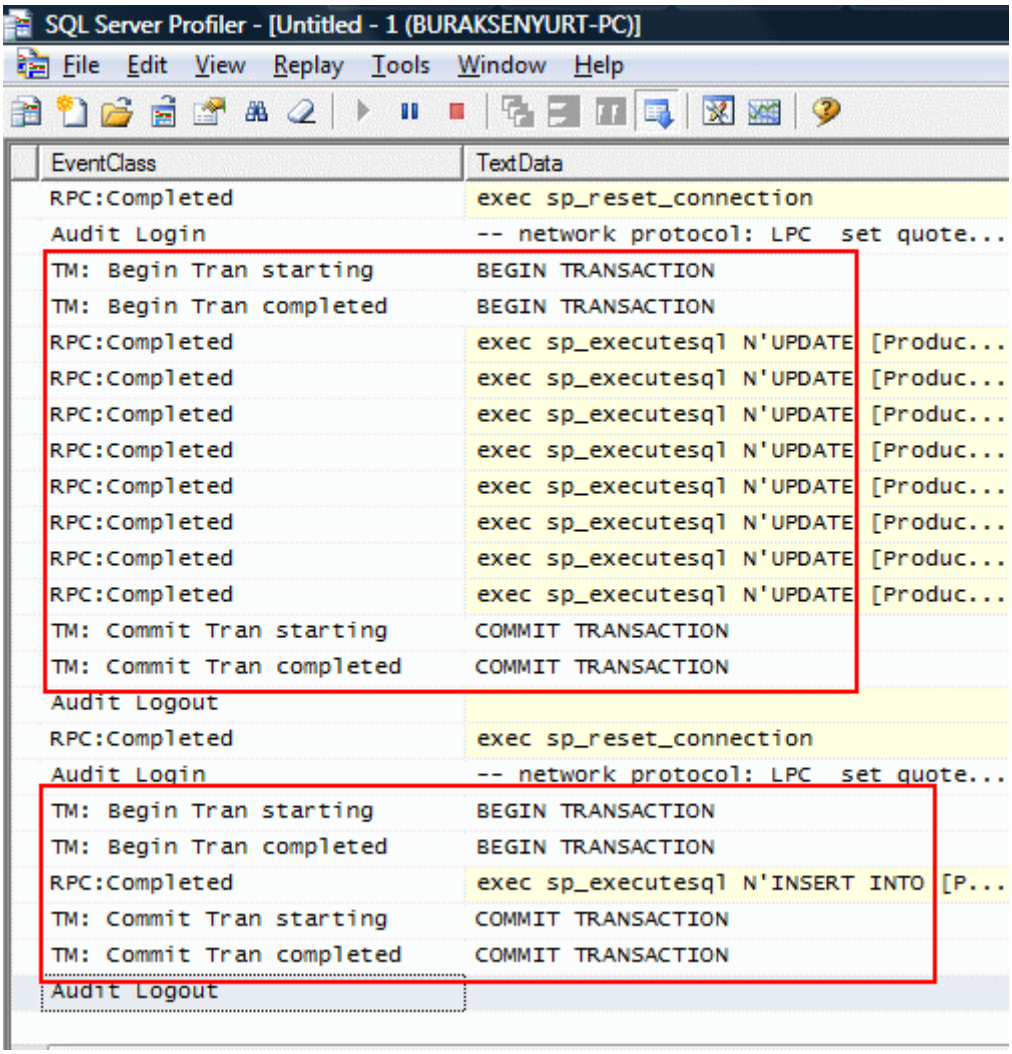


Dikkat edilecek olursa **Insert** ve **Update** ifadelerinin tamamı aynı **Transaction** içerisinde yürütülmektedir. Diğer taraftan aynı kodun aşağıdaki gibi değiştirildiğini düşünelim.

```
var guncellenecekler = from p in adwContext.Products
                       where p.Class == "M" && p.ProductSubcategoryID==1
                         select p;

foreach (Product prd in guncellenecekler)
   prd.ListPrice += 10;

adwContext.SubmitChanges();

Product newProduct = new Product()
   {
      Name = "Yeni Urun"
      , ProductSubcategoryID = 1, Color = "Red"
      , Class = "M", ListPrice = 100
      , ProductNumber = "PRD-1204", ReorderPoint = 10
      , StandardCost = 90, ProductModelID = 123
      , SafetyStockLevel = 45,SellStartDate=new DateTime(2007,1,1)
      , SellEndDate=new DateTime(2008,1,1), DiscontinuedDate=new DateTime(2006,6,6)
      , ModifiedDate=DateTime.Now
   };

adwContext.Products.InsertOnSubmit(newProduct);
```

adwContext.**SubmitChanges();**

Burada SubmitChanges metodu güncelleme ve ekleme işlemlerinden sonra birer kez ayrı ayrı çağırılmaktadır. Bu durumda **SQL Profiler** aşağıdaki sonuçları üretecektir.



Görüldüğü gibi **SubmitChanges** her çağırıldığında o ana kadar gerçekleştirilen ne kadar ekleme, güncelleme veya silme işlemi varsa ayrı bir **Transaction** kapsamı içerisinde çalışmaktadır. Elbetteki istenirse son kod parçasındaki tüm sorgu işlemlerin aynı **transaction kapsamı(Scope)** içerisine dahil edilmeside sağlanabilir. Bunun için **TransactionScope**nesnesinden yararlanılabilir. Aşağıdaki örnek bu durum basit olarak ele alınmaktadır.*(**TransactionScope** sınıfının kullanılabilmesi için **.Net Framework 2.0** ile birlikte gelen**System.Transactions.dll assembly'ının** projeye referans edilmesi gerekmektedir.)*

```
using (TransactionScope tScope = new TransactionScope())
{
   var guncellenecekler = from p in adwContext.Products
                          where p.Class == "M" && p.ProductSubcategoryID==1
```

```
                                select p;

   foreach (Product prd in guncellenecekler)
      prd.ListPrice += 10;

   adwContext.SubmitChanges();

   Product newProduct = new Product()
      {
         Name = "Yeni Urun"
         , ProductSubcategoryID = 1, Color = "Red"
         , Class = "M", ListPrice = 100
         , ProductNumber = "PRD-1204", ReorderPoint = 10
         , StandardCost = 90, ProductModelID = 123
         , SafetyStockLevel = 45, SellStartDate=new DateTime(2007,1,1)
         , SellEndDate=new DateTime(2008,1,1), DiscontinuedDate=new
DateTime(2006,6,6)
         , ModifiedDate=DateTime.Now
      };

   adwContext.Products.InsertOnSubmit(newProduct);

   adwContext.SubmitChanges();

   tScope.Complete();
}
```

Dikkat edileceği üzere tüm kod parçası TransactionScope nesne örneğine ait bir **using** bloğu içerisine alınmış ve en sonra Complete metodu çağırılmıştır. Bu durumda**SubmitChanges** çağrıları ayrı ayrı yapılmış olsada tüm işlemler aynı **transaction kapsamına(Transaction Scope**) dahil edilecektir. **TransactionScope** kullanılması çok doğal olarak programatik taraftan **Transaction** ile ilgili daha fazla yönetsel işlemin yapılabilmesi anlamınada gelmektedir. Söz gelimi izolasyon seviyeleri(Isolation Level) daha kontrollü bir şekilde ele alınabilir. Hatta dağıtık transaction(Distributed Transaction) geçişleri daha kolay programlanabilir.

TransactionScope kullanımı dışında yerel transaction kullanılarakta ilgili işlemlerin aynı **Transaction** içerisinde gerçekeştirilmesi sağlanabilir. Aşağıdaki örnek kod parçasında bu durum ele alınmaya çalışılmaktadır.

```
try
{
   var guncellenecekler = from p in adwContext.Products
                          where p.Class == "M" && p.ProductSubcategoryID == 1
                             select p;
```

```
   foreach (Product prd in guncellenecekler)
      prd.ListPrice += 10;

   // Transaction başlatılması için bağlantının açık olması gerekir.
   adwContext.Connection.Open();

   // Transaction başlatılır ve AdventureContext tipine bildirilir.
   adwContext.Transaction = adwContext.Connection.BeginTransaction();

   adwContext.SubmitChanges();

   Product newProduct = new Product()
      {
         Name = "Yeni Urun"
         , ProductSubcategoryID = 1, Color = "Red"
         , Class = "M", ListPrice = 100
         , ProductNumber = "PRD-1204", ReorderPoint = 10
         , StandardCost = 90, ProductModelID = 123
         , SafetyStockLevel = 45, SellStartDate=new DateTime(2007,1,1)
         , SellEndDate=new DateTime(2008,1,1), DiscontinuedDate=new
DateTime(2006,6,6)
         , ModifiedDate=DateTime.Now
      };

   adwContext.Products.InsertOnSubmit(newProduct);

   adwContext.SubmitChanges();

   // Herşey yolunda ise Transaction onaylanır
   adwContext.Transaction.Commit();
}
catch
{
   // Bir aksilik olduysa Transaction geri alınır
   adwContext.Transaction.Rollback();
}
finally
{
   if (adwContext.Connection.State == ConnectionState.Open)
      adwContext.Connection.Close();
}
```

Dikkat edileceği üzere **DataContext** tipinin **Transaction** özelliğine değer atanırken **Connection** üzerinden gidilmiş ve **BeginTransaction** metodu kullanılmıştır.

BeginTransaction metodunun parametresinden yararlanılarak, Transaction' ın **izolasyon seviyeside(Isolation Level)** değiştirilebilir. Burada **Transaction**' ın oluşturulabilmesi için bağlantının(Connection) açık olması gerekmektedir. Bu sebeptende **finally** bloğu içerisinde açık kalan bağlantının kontrollü bir şekilde kapatılması sağlanmaktadır. Transaction' a dahil olan işlemlerin onaylanması için**Commit** metodu kullanılırken bir sorun ile karşılaşılması halinde o ana kadar yapılan işlemlerin geri alınması içinde **Rollback** metoduna başvurulmaktadır. Sonuç olarak **SQL Profiler**aracı izlendiğinde **ekleme(insert)** ve **güncelleme(update)** işlemlerinin yine tek bir **Transaction** kapsamı içerisinde ele alındığı görülmektedir.

LINQ To SQL mimarisin şu ana kadar işlenen temel **CRUD** işlemlerinde ele alınması gereken başka hususlarda vardır. Söz gelimi eş zamanlı çalışan programların aynı veriler üzerinde işlemler yaptığı durumlarda oluşan **çakışmaların(Conflict)** ele alınması gibi. Bu ve benzeri konuları ilerleyen makalelerimizde ve görsel derslerimizde incelemeye çalışıyor olacağız. Böylece geldik bir makalemizin daha sonuna. Bu makalemizde çok basit ve temel seviyede **ekleme(Insert), Silme(Delete) ve Güncelleme(Update)** işlemlerini nasıl yapabileceğimizi incelemeye çalıştık. Ayrıca bu işlemleri yaparken **Transaction**' ların nasıl kullanılabildiğinide gördük. Varsayılan olarak bilinçsiz şekilde başlatılan Transaction' ları TransactionScope ile veya Local Transaction teknikleri nasıl kontrol edebileceğimiz gördük. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

LinqToSqlCRUD.rar (56,40 kb)

Nasıl Yapılır? Adım adım özel Http Handler Geliştirmek (2007-12-10T02:05:00)

asp.net 2.0,

Uzun zaman önce **Asp.Net 2.0** ile ilişkili makalelerimizden birisinde **HttpHandler** ve **HttpModule** kavramlarından bahsetmeye çalışmıştık. Bu makalemizde kendi Handler sınıfımızı geliştirmek isteyebileceğimiz örnek bir senaryo üzerinde daha durmaya çalışacağız. Bu sayede HttpHandler sınıfları yazarak neler yapılabileceğinide daha net bir şekilde görmüş olacağız. Konuyu daha net kavrayabilmek adına örnek senaryomuz üzerinden adım adım ilerleyeceğiz.

Bilindiği üzere web sunucusuna istemci tarafından gelen **talepler(Requests)** bazı **program ara yüzleri(API)** tarafından karşılanır ve uygun ortamlara işletilmek üzere iletilirler. özellikle**Asp.Net** ile geliştirilen web uygulamalarında, talep edilen dosya tipine göre devreye giren HttpHandler sınıfları bulunmaktadır. Söz gelimi **aspx** uzantılı dosyalar **PageHandlerFactory** isimli sınıf tarafından ele alınırlar. **PageHandlerFactory** ve benzer işlevselliklere sahip handler tipleri **IHttpHandler arayüzünü(interface)** uygularlar. Dolayısıyla geliştiriciler kendi Handler tiplerini**IHttpHandler** arayüzünü kullanarak yazabilirler.

Gelelim örnek senaryomuza. Sunucu üzerinde barındırılan **XML(eXtensible Markup Language)** tabanlı basit bir metin dosyasını ele alacak özel bir **Handler** tipi geliştiriyor olacağız. XML tabanlı dosya içerisinde yer alan bilgilerden yararlanılarak ekrana herhangibir sorguya ait raporlama sonuçları aktarılacak. Dosyanın uzantısının örnek olarak **rapx** olduğunu düşünebiliriz. Peki bu raporlama işleminde önemli bir rol oynacak olan XML içeriğinde neler olması gerekmektedir? Bunların tespiti **XML** dosyasının mantıksal ağaç yapısının oluşturulmasınıda kolaylaştıracaktır. Söz konusu ihtiyaçları aşağıdaki maddeler halinde sıralanabilirler.

- Hazırlanan rapor için bir **başlık(Title)** bilgisi tutulabilir. Hatta başlığın **arka plan rengi(Background Color)** verilerek zengin görünmesi sağlanabilir.
- Rapor dosyasının kaç adet **sorgu cümlesi(Query)** için destek vereceğine karar vermek gerekmektedir. örneğin basit olması açısından başlangıç itibariyle sadece tek bir sorgu sonucunun ele alınmasında fayda vardır.
- Sorgu cümlesi, **saklı yordam(stored procedure)** veya text tabanlı ifadeleri işaret edebilir. Hatta bir veritabanı **görünümünün(View)** desteklenmesi bile sağlanabilir.
- Sorgu cümlesinde parametre kullanımına destek verilebilir. Bu parametrelerinin **QueryString** yardımıyla Url üzerinden veya XML içeriğinden alınacağına dair tanımlamalar yapılabilir.
- Sorgu cümlesinin çalışacağı **sunucu(Server)** ve **veritabanı(Database)** adı belirtilmelidir. Hatta SSPI ile bağlantıya destek verilmeside göz önüne alınmalıdır.
- Hazırlanan raporların **XML** içeriğinde belirtilen mail adreslerine gönderilmesi de sağlanabilir.
- Raporun kimler tarafında görülebileceğine dair tanımlamalar yapılabilir. Bu tanımlamalar **kullanıcı(User)** ve hatta **rol(Role)** bazında gerçekleştirilebilir.

Bu ihtiyaçlar dahada arttırılabilir. Bir anlamda geliştiricinin hayal gücü burada önem kazanmaktadır. Yukarıdaki maddelerde sözü geçen ihtiyaçların tamamı **XML**' in temel kavramları ile karşılanabilir. Bir başka
deyişle **eleman(element)** ve **nitelikler(attributes)** sayesinde yukarıdaki istekler standartlara uygun olacak şekilde tasarlanabilir.

Not : *Söz konusu XML içeriğinin **çalışma zamanında(run time)** uygun bir standartta olduğunun garanti altına alınması için **şema (Xml Schema)** kullanılması yararlı olacaktır. Bu şema ile Xml içeriğinin çarpıştırılması Handler tipi içerisinde yapılabilir. Şemaya uymayan XML içerikleri için **Handler**, istekleri özel olarak tasarlanmış bir hata sayfasına doğru yönlendirebilir.*

Yukarıdaki maddeler ışığında aşağıdaki gibi örnek bir **XML** içeriği göz önüne alınabilir.

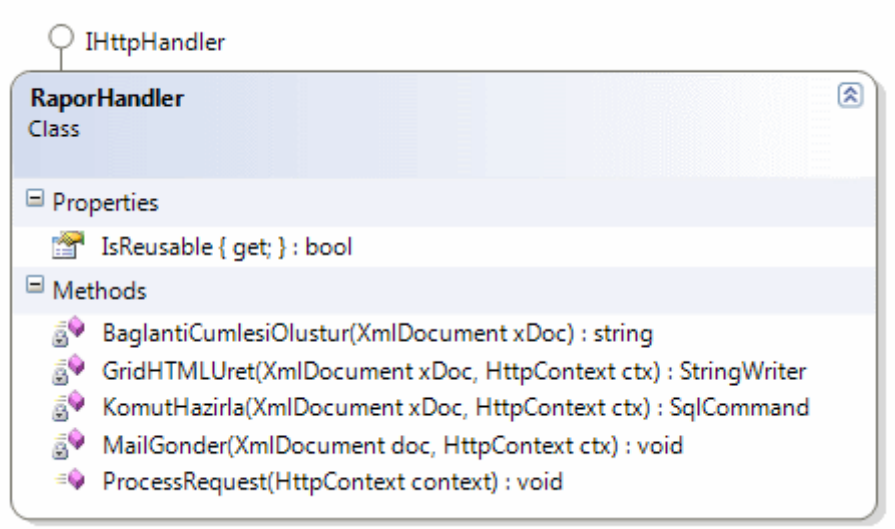
```
<?xml version="1.0" encoding="utf-8" ?>
<Raporlar>
   <Rapor Id="">
      <Baslik ArkaPlan=""></Baslik>
      <Baglanti SSPI="">
```

```
        <Sunucu></Sunucu>
        <Veritabani></Veritabani>
        <KullaniciAdi></KullaniciAdi>
        <Sifre></Sifre>
      </Baglanti>
      <Sorgu Sp="">
        <Cumle></Cumle>
        <Parametreler Nereden="">
          <Parametre Ad="" Deger=""/>
        </Parametreler>
      </Sorgu>
      <MailListesi MailGonder="">
        <Mail></Mail>
        <Mail></Mail>
      </MailListesi>
   </Rapor>
</Raporlar>
```

Raporlar **ana boğumu(Root Node)** birden fazla **Rapor** elementi içerebilir. Biz örneğin daha kolay ele alınabilmesi amacıyla tek bir Rapor elementi kullanıyor olacağız. Raporun alınacağı**sunucu(Server), veritabanı adı(Database Name), kullanıcı adı(User name)** ve **şifre(Password)** bilgileri ise Baglanti elementi altında tutulmaktadır. Hatta **SSPI** kullanımınada destek verilmesi amacıyla Baglanti elementi içerisinde bir **nitelik(attribute)** tanımlanmaktadır. Sorguların **saklı yordam(Stored Procedure)** olup olmadığı Sp niteliği ile belirtilebilir. Bunun dışında sorgu içerisinde kullanılan parametreler var ise bunların nereden alınacağı Nereden isimli **nitelik(attribute)** ile belirlenmektedir. öyleki raporun parametreleri, rapx dosyasına tarayıcı üzerinden yapılan çağrılarda **QueryString** yardımıyla gelebilir. Yada Parametre elementleri içerisindeki Deger niteliklerinde doğrudan tanımlanabilir. üretilen raporların mail yolu ile kimlere bildirileceğine dair MailListesi elementi ve Mail alt elementleri kullanılabilir.

Burada ortaya çıkan önemli bir ihtiyaç vardır. **XML** bilgilerinin tutulduğu rapx uzantılı dosyaların işlenmesi, bir **HTML** çıktısının üretilmesi ve talepte bulunan istemcilere gönderilmesi sağlanmalıdır. Bunu sadece özel bir Handler tipi karşılayabilir.

Dilerseniz vakit kaybetmeden **Handler** tipini yazarak işe başlayalım. Geliştirilecek olan Handler sınıfı ilk etapta tek bir web uygulamasında kullanılacaktır. Sonrasında ise bu handler tipinin sunucu üzerindeki her Asp.Net web uygulaması için geçerli olması sağlanacaktır. Bu nedenle **Handler** tipinin bir **sınıf kütüphanesi(Class Library)** içerisinde olmasında yarar vardır. Handler sınıfının **IHttpHandler arayüzünden(Interface)** türetilmesi içinde sınıf kütüphanesine **System.Web.dll assembly**' ının referans edilmesi şarttır. Söz konusu **Handler**sınıfının diagram görüntüsü ve içeriği aşağıdaki gibidir.




```
using System;
using System.Web;
using System.Data;
using System.Web.Hosting;
using System.Data.SqlClient;
using System.Xml;
using System.Web.UI.WebControls;
using System.Web.UI;
using System.IO;

namespace ReportHandlerLibrary
{
   public class RaporHandler:IHttpHandler
   {
      #region IHttpHandler Members

      public bool IsReusable
      {
         get { return false; }
      }

      // Gelen talep sonrası üretilecek HTML çıktısını bu metod içerisinden veriyor olacağız
      public void ProcessRequest(HttpContext context)
      {
         // Talep edilen sayfa Request.Path ile yakalanır.
         // VirtualPathProvider metodu ile sanal adresin karşılığı olan fiziki yoldaki dosya
açılarak Stream halinde elde edilir.
         // Stream'den yararlanılarak XML içeriği XmlDocument nesnesi içerisine alınır.
         XmlDocument xDoc = new XmlDocument();
         xDoc.Load(VirtualPathProvider.OpenFile(context.Request.Path));
         if (context.Request.QueryString["MailGonder"] == null)
```

```
        {
            // Baslik elementinden raporun başlığı bilgisi alınır
            string baslik = xDoc.SelectSingleNode("Raporlar/Rapor/Baslik").InnerText;
            string baslikArkaPlanRengi =
xDoc.SelectSingleNode("Raporlar/Rapor/Baslik").Attributes["ArkaPlan"].InnerText;

            // Ekran tasarımı oluşturulmaya başlanır
            // üretilen çıktı bir HTML sayfası olacağından HTML elementlerinin kullanılması
gerekmektedir.
            context.Response.Write("<HTML><HEAD><TITLE>" + baslik +
"</TITLE></HEAD>");
            context.Response.Write("<BODY>");
            // Table elementi oluşturulur.
            context.Response.Write("<TABLE border='1' width='100%' cellspacing='0'
cellpadding='5'>");
            // Tablo 3 satırdan oluşmaktadır. İlk satırın arka plan rengi ve içerisinde yer
alacak metin bilgisi Baslik elementi ve ArkaPlan niteliklerinden alınır
            context.Response.Write("<TD style='background-color:" + baslikArkaPlanRengi
+ "'>");
            context.Response.Write("<H3>" + baslik + "</H3>");
            context.Response.Write("</TD></TR>");
            // İkinci satır içerisinde rapor sonucu üretilen grid içeriği olmalıdır.
            context.Response.Write("<TR><TD>");

            // Bu hücreye veri ile doldurulan Grid içeriğinin HTML çıktısı yazdırılır
            context.Response.Write(GridHTMLUret(xDoc,context).ToString());

            context.Response.Write("</TD></TR>");
            context.Response.Write("<TR><TD>");

            // Mail gönderme aksiyonu için basit bir hyperLink elementi eklenir
            context.Response.Write("<a href='"+context.Request.Path + "?MailGonder=1'>"
+ "Raporu Mail Olarak Gönder" + "</a>");

            context.Response.Write("</TD></TR>");
            context.Response.Write("</BODY>");
            context.Response.Write("</HTML>");
        }
        else
        {
            bool mailGondersinmi = false;
            Boolean.TryParse(xDoc.SelectSingleNode("Raporlar/Rapor/MailListesi").Attrib
utes["MailGonder"].Value, out mailGondersinmi);
            if (mailGondersinmi)
                MailGonder(xDoc,context);
```

```
        }
    }

    #endregion

    // GridView kontrolünün içeriği doldurulduktan sonra HTML içeriği elde edilir ve bu
içeriği taşıyan StringWriter geriye döndürülür.
    private StringWriter GridHTMLUret(XmlDocument xDoc,HttpContext ctx)
    {
        // SqlDataAdapter nesne oluşturulur
        SqlDataAdapter adapter = new SqlDataAdapter(KomutHazirla(xDoc,ctx));
        DataTable table = new DataTable();
        // DataTable doldurulur
        adapter.Fill(table);

        // GridView kontrolü üretilir ve veriye bağlanır
        GridView grd = new GridView();
        grd.DataSource = table;
        grd.DataBind();

        //GridView kontrolünün HTML çıktısı elde edilir
        StringWriter strWriter = new StringWriter();
        HtmlTextWriter writer = new HtmlTextWriter(strWriter);
        grd.RenderControl(writer);
        return strWriter;
    }

    // Raporun üretilmesi için gerekli SqlCommand hazırlanıyor.
    private SqlCommand KomutHazirla(XmlDocument xDoc,HttpContext ctx)
    {
        bool sp = false;
        // Sorgu cümlesinin Stored Procedure olup olmadığı belirlenir.
        Boolean.TryParse(xDoc.SelectSingleNode("Raporlar/Rapor/Sorgu").Attributes["Sp
"].Value, out sp);

        // SqlCommand tipi hazırlanır
        SqlCommand cmd = new SqlCommand();
        // Sorgu cümlesi alınır
        cmd.CommandText =
xDoc.SelectSingleNode("Raporlar/Rapor/Sorgu/Cumle").InnerText.Trim();
        // Bağlantı cümlesi BaglantiCumlesiOlustur metodundan elde edilir ve Command
için gerekli SqlConnection hazırlanır.
        cmd.Connection = new SqlConnection(BaglantiCumlesiOlustur(xDoc));
        // Eğer cümle Stored Procedure adını işaret ediyorsa CommandType için
StoredProcedure enum sabiti değeri verilir
```

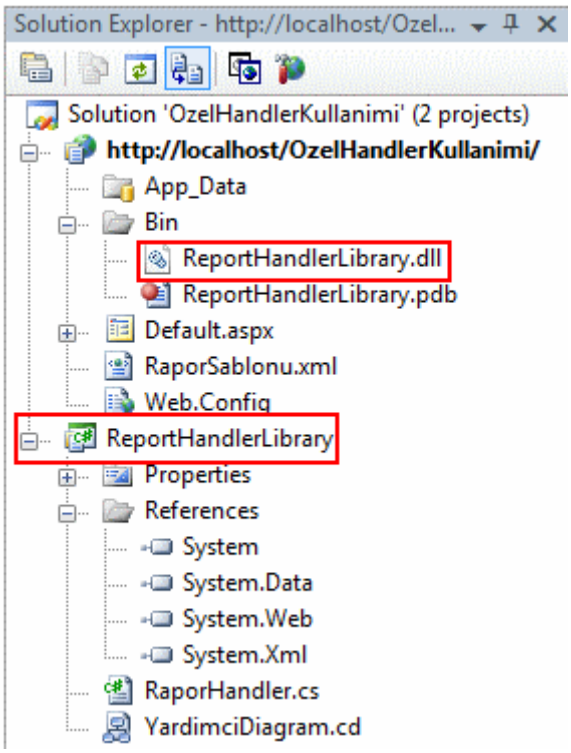
```
        if (sp)
            cmd.CommandType = CommandType.StoredProcedure;

        // Eğer girilmiş parametreler varsa bu parametreler Command nesnesine
AddWithValue metodu ile teker teker eklenir.
        if
(xDoc.SelectSingleNode("Raporlar/Rapor/Sorgu/Parametreler").ChildNodes.Count > 0)
        {
            string parametreNereden =
xDoc.SelectSingleNode("Raporlar/Rapor/Sorgu/Parametreler").Attributes["Nereden"].Valu
e;
            XmlNodeList parametreler =
xDoc.SelectSingleNode("Raporlar/Rapor/Sorgu/Parametreler").ChildNodes;
            foreach (XmlNode parametre in parametreler)
            {
                if(parametreNereden=="Xml")
                    cmd.Parameters.AddWithValue(parametre.Attributes["Ad"].Value,
parametre.Attributes["Deger"].Value);
                else if(parametreNereden=="QueryString")
                    cmd.Parameters.AddWithValue(parametre.Attributes["Ad"].Value,
ctx.Request.QueryString[parametre.Attributes["Ad"].Value.Substring(1,
parametre.Attributes["Ad"].Value.Length - 1)]);
            }
        }
        // Oluşturulan Command nesnesi geri döndürülür.
        return cmd;
    }

    // Sorguların çalıştırılması için gerekli Bağlantı cümlesini oluşturan metod
    private string BaglantiCumlesiOlustur(XmlDocument xDoc)
    {
        bool sspi = false;

        // Sql bağlantısı için gerekli bağlantı cümlesi(Connection String)
SqlConnectionStringBuilder sınıfı yardımıyla oluşturulur.
        SqlConnectionStringBuilder conStrBuilder = new SqlConnectionStringBuilder();

        // Sunucu ve veritabanı bilgileri XPath ifadeleri ile alınır.
        conStrBuilder.DataSource =
xDoc.SelectSingleNode("Raporlar/Rapor/Baglanti/Sunucu").InnerText;
        conStrBuilder.InitialCatalog =
xDoc.SelectSingleNode("Raporlar/Rapor/Baglanti/Veritabani").InnerText;
        Boolean.TryParse(xDoc.SelectSingleNode("Raporlar/Rapor/Baglanti").Attributes["
Sspi"].Value, out sspi);
        if (!sspi) // Eğer integrated security ile bağlanılmıyorsa kullanıcı adı(UserID) ve
```

```
şifre(Password) bilgileri alınır.
            {
               conStrBuilder.UserID =
xDoc.SelectSingleNode("Raporlar/Rapor/Baglanti/KullaniciAdi").InnerText;
               conStrBuilder.Password =
xDoc.SelectSingleNode("Raporlar/Rapor/Baglanti/Sifre").InnerText;
            }
            else
               conStrBuilder.IntegratedSecurity = true;

            // Oluşturulan bağlantı cümlesi(Connection String) geri döndürülür.
            return conStrBuilder.ConnectionString;
         }

         // Mail gönderme seçeneği aktif ise postaların gönderilme işlemini gerçekleştirecek
olan metod.
         private void MailGonder(XmlDocument doc,HttpContext ctx)
         {
            // Mail listesi MailListesi elementinin alt elementlerinden çekilir.
            XmlNodeList mailler =
doc.SelectSingleNode("Raporlar/Rapor/MailListesi").ChildNodes;
            foreach (XmlNode mail in mailler)
            {
               //TODO: Bu noktada raporun mail olarak gönderilmesine ait işlemler
yapılacaktır.
               ctx.Response.Write(mail.InnerText + " adresine rapor mail olarak
gönderildi<br/>");
            }
         }
      }
   }
```

Geliştirilen sınıf içerisinde parçaları daha kolay ele alabilmek adına yardımcı metodlar yer almaktadır. Dikkat edileceği üzere rapx içeriğinin **XML** formatında tasarlanması, işlemleri son derece kolaylaştırmaktadır. Nitekim içeriğin **XmlDocument** sınıfına ait bir nesne örneği ile belleğe alınması, içerisindeki elementlerin veya niteliklerin **XPath** ifadeleri ile yakalanması son derece kolaydır. üstelik XML, platform bağımsızlık sunduğundan bu dökümanın başka bir ortama gönderilerek ele alınmasının sağlanması daha kolaydır. Bu sebepten dolayı günümüzün popüler kavramlarından olan **Reporting Services** veya **LINQ To SQL** içerisinde yer alan **Database Markup Language(dbml)** gibi yapılar XML üzerine oturmaktadır. Sınıf içerisinde dikkat edilmesi gereken noktalardan bir diğeri ise, **HttpContext** tipinin ektin şekilde kullanımıdır. HttpContext üzerinden ele alınan üyeler ile, üretilecek **HTML** içeriğinin tasarlanması veya talep ile gelen QueryString' lerin yakalanması söz konusudur. Hatta talep edilen sayfanın **sanal yolu(Virtual Path)** elde edilip **XML** içeriğinin **VirtualPathProvider** sınıfının**OpenFile** metodu ile kolay bir

şekilde **Stream**' e dönüştürülmeside sağlanmaktadır. İşlemleri kolaylaştıran noktalardan biriside web kontrollerinin **RenderControl** metodudur. Bu metod ile kompleks bir **GridView** kontrolünün veri dolu içeriğinin **HTML** çıktısını almak son derece kolaylaşmaktadır.

Şimdi geliştirilen **Handler** tipini örnek bir web uygulamasında test edebiliriz. İlk olarak tek bir web uygulamasına özel olacak şekilde Handler tipinin kullanılmasını ele alacağız. Böyle bir durumda öncelikli olarak web uygulamasının Handler tipini içeren **sınıf kütüphanesini(Class Library)** referans etmesi gerekmektedir. örnek olarak OzelHandlerKullanimi isimli web uygulaması aşağıdaki ekran görüntüsünden de anlaşılacağı üzere ReportHandlerLibrary isimli **assembly**' ı referans etmektedir.

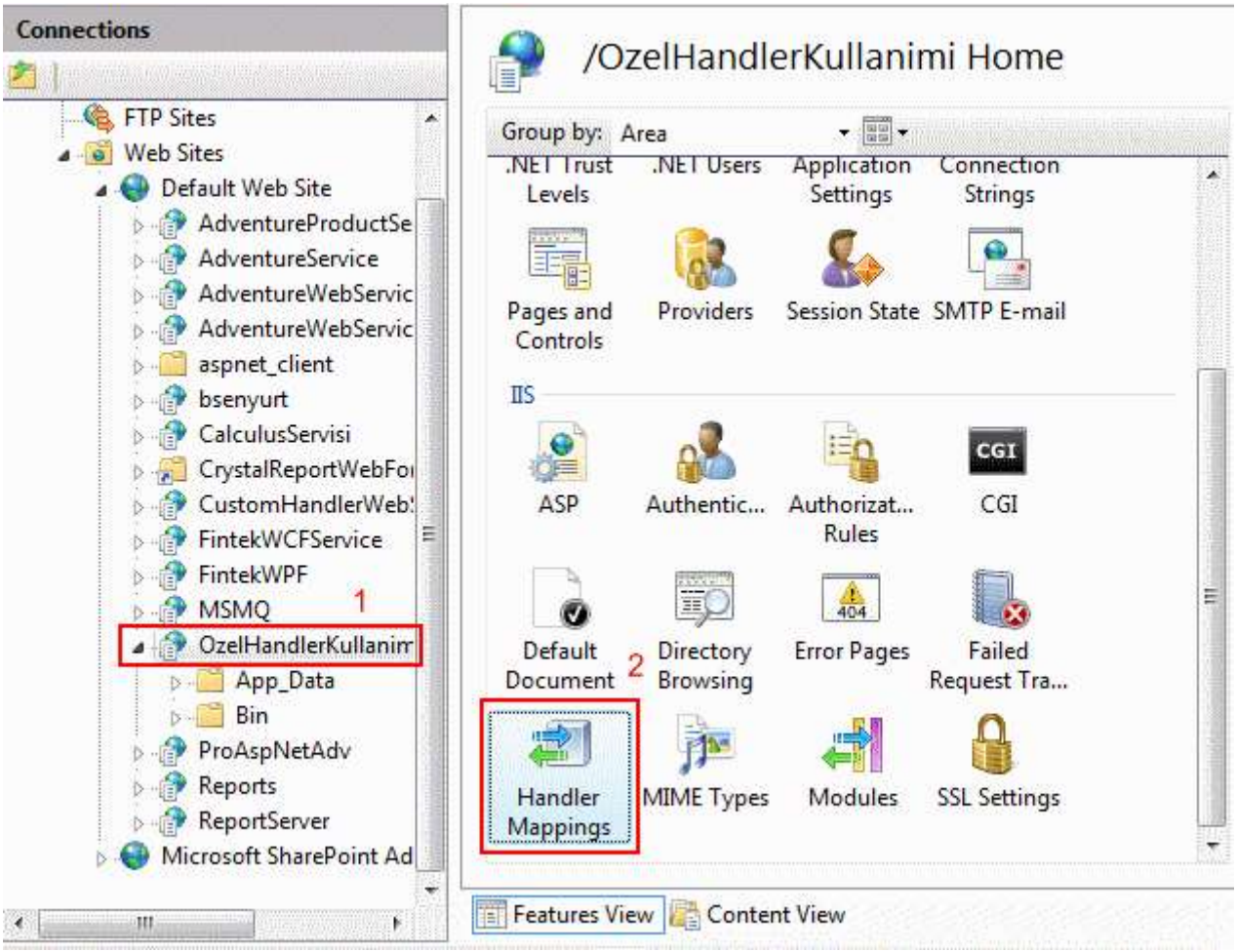


Bunun dışında web uygulamasına ait **web.config** dosyası içerisinde aşağıda görüldüğü gibi gerekli bildirimler yapılmalıdır.

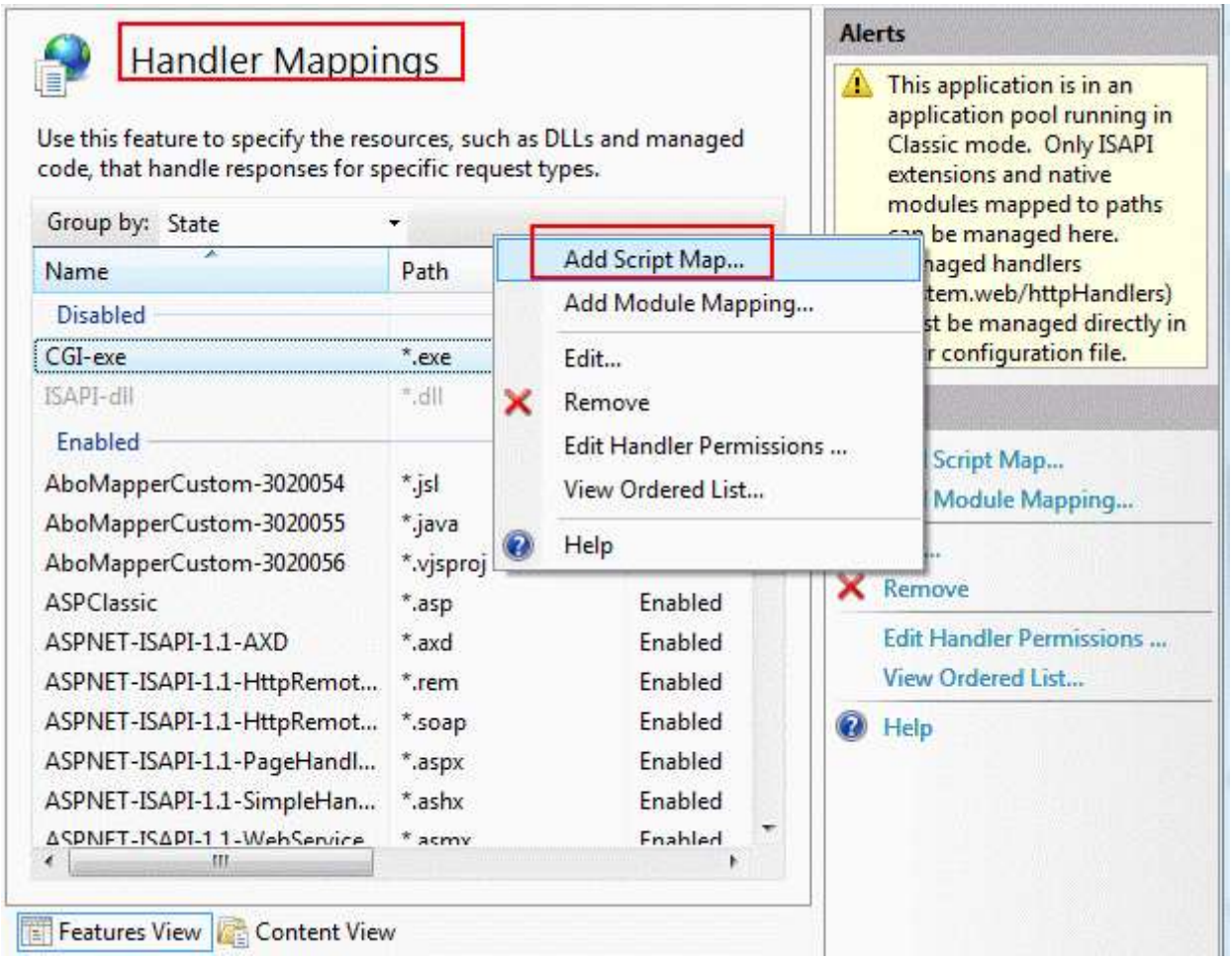
```
<?xml version="1.0"?>
<configuration>
   <appSettings/>
   <connectionStrings/>
   <system.web>
      <httpHandlers>
         <add path="*.rapx"
type="ReportHandlerLibrary.RaporHandler,ReportHandlerLibrary" verb="*"
validate="true"/>
      </httpHandlers>
      <compilation debug="true"/>
```

```
    <authentication mode="Windows"/>
  </system.web>
</configuration>
```

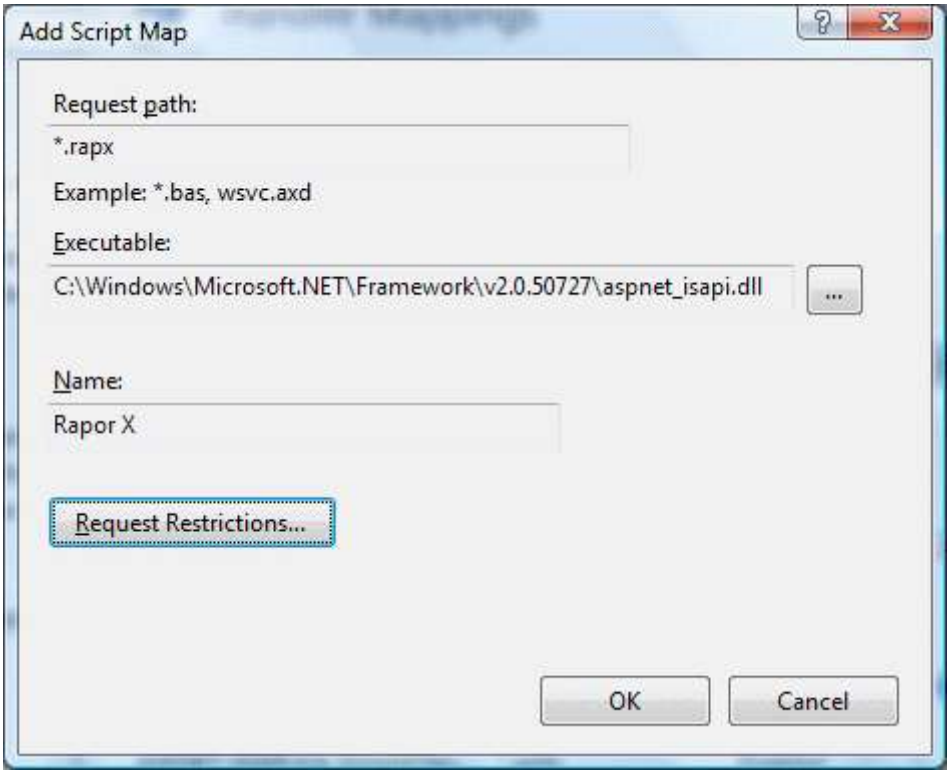
httpHandlers elementi **system.web** elementi içerisinde tanımlanmalıdır. örnekte **rapx** uzantılı dosyalara gelecek herhangibir talebin**(Get, Post vb...) type niteliğinde(attribute)** belirtilen**RaporHandler** sınıfına ait nesne örneği tarafından karşılanacağı belirtilmektedir. Bu adım tek başına yeterli değildir. Ayrıca **IIS(Internet Information Services**) üzerinden, rapx uzantılı dosyalara gelecek olan talepler için **AspNet_Isapi.dll**' inin devreye gireceğinin belirtilmesi gerekmektedir. örnek senaryo **IIS 7.0** üzerinde geliştirilmektedir. IIS 7.0 üzerinden**OzelHandlerKullanimi** isimli web uygulaması adına **rapx-AspNet_Isapi.dll** eşleştirmesi için öncelikli olaran **InetMgr** aracı üzerinden ilgili web uygulamasına geçilmeli ve **Handler Mappings** kısmı aşağıdaki ekran görüntüsünde olduğu gibi seçilmelidir.



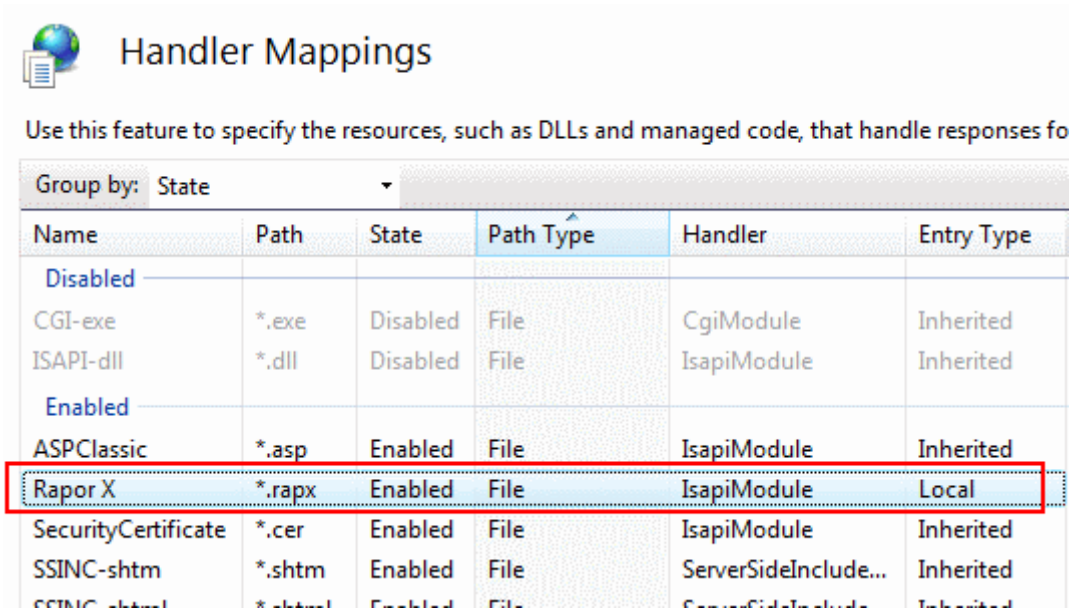
Handler Mappings kısmında varsayılan ve izin verilen uzantı eşleştirmeleri yer almaktadır. Bu kısımda sağ tıklanarak açılan menüden **Add Script Map** seçilmeli ve gerekli eşleştirme IIS tarafına bildirilmelidir.



Add Script Map ile açılan iletişim kutusunda ise aşağıdaki ayarların yapılması gerekmektedir.



Buna göre **rapx** uzantılı taleplerin **.Net 2.0** çalışma zamanında yer alan **AspNet_Isapi.dll** tarafından ele alınacağı belirtilmektedir. Uygulamaya ait **web.config** dosyası içerisinde gerekli Handler tanımlamaları yapıldığı için **AspNet_Isapi.dll** program arayüzü, gelen **talep(Request)** için **RaporHandler** sınıfının devreye girmesini sağlayacaktır. Bu işlemin ardından **Handler Mappings** kısmına aşağıdaki ekran görüntüsünde de görüldüğü gibi rapx uzantısı için gerekli eşleştirme eklenecektir.



Yapılan bu değişikliker sonrasında web uygulmasında ait **web.config** dosyası içerisindede aşağıdaki gibi yeni bir tanımlama oluşacaktır. Bir başka deyişle **IIS(Internet Information**

Services) üzerinde yapılan eşleştirmeye ait bilgiler **system.webServer** elementi içerisindeki **handlers** kısmına eklenmektedir.

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
   <appSettings />
   <connectionStrings />
   <system.web>
      <httpHandlers>
         <add path="*.rapx"
type="ReportHandlerLibrary.RaporHandler,ReportHandlerLibrary" verb="*"
validate="true" />
      </httpHandlers>
      <compilation debug="true" />
      <authentication mode="Windows" />
   </system.web>
   <system.webServer>
      <handlers>
         <add name="Rapor X" path="*.rapx" verb="*" modules="IsapiModule"
scriptProcessor="C:\Windows\Microsoft.NET\ Framework\v2.0.50727\aspnet_isapi.dll"
resourceType="File" />
      </handlers>
   </system.webServer>
</configuration>
```

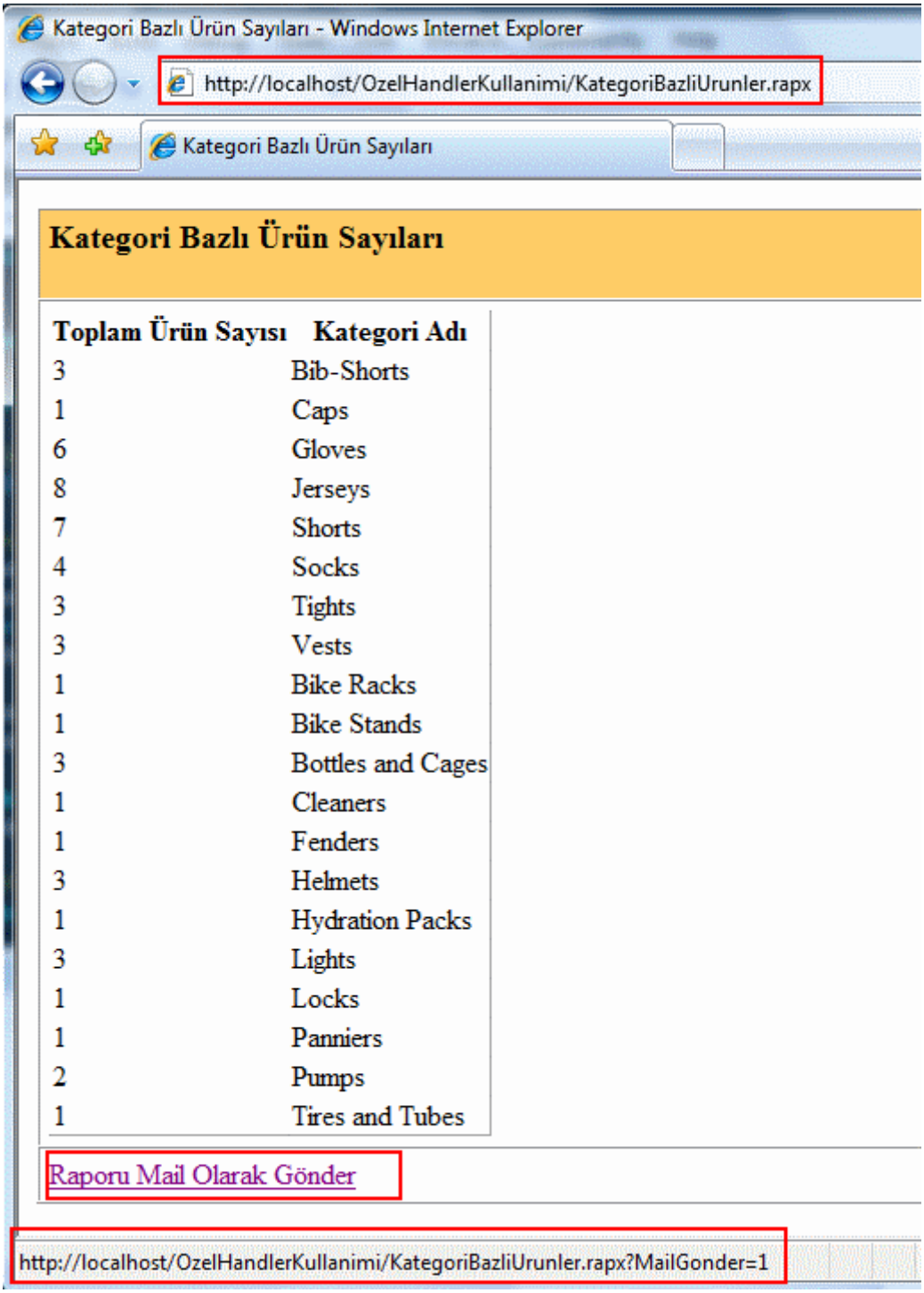
Artık test amacıyla web uygulamasına **rapx** uzantılı örnek bir içerik eklenebilir. örnek olarak **AdventureWorks** veritabanında yer alan **Products** tablosu ile ilişkili bir rapor düşünülebilir. Bu raporun stok seviyesi belirli bir değerin altında olan ürünlerin kategori bazlı sayılarını verdiğini düşünelim. Buna göre web uygulaması altında tutulacak olan rapx uzantılı metin dosyasının içeriği aşağıdaki gibi tasarlanabilir.*(Bu dosyayı eklerken **Add New Item-Text File** seçerek **KategoriBazliUrunler.rapx** gibi bir seçim yapılması gerektiğine dikkat edilmelidir.)*

KategoriBazliUrunler.rapx;

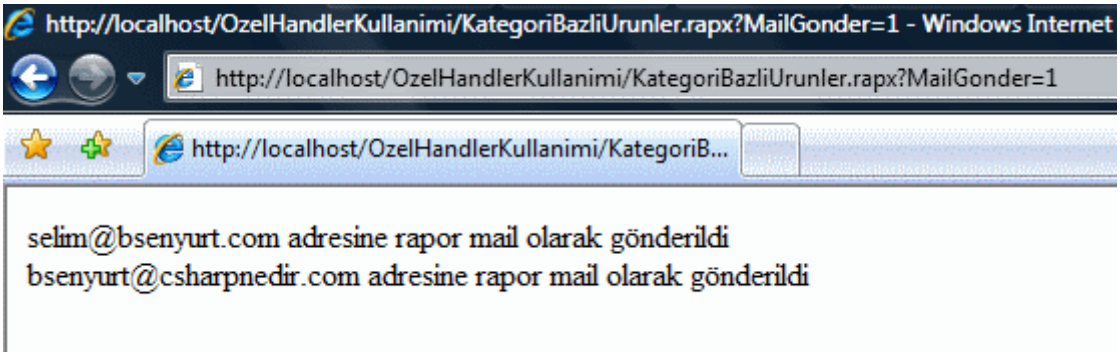
```
<?xml version="1.0" encoding="utf-8" ?>
<Raporlar>
   <Rapor Id="1">
      <Baslik ArkaPlan="#FFCC11">Kategori Bazlı ürün Sayıları</Baslik>
      <Baglanti Sspi="true">
         <Sunucu>localhost</Sunucu>
         <Veritabani>AdventureWorks</Veritabani>
         <KullaniciAdi></KullaniciAdi>
         <Sifre></Sifre>
      </Baglanti>
```

```
      <Sorgu Sp="false">
         <Cumle> Select Count(P.ProductID) [Toplam ürün Sayısı],PSC.Name [Kategori
Adı] From Production.Product P Join Production.ProductSubCategory PSC On
P.ProductSubCategoryID=PSC.ProductSubCategoryID Group By
PSC.Name,SafetyStockLevel Having P.SafetyStockLevel&lt;@StockLevel
         </Cumle>
         <Parametreler Nereden="Xml">
            <Parametre Ad="@StockLevel" Deger="10"/>
         </Parametreler>
      </Sorgu>
      <MailListesi MailGonder="true">
         <Mail>selim@bsenyurt.com</Mail>
         <Mail>bsenyurt@csharpnedir.com</Mail>
      </MailListesi>
   </Rapor>
</Raporlar>
```

Burada basit olarak **Product** ve **ProductSubCategory** tablolarının **Join** ile birleştirilmiş hali üzerinden bir gruplama sorgusu gerçekleştirilmektedir.
Sorgular **StockLevel** parametresinin değeri 10'dan küçük olanlar için yapılmaktadır. Parametrenin değerinin **XML** içerisinden geleceğini belirtmek için Nereden niteliğine **XML** değeri atanmıştır. Söz konusu rapor için**MailListesinde** belirtilen kişilere mail gönderme opsiyonu açık bırakılmıştır. Rapor, **localhost** isimli sunucudaki **AdventureWorks** veritabanı üzerinden **SSPI** ile gerçekleştirilen bir bağlantı üzerinden alınmaktadır. Artık **OzelHandlerKullanimi** adresi üzerinden **KategoriBazliUrunler.rapx** dosyasına bir talepte bulunulursa aşağıdaki gibi bir sonuç ortaya çıkacaktır.



Görüldüğü gibi sorgu sonucu elde edilen rapor ekrana basit bir tablo olarak basılmıştır. Mail gönderme seçeneği aktif olduğu içinde **Raporu Mail Olarak Gönder** başlıklı linkte sayfa çıktısında yer almaktadır. Bu linke basıldığı takdirde sunucuya yeni bir talep daha gönderilecektir. Şimdilik sadece ilgili MailListesi elementinin içeriği değerlendirilmiştir. Aşağıdaki ekran görüntüsünde bu durum gösterilmektedir.



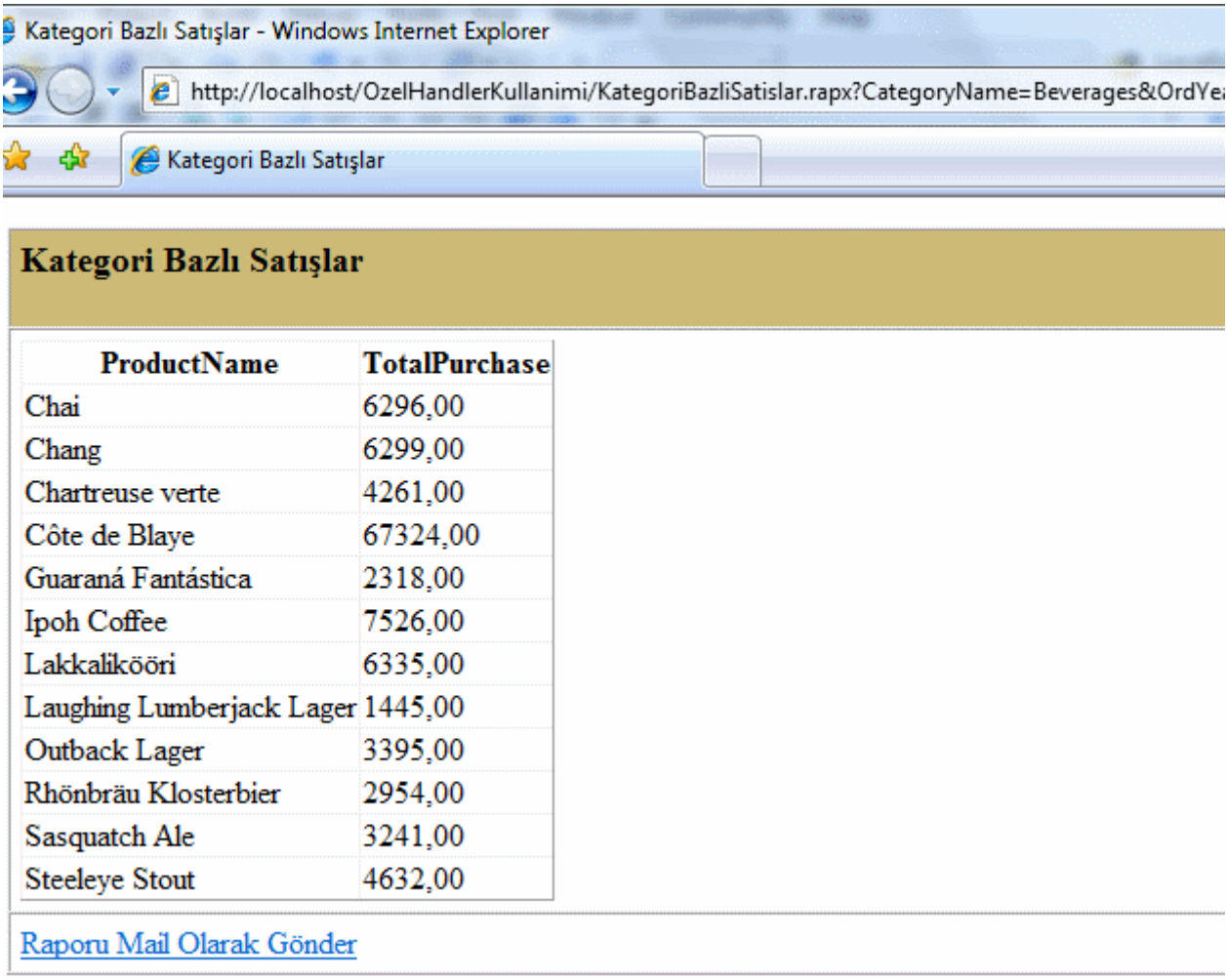
Yeni bir test ile devam edelim. Bu kez raporun parametrelerine ait değerlerin **QueryString** yardımıyla geldiğini göz önüne alalım. Ayrıca raporun sonuçlarının **Northwind** veritabanı altında yer alan **SalesByCategory** isimli **saklı yordam(stored procedure)** üzerinden elde edildiğini düşünelim. MailGonderme seçeneği yine aktif olsun. Bu durumda **rapx** dosyasının içeriğinin aşağıdaki gibi tasarlanması gerekecektir.

KategoriBazliSatislar.rapx;

```
<?xml version="1.0" encoding="utf-8" ?>
<Raporlar>
  <Rapor Id="1">
    <Baslik ArkaPlan="#CCBB77">Kategori Bazlı Satışlar</Baslik>
    <Baglanti Sspi="true">
      <Sunucu>localhost</Sunucu>
      <Veritabani>Northwind</Veritabani>
      <KullaniciAdi></KullaniciAdi>
      <Sifre></Sifre>
    </Baglanti>
    <Sorgu Sp="true">
      <Cumle>SalesByCategory</Cumle>
      <Parametreler Nereden="QueryString">
        <Parametre Ad="@CategoryName"/>
        <Parametre Ad="@OrdYear"/>
      </Parametreler>
    </Sorgu>
    <MailListesi MailGonder="false">
      <Mail>selim@bsenyurt.com</Mail>
      <Mail>bsenyurt@csharpnedir.com</Mail>
    </MailListesi>
  </Rapor>
</Raporlar>
```

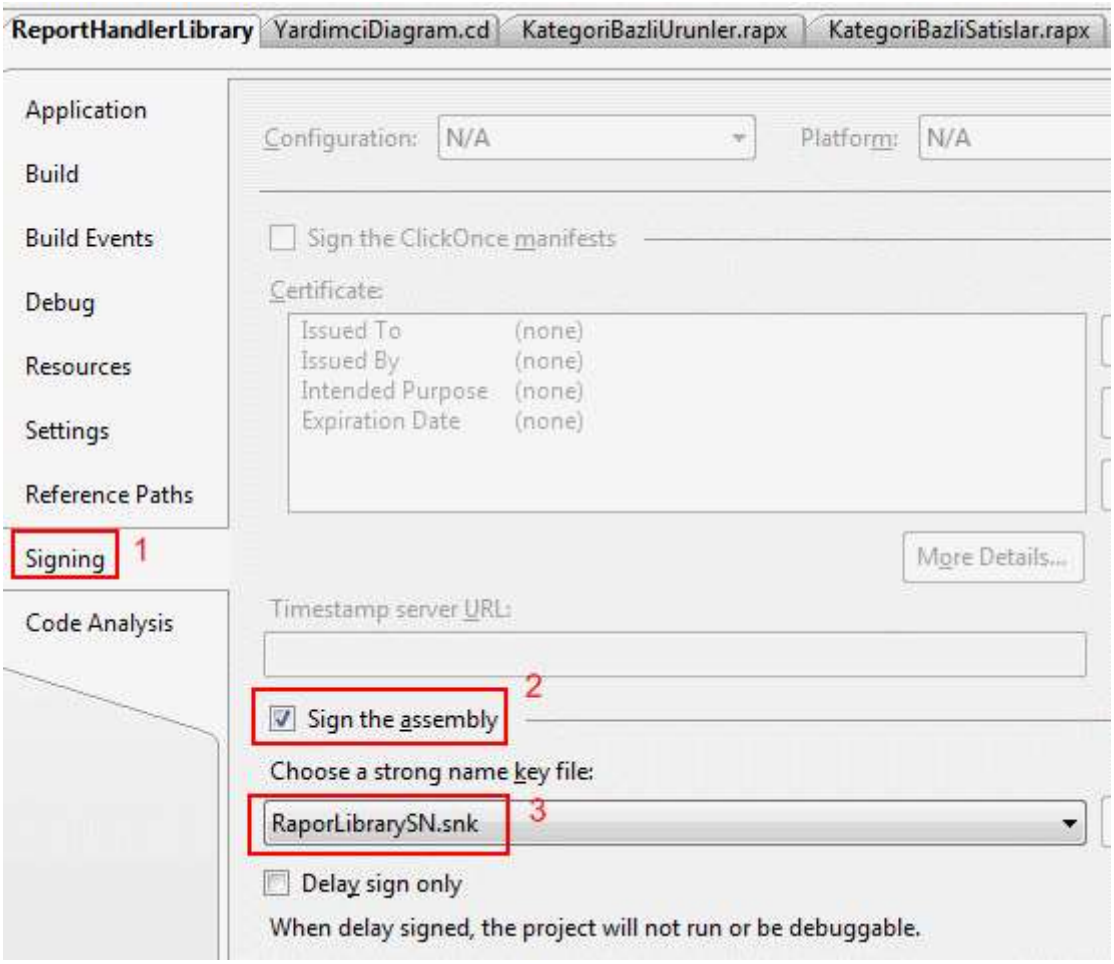
Buna göre tarayıcı penceresinden **http://localhost/OzelHandlerKullanimi/KategoriBazliSatislar.rapx?Cat**

egoryName=Beverages&OrdYear=1999 adresi talep edilirse aşağıdakine benzer bir ekran görüntüsü ile karşılaşılacaktır.



Elbette uygulamada pek çok hata göz ardı edilmektedir. Söz gelimi **KategoriBazliSatislar.rapx** için **QueryString** parametreleri kullanılmassa **çalışma zamanı hataları(Run time exceptions)**alınması çok doğaldır. Bu gibi hataların ele alınması bir başka deyişle kodun tekrardan revize edilerek düzenlenmesi gerekmektedir.

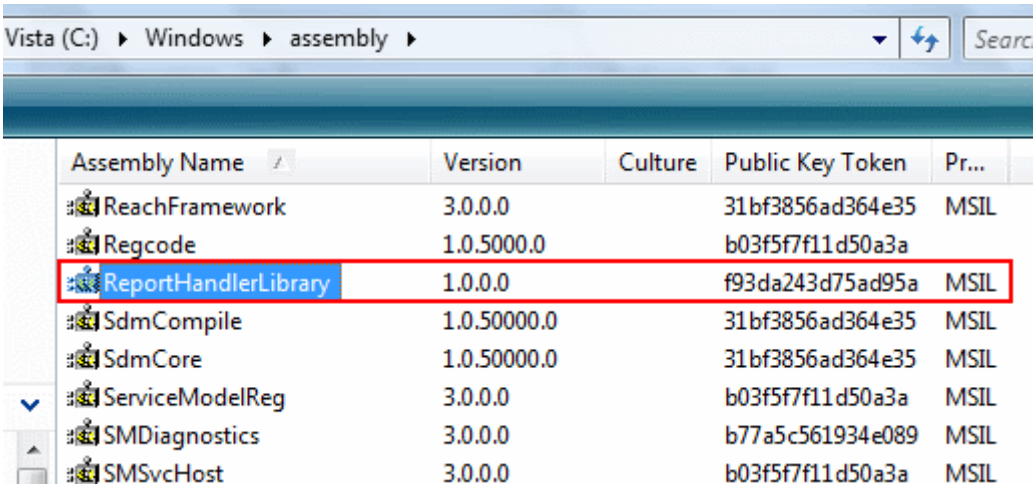
Gelelim **Handler** tipinin sunucu üzerinde nasıl ele alınabileceğine. Bunun için geliştirilen Handler tipini içeren **assembly**' ın **Global Assembly Cache(GAC)** içerisine atılması ve **root web.config**dosyası içerisindeki **httpHandlers** elementi altında bildirilmesi yeterlidir. örneğimizde geliştirdiğimiz **ReportHandlerLibraray.dll** isimli **assembly**' ı **GAC'a** atmadan önce **Visual Studio 2005** ortamında aşağıdaki gibi **Strong Name** ile imzalamamız gerekmektedir. *(İstenirse bu imzalama işlemi komut satırından **sn.exe** aracı ilede gerçekleştirilebilir.)*



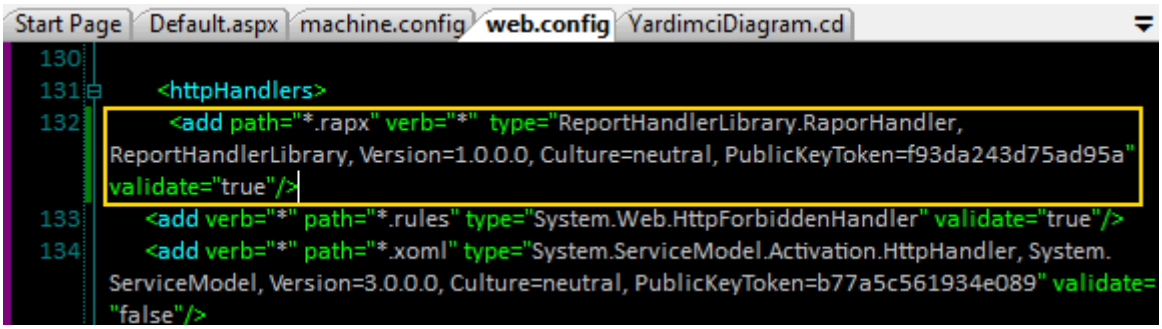
Bu işlemin ardından **ReportHandlerLibrary**' in **gacutil.exe** aracı yardımıyla yada sürükle bırak tekniği ile **Windows\Assembly** klasörüne atılarak **GAC**' a eklenmesi yeterlidir.

Not : *Burada ReportHandlerLibrary projesinin* ***Relase*** *modda üretiminin yapılarak,* ***Output Path*** *olarak*
çıktının ***C:\Windows\Microsoft.NET\Framework\v2.0.50727*** *klasörünü işaret edecek şekilde düzenlenmesi ve daha sonra buradan* ***GAC****' a atılmasıda tercih edilebilir.*

Sonuç olarak **Windows\Assembly** klasörü altına **ReportHandlerLibrary**' si aşağıdaki ekran görüntüsünde olduğu gibi eklenmiş olacaktır.

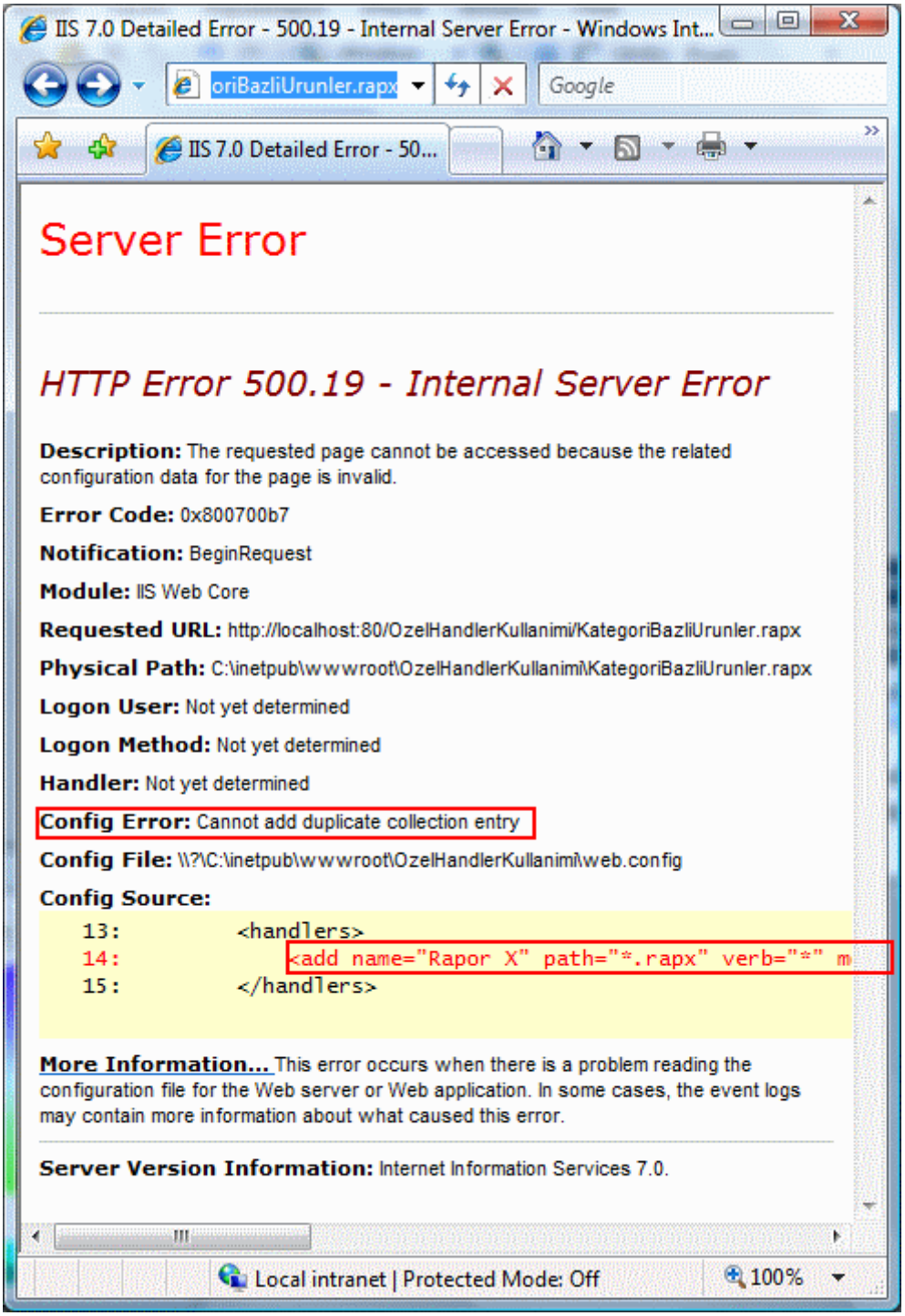


Şimdi tek yapılması gereken bu **assembly**' ın **root web.config** içerisinde bildirilmesidir. Bunun için **C:\Windows\Microsoft.NET\Framework\v2.0.50727\CONFIG** klasöründe yer alan web.config dosyasına aşağıdaki gibi handler bildiriminin yapılması gerekmektedir.



Bu işlemlerin ardından **IIS** üzerinden **rapx** uzantılı dosyalara gelecek olan taleplerin **AspNet_Isapi.dll** tarafından ele alınacağınında belirtilmesi gerekir. Makalemizin başında**OzelHandlerKullanimi** isimli web sitesi için yaptığımız bu işlemin aynısı bu kez **root web sitesi** için benzer şekilde yapılarak gerçekleştirilmelidir. Sonuç itibariyle sunucu üzerinde**Asp.Net 2.0** ile geliştirilen herhangibir web sitesi altında tasarlanacak olan tüm **rapx** uzantılı dosyalar **RaporHandler** sınıfı tarafından ele alınabilecektir.

Not : *Kendi handler tipimizi* ***root web.config*** *dosyasında tanımladığımızda, web uygulaması içerisinde yer alan web.config dosyası içerisinde bırakılan handler tanımlamaları nedeni ile* ***çalışma zamanı hataları(Run time exceptions)*** *alınır. Bu nedenle makaledeki örnekte yer alan OzelHandlerKullanimi sitesindeki* ***rapx*** *dosyaları, root web.config' de yapılan handler bildirimleri sonrası çalışmayacaktır.*



Sebep handler tanımlamasının hem root web.config hemde site içerisindeki web.config'de iki kez yapılmış olmasıdır. Bunu düzeltmek için OzelHandlerKullanimi sitesine ait web.config içerisinde yapılan handler bildirimlerini kaldırmak gerekmektedir.

Kendi **HttpHandler** tiplerimizi tasarlamak için ele aldığımız örnek senaryo çok daha fazla geliştirilebilir. Hatta bir web sunucusu üzerinde konuşlandırılacak olan **rapx** dosyalarının daha kolay hazırlanabilmesini sağlamak için görsel bir arabirimde geliştirilebilir. Sadece Rapx uzantılı dosyaları barındıracak olan bir **Asp.Net** uygulaması geliştirilerek tüm raporların tek bir merkezde toplanması sağlanabilir. Hatta bu işi dahada ileriye götürecek

olursak, raporların yetki tabanlı olacak şekilde ele alınabilmesi için gerekli hazırlıklarda yapılabilir. Bu noktada, tasarlanan bu sistemin **Reporting Service** alt yapısına ne kadar benzediğini tartışmakta yarar vardır. Sistemin herhangibir veritabanına destek verecek şekilde ele alınması ise çok daha etkili bir raporlama arayüzü oluşturulmasına ön ayak olacaktır.

Bu makalemizde daha önceden ele aldığımız ancak geçerli bir ihtiyaç veya senaryo üzerine oturtamadığımız **HttpHandler** kavramını bir örnek üzerinde adım adım incelemeye çalıştık. Sizlerde farklı senaryoları göz önüne alarak ve makalede yer alan örneği dahada geliştirirek son derece etkili sonuçlara varabilirsiniz. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

.Net Remoting Dünyasından WCF' e Geçmek (2007-12-03T04:23:00)

wcf,.net remoting,

Windows tabanlı olan **Servis Yönelimli Mimari(Service Oriented Architecture)** tekniklerinden biriside .Net Remoting' dir. .Net Remoting mimarisi ağırlıklı olarak **TCP** bazlı ve **Binary** tabanlı paket iletiminde kullanılır. En büyük özelliklerinden birisi, sadece Windows işletim sistemlerinden oluşan ağlarda koşabilmesidir.
Elbette **HTTP** üzerinden **SOAP-Simple Object Access Protocol**formatına uygun alt yapı kurulmasıda mümkündür. Bu sayede internet ağındada ektin şekilde kullanılabilir. Ancak Windows bağımlı olması platform bağımsızlığını ortadan kaldırmaktadır. Günümüzde **WCF(Windows Communication Foundation)** gibi daha ölçeklenebilir(Scalable), birleştirilmiş(Unified) bir Servis Yönelimli Mimari(SOA) açılımıda mevcuttur. Bu durumda geliştiricilerin karşısına önemli bazı sorular ve sorunlar çıkmaktadır. İşte bunlardan bir kaçı;

- Var olan .Net Remoting alt yapısı, WCF tabanlı bir hale dönüştürülebilir mi?
- Dönüştürülürse ne gibi düzenlemeler yapmak gerekir?
- Sıfırdan WCF tabanlı bir model geliştirmek yerine, .Net Remoting WCF'e göç etmek mantıklı mıdır?
- .Net Framework 2.0 hatta 1.1 uyumlu sistemler üzerine kurulmuş senaryolarda, var olan .Net Remoting uygulamalarında yapılacak WCF düzenlemeleri geçerli olacak mıdır? Yoksa bu makinelere Framework 3.0 yüklenmeli midir?

Bu sorular farklı açılardan bakıldığında artabilir. İlerleyen kısımlarda bu sorulara ve beraberinde gelen sorunlara cevap bulmamızı kolaylaştıracak şekilde ilerlemeye çalışacağız.

NOT : *Windows Communication Foundation,* ***.Net Remoting****,* ***Web Servisleri****,* ***MSMQ****,* ***Named Pipes****,* ***WSE*** *gibi pek çok dağıtık mimari modelini tek bir çatı altında birleştirip sunabilmesiyle ön plana çıkmış bir* ***Servis Yönelimli Mimari(Service Oriented Architecture)*** *alt yapısıdır. Bu alt yapı, önceki mimarilerde daha fazla kodlama*

*gerektiren yada geliştiricileri zorlayan standartların(örneğin **WS-Specifications**) kolay ve etkin bir şekilde uygulanabilmesini hatta bir arada tutulabilmesine de izin vermektedir.*

Bazı durumlarda çalışmakta olan sistemi yeni bir mimariye adapte etmenin maliyeti çok yüksek olabilir. Söz gelimi ölçek olarak çok büyük çaplı projelerde sistemi yeniden tasarlamak son derece sancılı bir süreç olabilir. Ancak böyle bir durum söz konusu değilse karşımızda iki alternatif olacaktır. Var olan sistemi modifiye ederek WCF' e taşımak yada sıfırdan tasarlamak. Aslında var olan bir .Net Remoting alt yapısını bir kaç küçük değişiklik ile WCF' e taşımak son derece kolaydır. Herşeyden önce .Net Remoting içerisinde yer alan temel parçaları göz önüne alarak ilerlemekte yarar vardır.

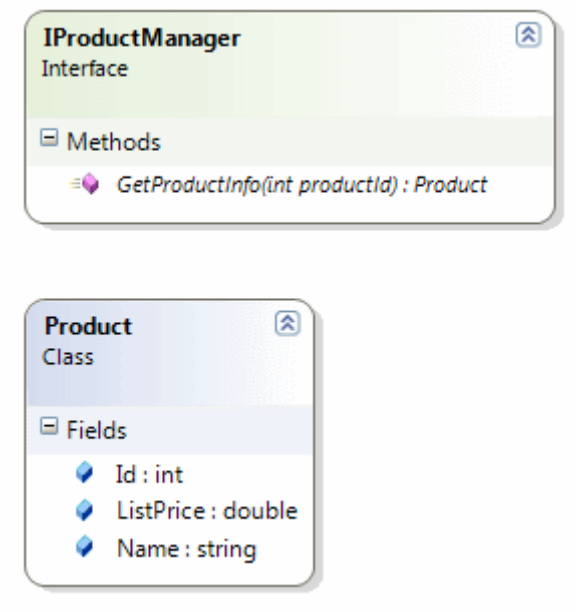
.Net Remoting ile hazırlanmış bir sistemde **istemcilerin(Clients)** kullanacakları fonksiyonellikleri barındıran **uzak nesneler(Remote Objects)** bir **sunucu(Server)** uygulama üzerinden **Client Activated Object** yada **Server Activated Object** modeline uygun olacak şekilde yayınlanırlar. Hatta SAO nesneleride **Singleton** veya **SingleCall** olarak tasarlanırlar. İstemci uygulamalar aslında uzak nesne referanslarını kullanırken bu referanslar **Marshal By Reference** modeline göre sunucu üzerinde örneklenirler. Bir başka deyişle istemci uzak nesneyi sanki kendi **uygulama alanı(Application Domain)** içerisindeymiş gibi kullanırlarken, tüm işlevler sunucu üzerinde gerçekleşmektedir.

çok doğal olarak burada istemci ve sunucu arasındaki mesaj trafiğinin ortak bir zemine oturtulması şarttır. Bu nedenle genellikle istemci ve sunucu uygulamalar hizmete ait nesne tasarımını içeren şartnamelerin yer aldığı bir **arayüz(Interface)** tipini ortaklaşa referans ederler. Böylece servis tarafındaki fonkisyonelliklerin iç yapılarının değiştirilmesi halinde istemciler üzerinde yeniden güncelleme yapılmasına gerek kalmayacaktır. Ayrıca istemciler sadece arayüzü görebildiklerinden, çağırdıkları fonksiyonelliklerin iç yapısını bilmeyeceklerdir ki buda servis kodunun tam olarak istemci tarafından görülmesini engellemektedir. Zaten WCF' in tamamen arayüzlere dayalı bir sistemi önermesinin ve kullanmasının en önemli nedenleride bunlardır.

Bu noktada geliştirici tarafından tanımlanan **serileştirilebilir tiplerinde (Serializable Types)** ortak bir **sınıf kütüphanesi(Class Library)** üzerinde olması gerekmektedir. Serileştirilebilir nesneler aslında servis tarafında örneklenip sadece **varlıkları(Entities)** istemci tarafına aktarılan örnekler olarak düşünülebilir. Söz gelimi istemcinin talep ettiği bir ürünün, Product isimli bir sınıfa ait nesne örneği olacak şekilde servis tarafında doldurulması ve istemciden elde edilmesi buna örnek olarak verilebilir. Burada istemcinin talebi sonrasında servis tarafından dönen veriler Product isimli sınıfın **özellik(Property)** veya **alanlarının(Fields)** değerleridir. Serileştirilebilir nesnelerin WCF mimarisinde ele alınması çok daha kolaydır. özellikle **versiyonlamanın(Versioning)** WCF üzerinden gerçekleştirilmesi daha esnektir.

***NOT :** Makalenin konusu .Net Remoting' den WCF' e bir iki adımda taşınmanın nasıl sağlanabileceğini göstermek olduğundan, .Net Remoting ve WCF mimarilerinin çok geniş detayları göz ardı edilmektedir.*

Artık bir örnek üzerinden hareket ederek devam edebiliriz. öncelikli olarak bir **.Net Remoting** uygulamasını **Visual Studio 2005** üzerinden geliştiriyor olacağız. İlk olarak istemci ve servis uygulamasının ortaklaşa kullanacağı **sınıf kütüphanesini(Class Library)** tasarlayarak işe başlanabilir. Sınıf kütüphanesi içerisindeki tiplere ait **sınıf diyagramı(Class Diagram)** ve kod içerikleri aşağıdaki gibidir.



IProductManager arayüzü(Interface);

```
using System;

namespace AdvLibrary
{
   public interface IProductManager
   {
      Product GetProductInfo(int productId);
   }
}
```

örneğin daha basit olarak ele alınabilmesi için arayüz(Interface) tanımlamasında, geriye Product tipinden nesne örneği döndürebilen ve integer tipinden bir parametre alabilen GetProductInfo isimli bir metod bildirimi yer almaktadır.

Product sınıfı(Class);

```
using System;

namespace AdvLibrary
{
   [Serializable]
```

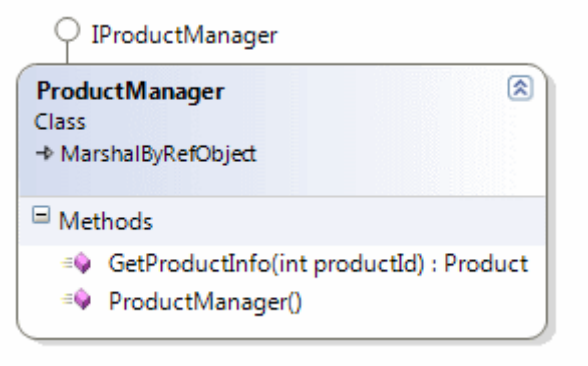
```
    public class Product
    {
        public int Id;
        public string Name;
        public double ListPrice;
    }
}
```

Product isimli sınıf içerisinde bir ürüne ait Id, Name ve ListPrice değerleri tutulmaktadır. Bununla birlikte Product sınıfı için dikkat çekici en önemli özelliklerden biriside **Serializable niteliği(attribute)** ile imzalanmış olmasıdır. Burada açık bir şekilde Product sınıfına ait nesne örneklerinin **Marshal By Value** olarak sunucudan istemciye serileşerek taşınabileceği garanti altına alınmaktadır.

Servis tarafında yer alan program, söz konusu örnekte bir **Console** uygulaması olarak tasarlanmaktadır. Bu uygulamanın çok doğal olarak yukarıda tasarlanan tipleri içeren AdvLibrary isimli **sınıf kütüphanesini(Class Library)** referans ediyor olması gerekmektedir. Nitekim sunucu uygulama içerisinde asıl iş yapan sınıfın uyarlayacağı arayüz ve kullanacağı serileştirilebilir tip bu sınıf kütüphanesi içerisinde bulunmaktadır.

NOT : *Gerçek hayat senaryolarında en çok tercih edilen .Net Remoting yöntemi, sunucu uygulamasını bir* ***Windows Service*** *olarak tasarlamaktadır. Bu yönetimi daha güçlü olan, ölçeklenebilir ve güvenli bir uygulama seçimidir. Bunun dışında* ***IIS(Internet Information Services)*** *üzerinden* ***barındırma(Host)*** *imkanıda bulunmaktadır. Diğer taraftan sunucu uygulama bir Windows uygulaması hatta bu örnekte olduğu gibi bir Console uygulamasıda olabilir.*

Tekrar sunucu uygulamasına dönülecek olursa, IProductManager isimli **arayüzü(Interface)** uygulayan uzak nesne sınıfının aşağıdaki gibi tasarlanabileceği düşünülebilir.




```
using System;
using AdvLibrary;
using System.Data.SqlClient;
```

```
namespace ServerApp
{
   public class ProductManager:MarshalByRefObject,IProductManager
   {
      public ProductManager()
      {
         Console.WriteLine("ProductManager nesnesi oluşturuldu...");
      }
      #region IProductManager Members

      public Product GetProductInfo(int productId)
      {
         Product prd = null;
         using (SqlConnection conn = new SqlConnection("data
source=.;database=AdventureWorks;integrated security=SSPI"))
         {
            using (SqlCommand cmd = new SqlCommand("Select
ProductId,Name,ListPrice,SellStartDate From Production.Product Where
ProductId=@PrdId", conn))
            {
               cmd.Parameters.AddWithValue("@PrdId",productId);
               conn.Open();
               SqlDataReader reader = cmd.ExecuteReader();
               if (reader.Read())
               {
                  prd = new Product();
                  prd.Id = productId;
                  prd.Name = reader["Name"].ToString();
                  prd.ListPrice = Convert.ToDouble(reader["ListPrice"]);
               }
               reader.Close();
            }
         }
         return prd;
      }

      #endregion
   }
}
```

Bu sınıf içerisinde yer alan metod uyarlamasında AdventureWorks isimli veritabanına gidilmekte ve Production **şemasında(Schema)** yer alan Product tablosundan parametre olarak gelen ProductID değerine sahip olan ürün bilgisi geriye döndürülmektedir. Nesne oluşumlarının kolay takip edilebilmesi amacıyla ProductManager sınıfı içerisine birde **varsayılan yapıcı metod(Default Constructor)** ilave edilmiştir. Burada dikkat

edilmesi gereken nokta sınıfın IProductManager isimli arayüz dışında **MarshalByRefObject** sınıfından türemiş olmasıdır.

Sunucu tarafındaki Remoting ayarlarını konfigurasyon bazlı olarak aşağıdaki config dosyasında tanımlayabiliriz.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.runtime.remoting>
      <application>
         <channels>
            <channel ref="Tcp Server" port="4500"/>
         </channels>
         <service>
            <wellknown mode="Singleton"
type="ServerApp.ProductManager,ServerApp" objectUri="ProductMng.rem" />
         </service>
      </application>
   </system.runtime.remoting>
</configuration>
```

Konfigurasyon dosyası içerisinde tahmin edileceği üzere **Wellknown** bir nesne tanımı yapılmıştır. Bu nesne **Singleton** modeline göre çalışacaktır. Bir başka deyişle tüm istemcilerin talepleri aynı uzak nesne referansı üzerinden karşılanmaktadır. Sunucu uygulama, **uzak nesne(Remote Object)** için **TCP** bazlı 4500 numaralı portu kullanıma sunmaktadır. Buradaki konfigurasyon ayarlarının sunucu uygulama üzerinde tesis edilmesi içinde aşağıdaki başlangıç kodlarının yazılması yeterlidir.

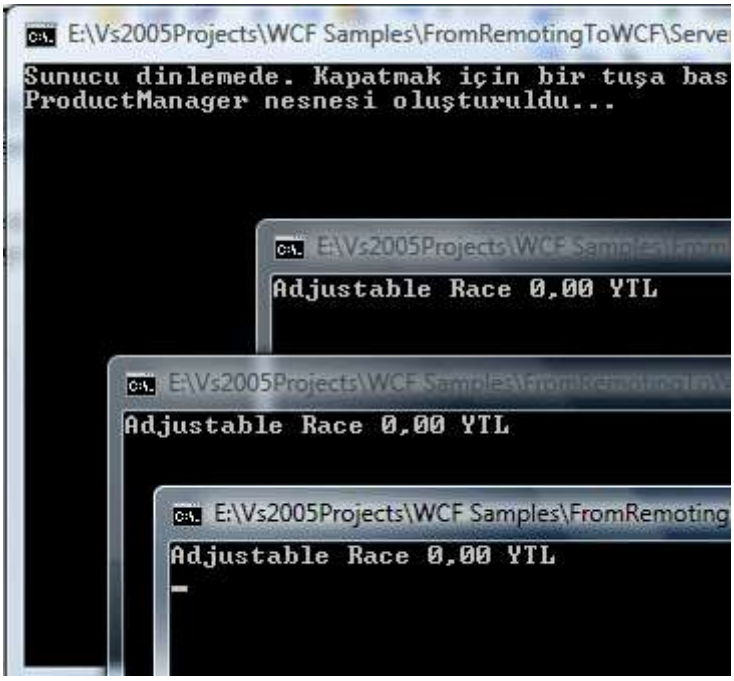
```
using System;
using AdvLibrary;
using System.Runtime.Remoting;

namespace ServerApp
{
   class Program
   {
      static void Main(string[] args)
      {
         RemotingConfiguration.Configure("..\\..\\App.config",false);
         Console.WriteLine("Sunucu dinlemede. Kapatmak için bir tuşa basınız...");
         Console.ReadLine();
      }
   }
}
```

Bu noktada şunu hatırlamakta yarar vardır; sunucu uygulama çalıştığı sürece istemcilerden gelecek taleplere cevap verilebilir.

Gelelim istemci uygulamaya. Herşeyden önce istemci uygulamanında arayüzü barındıran ortak kütüphaneyi referans etmesi gerekmektedir. Diğer taraftan istemci uygulamanın kod içeriği aşağıdaki gibi olabilir.

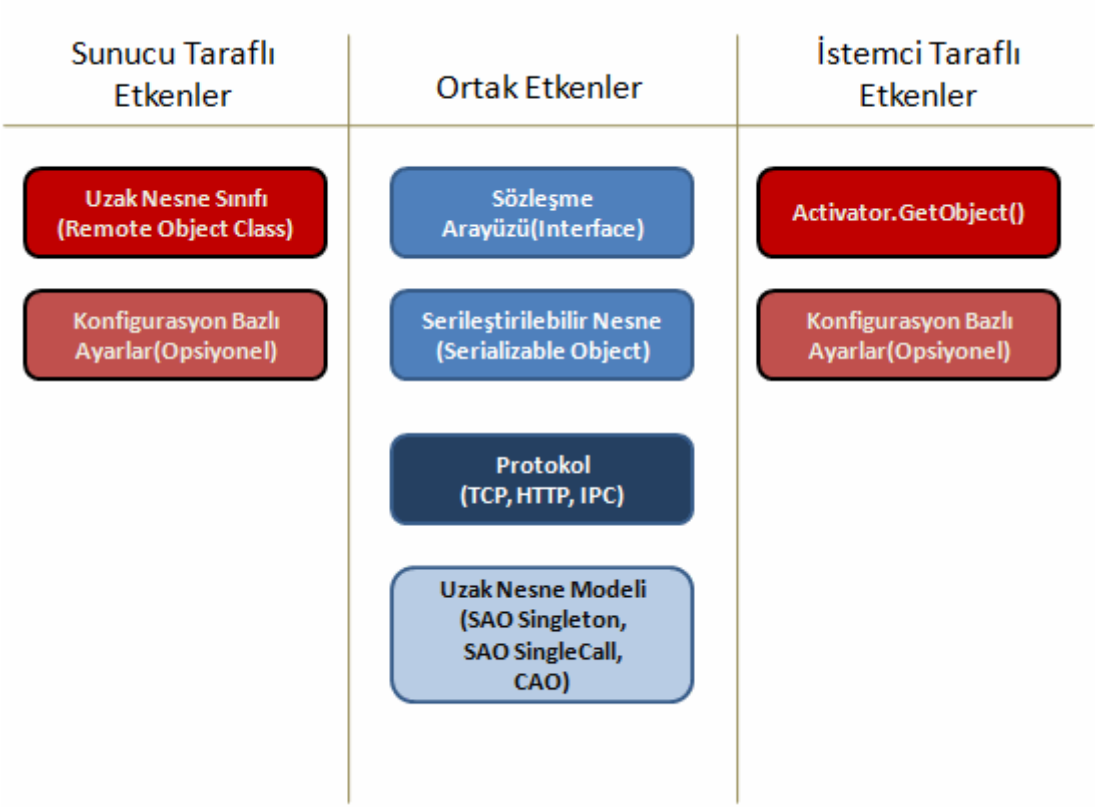
```
using System;
using AdvLibrary;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         IProductManager prdMng = (IProductManager)Activator.GetObject(
typeof(IProductManager), "tcp://localhost:4500/ProductMng.rem");
         Product prd=prdMng.GetProductInfo(1);
         Console.WriteLine(prd.Name+" "+prd.ListPrice.ToString("C2"));
         Console.ReadLine();
      }
   }
}
```

Burada en kritik nokta, **arayüz(interface)** tercihi nedeniyle **Activator** sınıfının **GetObject** metodunun kullanılmasıdır. Bu metod ile aslında ikinci parametre ile belirtilen adresten ProductMng.rem isimli tanımlayıcının**(Uniform Resource Identifier)** işaret ettiği uzak nesne referansı talep edilmektedir. İstemciye döndürülecek olan referansın taşınabileceği tek yer IProductManager arayüzü olduğundan da metodun dönüş tipi **bilinçli(Explicit)** olarak çevrilmektedir. Buraya kadar yapılan hazırlıklar sonucunda test işlemi yapılırsa aşağıdakine benzer çıktılar elde edilir.



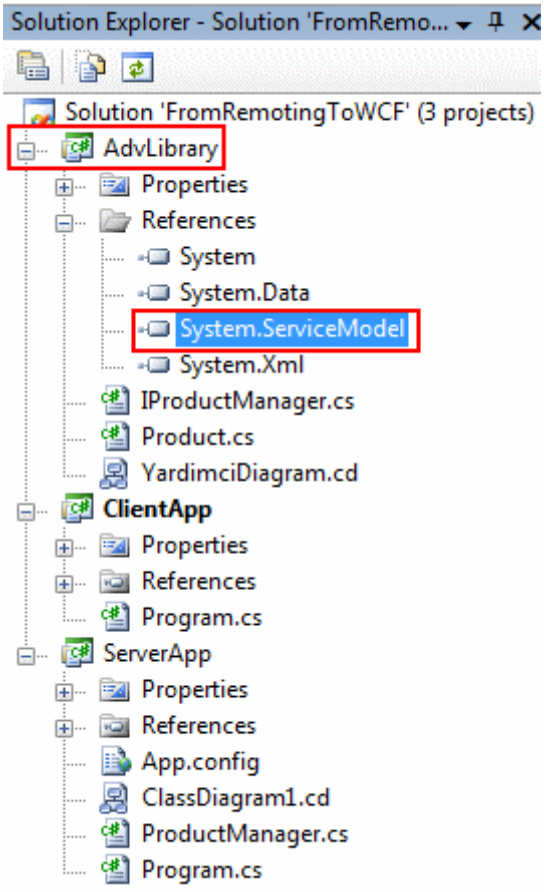
Görüldüğü gibi istemciler başarılı bir şekilde sunucu üzerinden taleplerini karşılayabilmektedirler. *(Burada oluşturulan sistemin düzgün çalıştığının kontrol edilmesi için sunucu uygulama çalıştırılmadan bir istemcinin çalıştırılması denenebilir. Eğer gerçekten sunucu üzerindeki nesne kullanılmaya çalışılıyorsa çalışma zamanında istemci tarafında "**No Connection Could Be Made, Becouse the Target Machine Actively Refused It...**" hatası alınır.)*

Buraya kadar anlatılanlar basit olarak bir **.Net Remoting** uygulamasının alt yapısını özetlemektedir. Aşağıdaki şekil üzerinden, yapılanlar ve sonuçları tartışılabilir.



Şekilde .Net Remoting içerisinde 50000 metre yukarıdan bakıldığında göze çarpan etkenler betimlenmeye çalışılmıştır. Şimdi bu sistemi **Windows Communication Foundation** alt yapısına taşımaya çalışıyor olacağız.

İlk yapılması gereken servis tarafında sunulmak istenen fonksiyonellikleri tanımlayacak bir sözleşme(Contract) oluşturmaktır. Burada söz konusu olan **servis sözleşmesi(Service Contract)** bir arayüze rahatlıkla uygulanabilir. Tek yapılması gereken **System.ServiceModel.dll** isimli **.Net Framework 3.0 assembly**' ının referans edilmesi ve **ServiceContract** ile **OperationContract niteliklerinin(attributes)** kullanılmasıdır. Aşağıdaki şekilde Visual Studio 2005 üzerinden söz konusu assembly' ın referans edilişi görülmektedir.



Bu işlemin ardından IProductManager isimli arayüzün yapısı aşağıdaki gibi değiştirilmelidir.

```
using System;
using System.ServiceModel;

namespace AdvLibrary
{
   [ServiceContract]
   public interface IProductManager
   {
      [OperationContract]
      Product GetProductInfo(int productId);
   }
}
```

Bu değişiklik arayüzün **Windows Communication Foundation** uyumlu bir **serviş sözleşmesi(Service Contract)** olması için yeterlidir. **ServiceContract** niteliği ile tanımlanan servis sözleşmesi içerisinden dışarıya sunulabilecek fonksiyonelliklerin tamamı **OperationContract** niteliği(attribute) ile imzalanmalıdır. Yapılan bu değişiklik her ne kadar ortak kütüphane içerisinde olsada sonuç itibariyle tüm istemcilere gerekli

assembly' ın yeniden dağıtılması gerekmektedir. Ama zaten bu göz önüne alınması gereken bir aşamadır.

Gelelim sunucu tarafına. Sunucu tarafındada çok doğal olarak yapılması gereken bazı işlemler vardır. Herşeyden önce konfigurasyon içeriğinin WCF mimarisine uyumlu olacak şekilde değiştirilmesi gerekmektedir. Bu amaçla sunucu uygulama tarafındaki **App.config** dosyasının içeriği aşağıdaki gibi yenilenmelidir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <!--<system.runtime.remoting>
            <application>
               <channels>
                  <channel ref="Tcp Server" port="4500"/>
               </channels>
               <service>
                  <wellknown mode="Singleton"
type="ServerApp.ProductManager,ServerApp" objectUri="ProductMng.rem" />
               </service>
            </application>
         </system.runtime.remoting>-->
   <system.serviceModel>
      <services>
         <service name="ServerApp.ProductManager">
            <endpoint address="net.tcp://localhost:4500/ProductMng"
binding="netTcpBinding" contract="AdvLibrary.IProductManager" />
         </service>
      </services>
   </system.serviceModel>
</configuration>
```

Windows Communication Foundation(WCF) mimarisinde servis için belkide en önemli kavram **EndPoint**' dir. EndPoint içerisinde **WCF mimarisinin ABC' si** yer almaktadır. **AddressBindingConfiguration**yardımıyla servisin hangi lokasyondan, hangi protokol ile, hangi nesneyi nasıl ve ne şekilde sunduğu belirtilmektedir. **.Net Remoting** bazlı geliştirilen örnekte **TCP** bazlı ve **Binary** serileştirme yapan bir sistem kullanıldığından bu özellikleri bünyesinde barındıran bir tipin WCF tarafında ele alınması gerekmektedir. Bu işi **NetTcpBinding** isimli sınıf üstlenmektedir. Bu aynı zamanda **bağlayıcı tiptir(Binding Type)**. **Contract** bilgisi ile tahmin edileceği üzere servisin dış ortama sunduğu sözleşme ve operasyonları belirtilmektedir. Son olarak **Address** bilgisi ile servis üzerinde sunulan nesneye hangi adres üzerinden erişilebileceği belirtilmektedir.

***NOT : WCF** mimarisinde bir servis birden fazla **EndPoint** sunabilir. Bu geliştiricilere senaryoya göre birden fazla bağlayıcı tipin(Binding Type) ve dolayısıyla imkanın*

*sunulabildiği bir ortam hazırlayacaktır. Söz gelimi bir servis uygulaması WCF' e göre yazıldığında, **MSMQ**, **TCP**, **HTTPS** üzerinden vb... hizmet verebilecek mesaj seviyesinde(Message Level) yada iletişim seviyesinde(Transport Level) korumalı hizmetleri sunabilir. üstelik bunlar için ayrı ayrı uygulama çeşitleri tasarlanmasına gerek kalmaz. Bir başka deyişle bu tip bir senaryo için ayrı bir .**Net Remoting,***
***Xml Web Service** veya **MSMQ** uygulaması yazılmasına gerek yoktur. İşte WCF' in birleştirici rolünün önemi burada belirgin bir biçimde ortaya çıkmaktadır.*

Elbetteki sunucu uygulamanın kodlarınıda aşağıdaki gibi değiştirmek gerekmektedir. Artık **ServiceHost** isimli sınıf başlatıcı rolü üstlenmektedir.

```
using System;
using AdvLibrary;
//using System.Runtime.Remoting;
using System.ServiceModel;

namespace ServerApp
{
   class Program
   {
      static void Main(string[] args)
      {
         //RemotingConfiguration.Configure("..\\..\\App.config",false);
         ServiceHost host = new ServiceHost(typeof(ProductManager));
         host.Open();
         Console.WriteLine("Sunucu dinlemede. Kapatmak için bir tuşa basınız...");
         Console.ReadLine();
         host.Close();
      }
   }
}
```

örnek koda göre **ServiceHost** nesne örneği, ProductManager isimli sınıfı uzak nesne olarak hizmete açmaktadır. **Open** metoduna yapılan çağrıdan sonra, sunucu uygulama istemciden gelecek olan talepleri dinleyecek konuma gelmektedir. **Close** metoduna yapılan çağrı sonrasında ise servis kapatılmaktadır. Elbette **ServiceHost** sınıfının kullanılabilmesi için**System.ServiceModel.dll assembly**' ının sunucu uygulama tarafınada referans edilmesi gerekmektedir.

Peki istemci tarafında neler yapılmalıdır? Herşeyden önce istemci tarafında gerekli ayarların konfigurasyon bazlı olacak şekilde geliştirilmesi önerilmektedir. Bu amaçla istemci tarafına eklenecek olan bir konfigurasyon dosyasının içeriği, örneğe göre aşağıdaki gibi tasarlanabilir.

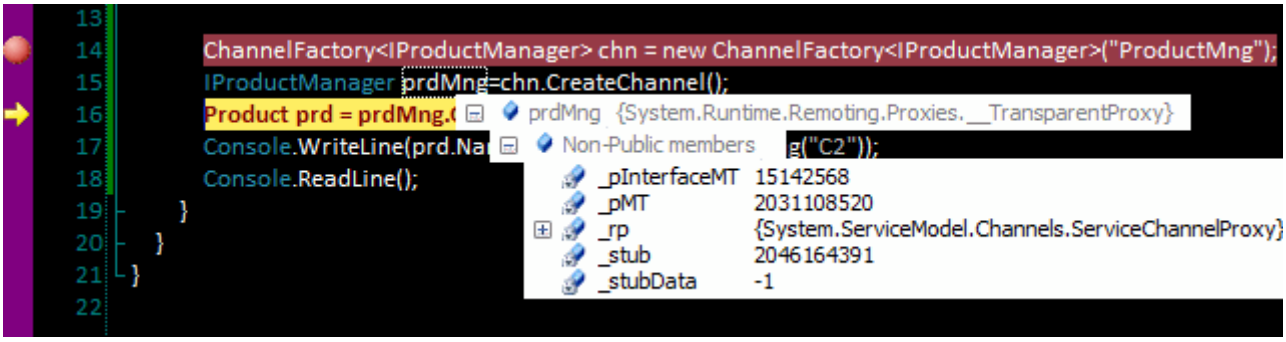
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <client>
         <endpoint name="ProductMng"
address="net.tcp://localhost:4500/ProductMng" binding="netTcpBinding"
contract="AdvLibrary.IProductManager"/>
      </client>
   </system.serviceModel>
</configuration>
```

Konfigurasyon içeriğine bakıldığında dikkat edileceği üzere yine bir **EndPoint** tanımlaması yapılmaktadır. Bu EndPoint aslında istemcinin bağlantı kuracağı servis tarafındaki noktayı tanımlamaktadır. İstemci tarafındaki kod içeriği ise aşağıdaki gibi geliştirilebilir.

```
using System;
using AdvLibrary;
using System.ServiceModel;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         //IProductManager prdMng =
(IProductManager)Activator.GetObject(typeof(IProductManager),
"tcp://localhost:4500/ProductMng.rem");
         //Product prd=prdMng.GetProductInfo(1);

         ChannelFactory<IProductManager> chn = new
ChannelFactory<IProductManager>("ProductMng");
         IProductManager prdMng=chn.CreateChannel();
         Product prd = prdMng.GetProductInfo(1);
         Console.WriteLine(prd.Name+" "+prd.ListPrice.ToString("C2"));
         Console.ReadLine();
      }
   }
}
```

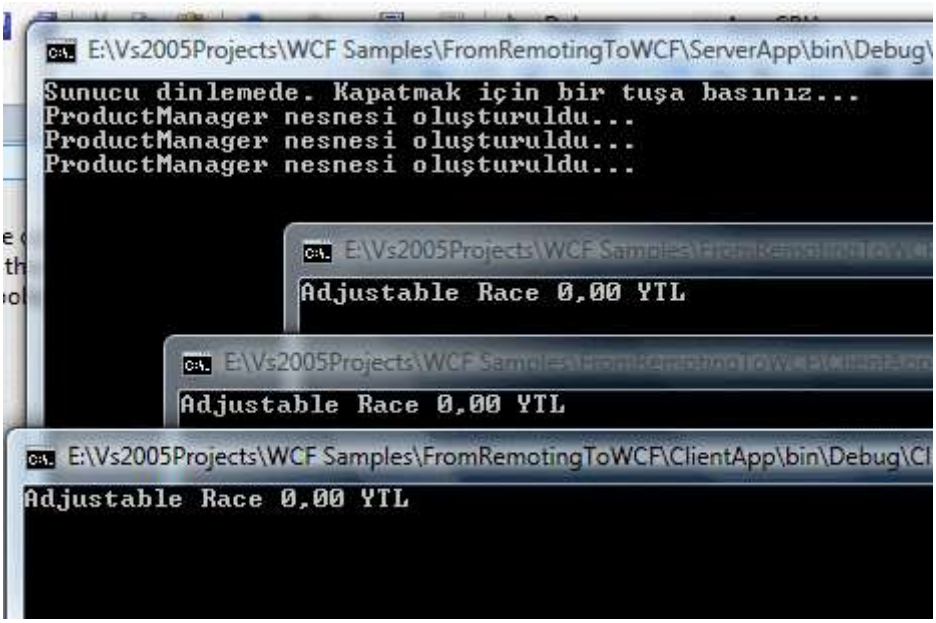
ChannelFactory nesne üretimi için gerekli bilgileri, parametre olarak verilen değere göre konfigurasyon dosyasından almaktadır. Böylece servis tarafındaki hangi **EndPoint** ile konuşabileceğini bilmektedir. **CreateChannel** metodu önceden hazırlanan ayarlara göre, servis tarafı ile konuşacak olan kanal nesnesini örneklemektedir. Açıkçası CreateChannel

metodunun sonucu **.Net Remoting**için kullanılan **Activator.GetObject** metodunun ürettiğine benzerdir. Nitekim aşağıdaki ekran görüntüsünden de anlaşılacağı üzere **CreateChannel** metoduda **Transparent** bir**Proxy** üretmektedir. Bu Proxy, çalışma zamanında servis üzerindeki **EndPoint** ile haberleşmektedir.



çok doğal olarak **proxy** nesnesinin üretilmesinde de kullanılan **ChannelFactory** sınıfı **System.ServiceModel isim alanı(Namespace)** altındadır. Bu nedenle söz konusu isim alanını içeren **assembly**' ın istemci uygulamayada referans edilmesi gerekmektedir.

Artık var olan .Net Remoting uygulaması bu bir kaç adım ile **WCF** formatına çevrilmiştir. Uygulamalar test edildiğinde aşağıdakine benzer sonuçlar alınacaktır.



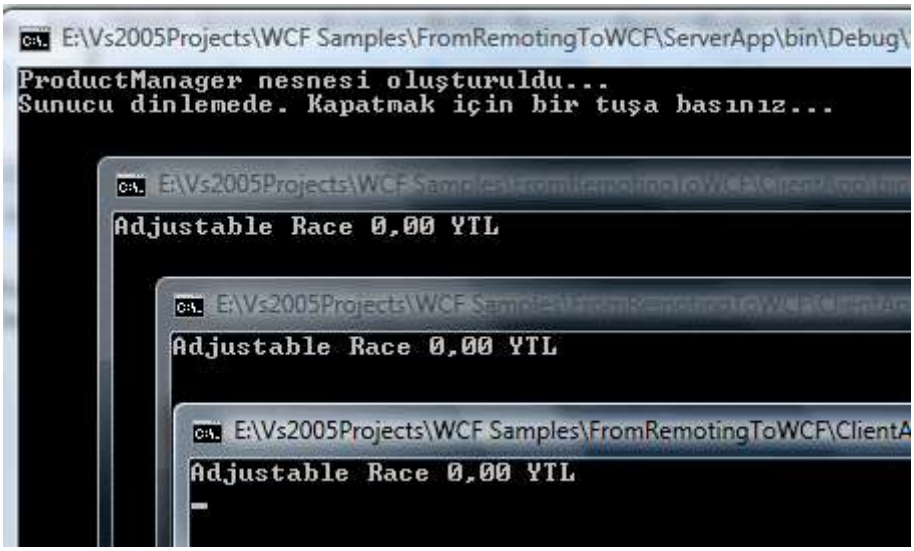
Elbetteki servisin ve operasyonların davranışları **WCF** mimarisi içerisinde çok daha geniş bir şekilde ele alınmaktadır. örnekte bunlara gerek kalmadanda WCF' e geçilebileceği gösterilmektedir. Ancak ele alınması gereken başka noktalarda olabilir.

örneğin her istemci için birer uzak nesne referansı sunucu üzerinde oluşturulmaktadır. Oysaki tasarlanan **.Net Remoting** alt yapısında **Singleton** modeli benimsenmiştir. Bir başka deyişle her istemci talebi(Request) aynı uzak nesne örneği üzerinden

karşılanmaktadır. **WCF** tarafında bu, servis tarafında sunulan uzak nesnenin bir **çalışma zamanı(run-time)** davranışıdır(Behavior). Dolayısıyla bu davranışın bir **nitelik(attribute)** yardımıyla belirlenmesi yada konfigurasyon içeriğinde tanımlanması gerekmektedir. Bu sebepten örnekte yer alan ProductManager sınıfı başında **ServiceBehavior** niteliğinin(attribute) aşağıdaki gibi kullanılması gerekir.

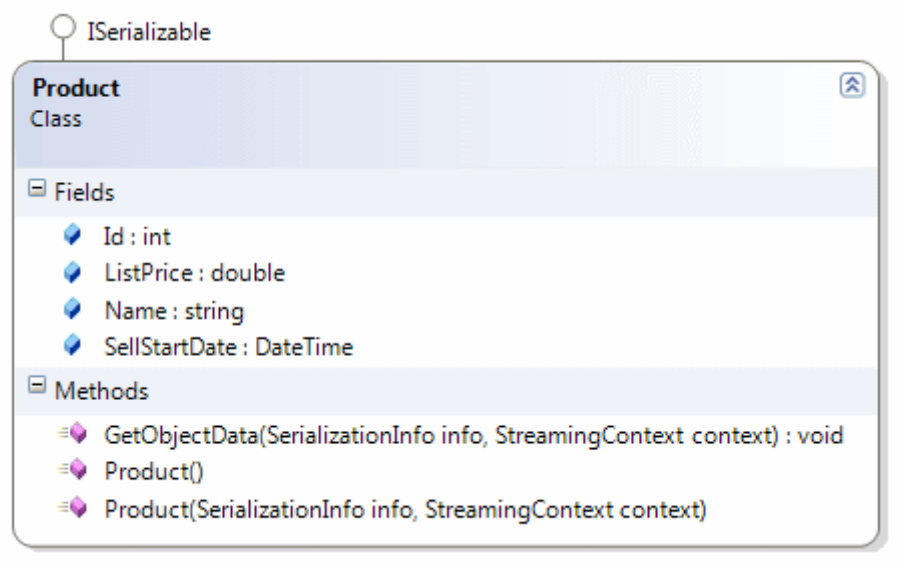
```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]
public class ProductManager:MarshalByRefObject,IProductManager
```

InstanceContextMode enum sabitinin **Single** olarak belirlenen değeri örnekte ele alınan senrayoya uygun olacak şekilde uzak nesne referanslarının oluşturulmasını sağlamaktadır. Bunun sonucunda çalışma zamanındaki durum aşağıdaki gibi olacaktır.



Bunun dışında binary olarak serileştirilebilen Product sınıfı için herhangibir ekstra çalışma yapılmadığı ortadadır. Nitekim böyle bir çalışma yapılmasına gerek kalmamıştır. Fakat **versiyonlama** yapıldığı durumlarda WCF mimarisi **.Net Remoting**' e göre daha efektif bir çözüm sunmaktadır.

Burada **versiyonlamanın(Versioning)** nasıl sorunlar yaşatabileceğini anlamak önemlidir. Söz gelimi Product isimli serileştirilebilir sınıfın sonradan yazılan ikinci bir versiyonu olduğu ve bu versiyonda işin içerisine **SellStartDate** isimli yeni bir alan(Field) katıldığı göz önüne alınsın. Eski Product sınıfını kullanan istemcilerin sorunsuz olarak onunla çalışmaya devam etmelerini, yeni versiyonu kullananların ise yeni eklenen alanı ele alabilmelerini sağlamak için kuvvetle muhtemel tasarımı aşağıdaki gibi değiştirmek gerekecektir.




```
using System;
using System.Runtime.Serialization;

namespace AdvLibrary
{
   [Serializable]
   public class Product:ISerializable
   {
      public int Id;
      public string Name;
      public double ListPrice;
      public DateTime SellStartDate; // Yeni versiyonlar için eklenen alan.

      public Product()
      {

      }

      public Product(SerializationInfo info, StreamingContext context)
      {
         Id = info.GetInt32("Id");
         Name = info.GetString("Name");
         ListPrice = info.GetDouble("ListPrice");
         // Versiyonlama nedeni ile oluşabilecek hataları görmezden gelmek için bir
try...catch
         try
         {
            SellStartDate = info.GetDateTime("SellStartDate");
         }
         catch
```

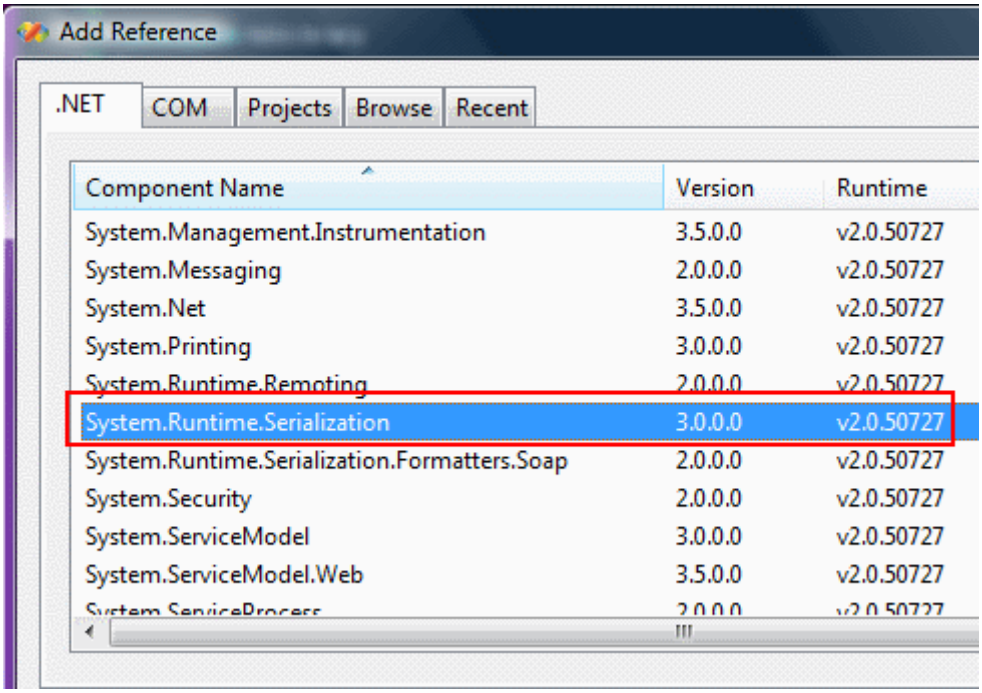
```
        {
        }
    }

    #region ISerializable Members

    public void GetObjectData(SerializationInfo info, StreamingContext context)
    {
        info.AddValue("Id", Id);
        info.AddValue("Name", Name);
        info.AddValue("ListPrice", ListPrice);
        info.AddValue("SellStartDate", SellStartDate); // Yeni versiyon için ekenen
kod satırı
    }

    #endregion
  }
}
```

Burada dikkat edilecek olursa serileştirme işlemleri **ISerializable arayüzünün(Interface)** kullanılmasıyla özelleştirilerek, değerlerin aktarımı kontrollü bir hale getirilmektedir. özelleştirme için **ISerializable** arayüzünün uygulanması haricinde yapıcı metodun(Constructor) aşırı yüklenmiş(Overload) bir versiyonuda ele alınmaktadır. Sorunun detayları çok önemli olmamakla birlikte Windows Communication Foundation içerisinde bu tip versiyonlamalarında kontrol edilebilmesi son derece kolaydır. Tek yapılması gereken aşağıdaki gibi Product sınıfını bir **veri sözleşmesi(Data Contract)** haline getirmektir.

```
using System;
using System.Runtime.Serialization;

namespace AdvLibrary
{
  [DataContract]
  public class Product
  {
    [DataMember(IsRequired=true)] // Mutlaka olmalı
    public int Id;

    [DataMember(IsRequired = true)] // Mutlaka olmalı
    public string Name;

    [DataMember(IsRequired = true)] // Mutlaka olmalı
    public double ListPrice;
```

```
        [DataMember] // Opsiyonel oldu.
        public DateTime SellStartDate; // Yeni versiyonlar için eklenen alan.
    }
}
```

DataMember niteliği(attribute) varsayılan olarak uygulandığı üyeye opsiyonel bir davranış getirmektedir. Bir başka deyişle çalışma zamanında karışa tarafın bu alanı içermeyen bir tipi kullanması halinde hiç bir problem oluşmayacak ve bu alan görmezden gelinecektir. Tam tersine bu alanı kullanan bir versiyon varsa SellStartDate alanıda hesaba katılacaktır. Bu sayede versiyonlamada çok kolay bir şekilde ele alınabilmektedir. Product sınıfının tasarlanmasında dikkat edilmesi gereken önemli bir nokta vardır. Buna göre **DataContract** ve **DataMemberniteliklerinin(attributes)** tanımlandıkları isim alanı(Namespace), **System.Runtime.Serialization.dll** assembly' ı içerisindedir. Dolayısıyla sınıf kütüphanesine bu assembly' ın aşağıdaki ekran görüntüsünde olduğu gibi referans edilmesi gerekmektedir.

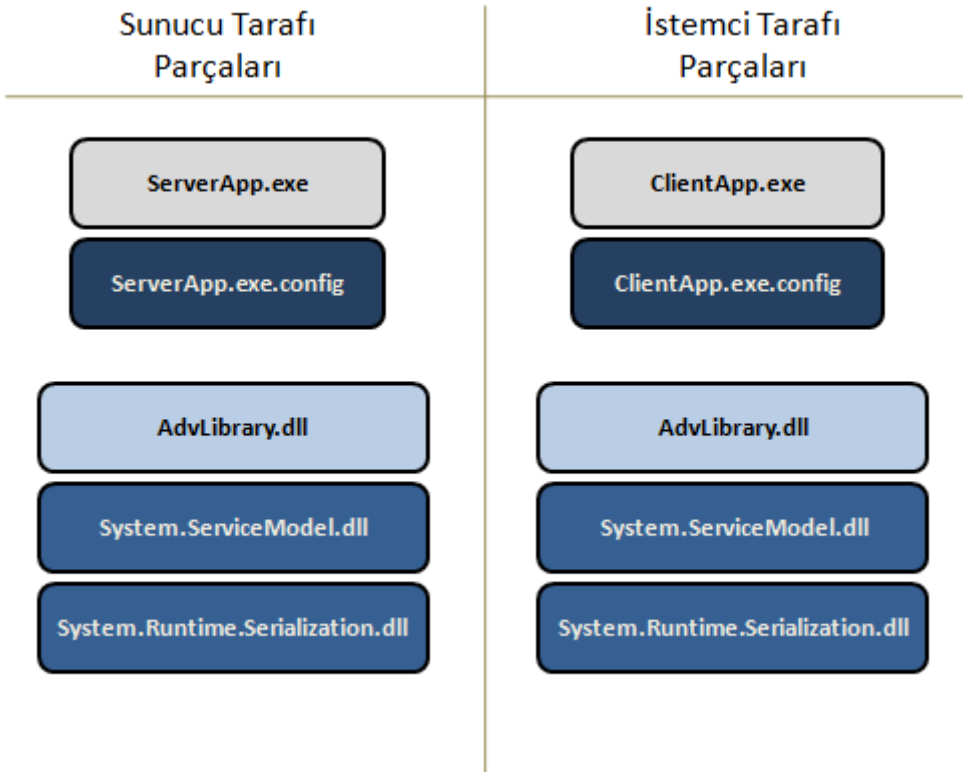


Son olarak ele alınması gereken bir durum daha olabilir. İstemci ve servis tarafının **.Net Framework 2.0** yüklü sistemler olduğu göz önüne alındığında yapılan son değişikliker sonrasında uygulamalar sorunsuz bir şekilde çalışabilecek midir?*(Burada iyimser bir yaklaşım güdülerek .Net Framework 1.1 versiyonu hiç hesaba katılmamıştır.)*

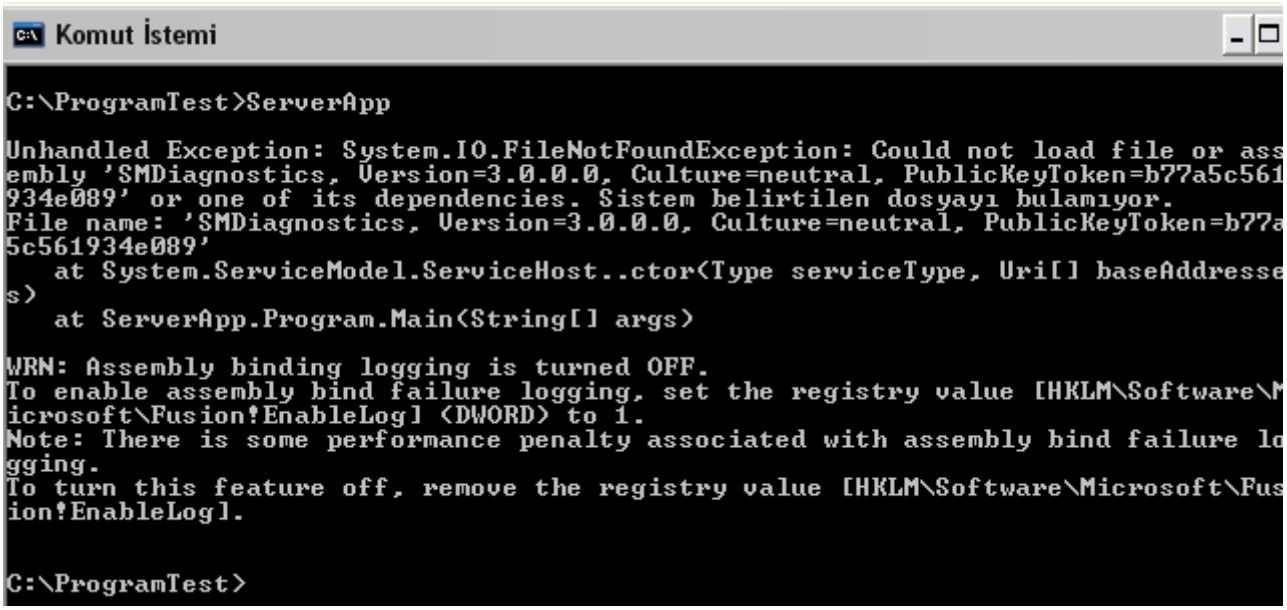
***NOT :** Yukarıdaki referans ekleme kısmında dikkat çeken noktalardan biriside **Framework 3.0** için geliştirilmiş pek çok **assembly**' ın **çalışma zamanında(Run Time)** **v2.0.50727**versiyonuna ihtiyaç duymasıdır. Bir başka deyişle **System.ServiceModel** veya **System.Runtime.Serialization** gibi 3.0 assembly' ları, **.Net Framework 2.0** yüklü bir makinede belleğe yüklenip çalışabilirler. Nitekim ihtiyaçları olan **v2.0.50727** versiyonlu **Common Language Runtime(CLR)**' dır.*

Aslında sistemler üzerinde .Net Framework 3.0' ın yüklü olması sorunun çözümü için yeterlidir. Lakin sadece .Net Framework 2.0 yüklü olan bir sistemde değiştirilen uygulamalar çalıştırılabilirse, bir .Net Framework 3.0 yükleme derdinden kurtulunabilinir. Ancak bunun bir dert olup olmadığıda tartışılması gereken bir vakadır.

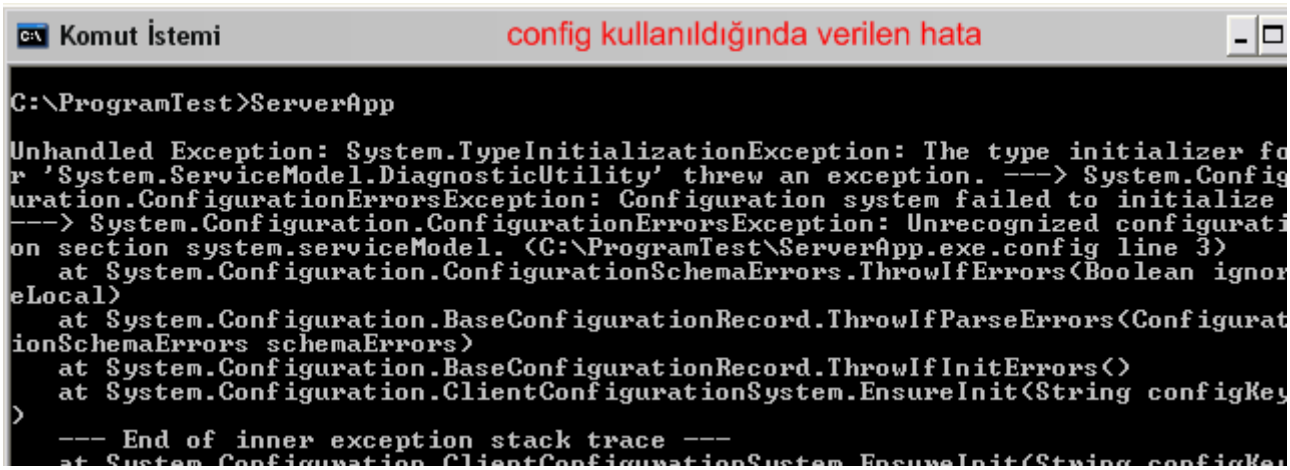
Söz konusu durumu test etmek için servis veya istemci uygulamaları, üzerinde **.Net Framework 2.0** yüklü olan bir sistemde, **System.ServiceModel.dll** ve **System.Runtime.Serialization.dll assembly**' larınıda kopyalayarak çalıştırmak yeterli olacaktır. Başlangıçta servis ve istemci tarafında var olması düşünülen exe, config ve dll dosyaları aşağıdaki gibidir.



Ne yazıkki buradaki dll' lerin hedef bilgisayara(bilgisayarlara) taşınmaları yeterli olmamaktadır. **.Net Framework 2.0** yüklü makinede örnek olarak ServerApp uygulaması çalıştırıldığında aşağıdaki ekran görüntüsünde yer alan hata ile karşılaşılabilir.

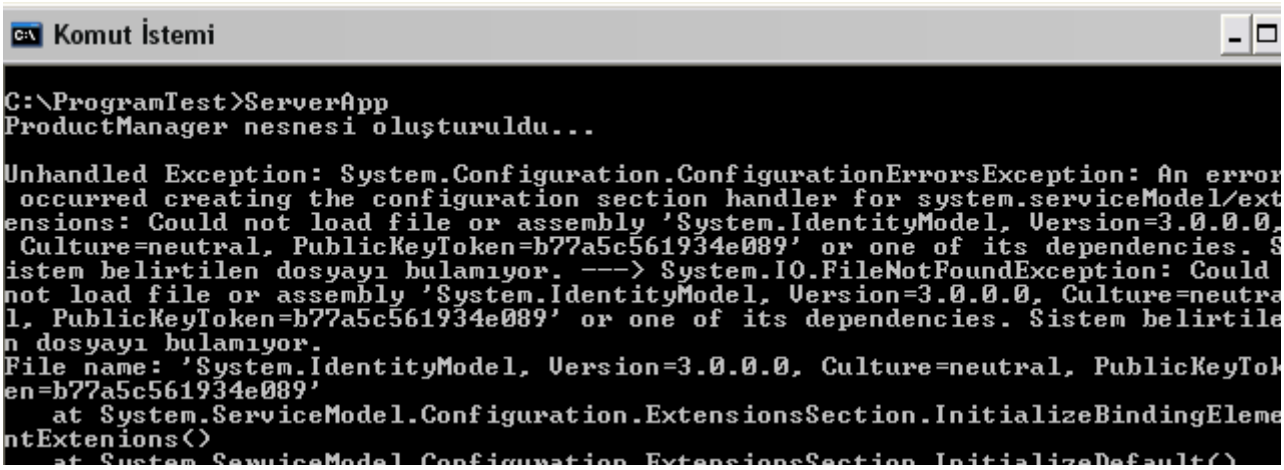


Bunun üzerine **SMDiagnostics.dll** isimli assembly' ında hedef bilgisayarda **ServerApp** ve **ClientApp** program dosyalarının olduğu yere konması gerekir. Söz konusu assembly hedef programların bulunduğu lokasyona taşındıktan sonra bir deneme daha yapılırsa aşağıdakine benzer bir hata mesajı ile daha karşılaşılabilinir.



Bu hatanın oluşmasının en büyük nedeni, programın çalıştığı makine üzerinde **.Net Framework 3.0** yüklü olmadığı için, **machine.config**' de olması gereken bazı ayarların bulunmayışından kaynaklanmaktadır. Bu nedenle **ServiceModelReg** aracının kullanılarak gerekli konfigurasyon bilgilerinin hedef makineye yüklenmesi gerekecektir. Bu sebepten komut satırından aşağıdaki gibi bir çalışma yapılmalıdır.

Bu çalışmanın ardından hedef bilgisayarda yer alan konfigurasyon dosyas için gerekli Handler ve Module yüklemeleri gerçekleştirilecektir. Artık herşey halloldu diye düşünülebilir ama bu seferde ServerApp uygulaması çalıştırıldığında aşağıdakine benzer bir hata mesajı daha alınabilir.



çok doğal olarak **System.IdentityModel.dll**' ininde kopyalanmış olması gerekecektir. Bu hata beraberinde **System.IdentityModel.Selectors.dll** isimli assembly' ında yüklenmesini gerektirmektedir. Her iki **assembly** yüklensede bu kez **doğrulama(Authentication)** ve **yetkilendirme(Authorization)** problemleri oluşmaktadır. Bunların çözümü adına config dosyalarında gerekli düzenlemele yapılabilir. Yinede bu sancılı süreç hemen çözümlenemeyecektir. Yazının ilerleyen kısımlarında konunun daha fazla dağılmaması adına bu yöntemler göz ardı edilmiştir.

Sonuç olarak her ne kadar **.Net Remoting**' den **Windows Communication Foundation** tarafına geçiş yapmak kolay olsada, gerçek çalışma ortamında özellikle **.Net Framework 2.0** yüklü sistemlerde bazı ayarlamaların yapılması gerekmektedir. Bu durum .Net Framework 1.1 yüklü makinelerde tam bir kabus haline gelebilir. Nitekim kopyalanan dll' lerin bazıları **CLR 1.1** versiyonunda çalışmayacaktır. Bu nedenle **.Net Framework 3.0**' ın **Redistruable** paketinin hedef sistemlere yüklenerek çalışmalara devam edilmesi doğru bir davranış olacaktır. Hatta istemciler için bir setup paketinin hazırlanması ve

gerekliliklerden birisi olarak .Net Framework 3.0 seçiminin eklenmesi çok yerinde bir davranıştır. O halde cevaplanması gereken bir soru daha vardır. Bu kadar zahmete girilecekse, sistemin WCF için yeninde yazılması yoluna gidilemez midir? Aslında bu sorunun cevabı daha önceden tasarlanıp kullanılılmakta olan .Net Remoting sistemine bağlıdır. Eğer yazılan çok fazla kod ve birbirlerine bağlı parça var ise, bir iki düzenleme ile WCF' e uyumlu bir sistem oluşturmak son derece kolaydır.

Bu noktalarda vakayı doğru değerlendirip karar vermekte ve gerekirse cevap için profesyonel destek almakta yarar olabilir.Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

FromRemotingToWCF.rar (15,06 kb)

WCF - MTOM ve Stream Kullanarak Veri Aktarımı (2007-11-26T04:33:00)

wcf,mtom,

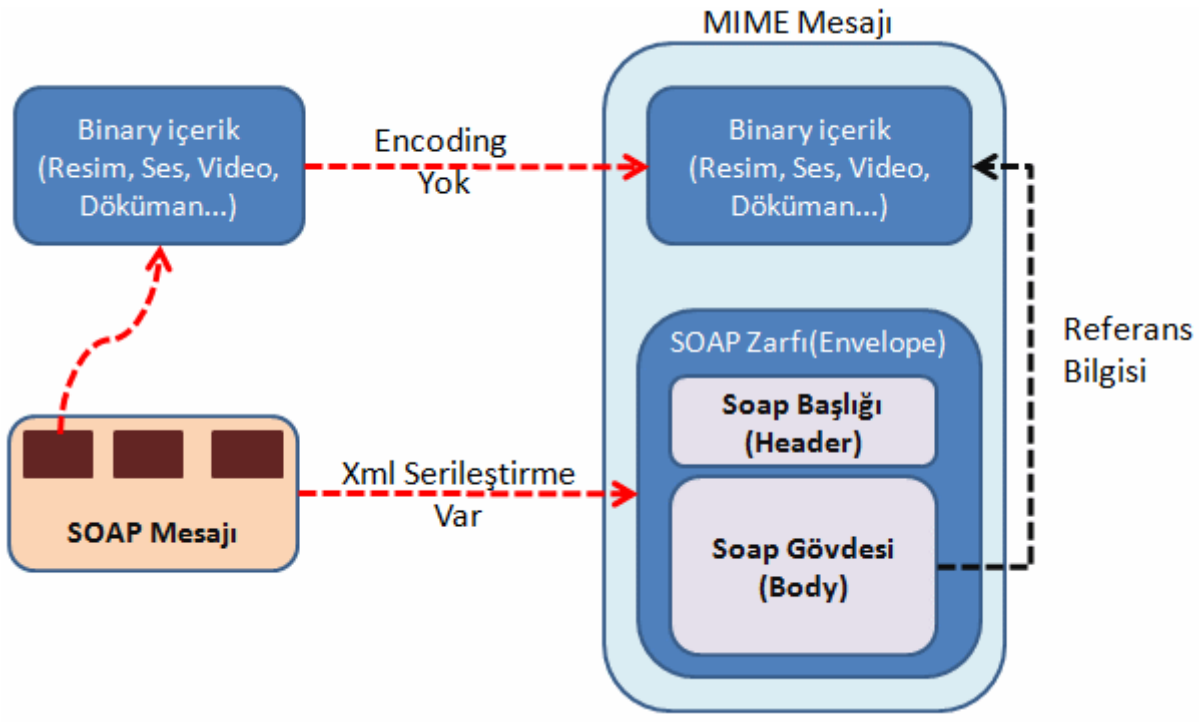
Günümüzde resim,ses, video, doküman formatında kaynakların yoğun olarak kullanıldığı pek çok sistem bulunmaktadır. Söz gelimi **içerik yönetim sistemleri(Content Management Systems)** neredeyse sırf bu tip verilerin kullanılması üzerine kurulmuştur. Resim, ses, video formatındaki veri kaynaklarının oluşturduğu en büyük problem ise boyutlarının söz konusu sistemlerde ne kadar etkin bir şekilde ölçeklenebildiğidir. Büyük boyutlu dosyalar çeşitli amaçlarla kullanılabilirler. örneğin bir şirketin tüm dökümantasyon alt yapısı bu tip büyük büyük dosyalar üzerine kurulu olabilir. Yada üretim sektöründe görev alan bir firmanın teknik çizimleri **ikili(binary)** formatta olacak şekilde veritabanı üzerinde saklanıyor olabilir. Bu tarz verilerin(ister dosyalarda ister veritabanı alanlarında saklanıyor olsunlar) aynı bilgisayarda kullanıldığı uygulamalarda boyutların artması çok fazla problem teşkil etmeyebilir. Ancak işin içerisine **istemci-sunucu(Client-Server)** tabanlı bir ortam girdiğinde boyutların artması özellikle **ağ(Network)** üzerindeki trafiğe olumsuz etkiler yansıtmaktadır. Dolayısıyla farklı makinelerde koşan uygulamaların arasında bu tip büyük boyutlu verilerin aktarımında dikkat edilmesi gereken bazı önemli noktalar vardır.

Servis yönelimli mimari(Service Oriented Architecture) uygulamaların geliştirilmesinde kullanılan bazı platformalar, istemci(Client) ve servisler(Service) arasında resim, ses, video gibi yüksek boyutlu içeriklere sahip olabilecek verilerin taşınması amacıyla çoğunlukla **MTOM(Message Transimision Optimization Mechanism)** standardını kullanırlar. **WC3** tarafından kabul edilmiş bu standarda göre **ikili(binary)** formattaki verilerin aktarılması daha performanslıdır. **Windows Communication Foundation** mimarisi de, mesajların MTOM ile taşınabilmesine olanak sağlamaktadır.

Normal şartlarda **HTTP** üzerinden hizmet veren herhangibir servis içerisindeki metodların parametrik yapısı ve dönüş tipine bakıldığında, istemci ve servis arasında hareket eden**SOAP(Simple Object Access Protocol)** paketlerinin ikili formattaki verileri text

formatına kodlandığı görülür. çok doğal olarak bu **kodlama(encoding)** işlemleri servis tarafına ek bir yük getirmektedir. Aynı yük istemciye ulaşan mesajın içeriğindeki **text** tabanlı bilginin tekrar **çözümlenerek(decoding)** orjinal ikili formattaki haline getirilmesi içinde söz konusudr. Buradaki metodun yapısı gereği sahip olduğu sayısal nitelikteki ikili değerleri text olarak kodlamalası(encoding) çok büyük problem değildir. Ne varki bir resim dosyasının bu şekilde ele alınması halinde boyutun artması, kodlama(encoding) işleminin uzaması ve zaman alması anlamınada gelmektedir. Burada ikili formattaki veri içeriğinin text formatlı olması yerine 1 ve 0' lardan oluşacak şekilde kodlanması göz önüne alınabilir. Ancak bu durumdada milyonlarca 1 ve 0' dan ibaret bir veri yığınının oluşmasıda kaçınılmazdır.

WCF(Windows Communication Foundation) mimarisinde servis tarafındaki metodlar varsayılan olarak **Base64** tabanlı olacak şekilde bir kodlama(encoding) ve çözümleme(decoding) işlemine tabi tutulurlar. Bu işlem sayesinde ikili (binary) formattaki verinin daha az yer tutması sağlanabilir. Ne varki Microsoft kaynaklarının belirttiğine göre, bu kodlama orjinal veri boyutunu yaklaşık olarak **%140**oranında da arttırmaktadır. İşte **MTOM(Message Transmision Optimization Mechanism)** tabanlı mesajlaşma ile veri içeriğinin **kodlama(encoding)** ve **çözümleme(decoding)** işlemine gerek kalmadan taraflar arasında taşınabilmesi sağlanmaktadır. Bunun en büyük nedeni MTOM' un ikili formattaki veri içeriğini orjinal mesaja bir **ek(Attachment)** olacak şekilde taşıtmasıdır. Aşağıdaki şekilde bu durum biraz daha net bir şekilde ifade edilmektedir.



Şekilde dikkat edileceği üzere, binary formattaki içerik bozulmadan bir **MIME(Multipurpose Internet Mail Extension)** paketi içerisine alınmaktadır. Kalan içerik yine bozulmadan text tabanlı olacak şekilde **SOAP Zarfı(Envelope)** halinde **MIME** Mesajı içerisine aktarılır. **SOAP gövdesinde(Body)** artık MIME mesajına ilave edilen binary içeriğin referansı tutulmaktadır. Böylece servis veya

istemi tarafında, binary içeriğin text formatına kodlanması(ve tam tersinin yapılması) işlemleri ortadan kalkmaktadır. Buda çok doğal olarak mesajların hazırlanma, gönderme ve işlenme sürelerinin inanılmaz derecede azalması anlamına gelmektedir.

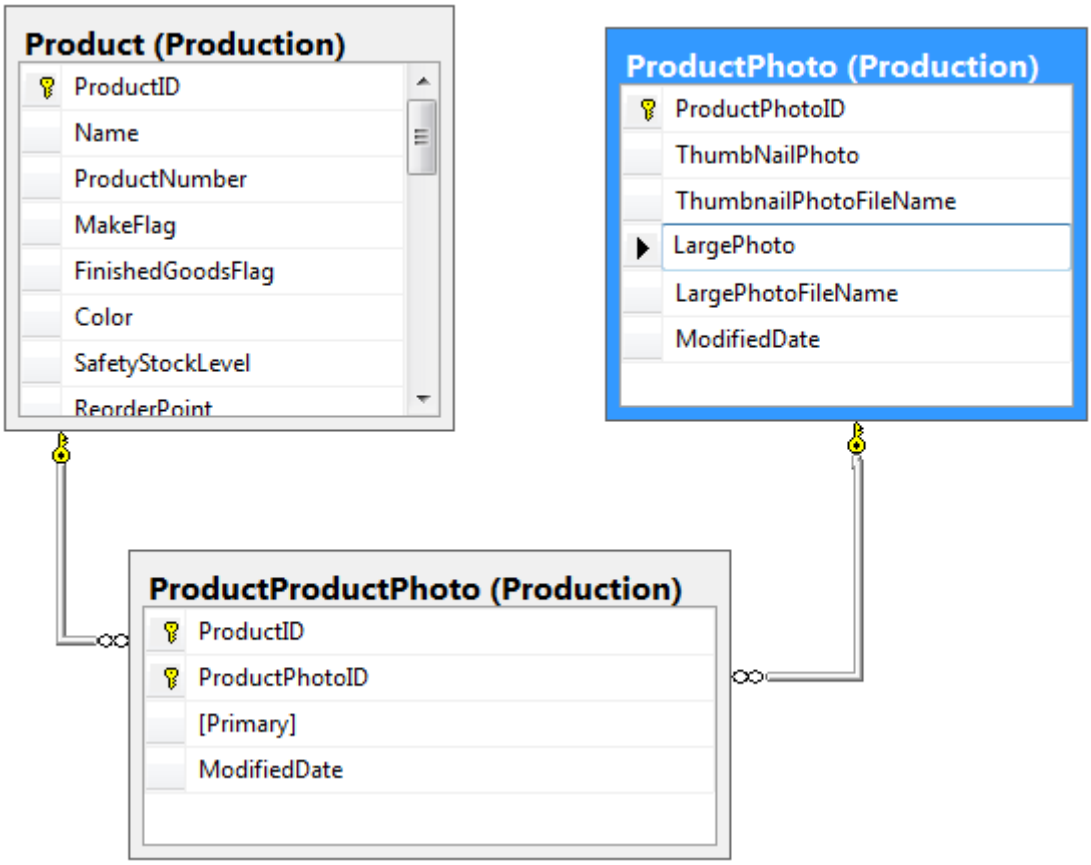
NOT : *MTOM(Message Transmision Optimization Mechanism) tabanlı mesajların güvenliğinde* ***imzalar(signature)*** *kullanılır. WCF bu imzaları kontrol ederek alınan veya gönderilen MIME içeriğindeki eklerin(attachments) bozulup bozulmadığını anlayabilir ve buna göre uygun çalışma zamanı istisnalarını fırlatabilir.*

Windows Communication Foundation mimarisinde **HTTP** bazlı olan bağlayıcı tiplerin tamamı **MTOM** tipinde mesajlaşmayı desteklemektedir. Ne varki **TCP**, **MSMQ** gibi protokoller üzerinde çalışan **bağlayıcı tiplerin(Binding Type)** bu tip bir desteği yoktur. Bunun en büyük sebebi ise, bu protokollerin **ikili(binary)** formattaki veriler için kendi standartlarını kullanıyor olmalarıdır. Ancak daha öncedende değinildiği gibi, **özel bağlayıcı tipler(Custom Binding Types)** geliştirilerek **TCP** gibi bir protokol üzerinde **MTOM** kullanımı teorik olarak sağlanabilmektedir. Aşağıdaki tabloda MTOM tipinde mesajlaşmaya destek veren(vermeyen) bağlayıcı tipler işaret edilmektedir.

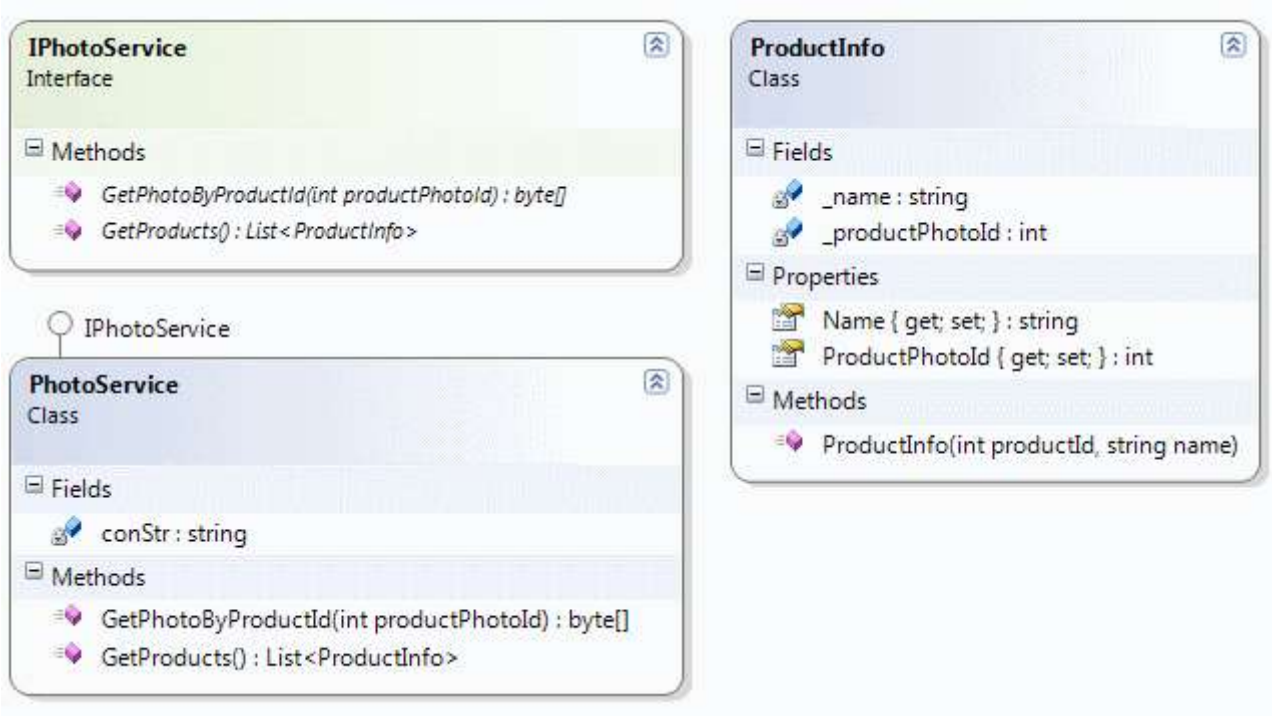
Bağlayıcı Tip (Binding Types)	**Mesaj Kodlama/çözümleme Tipleri (Message Encoding/Decoding Types)**
BasicHttpBinding	**Text / MTOM**
NetTcpBinding	Binary
NetPeerTcpBinding	Binary
NetNamedPipeBinding	Binary
WSHttpBinding	**Text / MTOM**
WSFederationBinding	**Text / MTOM**
NetMsmqBinding	Binary
MsmqIntegrationBinding	Binary
WSDualHttpBinding	**Text / MTOM**

Konuyu daha iyi anlayabilmek için örnek bir senaryo üzerinden hareket etmekte fayda vardır. Bu amaçla **AdventureWorks** veritabanında yer alan ürünlere ait fotoğrafların tutulduğu**ProductPhoto** tablosundan yararlanılabilir. Söz konusu tabloda **varBinary(MAX)** SQL tipinden **LargePhoto** isimli bir alan yer almaktadır. İlerleyen örnekte LargePhoto isimli alanın içeriğinin MTOM ve Stream bazlı olacak şekilde istemci tarafına aktarılması üzerinde durulacaktır. Senaryoda göz önüne alınacak tablolar aşağıdaki **veritabanı diagramında(Database Diagram)**olduğu gibidir. öncelikli olarak en azından istemciye ürün adı ve fotoğraf numarası bilgilerinin gönderilmesinde

yarar vardır. Sonrasında istemcinin seçtiği ürüne ait fotoğrafın binary içeriği servis tarafından geriye doğru aktarılacaktır.



Servis tarafındaki uygulamanın istemciye ürün listesini ve seçtiği ürüne ait resmi veren fonksiyonellikler içerdiği düşünülebilir. Bu fonksiyonellikleri içeren servis sınıfı ve üyeleri her zamanki gibi bir **WCF Servis Kütüphanesi(WCF Servis Library)** olacak şekilde aşağıdaki gibi tasarlanabilir. Söz konusu kütüphane içerisinde yer alacak temel tiplerin sınıf diagramındaki(Class Diagram) görüntüsü ise aşağıdaki gibidir.



örnekte ürünlere ait fotoğraf bilgilerinin istemciye taşınması için **ProductInfo** isimli bir sınıftan yararlanılmaktadır. Bu sınıfın içeriği aşağıdaki gibidir.

```
using System;
using System.Collections.Generic;
using System.Runtime.Serialization;

namespace ProductPhotoServiceLibrary
{
   [DataContract]
   public class ProductInfo
   {
      private int _productPhotoId;
      private string _name;

      [DataMember]
      public string Name
      {
         get { return _name; }
         set { _name = value; }
      }

      [DataMember]
      public int ProductPhotoId
      {
         get { return _productPhotoId; }
         set { _productPhotoId = value; }
```

```
        }

        public ProductInfo(int productId, string name)
        {
            ProductPhotoId = productId;
            Name = name;
        }
    }
}
```

Bu sınıfta dikkat edilmesi gereken nokta, bir **veri sözleşmesi(Data Contract)** tanımlıyor olmasıdır. Bu nedenle **sınıf(Class)** **DataContract**, **özellikleri(Properties)** ise **DataMember**nitelikleri ile imzalanmıştır. **Servis sözleşmesi(Service Contract)** içeriği aşağıdaki gibidir.

```
using System;
using System.ServiceModel;
using System.Collections.Generic;

namespace ProductPhotoServiceLibrary
{
    [ServiceContract(Name="Photo
Service",Namespace="http://www.bsenyurt.com/ProductPhotoService")]
    public interface IPhotoService
    {
        [OperationContract(IsInitiating=true)]
        List<ProductInfo> GetProducts();

        [OperationContract]
        byte[] GetPhotoByProductId(int productPhotoId);
    }
}
```

Arayüzün uygulandığı **PhotoService** sınıfının içeriği ise aşağıdaki gibidir**.**

```
using System;
using System.Data.SqlClient;
using System.Collections.Generic;

namespace ProductPhotoServiceLibrary
{
    public class PhotoService : IPhotoService
    {
        string conStr = "data source=.;database=AdventureWorks;integrated security=SSPI";
```

```
#region IPhotoService Members

// İstemci için gerekli olan ürün adları ve fotoğraf numaraları, generic bir List koleksiyonu ile geriye döndürülmektedir.
public List<ProductInfo> GetProducts()
{
   // Generic List koleksiyonu oluşturulur
   List<ProductInfo> infos = new List<ProductInfo>();

   using (SqlConnection conn = new SqlConnection(conStr))
   {
      // Sql sorgusunda Join kullanılmasının sebebi ProductPhotoId değerleri ve Name alanlarının değerlerinin elde edilmesidir.
      using (SqlCommand cmd = new SqlCommand("Select PP.ProductPhotoId,[Name] From Production.Product P Join Production.ProductProductPhoto PP on P.ProductId=PP.ProductID", conn))
      {
         conn.Open();
         SqlDataReader reader = cmd.ExecuteReader();
         while (reader.Read())
         {
            // Elde edilen her bir satır için ProductInfo sınıfına ait bir nesne örneklenip generic koleksiyona eklenir.
            infos.Add(new ProductInfo(Convert.ToInt16(reader[0]), reader[1].ToString()));
         }
         reader.Close();
      }
   }
   return infos;
}

// Belirli bir fotoğraf numarasına sahip olan LargePhoto alanının içeriği byte dizisi olacak şekilde geri döndürülür.
public byte[] GetPhotoByProductId(int productPhotoId)
{
   byte[] photoContent = null;

   using (SqlConnection conn = new SqlConnection(conStr))
   {
      using (SqlCommand cmd = new SqlCommand("Select LargePhoto From Production.ProductPhoto Where ProductPhotoId=@PId", conn))
      {
         cmd.Parameters.AddWithValue("@PId", productPhotoId);
         conn.Open();
```

```
                SqlDataReader reader = cmd.ExecuteReader();
                // Burada GetSqlBytes metodu ilgili alanın içeriğinin binary olarak byte
dizisine aktarılmasında yardımcı rol oynamaktadır.
                if (reader.Read())
                   photoContent = reader.GetSqlBytes(0).Value;
                reader.Close();
             }
          }

          return photoContent;
       }

       #endregion
    }
}
```

PhotoService isimli sınıf şu an için iki adet metoda sahiptir. **GetProducts** isimli metod basit olarak geriye **ProductInfo** tipinden nesne örneklerinden oluşan **generic** bir **List<>** koleksiyonu döndürmektedir. **GetPhotoByProductId** metodu geriye **byte[]** tipinden bir dizi döndürmektedir. Bu dizi tahmin edileceği üzere tabloda **varbinary(MAX)** tipinden olan **LargePhoto** isimli alanın içeriğini taşımaktadır.

***NOT : Binary** dosyanın boyutunun çok büyük olması halinde parçalı olacak şekilde veri içeriğinin toplanması göz önüne alınabilir. Yapılması gereken servis tarafındaki fonksiyonellik içerisinde resim içeriğini veritabanı tablosunda parçalı olacak şekilde okuyup toparlamaktır.*

Şimdilik servis tarafı **MTOM(Message Transmision Optimization Mechanism)** formatı yerine **Text** formatında mesaj kodlaması yapmaktadır. Olayın daha net analiz edilmesi amacıylada gerekli**Diagnostics** seçenekleri aktif hale getirilmektedir. Servis tarafının içeriği basit olarak aşağıdaki gibidir. *(Servis IIS üzerinde host edilecek şekilde tasarlanmaktadır.)*

Service.svc;

```
<%@ ServiceHost Language="C#"
Debug="true" Service="ProductPhotoServiceLibrary.PhotoService" %>
```

Servis(Service) tarafında mesajlaşma ile ilişkili olan içeriği izleyebilmek adına **Diagnostics** özellikleri açılmıştır. Buna göre **web.config** dosyasının içeriği aşağıdaki gibidir.

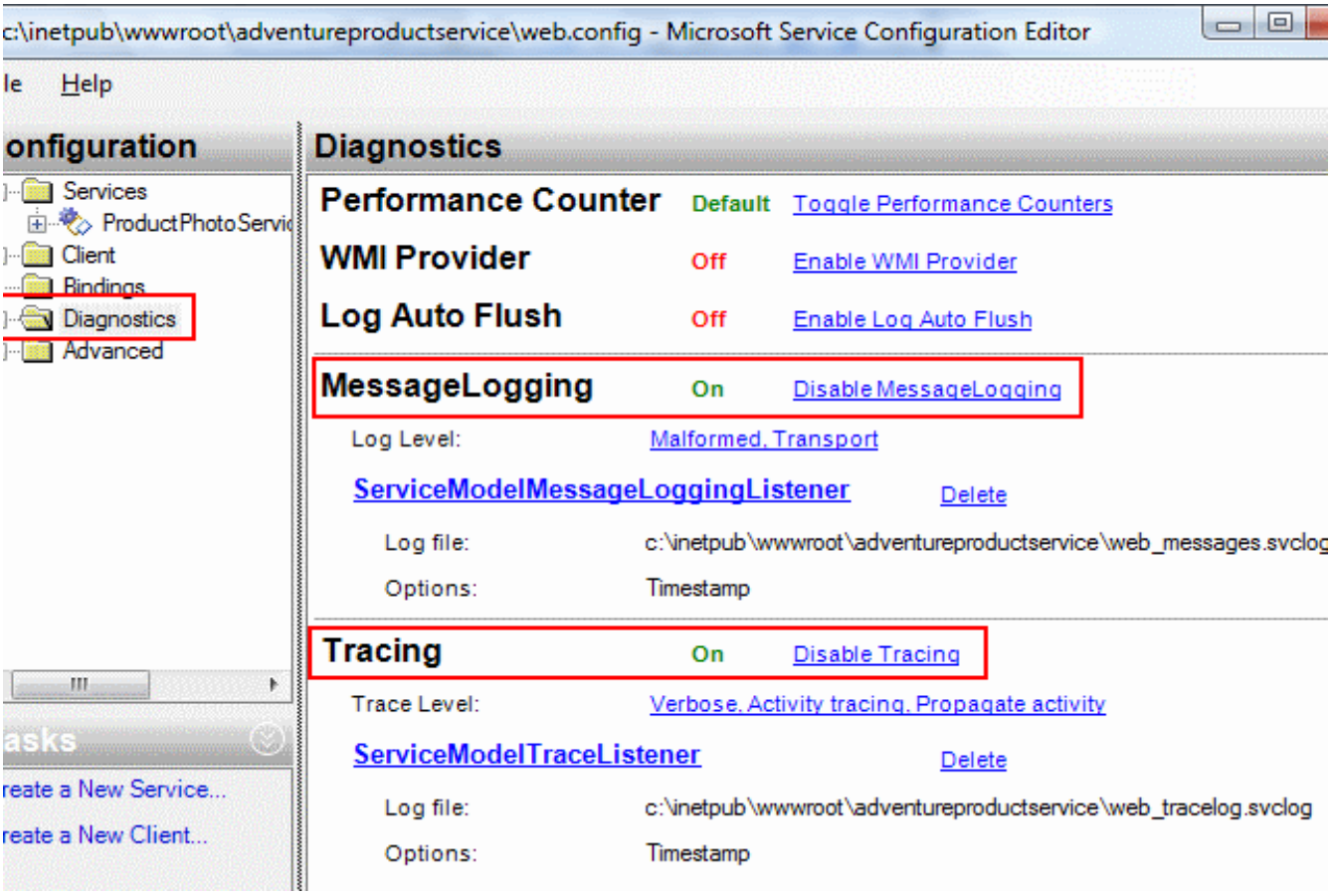
web.config;

```
<?xml version="1.0"?>
<configuration>
  <system.diagnostics>
    <sources>
      <source name="System.ServiceModel.MessageLogging"
switchValue="Verbose,ActivityTracing">
        <listeners>
          <add type="System.Diagnostics.DefaultTraceListener" name="Default">
            <filter type="" />
          </add>
          <add name="ServiceModelMessageLoggingListener">
            <filter type="" />
          </add>
        </listeners>
      </source>
      <source name="System.ServiceModel"
switchValue="Verbose,ActivityTracing" propagateActivity="true">
        <listeners>
          <add type="System.Diagnostics.DefaultTraceListener" name="Default">
            <filter type="" />
          </add>
          <add name="ServiceModelTraceListener">
            <filter type="" />
          </add>
        </listeners>
      </source>
    </sources>
    <sharedListeners>
      <add initializeData="C:\inetpub\wwwroot\AdventureProductService\web_mes
sages.svclog" type="System.Diagnostics.XmlWriterTraceListener, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
name="ServiceModelMessageLoggingListener" traceOutputOptions="Timestamp">
        <filter type="" />
      </add>
      <add initializeData="C:\inetpub\wwwroot\AdventureProductService\web_trac
elog.svclog" type="System.Diagnostics.XmlWriterTraceListener, System,
Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
name="ServiceModelTraceListener" traceOutputOptions="Timestamp">
        <filter type="" />
      </add>
    </sharedListeners>
  </system.diagnostics>
  <appSettings/>
  <connectionStrings/>
  <system.web>
```

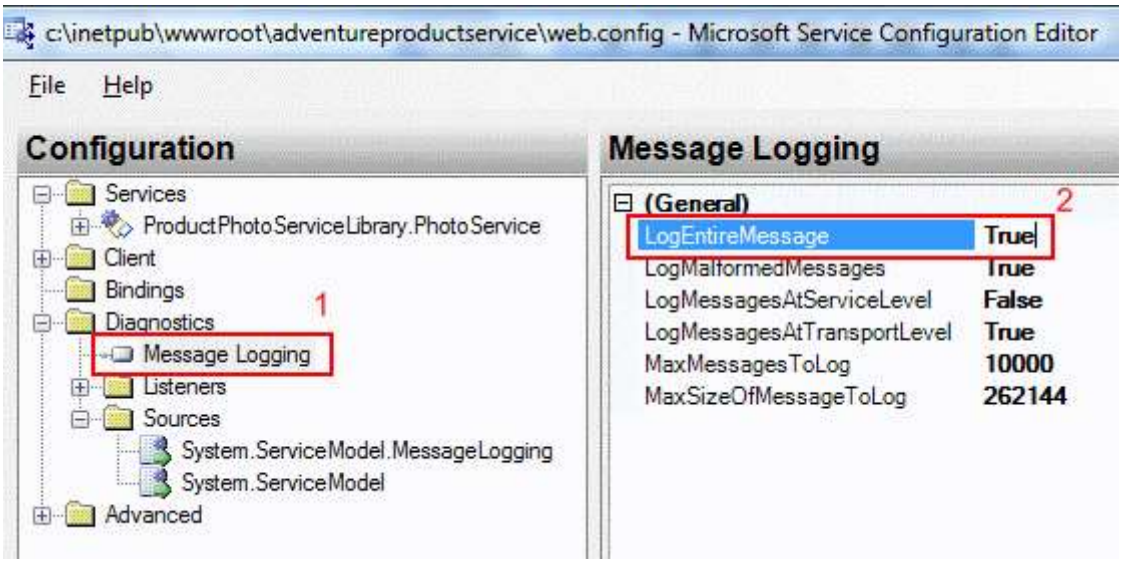
```
    <compilation debug="true">
    </compilation>
    <authentication mode="Windows" />
  </system.web>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="PhotoServiceBehavior">
          <serviceMetadata httpGetEnabled="true" />
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <diagnostics>
      <messageLogging logEntireMessage="true" logMalformedMessages="true"
logMessagesAtTransportLevel="true" />
    </diagnostics>
    <services>
      <service behaviorConfiguration="PhotoServiceBehavior"
name="ProductPhotoServiceLibrary.PhotoService">
        <endpoint address="http://localhost/AdventureProductService/Service.svc"
binding="basicHttpBinding" bindingConfiguration=""
name="ProductPhotoHttpEndPoint" contract="ProductPhotoServiceLibrary.IPhotoService"
/>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Servis tarafı **basicHttpBinding** bağlayıcı tipini kullanan bir **EndPoint** sunmaktadır. Bununla birlikte istemcilerin **HTTP** üzerinden **proxy** sınıflarını elde edebilmeleri için **metaData Publishing**etkin kılınmıştır. Mesajların içeriğinin daha detaylı bir şekilde analiz edilebilmesi amacıyla **Tracing** ve **Message Logging** özellikleri aktif hale getirilmiştir. Bu ayarlar elbetteki **Microsoft Service Configuration Editor** yardımıyla belirlenmek istenirse aşağıdaki adımlar takip edilebilir.

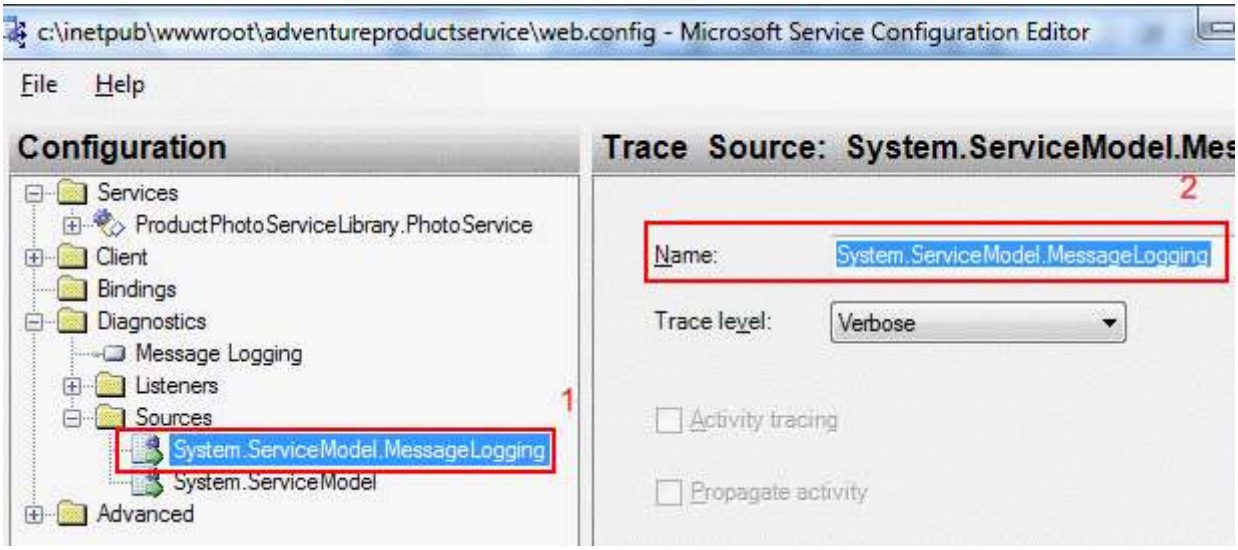
öncelikli olarak Enable MessageLoging ve Enable Tracing tıklanarak aktif hale getirilir.

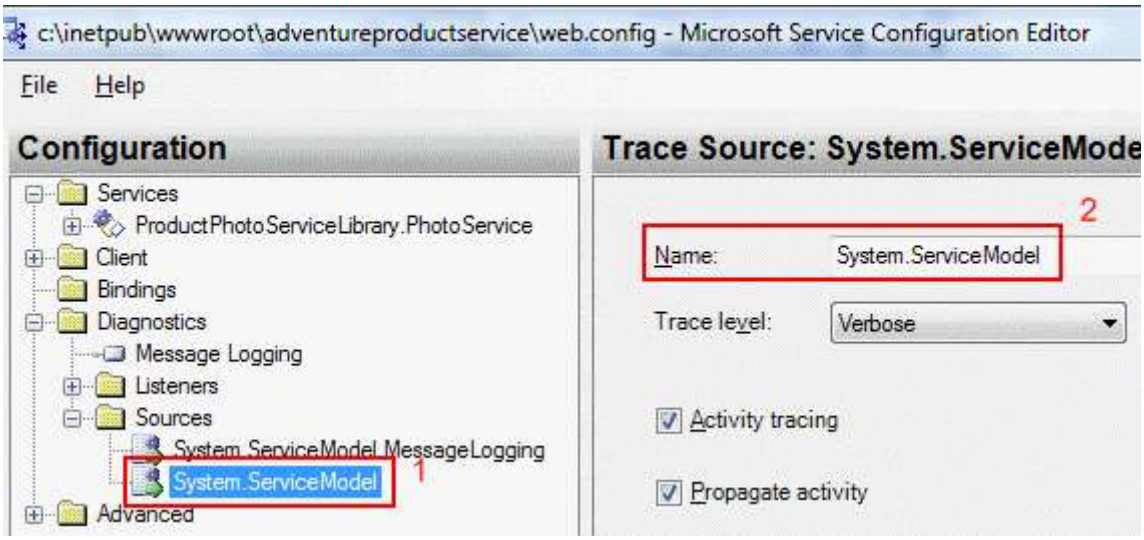


Sonrasında ise **Diagnostics** altındaki **Message Logging** klasörüne tıklandıktan sonra özellikler penceresinden **LogEntireMessage** değeri **true** olarak ayarlanmalıdır.

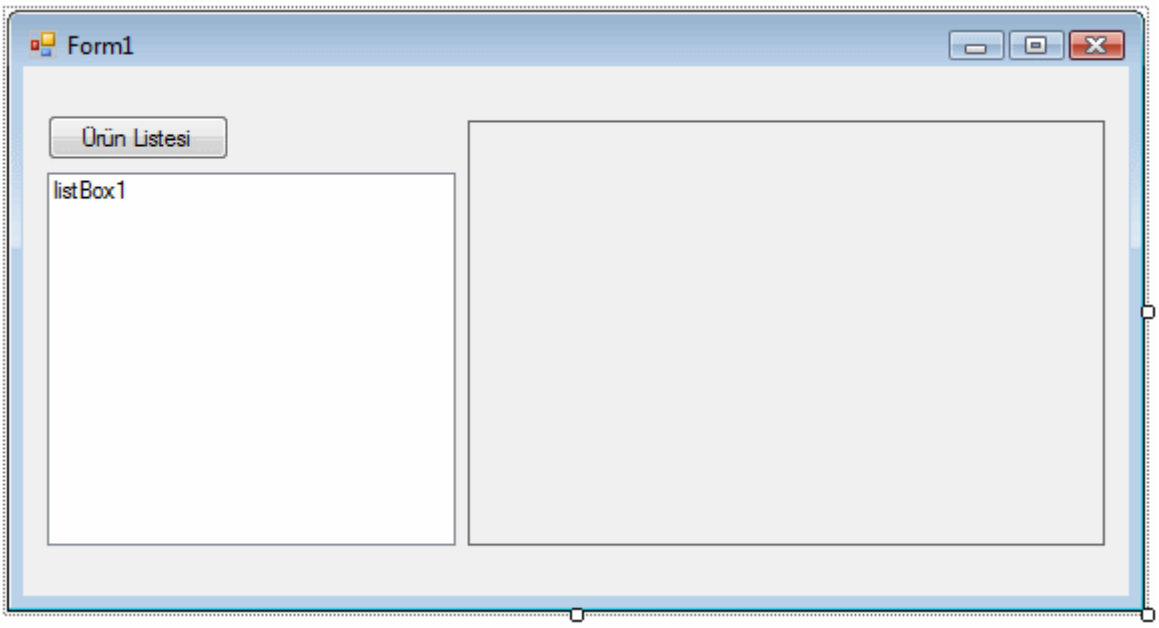


Sources klasörü altında yer alan **System.ServiceModel.MessageLogging** ve **System.ServiceModel** elementlerinin **Tr ace Level** özelliklerine **Verbose** değeri atanmalıdır. Böylece mesaj aktivitelerin en ince ayrıntısına kadar izlenmesi olanaklı hale gelecektir.





Gelelim istemci tarafına. İstemci program basit bir **Windows** uygulaması olarak tasarlanabilir. Programın arayüzü ve kodlarının içeriği aşağıdaki gibidir.



*(Servis tarafına ait Proxy sınıfının oluşturulması için **Add Service Reference** seçeneği kullanılmaldırı.)*

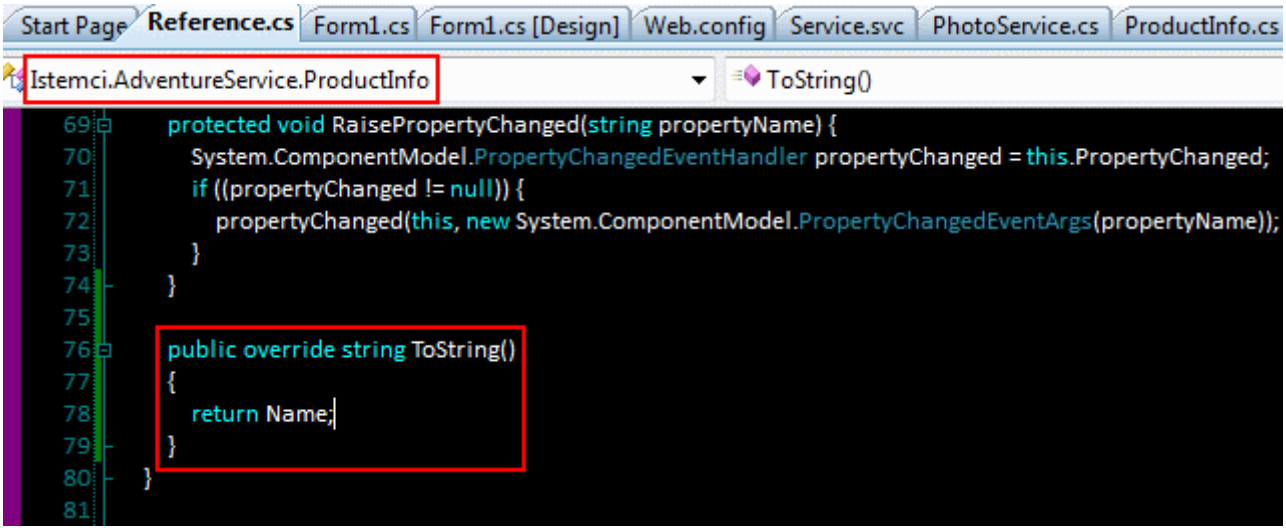
Add Service Reference sonrası otomatik olarak bir **App.config** dosyası oluşturulacak ve içerisine başlangıç değerleri atanacaktır.

Windows uygulamasına ait kodlar ise aşağıdaki gibidir.

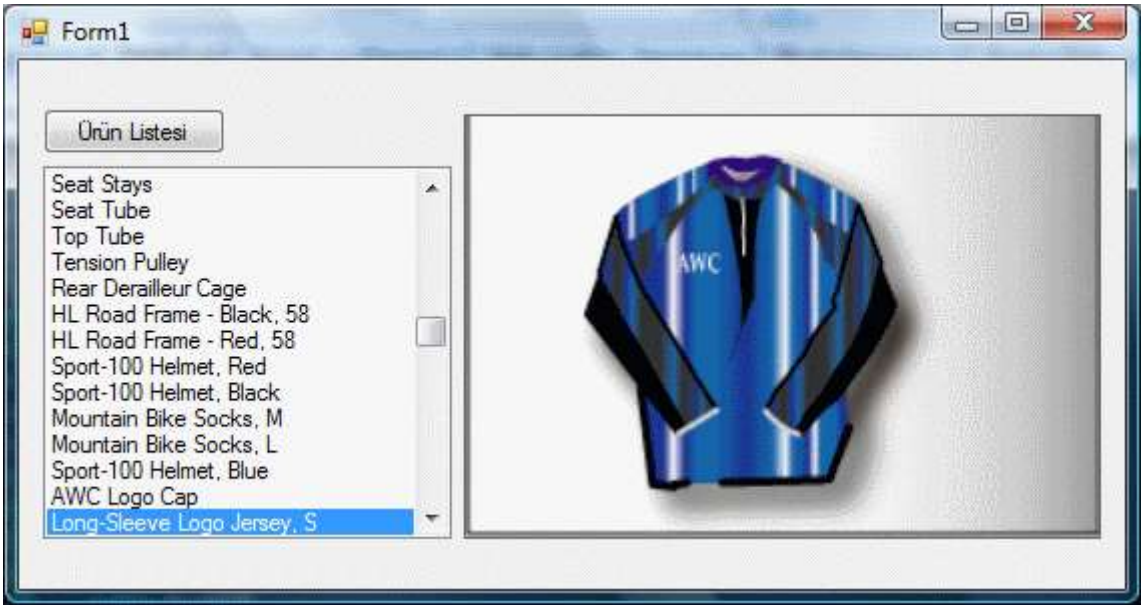
```
using System;
using System.IO;
using System.Text;
using System.Data;
using System.Drawing;
using System.Windows.Forms;
using System.ComponentModel;
using Istemci.AdventureService;
using System.Collections.Generic;

namespace Istemci
{
   public partial class Form1 : Form
   {
      PhotoServiceClient client = null;

      public Form1()
      {
         InitializeComponent();
         client = new PhotoServiceClient();
      }
```

```
        private void button1_Click(object sender, EventArgs e)
        {
            // Servis tarafındaki GetProducts metodu ile ProductInfo tipinden dizi elde edilir ve
her bir elemanı listBox1 isimli ListBox kontrolüne eklenir.
            ProductInfo[] infos=client.GetProducts();
            foreach (ProductInfo info in infos)
                listBox1.Items.Add(info);
        }

        private void listBox1_SelectedIndexChanged(object sender, EventArgs e)
        {
            short photoId = 0;
            // Seçilen ListBox elemanı ProductInfo tipine dönüştürüldükten sonra
ProductPhotoId özelliğinin değeri elde edilir.
            Int16.TryParse(((ProductInfo)listBox1.SelectedItem).ProductPhotoId.ToStrin
g(), out photoId);
            // GetPhotoByProductId metoduna photoId değeri aktarılarak byte[] dizisi elde
edilir ve bir MemoryStream nesnesi örneklenir.
            MemoryStream stream=new
MemoryStream(client.GetPhotoByProductId(photoId));
            // MemoryStream örneğinden yararlanılarak Image nesnesi oluşturulur ve
PictureBox kontrolünün Image özelliğine atanır.
            pictureBox1.Image=Image.FromStream(stream);
        }
    }
}
```

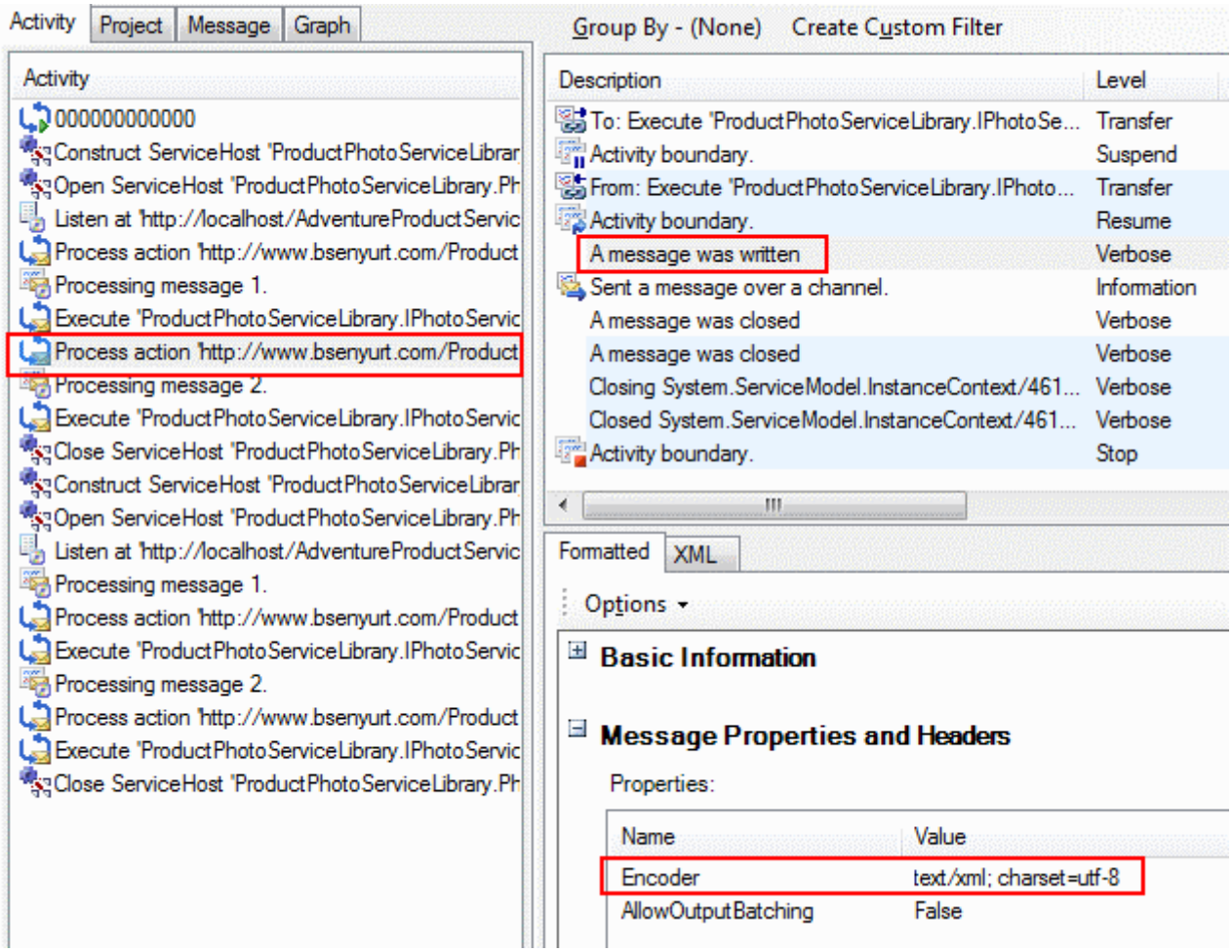
Burada dikkat edilmesi gereken birkaç nokta vardır. **GetProducts** metodu geriye **List<ProductInfo>** tipinden bir generic koleksiyon yerine **ProductInfo** tipinden bir dizi döndürmektedir. Diğer taraftan istemci bir Windows uygulaması olduğundan ComboBox kontrolü içerisine dizi doğrudan bağlanamaz. Bir başka deyişle **DataSource** özelliğini burada kullanmak yeterli olmayacaktır. Bu nedenle ProductInfo tipinden dizi içerisindeki elemanların tek tek dolaşılması ve öğe olarak eklenmesi gerekmektedir. Ancak bu durumdada **ListBox** içerisinde ürün adlarının görülebilmesi için**ToString** metodunun ezilmiş olması şarttır. Bilindiği gibi servis tarafındaki geliştirici tanımlı tiplere ait metodlar istemci tarafında serileştirilmemektedir. O nedenle, ToString metodunun ezilmiş hali sadece bu örnekte yardımcı olması açısından istemci tarafındaki uygulamada bilinçli olarak aşağıdaki gibi ezilmektedir. Bir gerçek hayat senaryosunda bu çok kontrol edilebilir bir durum olamayabilir. Nitekim güncelleştirme ve ölçeklendirme zorlaşmaktadır.



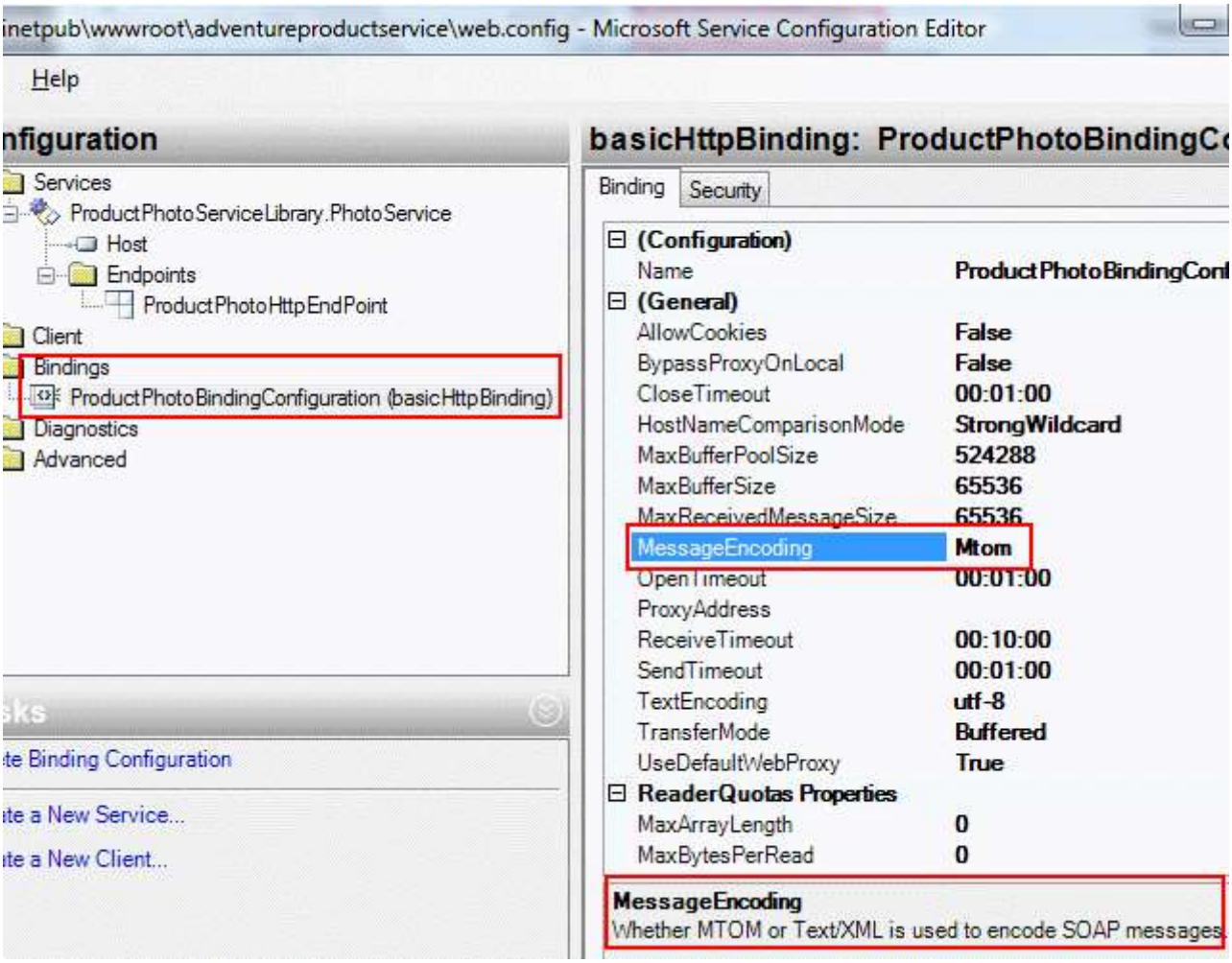
Bu değişikliğin arkasından **ListBox** üzerinden **SelectedItem** özelliği ile elde edilen **Object** referansı **ProductInfo** tipine dönüştürülüp **ProductPhotoId** özelliğinin değeri elde edilebilir. Dolayısıyla örnek çalıştırıldığında aşağıdaki ekran görüntüsündekine benzer sonuçlar elde edilecektir.



Gelelim asıl konumuza. Bakalım log dosyalarında ne gibi sonuçlara varacağız. öncelikli olarak **Service Trace Viewer** yardımıyla **web_tracelog.svc** dosyasının açılması gerekmektedir. Bu ana kadar yapılanların tek amacı söz konusu dosya içerisinde, istemciye gönderilen mesaj formatının text tabanlı olduğunun ispat edilmesidir. İlk olarak **Activity** kısmından **Process action "http://www.bsenyurt.com/ProductPhotoService/Photo_x0020_Service/GetPhotoByProductId** bölümü seçilir. Bu işlemin ardından **A message was written Description** bölümüne bakılırsa **Encoder** isimli özelliğin değerinin aşağıdaki ekran görüntüsünde olduğu gibi **text/xml; charset=utf-8** olduğu görülür.

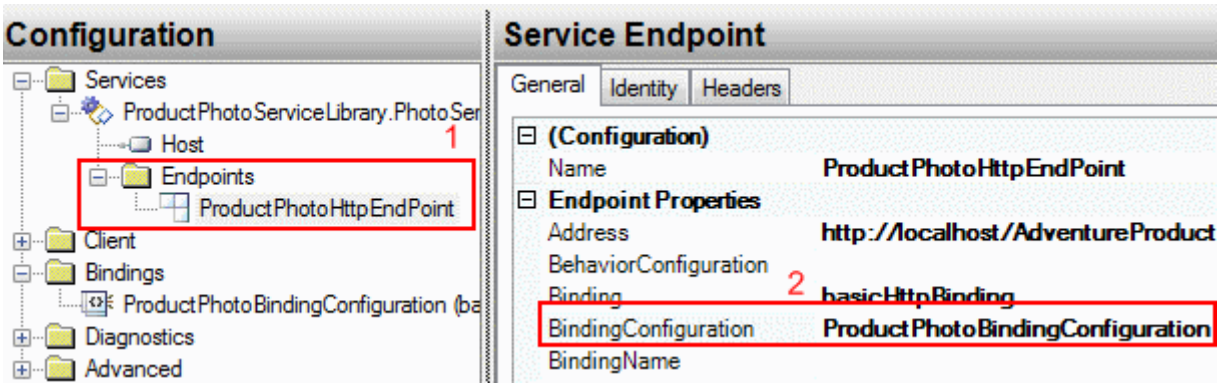


Bu aslında istemciye döndürülen **byte[]** dizisinin **text** tabanlı olacak şekilde **XML** içeriğine kodlandığının bir göstergesidir. Servis tarafından **MTOM**' a uygun olacak şekilde mesaj döndürmek için tek yapılması gereken **BasicHttpBinding** bağlayıcı tipinin **MessageEncoding** özelliğinin değerini **MTOM** olarak değiştirmektir. Bunun için servis tarafında yeni bir**Binding Configuration** elementinin **BasicHttpBinding** tipi için aşağıdaki ekran görüntüsünde olduğu gibi eklenmesi gerekmektedir.

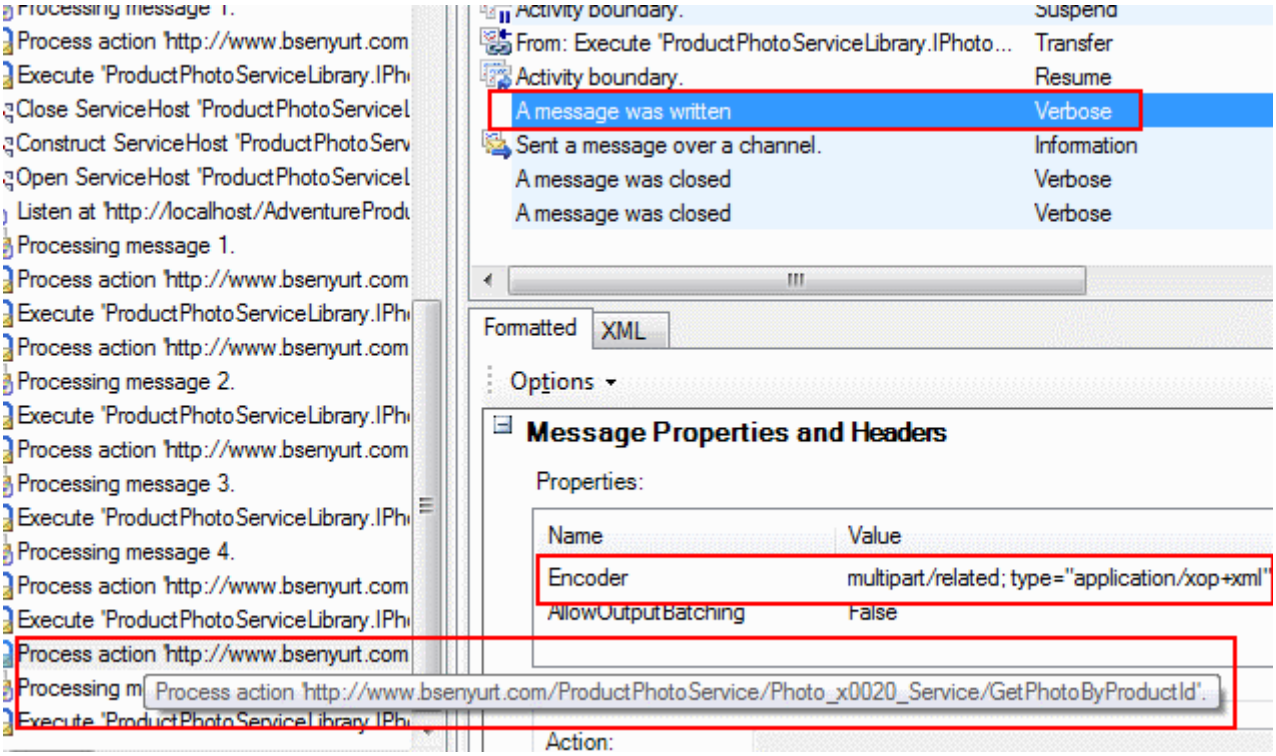


Sonrasında **bağlayıcı tipin(Binding Type)** bu konfigurasyon ile eşleştirilmesi yeterli olacaktır. Bunun
için **ProductPhotoHttpEndPoint** özelliklerinden **BindingConfiguration** elementinin değerinin aşağıdaki ekran görüntüsünde olduğu gibi **Microsoft Service Configuration Editor** üzerinden **ProductPhotoBindingConfiguration** olarak set edilmesi yeterlidir.

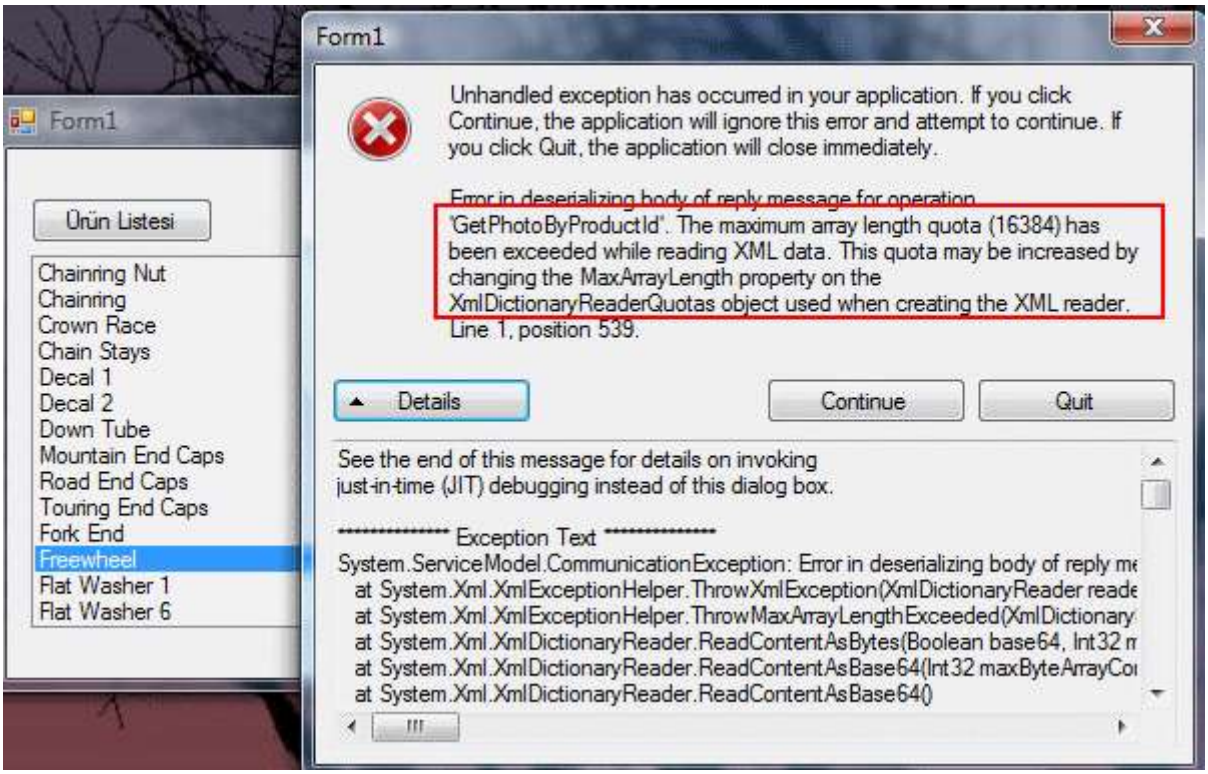
NOT : *Elbetteki servis tarafında yer alan konfigurasyon dosyasında da* ***MessageEncoding*** *özelliğinin değerinin* ***true*** *olarak set edilmesi gerekir. Bu yapılmadığı takdirde* ***çalışma zamanında(Run Time) ProtocolException*** *tipinden bir istisna(Exception) alınabilir.*



Bu işlemin ardından istemci yeniden çalıştırılır ve **Service Trace Viewer** kullanılarak GetProductPhotoId' den geriye döndürülen mesaj özelliklerine bakılırsa aşağıdaki sonucun ortaya çıktığı görülecektir.



Uygulama test edilirken dikkat çekici noktalardan biriside bazı resim dosyaları açılırken aşağıdaki gibi bir çalışma zamanı hata mesajı alınmasıdır.



Bu hatanın sebebi aktarılmak istenen veri boyutunun varsayılan MaxArrayLength değerinden büyük olmasıdır. Bu sorunu çözmek için bağlayıcı tipin konfigurasyon ayarlarında yer alan**MaxArrayLength** değerini değiştirmek yeterlidir. Bu özelliğin varsayılan değeri **16384** byte' dır. örnekte bu değer sembolik olarak **65536** yapılmaktadır. Elbetteki bu özelliğin değerinin hem istemci hemde servis tarafında yapılması şarttır. Söz konusu özellik aşağıdaki ekran görüntüsünde olduğu gibi **Microsoft Service Configuration Editor** yardımıyla değiştirilebilir.

Bu işlemin ardından örnek uygulama çalıştırılırsa sonuçların başarılı bir şekilde alındığı görülebilir. Elbetteki boyutun dahada artması yine aynı **istisnanın(Exception)** alınmasına neden olacaktır.

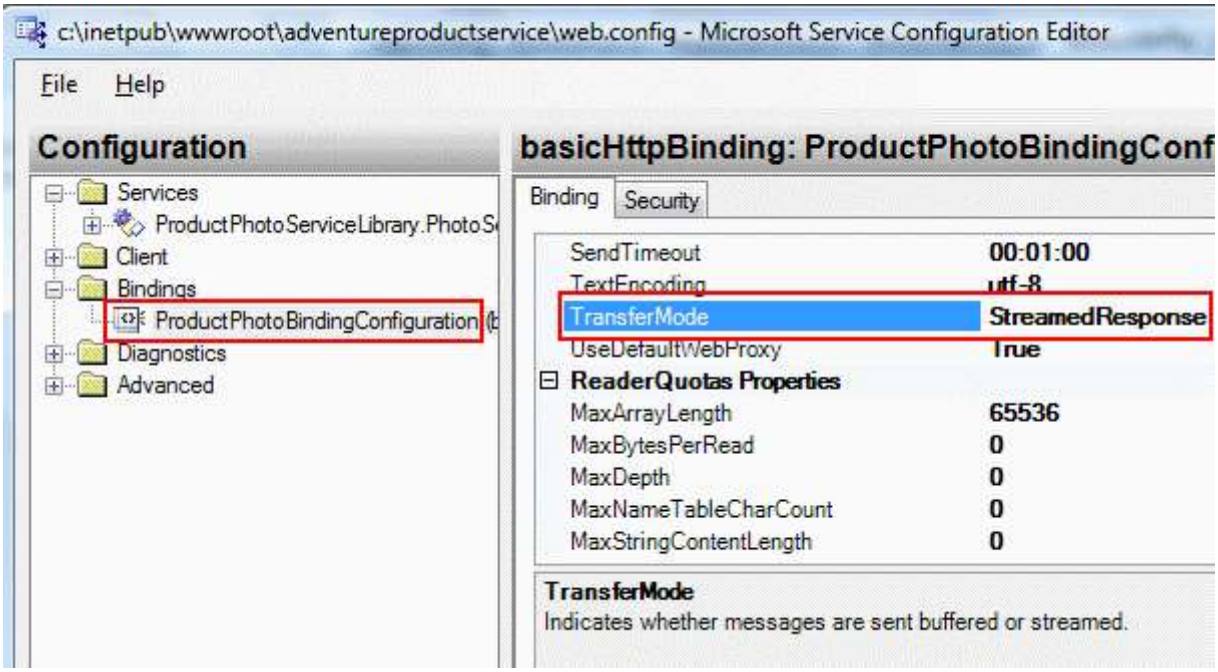
MTOM standardına uygun olarak hazırlanan mesajlar sayesinde büyük boyutlu **ikili(Binary)** verilerin gereksiz yere kodlanmadan aktarılması ve çözülmeden alınması mümkün olabilmektedir. Yinede veri boyutunun çok fazla olması halinde MTOM tabanlı mesajların hazırlanmasıda zaman kaybına neden olacaktır. Nitekim söz konusu büyük veri dosyasının tek seferde **istemciye(veya servise)** doğru aktarılması söz konusudur. Ayrıca bu büyük verinin alınması sırasında **timeout**' lar oluşabilir. Peki çözüm olarak neler yapılabilir? Eğer protokol uygunsa söz konusu büyük verilerin istemci ve servis arasında bir **Stream** üzerinden taşınması tercih edilmelidir. Böylece söz konusu performans kayıplarıda ortadan kalkacaktır. Nitekim mesajı tek seferde hazırlanıp tamamının gönderilmesi yerine iki taraf arasında açılan hat üzerinden akması sağlanabilmektedir.

WCF(Windows Communication Foundation) içerisinde yer alan **basicHttpBinding**, **netTcpBinding**, **netNamedPipeBinding** tipleri **Stream** aktarımın ı desteklemektedir. Tahmin edileceği üzere diğer bağlayıcı tiplerin buna tam desteği olmadığından özel **bağlayıcı tipleri(Custom Binding Types**) kullanmak gerekmektedir. **Stream** kullanılması için tek yapılması gereken**TransferMode** özelliğinin değerini aşağıdaki tabloda yer alan değerlerden birisine set etmektir. *(Bağlayıcı tipin TransferMode özelliği, TransferMode enum sabiti tipinden değerler alabilmektedir.)*

TransferMode Değeri	Açıklama
Streamed	Serivise gelen ve servisten çıkan mesajlar stream üzerinden hareket ederler.
StreamedRequest	Sadece servis tarafına gelen **taleplere(Request)** ait mesajların stream üzeri parametresinin tek ve Stream sınıfından türeyen bir şekilde tasarlanmış olm
StreamedResponse	Sadece servis tarafından istemciye geri dönen **cevaplara(Response)** ait me tarafında yer alan operasyonun geriye Stream sınıfından türemiş bir tip dön şarttır. *(Burada Stream yerine IXmlSerializable arayüzünü-interface- uyarl*
Buffered	Mesajlar stream üzerinden hareket etmezler. Hangi taraf olursa olsun, **ikili(** Karşı tarafa ulaştığında ise tamamı tampona alındıktan sonra uygulamaya iş değeri **Buffered** olarak belirlenmiştir.

Her ne kadar stream üzerinden mesaj göndermek avantajlı gözüksede dikkat edilmesi gereken bazı hususlarda vardır. Herşeyden önce Stream kullanılması halinde **mesaj seviyesinde güvenlik(Message Level Security)** tesis edilememektedir. Bu nedenle **iletişim seviyesinde güvenlik(Transport Level Security)** kullanılmalıdır.
örneğin **HTTPS** kullanımı **HTTP** üzerinden kullanılacak **Stream**' ler için geçerlidir.
Bunların dışında Stream kullanımı içinde bir mesaj alma sınırı vardır. Biraz önceki örnekte olduğu gibi belirlenen boyutun dışına çıkıldığında çalışma zamanı istisnaları alınabilir. Ancak en önemli kısıtlardan birisi **güvenilir oturum(Reliable Session)** açmanın mümkün olmayışıdır. Nitekim güvenilir oturumlar mesajların tamponlanması ve sıralanması ilkelerine göre çalışmaktadır.

Geliştirilen örnekte **Stream** kullanımı için tek yapılması gereken servis ve istemci tarafındaki konfigurasyon dosyalarında, bağlayıcı tip ile ilişkili **TransferMode** özelliğinin değerini aşağıdaki gibi değiştirmektir.*(Bu ayarı istemci tarafı içinde yapmak gerekmektedir.)*



Buraya kadar anlatılanlar göz önüne alındığında **MTOM(Message Transmision Optimization Mechanism)** sayesinde mesajların içeriklerinin **encoding/decoding** işlemlerine tabi tutulmadan karşı tarafa aktarılabilmesi sağlanmaktadır. Bu aynı zamanda **WCF(Windows Communication Foundation)** dışındaki platformlar ile haberleşmedede**(Interoperability Desteği)** önemli bir yere sahiptir. Diğer taraftan **MTOM**' un bazı dezavantajlarını ortadan kaldırmak adına **Stream** kullanımı tercih edilebilir. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

MTOMKullanimi.rar (77,63 kb)

WCF - MSMQ(MicroSoft Message Queue) ile Entegrasyon (2007-11-13T04:42:00)

wcf,msmq,

TCP veya **HTTP** bazlı iletişimlerde, tarafların aynı zaman dilimi içerisinde çalışıyor olmaları gerekmektedir. Böyle bir mesajlaşma sürecinde taraflardan herhangibirin çalışmaması, aradaki bağlantının kopması gibi nedenlerden dolayı tüm iletişimin aksamasıda muhtemeldir. Bazı gerçek hayat senaryolarında, sistemin tarafları olan **istemci(Client), sunucu(Server), ağ(Network)** bileşenlerinin çökmesi durumlarında dahi işlevselliğin devam edebilmesi istenebilir. Bunun dışında, çalışan sistemin içerisindeki bileşenlerin sürekli bir bağlantıda olmadığı durumlarda bu tip iletişimleri zorlaştırmaktadır. Bir başka deyişle ağa sürekli olarak bağlanamayan ama **offline** olarak çalışabilen istemcilerin bu tip bir mesajlaşma sisteminin parçası olması istendiğinde senkronizasyon güçlükleri ortaya çıkmaktadır.

Ne varki mesajlaşmaların kuyruk(Queue) modeline göre taşındığı bir sistemde yukarıda bahsedilen iletişim sorunlarının yaşanmaması sağlanabilir. Nitekim **mesaj kuyruğu(Message Queue)** sisteminin geliştirilme amacıda aslında budur. Bu sistem temelde Windows tabanlı bilgisayarlar arasında kuyruk sistemine dayalı bir mesaj alışverişinin tesis edilmesini sağlamaktadır. Bu iletişimde **istemci(Client), servis(Service)** ve **ağ(Network)** arasında bir izolasyon sağlanmaktadır. Bu izolasyon sayesinde istemci, servis yada ağ çökse, hata üretse dahi fonksiyonelliklerini devam ettirebileceklerdir. Kuyruk temelli iletişimde **güvenilir(Reliable)** bir iletişim ortamı tesis edilmekte olup transaction kullanılabilmekte, kuyruğa atılan mesajlar fiziki yada**değişken(Volatile)** olarak saklanabilmektedir. Kuyruk temelli mesajlaşmanın en bilinen uyarlaması **MicroSoft Message Queue bileşenidir(Component).**

MicroSoft Message Queue(MSMQ), **Windows NT** sürümünden beri gelen bir bileşendir. NT işletim sistemi için **1.0** versiyonu duyrulan MSMQ bileşeninin çok kısa tarihçesi aşağıdaki tabloda olduğu gibidir.

MSMQ Versiyonu	Desteklenen İşletim Sistemi(Sistemleri)	özellikler
1.0	Windows NT	Varsayılan özellikler...
2.0	Windows 2000	• **Public** mesaj kuyruklarının **Active Directory**destekli olarak saklanması • **128** bitlik şifreleme desteği • Dijital imza desteği • Tam **COM** desteği
3.0 *	Windows XP Windows Server 2003	• **HTTP** , **SOAP** desteği ile **internet** üzerinden mesajlaşabilme • **IIS** için **MSMQ** desteği • Mesajların çoklu Dönüştürülebilmesi**(MultiCasting)**
4.0 **	Vista Longhorn Server	• Alt kuyruk desteği**(Subqueue)** • Zehirli mesajların ele alınması**(Posion Message Handling)** • Uzak kuyruklardan transaction bazlı mesaj alma desteği**(Transactional Remote Receive)**
* *http://www.microsoft.com/windowsserver2003/technologies/msmq/whatsnew.mspx* ** *http://windowssdk.msdn.microsoft.com/en-us/library/ms701784.aspx*		

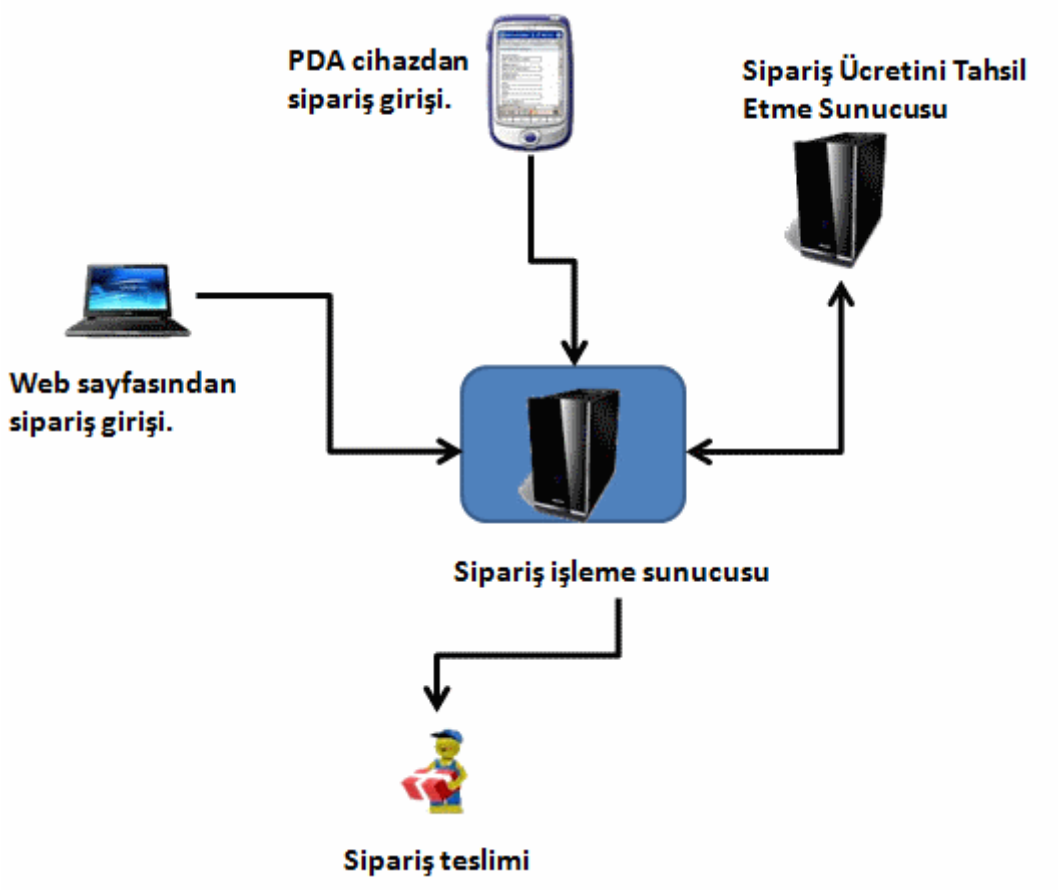
Ayrıca 1999 yılında MSMQ' nun mobil cihazlar için olan desteğide **Windows CE 3.0** işletim sistemi ile birlikte başlamıştır.

MSMQ Windows tabanlı bir bileşendir. Ancak farklı bilgisayarlar ilede haberleşme olanağı mevcuttur. MSMQ' nun görevi aynen diğer iletişim sistemlerinde olduğu gibi farklı bilgisayarlar arasında mesajlaşmayı sağlayabilmektir. Bu mesajlaşmadaki tek fark, tarafların aynı anda çalışıyor olma zorunluluklarının bulunmayışıdır. Hatta söz konusu taraflarda çalışan uygulamalar bir ağa(Network) bağlı olmak zorundada değildirler. Böyle bir durumda ortaya çıkan soru iletilmek istenen mesajların karşı taraf kapalı iken nasıl ulaştırılacağıdır? MSMQ bileşenleri, mesajları, karşı taraf hazır oluncaya kadar bir depoda saklamaktadır. Bu deponun tutuluş yeri varsayılan olarak fiziki bir ortamdır. Ancak istenirse performası arttırmak adına geçici bir ortamda da saklanabilirler. Ne varki **Volatile** olarak adlandırılan bu mesajlar sistemlerin çökmesi (MSMQ bileşeninin yüklü olduğu bilgisayarın örneğin yeniden başlatılması-**restart**) halinde kaybolacaktır. Oysaki fiziki saklanan mesajlar korunmaya devam edecektir. Buda MSMQ' nun mesajlar için uzun süreli bir saklama ortamı kullanabildiği anlamına gelmektedir**(Durable)**.

MSMQ bileşenlerine .**NET** ortamı içerisinde **System.Messaging isim alanında(Namespace)** bulunan tipler yardımıyla erişilebilmektedir. Bu sebepten programatik olarak **C#, VB.Net** ve diğer .NET destekli diller yardımıyla kuyruklara bakılması, mesaj eklenmesi ve daha gelişmiş yönetsel işlemlerin yapılması mümkündür. Diğer taraftan MSMQ' nun **C/C++** ile yazılmış olan ve**COM** tabanlı ortamlarda kullanılabilen bir **API'side** mevcuttur. Elbette **WCF** içerisinde MSMQ için hazır olan bazı **bağlayıcı tipler(Binding Types)** kullanılmaktadır.

MSMQ sonuç itibariyle taraflar arasında mesaj taşıdığından mesajların boyutlarıda performans açısından önemlidir. Varsayılan olarak minimum mesaj büyüklüğü **150 byte**' tır. Bu mesaj içerisinde, **imza(Signature)**, **kaynak ve hedef bilgisayar ID**' leri, **hedef kuyruğun adı(Target Queue Name)**, **mesaj özellikleri** yer alır.
Ancak **transaction** kullanılıyorsa yada**doğrulama(authentication)** ve **şifreleme(encryption)** söz konusuysa bu boyutun artması muhtemeldir. örneğin dahili bir sertifika kullanımında mesaj boyutu **400 byte** kadar artmaktadır. Harici sertifika**(Certificate)** kullanımında en az **1 kb**' lık bir mesaj boyutu oluşmaktadır. Bunların dışında **HTTP** veya **MultiCast** kullanımı söz konusu ise **SOAP** formatlamadan dolayı mesajın sadece **başlık kısmı(Message Header) 1 kb** boyutunda olacaktır. çok doğal olarak mesaj boyutlarındaki bu artışlar iletişim hızını olumsuz yönde etkileyebilir. Dolayısıyla en uygun senaryoları**(Best Practices)** ele almak gerekir. *(Bölümün sonunda bu konuya kısaca değinilecektir.)*

Peki MSMQ kullanmak için gerekli senaryolar neler olabilir? Aslında kuyruk tabanlı iletişim, güvenilir ağ ortamlarının söz konusu olmadığı, **ağ ortamına(Network)** arada sırada bağlanabilen uygulamaların haberleşmesi gerektiği durumlarda sıklıkla ele alınmaktadır. örneğin bir servis uygulaması tarafından, gün içinde **offline** ortamlardan gelen siparişlerin gün sonunda toplu olarak işlendiği bir senaryo göz önüne alınabilir. Aşağıdaki şekilde buna benzer bir senaryo incelenmeye çalışılmaktadır.



Senaryoda, zaman zaman **offline** duruma düşebilecek olan istemciler söz konusudur. Bu istemciler üzerinden gelen siparişler bir sunucuda toplanmaktadır. Siparişleri işleyen sunucu tahsilatlar için başka bir sunucuyu kullanmaktadır. Burada söz konusu siparişleri işleyen servis ile, tahsilatın yapıldığı sunucu ve **PDA** cihazında kullanılan uygulamaların sürekli bir bağlantıda olmadığı göz önüne alınmaktadır. Böyle bir durumda bir siparişin işlenmesi zaman alabilir. Ayrıca bu bekleme sürelerinin siparişleri işleyen uygulamayı etkilemesi istenmez. Yani, bekleyen siparişlerin gecikmeli olarakta olsa işlenebildiği ve yeni gelen siparişlerinde toplanabildiği bir ortamın hazırlanması gerekmektedir. Böylece **PDA** satıcılar siparişleri **offline** olarak toplayabilir ve daha sonra işlenmek üzere(servis ile bağlantı tekrardan sağlandıktan sonra) servise gönderebilir. Ayrıca web üzerinden gelen siparişlerin işleme alındığı bilgisi anında kullanıcılara gösterilebilir(Laptop ile ifade edilen kullanıcılar). Bu tip senaryolar genişletilebilir. Ancak uygun vakalar çoğunlukla **offline** çalışma ihtiyacı olan, güvenilir bir ağ ortamında bulunamayan veya bir ağa hiç bağlı olmayan bilgisayarların, asenkron olarak haberleşmesi gereken durumlardır.

NOT : *Burada çok önemli bir nokta vardır.* ***MSMQ*** *gibi kuyruk tabanlı iletişimi benimsiyen sistemler* ***asenkron(asynchronous)*** *olarak çalışmaktadır. MSMQ bu anlamda,****güvenilir(Reliable)*** *ortam oluşturabilen,* ***Transaction*** *desteği sağlayan,* ***bağlantısız(Disconnected)*** *çalışmaya olanak tanıyan,* ***sürekli depolama(Durable)*** *desteği veren mükemmel bir* ***asenkron(Asynchronous)*** *iletişim ortamı sağlamaktadır.*

MSMQ bileşenlerinin kullanıldığı uygulamalar herhangibir **Windows** programı olabilir. Bu nedenle bir Windows uygulamasından**(Windows Application)**, Windows Servisinden**(Windows Service)**, akıllı istemciden**(Smart Client)**, web sitesinden MSMQ kullanılabilir. Bu noktada MSMQ ile ilgili önemli sorunlardan birisi gündeme gelmektedir. örnek bir windows işletim sistemi tabanlı uygulama, MSMQ kullanarak, kuyruk tabanlı başka bir sistem ile nasıl haberleşebilir? Söz gelimi kuyruk tabanlı iletişimi kullanan **IBM MQSeries** ile bir **MSMQ** istemcisi nasıl haberleşebilir?

MSMQ istemcileri **yabancı bilgisayarlar(Foreign Computers)** ile **Connector** adı verilen uygulamalar aracılığıyla haberleşebilirler. **Connector** uygulamaları aslında bir MSMQ sunucusunda çalışır ve yabancı bilgisayarların MSMQ istemciler ile olan haberleşmelerinde aracılık yapar. Bu aracılıkta transaction desteği olan veya olmayan kanallar söz konusudur. **IBM MQSeries** ile iletişimi sağlayan Connector uygulama **Microsoft MSMQ-MQSeries Bridge** programıdır. Tabi farklı kuyruk tabanlı sistemler için farklı Connector uygulamaları mevcuttur.

NOT : *MSMQ' nun yazının hazırlandığı tarih itibariyle **4.0** versiyonu bulunmaktadır. Bu versiyon ile gelen bazı belirgin özelliklerin bilinmesi gerekmektedir. 4.0 versiyonu **alt kuyruklar(SubQueue)**, zehirli mesajların ele alınması**(Poison Message Handling)**, transaction bazlı uzak kuyruk desteği**(Transactional Remote Receive)** şeklinde 3 önemli yenilik içermektedir.*

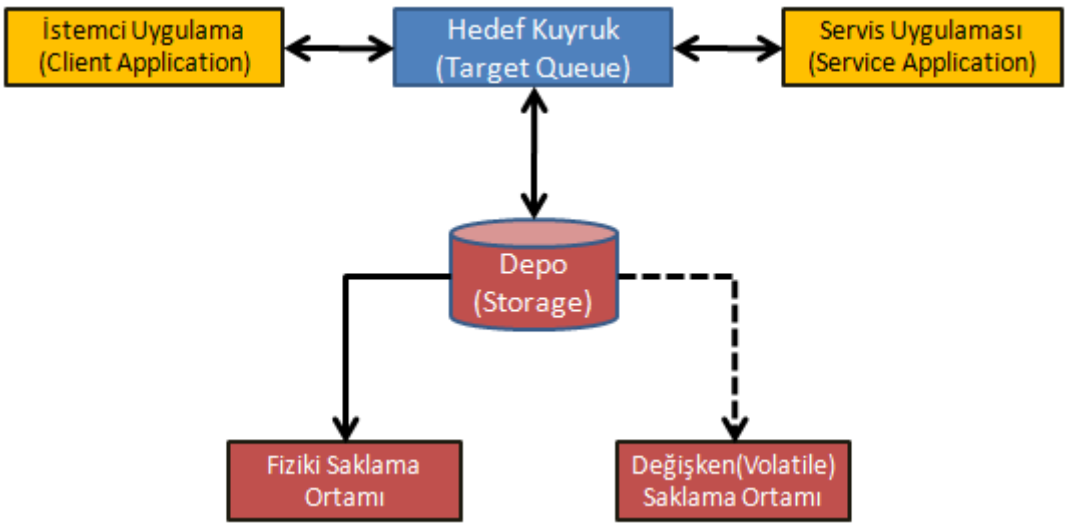
SubQueues(Alt Kuyruklar): *Mesajların ek fiziki kuyruklar oluşturmadan mantıksal olarak çeşitli kriterlere göre alt gruplara ayrılabilmesi desteğidir.*

Zehirli Mesajların Ele Alınmas(Poison Message Handling): *Zehirli mesajlar, bir kuyrukta arkadan gelen mesajları engelleyen niteliktedir. Bir uygulama kuyruktan bir mesajı okuyup işlemek isterken, mesajdaki hata nedeni ile bu işlemi yapamadığı durumlarda, (eğer süreç bir transaction içerisindeyse) **Abort** işlemini gerçekleştirip mesajı tekrardan kuyruğa döndürecektir. Bu mesaj daha sonradan tekrardan işlenmek amacıyla uygulama tarafından yeniden okunacak ama hatası nedeni ile yine kuyruğa geri döndürülecektir. Bu durumda transaction' ın sürekli olarak abort edildiği ve mesajın kuyruğa gönderildiği sonsuz bir döngü oluşacaktır. Bu döngüye neden olan mesaj bir süre sonra tekrardan deneme sayısınında aşılması sebebiyle arkadan gelen mesajlarında işlenmesini engelleyecektir. İşte **MSMQ 4.0** versiyonu ile birlikte bu tip mesajların daha güçlü bir şekilde ele alınması ve sorunun önüne geçilmesi sağlanmaktadır. Bunun içinde **Retry Queue**(yeniden deneme kuyruğu) adı verilen kuyruklar kullanılmaktadır. (WCF içerisindeki bağlayıcı tipler buna destek veren özellikler(Properties) içermektedirler)*

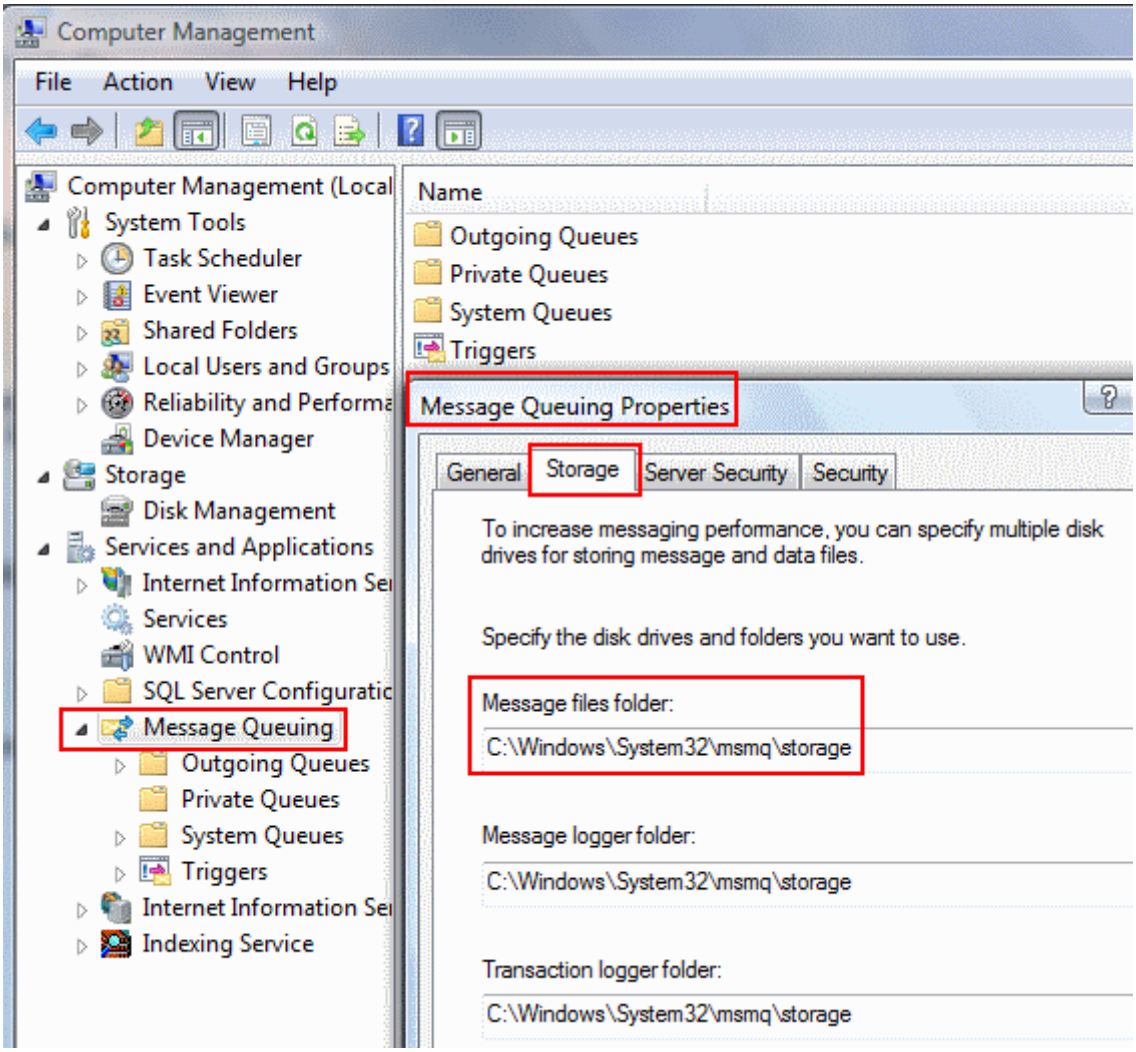
Transaction bazlı uzak kuyruk desteği(Transactional Remote Receive): *Tek bir mesaj kuyruğunun işleri aldığı bir durumda, başka bilgisayarlardaki uygulamalarında bu işleri değerlendirdiği düşünülsün. Uygulamardan biri sıradaki bir işi işleyemez ise bu işin sonradan değerlendirilmek üzere tekrardan kuyruğa gönderilmesi tercih edilir. Bu tip bir süreç için transaction desteği gerekmektedir. Ayrıca istemci bilgisayarların ve*

*kuyruğun **ağ(Network)** üzerinden **DTC(Distributed Transaction Coordinator)** erişiminede izin vermesi şarttır.*

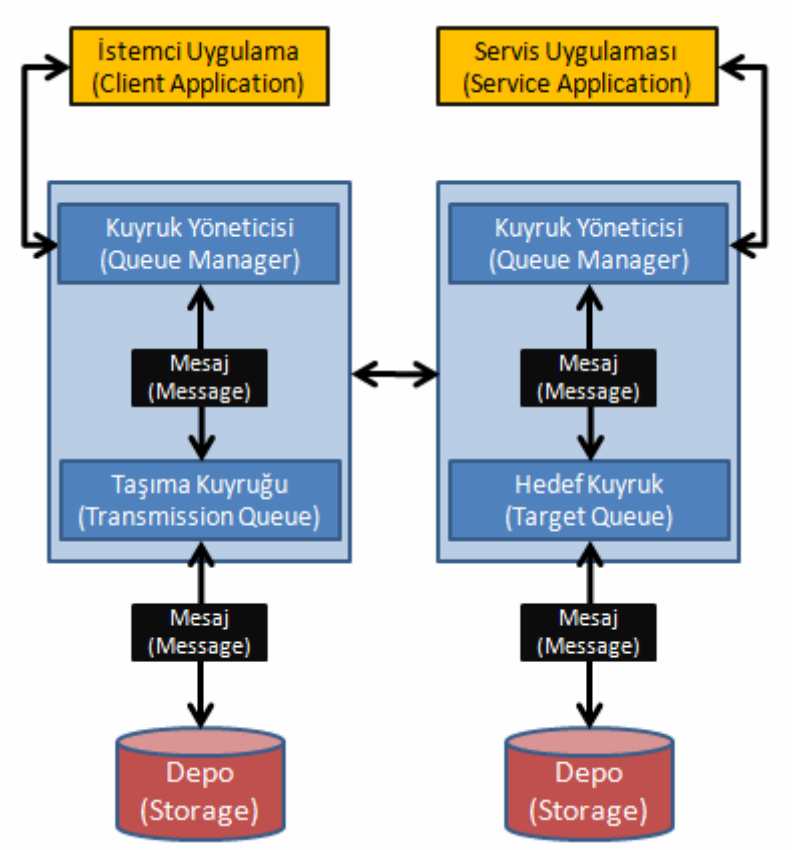
MSMQ, WCF içerisinde de doğrudan desteklenmektedir. üstelik WCF' in sağladığı birleştirilmiş**(Unified)** model sayesinde, geliştiriciler MSMQ' nun karmaşık alt yapısından uzaklaşmaktadır. Basit anlamda WCF üzerinden MSMQ konseptine bakıldığında aşağıdaki durum söz konusudur.



Burada istemci ve servis uygulamaları ortak bir kuyruğu kullanarak iletişim sağlamaktadır. Kuyruk içerisinde yer alan mesajlar fiziki bir ortamda sakalanbilmektedir. **Microsoft Managament Console**kullanıldığı takdirde kuyruktaki mesajların tutulacağı yer genellikle aşağıdaki şekildende görüldüğü gibi **Windows\System32\msmq\Storage** klasörüdür.



Diğer taraftan gerçek bir dağıtık mimari vakasında mesaj kuyrukları çoğunlukla istemci ve servis uygulamaları için ayrı ayrı tutulurlar. Bu durum aşağıdaki şekilde olduğu gibi ifade edilebilir.

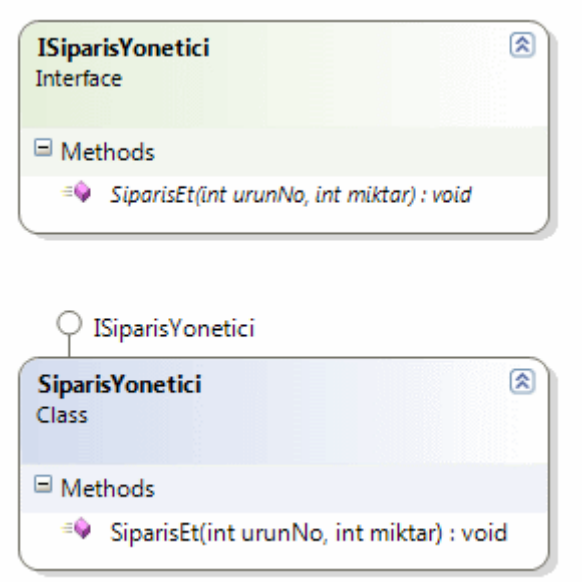


Bu senaryoda istemci ve servis uygulamaları kendi ortamlarındaki **kuyruk yöneticileri(Queue Manager)** yardımıyla haberleşmektedir. Kuyruk yöneticileri mesajları kuyruğa atmak veya kuyruktan okumak için gerekli yönetsel işlemleri üstlenmektedir. İstemci uygulamalar, servis uygulamasına göndermek istedikleri mesajları kuyruk yöneticisine iletirler. Kuyruk yöneticisi bu mesajları **taşıma kuyruğuna(Transport Queue)** aktarır. Karşı taraf hazır olduğundada servis uygulamasındaki kuyruk yöneticisine gönderilir. Servis tarafındaki kuyruk yöneticisi gelen mesajı alarak hedef kuyruğa**(Target Queue)** atar. Sonrasında ise servis tarafındaki kuyruk yöneticisi, hedef kuyruğa kabul ettiği mesajı işlenmek üzere servis uygulamasına yönlendirir. Burada iletişim, güvenilir bir ortamda gerçekleştirilir.

WCF(Windows Communication Foundation) tarafında **MSMQ** desteği için iki farklı **bağlayıcı tip(Binding Type)** kullanılmaktadır.
Bunlar **netMsmqBinding** ve **msmqIntegrationBinding**tipleridir. netMsmqBinding iki **WCF End Point** arasında MSMQ tabanllı bir iletişimi sağlamak amacıyla kullanılır. msmqIntegrationBinding tipi ise bir WCF End Point ve **C, C++, COM** veya**System.Messaging API**' si yardımıyla geliştirimiş bir MSMQ destekli uygulama arasında iletişimin sağlanması için kullanılmaktadır. Yanlız bu tip kullanılırken önemli bir kısıt vardır. Bu kısıta göre operasyon sözleşmesi**(Operation Contract)** tanımlanırken **MsmqMessage** tipinden tek parametre alan bir metod söz konusu olabilir. Bunun tek sebebi WCF dışında olan bir ortam ile ortak haberleşebilmeyi sağlamaktır.

***NOT : WCF** tabanlı **MSMQ(MicroSoft Message Queue)** uygulamlarında servis tarafında tanımlanan operasyonların **tek yönlü(OneWay)** olacak şekilde tanımlanmaları gerekmektedir. Bunun sebepi **MSMQ**' nun normal şartlarda **tek yönlü iletişimi(One-Way Transport)** kullanıyor olmasıdır.*

Artık bir örnek üzerinden hareket edilerek WCF içerisinde **MSMQ**' nun nasıl kullanılabileceğine bakılabilir. öncelikli olarak **servis sözleşmesi(Service Contract)** ve uygulayıcı sınıfın yer aldığı **WCF Servis Kütüphanesinin(WCF Service Libraray)** örnek olarak aşağıdaki gibi tasarlandığı göz önüne alınsın.



Servis Sözleşmesi;

```
using System;
using System.ServiceModel;

namespace SiparisKutuphanesi
{
   [ServiceContract(Name="Siparis
Servisi",Namespace="http://www.bsenyurt.com/SiparisServisi")]
   public interface ISiparisYonetici
   {
      [OperationContract(IsOneWay=true)]
      void SiparisEt(int urunNo, int miktar);
   }
}
```

Servis sözleşmesinde dikkat edilmesi gereken noktalardan birisi **OperationContract** isimli **nitelikle(Attribute)** kullanılan **IsOneWay** özelliğine true değeri atanmasıdır. Daha önceden de bahsedildiği gibi **MSMQ** tipinde haberleşmelerde mesajların tek yönlü çalışması söz konusudur. Bu doğal olarak mesajın gönderildikten

sonra işlenip işlenmediğinden haberdar olunamaması anlamına da gelmektedir. Dolayısıyla mimari gereği operasyonun **tek yönlü(One Way)** olacağı belirtilmelidir.

Uygulayıcı Sınıf;

```
using System;
using System.Threading;
using System.ServiceModel;

namespace SiparisKutuphanesi
{
   public class SiparisYonetici:ISiparisYonetici
   {
      #region ISiparisYonetici Members

      public void SiparisEt(int urunNo, int miktar)
      {
         Thread.Sleep(7000);
         // Sipariş ile ilgili işlemler
      }

      #endregion
   }
}
```

Uygulayıcı sınıf içerisinde sembolik olarak **SiparisEt** isimli bir metod bulunmaktadır. Bu metodun herhangibir işlevi yoktur. Sadece **MSMQ** senaryosunu tamamlayıcı bir öğedir. Diğer taraftan kuyruk oluşumlarını daha kolay izleyebilmek ve asenkron çalışmayı daha kolay takip edebilmek adına metod içerisinde bilinçli olaraktan, çalışan **Thread**' in 7 saniye süreyle uyutulması sağlanmaktadır.

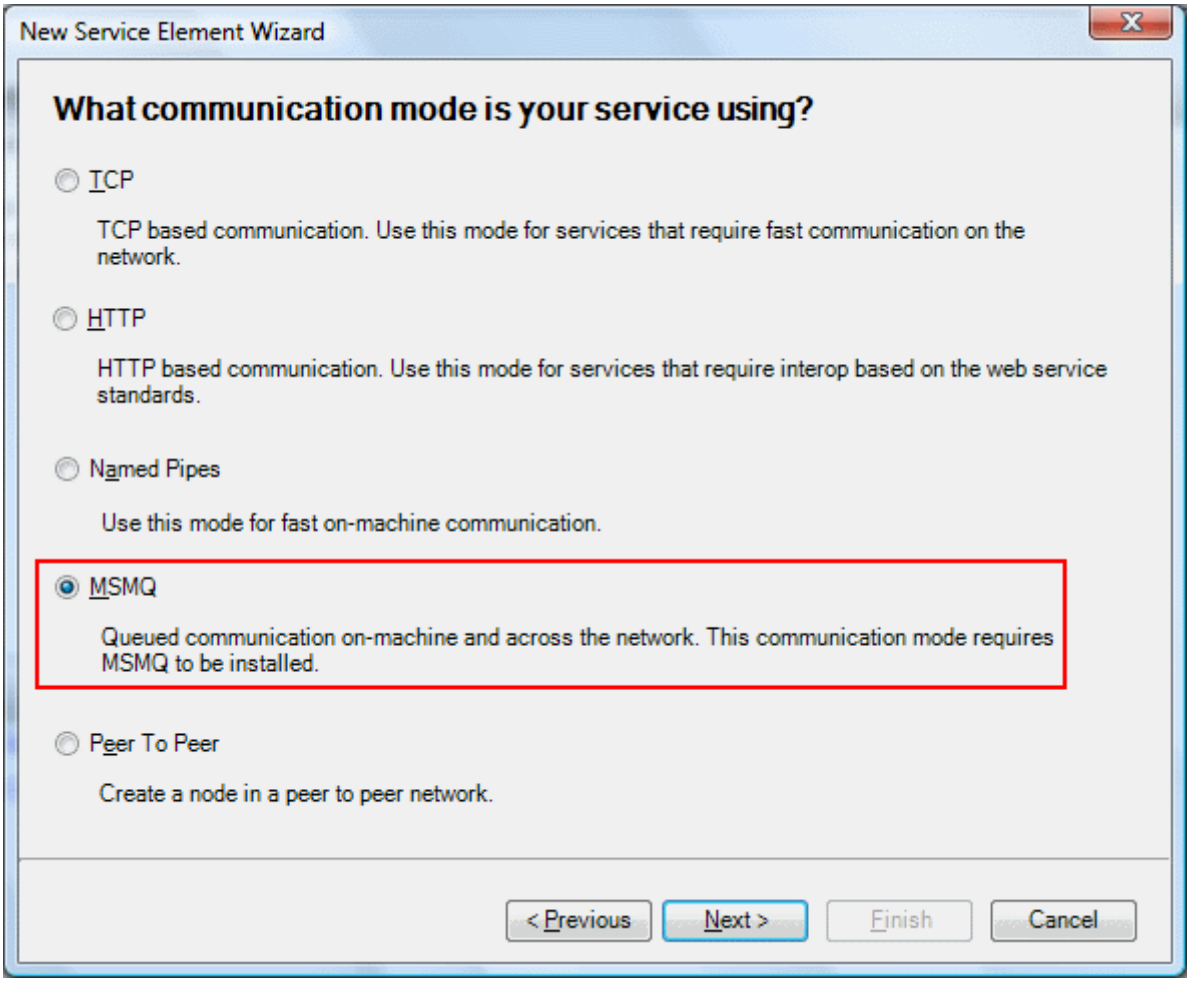
Servis uygulaması basit bir **Console** uygulaması olarak tasarlanabilir. Nitekim burada önemli olan noktalar konfigurasyon dosyası içerisinde yapılan değişikliklerdir. Servis uygulamasına ait konfigurasyon dosyasının içeriği ve kaynak kodları aşağıdaki gibidir. *(Servis uygulamasında, WCF Servis Kütüphanesi ve **System.ServiceModel.dll assembly**' larının referanslarının eklenmemesi unutulmamalıdır.)*

Servis tarafı konfigurasyon dosyası;

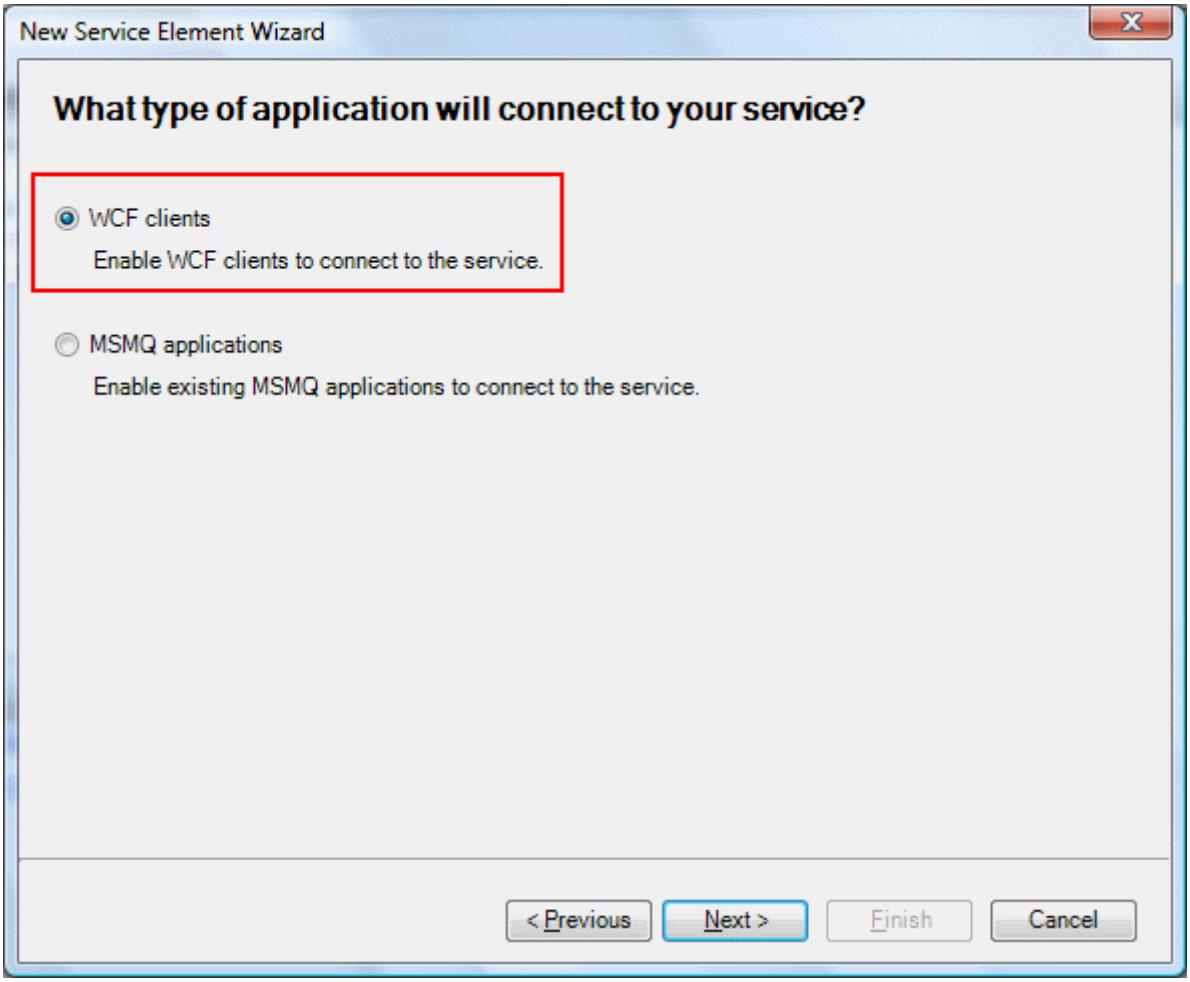
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <bindings>
         <netMsmqBinding>
            <binding name="SiparisServisiBindingConfig" receiveTimeout="00:10:00"
```

```
deadLetterQueue="System" durable="true" exactlyOnce="true"
maxRetryCycles="2" receiveErrorHandling="Fault" receiveRetryCount="5"
retryCycleDelay="00:30:00">
                <security mode="None" />
            </binding>
        </netMsmqBinding>
    </bindings>
    <services>
        <service name="SiparisKutuphanesi.SiparisYonetici">
            <endpoint address="net.msmq://localhost/private/SiparisKuyrugu" binding=
"netMsmqBinding" bindingConfiguration="SiparisServisiBindingConfig"
name="SiparisServisiEndPoint" contract="SiparisKutuphanesi.ISiparisYonetici" />
        </service>
    </services>
  </system.serviceModel>
</configuration>
```

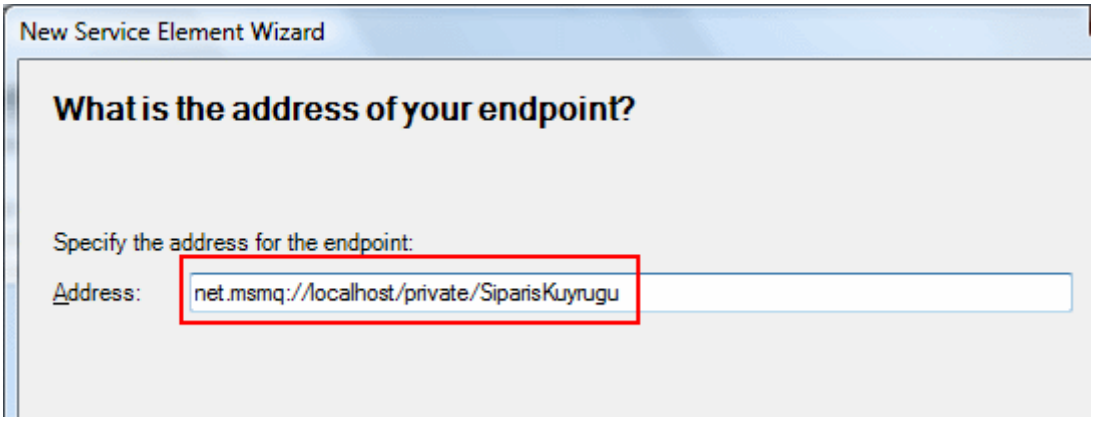
Konfigurasyon dosyasının içeriği **Microsoft Service Configuration Editor** yardımıyla da oluşturulabilir. Bu yol izlenirse eğer, daha önceden değinilen **TCP**, **HTTP**, **Named-Pipes, Peer To Peer**gibi bağlantılardan farklı olarak bağlayıcı tip seçilirken aşağıdaki ekran görüntüsünde olduğu gibi **MSMQ** seçilmelidir.



Yapılan bu seçimin ardından editor, istemcilerin **MSMQ** yada **WCF** tabanlı bir uygulama olup olmadığı sorulacaktır. Daha öncedende belirtildiği gibi, **WCF End Point**' leri arasında yada bir**WCF End Point** ile bir MSMQ uygulaması arasında mesaj kuyruğu tabanlı iletişim sağlanabilmektedir. Bu seçime göre kullanılacak olan **bağlayıcı tip(Binding Type)** belirlenmektedir. örnekte **netMsmqBinding** ele alınmıştır. Bir başka deyişle WCF istemcileri ile MSMQ üzerinden konuşacak bir servis tasarlanmaktadır.

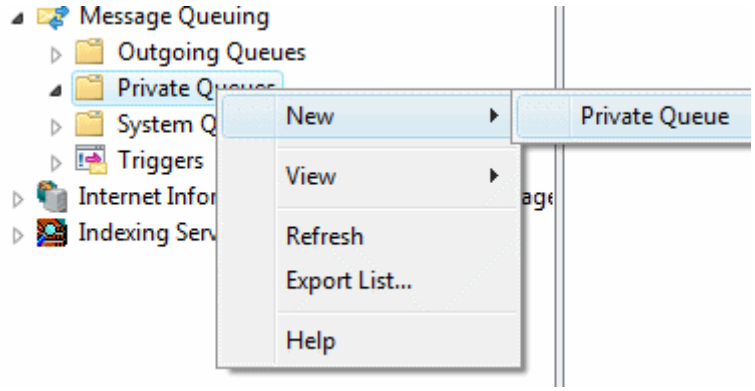


Bu seçimin ardından adres bilgisinin girilmesi gerekir. **MSMQ** kullanıldığı için söz konusu adres aslında yerel makinedeki fiziki bir adresi işaret etmelidir. Burada örnek olarak aşağıdaki ekran görüntüsünde yer alan adres tanımı kullanılmaktadır. Bir başka deyişle, yerel makinede yer alan özel SiparisKuyrugu isimli kuyruk(Queue) adres olarak ele alınmaktadır.

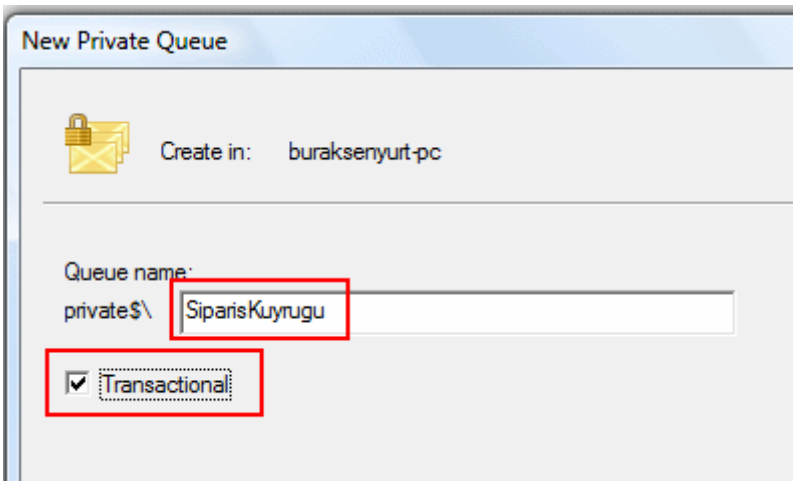


NOT : *Burada üzerinde durulması gereken noktalardan biriside şudur. SiparisKuyrugu isimli* ***private*** *kuyruk nasıl oluşturulmuştur? Bunun için sistemde* ***MSMQ****' nun kurulu olması elbette şarttır.* ***MSMQ****' nun kurulması halinde örneğin* ***Vista*** *işletim sistemi*

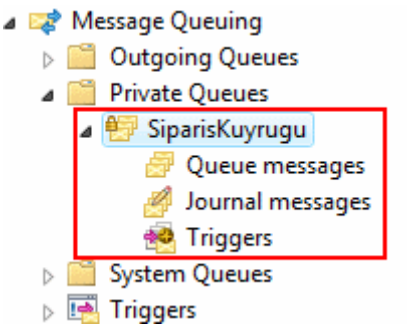
*üzerinde **Microsoft Management Console** açılarak **Meesage Queuing** klasörüne gidilir. Ardından menüden **New->Private Queue** seçeneği işaretlenir.*



*Bu işlemin ardından, **New Private Queue** öğesinden yararlanarak kuyruğun adı girilir. örnek uygulamada **ExactlyOnce** özelliğine **true** değeri atanmıştır. Bu, mesajların yanlız bir tane gitmesini ve alınmasını sağlamaktadır. Yani tekrar eden mesajların önüne geçilmektedir. Ancak bunun için **MSMQ**' nun **transactional** destekli oluşturulması gerekir. Bu nedenle**Transactional** seçeneği işaretlenmelidir.*



*Sonuç olarak işlemler tamamlandığında, ilk haliyle aşağıdaki gibi bir sonuç ortaya çıkacaktır. (Servis uygulaması çalıştırıldıktan sonra burada **retry** isimli bir alt klasör daha oluşturulduğu görülebilir.)*



Burada **private**' ın özel bir anlamı vardır. Bu tanımlamaya göre mesaj kuyruğunun, servis uygulamasının çalıştığı yerel makinede saklanacağı belirtilmektedir. Ancak **Windows Domain**' e dahil edilmiş bir kuyruk tanımlaması **public** olarak yapılarak başka bilgisayarlarında aynı kuyruğu kullanmaları sağlanabilir. Bu özellikle ağa çıkarak hedef kuyruk ile doğrudan konuşamayan alt yapılarda işe yarayabilir. Nitekim böylece ağa çıkamayan istemciler aslında public lokasyonda duran bir kuyruğa mesajlarını gönderirler. Bu kuyruğun bulunduğu makinede kuyruktaki mesajları hedef servise yönlendirir.

Konfigurasyon içerisinde kullanılan bazı önemli ayarlar bulunmaktadır. Bu özelliklerin kısa açıklamaları aşağıdaki tabloda görüldüğü gibidir.

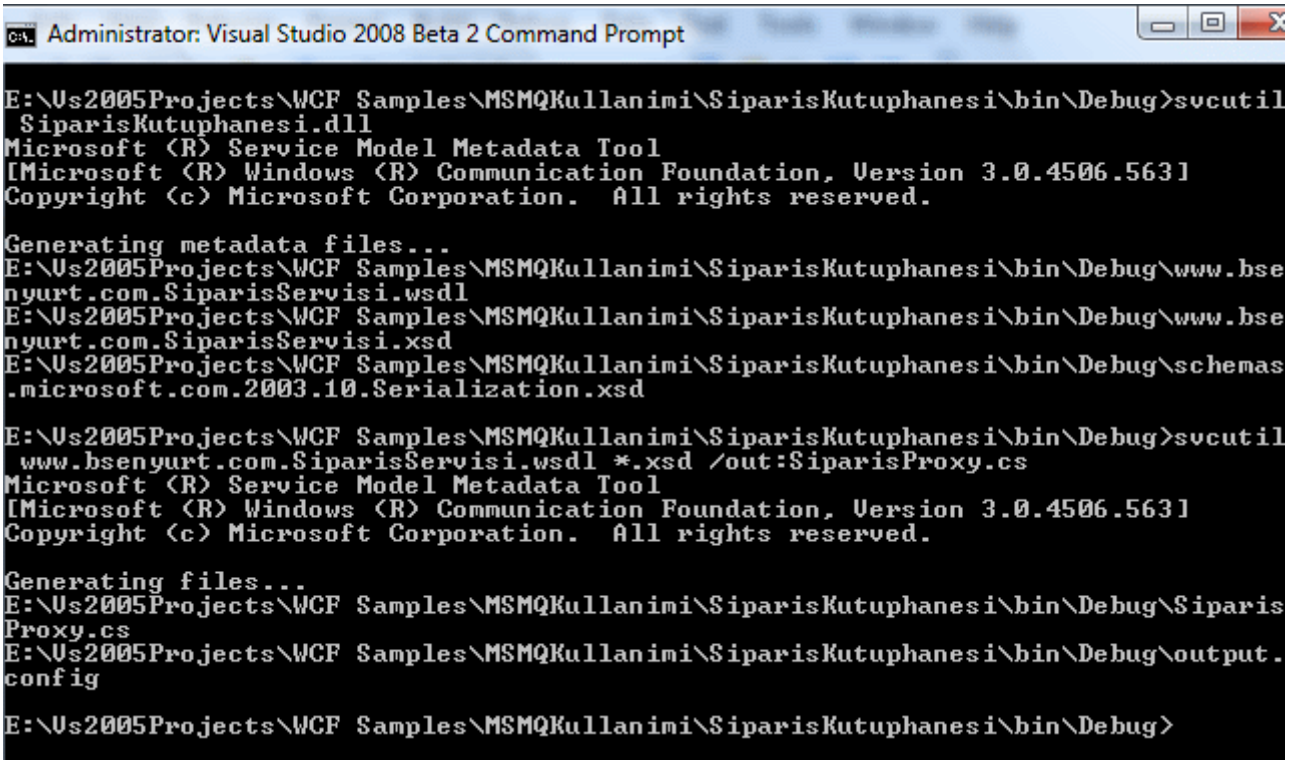
özellik	**Açıklama**
Durable	**true** veya **false** değerini alabilir. True olması halinde mesajların fiziki diskte saklanacağı belirtilir. Bu **I/O** işlemlerinin yoğun şekilde kullanılmasını gerektirmektedir. Dolayısıyla sistemin en iyi performansının elde edilmesinde genellikle false olarak ayarlanır. Ne varki false değer verilmesi halinde mesajlar**Volatile** olarak tutulurlar. Bir başka deyişle makine kapatılıp açıldığında Volatile mesajlara ulaşılamaz.
ExactlyOnce	**true** veya **false** değer alabilir. True olması halinde sistemde hareket eden mesajların tekrar edilmeyeceği garanti altına alınır. Bir başka deyişle mesajlar kaybolmaz veya yanlışlıklada olsa iki kere gönderilmez. Bu özelliğin true olmasının bir şartıda kuyruğun transactional olarak tanımlanma zorunluluğudur.
Security kısmından Mode	**MSMQ** normalde iletişim**(Transport Level)** ve mesaj**(Message Level)** seviyesinde güvenlik kullanımına izin verir. Ancak kendisine ait bir iletişim güvenliği söz konusu olduğundan**SSL** kullanılmasını gerektirmez. Mesaj seviyesinde güvenlik ayarları yapıldığında tek şart vardır o da sertifika desteğinin MSMQ için sağlanmış olması gerekmektedir.
ReceiveRetryCount	Kuyruktan uygulamaya doğru bir mesajın ulaştırılması için maksimum deneme sayısını belirtir. Varsayılan değeri 5 tir.
MaxRetryCycles	Bir mesaj uygulama kuyruğundan retry **alt kuyruğuna(SubQueue)**transfer edildiğinde, **RetryCycleDelay** süresinden sonra uygulama kuyruğunda işlenmek üzere tekrardan gönderdilir. Buradaki tekrar sayısı **MaxRertyCycle** ile belirlenir. Varsayılan değeri 2 dir.

RetryCycleDelay	**RetryCycle**' lar arasındaki süredir. Varsayılan olarak **30**dakikadır. Bir başka deyişle **retry alt kuyruğundan**uygulama kuyruğuna ne kadar sürede bir gönderilme denemesi yapılacağı belirtilir.
ReceiveErrorHandling	Maksimum ulaştırma sayısı aşılıp hata oluşması halinde nasıl bir etki olacağı belirtilir. Varsayılan olarak bu değer **Fault** şeklindedir. Yani uygulamaya bir istisna mesajı fırlatılacaktır. Diğer değerleri ise **Drop**, **Reject** ve **Move**' dur. Vakaya göre uygun olan değerin seçilmesi gerekmektedir.
*(Burada söz konusu olan **ReceiveRetryCount, MaxRetryCycles, RetryCycleDelay, ReceiveErrorHandling** özellikleri **zehirli mesajların ele alınması-Poison Message Handling** için önemlidir)*	

Servis tarafı uygulama kodları;

```
using System;
using System.ServiceModel;
using SiparisKutuphanesi;

namespace Servis
{
   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(SiparisYonetici));
         host.Open();
         Console.WriteLine(host.State.ToString());
         Console.WriteLine("Kapatmak için bir tuşa basınız.");
         Console.ReadLine();
         host.Close();
      }
   }
}
```

Artık istemci tarafı programlanmaya başlanabilir. İstemci uygulamada basit bir **Console** uygulaması olarak tasarlanabilir. İstemci için gerekli **Proxy** sınıfının üretimi için **SvcUtil.exe** aracından aşağıdaki ekran görüntüsünde olduğu gibi faydalanmak gerekmektedir.



Bu işlemin ardından istemci uygulamanın kodları ve konfigurasyon içeriği aşağıdaki gibi düzenlenmelidir.

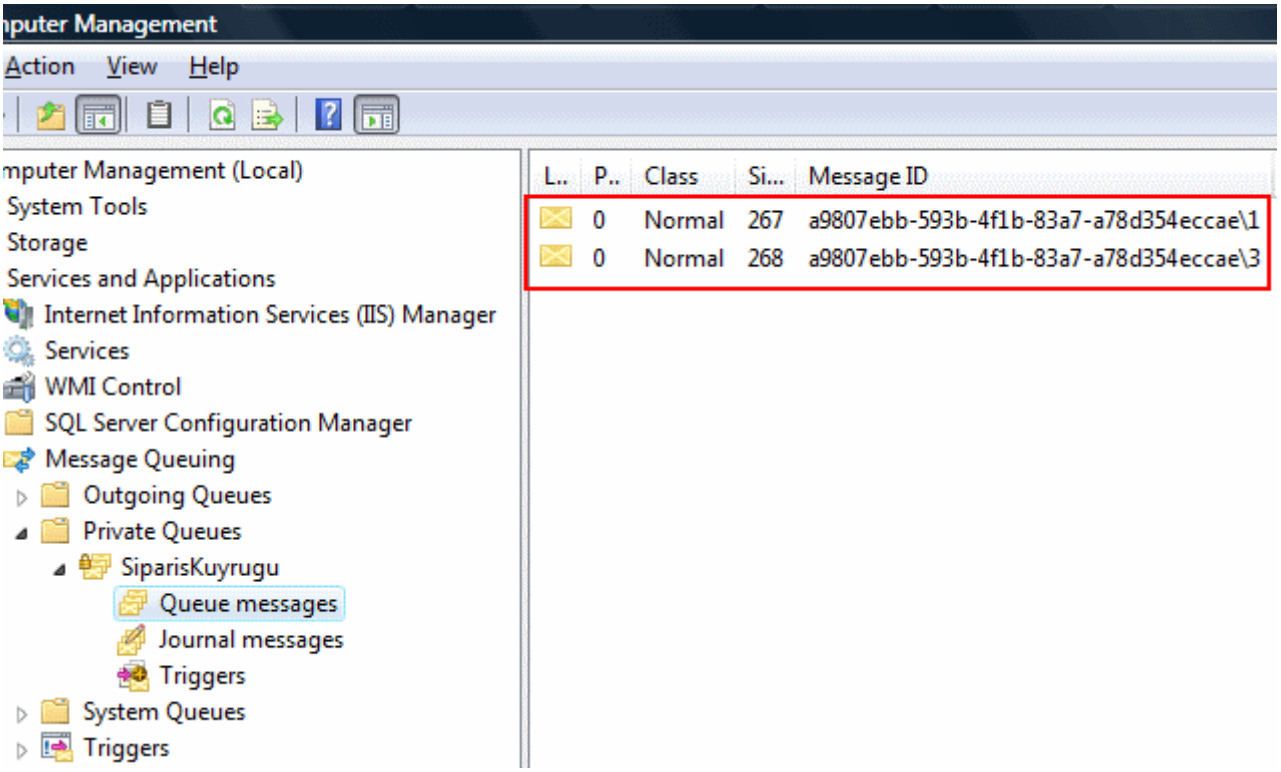
İstemci tarafı konfigurasyon dosyası;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <bindings>
         <netMsmqBinding>
            <binding name="SiparisIstemciBindingConfig" receiveTimeout="00:10:00"
maxRetryCycles="2" receiveErrorHandling="Fault" receiveRetryCount="5"
retryCycleDelay="00:30:00">
               <security mode="None" />
            </binding>
         </netMsmqBinding>
      </bindings>
      <client>
         <endpoint address="net.msmq://localhost/private/SiparisKuyrugu" binding="n
etMsmqBinding"
bindingConfiguration="SiparisIstemciBindingConfig" contract="SiparisServisi"name="
SiparisIstemciEndPoint">
         </endpoint>
      </client>
   </system.serviceModel>
</configuration>
```

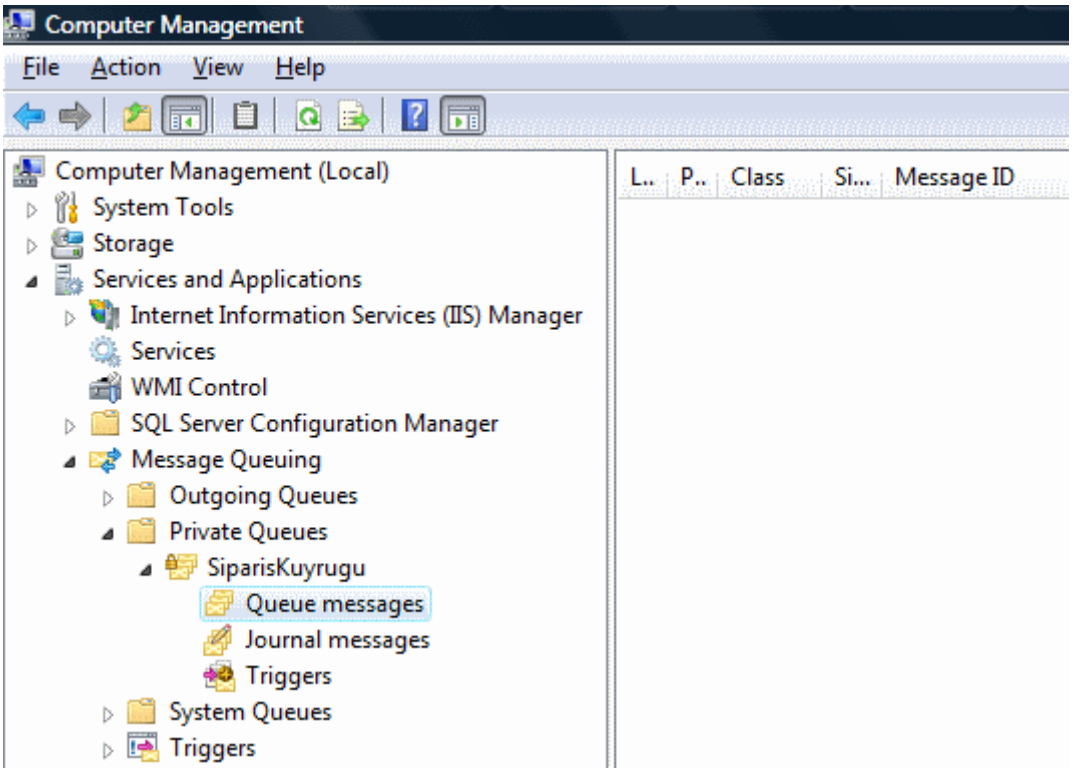
İstemci tarafı kodları;

```
using System;
using System.ServiceModel;

namespace Istemci
{
   class Program
   {
      static void Main(string[] args)
      {
         SiparisServisiClient client = new
SiparisServisiClient("SiparisIstemciEndPoint");
         Console.WriteLine("Başlamak için bir tuşa basınız...");
         client.SiparisEt(1, 100);
         client.SiparisEt(2, 100);
         Console.WriteLine("Kapatmak için bir tuşa basınız...");
         Console.ReadLine();
      }
   }
}
```

Gelelim test aşamasına. İlk olarak istemci uygulamanın çalıştırıldığı varsayılsın. Bu noktada servisin çalışmadığı düşünülebilir. Bu durumda uygulamadaki fonksiyonellikler çalışacak ve metod çağrılarına ait mesajlar kuyruğa yazılacaktır. Bu mesajlar **Microsoft Management Console** yardımıyla **SiparisKuyrugu** isimli kuyruktan aşağıdaki gibi görülebilir.



Görüldüğü gibi iki metod çağrısı sonrası oluşan mesajlar **SiparisKuyrugu** isimli kuyruğa atılmıştır. Bunların içeriklerinede bakılabilir. Bu içerikte tahmin edileceği gibi, mesajın hedef(Target) ve kaynak(Source) alıcıları, mesaj içeriği gibi bilgiler yer almaktadır. Bu noktadan sonra servis uygulaması tekrardan çalıştırılırsa kuyruktaki mesajların toplandığı ve işletildiği tespit edilebilir. Nitekim servis uygulaması çalıştırıldıktan sonra SiparisKuyrugu altında başka bir mesaj görünmeyecektir.



Buraya kadar anlatılanlar ile **WCF** mimarisinde basit olarak MSMQ bazlı uygulamaların nasıl geliştirilebileceği incelenmeye çalışılmıştır. **MSMQ**' nunda bazı dezavantajları elbetteki vardır. Herşeyden önce tek yönlü bir iletişimin olması, diğer iletişim tekniklerine göre daha yavaş çalışması(özellikle kuyruğun fiziki olarak tutulması nedeni ile) bu dezavantajlar arasında sayılabilir.

NOT : *MSMQ kullanılacağı yerlerde en iyi senaryoları elde edebilmek için bazı kriterlere dikkat etmek gerekebilir. Söz gelimi* ***hızlı ve yüksek performanslı bir mesajlaşma(Fast-Best Effort Queued Messaging)*** *için* ***transaction bazlı olmayan(Non-Transactional)*** *kuyruklar tercih edilmeli ve buna göre* ***ExactlyOnce*** *özelliği ile* ***Durable*** *özelliklerinin değeri* ***false*** *yapılmalıdır.*

Güvenilir olarak noktadan noktaya bir ***mesajlaşma söz konusu ise(Reliable End-To-End Queued Messaging)****, dört alternatif vardır.* ***BasicTransfer****,* ***Transactional Based****,* ***Dead-Letter Based*** *ve****Poison Message Based****. Bu alternatiflerden hangisinin kullanılacağına göre yine konfigurasyon elemanlarında bazı değişiklikler yapılmalıdır.*

WCF tabanlı olmayan uygulamalar ile haberleşebilmek için ***msmqIntegrationBinding*** *tipinin kullanmak gerekmektedir. Buda servisin* ***interoperability*** *desteği olmasını sağlamaktadır.*

Bu tip en iyi çözümler için ***http://msdn2.microsoft.com/en-us/library/ms731093.aspx*** *adresinden bilgi alınabilir.*

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde özet olarak **MSMQ** teknolojisini **WCF** içerisinde ele aldık. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

MSMQKullanimi.rar (60,00 kb)

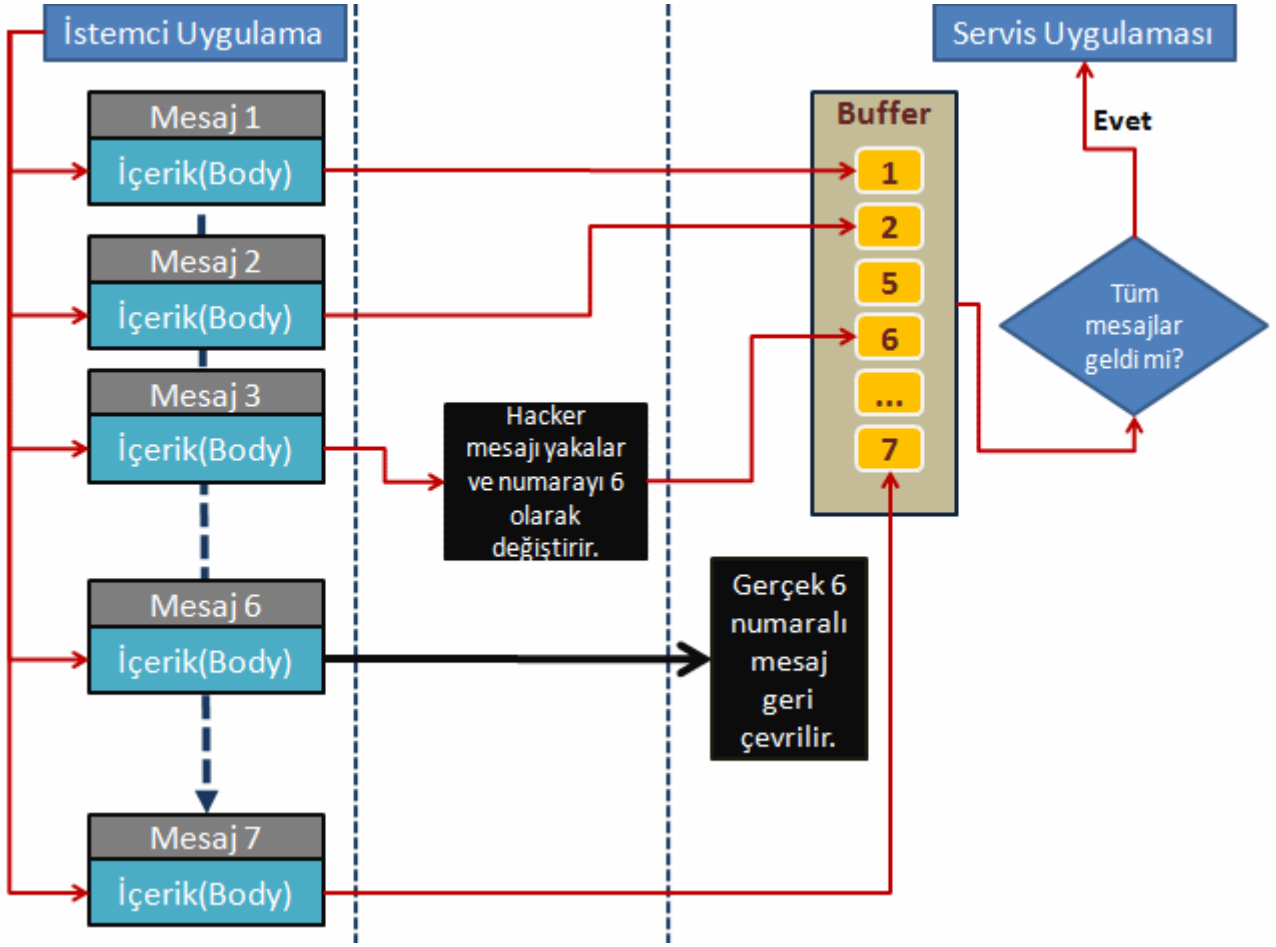
WCF - Replay Attack Etkisini Hafifletmek (2007-11-07T04:50:00)

wcf,

WCF(Windows Communication Foundation) ile ilgili bir önceki makalemizde, istemci ve servis arasında **güvenilir oturumların(Reliable Session)** nasıl açılabileceğinden bahsetmiştik. Güvenilir oturumların yararlarından biriside, olası **cevaplama saldırılarının(Replay Attacks)** önüne geçmektir. Bilindiği üzere cevaplama saldırılarında, istemci ve servis arasında hareket eden mesajların yakalanarak bozulması, değiştirilmesi, kesilmesi gibi problemler söz konusu olmaktadır. üstelik değişikliğe uğratılan mesajların zaman içerisinde her hangibir anda, orjinal servis kaynağına yada farklı bir yöne doğru defalarca gönderilmeleride söz konusudur.

Güvenilir oturumlarda, mesajların tekrar etmesini önlemek adına **WS-ReliableMessaging** şartnamelerine uygun şekilde hareket edilir. Buna göre mesajların

benzersiz bir id ile işaretlenmesi ve sıralarının belirlenmesi için numaralandırılması söz konusudur. Ne yazıkki güvenilir bir oturum cevaplama saldırılarını tek başına karşılamakta yeterli olmayabilir. Aslında bunun için geçerli ve yeterli bir senaryo vardır. Aşağıdaki şekil cevaplama saldırılarına ait bir vakayı ifade etmektedir.



Bu senaryoda istemci ve servis arasında hareket etmekte olan 7 adet mesaj olduğu varsayılmaktadır. Güvenilir bir oturum sağlandığı düşünülecek olursa, istemcinin servis tarafına göndereceği her mesajda bir sıra numarası olacağı söz konusudur. Ne varki senaryoya göre 3 numaralı mesaj **hacker** tarafından yakalanır. Sonrasında ise mesajın içeriği ve numarası 6 olarak değiştirilerek servis tarafına gönderilir. Bilindiği gibi güvenilir oturumlarda, servise gelen mesajların farklı zamanlarda ulaşma ihtimali göz önüne alınarak, bu mesajların istemciden gönderildikleri sırada alınmaları için **buffer(tampon)** sistemi kullanılır. İlerleyen zaman aralığında istemci gerçek 6 numaralı mesajı servis tarafında gönderir. Oysaki 6 numaralı mesaj zaten servis için açılan buffer içerisinde durmaktadır. Dolayısıyla asıl 6 numaralı mesaj servis tarafından geri çevrilerek işlenmeyecektir. İşte bu durum güvenilir oturumların cevaplama saldırıları için tek başına yeterli bir çözüm olmayışının ispatıdır.

çözümsel bir yaklaşım olarak **mesaj seviyesinde(Message Level)** veya **iletişim seviyesinde(Transport Level)** güvenlik göz önüne alınabilir. Ancak bunlarda tek başlarına yeterli değildir. Mesaj seviyesinde güvenlikte, istemci ve servis arasındaki mesaj bilgileri

şifrelenmektedir. Ne varki mesaj seviyesinde güvenlik uyarlamaları, çoğunlukla mesaj gövdesindeki içeriğin şifrelenmesi ile ilgilenir. Bu sebepten mesajın **başlık(Header)** kısmı yinede **hacker**' lar tarafından yakalanabilir ve sıra numaraları bozulabilir. Peki iletişim seviyesinde güvenlik uyarlamalarında durum nedir? Bu seviyede güvenlik oldukça güçlü bir çözümdür. Ancak iletişim seviyesinde güvenlik uyarlamaları, noktadan-noktaya çalışırlar. Bu konu için şöyle bir örnek düşünülebilir; bir servisin mesajını gönderdiği hedefte başka bir servis olabilir. Mesajı alan servisin, mesajların içeriğine sınırsız erişim yetkisi vardır. Eğer merkez servis aldığı mesajı başka bir servisede gönderiyorsa mesajı bozmadığından emin olmak gerekmektedir.

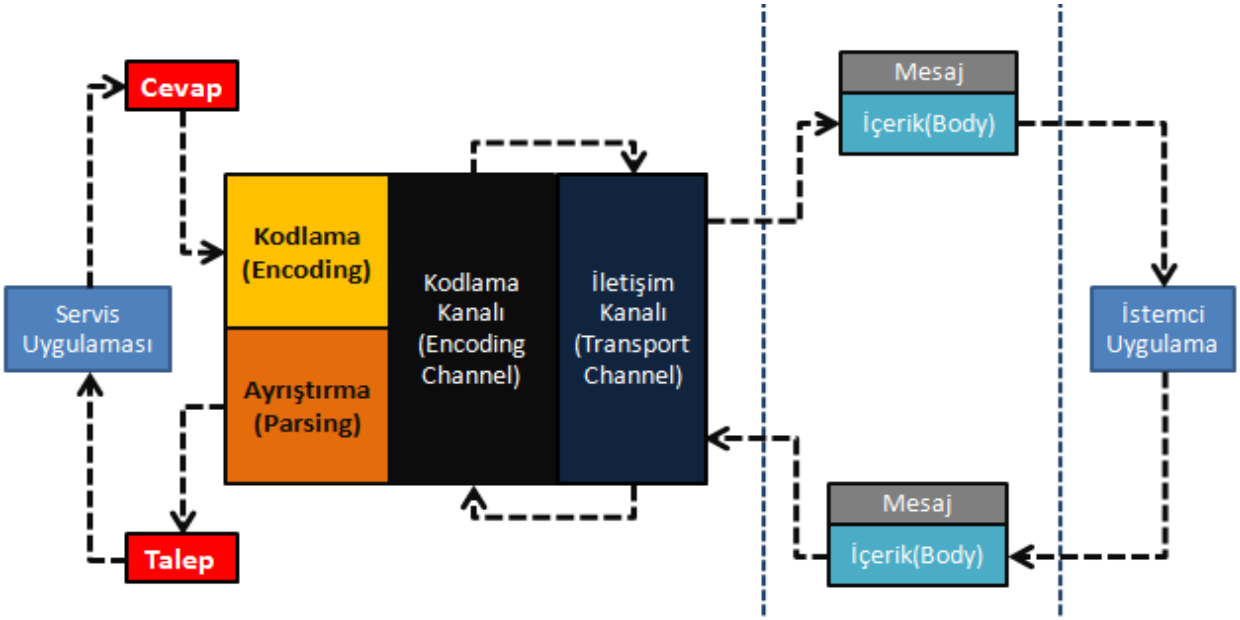
Bu ana kadar yazılıp çizilenler söz konusu olduğunda, cevaplama saldırılarına karşı tam anlamıyla bir çözüm bulunamadığı düşünülebilir. Lakin, WCF içerisinde özel bir takım teknikler yardımıyla cevaplama saldırılarına karşı daha güçlü durulabilir. Tabi tüm bu tedbirsel teknikler **güvenilir oturumlar(Reliable Session)** üzerine kurulmaktadır. Bu tedbirler aktifleştirildiğinde, **WCF çalışma zamanı(Run-Time)** her mesaj için aşağıdaki kriterlere uygun bir **tanımlayıcı(Identifier)** üretir.

- Benzersiz(Unique)
- Rastgele elde edilmiş(Random)
- İşaretlenmiş(Signed)
- Zaman Damgalı(TimeStamp)

Bu değerlerden oluşan **tanımlayıcılar(Identifiers)** nonce olarak isimlendirilmektedir. **Nonce**' ların kullanıldığı bu korunma sisteminin çalışma şekline göre, servise gelen mesajların bozulup bozulmadıkları**(Corrupt)** bir dizi işlem ile anlaşılmaya çalışılır. Normalde servis tarafına gelen mesajların nonce değerleri **tampona(buffer)** alınır. Başka bir mesaj geldiğinde başlık bilgilerinden elde edilen nonce değerinin, tamponda olup olmadığına bakılır. Elde edilen nonce değerinin aynısı tamponda var ise gelen mesaj geri çevrilir. Elbette nonce değeri tamponda yoksa mesaj kabul edilir. Kabul edilen bu mesajın nonce değeri yine tampona eklenir. Nonce' ların kullanıldığı bu sistemi tesis edebilmek için **WCF** içerisinde **özel bağlayıcı tiplerin(Custom Binding Types)** tanımlanması gerekmektedir.

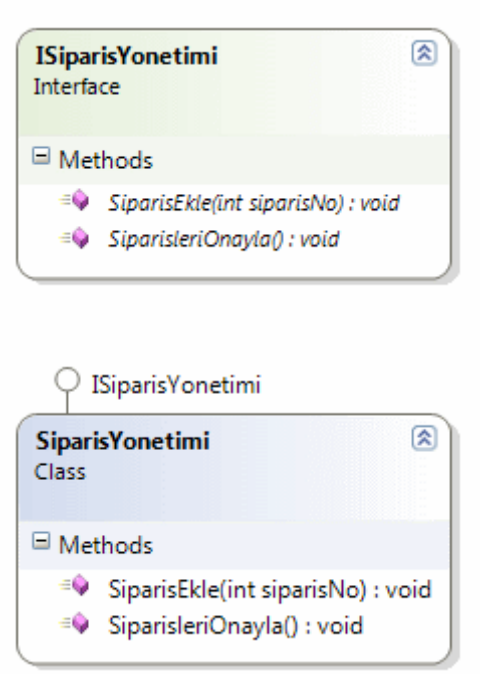
Windows Communication Foundation çalışma zamanı(WCF Run-Time) mesajların alınıp gönderilmesi sırasında **kanal yığınlarını(Channel Stack)** kullanır. Mesajlar normal şartlarda bir adrese doğru hareket ederler. Bu hareket **TCP**, **HTTP** gibi bir iletişim protokolü üzerinden aktarılırlar. Bu sebepten mesajın ulaştığı yerde bulunan WCF çalışma zamanı(WCF Run-time) karşılama için **iletişim kanallarını(Transport Channels)** kullanır. Bu kanal, servis tarafındaki host nesnesi açıldığı andan itibaren gelen mesajları dinlemektedir. Servis tarafında uygun iletişim kanalı tarafından alınan mesajlar, daha sonra **kodlama kanallarına(Encoding Channels)** yönlendirilir. Kodlama kanallarının görevi istemciden gelen mesajları çözümleyerek servis nesnesine iletmek ve servisin istemiceye göndereceği mesajlarıda kodlayarak **iletişim kanalına(Transport Channel)** iletmektir. çok doğal olarak iletişim kanalından geçerek mesaj kanalına gelen bilginin **Text** veya **Binary** tabanlı olma durumu (hatta WS şartnamelerine göre **Microsoft**

Transmision Opitimision Mechanism'a uygun bir biçimde olması) söz konusudur. Aşağıdaki şekil temel olarak söz konusu kanallar arasındaki iletişimi ifade etmektedir.



Servis tarafı göz önüne alındığında en azından iki kanalın var olması gerekmektedir. Dolayısıyla cevaplama saldırılarına karşı savunma yapılırken oluşturulacak olan**özel kanal tipleri(Custom Binding Types)** geliştirilirken bunlara dikkat edilmelidir. Diğer taraftan **WCF** içerisinde bu tip bağlayıcı tipleri hazırlamak son derece kolaydır. Basit olarak **Service Configuration Editor** bu amaçla kullanılabilir. Bu hazırlıklar yapılırken bağlayıcı tip içerisine katılan kanalların sırasıda önemlidir.

Artık bir örnek üzerinden hareket edilerek devam edilebilir. Her zamanki gibi **servis sözleşmesi(Service Contract)** ve uyarlamasını içeren bir **WCF Servis kütüphanesi** tasarlayarak işe başlamak gerekir. Söz konusu servis sözleşmesi içerisinde basit olarak işlevselliği çok önemli olmayan iki metod yer almaktadır. **Güvenilir oturum(Reliable Session)** sağlanması adına bir oturum var olma gerekliğine uygun olacak şekilde tanım yapılmaktadır. Söz konusu servis kütüphanesi içerisinde tanımlı tipler aşağıda görüldüğü gibidir.



Servis sözleşmesi(Service Contract);

```
using System;
using System.ServiceModel;

namespace SiparisKutuphanesi
{
   [ServiceContract(Name="Siparis Servisi",
Namespace="http://www.bsenyurt.com/SiparisServisi", SessionMode=
SessionMode.Required)]
   public interface ISiparisYonetimi
   {
      [OperationContract(IsInitiating=true)]
      void SiparisEkle(int siparisNo);
      [OperationContract(IsInitiating=false,IsTerminating=true)]
      void SiparisleriOnayla();
   }
}
```

Sözleşme Uyarlaması;

```
using System;
using System.ServiceModel;

namespace SiparisKutuphanesi
{
   [ServiceBehavior(InstanceContextMode= InstanceContextMode.PerSession)]
```

```
   public class SiparisYonetimi:ISiparisYonetimi
   {
      #region ISiparisYonetimi Members

      public void SiparisEkle(int siparisNo)
      {
         // Siparis ekleme adımı
      }

      public void SiparisleriOnayla()
      {
         // Siparis onaylama adımı
      }

      #endregion
   }
}
```

Bu işlemin ardından servis ve istemci taraflarının tasarlanmasına geçilebilir. Olayın basit bir şekilde ele alınması için servis ve istemci tarafındaki programlar basit birer **Console** uygulaması olarak tasarlanabilir. Bu uygulamalarda belkide en önemli kısımlar konfigurasyon ayarlarıdır. Nitekim konfigurasyon tarafında **özel bağlayıcı tip(Custom Binding Type)** tanımlamaları yapılacaktır. Servis tarafındaki Console uygulamasının başlangıçtaki hali aşağıdaki gibidir.

Servis uygulaması kod içeriği;

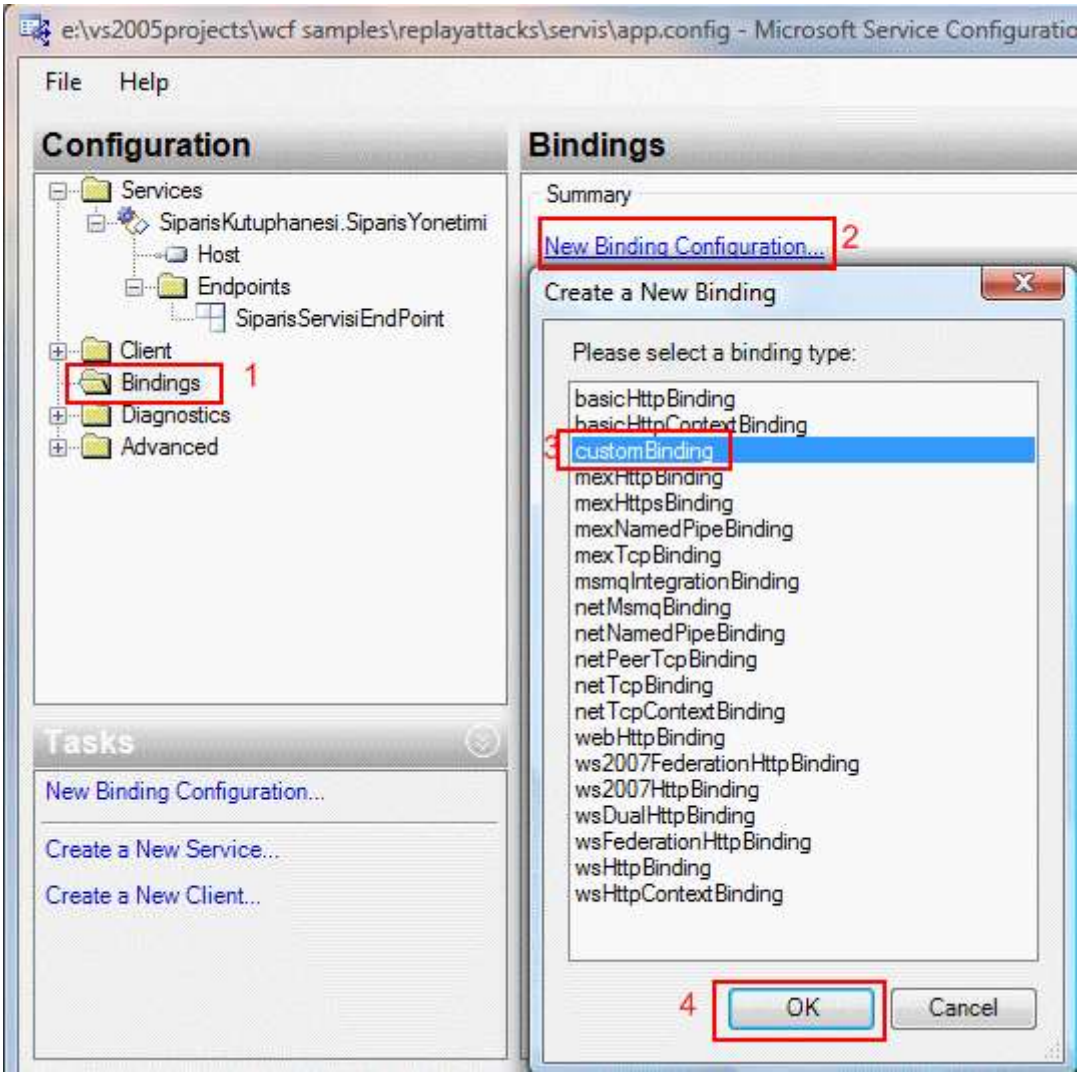
```
using System;
using System.ServiceModel;
using SiparisKutuphanesi;

namespace Servis
{
   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(SiparisYonetimi));
         host.Open();
         Console.WriteLine(host.State.ToString());
         Console.WriteLine("Kapatmak için bir tuşa basınız");
         Console.ReadLine();
         if (host.State == CommunicationState.Opened)
         host.Close();
      }
```

```
    }
}
```

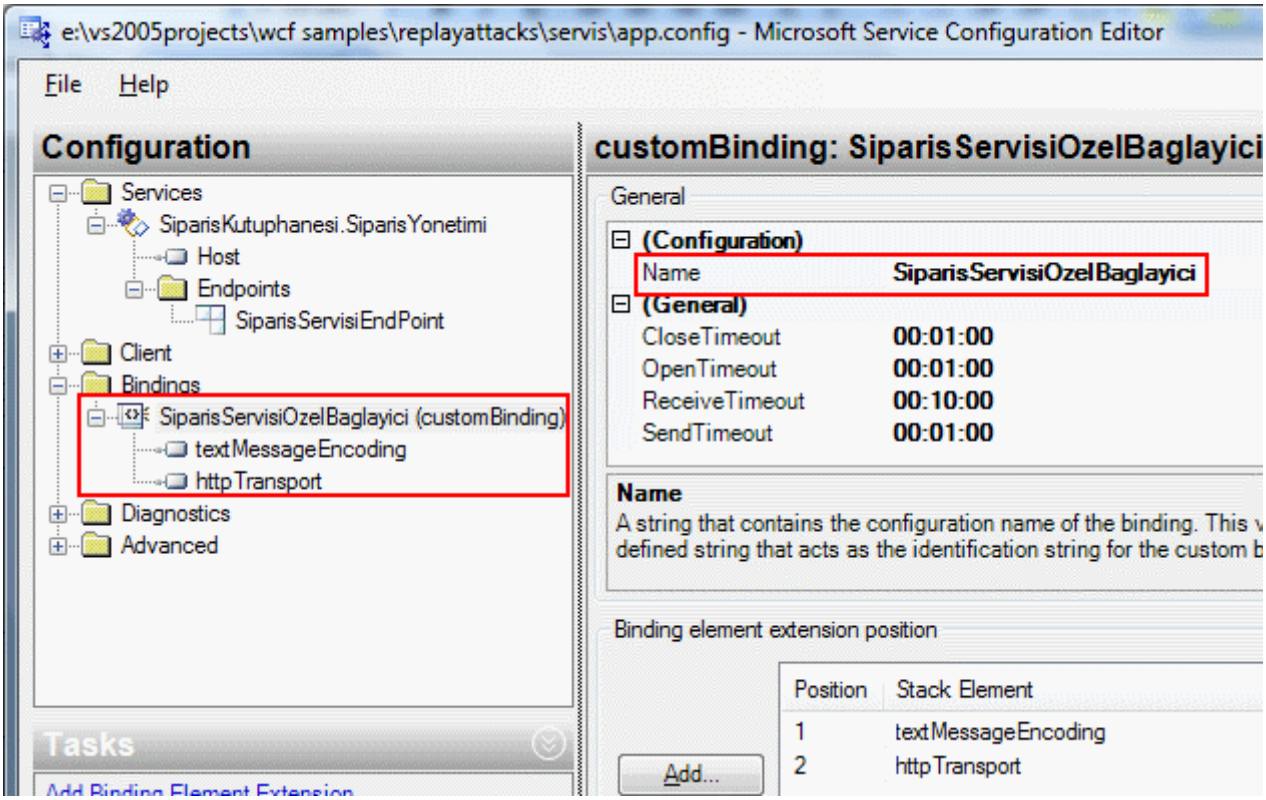
Servis tarafındaki konfigurasyon dosyasının ilk hali;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="SiparisKutuphanesi.SiparisYonetimi">
        <endpoint
address="net.tcp://localhost:4500/SiparisServisi.svc" binding="netTcpBinding" name="
SiparisServisiEndPoint" contract="SiparisKutuphanesi.ISiparisYonetimi"/>
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

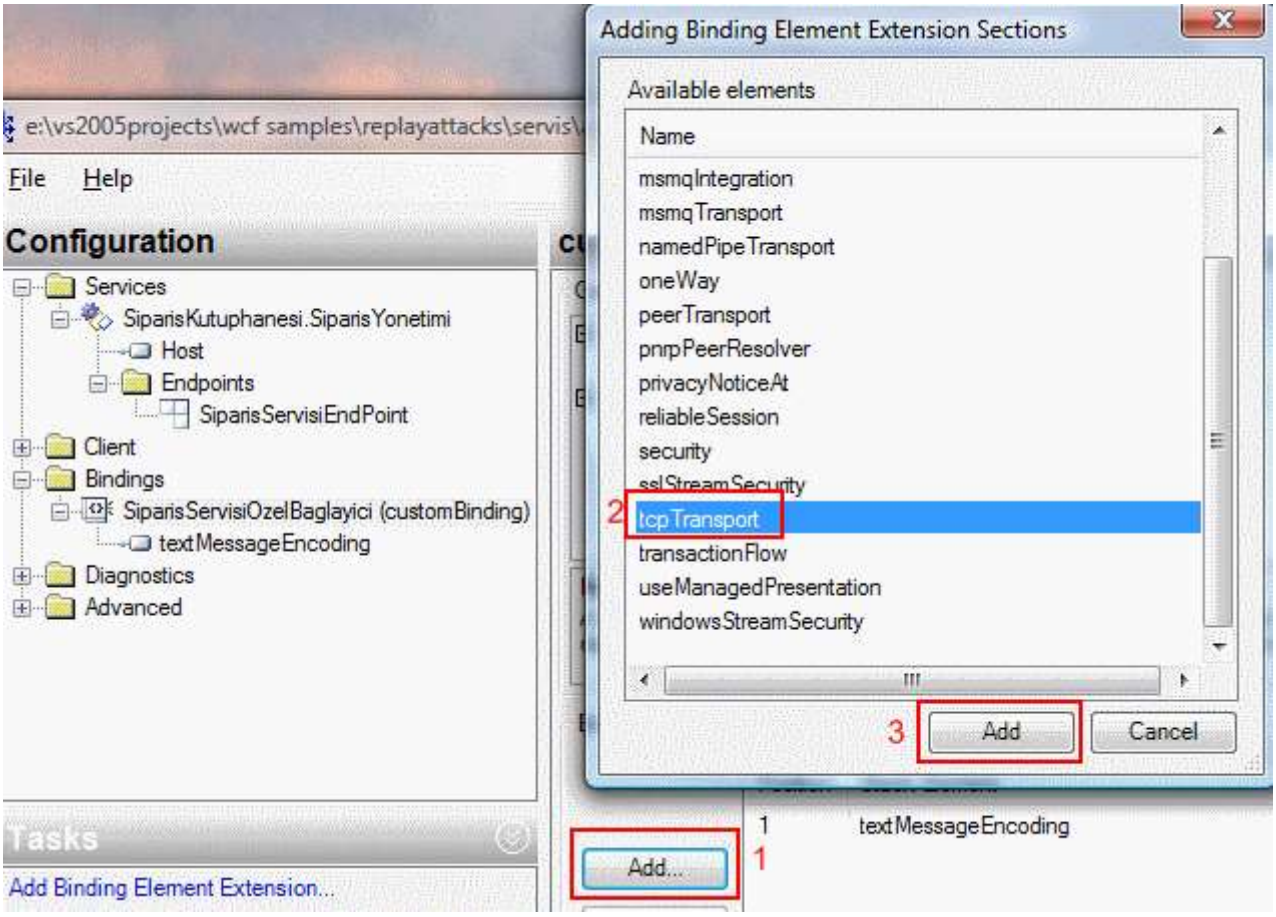
Başlangıçta servis tarafı **netTcpBinding** bağlayıcı tipini kullanacak şekilde tasarlanmıştır. Ancak cevaplama saldırılarının önüne kesmek için burada özel bir bağlayıcı tip tasarlanacaktır. Şimdi adım adım bu işlemler gerçekleştirilecektir. İlk olarak, **Service Configuration Editor** üzerinden aşağıdaki ekran görüntüsünde yer aldığı gibi **New Binding Configuration** bağlantısına tıklanır.



Sonrasıdan açılan **Create a New Binding** kısmından **customBinding** seçilerek **OK** tuşuna basılır. Bu işlemin sonrasında oluşan duruma göre özel bağlayıcı tipin adı, aşağıdaki ekran görüntüsünde olduğu gibi SiparisServisiOzelBaglayici olarak değiştirilebilir.

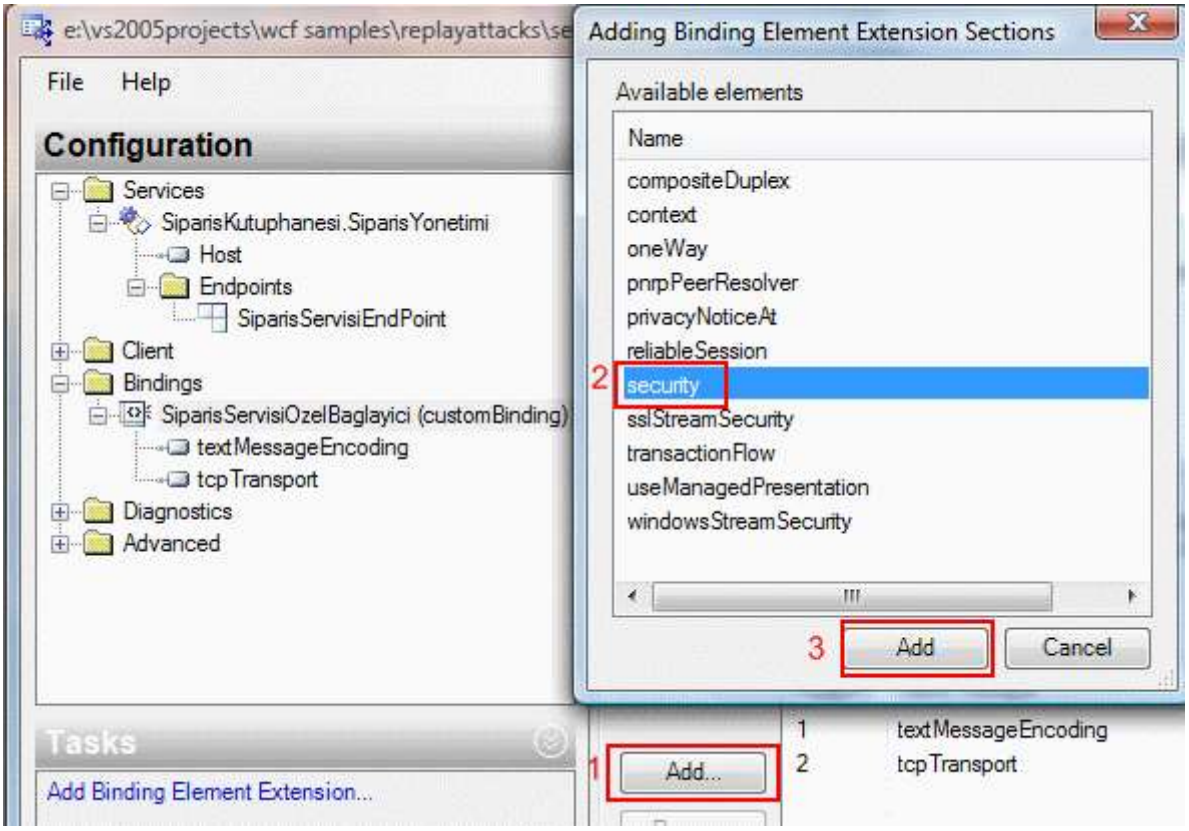


İlk bakıldığında iletişim kanalı olarak **httpTransport** tipinin ve mesajlaşma kanalı olarakta **textMessageEncoding**' in kullanıldığı görülmektedir. örnekte **TCP** protokolü üzerinden bir haberleşme hedeflendiği için **httpTransport** kanalı kaldırılmalıdır. Bu işlem için aynı ekranda, **httpTransport** seçili iken **Remove** tuşuna basılması yeterlidir.**tcpTransport** kanalının eklenmesi için aşağıdaki ekran görüntüsünde işaretlenen adımların sırasıyla yapılması yeterlidir.

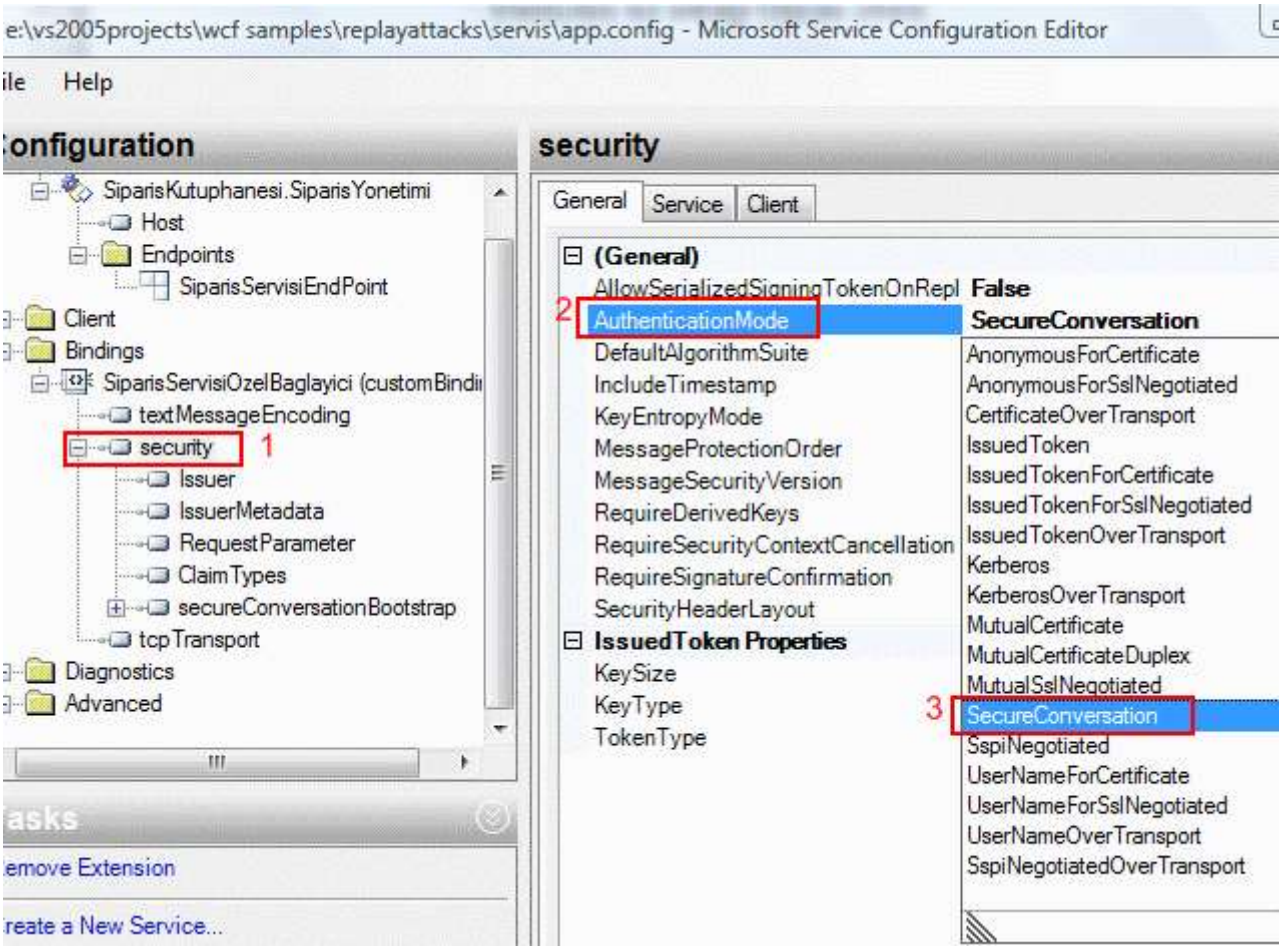


Dikkat edilecek olursa **Available Elements** kısmında, özel bağlayıcı tip içerisinde kullanılabilecek pek çok kanal çeşidi bulunmaktadır. **tcpTransport** tahmin edileceği üzere bir iletişim kanalı olarak servise gelen mesajların TCP protokolü üzerinden alınmasını sağlamaktadır. örnekte mesajlaşma kanalı olarak **textMessageEncoding** kullanılmaktadır. Bu nedenle oluşturulan özel bağlayıcı tip içerisinde gelen ilgili kanalın kaldırılmasına gerek yoktur.

Sıradaki adımda güvenlik ile ilgili bir kanalın eklenmesi gerekmektedir. Bu amaçla yine **Add** düğmesi ile(yada **Add Binding Element Extension** bağlantısı kullanılarak) açılan pencereden security elementi seçilmelidir.



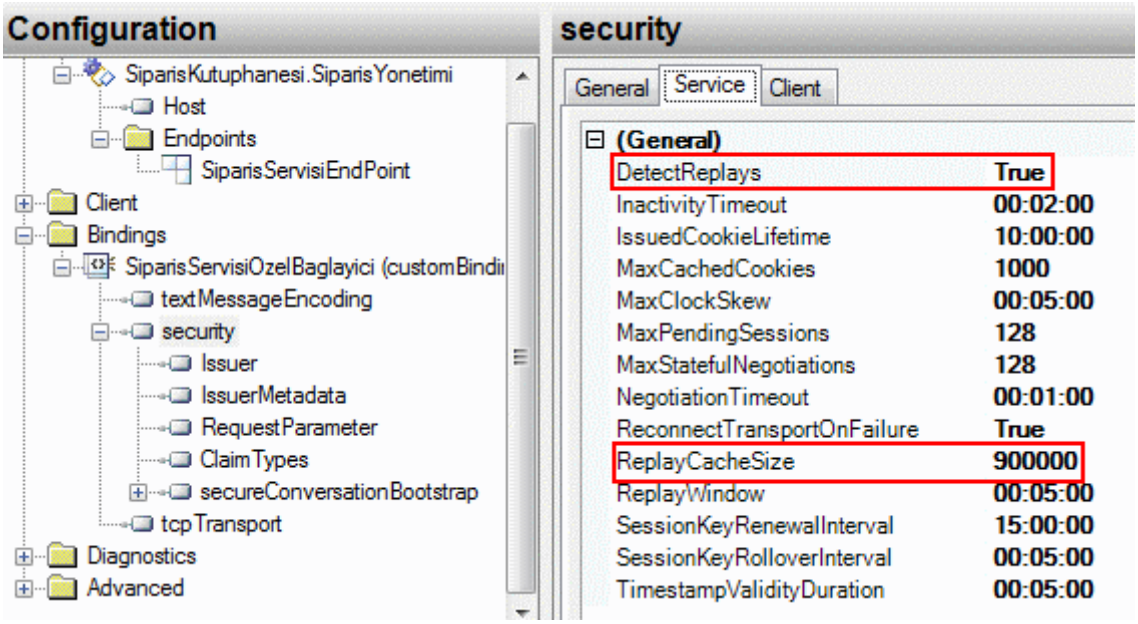
security elementi içerisinde ise **AuthenticationMode** özelliğinin değeri aşağıdaki ekran görüntüsünde olduğu gibi **SecureConversation** olarak ayarlanır.



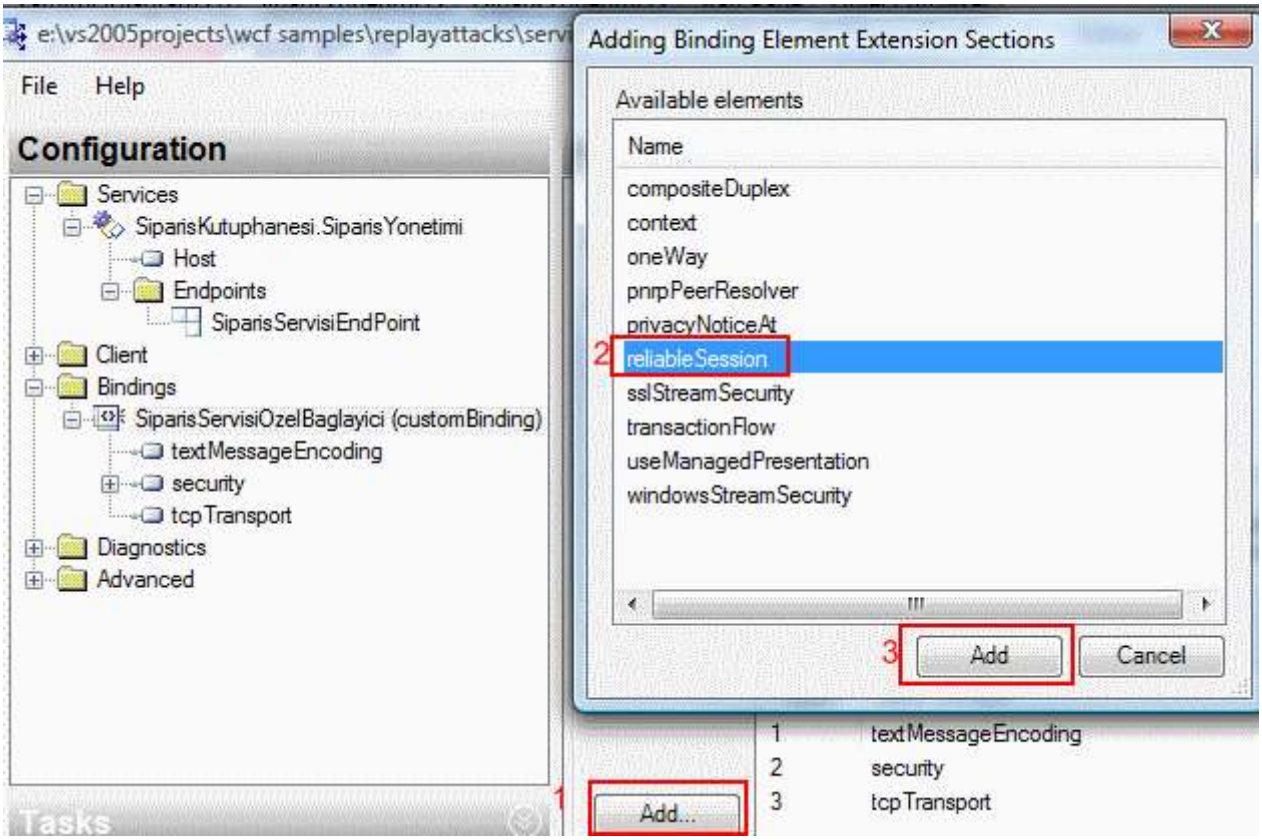
SecureConversation, **Organization for the Advancement of Structured Information Standards** (**OASIS** - http://www.oasis-open.org/home/index.php) tarafından kabul edilmiş olan **WS-SecureConversation**şartnamelerine uygun olacak şekilde güvenli bir oturumun sağlanması garanti etmektedir.

özet olarak WS-SecureConversation, iki katılımcı arasındaki(örnek senaryoya göre istemci ve servis) mesajlaşmada ehliyet bilgilerinin tamamının gönderilmesini gerektirmeyecek bir ortam sağlamaktadır. Bunun sağlanabilmesi için oturumun en başında, istemci ve servis arasında **ehliyet(Credential)** bilgileri değiş tokuş edilir ve doğrulanır. Geri kalan mesajlaşmalarda başlangıçtaki ehliyet bilgilerinden türeyen **güvenlik fişleri(security tokens)** kullanılır. Bir başka deyişle oturum başında zaten taraflar ehliyet bilgileri ile birbirlerini doğruladıklarından, kalan mesajlaşmalarda aynı bilgiler tekrardan kontrol edilmez. Buda çok doğal olarak mesajlaşmanın daha hızlı gerçekleştirilmesini sağlamaktadır.

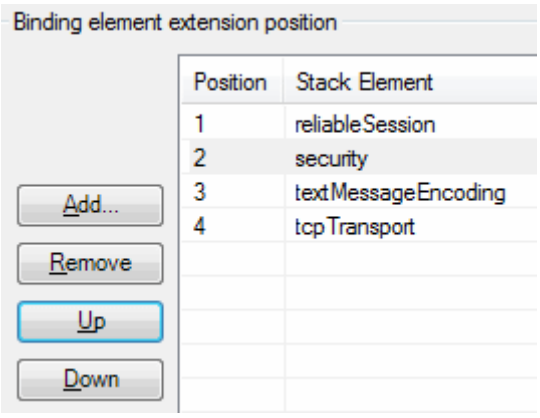
SecureConversation seçildikten sonra, yine security elementinde servis tarafına yönelik olacak şekilde bazı ayarları kontrol etmek gerekmektedir. İlk olarak aşağıdaki ekran görüntüsünde görüldüğü gibi **DetectReplays** seçeneğinin **true** olması sağlanmalıdır ki varsayılan olarak böyledir.



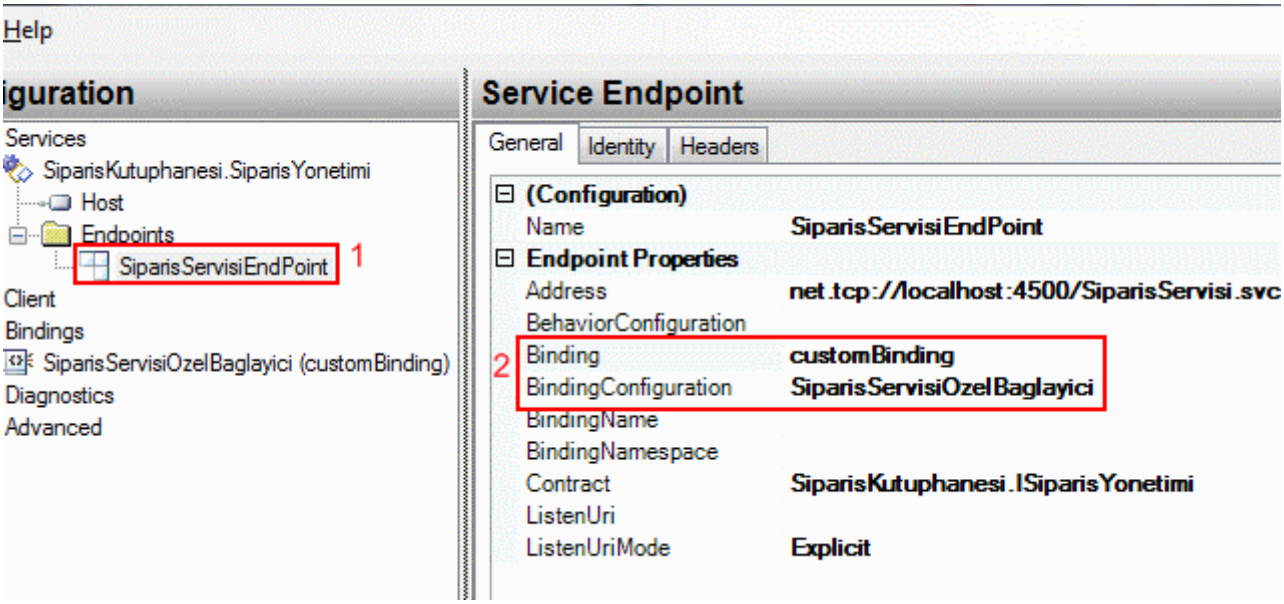
Bu kısımda oldukça fazla sayıda ayar ve kafa karıştırıcı özellik yer almaktadır. Ancak en çok dikkat çekenlerden birisi **ReplayCacheSize** değeridir. Buraya atanan değer, tamponda tutulacak olan nonce' ların sayısıdır. Söz konusu değerin dışına taşılması durumunda, tamponda duran en eski nonce değeri atılacak ve yerine son gelen nonce değeri yazılacaktır. Ancak varsayılan **900000** değeri pek çok vaka için yeterli bir sayıdır. Bellek optimizasyonu adına bu değerin azaltılmasıda düşünülebilir. Fakat az öncede belirtildiği gibi, sayının dışına çıkılması halindeki durumlar göz önüne alınmalıdır. Nitekim eski **nonce**' ların silindiği ve yeni gelenlerin yazıldığı sıralarda sistem cevaplama **saldırılarına(Replay Attack)** karşı kısa sürelide olsa savunmasız kalabilir. Bu işlemlerin ardından elbetteki bağlayıcı tipin **güvenilir bir oturum(Reliable Session)** açabilmesi için, yine **Add**düğmesi ile açılan pencereden **ReliableSession** elementi seçilmelidir.



Artık özel bağlayıcı tip içerisinde kullanılacak olan tüm kanal ve özellikler belirlenmiştir. Son aşamada dikkat edilmesi gereken nokta söz konusu kanalların uygulanış sırasıdır. Bir başka deyişle **kanal yığını(Channel Stack)** içerisindeki sıranın önemi vardır. Yukarıda geliştirilen örneğe göre sıra aşağıdaki ekran görüntüsündeki gibi olmalıdır. Buna göre reliableSession ile başlayan sıra security, textMessageEncoding(Mesajlaşma Kanalı) ve tcpTransport(iletişim kanalı) şeklinde devam etmelidir. Bu sırayı ayarlamak için **Up** ve**Down** başlıklı düğmeler kullanılabilir.



Elbette oluşturulan bu **özel bağlayıcı tipin(Custom Binding Type)** kullanılabilmesi için **endPoint** ile ilişkilendirilmesi gerekmektedir. Bu sebepten tek yapılması gereken **SiparisServisiEndPoint**isimli **endPoint** seçili iken, **Binding** özelliğinin değerinin **customBinding** olarak işaretlenmesidir.

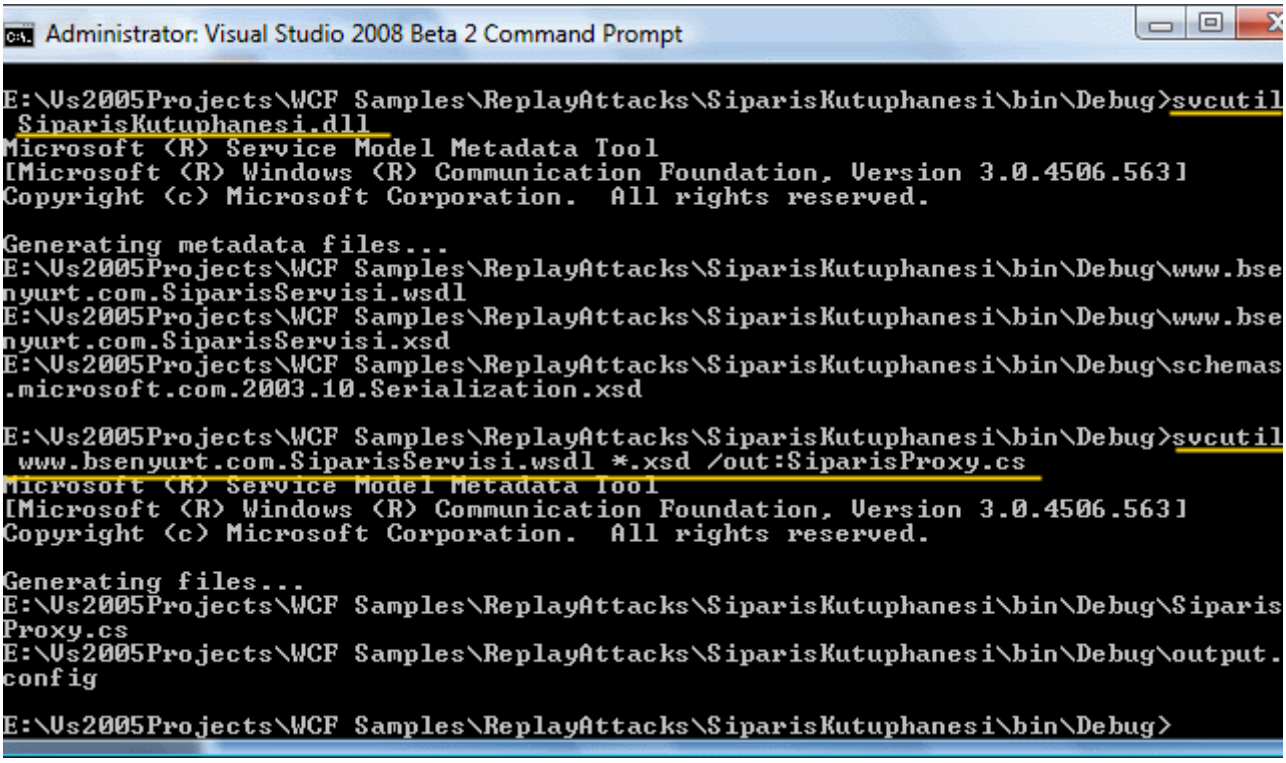


Bu işlem sırasında **Binding** değeri **customBinding** olarak seçildiğinde **BindingConfiguration** özelliğinin değeri otomatik olarak **SiparisServisiOzelBaglayici** olarak değişecektir. Servis tarafında yapılan bu ayarlardan sonra konfigurasyon dosyasının son hali aşağıdaki gibi olacaktır.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <customBinding>
        <binding name="SiparisServisiOzelBaglayici">
          <reliableSession />
          <security authenticationMode="SecureConversation">
            <secureConversationBootstrap />
          </security>
          <textMessageEncoding />
          <tcpTransport />
        </binding>
      </customBinding>
    </bindings>
    <services>
      <service name="SiparisKutuphanesi.SiparisYonetimi">
        <endpoint
address="net.tcp://localhost:4500/SiparisServisi.svc" binding="customBinding" binding
Configuration="SiparisServisiOzelBaglayici" name="SiparisServisiEndPoint"
contract="SiparisKutuphanesi.ISiparisYonetimi" />
      </service>
    </services>
```

```
    </system.serviceModel>
</configuration>
```

çok doğal olarak burada yapılan konfigurasyon değişikliklerinin istemci tarafındaki uygulamada da yapılması gerekmektedir. Ama öncesinde istemci için gerekli **proxy** sınıfının **svcutil**aracı yardımıyla aşağıdaki gibi üretilmesi sağlanmalıdır.



Sonrasında ise bu proxy sınıfı **Console** tipinden tasarlanan istemci uygulamaya taşınarak kullanılır. İstemci uygulamanın kodları ve konfigurasyon dosyasının içeriği ise aşağıdaki gibidir.

İstemci uygulama kodları;

```
using System;
using System.ServiceModel;

namespace Istemci
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Sipariş vermek için tuşa basın...");
            Console.ReadLine();
            SiparisServisiClient cli = new SiparisServisiClient("IstemciEndPoint");
            cli.SiparisEkle(1);
```

```
            cli.SiparisEkle(4);
            cli.SiparisleriOnayla();
            Console.WriteLine("İşlemler tamamlandı...çıkmak için bir tuşa basınız");
            Console.ReadLine();
        }
    }
}
```

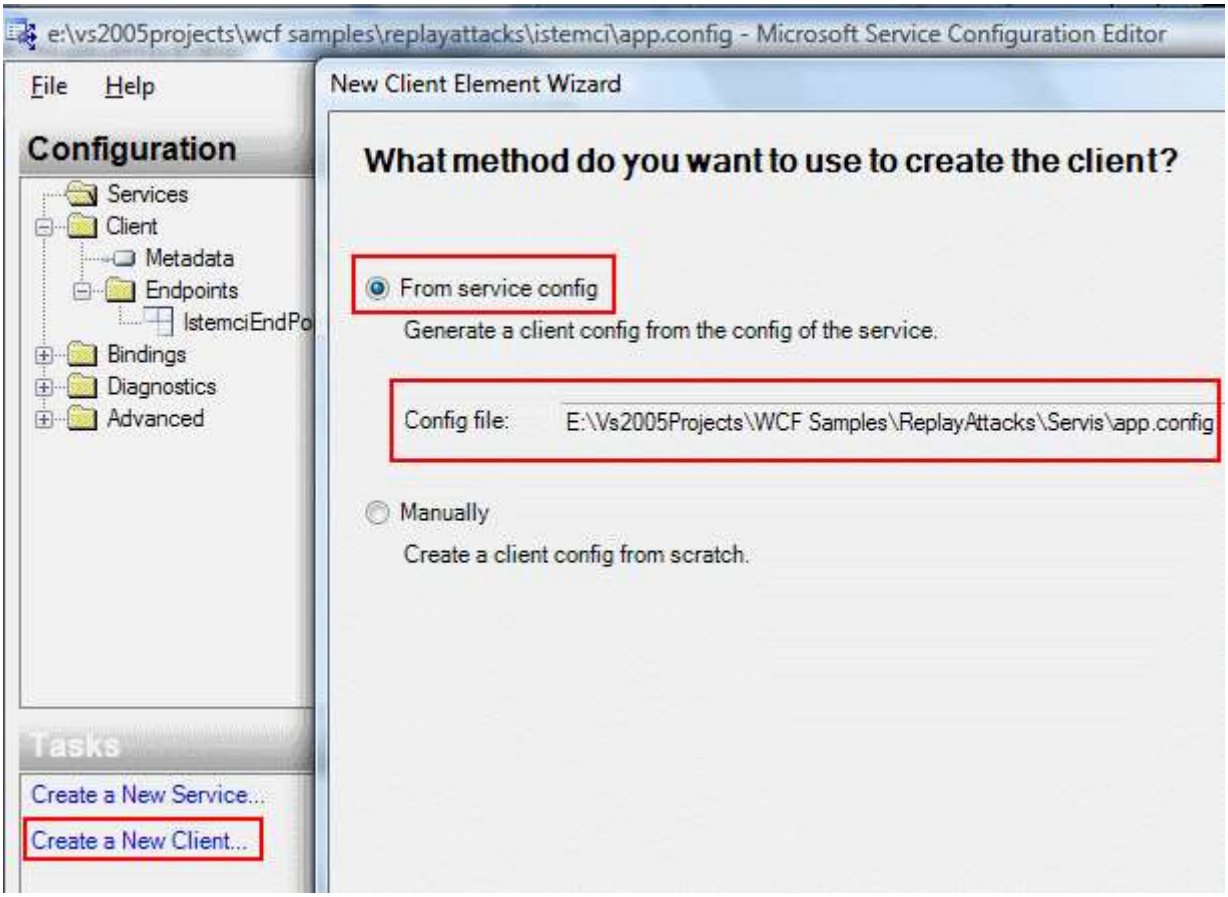
İstemci uygulama basit olarak servis üzerinden SiparisEkle ve SiparisleriOnayla metodlarını çağırmaktadır.

İstemci tarafı konfigurasyon dosyası;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.serviceModel>
        <bindings>
            <customBinding>
                <binding name="SiparisIstemciOzelBaglayici">
                    <reliableSession />
                    <security authenticationMode="SecureConversation">
                        <secureConversationBootstrap />
                    </security>
                    <textMessageEncoding />
                    <tcpTransport />
                </binding>
            </customBinding>
        </bindings>
        <client>
            <endpoint address="net.tcp://localhost:4500/SiparisServisi.svc"
binding="customBinding" bindingConfiguration="SiparisIstemciOzelBaglayici"
contract="SiparisServisi" name="IstemciEndPoint">
            </endpoint>
        </client>
    </system.serviceModel>
</configuration>
```

İstemci tarafındaki konfigurasyon dosyasının daha kolay üretilmesi amacıyla, istemci uygulamaya standard bir **app.config** dosyası eklendikten sonra **Service Configuration Editor** yardımıyla açılan pencerede **Create a New Client** sonrası açılan **New Client Element Wizard** bölümü kullanılabilir. Burada **From service config** seçili iken **Config File** kısmına servis uygulamasındaki konfigurasyon dosyasını işaret etmek yeterlidir. Yanlız burada **proxy** sınıfı kullanıldığı için **sözleşme arayüzü(Contract Interface)** adının servis tarafındaki gibi olmadığı unutulmamalıdır. Bu nedenle bu üretim sonrasında oluşan konfigurasyon içerisinde **endPoint** elementinde yer alan

binding **niteliğinin(attribute)** değeri uygun şekilde değiştirilmelidir. örnekte bu değer SiparisServisi olarak değiştirilmelidir.



Gelinen bu noktadan sonra sistem test edilebilir. Söz konusu sistem cevaplama saldırılarına karşı önlem alan özel bir bağlayıcı tipi kullanmaktadır. örnek uygulamalarda yine logların izlenmesi işlemi gerçekleştirilirse mesajların içeriğinde yazının başındada belirtilen tanımlama değerlerinin yer aldığı (**timestamp** gibi...), bununla birlikte mesajların boyutlarının dahada arttığı görülür. Mesaj boyutlarındaki bu artış çok doğal olarak paketlerin büyümesi anlamınada gelmektedir. Ancak vaka içerisinde **Replay Attack** olasılığı var ise bu göz ardı edilmeli ve gereken tedbirler alınmalıdır. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

ReplayAttacks.rar (96,50 kb)

WCF - Güvenilir Oturumlar(Reliable Sessions) (2007-11-01T04:57:00)

wcf,

WCF(Windows Communication Foundation) bilindiği üzere bir **servis yönelimli mimari(Service Oriented Architecture)** yaklaşımıdır. Buda basitçe, birbirleriyle haberleşen **istemci(client)** ve**sunucu(server)** uygulamaların var olması anlamına

gelmektedir. Bu haberleşme çok doğal olarak bir **ağ(network)** ortamı üzerinde gerçekleşir. Ağ ortamı intranet gibi bir sistem olabileceği gibi kablolu veya **kablosuz(wireless)** bir internet ortamıda olabilir. Hal böyle olunca arada hareket etmekte olan mesajların güvenliği önem arz eden bir konudur. Mesaj güvenliğinden kasıt sadece şifreleme yada sertifikalı bir iletişimin sağlanması demek değildir. Bunların sağlanması için WCF mimarisi içerisindede çeşitli teknikler bulunmaktadır. Bu teknikler bir yana istemci ve sunucu(servis) arasında **güvenilir bir oturumun(reliable session)** var olması gereken durumlarda söz konusudur. Güvenilir bir oturum sağlanması için gereken sebepler arasında aşağıdaki maddeler göz önüne alınabilir;

- Ağ ortamında istemci ve servis arasındaki bağlantının kopması olasıdır.
- Arada hareket etmekte olan mesajlar kesintiye uğrayabilir.
- İstemciden servis tarafına gelmekte olan mesajlar farklı yollar üzerinden hedeflerine ulaşmaktadır. Böyle bir durumda servise farklı sıralarda ulaşmaları söz konusudur. Ancak mesaj sırası önemli olabilir.
- Mesajlar beklenmedik bir şekilde kaybolabilir yada farklı bir yere doğru yönlendirilebilir.

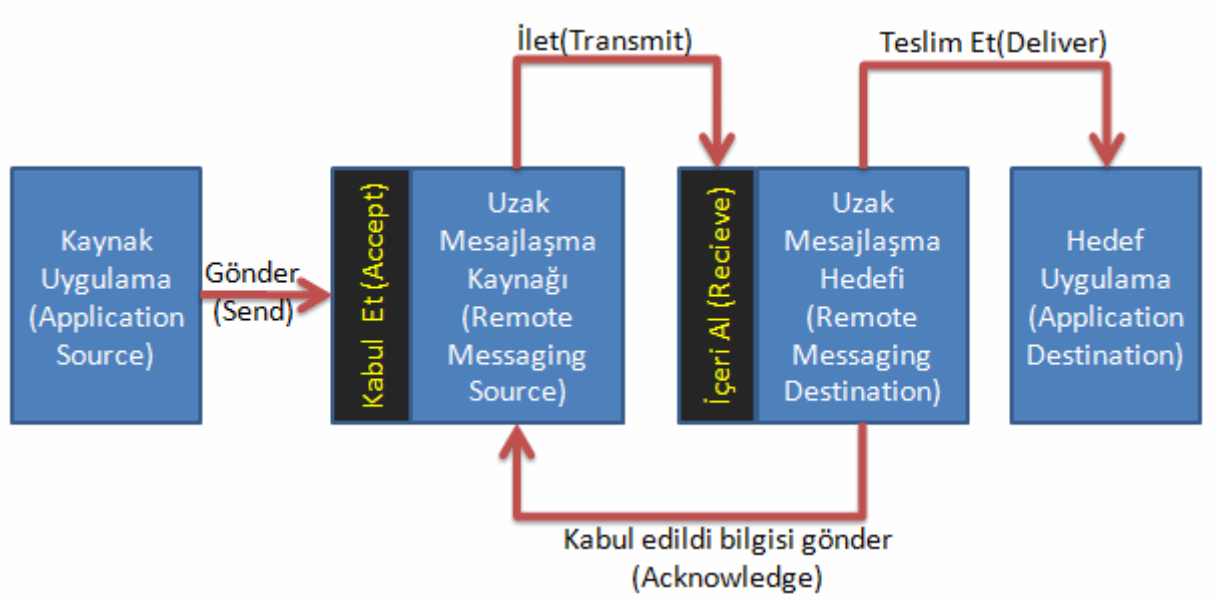
Bu seçenekler göz önüne alındığında istemci ve sunucu arasında güvenilir bir oturum açılma ihtiyacı daha belirgin bir şekilde ortaya çıkmaktadır. Nitekim arada hareket eden mesajların üçüncü kişiler tarafından yakalanması özellikle **cevaplama saldırıları(Reply Attacks)** ile çok farklı amaçlarla kullanılabilir. Bunun için örnek olarak bir alışveriş sitesinin işleyişi ele alınabilir. üçüncü şahıs, bir sipariş bilgisine ait mesajı yakalayıp defalarca sunucuya işlenmesi için gönderebilir. Bunun sonucunda alışverişi yapan gerçek kişi, hiç yapmamış olduğu siparişlerle karşı karşıya kalacaktır. üstelik üçüncü şahıs, bu mesajları istediği zaman gönderme imkanına sahip olabilir.

Cevaplama saldırılarının(Reply Attack) dışında istemcilerin göndereceği taleplere ait mesajların servis tarafında, istemciden gönderildiği sırada ele alınması gerekebilir ki buda güvenilir oturumların sağlanması için yeterli nedenlerdendir. Windows Communication Foundation mimari alt yapısı içerisinde **güvenilir oturumlar(Reliable Sessions)** açılabilmesi için gereken özellikler yer almaktadır. Nitekim güvenilir oturumların açılabilmesi için gerek ve yeter şart **WS-ReliableMessaging** protokolüne uygun bir ortamın sağlanmış olmasıdır. WCF bu protokolü doğrudan destekleyen çeşitli **bağlayıcı tipler(bindiny types)** içermektedir.

NOT :* WS-ReliableMessaging**, *Web servislerine yönelik olarak geliştirilmiş platform bağımsız pek çok standarttan sadece bir tanesidir. Toplu olarak* **WS- *şeklinde ifade edilen tüm Web servisi standartları için http://en.wikipedia.org/wiki/List_of_Web_service_specifications adresinden bilgi alınabilir.*

WS-ReliableMessaging, tek bir kaynak(Source) ve tek bir hedef(Target) arasında güvenilir bir mesajlaşma için gereken şartnameleri içeren **Organization for the Advancement of Structured Information Standards** (**OASIS** - http://www.oasis-

open.org/home/index.php) organizasyonu tarafından kabul edilmiş bir protokoldür. Son olarak 14 Haziran 2007 tarihinde 1.1 versiyonu yayınlanmıştır. WS-ReliableMessaging standardının amacı; güvenilir olmayan bir altyapı(Unreliable Infrastructure) üzerinde koşan bir kaynak uygulamadan hedef uygulamaya doğru güvenilir bir şekilde mesaj gönderilmesini sağlamaktır. Mesajın içeriğinin şifrelenmesi veya iletişim kanalının güvenli hale getirilmesi(örneğin **Secure Socket Layer** ile) konuları ile ilgilenmez. Bu**şartnameye(Specification)** göre güvenilir oturumlarda söz konusu olan mimari model aşağıdaki şekilde olduğu gibidir.



Şekilde kaynak uygulama ile hedef uygulama arasındaki bir mesajlaşma trafiği yer almaktadır. Burada Uzak Mesajlaşma Kayanağı mesajı gönderirken **WS-ReliableMessaging** kullanır. Diğer taraftan burada tek bir kaynak ve tek bir hedef vardır. WCF(Windows Communication Foundation) bu protokolün kullanımına destek vererekten aşağıdaki maddelerde görülen kazanımları sağlamaktadır;

- Kaynaktan gönderilen tüm mesajların hedefe varması garantilenir.
- Kaynaktan hedefe gönderilen mesajların tekrar edilmesi önlenir. Bir başka deyişle mesajın sadece bir tane gönderilmesi garanti edilir.
- Kayıp mesajlar tespit edilir ve mümkünse bunlarında kaynaktan hedefe doğru yeniden gönderilmesi sağlanır.
- Kaybolan mesajların geri alınmayacak durumda olması halinde **istisna(exception)** fırlatılması sağlanır.
- Opsiyonel olarak mesajların gönderildikleri sırada işlenmeleri garantilenir. WCF bu amaçla **tampon(buffer)** sistemini kullanılır. Buna göre tüm mesajlar tamponda toplanır ve gönderildikleri sıraya göre hedef tarafında işlenir.

WS-ReliableMessaging, mesajların servis tarafında gönderildikleri sırada ele alınmalarını sağlayan **şartnameler(Specifications)** sunmasına rağmen gerçek

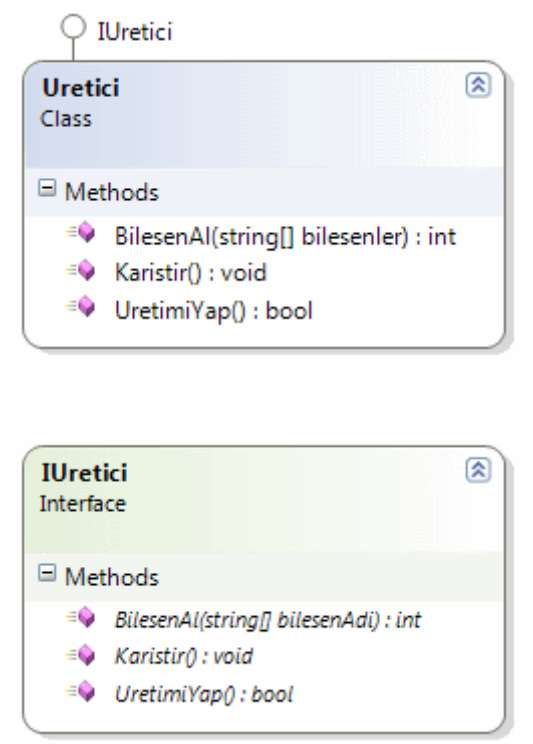
anlamda, **MSMQ(MicroSoft Messaging Queing)** sisteminde olduğu gibi bir mesaj kuyruğu yapısı bildirmez. **MSMQ** bunun için farklı bir form kullanır.

NOT : *WS-ReliableMessaging, Windows Communication Foundation dışında* **BEA WebLogic, IBM WebSphere, Apache Sandesha** *gibi sistemler tarafındanda ele alınmaktadır.*

WCF mimarisinde aslında söz konusu protokolün sağlanması için gereken teş şey kaynak ve hedef uygulamaların aynı zaman dilimi içerisinde çalışıyor olmalarıdır. WCF sistemi içerisinde yer alan bağlayıcılardan **basicHttpBinding**, **netNamedPipeBinding**, **netPeerTcpBinding** tipleri **güvenilir oturumları(Reliable Sessions)** desteklememektedir. Bununla birlikte**wsDualHttpBinding** tipi için güvenilir oturumlar kaldırılamaz. **MSMQ** desteği veren bağlayıcılardan olan **msmqIntegrationBinding** ve **netMsmqBinding** tipleri ise kendi güvenilir oturum şartnamelerini uygularken WS-ReliableSession standardını kullanmazlar. Aslında konu ile ilişkili olarak aşağıdaki tablonun göz önüne alınması önemlidir.

Bağlatıcı Tip (Binding Type)	**Güvenilir Oturum Desteği**	**Varsayılan Güvenilir Oturum Hali**	**Sıralı Mesaj Desteği**	**Varsayılan Sıralı Mesaj Desteği**	**Ek Bilgi**
netNamedPipeBinding	Yok	**X**	Var	**X**	
netTcpBinding	Var	Açık	Var	Kapalı	
netPeerTcpBinding	**X**				
wsDualHttpBinding	Var	Açık	Var	Açık	
wsHttpBinding	Var	Kapalı	Var	Açık	
wsFederationHttpBinding	Var	Kapalı	Var	Açık	
basicHttpBinding	**X**				Bu bağlayıcı standar oturumlar bulunman
netMsmqBinding					MSMQ tabanlı bir k
msmqIntegrationBinding					

Bu teorik bilgilerden sonra artık bir örnek ile devam etmekte fayda bulunmaktadır. örnekte basit olarak **netTcpBinding** kullanan bir WCF sistemi yer almaktadır. Sistemde yer alan servis sözleşmesi(Service Contract) ve uygulayıcı tipe ait içerikler aşağıda görüldüğü gibi olmakla birlikte, FabrikaLib isimli bir **WCF Sınıf Kütüphanesi(WCF Class Library)** içerisinde yer almaktadırlar.



Servis sözleşmesi;

```
using System;
using System.ServiceModel;

namespace FabrikaLib
{
   [ServiceContract(Name="UretimServisi",
Namespace="http://www.bsenyurt.com/FabrikaLib/UretimServisi"
,SessionMode=SessionMode.Required)]
   public interface IUretici
   {
      [OperationContract(IsInitiating=true)]
      int BilesenAl(string[] bilesenAdi);
      [OperationContract(IsInitiating=false)]
      void Karistir();
      [OperationContract(IsInitiating=false,IsTerminating=true)]
      bool UretimiYap();
   }
}
```

Servis sözleşmesinde**(Service Contract)** dikkat edileceği üzere her istemci(Client) için bir oturum(Session) açılmasını garantilemek
adına **SessionMode** özelliğine **SessionMode.Required**değeri atanmıştır. Güvenilir oturumlarda ilk şartlardan birisi, istemci ile servis arasında bir oturumun söz konusu olmasıdır. Bu nedenle bir oturumun mutlaka hazırlanması isteğinin Servis sözleşmesinde

belirtilmesi yerinde bir karardır. Bununla birlikte metodların işleyiş sıralarıda **OperationContract** niteliklerine(attribute) ait özellikler ile belirlenmiştir. Buna göre istemci tarafından ilk çağrılabilecek metod BilesenAl fonksiyonu iken oturumu sonlandırma işlemini üstlenecek olan işlevse UretimiYap isimli fonksiyondur.

Uygulayıcı sınıf;

```
using System;
using System.ServiceModel;

namespace FabrikaLib
{
   [ServiceBehavior(InstanceContextMode= InstanceContextMode.PerSession)]
   public class Uretici:IUretici
   {
      #region IUretici Members

      public int BilesenAl(string[] bilesenler)
      {
         // Bileşenin eklenme işlemi
         return bilesenler.Length;
      }

      public void Karistir()
      {
         // Bileşenin karıştırılma işlemi
      }

      public bool UretimiYap()
      {
         return true;
      }

      #endregion
   }
}
```

Her bir istemci için bir oturumun açılmasını sağlamak adına Uretici sınıfına **ServiceBehavior** niteliği ile **PerSession** ataması yapılmıştır. Servis tarafından sunulmakta olan fonksiyonelliklerin ne iş yaptığı şu aşamada çok önemli değildir. örnekteki asıl amaç, istemci ve servis arasında **güvenilir bir oturum(Reliable Session)** açılması ve arada hareket eden mesajların **izlenerek(Trace)** durumunun detaylı analizinin yapılmasıdır. Servis tarafı basit olması açısından bir **Console** uygulaması olarak tasarlanmıştır. Servis tarafına ait**Main** kodları ile konfigurasyon dosyasının içeriği başlangıçta aşağıdaki gibidir.

Servis uygulaması kodları;

```
using System;
using System.ServiceModel;
using FabrikaLib;

namespace Sunucu
{
   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(Uretici));
         host.Opened += new EventHandler(host_Opened);
         host.Closed += new EventHandler(host_Closed);
         host.Open();
         Console.ReadLine();
         host.Close();
      }

      static void host_Closed(object sender, EventArgs e)
      {
         Console.WriteLine("Servis kapatıldı");
      }

      static void host_Opened(object sender, EventArgs e)
      {
         Console.WriteLine("Servis dinlemede");
      }
   }
}
```

Servis tarafı konfigurasyon içeriği;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <bindings />
      <behaviors>
         <serviceBehaviors>
            <behavior name="UretimServisiBehavior">
               <serviceDebug includeExceptionDetailInFaults="true" />
            </behavior>
         </serviceBehaviors>
      </behaviors>
```

```
      <services>
         <service behaviorConfiguration="UretimServisiBehavior"
name="FabrikaLib.Uretici">
            <endpoint address="net.tcp://localhost:9000/Fabrika/UretimServisi.svc"
binding="netTcpBinding" name="UretimServisiEndPoint" contract="FabrikaLib.I
Uretici" />
         </service>
      </services>
   </system.serviceModel>
</configuration>
```

İstemci uygulama üzerinde, servis tarafında oluşabilecek **istisnaları(Exception)** detaylı bir şekilde ele alabilmek
için **serviceDebug** elementinde **includeExceptionDetailInFaults** niteliğinin
değeri**true** olarak set edilmiştir. İstemci tarafıda servis tarafı gibi bir **Console** uygulaması olarak tasarlanabilir.

NOT : *İstemci tarafı için gerekli olan* ***proxy*** *sınıfı ve servise göre otomatik oluşturulan konfigurasyon dosyasının üretimi için* ***svcutil.exe*** *aracından aşağıdaki gibi yararlanılması gerekmektedir.*

svcutil FabrikaLib.dll

svcutil www.bsenyurt.com.FabrikaLib.UretimServisi.wsdl *.xsd /out:UretimServisi.cs

Bu işlemin ardından proxy sınıfı ve konfigurasyon dosyası, istemci uygulamaya taşınır.

İstemci tarafındaki kodlar ve konfigurasyon içeriği ise aşağıdaki gibidir.

İstemci uygulaması kodları;

```
using System;
using System.ServiceModel;

namespace Istemci
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            Console.WriteLine("İşlemi başlatmak için bir tuşa basınız...");
            Console.ReadLine();
            UretimServisiClient cli = new
```

```
UretimServisiClient("UretimServisiClientEndPoint");
            cli.BilesenAl(new string[] { "C", "O2", "H2SO4" });
            cli.Karistir();
            string durum = cli.UretimiYap()==true?"üretim gerçekleştirildi":"üretim
yapılamadı";
            Console.WriteLine(durum);
            Console.ReadLine();
         }
         catch (Exception excp)
         {
            Console.WriteLine(excp.Message);
         }
      }
   }
}
```

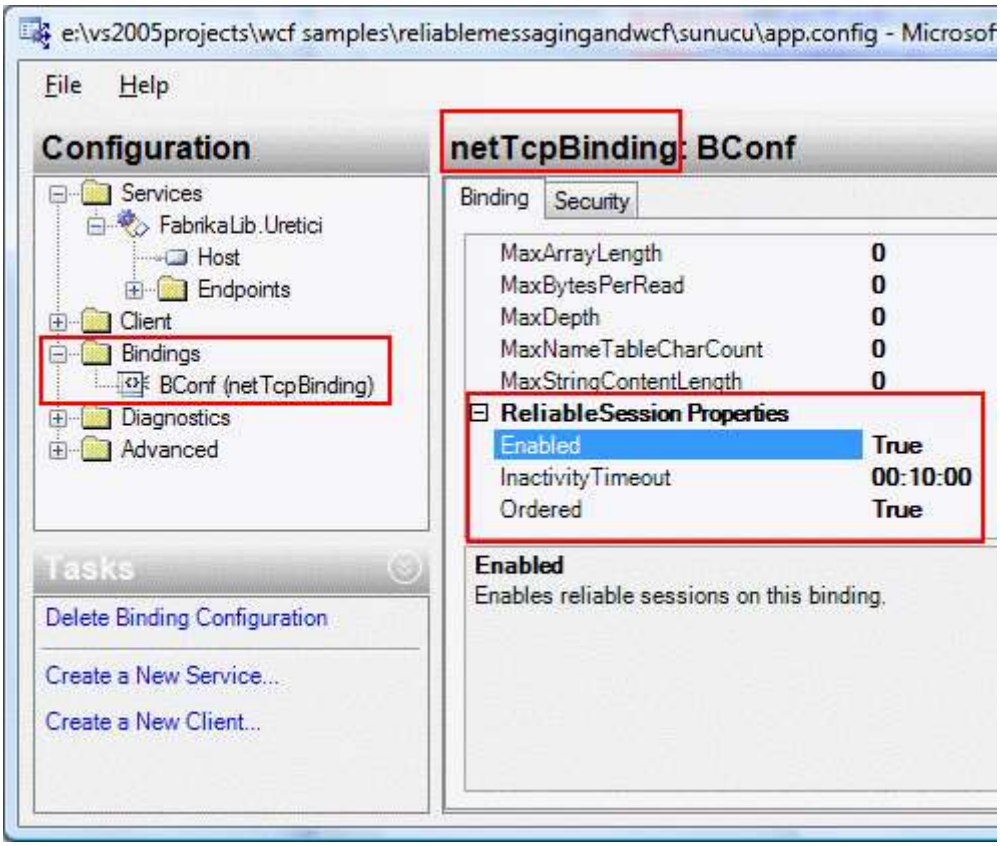
İstemci tarafı konfigurasyon içeriği;

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <system.serviceModel>
      <bindings />
      <client>
         <endpoint address="net.tcp://localhost:9000/Fabrika/UretimServisi.svc"
binding="netTcpBinding" contract="UretimServisi"
name="UretimServisiClientEndPoint" />
      </client>
   </system.serviceModel>
</configuration>
```

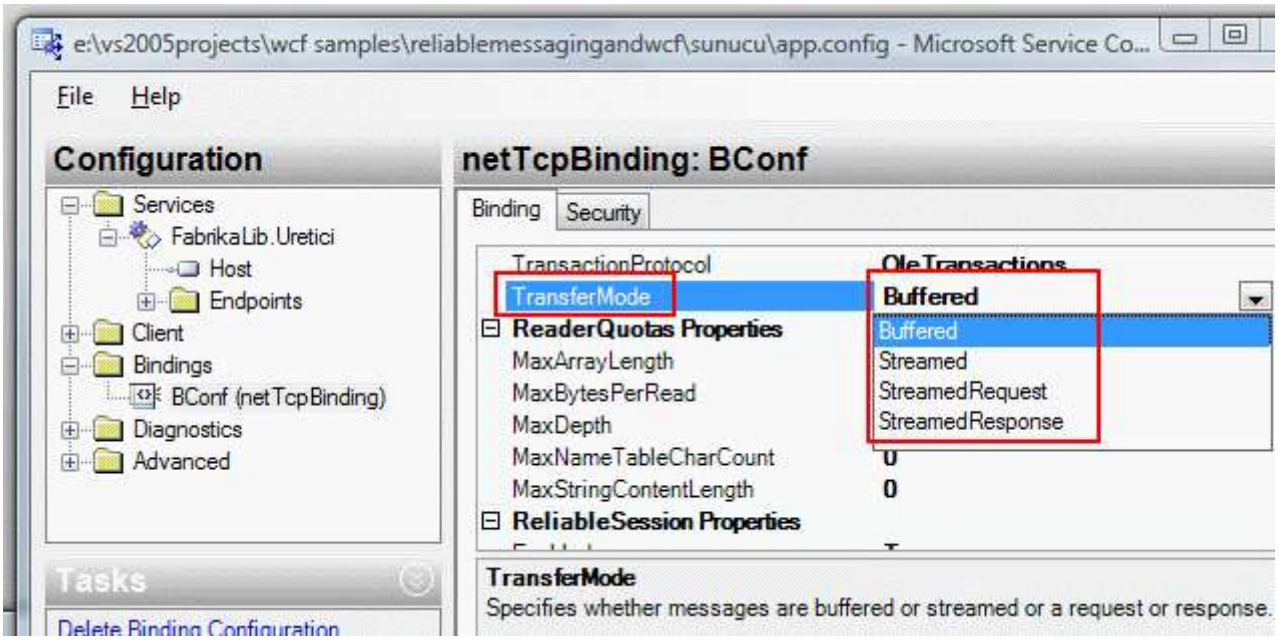
Normalde **svcutil** tarafından üretilen konfigurason dosyası içerisinde çok daha fazla özellik yer almaktadır. Bu niteliklerin değerleri şu aşamada önemli olmadığından istemci konfigurasyon dosyası bilinçli olarak yukarıdaki gibi sadeleştirilmiştir.

Asıl işlemler bundan sonra başlamaktadır. Amaç güvenilir bir oturum ortamı hazırlamaktır. Bunu gerçekleştirmek son derece basittir. Tek yapılması gereken bağlayıcı ile ilgili özel konfigurasyonları ve içinde yer alan özelliklerin değerlerini uygun bir şekilde belirlemektir. Servis tarafında **netTcpBinding** için bir **BindingConfiguration** eklenmelidir. Söz konusu kısımda, aşağıdaki şekildende görüldüğü
gibi **ReliableSession Properties** bölümünden **Enabled** özelliği **true** olarak belirlenmelidir. Böylece güvenilir bir oturumun tesis edileceği belirtilmiş
olunur. **InactivityTimeout** özelliğinin aldığı değer ile, mesajların kaybolma ihtimali için gereken bekleme süresi ayarlanır. Yani, 10 dakikalık süre içerisinde beklenen mesaj alınmassa ters giden bir şeyler olduğuna karar verilir ve **WCF çalışma zamanı(Run Time)** bir **Fault Exception** üreterek bunu istemci tarafına gönderir. Aynı zamanda o ana

kadar yapılmış olan işlemler **geri alınır(Rollback)** ve istemci ile servis arasındaki güncel oturum sonlandırılır. **Ordered** özelliğine atanan değerin **true** olarak set edilmesi ile, servise gelen mesajların istemcinin gönderdiği sırada ele alınmaları garanti edilmiş olunmaktadır. Bu zorunlu olmamasına rağmen güvenilir oturumların sağlanması adına önemlidir.



Binding ayarlarında dikkat edilmesi gereken noktalardan biriside **TransferMode** özelliğinin değeridir. **netTcpBinding bağlayıcı tipi(Binding Type)** için bu değer varsayılan olarak aşağıdaki şekilde görüldüğü gibi **Buffered** olarak belirlenmiştir.



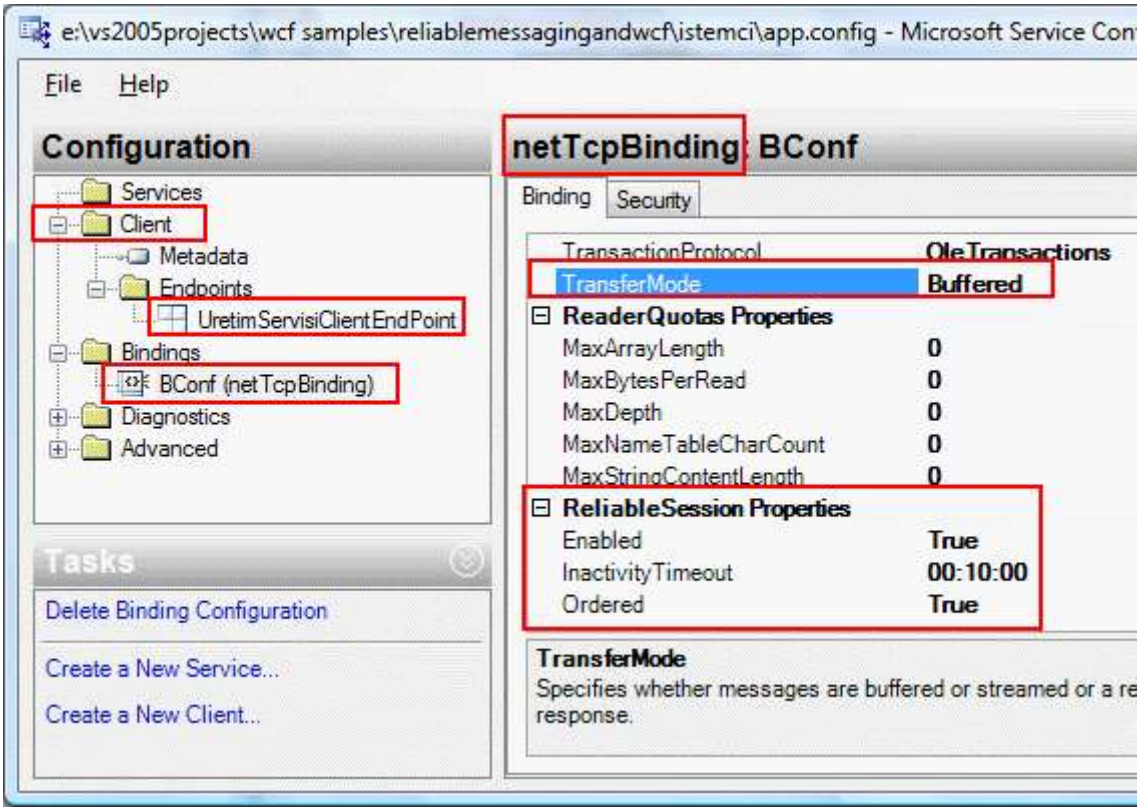
NetTcpBinding kullandığı transfer protokolü(TCP) nedeni ile **Stream**' lere izin vermektedir. Bir başka deyişle istemcinin gönderdiği mesajların servis tarafında tamamlanmasını beklemeden işlenmesine başlanabilmektedir. Ancak güvenilir oturumlarda, mesajların tamamlandıktan sonra, bir başka deyişle servis tarafına ulaştıktan sonra ele alınmaları doğru sırada işlenmeleri söz konusu olduğunda önemlidir. Bu sebepten bu varsayılan değerin değiştirilmemesi önerilir. Diğer taraftan özellikle **WsHttpBinding** gibi bağlayıcı tipler, **Stream** yapısını **HTTP**protokolü nedeni ile desteklemediklerinden her zaman için **Buffered** sistemi ile çalışırlar.

Yukarıda yapılan değişiklikler sonrasında servis tarafında yer alan konfigurasyon dosyasının son hali aşağıdaki gibi olacaktır.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <netTcpBinding>
        <binding name="BConf" transferMode="Buffered">
          <reliableSession ordered="true" inactivityTimeout="00:10:00"
enabled="true" />
        </binding>
      </netTcpBinding>
    </bindings>
    <behaviors>
      <serviceBehaviors>
        <behavior name="UretimServisiBehavior">
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </serviceBehaviors>
```

```
    </behaviors>
    <services>
      <service behaviorConfiguration="UretimServisiBehavior"
name="FabrikaLib.Uretici">
        <endpoint address="net.tcp://localhost:9000/Fabrika/UretimServisi.svc"
binding="netTcpBinding" bindingConfiguration="BConf" name="UretimServisiEndPoi
nt" contract="FabrikaLib.IUretici" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

çok doğal olarak istemci uygulama tarafındada servis tarafındakine benzer olacak şekilde konfigurasyon ayarlarının yapılması gerekmektedir. Aynı adımlar izlenildiğinde istemci tarafındaki konfigurasyon dosyasın son halide aşağıdaki gibi olacaktır.

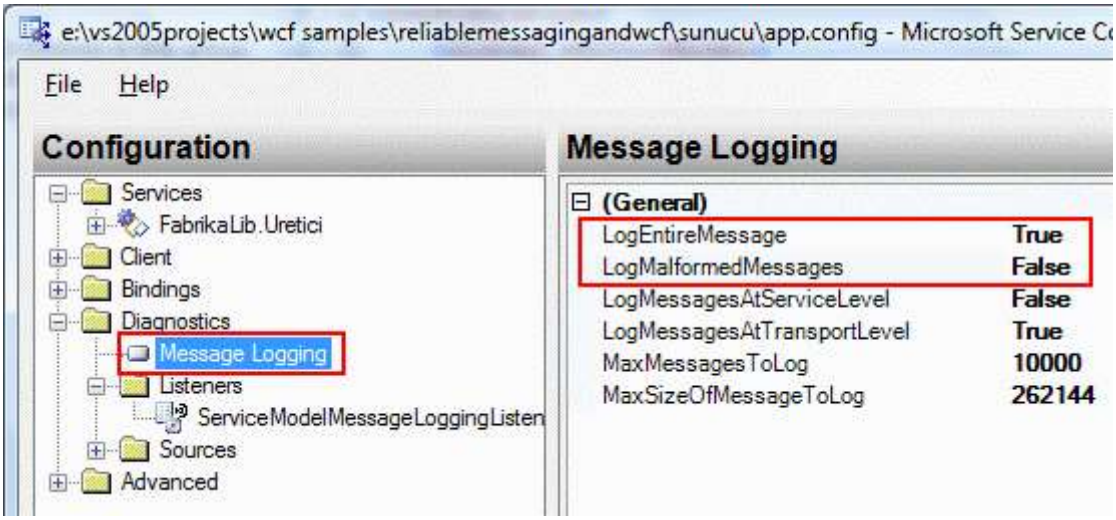



```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <system.serviceModel>
    <bindings>
      <netTcpBinding>
        <binding name="BConf" transferMode="Buffered">
          <reliableSession ordered="true" inactivityTimeout="00:10:00"
enabled="true" />
```

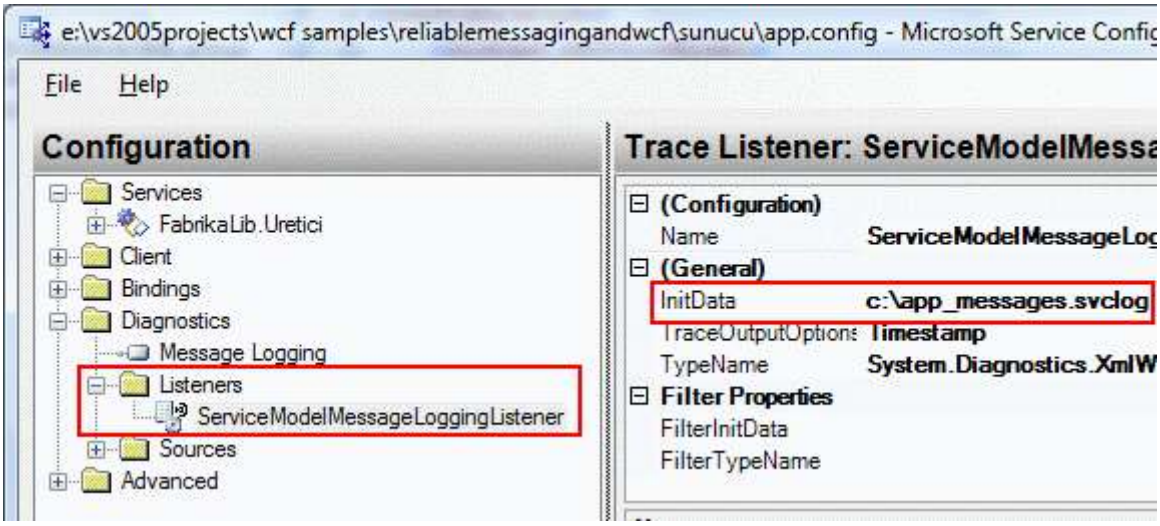
```
            </binding>
          </netTcpBinding>
        </bindings>
        <client>
          <endpoint address="net.tcp://localhost:9000/Fabrika/UretimServisi.svc"
binding="netTcpBinding" bindingConfiguration="BConf" contract="UretimServisi"
name="UretimServisiClientEndPoint" />
        </client>
    </system.serviceModel>
</configuration>
```

Olaya istemci açısından bakıldığında çalışma aşamasında dikkate değer bazı noktalar vardır. Herşeyden önce istemci uygulama, servis tarafında belirtilen **timeout** süresi dahilinde mesaj göndermeyi bırakmış olabilir. Bu çoğunlukla istemci uygulama kullanıcısının bu yönde bir aksiyon gerçekleştirmediği durumlarda söz konusu olabilir. Tabi burada istemci ile servis arasında bir oturum açıldıktan sonraki süre zarfı ele alınmaktadır. Bu tarz bir durumda istemcinin ağ üzerinde asılı kaldığı yorumu yapılır. Dolayısıyla servisin istenmeyen bir şekilde **Fault Exception** döndürmesi olasıdır. Bu sebepten, istemci taraftaki **WCF çalışma Zamanı(Run Time)** belirli periyodlarda servis tarafına canlı olduğuna dair mesajlar gönderir. Böylece servis tarafı, kendisine bağlı oturumun sahibi olan istemcinin halen daha canlı olduğundan haberdar olur. Bununla birlikte istemci uygulama, servis tarafından bir **onay mesajı(Acknowledge Message)** bekler. Eğer bu mesaj istemci tarafında belirtilen **InactivityTimeout** süresinde alınamıyorsa, servisin bir şekilde öldüğü sonucuna varılır ve istemci tarafında**WCF çalışma zamanı(Run Time)** bir istisna(Exception) fırlatır. Bu istisna, istemci tarafında ele alınmalı ve uygulamanın istem dışı şekilde sonlanmasının önüne geçilmelidir.

Artık istemci ve servis arasında güvenilir bir oturum açılması için gereken ayarlar tamamlanmıştır. Bu oturumun tesis edilmesi halinde, arada gidip gelen mesajların incelenebilmesi adına servis tarafında gerekli ayarların yapılması gerekmektedir. Bu amaçla yine konfigurasyon içerisinde **Diagnostics** ayarları yapılmalıdır. İlk olarak **EnableMessageLogging** linkine tıklanarak mesaj günlüğü aktif hale getirilir. Sonrasında ise **Diagnostics** klasöründe yer alan **Message Logging** kısmına gidilerek **LogEntireMessage** değeri **true**, **LogMalformedMessages** değeride **false**olarak set edilir.



Bu işlemin
ardından **Listeners** klasöründeki **ServiceModelMessageLoggingListener** kısmına gidilerek **InitData** özelliğine bir **svclog** dosyası adı ve tam adresi aşağıdaki şekilde olduğu gibi bildirilir. Tahmin edileceği üzere servis tarafına ulaşan ve istemciye giden mesaj içerikleri bu dosya içerisinde toplanacaktır.



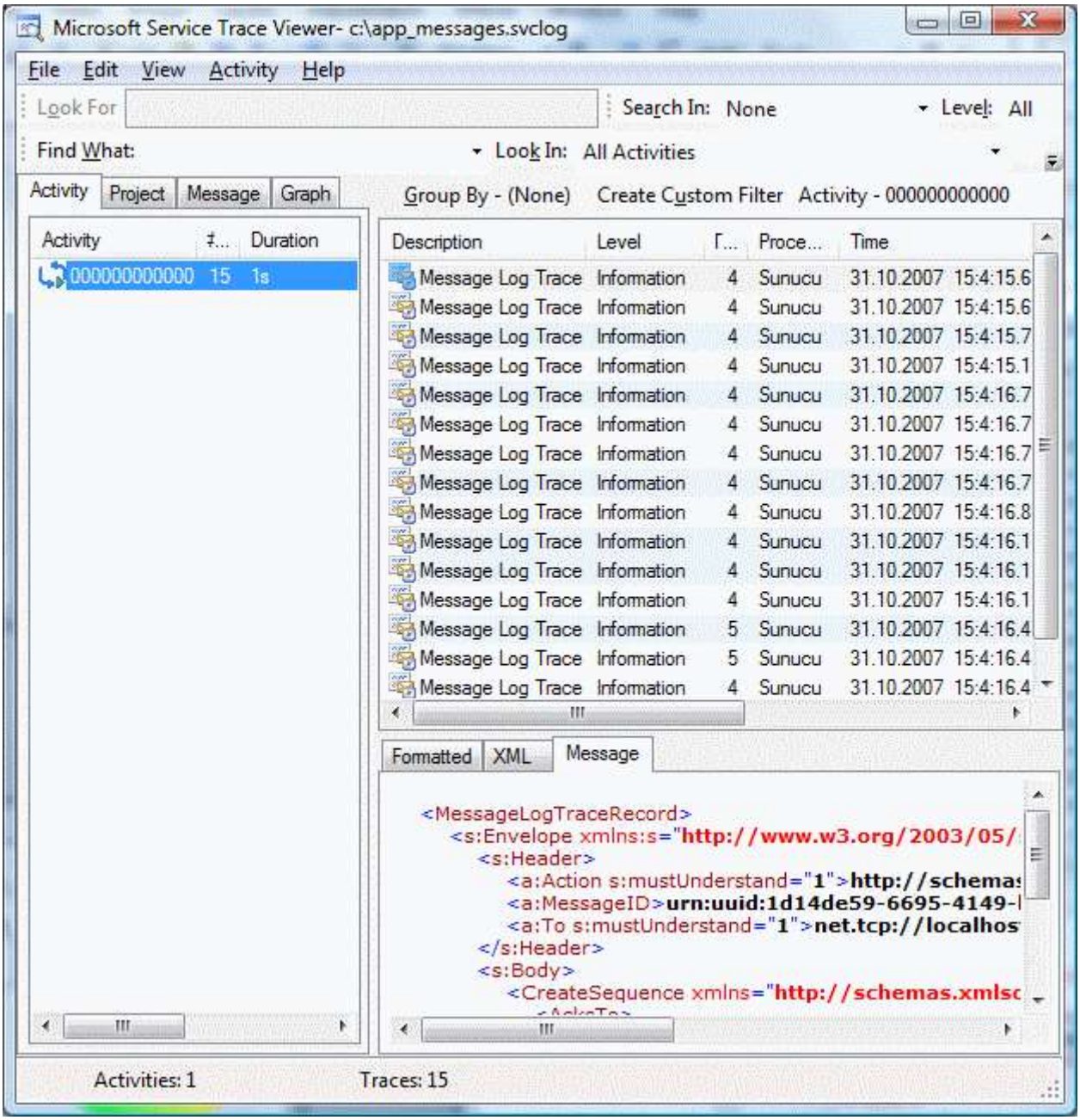
Bu işlemlerin ardından servis tarafındaki konfigurasyon dosyasının içeriği aşağıdaki gibi olacaktır.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <source name="System.ServiceModel.MessageLogging"
switchValue="Warning, ActivityTracing">
        <listeners>
          <add type="System.Diagnostics.DefaultTraceListener" name="Default">
            <filter type="" />
```

```
            </add>
            <add name="ServiceModelMessageLoggingListener">
              <filter type="" />
            </add>
          </listeners>
        </source>
      </sources>
      <sharedListeners>
        <add
initializeData="c:\app_messages.svclog" type="System.Diagnostics.XmlWriterTraceList
ener, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089"
name="ServiceModelMessageLoggingListener" traceOutputOptions="Timestamp">
          <filter type="" />
        </add>
      </sharedListeners>
    </system.diagnostics>
    <system.serviceModel>
      <diagnostics>
        <messageLogging logEntireMessage="true" logMalformedMessages="false"
logMessagesAtTransportLevel="true" />
      </diagnostics>
      <bindings>
        <netTcpBinding>
          <binding name="BConf" transferMode="Buffered">
            <reliableSession ordered="true" inactivityTimeout="00:10:00" enabled="true"
/>
          </binding>
        </netTcpBinding>
      </bindings>
      <behaviors>
        <serviceBehaviors>
          <behavior name="UretimServisiBehavior">
            <serviceDebug includeExceptionDetailInFaults="true" />
          </behavior>
        </serviceBehaviors>
      </behaviors>
      <services>
        <service behaviorConfiguration="UretimServisiBehavior"
name="FabrikaLib.Uretici">
          <endpoint address="net.tcp://localhost:9000/Fabrika/UretimServisi.svc"
binding="netTcpBinding" bindingConfiguration="BConf" name="UretimServisiEndPoint"
contract="FabrikaLib.IUretici" />
        </service>
      </services>
```

```
    </system.serviceModel>
</configuration>
```

Mesajların izlenmesi için **Windows SDK** ile birlikte gelen **Service Trace Viewer** programına ihtiyaç vardır. İstemci ve servis uygulaması çalıştırılarak test edildikten sonra Service Trace Viewer yardımıyla mesajlaşma trafiği izlenebilir. İlk testin ardından C klasörü altında **app_messages.svclog** isimli bir dosya otomatik olarak oluşturulup içeriği doldurulacaktır. Dosya, Service Trace Viewer programı ile açıldığında aşağıdaki ekran görüntüsüne benzer bir şekilde 15 adet mesajın üretildiği görülecektir.



Şimdi bu mesajlar kısaca analiz edilebilir. İlk mesajın içeriği aşağıdaki gibidir.

```
<MessageLogTraceRecord>
   <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing">
      <s:Header>
         <a:Action
s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/rm/CreateSequence</a:Act
ion>
         <a:MessageID>urn:uuid:1d14de59-6695-4149-b218-
cb41783db18d</a:MessageID>
         <a:To
s:mustUnderstand="1">net.tcp://localhost:9000/Fabrika/UretimServisi.svc</a:To>
      </s:Header>
      <s:Body>
         <CreateSequence xmlns="http://schemas.xmlsoap.org/ws/2005/02/rm">
            <AcksTo>
               <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
            </AcksTo>
            <Offer>
               <Identifier>urn:uuid:be73ae30-145a-4bd6-bf04-
ee6a3b1edfce</Identifier>
            </Offer>
         </CreateSequence>
      </s:Body>
   </s:Envelope>
</MessageLogTraceRecord>
```

Bu mesaj ile istemci ve servis arasında güvenilir bir oturum başlatılmaktadır. Tahmin edileceği üzere mesaj istemci tarafından servise gönderilmiştir. Aynı güvenilir oturumda yer alan tüm mesajlar aynı benzersiz numara kümesini **(Unique Identifier Set)** kullanırlar. Bu anlamda ilk mesajın içerisinde yer alan **MessageID** değeri ikinci mesaj içerisinde de ele alınmaktadır. İkinci mesaj, servis tarafından istemciye gönderilen mesajdır ve içeriği aşağıdaki gibidir.

```
<MessageLogTraceRecord>
   <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:a="http://www.w3.org/2005/08/addressing">
      <s:Header>
         <a:Actions:mustUnderstand="1"> http://schemas.xmlsoap.org/ws
/2005/02/rm/CreateSequenceResponse</a:Action>
         <a:RelatesTo>urn:uuid:1d14de59-6695-4149-b218-
cb41783db18d</a:RelatesTo>
         <a:To
s:mustUnderstand="1">http://www.w3.org/2005/08/addressing/anonymous</a:To>
      </s:Header>
      <s:Body>
```

```
        <CreateSequenceResponse xmlns="http://schemas.xmlsoap.org/ws/2005/02/rm">
          <Identifier>urn:uuid:27079656-f1cf-460d-9289-28f43f664fbe</Identifier>
          <Accept>
            <AcksTo>
              <a:Address>net.tcp://localhost:9000/Fabrika/UretimServisi.svc</a:Address>
            </AcksTo>
          </Accept>
        </CreateSequenceResponse>
      </s:Body>
   </s:Envelope>
</MessageLogTraceRecord>
```

İkinci mesajda üretilen **Identifier** elementinin değerinin, istemci tarafından sonradan gönderilecek mesajlarda mutlaka sağlanması gerekmektedir. Böylece aradaki mesajların aynı güvenilir oturum(Reliable Session) içerisinde olacağı anlaşılabilir. Buna göre istemci tarafından ilk metod çağrısı için gönderilen üçüncü mesaj içeriği aşağıdaki gibi olacaktır.

```
<MessageLogTraceRecord>
   <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing">
      <s:Header>
         <r:AckRequested>
            <r:Identifier>urn:uuid:27079656-f1cf-460d-9289-
28f43f664fbe</r:Identifier>
         </r:AckRequested>
         <r:Sequence s:mustUnderstand="1">
            <r:Identifier>urn:uuid:27079656-f1cf-460d-9289-
28f43f664fbe</r:Identifier>
            <r:MessageNumber>1</r:MessageNumber>
         </r:Sequence>
         <a:Action s:mustUnderstand="1">http://www.bsenyurt.com/FabrikaLib/
UretimServisi/UretimServisi/BilesenAl</a:Action>
         <a:MessageID>urn:uuid:a6127691-9534-4f46-a3ff-
97943cd19b30</a:MessageID>
         <a:ReplyTo>
            <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
         </a:ReplyTo>
         <a:To
s:mustUnderstand="1">net.tcp://localhost:9000/Fabrika/UretimServisi.svc</a:To>
      </s:Header>
      <s:Body>
         <BilesenAl xmlns="http://www.bsenyurt.com/FabrikaLib/UretimServisi">
            <bilesenAdi xmlns:b="http://schemas.microsoft.com/2003/10/Serialization/Arra
ys" xmlns:i="http://www.w3.org/2001/XMLSchema-instance">
```

```
            <b:string>C</b:string>
            <b:string>O2</b:string>
            <b:string>H2SO4</b:string>
          </bilesenAdi>
        </BilesenAl>
      </s:Body>
   </s:Envelope>
</MessageLogTraceRecord>
```

üçüncü mesaj **BilesenAl** isimli metod için bir çağrı olduğundan **SOAP** paketinin **gövdesi (Body)** içerisinde parametre değerleride gönderilmektedir. Diğer taraftan **MessageNumber**elementi ile gönderilen mesajın sıra numarasıda belirlenmiş olmaktadır. üçüncü mesaj ile gelen metod çağrısına karşılık olaraktan servis tarafı bir cevap mesajını dördüncü mesaj olarak istemciye gönderecektir. Dördüncü mesajın içeriği aşağıdaki gibidir.

```
<MessageLogTraceRecord>
   <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing">
      <s:Header>
         <r:SequenceAcknowledgement>
            <r:Identifier>urn:uuid:27079656-f1cf-460d-9289-
28f43f664fbe</r:Identifier>
            <r:AcknowledgementRange Lower="1"
Upper="1"></r:AcknowledgementRange>
            <netrm:BufferRemaining
xmlns:netrm="http://schemas.microsoft.com/ws/2006/05/rm">8</netrm:BufferRemaining>
         </r:SequenceAcknowledgement>
         <r:AckRequested>
            <r:Identifier>urn:uuid:be73ae30-145a-4bd6-bf04-
ee6a3b1edfce</r:Identifier>
         </r:AckRequested>
         <r:Sequence s:mustUnderstand="1">
            <r:Identifier>urn:uuid:be73ae30-145a-4bd6-bf04-
ee6a3b1edfce</r:Identifier>
            <r:MessageNumber>1</r:MessageNumber>
         </r:Sequence>
         <a:Action s:mustUnderstand="1">http://www.bsenyurt.com/FabrikaLib/
UretimServisi/UretimServisi/BilesenAlResponse</a:Action>
         <a:RelatesTo>urn:uuid:a6127691-9534-4f46-a3ff-
97943cd19b30</a:RelatesTo>
         <a:To
s:mustUnderstand="1">http://www.w3.org/2005/08/addressing/anonymous</a:To>
      </s:Header>
```

```
      <s:Body>
         <BilesenAlResponse
xmlns="http://www.bsenyurt.com/FabrikaLib/UretimServisi">
         <BilesenAlResult>3</BilesenAlResult>
         </BilesenAlResponse>
      </s:Body>
   </s:Envelope>
</MessageLogTraceRecord>
```

Bu mesaj içerisindede istemciye biraz önce gönderdiği mesajın başarılı bir şekilde alındığı ve onaylandığı bilgisi iletilmektedir. Sıradaki beşinci mesaj ile istemci servise bir onaylama bildirisi göndermektedir ve içeriği aşağıdaki gibidir.

```
<MessageLogTraceRecord>
   <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing">
      <s:Header>
         <r:SequenceAcknowledgement>
            <r:Identifier>urn:uuid:be73ae30-145a-4bd6-bf04-
ee6a3b1edfce</r:Identifier>
            <r:AcknowledgementRange Lower="1"
Upper="1"></r:AcknowledgementRange>
            <netrm:BufferRemaining
xmlns:netrm="http://schemas.microsoft.com/ws/2006/05/rm">8</netrm:BufferRemaining>
         </r:SequenceAcknowledgement>
         <a:Action s:mustUnderstand="1">http://schemas.xmlsoap.org/
ws/2005/02/rm/SequenceAcknowledgement</a:Action>
         <a:To
s:mustUnderstand="1">net.tcp://localhost:9000/Fabrika/UretimServisi.svc</a:To>
      </s:Header>
   <s:Body></s:Body>
</s:Envelope>
</MessageLogTraceRecord>
```

Bu mesaj dördüncü mesajın ve içeriğinin istemci tarafından alındığını servis tarafına bildirmektedir. **SequenceAcknowledgment** elementinde yer alan **identifier** değerine bakılırsa birinci mesajda üretilen **identifier** ile aynı olduğu görülebilir. Hemen arkasından gelen altıncı mesaja bakıldığında istemcinin ikinci metod çağrısını gerçekleştirdiği görülür.

```
<MessageLogTraceRecord>
   <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing">
      <s:Header>
```

```
        <r:Sequence s:mustUnderstand="1">
          <r:Identifier>urn:uuid:27079656-f1cf-460d-9289-
28f43f664fbe</r:Identifier>
          <r:MessageNumber>2</r:MessageNumber>
        </r:Sequence>
        <a:Action s:mustUnderstand="1">
http://www.bsenyurt.com/FabrikaLib/UretimServisi/UretimServisi/Karistir</a:Action>
        <a:MessageID>urn:uuid:180ea9af-ddbc-4249-8880-
40c412088ce3</a:MessageID>
        <a:ReplyTo>
          <a:Address>http://www.w3.org/2005/08/addressing/anonymous</a:Address>
        </a:ReplyTo>
        <a:To
s:mustUnderstand="1">net.tcp://localhost:9000/Fabrika/UretimServisi.svc</a:To>
      </s:Header>
      <s:Body>
        <Karistir xmlns="http://www.bsenyurt.com/FabrikaLib/UretimServisi"></Karistir>
      </s:Body>
    </s:Envelope>
</MessageLogTraceRecord>
```

İçeriktende görüldüğü gibi Karistir metoduna bir çağrı gelmektedir. Ancak burada önemli olan noktalardan biriside **MessageNumber** değeridir. Dikkat edilecek olursa 2 değeri gelmektedir. Bir başka deyişle bu mesajın ikinci sırada ele alınması gerektiği belirtilmiş olmaktadır. İstemci tarafından metod çağrıları sonucu oluşturulan mesajların aynı güvenilir oturumda yer alması ama farklı mesajlar olarak ele alınması **MessageID** değerleri sayesinde gerçekleştirilir. Bu nedenle burada üretilen MessageID değeri bir önceki metod çağrısında üretilenden farklı olarak belirlenmektedir. Altıncı mesaj için servisin ürettiği yedinci mesajın içeriği aşağıdaki gibidir.

```
<MessageLogTraceRecord>
  <s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope"
xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm"
xmlns:a="http://www.w3.org/2005/08/addressing">
    <s:Header>
      <r:SequenceAcknowledgement>
        <r:Identifier>urn:uuid:27079656-f1cf-460d-9289-
28f43f664fbe</r:Identifier>
        <r:AcknowledgementRange Lower="1"
Upper="2"></r:AcknowledgementRange>
        <netrm:BufferRemaining
xmlns:netrm="http://schemas.microsoft.com/ws/2006/05/rm">8</netrm:BufferRemaining>
      </r:SequenceAcknowledgement>
      <r:Sequence s:mustUnderstand="1">
        <r:Identifier>urn:uuid:be73ae30-145a-4bd6-bf04-
```

ee6a3b1edfce</r:Identifier>
<r:MessageNumber>2</r:MessageNumber>
</r:Sequence>
<a:Action s:mustUnderstand="1">http://www.bsenyurt.com/FabrikaLib/UretimServisi/UretimServisi/KaristirResponse</a:Action>
<a:RelatesTo>urn:uuid:180ea9af-ddbc-4249-8880-40c412088ce3</a:RelatesTo>
<a:To s:mustUnderstand="1">http://www.w3.org/2005/08/addressing/anonymous</a:To>
</s:Header>
<s:Body>
<KaristirResponse xmlns="http://www.bsenyurt.com/FabrikaLib/UretimServisi"></KaristirResponse>
</s:Body>
</s:Envelope>
</MessageLogTraceRecord>

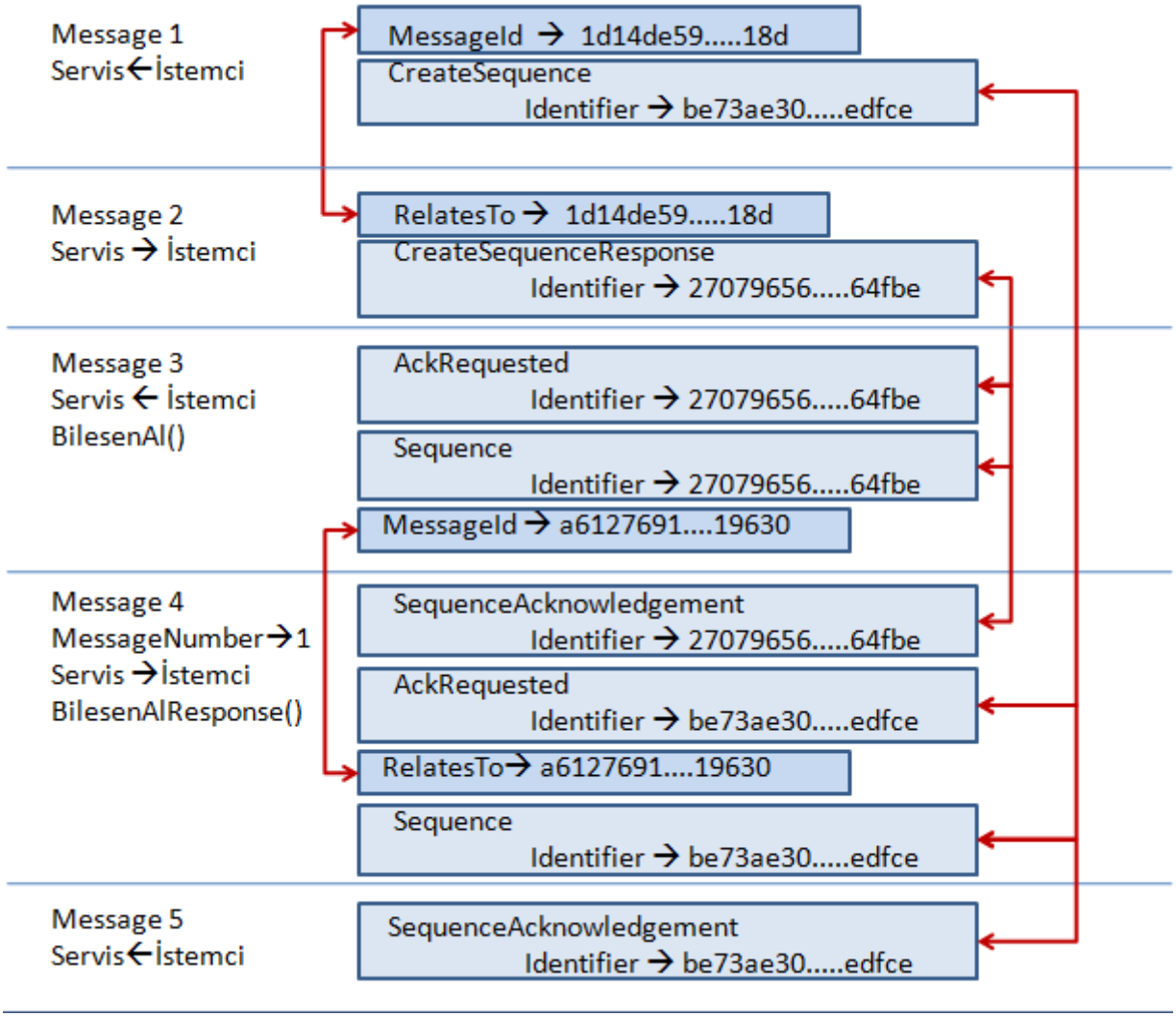
Bu mesajlaşma trafiği diğer metod çağrısı içinde benzer şekilde işleyecektir. İstemcinin yaptığı metod çağrıları sona erdikten sonra ise, servis tarafına aşağıdaki mesaj gönderilir. *(örnek uygulamadaki çalışma sistemine göre **Service Trace Viewer** sonlandırma talebi onuncu mesaj olarak elde edilmektedir.)*

<MessageLogTraceRecord>
<s:Envelope xmlns:s="http://www.w3.org/2003/05/soap-envelope" xmlns:r="http://schemas.xmlsoap.org/ws/2005/02/rm" xmlns:a="http://www.w3.org/2005/08/addressing">
<s:Header>
<r:SequenceAcknowledgement>
<r:Identifier>urn:uuid:be73ae30-145a-4bd6-bf04-ee6a3b1edfce</r:Identifier>
<r:AcknowledgementRange Lower="1" Upper="3"></r:AcknowledgementRange>
<netrm:BufferRemaining xmlns:netrm="http://schemas.microsoft.com/ws/2006/05/rm">8</netrm:BufferRemaining>
</r:SequenceAcknowledgement>
<r:Sequence s:mustUnderstand="1">
<r:Identifier>urn:uuid:27079656-f1cf-460d-9289-28f43f664fbe</r:Identifier>
<r:MessageNumber>4</r:MessageNumber>
<r:LastMessage></r:LastMessage>
</r:Sequence>
<a:Action s:mustUnderstand="1">http://schemas.xmlsoap.org/ws/2005/02/rm/LastMessage</a:Action>
<a:To s:mustUnderstand="1">net.tcp://localhost:9000/Fabrika/UretimServisi.svc</a:To>

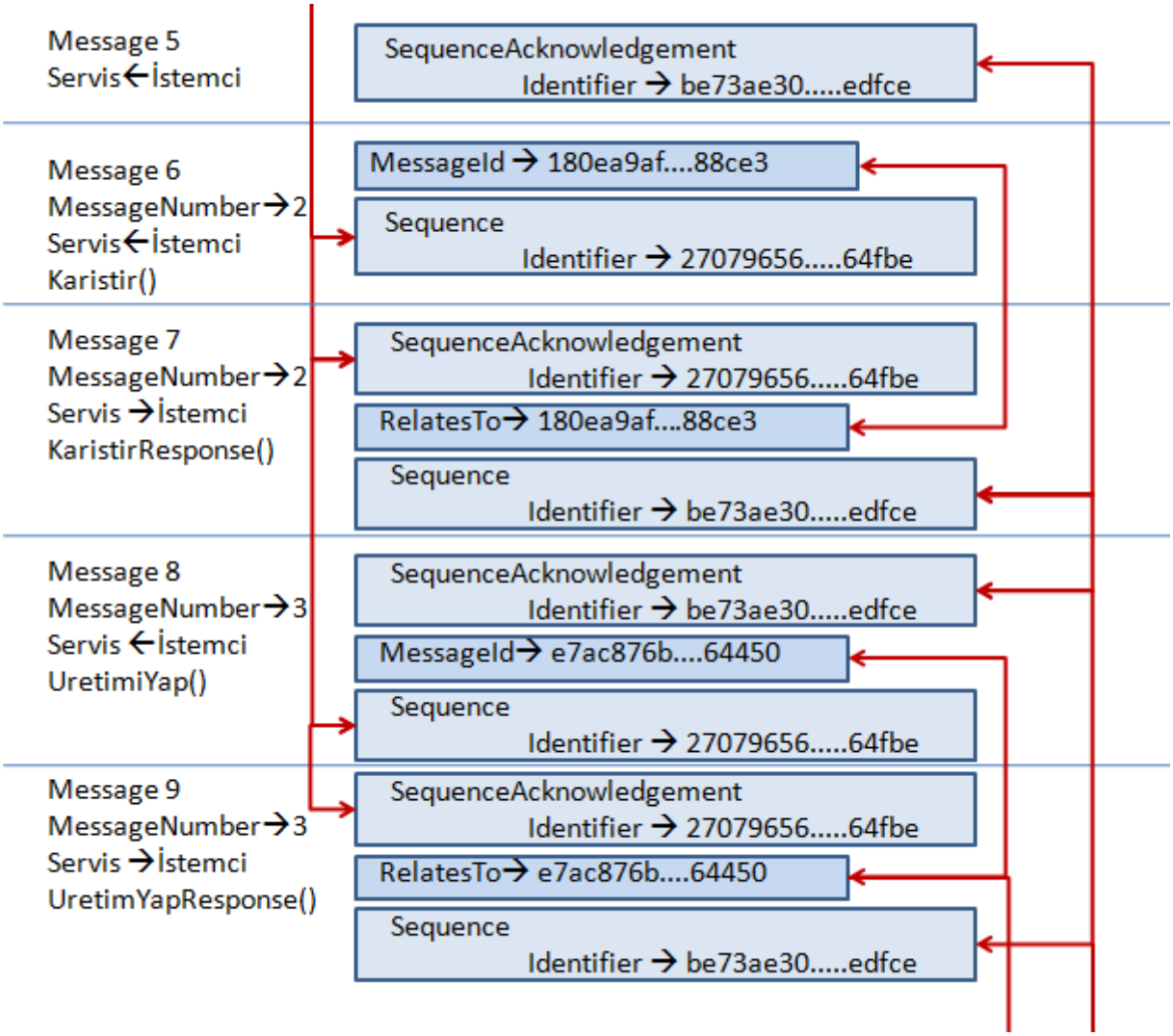
```
      </s:Header>
      <s:Body></s:Body>
   </s:Envelope>
</MessageLogTraceRecord>
```

Son mesaj olduğunun bildirimi için r adındaki **xml isim alanındaki(Xml Namespace) LastMessage** elementi kullanılır. Bununla birlikte **Action** elementi içerisinde yapılan çağrıda**LastMessage** bildirimi yapılır. Bu mesajın oluşması için istemcinin oturumu kapatıyor olması gerekmektedir ki geliştirilen örnekte zaten **IsTerminating** olarak işaretlenmiş metod çağrısından sonra açık olan oturum(Session) kapatılma sürecine girecektir. Herşey bittikten sonra üretilen mesajlarda yer alan **TerminateSequence** elementleri içerisinde yer alan **Identifier** değerleri ile istemci ve servis birbilerine artık kaynaklarını kullanmayacaklarını ve güvenilir oturumu(Reliable Session) kapatacakları bilgilerini vermektedirler. Yukarıdaki işleyiş aşağıdaki şekil ilede değerlendirilebilir. *(Mesajlar üzerinden bu tarz bir görseli hazırlamak oldukça zorlayıcı olmuştur. Bu nedenle gözden kaçan noktalar söz konusu olabilir. Asıl dayanak noktası Service Trace Viewer programının ürettiği Message içerikleri olmalıdır.)*

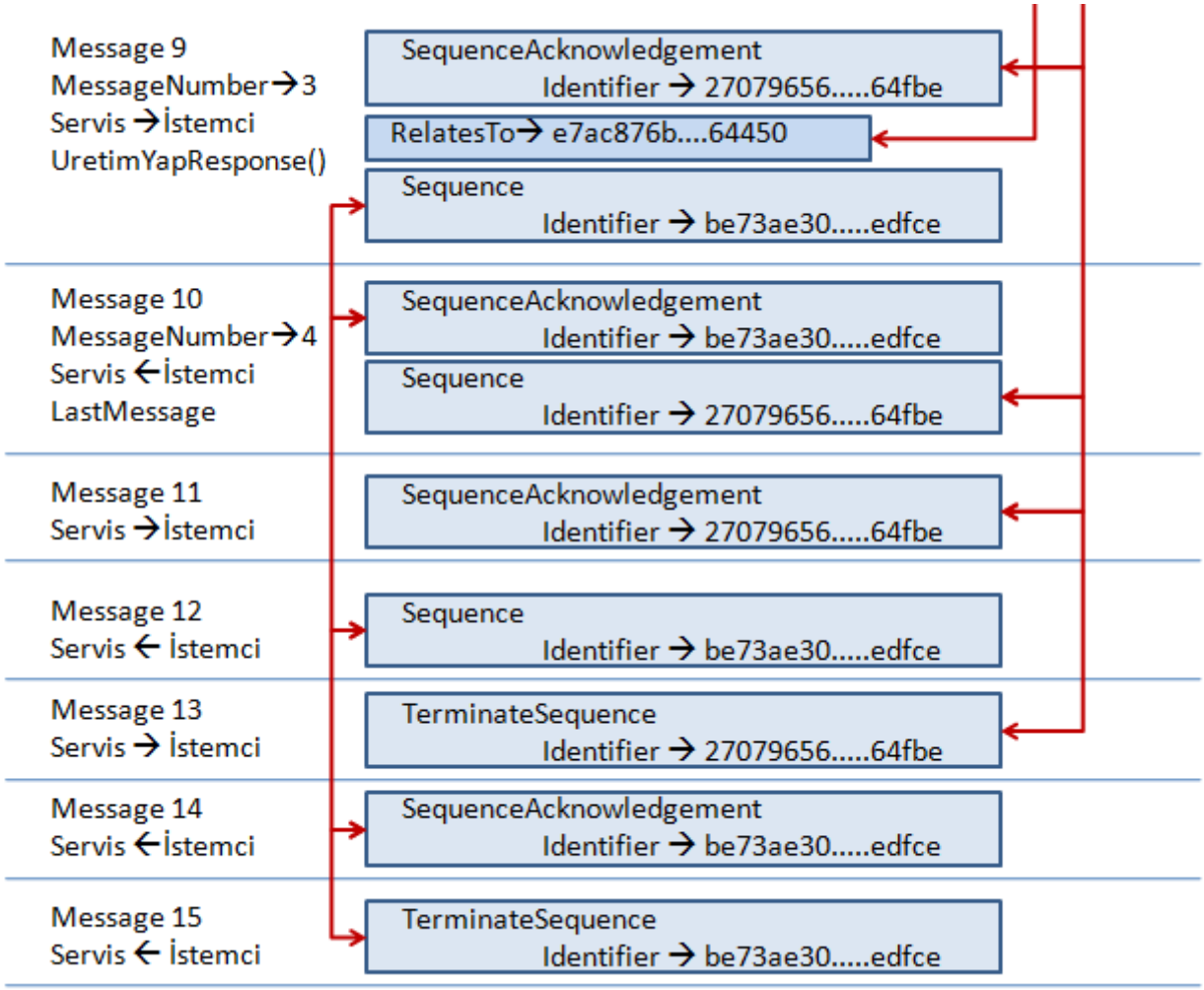
Mesaj 1 ile Mesaj 5 arası durum;



Mesaj 5 ile Mesaj 9 arası durum;



Mesaj 9 ile Mesaj 15 arası durum;



Elbette geliştirici olarak arka tarafta hareket eden mesajların içerikleri çok önemli olmayabilir. Ancak güvenilir oturumlarda söz konusu olan bir dezavantaj vardır. Buda örnektende görüleceği üzere ağ üzerinde istemci ile servis arasında meydana gelen ekstra mesaj yüküdür. Dolayısıyla bu ekstra mesaj yükü, özellikle çok fazla istemcinin olduğu sistemlerde performans kaybı yaşatmaktadır. Dolayısıyla **güvenilir oturumları(Reliable Session)** kullanmadan önce gerekliliklerin ortaya konması doğru bir çözümsel yaklaşım olacaktır. Güvenilir oturumlarda mesajlar için benzersiz ve tekrar etmeyen tanımlayıcı değerler kullandığından **cevaplama saldırılarının (Reply Attack)** azaltılmasıda söz konusudur. Yinede güvenilir oturumlar açılması cevaplama saldırıları için yeterli bir savunma mekanizması sunmaz. Kesin çözüm için özel bir bağlayıcı kullanmak gerekir. özel bir bağlayıcı yardımıyla cevaplama saldırılarına(Reply Attack) karşı nasıl ayakta kalınabileceğini bir sonraki makalemizde incelemeye çalışıyor olacağız.

Böylece geldik uzun bir makalemizin daha sonuna. Bu makalemizde istemci ve servis arasında güvenilir bir oturumun nasıl sağlanabileceğini ele alırken arada hareket eden mesajlarıda analiz etmeye gayret ettik. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

ReliableMessagingAndWCF.rar (67,82 kb)

WCF - Windows CardSpace ile Güvenlik (2007-10-25T05:07:00)

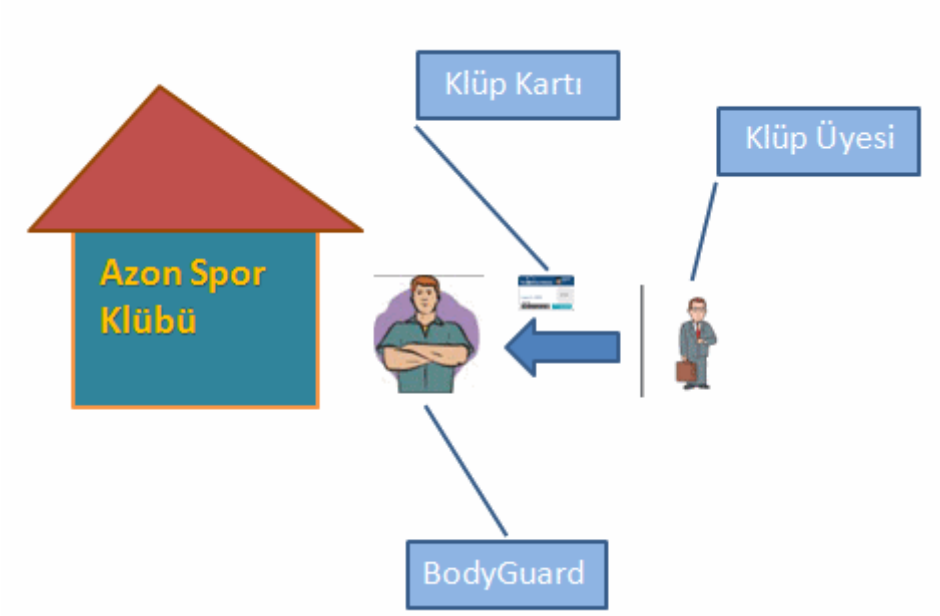
wcf,

WCF(Windows Communication Foundation) mimarisini baz alan **SOA(Service Oriented Applications)** uygulamaları geliştirilirken, güvenlik(security) başlığı altında ele alınmakta olan pek çok konu vardır. Geliştirilen bir WCF servisinin sadece izin verilen istemciler(clients) tarafından kullanılmasıda bu konulardan bir tanesidir. Bu amaçla istemci uygulamaların veya onları kullanan hesapların servis tarafında **doğrulanması(authenticate)** ve **yetkilendirilmesi(authorize)** adına bazı teknikler ele alınır. Temel olarak bir istemcinin doğrulanması ve yetkilendirilmesi, onun kim olduğunun bilinmesine **bağlıdır(Identification)**. Kimlik tespiti için kullanıcı adı-şifre, sertifika(certificate) yada Kerberos kartı(token) gibi elemanlar göz önüne alınır. Doğrulama işlemi sırasında kullanılan kimlik tespiti tekniklerinden biriside .Net Framework 3.0 ile birlikte gelen **Windows Cardspace** teknolojisidir.

NOT : *Windows CardSpace teknolojisi* ***Windows Vista*** *ile birlikte doğrudan gelmektedir. Nitekim Vista varsayılan olaran .Net Framwork 3.0 yüklü olarak yayınlanmaktadır. Diğer taraftan****.Net Framework 3.0*** *yüklendiğinde Windows XP sürümlerinde de Windows CardSpace teknolojisi kullanılabilmektedir.*

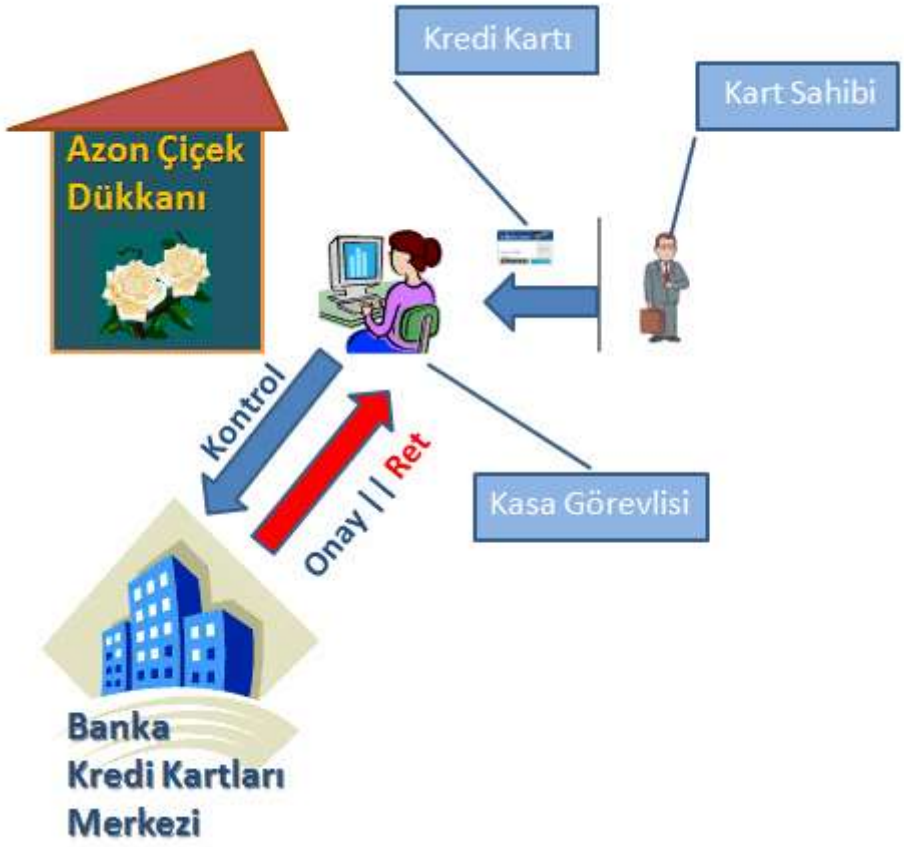
CardSpace teknolojisi sayesinde, istemciler kendi hazırladıkları kart bilgilerini güvenli bir şekilde servis uygulamasına iletebilirler. Eğer istemcilerin gönderdiği kart bilgileri içerisinde servisin ilgilendiği bilgiler var ise **doğrulama(authentication)** gerçekleşmiş olur. Bu aşamadan sonra ise doğrulanan kullanıcı için yine **yetkilendirme(authorization)** işlemlerine geçilir. İşte bu makalemizde WCF(Windows Communication Foundation) ile geliştirilen servis yönelimli uygulamalarda CardSpace teknolojisini nasıl kullanabileceğimizi incelemeye çalışıyor olacağız. Her zamanki gibi konuyu daha iyi kavrayabilmek adına örnek bir senaryo ve uygulama üzerinden adım adım ilerliyor olacağız. Geliştireceğimiz örnekler **Windows Vista Business** işletim sistemi yüklü bir makine üzerinde geliştirilecektir. Ancak başlamadan önce **kimlik(Identity)** kontrolünü daha iyi kavrayabilmek adına aşağıdaki gerçek dünya senaryoları göz önüne alınabilir.

Birinci Senaryo; Kimlik bilgilerindeki detayların önemli olmadığı durumlar.



Yukarıdaki şekildeki senaryoya göre bir spor klübüne girmek isteyen bir birey yer almaktadır. Kapıdaki iri kıyım görevlinin bu kişiyi içeri almasının tek şartı üye kartının(Membership Card) mevcut olmasıdır. Bunun dışında üye kartında yer alan detay bilgilerinin iri kıyım kapı görevlisi için hiç bir önemi bulunmamaktadır. Söz gelimi kart üzerinde yer alan üyelik numarası, üye adı veya üye sahibinin doğum tarihi gibi bilgiler çokda önemli değildir. Geçerli bir kartın olması yeterlidir. Bu senaryodaki yaklaşım maç günlerinde sadece kendi üyelerine açık olan klüp fanları içinde düşünülebilir.

İkinci Senaryo; Kimlik bilgisinin üçüncü bir sistem tarafından sağlandığı ve kontrol edildiği durumlar.



Bu senaryoda çicek dükkanından kredi kartı ile alışveriş yapmakta olan bir kişi yer almaktadır. Kredi kartı ile yapılan alışverişlerde kartın üzerinde yer alan bilgiler kasa görevlisi tarafından olmasada banka merkezine bağlanan pos cihazı açısından önemlidir. Bu tip bir durumda kartın gerçekten kullanan kişiye ait olduğunun anlaşılması gerekmektedir. Diğer taraftan kart sahibinin bu alışverişi yapması için yeterli hakka sahip olup olmadığıda önemlidir. Eğer bu haklara sahip ise son aşamada kart sahibi olduğunu ispat etmek için uzun zamandır ülkemizdede uygulanan pin numarasınıda girmesi gerekmektedir. Burada kartın orjinal ve bakiyesinin yeterli olup olmadığını banka sistemi tarafından kontrol etmektedir.

Bu senaryolar kulağa hoş ve mantıklı gelsede, acaba WCF(Windows Communication Foundation) ve Windows CardSpace ile aralarında nasıl bir ilişkileri vardır? Burada bahesedilen senaryolar **hak-tabanlı güvenlik (claims-based security)** vakkalarına örnek olabilecek gerçek dünya yansımalarıdır. Bu tip bir güvenlik sisteminde kişilerin kim olduğundan ziyade, yapılması istenen işlemler için ilgili kişinin hakkı olup olmadığının tespit edilmesi önemlidir. İşte WCF mimarisi altında geliştirilen uygulamalarda **Windows CardSpace** teknolojisini sayesinde hak-tabanlı güvenlik (claims-based security) **altyapısı(infrastructure)** sağlanabilir. Hak-tabanlı güvenlik(Claim Based Security) aslında üç ana unsurdan oluşmaktadır. Bunlar aşağıdaki listede belirtildiği gibidir.

- **Kullanıcı(Subject):** Servise erişmek ve fonksiyonelliklerini çalıştırmak isteyen kullanıcı veya farklı bir sistem asıl öznenin(subject) kendisidir. Kullanıcı, erişmek

istediği servise, onun kabul edebileceği haklar sunmakla yükümlüdür. Yukarıdaki senaryolar göz önüne alındığında, kredi kartı ile klüp kapısından girmek ne kadar mantıksızsa, klüp kartı ile çiçekçiden çiçek satın almakta o kadar mantıksızdır. Bir başka deyişle istemci tarafından sunulan hakların, servis tarafında kabul edilebilir nitelikte olması gerekmektedir.

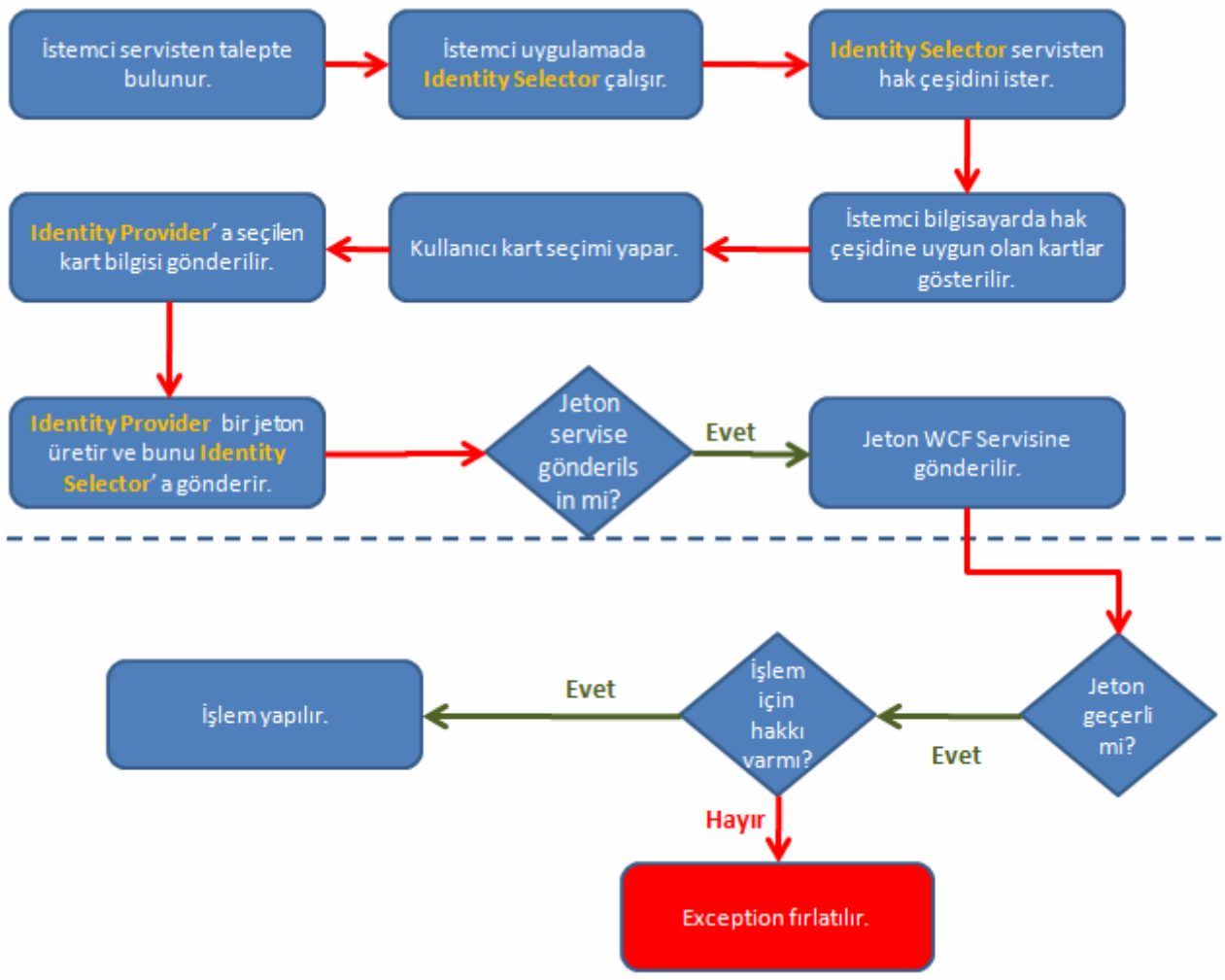
- **Kimlik Sağlayıcı (Identity Provider):** Adındanda anlaşılacağı üzere, haklar için gerekli kimliği sağlamakta olan organizasyon veya varlıktır(entity). Yukarıdaki senaryolar göz önüne alındığında kredi kartını veren banka veya üye griş kartını veren fan klübü, kimlik sağlayıcı konumundadır.
- **Güvenilir Şahıs veya Grup(Relying Party):** Koruma altına alınmış hizmeti**(Protected Service)** sunan organizasyon yada varlıktır. Kimlik sağlayıcısına(Identity Provider), kullanıcının verdiği kartın, kullanıcının yapmak istediği işlem ile ilişkili haklara sahip olup olmadığını sormakla yükümlüdür. Söz gelimi kredi kartı ile alışveriş işlemini tasvir eden senaryoda çiçekçi yada bir başka deyişle **satıcı(vendor) güvenilir şahıs(Relying Party)** rolündedir.

Windows CardSpace kullanılaraktan istemciler farklı bilgiler içeren çeşitli **bilgi kartları (Information Card)** oluşturabilirler. Servis tarafında yer alan uygulamanın kendisi, doğrulayacağı kullanıcılardan gelecek olan bilgi kartlarını kendi belirleyeceği **politikalara(Policy)** göre kontrol edebilir. Bilgi kartları içerisinde çok farklı veriler yer alabilir. Kullanıcının adı, email adresi, yaşı, doğum tarihi, hatta sürücü belgesi veya pasaportu ile ilgili bilgiler dahi olabilir. Bu noktada **Windows CardSpace** ile **hak tabanlı güvenlik(Claim-Based Security)** ortamı sunulan bir WCF uygulamasında, istemcinin bir servis talebi sonrası neler olacağına bakmakta yarar vardır. Aşağıdaki maddelerde, istemci(client) ve WCF servisi(Service) arasında hak-tabanlı güvenlik(Claim-Based Security) gerçekleştirildiğinde izlenen sürece ait adımlar yer almaktadır.

- İlk olarak istemci(Client), servisten(WCF Service) bir talepte(Request) bulunur.
- Sonrasında, istemci uygulama üzerinde çalışan **WCF çalışma zamanı(runtime),** Windows CardSpace içerisinden kimlik seçici uygulamayı**(Identity Selector)** çağırır.
- Kimlik seçici(Identity Selector), servisten hak çeşidini**(Claim Type)** talep eder. Bir başka deyişle hakların neye göre doğrulanacağını ister. örneğin kullanıcının pin numarası, email adresi, ev telefonu vb... bir hak çeşidi olarak elde edilebilir. *(Bu bilgi elbetteki WCF servis uygulaması ve istemci yazılırken konfigursayon içerisinde belirtilir.)*
- İstemci uygulamada çalışan **Identity Selector**, servisten gelen hak çeşidini bünyesinde barındırdan kartları kullanıcıya görsel bir arayüz ile sunar.
- İstemci uygulamayı çalıştırmakta olan kullanıcı bir kart seçimi gerçekleştirir. *(Kullanıcılar istemci uygulamanın çalıştığı sistemde **Windows CardSpace**' i kullanarak istedikleri biçimde kart bilgileri oluşturabilirler. Bunu kart seçim aşamasında dahi yapabilirler.)*
- İstemci uygulamada çalışan Identity Selector programı, **kimlik sağlayıcı(Identity Provider)** ile iletişim kurar ve istemciye ait kart bilgisi içinden hak çeşidi(Claim Type) ile ilişkili olanını gönderir.

- Kimlik sağlayıcısı(Identity Provider) gelen metadata bilgisini alır ve bir **fiş(token)** üreterek bunu Kimlik seçiciye(Identity Selector) gönderir.
- Identity Selector istemci programı çalıştıran kullanıcıya, oluşturulan fişin WCF servisine gönderilmesini onaylayıp onaylamadığını sorar. Eğer kullanıcı onaylarsa seçtiği bilgilere göre oluşturulan fiş(token) WCF servisine gönderilir.
- WCF Servisi gelen fiş bilgisini alır ve kullanıcının hakkının doğruluğunu kontrol eder. Eğer fiş(token) bilgisi geçerli ise içerisinde yer alan **kimlik bilgisine(Identity Information)**bakarak kullanıcının talepte bulunduğu operasyonu yapıp yapamayacağına karar verir. Bir başka deyişle bu adımda **yetkilendirme(authorization)** durumu ele alınır. Eğer yetki verilirse istemcinin talep ettiği fonksiyonellik çalıştırılır.

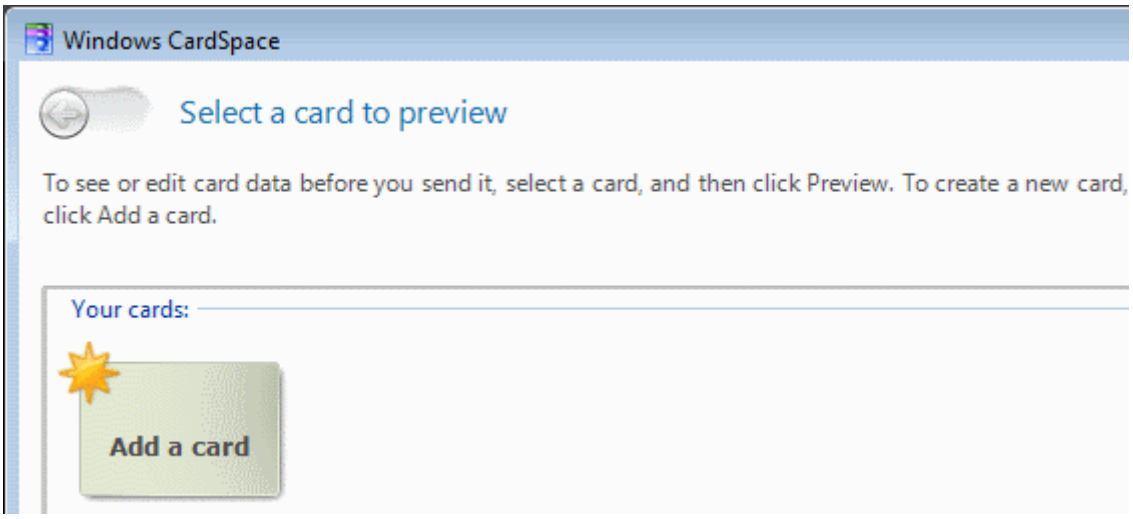
Durumu biraz daha görselleştirmek adına aşağıdaki akış diagramından da yararlanılabilir.



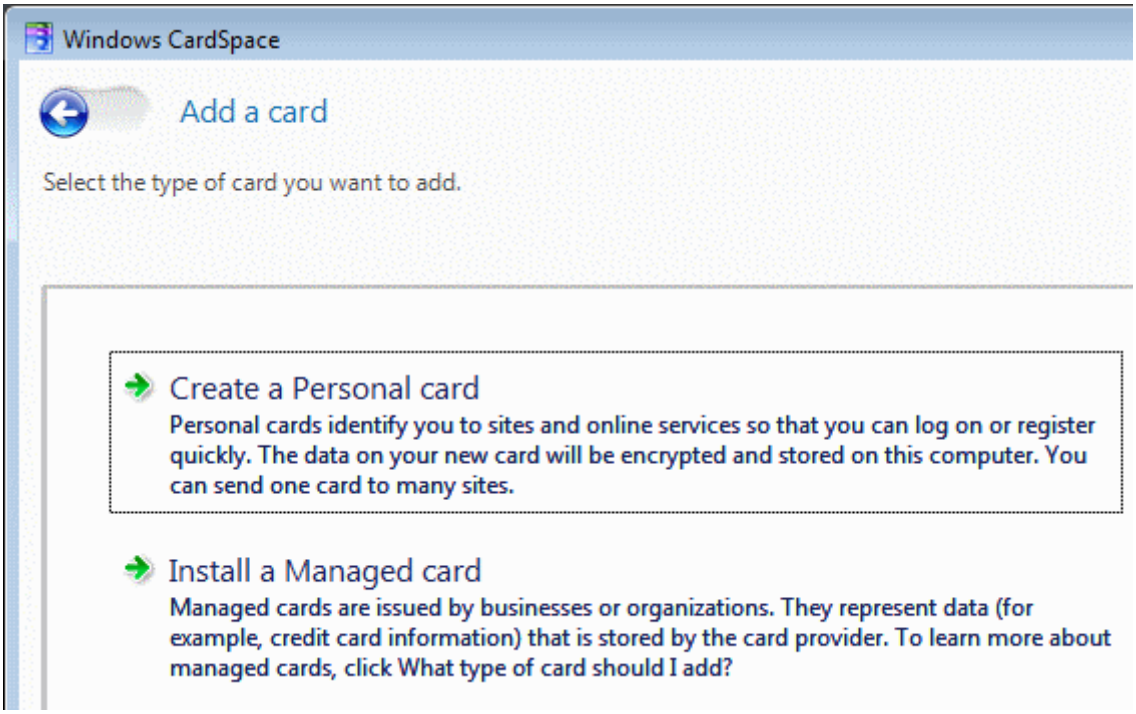
Bu kadar teorik bilgiden sonra örnek bir senaryo üzerinden hareket etmekte yarar vardır. örnekte servis tarafında basit bir fonksiyonellik sunulmaktadır. Söz gelimi standart toplama fonksiyonunu içeren bir WCF servis uygulaması tasarlanabilir. Servis ve istemci uygulamalar aynı makine üzerinde yer almaktadır. Bunun dışında servis ve istemcilerin **mesaj seviyesinde güvenli (Message Level Security)**bir şekilde haberleşmeleri önemlidir. Burada mesaj seviyesindeki haberleşmenin güvenli olabilmesi

için **sertifika(Certificate)** kullanımı tercih edilmiştir. İlk önce işe, istemci tarafında Windows CardSpace teknolojisini kullanarak basit bir **bilgi kartı(Information Card)** hazırlayarak başlamakta yarar vardır. Bu amaçla **Windows Vista** üzerinde **Control Panel** içerisinde yer alan **Windows CardSpace** programının kullanılması yeterlidir. *(Windows XP tabanlı sistemlerde de Windows CardSpace uygulamasına yine Control Panel üzerinde erişilebilir)*

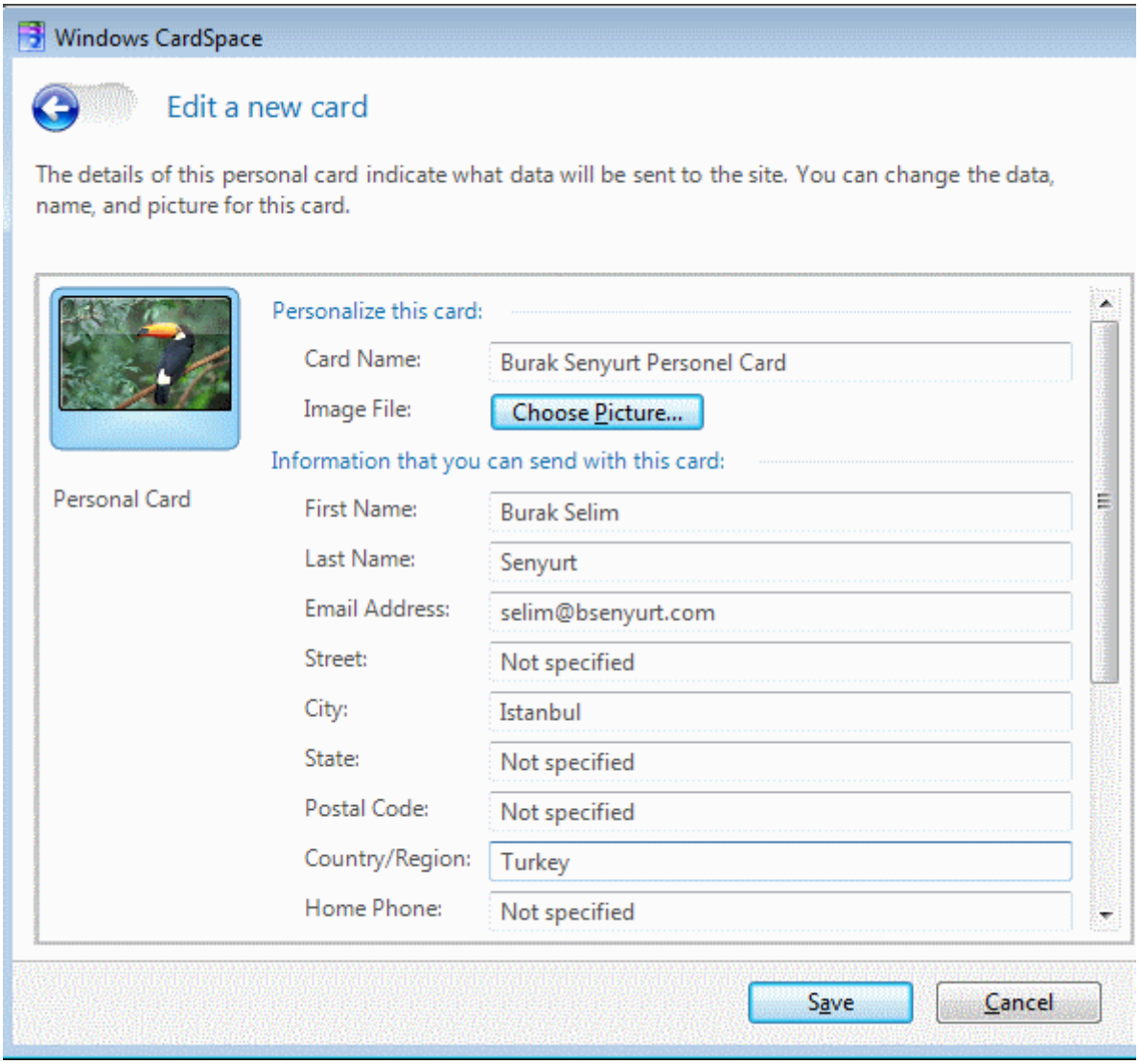
Eğer daha önceden yüklenmiş bir bilgi kartı yok ise aşağıdaki gibi bir ekran ile karşılaşılacaktır.



Buradan **Add a Card** opsiyonunu işaretlenerek devam edilir.



Sıradaki adımda ise **Create a Personel card** opsiyonunu seçilir. Şu aşamada, kredi kartı yada pasaport bilgisi gibi verileri saklayacak bir bilgi kartı(Information Card) oluşturulmadığından **Install a Managed Card** seçeneğini ele alınmamaktadır. İzleyen adımda personel kart bilgilerinin girilmesi gerekmektedir. Bu ekran örnek olarak aşağıdaki gibi doldurulabilir.

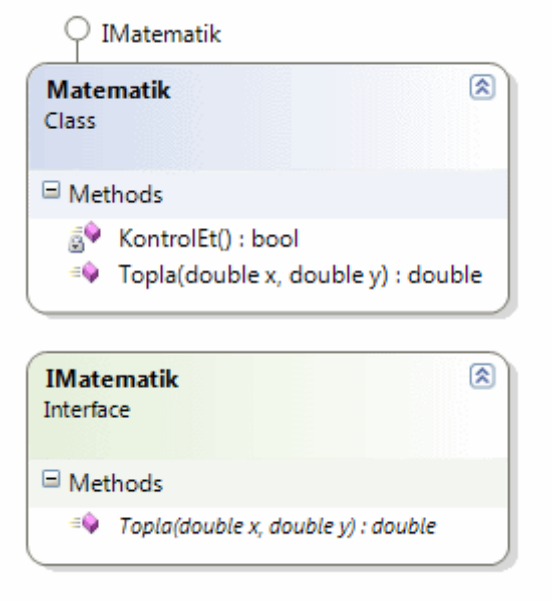


Hak(Claim) işlemleri email adresi üzerinden yapılacağı için bu bilginin mutlaka girilmesi önemlidir. İşlemler tamamlandıktan sonra hazırlanan kartın aşağıdaki gibi eklendiği görülecektir.

Test amacıyla Garfi isimli hayali kişi için bir test kartı daha oluşturulmuştur. Bu kişinin email adresindeki bilgiye göre, servis üzerinde gerekli fonksiyonelliği çalıştırma yetkisi olmayacaktır. Amaç böyle bir durumda servisin nasıl bir davranış sergileyeceğinin izlenmesidir.

Artık servis tarafı tasarlanmaya başlanabilir. Servis uygulaması web üzerinden **WsFederationHttpBinding** tipini baz alacak şekilde tasarlanacaktır. Burada WsHttpBinding tipide göz önüne alınabilir. Federasyon yada birlik(Federation), farklı sistemler arasında doğrulama(authentication) ve yetkilendirme(authorization) adına kimlik bilgilerinden*(örnekteki Windows CardSpace ile oluşturulanlar gibi)* yararlanılmasını sağlayan bir kavramdır. Federation tarafından göz önüne alınan kimlik(Identity) bilgileri bir bilgisayar yada kullanıcıyı işaret edebilir. WsFederationHttpBinding bunun için gerekli olan alt yapıyı sunan hazır bir **bağlayıcı tiptir(Binding Type)** ve WS-Federation protokolünü desteklemektedir. Bu sebepten WsFederationHttpBinding tipi **iletişim seviyesinde güvenliği(transport level security)** desteklemez ve **HTTP** üzerinden iletişimi zorunlu kılar. Tekrardan uygulamaya dönülecek olursa; servis kütüphanesi**(WCF Service Library)** içerisinde basit olarak toplama fonksiyonelliği sunulmaktadır. WCF servis kütüphanesindeki tiplerin şematik gösterimi ve kod içerikleri aşağıda olduğu gibidir.

Sınıf Diagramı(Class Diagram);



Sözleşme(IMatematik);

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.ServiceModel;

namespace MatematikKutuphanesi
{
   [ServiceContract(Name="Matematik
Servisi",Namespace="http://www.bsenyurt.com/Matematik/MatematikServisi")]
   public interface IMatematik
   {
      [OperationContract]
      double Topla(double x, double y);
   }
}
```

Matematik.cs;

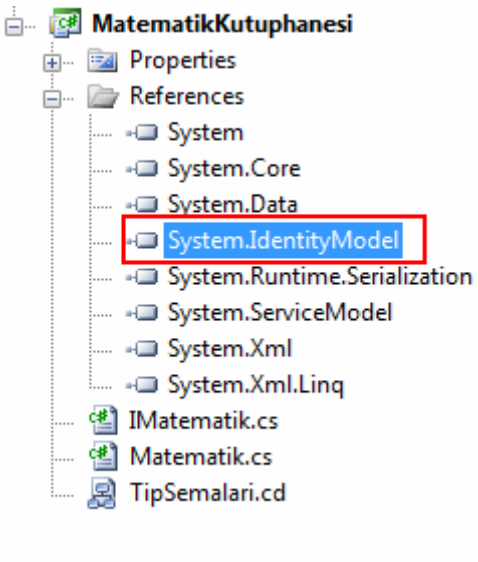
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
// System.IdentityModel.dll referans edilmelidir.
using System.IdentityModel.Claims;
using System.IdentityModel.Policy;
using System.ServiceModel;
```

```
namespace MatematikKutuphanesi
{
   public class Matematik:IMatematik
   {
      #region IMatematik Members

      public double Topla(double x, double y)
      {
         // önce talepte bulunan istemcinin ilgili işlem için hakkı olup olmadığı kontrol edilir.
         if (KontrolEt())
            return x + y;
         else
            throw new Exception("Doğrulanmadı");
      }

      private static bool KontrolEt()
      {
         // Doğrulama içeriği çekilir
         AuthorizationContext ctx = OperationContext.Current.ServiceSecurityContext.AuthorizationContext;
         // Doğrulama içeriğindeki her bir ClaimSets gezilir
         foreach (ClaimSet cSet in ctx.ClaimSets)
         {
            // ClaimSet' ler içerisinde Email hakkını içeren Claim' ler gezilir
            foreach (Claim clm in cSet.FindClaims(ClaimTypes.Email, Rights.PossessProperty))
            {
               // O anki Claim' in Email değeri selim@bsenyurt.com ise metod geriye true döndürür.
               if (clm.Resource.ToString() == "selim@bsenyurt.com")
                  return true;
            }
         }
         return false;
      }

      #endregion
   }
}
```

Bu sınıf içerisinde yer alan Topla metodunu, gönderdiği kart bilgisinde yer alan email adresi selim@bsenyurt.com olan kullanıcı çalıştırabilir. Bunun kontrolü için KontrolEt isimli geriye **bool**değer döndüren bir metod geliştirilmiştir. Bu metod kendi içerisinde,

istemciden gelen **fiş(token)** bilgisi ve içerisindeki **Claim Set**' lerin elde edilmesi işlemlerini gerçekleştirilir. İstemciden gelen fiş(token) bilgilerinin içeriğine bakabilmek için elde edilen **AuthorizationContext** referansının **ClaimSets** koleksiyonuna gidilir. **ClaimSets** koleksiyonu içerisinde yer alan **ClaimSet** bilgilerinden **hak tipi(Claim Type)** email adresi olanların yakalanabilmesi içinde **FindClaims** metodu kullanılır. FindClaims metodunun döndüreceği koleksiyonun her bir elemanıda **Claim** tipindendir. Claim referanslarının **Resource** özelliklerinin değerleri **object** tipindendir. Bunun sebebi hak tipine göre gelen verinin farklı tiplerde olabilmesidir. Bunu geçerli bir mail adresi ile kıyaslamak için **ToString** metodu ile string tipine dönüştürme işlemi yapılmıştır. Burada elbetteki email adresi kontrolünü bir veritabanı(database) kaynağından yada hakkı olan email adreslerinin tutulduğu bir XML dosyasından yapmak çok daha mantıklıdır. örneğin amacı şu aşamada sadece test olduğundan bu işlemler göz ardı edilmiştir.

***NOT : AuthorizationContext** tipi **System.IdentityModel.Policy isim alanı(Namespace)** altında, **ClaimSets** ve **Claim** tipleri ise **System.IdentityModel.Claims** isim alanı altındadır. Her iki isim alanıda **System.IdentityModel.dll** assembly' ında yer aldıklarından söz konusu dll referansının projeye açıkça eklenmesi gerekmektedir. Aşağıdaki şekillde System.IdentityModel referansının MatematikKutuphanesi isimli WCF Servis Kütüphanesine eklenmiş hali görülmektedir.*

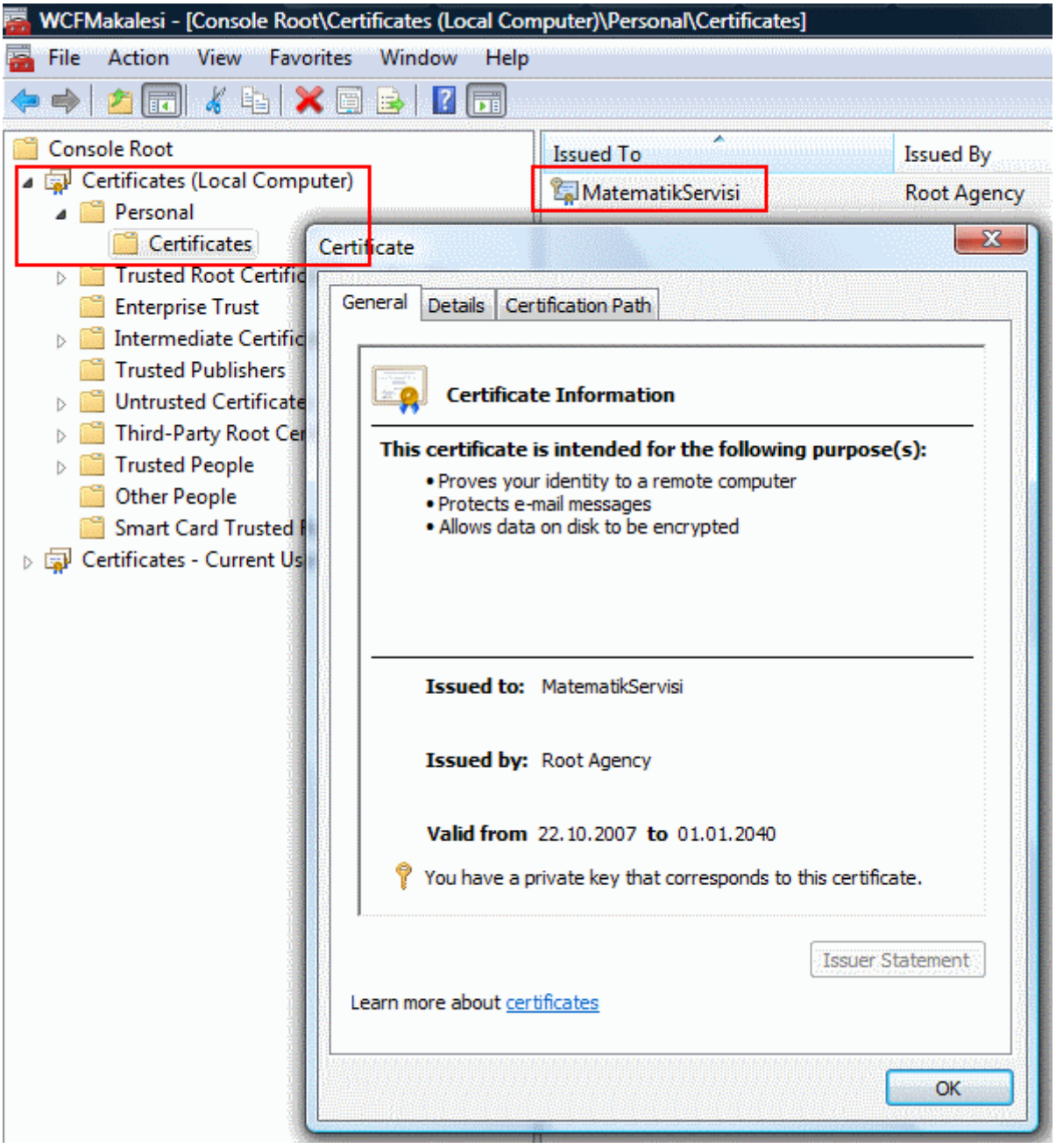


Artık servis tarafında, ilgili sözleşmeyi ve fonksiyonelliği sunacak olan uygulama tasarlanabilir. Burada istemci ve servis arasında kart bilgileri taşınacağından **güvenilir bir ortam(Reliable Session)** hazırlanması gerekmektedir. Bu nedenle servisin kart bilgisi gönderecek olan istemcileri ile, **sertifika(certificate)** aracılığıyla haberleşmesi gerekmektedir. Bir başka deyişle servis uygulamasının **taleplerde(requests)** kullanacağı ve istemcilerinde referans edeceği bir sertifika tanımlamasının yapılması gerekmektedir. Bu amaçla ilk olarak, servis uygulaması için gerekli test sertifikasını üreterek işe başlanabilir. Bu sertifikanın üretimi ve servisin çalıştığı makine hesabına kaydı için **Visual Studio 2008**

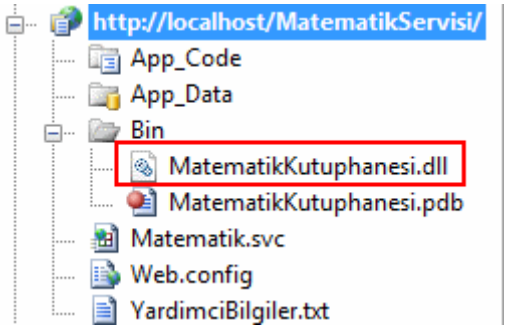
Beta 2 Command Prompt üzerinden**makecert.exe** aracının aşağıdaki gibi kullanılması yeterlidir. **Makecert** aracı **X.509** tabanlı test sertifikalarının üretilmesinde kullanılmaktadır.

makecert -sr LocalMachine -ss My -n CN=MatematikServisi -sky exchange

sr ile sertifikanın yükleneceği **yer(location)** belirtilirken, ss sonrasında gelen My ilede **Certificate Store** bilgisi tanımlanır. CN ifadesinden sonra oluşturulacak olan sertifikanın adı belirlenir. sky parametresinden sonra gelen değer ilede subject' in anahtar tipinin(Key Type) ne olacağı belirtilir. Burada **exchange** dışında **signature** değeride verilebilir. *(Makecert aracının kullanımı ile ilişkili olarak daha detaylı bilgi için http://msdn2.microsoft.com/en-us/library/bfsktky3(VS.80).aspx adresinden yardım alınabilir.)* Bu komutun çalıştırılmasının ardından **Microsoft Management Console(MMC)** yardımıyla sertifika ayarlarına bakıldığında aşağıdaki ekran görüntüsünde olduğu gibi MatematikServisi isimli test sertifikasının başarılı bir şekilde Local Computer altında yer alan Personal kısmı altına eklendiği görülür.



Sertifika tanımlaması yapıldığına göre servis tarafındaki uygulamanın yazılması ile işlemlere devam edilebilir. Servis uygulaması IIS üzerinde host edilmek üzere tasarlanmalıdır. Bu amaçla yeni bir **WCF Service** şablonu oluşturulur. Servis uygulaması, MatematikKutuphanesi isimli **servis sınıf kütüphanesini(WCF Service Library)** referans etmelidir.



Servis tarafında yer alan Matematik.svc dosyasının içeriği aşağıdaki gibidir.

```
<%@ ServiceHost Language="C#"
Debug="true" Service="MatematikKutuphanesi.Matematik" %>
```

Servis tarafındaki belkide en önemli kısım konfigurasyon dosyasının içeriğidir. Nitekim burada kullanılacak olan sertifikanın bildirimi, hak(Claim) tipinin ne olacağı gibi ayarlamaların yapılması gerekmektedir. Bu noktada **Microsoft Service Configuration Editor** yardımıyla görsel olarak hazırlanan web.config dosyasındaki **ServiceModel** elementinin içeriği aşağıdaki gibidir.

```
<system.serviceModel>
  <bindings>
    <wsFederationHttpBinding>
      <binding name="MatematikServisiBindingConf" transactionFlow="true">
        <reliableSession enabled="true" />
          <security>
            <message
issuedTokenType="urn:oasis:names:tc:SAML:1.0:assertion">
              <claimTypeRequirements>
                <add
claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"
isOptional="false" />
              </claimTypeRequirements>
            </message>
          </security>
        </binding>
      </wsFederationHttpBinding>
    </bindings>
  <behaviors>
    <serviceBehaviors>
      <behavior name="MatematikServisiBehavior">
        <serviceMetadata httpGetEnabled="true" />
        <serviceCredentials>
          <serviceCertificate findValue="MatematikServisi"
x509FindType="FindBySubjectName" />
```

```
            <issuedTokenAuthentication allowUntrustedRsaIssuers="true" />
          </serviceCredentials>
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service behaviorConfiguration="MatematikServisiBehavior"
name="MatematikKutuphanesi.Matematik">
        <endpoint address="http://localhost/MatematikServisi/Matematik.svc"
binding="wsFederationHttpBinding"
bindingConfiguration="MatematikServisiBindingConf"
name="MatematikServisiEndPoint" contract="MatematikKutuphanesi.IMatematik" />
      </service>
    </services>
</system.serviceModel>
```

Oluşturulan dosyada dikkat edilmesi gereken bir kaç nokta vardır. İlk olarak istemci ile servis arasında güvenilir bir oturum açılması gerekmektedir. Bu nedenle bağlayıcı tipe ait olan**relaiableSession** elementinin **enabled** niteliğinin(attribute) değeri **true** olarak belirlenmiştir. Bir sertifika kullanımı söz konusu olduğundan servis davranışlarından **serviceCredentials** ayarlarının yapılması gerekmektedir. Bunu sağlayabilmek için serviceCredential elementi aşağıdaki gibi oluşturulmuştur.

```
<serviceCredentials>
            <serviceCertificate findValue="MatematikServisi"
x509FindType="FindBySubjectName" />
```

Burada **findValue** niteliğine verilen değer, daha önceden oluşturulan MatematikServisi isimli sertifikadır. Bu **X.509** tipindeki sertifikasının bulunabilmesi içinde **x509FindType** niteliğine**FindBySubjectName** değeri verilmiştir. Buna göre ilgili sertifika, nesne adına göre aranacaktır.

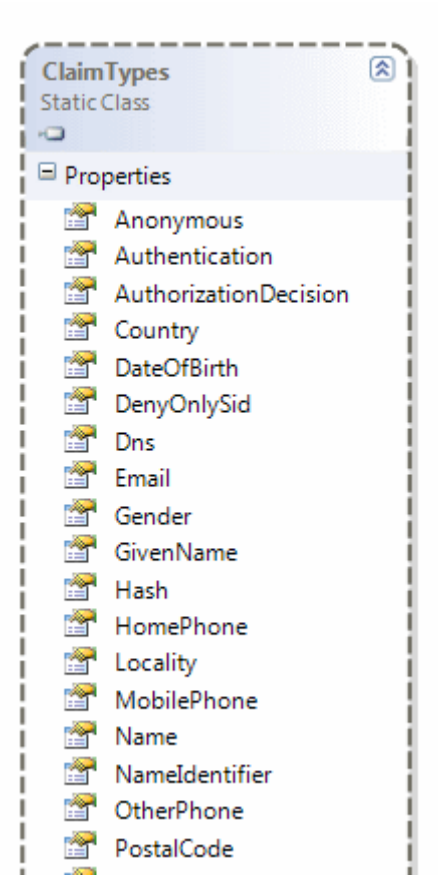
Servis tarafı, istemcinin talepte bulunduğu hizmetler için hakkı olup olmadığını, kart bilgisi ile gelen email adreslerine göre yapmaktadır. Burada **politika(policy)** olarak kart bilgisindeki email adresine bakılacağının söylenmesi gerekmektedir. Bu amaçla **security** elementi içerisinde aşağıdaki ayarlamalar yapılmıştır.

```
<security>
  <message issuedTokenType="urn:oasis:names:tc:SAML:1.0:assertion">
    <claimTypeRequirements>
      <add
claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"
isOptional="false" />
    </claimTypeRequirements>
```

```
    </message>
</security>
```

Burada en önemli nokta **claimType** niteliğinin değeridir. Bu niteliğe(attribute) **well-known URI** formatında bir değer atanmıştır. Değerin sonunda yer alan **emailaddress**, hak için gerekli politikayıda**(Claim Policy)** belirlemektedir.

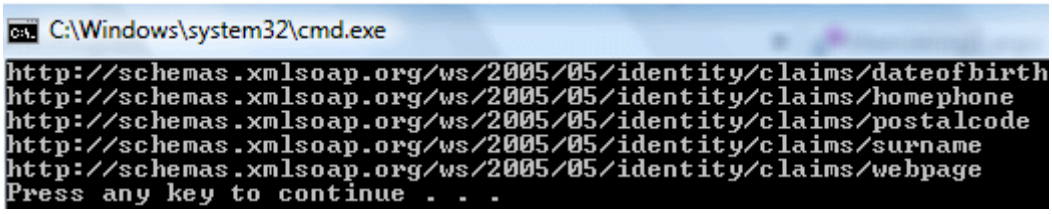
Burada yazılan URI bilgisi programatik olarak static olarak tanımlanmış **ClaimTypes** sınıfındaki özellikler(properties) üzerindende elde edilebilir. ClaimTypes sınıfının üyelerinin bir kısmının sınıf diagramındaki görüntüsü aşağıdaki gibidir. ClaimTypes sınıfı static bir sınıftır. Bu nedenle içerisindeki üyelerin tamamı static olarak tanımlanmıştır. Dolayısıyla bu üyelere nesne örneği olmadan tip adı ile erişilebilmektedir.



Söz gelimi çalışma zamanında aşağıdaki kod parçası kulanıldığında,

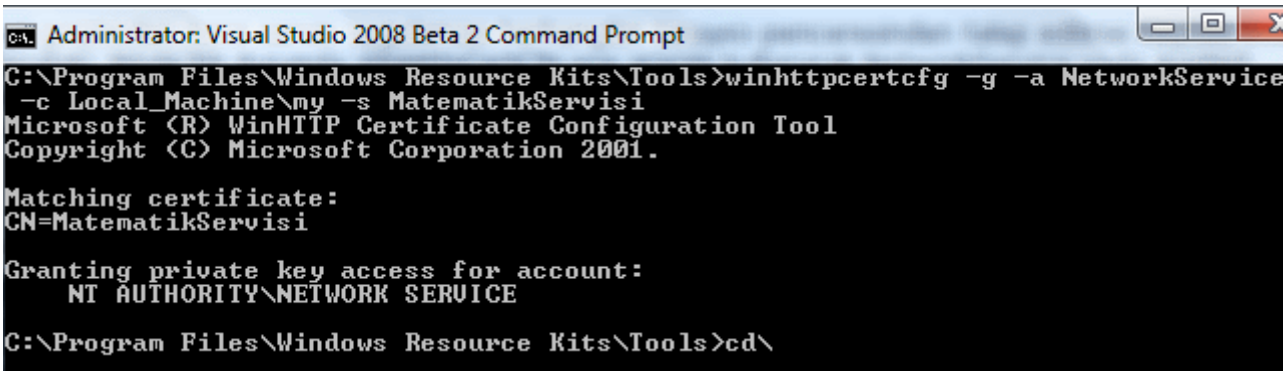
```
Console.WriteLine(ClaimTypes.DateOfBirth);
Console.WriteLine(ClaimTypes.HomePhone);
Console.WriteLine(ClaimTypes.PostalCode);
Console.WriteLine(ClaimTypes.Surname);
Console.WriteLine(ClaimTypes.Webpage);
```

Claim Tipi olarak kullanılabilecek URI bilgileride şu şekilde elde edilecektir. Burada örnek olarak doğum günü, ev telefonu, posta kodu, soyadı ve web sayfası gibi bilgiler için gereken Well-Known URI bilgileri gösterilmektedir.

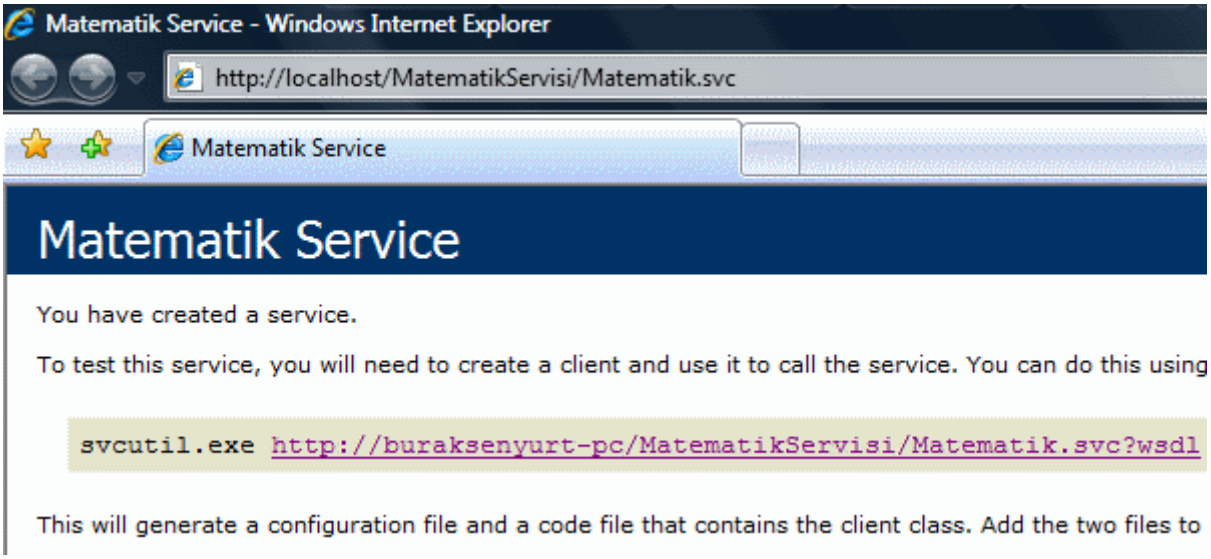


Diğer taraftan **message** elementi içerisinde **issuedTokenType** niteliğinede **urn:oasis:names:tc:SAML:1.0:assertion** değeri atanmıştır. Bu tanımlamaya göre WCF servisinin, **Secure Application Markup Language 1.0** uyumlu bir **fiş(token)** beklediği belirtilmektedir. SAML, **kimlik sağlayıcı(Identity Provider)** ile **servis sağlayıcı(Service Provider)** arasında doğrulama(authentication) ve yetkilendirme(authorization) verilerinin değiş tokuş şeklini belirleyen bir XML standardıdır. örnek uygulamada, aynı bilgisayardaki **Windows CardSpace** ile hazırlanmış kimlik bilgileri baz alınmaktadır. Bir başka deyişle üçüncü parti bir kimlik sağlayıcı(Identity Provider) tarafından üretilmiş bir kart mevcut değildir. Bu sebepten servisin **güvenilmez(untrusted)**kaynaklardan gelecek SAML fişlerinş(tokens) kabul edecek şekilde ayarlanması gerekir. Bunu sağlamak için **serviceCredentials** elementi altındaki **issuedTokenAuthentication** elemeninin**allowUntrustedRsaIssuers** seçeneğine **t rue** değeri atanmıştır.

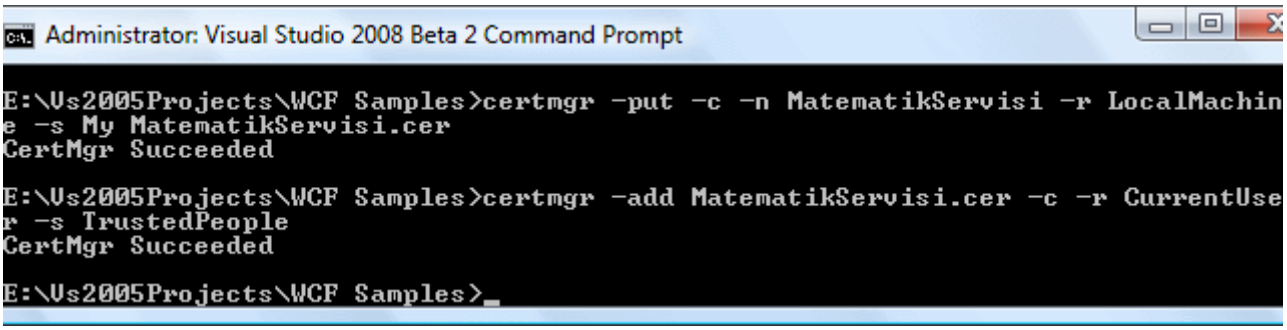
İstemci uygulamaya geçmeden önce servis uygulaması herhangibir tarayıcı penceresinden talep edilirse **KeySet Does Not Exist** mesajlı bir çalışma zamanı istisnası alınabilir. Böyle bir durumda **WinHttpCertCfg.exe** aracı kullanılarak **NetworkService** yada **AspNet** hesaplarına(accounts), uygulamada kullanılan sertifika(Certificate) için kabul edilmiş erişim**(grant access)** haklarının verilmesi gerekmektedir. WinHttpCertCfg aracı http://www.microsoft.com/downloads/details.aspx?familyid=c42e27ac-3409-40e9-8667-c748e422833f adresinde tedarik edilebilir. WinHttpCertCfg aracı aşağıdaki ekran görüntüsünde olduğu gibi kullanılmalıdır.



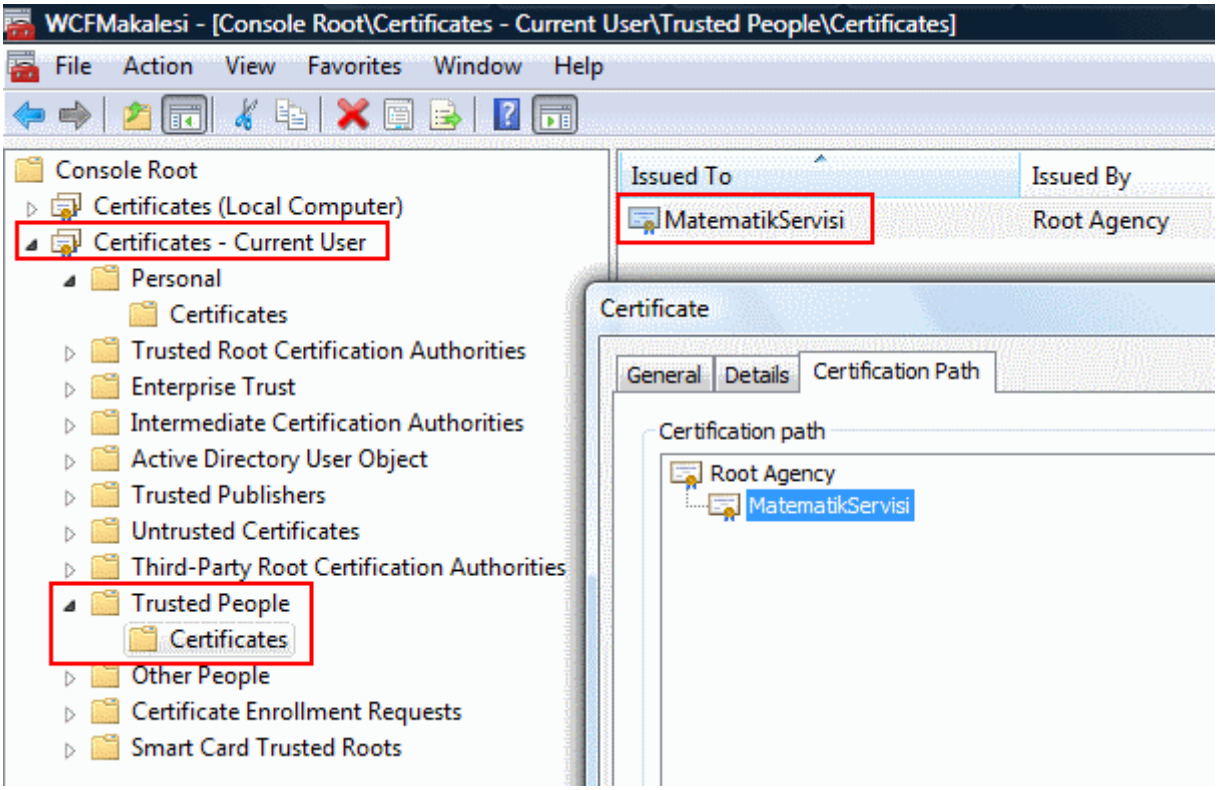
Yukarıdaki komut ifadesine göre NetworkService hesabı için, MatematikServisi üzerine grant access hakkı verilmiştir. Artık servis tarafı internet tarayıcısı üzerinden elde edilebilir.



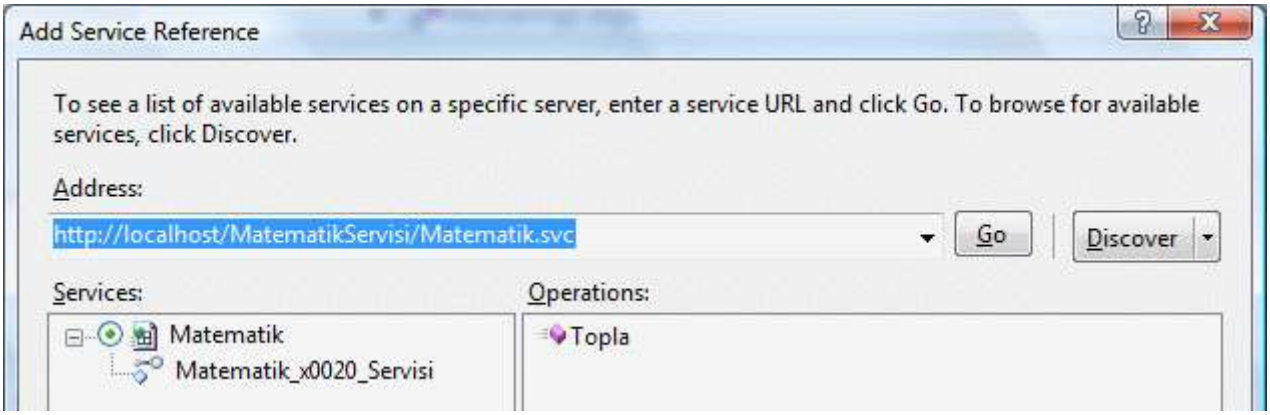
Artık istemci tarafını yazmak için gerekli hazırlıklara başlanabilir. Sistemin çalışabilmesi için, üretilen test sertifikasının öncelikli olarak o anki kullanıcı için **Trusted People** olarak eklenmesi gerekmektedir. Bunu sağlayabilmek için **Visual Studio 2008 Beta 2 Command Prompt** üzerinden **certmgr** aracı aşağıdaki görüldüğü gibi kullanılmalıdır.



İlk olarak söz konusu sertifikanın bir kopyası sadece **public key** değerini içerecek şekilde **MatematikServisi.cer** isimli dosyaya yazdırılır. Sonrasında çalıştırılan komut ilede, güncel kullanıcı**(Current User)** için **Trusted People** olacak şekilde eklenir. Bu işlemlerin ardından Microsoft Management Console programından yararlanılarak Current User altındaki Personel bölümüne bakıldığında aşağıdaki ekran görüntüsünde yer aldığı gibi ilgili sertifika bildiriminin yapıldığı görülür.



Bu ön hazırlıkları tamamladıktan sonra istemci uygulamanın geliştirilmesine başlanabilir. İstemci program basit bir Console uygulaması olarak ele alınmaktadır. Console uygulamasında, geliştirilen servisin **Add Service Reference** seçeneği ile aşağıdaki ekran görüntüsünde olduğu gibi eklenmesi gerekmektedir.



Sonrasında ise yine üretilen App.config dosyası istenirse görsel olarak istenirse doğrudan yazılarak aşağıdaki hale getirilmelidir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <endpointBehaviors>
        <behavior name="ClientEndPointBehavior">
```

```
            <clientCredentials>
              <serviceCertificate>
                <authentication certificateValidationMode="PeerTrust"
revocationMode="NoCheck" />
              </serviceCertificate>
            </clientCredentials>
          </behavior>
        </endpointBehaviors>
      </behaviors>
      <bindings>
        <wsFederationHttpBinding>
          <binding name="MatematikServisiEndPoint" transactionFlow="true">
            <reliableSession enabled="true" />
            <security mode="Message">
              <message algorithmSuite="Default" issuedKeyType="SymmetricKey"
issuedTokenType="urn:oasis:names:tc:SAML:1.0:assertion"
negotiateServiceCredential="true">
                <claimTypeRequirements>
                  <add
claimType="http://schemas.xmlsoap.org/ws/2005/05/identity/claims/emailaddress"
isOptional="false" />
                </claimTypeRequirements>
              </message>
            </security>
          </binding>
        </wsFederationHttpBinding>
      </bindings>
      <client>
        <endpoint address="http://localhost/MatematikServisi/Matematik.svc"
behaviorConfiguration="ClientEndPointBehavior" binding="wsFederationHttpBinding"
bindingConfiguration="MatematikServisiEndPoint"
contract="ServiceReference.MatematikServisi" name="MatematikServisiEndPoint">
          <identity>
            <certificateReference x509FindType="FindBySubjectName"
findValue="MatematikServisi" />
          </identity>
        </endpoint>
      </client>
    </system.serviceModel>
</configuration>
```

İstemci tarafındaki konfigurasyon dosyasının pek çok özelliği servis referansı eklendikten sonra otomatik olarak oluşturulur. Herşeyden önce istemci tarafında da bağlayıcı tip olarak**wsFederationHttpBinding** ele alınmaktadır. Diğer taraftan istemci ile servis arasında **güvenilir bir oturum için(reliable session)** gerekli ayarlamalar yapılmıştır.

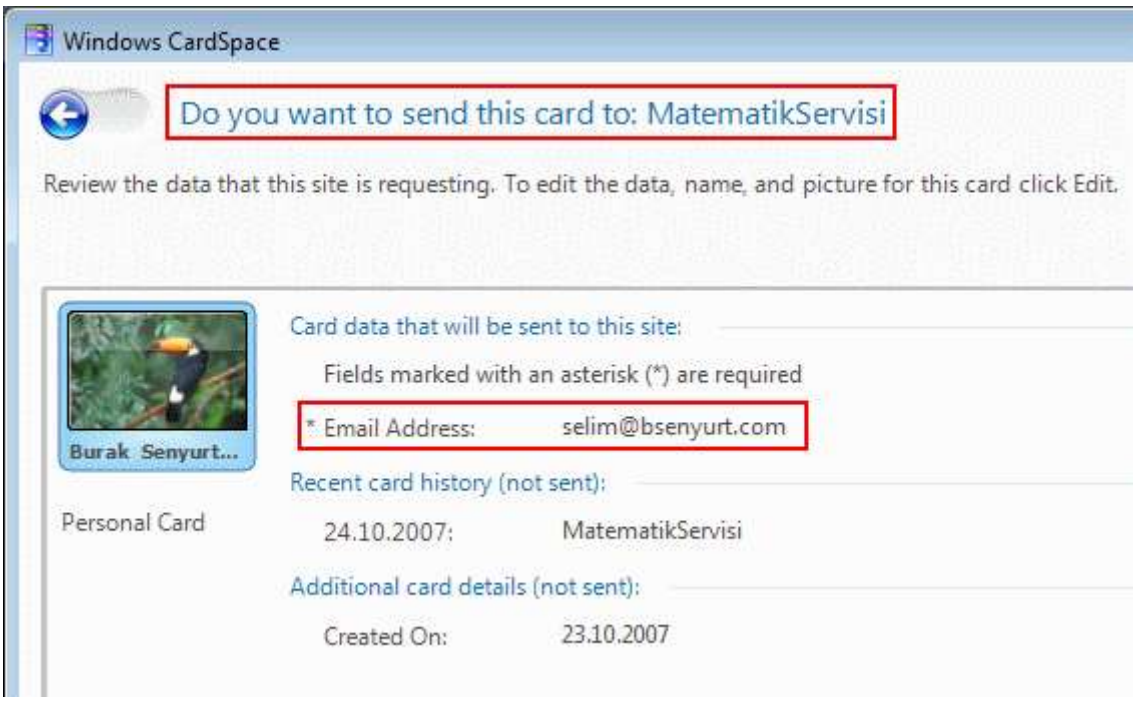
Aynen servis tarafında olduğu gibi **hak tipi(Claim Type)** email olacak şekilde belirlenmiş ve **issuedTokenType** niteliğine atanan değer ile **SAML** standardında bir **fiş(token)** yayınlanacağı bildirilmiştir. Sertifika(Certificate) bildirimi **Identity** elementi içerisinde yapılmaktadır. Servis tarafındakine benzer olarak sertifika adı ve neye göre aranacağı belirtilmektedir **findValue** ve **x509FindType** nitelikleri ile belirtilmektedir.

Konfigurasyon dosyasında gerekli değişiklikler yapıldıktan sonra Main metodu içerisinde aşağıdaki gibi bir kod bloğu geliştirilerek servisin kullanılması sağlanabilir.

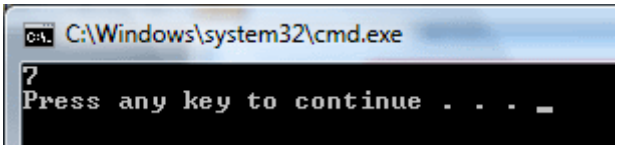
```
using System;
using System.Collections.Generic;
namespace CardSpaceIstemci
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            ServiceReference.MatematikServisiClient cli =
new CardSpaceIstemci.ServiceReference.MatematikServisiClient();
            Console.WriteLine(cli.Topla(3, 4).ToString());
         }
         catch (Exception excp)
         {
            Console.WriteLine(excp.Message);
         }
      }
   }
}
```

Servis tarafında **includeExceptionDetailsInFaults** özelliğinin değerini **true** olarak belirlenmiş olduğundan, servis tarafından fırlatılacak olan istisna mesajları(Exception Messages) istemci uygulama üzerinden kolaylıkla ele alınabilecektir. Tüm bu işlemlerin ardından istemci uygulama çalıştırıldığında, ekrana kart seçimi yapılması için bir sorgu penceresi açılacaktır. Bu sorgu penceresinde var olan kartlardan yararlanılabilir yada yeni bir kart oluşturularak gönderilmesi sağlanabilir.

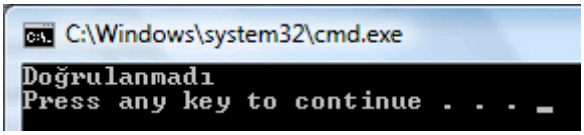
İlk olarak Burak Senyurt Personel Kartı isimli bilgi kartı(Information Card) seçildiğinden, kullanıcıya bir soru sorulacaktır. Bu soruda kullanıcının ilgili kart bilgisini servise göndermek isteyip istemediği belirtilir. Kullanıcı bunu kabul etmesse bir başka deyişle örneğin Esc tuşu ile arabirimden çıkarsa istemci tarafında yine bir çalışma zamanı hatası oluşacak ve bununla ilişkili bilgiler Log dosyasına aktarılacaktır.



Send başlıklı düğmeye bastıktan sonra aşağıdaki ekran görüntüsünde olduğu gibi servis tarafında yapılan toplama işleminin sonucunun elde edilebildiği görülür.



Ancak aynı uygulamada Garfi isimli diğer kart bilgisi gönderildiğinde servis tarafında üretilen istisna(Exception) mesajının alındığını görürüz. Bir başka deyişle istemci uygulamadan gönderilen kart bilgisi içerisindeki email adresine göre, servis tarafında toplama fonksiyonunun çalıştırılması için gereken hak koşulları sağlanamamıştır.



Görüldüğü gibi **Windows CardSpace** kullanılarak, WCF uygulamalarında **hak tabanlı güvenliği(Claim Based Security)** sağlamak oldukça kullanışlı ve etkili bir yoldur. Burada, teknik detaylara çok fazla girilmeyerek adım adım bu tarz bir sistemin nasıl kurulabileceğinden bahsedilmeye çalışılmıştır. Nevarki kullanılan sertifika bir test sertifikası olup üçüncü parti bir sağlayıcı tarafından üretilen kart bilgileri ele alınmamıştır. Bu tarz gerçek senaryo uygulamalarındada sistemin tasarlanması ve kuralları çok fazla değişiklik göstermeyecektir.

Bu konu ile ilişkili detaylı bilgiyi benimde yakından takip ettiğim ve faydalandığım **John Sharp** imzalı **MsPress** yayınlarına ait **Microsoft Communication Foundation Step by Step** kitabından da bulabilirsiniz. Sanılanın aksine Step by Step olarak belirtilmesine rağmen konu WCF olunca oldukça zor ve sıkı çalışılması gereken bir kitap olduğunuda vurgulamak isterim. Böylece geldik uzun bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

WCFCardSpace.rar (51,15 kb)

Daha Etkili Profil Yönetimi (2007-10-17T05:12:00)

asp.net 2.0,

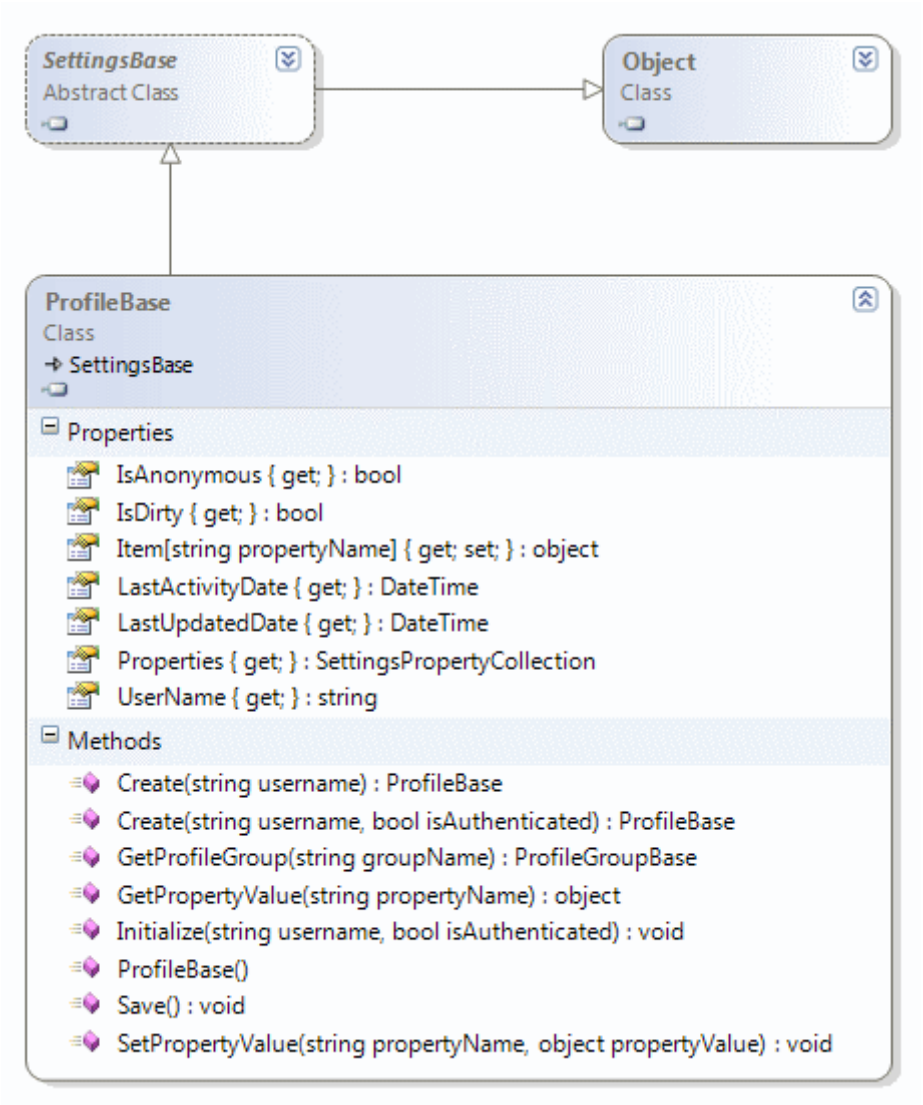
Uzun süre önce Asp.Net 2.0 ile geliştirilen web uygulamalarında **Profile API**' sinin nasıl kullanıldığını kısa bir makale üzerinden incelemeye çalışmıştık. Geçtiğimiz günlerde Asp.Net 2.0 ile ilgili bilgilerimi tazelerken profil yönetiminin daha etkin bir şekilde nasıl kullanılabileceğine dair pek çok örnek ile karşılaştım. İşte bu makalemizde temel olarak profil yönetiminin daha etkin hale getirilmeye çalışması için uğraşıyor olacağız. İnceleyeceğimiz temel konu başlıklarını aşağıdaki gibi sıralayabiliriz.

- **ProfileBase** tipinden türetmek(Inherit).
- Profil bilgilerini kod üzerinden yönetebilmek**(ProfileManager)**.
- **İsimsiz(Anonymous)** kullanıcılar için profil bilgilerini kullanabilmek.

Başlamadan önce profil kavramını kısaca tanımlamakta yarar olduğu kanısındayım. Bir web uygulamasına bağlanan kullanıcıların her biri için ortak tanımlanıp değerleri farklı olabilecek özellikler topluluğu profil bilgisini oluşturmaktadır. Bu anlamda özellikle, bir **doğrulama(authentication)** ve **yetkilendirme(authorization)** sistemine sahip olan web uygulamalarında her kullanıcı için değerleri farklı olabilecek özelliklerin tutulması ve kullanılması mümkün olabilmektedir. Bu tip bir sistemin özellikle **Asp.Net 1.1** ile geliştirilmesi ekstra kodlamayı gerektirirken Asp.Net 2.0 üzerinde yer alan Profile API sayesinde son derece kolaylaşmıştır. Gelelim Profile API yeteneklerini daha etkili bir şekilde nasıl ele alabileceğimize.

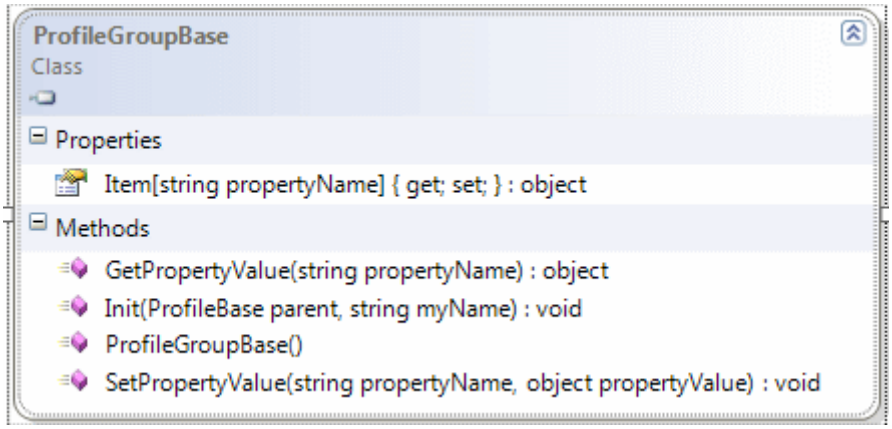
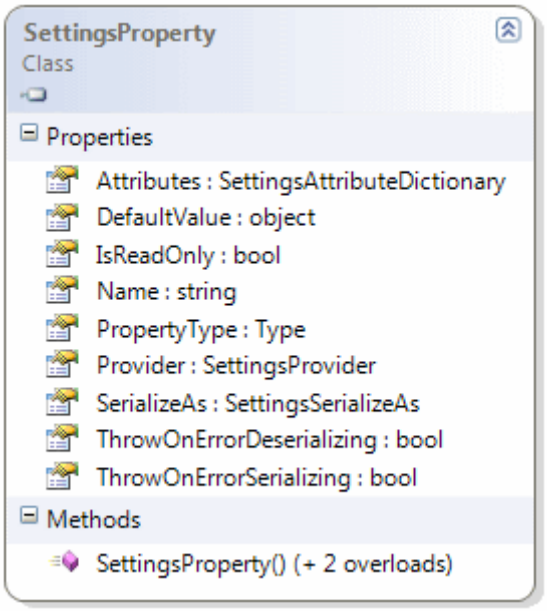
ProfileBase Tipinden Türetmek(Inherit);

Normal şartlarda bir web uygulaması içerisinde profil bilgilerini kullanabilmek için **web.config** dosyası içerisinde **profile** elementinin ele alınması gerekmektedir. Nitekim bir web uygulamasında kullanılan profil bilgilerinin, başka web uygulamasında(web uygulamalarında) ele alınmasının istendiği vakkalarda mevcuttur. Bu tip bir durumda çözüm olarak, **ProfileBase** tipinden türetme yapılaraktan birden fazla web uygulamasında ele alınabilecek bir profil sınıfı geliştirmek mümkündür. ProfileBase sınıfının temel üyeleri aşağıdaki sınıf diagramında(Class Diagram) görüldüğü gibidir.



PorfileBase tipi sınıf diagramından(class diagram) da görüldüğü gibi **SettingBase** isimli abstract sınıftan(class) türemeketedir. ProfileBase tipine ait üyelerden bazılarının görevleri aşağıdaki tabloda belirtildiği gibidir.

Metodlar(Methods)	**Açıklama**
Create	Bu metod ile bir kullanıcı için profil nesne örneği oluşturulur. özel profil konusu olduğunda ele alınmaktadır. Metod geriye **ProfileBase** tipinin taş parametre olarak kullanıcı adını alır. İkinci parametre **bool** bir değerdir v olmadığını belirtir. Metodun dönüş değerinin true olması kullanıcının doğ
Save	Profil bilgilerini kaydetmek amacıyla kullanılır. Herhangibir parametre al üzerinden çağrılabildiği için ilgili tipe ait özelliklerde yapılan değişiklikle yapılmaktadır. Bu işlem sırasında **IsDirty** özelliği true değerini alır. İşlen
GetPropertyValue	Parametre olarak verilen özelliğin değerini object tipinden döndürür.
SetPropertyValue	İki parametre alan bu metodun ilk parametresi değeri verilecek özellik ad

	yardımıyla profil içerisindeki bir özelliğe değer atanabilmesi sağlanabilm
GetProfileGroup	Profil içerisinde yer alan özellikler istenirse grup halinde ayrılabilirler. B elementi kullanılmaktadır. Böyle bir durumda gruplanan özelliklerin liste geriye **ProfileGroupBase** tipinden bir referans döndürmektedir. Bu refer mümkün olmaktadır. Aşağıdaki sınıf diagramı görüntüsünde ProfileGruo 
özellikler(Properties)	**Açıklama**
Item	Profil içerisinde tanımlanmış olan özellik adlarını parametre olarak alabil **edilmesi(get)**mümkündür. özellik adı string tipinden verilmekte olup ind
Properties	**Static** olarak tanımlanmış olan bu özellik sayesinde profil özelliklerinin b mümkündür. Bu koleksiyonun her bir elemanı **SettingsProperty** sınıfı tip edilecek olursa bu üyelerden yola çıkarak profilin özelliği hakkında detay 
UserName	Profilin sahibi olan kullanıcı adını verir. Eğer isimsiz bir kullanıcı girişi s
IsDirty	Profil özelliklerinden herhangibiri değiştiriliyorken **true** değerini döndür

IsAnonymous	Eğer kullanıcı **isimsiz(anonymous)** ise **true** değerini döndürür. Aksi dur

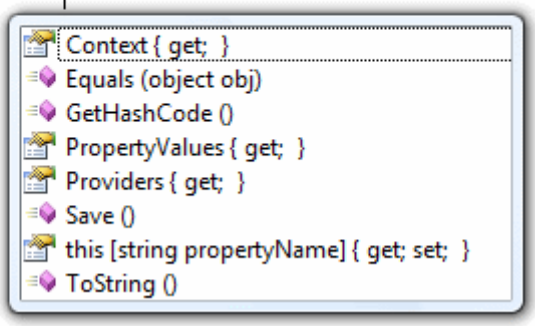
Şimdi örnek bir senaryo üzerinden hareket ederek konuyu biraz daha iyi kavramaya çalışalım. öncelikli olarak hedefimiz birden fazla web uygulamasının kullanabileceği bir **Profile** tipi geliştirmek olduğundan bir **sınıf kütüphanesi(class library)** geliştirerek işe başlanabilir. çok doğal olarak bu sınıf kütüphanesi içerisinde **ProfileBase** tipi kullanılacağından ve yeri geldiğinde güncel HTTP içeriğine**(HttpContext)** erişilmesi gerektiğinden **System.Web.dll** assembly' ının ilgili sınıf kütüphanesine referans edilmesi gerekmektedir.

özel olarak hazırlanacak sınıfın sağlaması gereken bazı özellikler vardır. İlk olarak bu sınıfın en azından XML serileştirilebilir**(XML Serializable)** olması gerekmektedir. çok doğal olarak bu sınıf içerisinde kullanılacak özelliklerin **veri tipleride(data types)** serileştirilebilir olmalıdır. *(Kendi tiplerimizden özellik türleri yazmadığımızda çoğunluklu ilkel tipleri(primitive types) kullanırız. Bu tiplerin çoğu zaten serileştirilebilir olduğundan sorun çıkma olasılığı azalmaktadır. Ancak kendi tiplerimizi ele aldığımızda serileştirilebilir olmalarına dikkat etmek gerekmektedir.)* İkinci olarak özel tip içerisinde, web uygulamalarındaki kullanıcılar için gerekli profil bilgisini oluşturacak **özellikler(property)** ayrı ayrı tanımlanmalıdır. Bu özelliklerin kullanımı sırasında base anahtar kelimesi ile üst sınıfa aktarma yapılmasına dikkat edilmelidir. üçünü olarak elbetteki özel sınıfın**ProfileBase** tipinden türetilmiş olması gerekmektedir. Bu türetmenin doğal sonucu olarak ProfileBase sınıfı içerisinde tanımlanmış bazı üyelerin **ezilebileceği(override)** ortadadır.

NOT : *Hatırlanacağı üzere **üst sınıfta(base class) sanal(virtual)** olarak tanımlanış olan üyelerin, **türeyen sınıflarda(derived class) ezilme(override)** zorunluluğu yoktur. Eğer bir zorunluluk getirilmesi isteniyorsa **abstract üyelerin** yer alabildiği **abstract sınıflar** veya **arayüzler(interface)** kullanılmalıdır.*

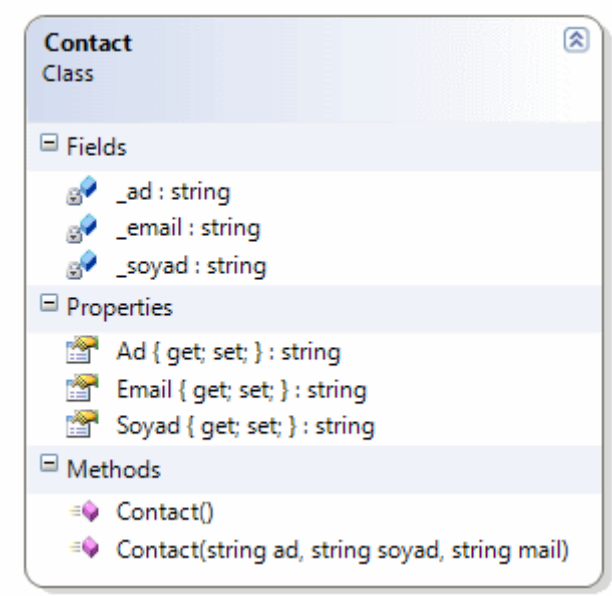
Aşağıdaki şekilde **türeyen sınıf(derived class)** içerisinde ezilebilecek üyeler gösterilmektedir. Doğal olarak herkes bir Object olduğundan, object sınıfından gelen bazı virtual üyelerde bu listede yer almaktadır.

```
public class MyProfile:ProfileBase
{
    override
}
```



örnek olarak MyProfile sınıfı aşağıdaki gibi tasarlanabilir. Sınıf içerisinde kontak bilgilerini saklamak amacıyla **Contact** isimli bir sınıf daha kullanılmaktadır. Bu sınıfa ait nesne örneklerini kullanan özellikler MyProfile sınıfı içerisinde ele alınarak, kullanıcı tanımlı tiplerin durumuda rahatlıkla incelenebilir.

Contact Sınıfı;

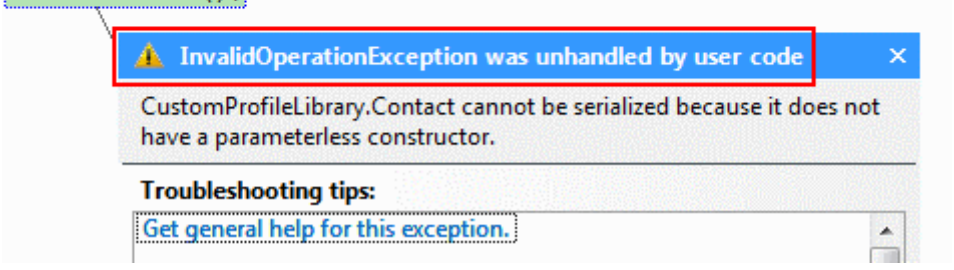



```
public class Contact
{
   private string _ad;
   private string _soyad;
   private string _email;

   public string Email
   {
      get { return _email; }
      set { _email = value; }
   }

   public string Soyad
   {
      get { return _soyad; }
      set { _soyad = value; }
   }

   public string Ad
   {
      get { return _ad; }
      set { _ad = value; }
```

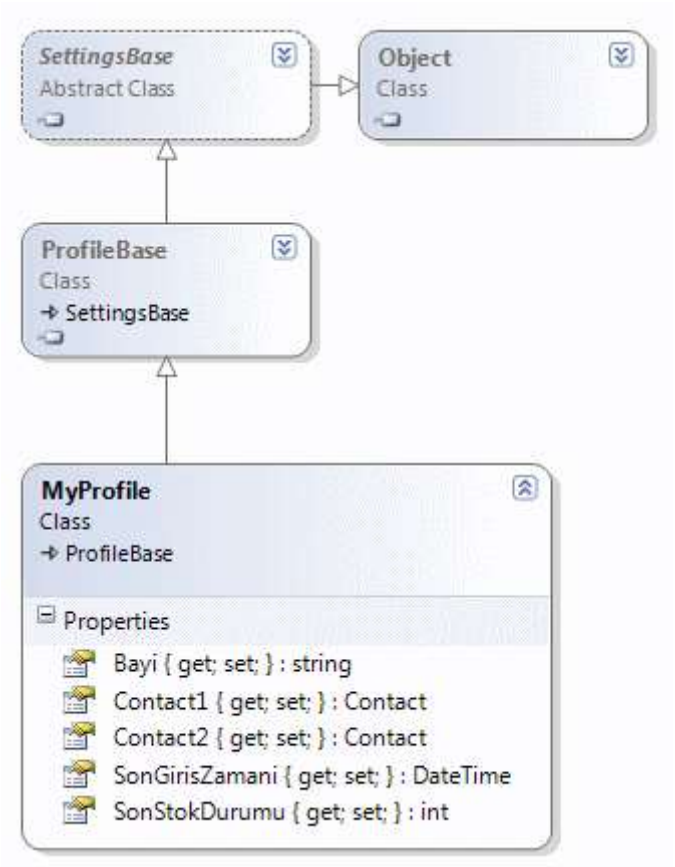
```
    }
    public Contact(string ad, string soyad, string mail)
    {
        Ad = ad;
        Soyad = soyad;
        Email = mail;
    }
    // XML Serileştirme kuralı olarak parametresiz bir yapıcı metod(constructor) olması gerekmektedir.
    public Contact()
    {
    }
}
```

Burada dikkat edilmesi gereken noktalardan biriside Contact sınıfının **varsayılan yapıcı metodunun(Default Constructor)** yazılmış olmasıdır. Profil bilgilerinde kendi tiplerimizi kullandığımız durumlarda varsayılan olarak XML serileştirme gerçekleştirilmektedir. Dolayısıyla yine varsayılan olarak **AspNetDb** veritabanındaki **aspnet_Profile** tablosuna yazılacak olan özellik değerlerinin XML formatında serileştirilebilir olması gerekmektedir. Bu sebepten ilgili tipin varsayılan yapıcı metoda sahip olması gerekir ki bu XML serileştirmenin kurallarından birisidir. Eğer varsayılan yapıcı metodu yazmassak çalışma zamanında profil bilgisini kaydederken aşağıdaki ekran görüntüsünde yer alan **istisnayı(exception)** alırız.

```
Contact kontak1 = new Contact(txtKontak1Ad.Text, txtKontak1
Profile.Contact1 = kontak1;
Contact kontak2 = new Contact(txtKontak2Ad.Text, txtKontak2
Profile.Contact2 = kontak2;
Profile.SonGirisZamani = DateTime.Now;
int stok = 1;
Int32.TryParse(txtStokDurumu.Text, out stok);
Profile.SonStokDurumu = stok;
Profile.Save();
```



MyProfile Sınıfı;




```
public class MyProfile:ProfileBase
{
   public int SonStokDurumu
   {
      get { return Convert.ToInt32(base["SonStokDurumu"]); }
      set { base["SonStokDurumu"] = value; }
   }
   public DateTime SonGirisZamani
   {
      get { return Convert.ToDateTime(base["SonGirisZamani"]); }
      set { base["SonGirisZamani"] = value; }
   }
   public Contact Contact2
   {
      get { return (Contact)base["Contact2"]; }
      set { base["Contact2"] = value; }
   }
   public Contact Contact1
   {
      get { return (Contact)base["Contact1"]; }
      set { base["Contact1"] = value; }
   }
   public string Bayi
```

```
    {
        get { return base["Bayi"].ToString(); }
        set { base["Bayi"] = value; }
    }
}
```

Dikkat edilecek olursa sınıf içerisinde tasarlanan özelliklere ait **get** ve **set** bloklarında **base** anahtar kelimesi ile **ProfileBase** sınıfına çıkılmakta ve indeksleyici operatöründen yararlanılarak ilgili alanlara erişilmesi sağlanmaktadır. Elbetteki base ile ulaşılan referansın özellikleri object veri türü ile çalıştığından set blokları içerisinde uygun tür dönüşümlerinin**bilinçli(explicit)** olarak yapılması şarttır. MyProfile sınıfı içerisinde tanımlanacak **özel alanlara(private fields)** değer atanması durumunda veritabanına herhangibir şekilde bilgi yazılmayacaktır. Böyle bir işlem için **Save** metodunun bu sınıf içerisinde ezilemesi ve kodlanması gerekmektedir.

Şimdi MyProfile sınıfını test etmek amacıyla basit bir web uygulaması tasarlayalım. Bu web uygulamasında standart olarak **AspNetDb** veritabanı kullanılabilir. **Form tabanlı doğrulama(Form Based Authentication)** sisteminin yer aldığı web uygulamasında standart olarak Login.aspx sayfası kullanıcı girişi için kullanılırken, Default.aspx sayfası içerisinde örnek test kodları yer almaktadır. Web uygulamasına ait konfigurasyon dosyasının**(Web.config)** içeriği aşağıdaki gibi olmalıdır.

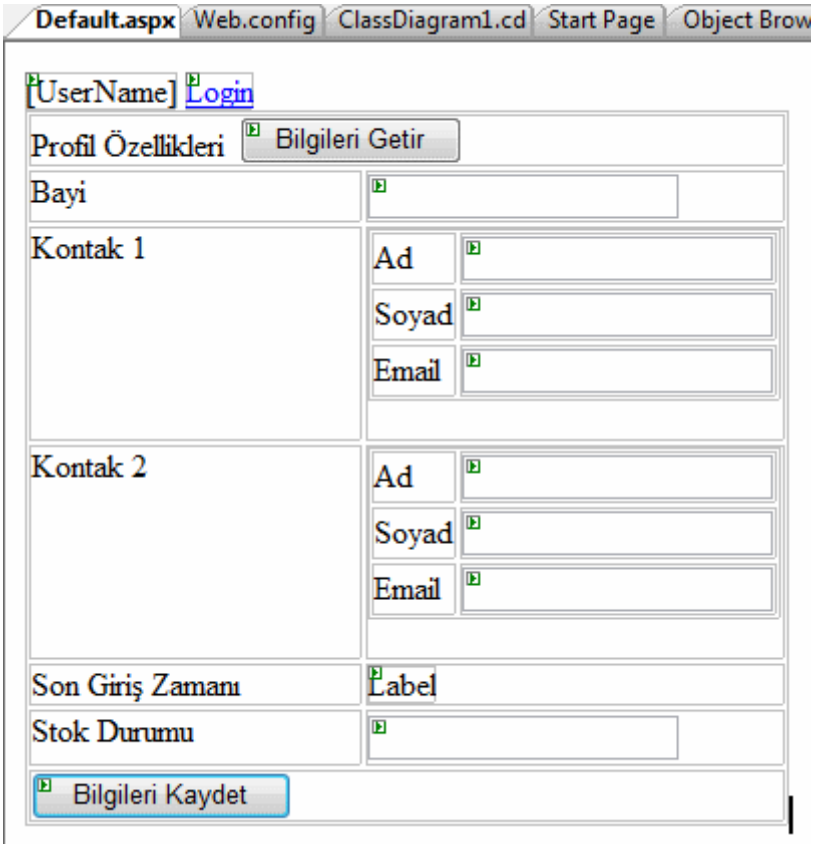
web.config;

```
<?xml version="1.0"?>
<configuration>
    <appSettings/>
    <connectionStrings/>
    <system.web>
        <compilation debug="true" />
        <authentication mode="Forms" />
        <authorization>
            <deny users="?"/>
        </authorization>
        <profile enabled="true" inherits="CustomProfileLibrary.MyProfile"/>
    </system.web>
</configuration>
```

profile elementi içerisinde **inherits** niteliğine(attribute) atanan değer ile **Profile** tipinin ne olduğu söylenir. Burada **İsimAlanıAdı.TipAdı(NamespaceName.TypeName)** notasyonu kullanılarak profile bilgilerinin yönetiminin MyProfile sınıfı tarafından yapılacağı belirtilmektedir. Form tabanlı doğrulama(Form Based Authentication) kullanıldığından **authentication** elementinin**mode** niteliğine **Forms** değeri atanmıştır. **İsimsiz kullanıcıların(anonymous users)** siteye giriş yapması istenmediğinden **authorization** elementi içerisinden söz konusu kullanıcılar **?**karakteri

ile **deny** edilmiştir. Gelelim uygulamanın örnek web sayfasına. Sayfanın tasarımı aşağıda görüldüğü gibidir.

Default.aspx sayfası;




```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>Untitled Page</title>
   </head>
   <body>
      <form id="form1" runat="server">
         <div>
            <asp:LoginName ID="LoginName1" runat="server" /><asp:LoginStatus
ID="LoginStatus1" runat="server" />
            <br />
            <table>
               <tr>
                  <td colspan="2" valign="top">Profil özellikleri <asp:Button
```

```
ID="btnGetir" runat="server" OnClick="btnGetir_Click" Text="Bilgileri Getir" /></td>
            </tr>
            <tr>
               <td style="width: 164px" valign="top">Bayi</td>
               <td style="width: 100px"><asp:TextBox ID="txtBayi"
runat="server"></asp:TextBox></td>
            </tr>
            <tr>
               <td style="width: 164px" valign="top">Kontak 1</td>
               <td style="width: 100px">
               <table>
                  <tr>
                     <td style="width: 100px">Ad</td>
                     <td style="width: 100px"><asp:TextBox ID="txtKontak1Ad"
runat="server"></asp:TextBox></td>
                  </tr>
                  <tr>
                     <td style="width: 100px">Soyad</td>
                     <td style="width: 100px"><asp:TextBox ID="txtKontak1Soyad"
runat="server"></asp:TextBox></td>
                  </tr>
                  <tr>
                     <td style="width: 100px">Email</td>
                     <td style="width: 100px"><asp:TextBox ID="txtKontak1Email"
runat="server"></asp:TextBox></td>
                  </tr>
               </table>
               </td>
            </tr>
            <tr>
               <td style="width: 164px" valign="top">Kontak 2</td>
               <td style="width: 100px">
               <table>
                  <tr>
                    <td style="width: 100px">Ad</td>
                     <td style="width: 100px"><asp:TextBox ID="txtKontak2Ad"
runat="server"></asp:TextBox></td>
                  </tr>
                  <tr>
                     <td style="width: 100px">Soyad</td>
                     <td style="width: 100px"><asp:TextBox ID="txtKontak2Soyad"
runat="server"></asp:TextBox></td>
                  </tr>
                  <tr>
                     <td style="width: 100px">Email</td>
```

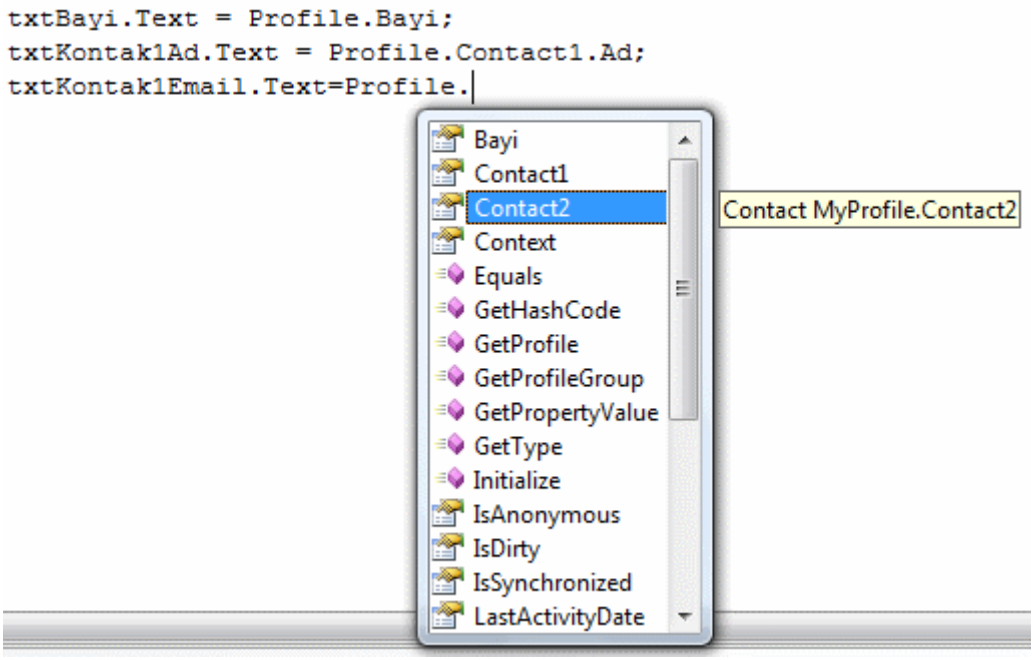
```
                        <td style="width: 100px"><asp:TextBox ID="txtKontak2Email"
runat="server"></asp:TextBox></td>
                    </tr>
                </table>
                </td>
            </tr>
            <tr>
                <td style="width: 164px; height: 10px" valign="top">Son Giriş
Zamanı</td>
                <td style="width: 100px; height: 10px"><asp:Label
ID="lblSonGirisZamani" runat="server" Text="Label"></asp:Label></td>
            </tr>
            <tr>
                <td style="width: 164px; height: 26px" valign="top">Stok Durumu</td>
                <td style="width: 100px; height: 26px"><asp:TextBox
ID="txtStokDurumu" runat="server"></asp:TextBox></td>
            </tr>
            <tr>
                <td colspan="2" valign="top"><asp:Button ID="btnKaydet" runat="server"
OnClick="btnKaydet_Click" Text="Bilgileri Kaydet" /></td>
            </tr>
        </table>
        <br />
        <br />
    </div>
  </form>
 </body>
</html>
```

Default.aspx.cs;

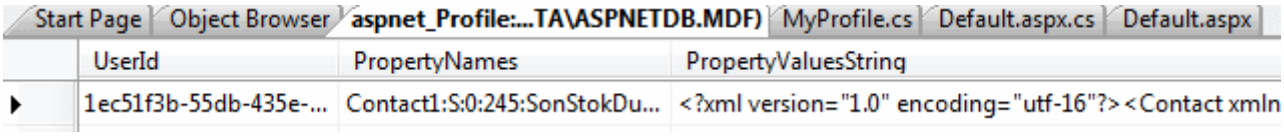
```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using CustomProfileLibrary;

public partial class _Default : System.Web.UI.Page
{
   protected void btnGetir_Click(object sender, EventArgs e)
```

```
    {
        Getir();
    }

    protected void btnKaydet_Click(object sender, EventArgs e)
    {
        Kaydet();
    }

    private void Getir()
    {
        try
        {
            txtBayi.Text = Profile.Bayi;
            txtKontak1Ad.Text = Profile.Contact1.Ad;
            txtKontak1Email.Text = Profile.Contact1.Email;
            txtKontak1Soyad.Text = Profile.Contact1.Soyad;
            txtKontak2Ad.Text = Profile.Contact2.Ad;
            txtKontak2Email.Text = Profile.Contact2.Email;
            txtKontak2Soyad.Text = Profile.Contact2.Soyad;
            lblSonGirisZamani.Text = Profile.SonGirisZamani.ToString();
            txtStokDurumu.Text = Profile.SonStokDurumu.ToString();
        }
        catch (Exception excp)
        {
            Response.Write(excp.Message);
        }
    }

    private void Kaydet()
    {
        Profile.Bayi = txtBayi.Text;
        Contact kontak1 = new Contact(txtKontak1Ad.Text, txtKontak1Soyad.Text,
txtKontak1Email.Text);
        Profile.Contact1 = kontak1;
        Contact kontak2 = new Contact(txtKontak2Ad.Text, txtKontak2Soyad.Text,
txtKontak2Email.Text);
        Profile.Contact2 = kontak2;
        Profile.SonGirisZamani = DateTime.Now;
        int stok = 1;
        Int32.TryParse(txtStokDurumu.Text, out stok);
        Profile.SonStokDurumu = stok;
        Profile.Save();
    }
}
```

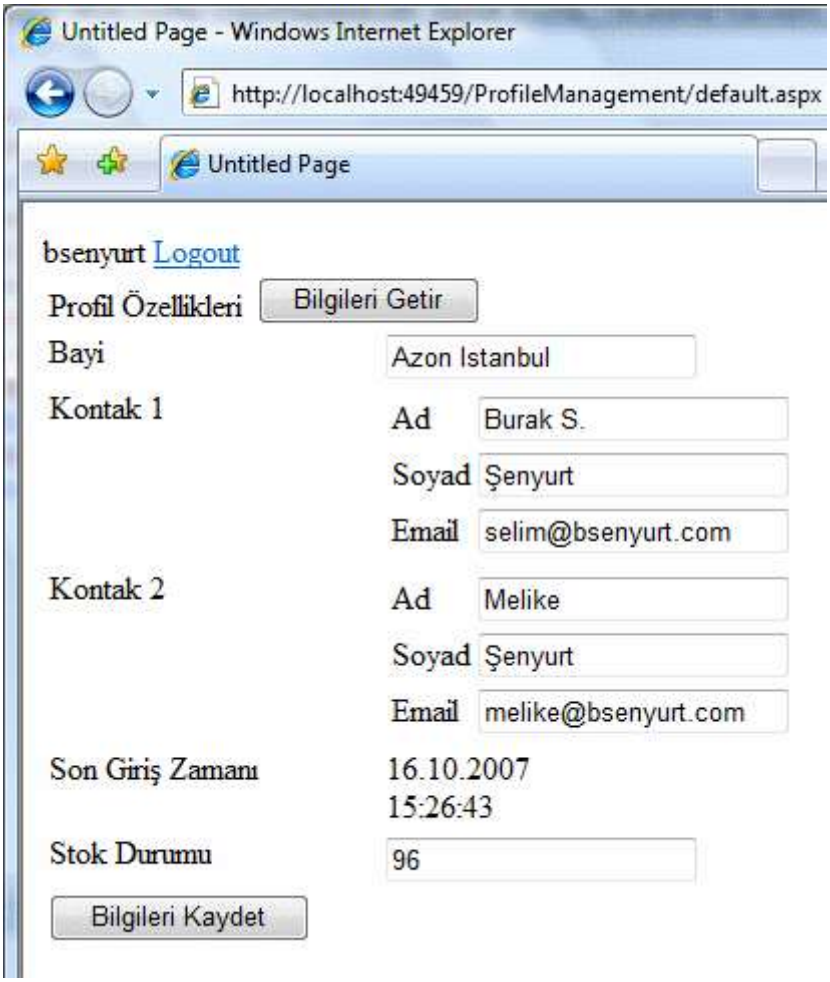
Sayfa basit olarak profil bilgilerinin getirilmesi veya kaydedilmesi için gereken kodları içermektedir. Dikkat edilirse Profile tipi kullanılmaktadır.
Nitekim **web.config** dosyasındaki bildirim nedeni ile **Profile**özelliği üzerinden MyProfile içerisinde tanımlanmış olan özelliklere erişilebilmektedir. Aşağıdaki şekilde bu durum daha net bir şekilde görülebilir.



çok doğal olarak daha önceden bir profil bilgisi oluşturulmaması durumuna karşılık bilgiler getirilirken **Null Reference Exception** alınma olasılığı vardır. Bu nedenle profil bilgileri ortama çekilirken bir **try...catch** bloğu kullanılması yararlı olabilir. Sayfa üzerinde test bilgilerinin kaydedilmesi sonrasında **aspnet_profile** tablosunun içeriğine bakılırsa profil bilgisinin başarılı bir şekilde kaydedildiği görülebilir. Dikkat edileceği üzere bilgiler XML serileştirmeye uygun olacak şekilde **XML** formatında **PropertyValueString** alanına yazılmıştır.

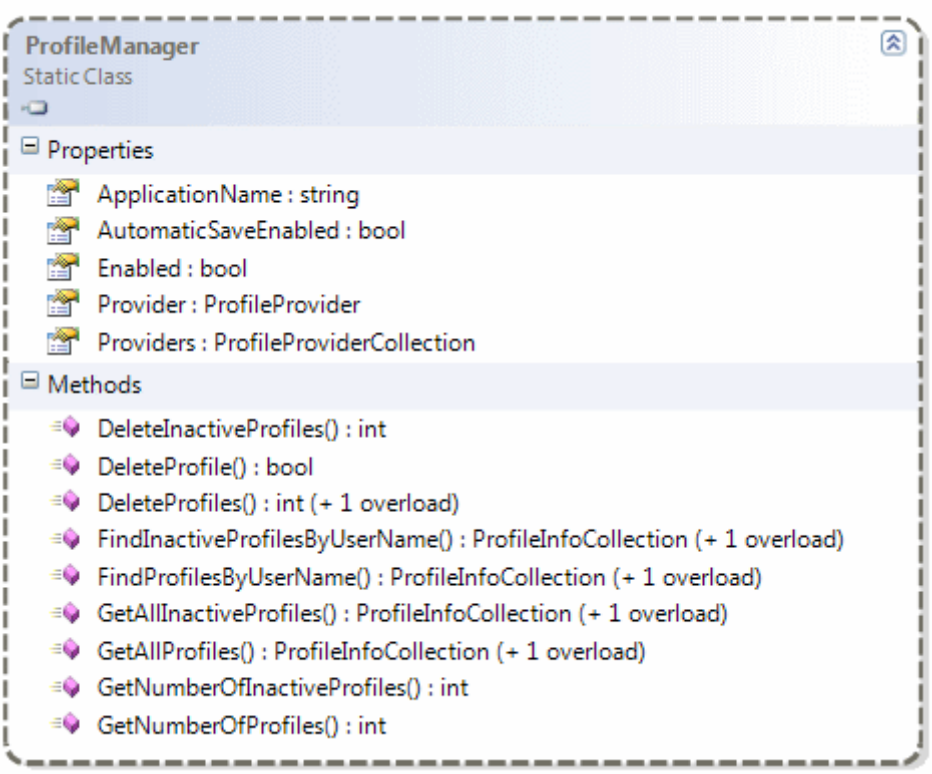


çalışma zamanında(run-time) Bilgileri Getir başlıklı düğmeye basıldığında ise söz konusu verilerin ilgili kontrollere doldurulduğu görülecektir. Aşağıda bu duruma ait örnek bir ekran çıktısı yer almaktadır.



Profil Bilgilerini Kod üzerinden Yönetebilmek;

Bazı durumlarda çalışma zamanında(run-time) uygulamada kullanılan profil bilgilerinin yönetilmesi istenebilir. Söz gelimi profilde kayıtlı bilgilerin gösterilmesi, belirli bir tarihten öncekilerin kaldırılması, bir kullanıcının profil bilgisinin silinmesi, aktif olmayan profillerin elde edilmesi vb... işlemler yapılabilir. Bu aslında basit olarak veritabanına ulaşmak ve ilgili tablonun alanlarına bakmaktan başka bir şey değildir. Ne varki **Asp.Net Profile API** içerisinde söz konusu yönetsel işlemlerin daha kolay yapılmasını sağlayan **ProfileManager** sınıfı mevcuttur. Bu sınıfın diagram görüntüsü aşağıdaki gibidir.



Dikkat edileceği üzere ProfileManager, **static** bir sınıftır(static class).

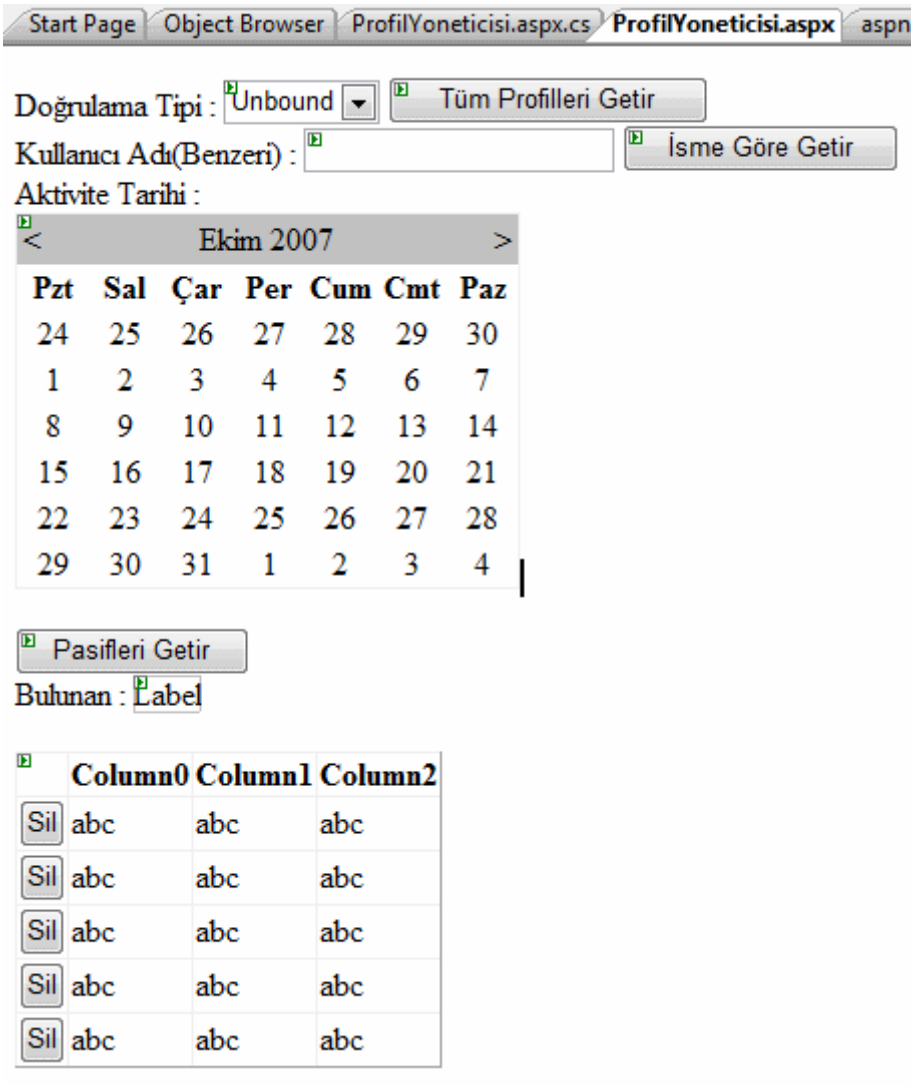
NOT : *Static sınıf kavramı C# 2.0 ile birlikte gelmiştir. Static sınıflar sadece static üyeler**(static members)** içerebilir, **türetme(Inheritance)** amacıyla kullanılamaz veya örneklenemezler. Normal sınıflara göre daha hızlı ve performanslı çalıştıkları ortadadır. Bununla birlikte C# 3.0 ile birlikte gelen **extension methods** kavramında önemli bir yerede sahiptir.*

Sınıfın önemli metodları ve yaptığı işler ise aşağıdaki tabloda görüldüğü gibidir.

Metodlar(Methods)	**Açıklama**
GetAllProfiles	İki versiyonu olan bu metod sayesinde bir uygulamadaki pr edilebilir. Her iki versiyonda**ProfileAuthenticationOption** değerler **All, Anonymous, Authneticated**' dır. Bir başka de kullanıcıların profile bilgilerinin elde edilmesi sağlanabilir. sağlanabilmektedir.
GetNumberOfProfiles	Veritabanında kayıtlı olan profil sayısının elde edilmesini s alır. Buna göre sadece **isimsiz kullanıcıların(anonymous u users)** kayıtlı olan profil sayıları elde edilebilir.
GetAllInactiveProfiles	Profil sahibi kullanıcıların **LastActivityDate** özelliklerinin edilmesini sağlayan metoddur. İki versiyonu vardır ve her i ikincisinde ise tarih bilgisi belirlenir.
GetNumberOfInactiveProfiles	Profil sahibi kullanıcıların **LastActivityDate** özelliklerinin

	sayısının integer olarak elde edilmesini sağlayan metoddur.
FindProfilesByUserName	Belirlenen kullanıcı adıyla eşleşen profil bilgilerinin **Profile** Her ikiside ilk iki parametresinde sırasıyla **ProfileAuthenti** sağlanabilmektedir. Burada kullanıcı adı girilirken **%** karak
FindInactiveProfilesByUserName	Profil sahibi kullanıcıların **LastActivityDate** özelliklerinin kullanıcı adı ile eşleşenlerinin bulunmasını sağlayan metodd sorgulama yapılabilmektedir. örneğin kullanıcı adı içerisind listelenebilir.
DeleteProfile	Parametre olarak verilen kullanıcıya ait profile bilgisinin si
DeleteProfiles	Bu metodun iki farklı versiyonu vardır. Bunlardan birisi **Pr** Dolayısıyla profile bilgilerinin silinmesi istenen kullanıcı tip
DeleteInactiveProfiles	Profil sahibi kullanıcıların **LastActivityDate** özelliklerinin ProfileAuthenticationOption enum sabiti tipinden değerde a

Şimdi bu sınıfın kullanıldığı örnek bir web sayfasını projeye dahil edelim.
Sayfanın **tasarım zamanındaki(design-time)** görüntüsü ve kodları aşağıdaki gibidir.



ProfilYoneticisi.aspx;

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ProfilYoneticisi.aspx.cs"
Inherits="ProfilYoneticisi" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>Profil Yönetim Ekranı</title>
   </head>
   <body>
      <form id="form1" runat="server">
         <div>
            Doğrulama Tipi :<asp:DropDownList ID="ddlDogrulamaTipi"
runat="server"></asp:DropDownList>
            <asp:Button ID="btnTumProfiller" runat="server"
```

```
OnClick="btnTumProfiller_Click" Text="Tüm Profilleri Getir" /><br />
            Kullanıcı Adı(Benzeri) :
            <asp:TextBox ID="txtKullaniciAdi" runat="server"></asp:TextBox>
            <asp:Button ID="btnIsmeGoreGetir" runat="server"
OnClick="btnIsmeGoreGetir_Click" Text="İsme Göre Getir" /><br />
            Aktivite Tarihi :
            <br />
            <asp:Calendar ID="dtTarih" runat="server"></asp:Calendar>
            <br />
            <asp:Button ID="btnPasifleriGetir" runat="server"
OnClick="btnPasifleriGetir_Click" Text="Pasifleri Getir" /><br />
            Bulunan :
            <asp:Label ID="lblKullaniciSayisi" runat="server"
Text="Label"></asp:Label><br />
            <br />
            <asp:GridView ID="grdTumProfiller" runat="server"
OnRowCommand="grdTumProfiller_RowCommand"
OnRowDeleting="grdTumProfiller_RowDeleting">
                <Columns>
                    <asp:CommandField ButtonType="Button" DeleteText="Sil"
ShowCancelButton="False" ShowDeleteButton="True" />
                </Columns>
            </asp:GridView>
        </div>
    </form>
  </body>
</html>
```

ProfilYoneticisi.aspx.cs;

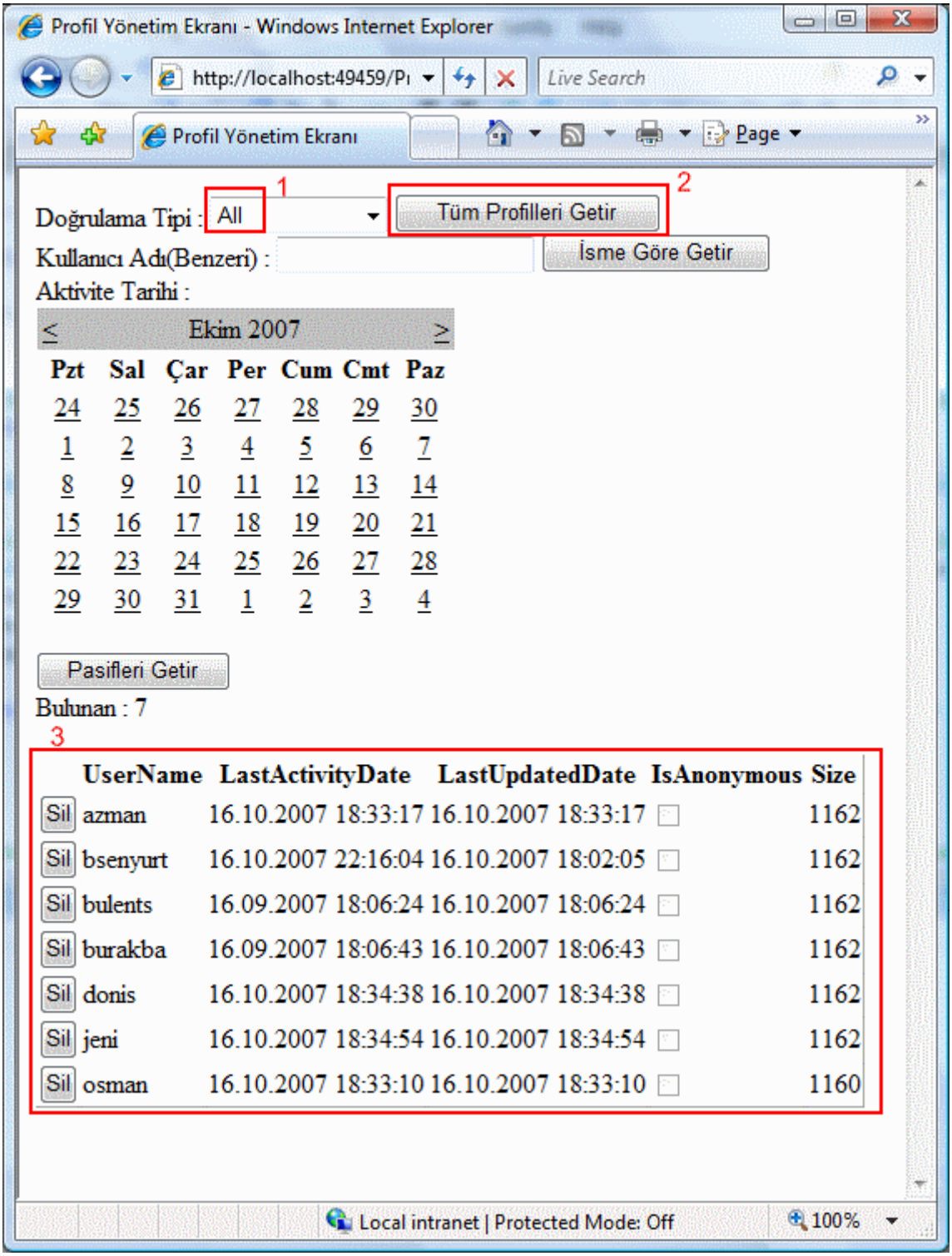
```
using System;
using System.Data;
using System.Configuration;
using System.Collections;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Web.Profile;

public partial class ProfilYoneticisi : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
```

```
    {
        if (!Page.IsPostBack)
        {
            // ProfileAuthenticationOption enum sabitinin içeriğini Enum sınıfın GetNames
metodu ile alarak DropDownList kontrolüne dolduruyoruz.
            ddlDogrulamaTipi.DataSource
= Enum.GetNames(typeof(ProfileAuthenticationOption));
            ddlDogrulamaTipi.DataBind();
        }
    }

    // Bu metod var olan tüm profil bilgilerini getirmek üzere tasarlanmıştır
    private void TumProfilleriGetir()
    {
        // DropDownList kontrolünden seçilen doğrulama kriterinin enum sabitindeki karşılığı
seçiliyor.
        ProfileAuthenticationOption
dogrulamaKriteri=(ProfileAuthenticationOption)Enum.Parse(typeof(ProfileAuthenti
cationOption), ddlDogrulamaTipi.SelectedValue);
        // GetAllProfiles metodu ile doğrulama kriterine uygun olan profil bilgileri
ProfileInfoCollection tipinden bir koleksiyona aktarılır.
        ProfileInfoCollection tumProfiller =
ProfileManager.GetAllProfiles(dogrulamaKriteri);
        grdTumProfiller.DataSource = tumProfiller;
        grdTumProfiller.DataBind();
        // Söz konusu doğrulama kriterine uyan profillerin sayısı elde edilir.
        lblKullaniciSayisi.Text =
ProfileManager.GetNumberOfProfiles(dogrulamaKriteri).ToString();
    }

    // GridView kontrolünde Delete işlemi gerçekleştirildiğinde çalışan olay metodudur.
    protected void grdTumProfiller_RowCommand(object sender,
GridViewCommandEventArgs e)
    {
        if (e.CommandName == "Delete")
        {
            int rowIndex=Convert.ToInt32(e.CommandArgument);
            string userName=grdTumProfiller.Rows[rowIndex].Cells[1].Text;
            // DeleteProfile metodu ile seçilen satırdaki UserName bilgisine ait Profil silinir.
            ProfileManager.DeleteProfile(userName);
            TumProfilleriGetir();
        }
    }
    protected void btnTumProfiller_Click(object sender, EventArgs e)
    {
```

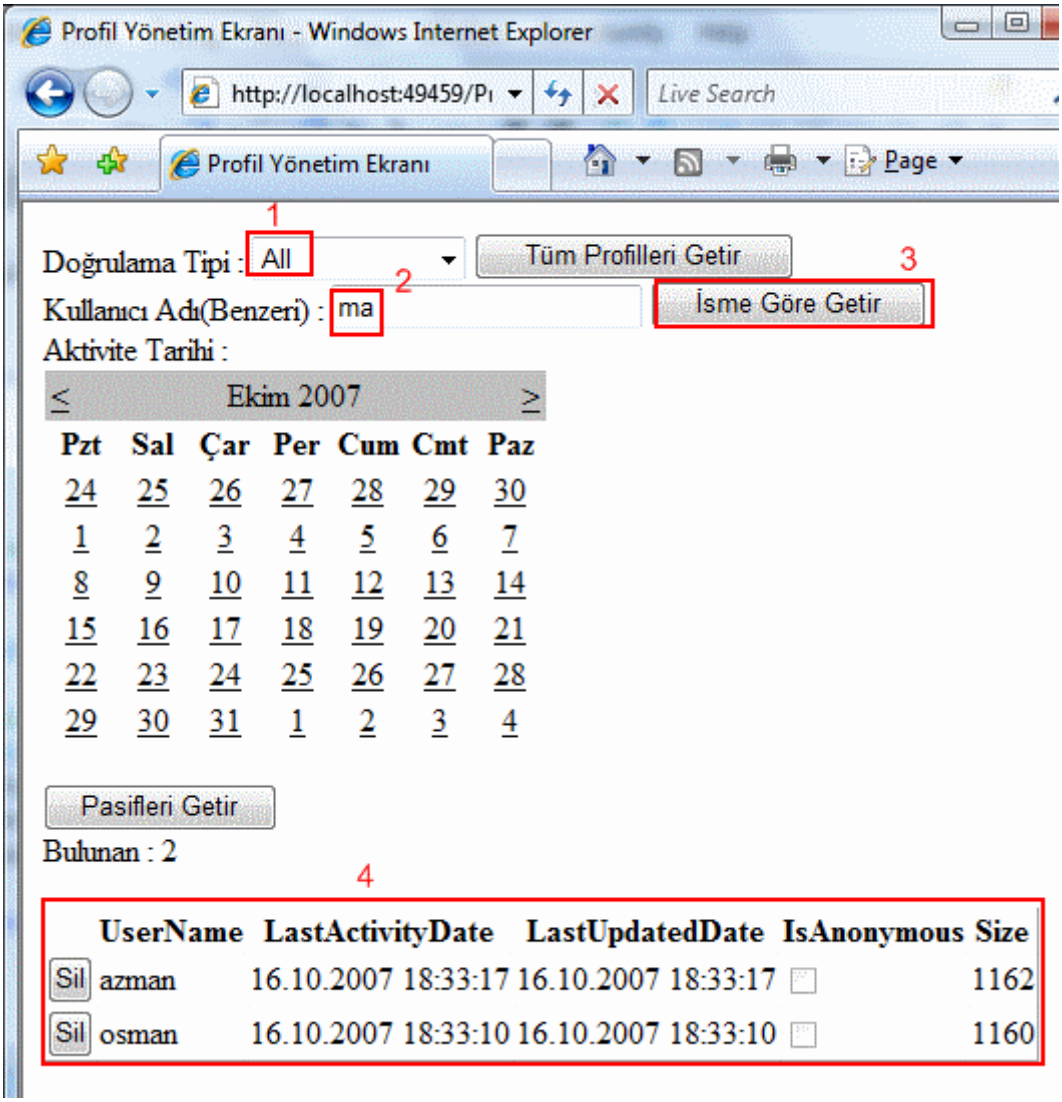
```
        TumProfilleriGetir();
    }

    // TextBox içerisine girilen kullanıcı adına benzer olanların Profil bilgilerini getirir.
    protected void btnIsmeGoreGetir_Click(object sender, EventArgs e)
    {
        ProfileAuthenticationOption dogrulamaKriteri =
(ProfileAuthenticationOption)Enum.Parse(typeof(ProfileAuthenticationOption),
ddlDogrulamaTipi.SelectedValue);
        // % kullanılması sayesinde içerisinde TextBox' taki bilgi geçen kullanıcı adlarını elde
ederiz.
        ProfileInfoCollection profiller =
ProfileManager.FindProfilesByUserName(dogrulamaKriteri, "%" +
txtKullaniciAdi.Text + "%");
        grdTumProfiller.DataSource = profiller;
        grdTumProfiller.DataBind();
        lblKullaniciSayisi.Text = profiller.Count.ToString();
    }
    protected void grdTumProfiller_RowDeleting(object sender, GridViewDeleteEventArgs
e)
    {

    }

    /* aspnet_Users tablosunda LastActivityDate değeri, seçilen tarihten daha önce olanların
elde edilmesini sağlar. Bunun için GetAllInactiveProfiles metodunun ikinci parametresi
kullanılır. */
    protected void btnPasifleriGetir_Click(object sender, EventArgs e)
    {
        ProfileAuthenticationOption dogrulamaKriteri =
(ProfileAuthenticationOption)Enum.Parse(typeof(ProfileAuthenticationOption),
ddlDogrulamaTipi.SelectedValue);
        grdTumProfiller.DataSource=ProfileManager.GetAllInactiveProfiles(dogrulamaK
riteri, dtTarih.SelectedDate);
        grdTumProfiller.DataBind();
        lblKullaniciSayisi.Text=ProfileManager.GetNumberOfInactiveProfiles(dogrulam
aKriteri, dtTarih.SelectedDate).ToString();
    }
}
```

Söz konusu örnekte profil yönetimi adına basit işlemler yapılmaktadır. örnek olması açısında var olan tüm profillerin elde edilmesi, seçilen profillerden herhangibirinin silinmesi, içerisinde belirtilen ada benzer olan profillerin çekilmesi veya belirli bir tarihten öncesine kadar aktivitesi olmayan profillerin getirilmesi gibi işlemler yapılmaktadır.

örneğin tüm profillerin listelenmesi istendiğinde aşağıdakine benzer bir ekran görüntüsü ile karşılaşılır.



Kullanıcı adı benzer olanların listelenmesine örnek olarak aşağıdaki ekran görüntüsünde olduğu gibi içerisinde ma kelimesi geçenler listelenebilir.

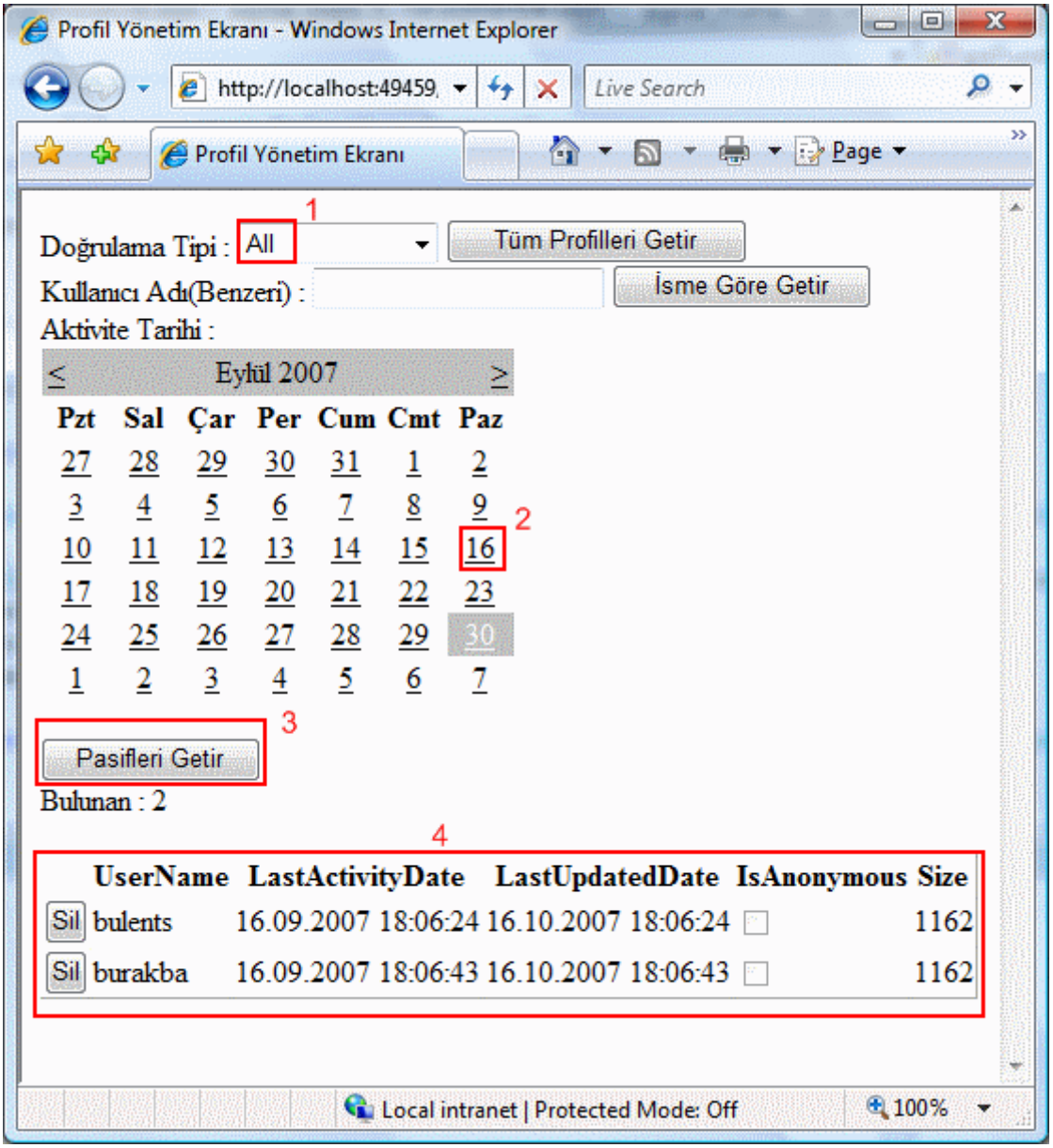


Belirli bir süreden öncesine kadar pasif olan kullanıcıların profil listesini elde etmek istediğimizde ise **aspnet_User** tablosunda yer alan **LastActivityDate** değeri baz alınır. Bir başka deyişle aktivite durumuna göre hareket eden **ProfileManager** sınıfı metodları tarih parametresinin değerlerini, söz konusu tablodaki alan ile kıyaslayarak çalışmaktadır. Bu tabloya ait örnek bir ekran görüntüsü aşağıda görüldüğü gibidir.

.TA\ASPNETDB.MDF) | aspnet_Profile:...TA\ASPNETDB.MDF) | Start Page | Object Browser

Id	UserId	UserName	LoweredUserN...	MobileAlias	IsAno...	LastActivityDate
15-...	a5b7d95e-1fdb...	azman	azman	*NULL*	False	16.10.2007 15:33:17
15-...	1ec51f3b-55db-...	bsenyurt	bsenyurt	*NULL*	False	16.10.2007 15:10:06
15-...	04d4a090-8bcc...	bulents	bulents	*NULL*	False	16.09.2007 15:06:24
15-...	6e3131ae-7547-...	burakba	burakba	*NULL*	False	16.09.2007 15:06:43
15-...	094d98cb-7ca0...	donis	donis	*NULL*	False	16.10.2007 15:34:38
15-...	76c5ee9a-aac6-...	jeni	jeni	*NULL*	False	16.10.2007 15:34:54
15-...	9e41fd05-e5d5-...	maykıl	maykıl	*NULL*	False	16.10.2007 14:54:14
15-...	e6aa5ec8-3141-...	osman	osman	*NULL*	False	16.10.2007 15:33:10
	NULL	*NULL*	*NULL*	*NULL*	*NULL*	*NULL*

çalışma zamanında örneğin 16.09.2007 tarihi ve öncesindeki profil bilgileri elde edilmek istendiğinde aşağıdakine benzer bir ekran görüntüsü ile karşılaşırız.



İsimsiz(Anonymous) kullanıcılar için profil bilgilerini kullanabilmek;

Bazı durumlarda siteyi ziyaret eden **isimsiz kullanıcılar(anonymous users)** içinde profil bilgilerinin tutulması ve saklanması istenebilir. **Profile API**, isimsiz kullanıcılar için oldukça güçlü bir hizmet sağlamaktadır. Sistemin çalışması aslında son derece basittir. Gerekli konfigurasyon ayarlarının yapılmasının ardından siteye bağlanan isimsiz kullanıcılar için birer **GUID**üretilmektedir. Bu **GUID(Global Unique IDentifier)** değerleri bir **çerez(cookie)** yardımıyla istemcinin bilgisayarında saklanırlar. Böylece isimsiz kullanıcıların sunucu tarafında tespit edilebilmesi kolay bir şekilde sağlanmaktadır. Dikkat edilmesi gereken durumlardan birisi istemcilerin **oturum(session)** açışları arasında farklı tarayıcılar kullanabilecek olmasıdır. Bu durumu ilerleyen kısımlarda analiz etmeye çalışacağız. İsimsiz kullanıcılara destek sağlayabilmek için web.config dosyası içerisinde **anonymousIdentification** elementinin ilgili özelliklerinin değerlerinin atanması gerekmektedir. Bir önceki örnek ile karışmaması açısından bu kez yeni bir web uygulaması ile devam edelim. Bu seferki örnekte profil özelliklerini**web.config** dosyası içerisinde

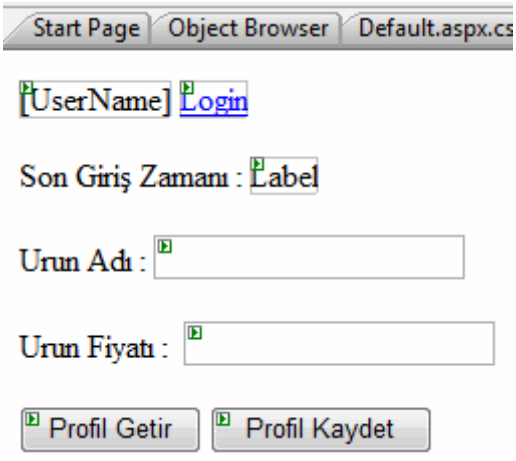
tanımlıyor olacağız. Bu amaçla web.config dosyasının içeriğini aşağıdaki gibi tasarladığımızı varsayalım.

Web.config;

```
<?xml version="1.0"?>
<configuration>
   <appSettings/>
   <connectionStrings/>
   <system.web>
      <compilation debug="true"/>
      <authentication mode="Forms"/>
      <authorization>
         <allow users="*"/>
      </authorization>
      <anonymousIdentification enabled="true" cookieSlidingExpiration="true"
cookieTimeout="3"/>
      <profile enabled="true">
         <properties>
            <add name="SonGirisZamani"
type="System.DateTime" allowAnonymous="true"/>
            <group name="Urun">
               <add name="UrunAdi" type="System.String" allowAnonymous="true"/>
               <add name="UrunFiyati" type="System.Double" allowAnonymous="true"/>
            </group>
         </properties>
      </profile>
   </system.web>
</configuration>
```

anonymousIdentification elementi içerisinde tanımlanabilecek pek çok nitelik(attribute) yer almaktadır. Bu niteliklerden bazıları yukarıdaki örnekte kullanılmıştır. **cookieSlidingExpiration**niteliğine **true** değeri atandığı için istemciler, timeout süresi içerisinde talepte bulundukça çerezlerin(cookies) istemci bilgisayarda kalma süresi uzamaya devam edecektir. *(Bu tipik olarak Caching mimarisinde kullanılan Sliding Expiration çalışma sistemi ile aynıdır.)* Bununla birlikte **cookieTimeout** niteliğine atanan 3 değeri ile çerezin istemci bilgisayarda 3 dakika süreyle saklanacağı belirtilmektedir. *(Uygulamanın **doğrulanmış kullanıcılar(authenticated users)** ile birlikte **isimsiz kullanıcılarada(anonymous users)** hizmet vermesi için **authorization** elementi içerisinde bilinçli olaraktan **tüm kullanıcılar(*) allow** edilmiştir.)*

İlk olarak isimsiz kullanıcılara ait profil bilgilerinin saklandığını ispat etmeye çalışalım. Bu amaçla default.aspx sayfasının içeriğini ve kodlarını aşağıdaki gibi tasarladığımızı düşünelim.



Default.aspx;

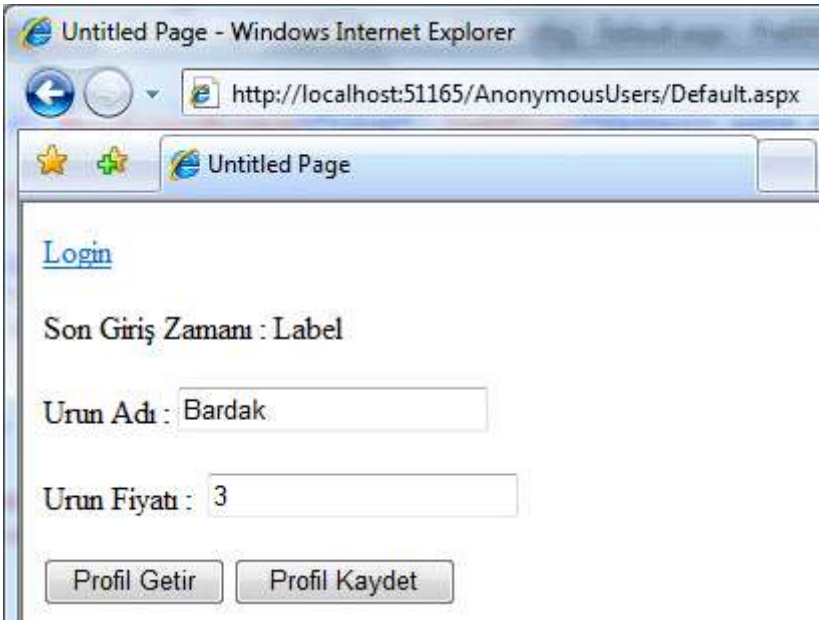
```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>Untitled Page</title>
   </head>
   <body>
      <form id="form1" runat="server">
         <div>
            <asp:LoginName ID="LoginName1" runat="server" />
            <asp:LoginStatus ID="LoginStatus1" runat="server" />
         <br />
         <br />
         Son Giriş Zamanı :<asp:Label ID="lblSonGirisZamani" runat="server"
Text="Label"></asp:Label><br />
         <br />
         Urun Adı :<asp:TextBox ID="txtUrunAdi" runat="server"></asp:TextBox><br />
         <br />
         Urun Fiyatı : <asp:TextBox ID="txtUrunFiyati"
runat="server"></asp:TextBox>
         <br />
         <br />
         <asp:Button ID="btnProfilGetir" runat="server" OnClick="btnProfilGetir_Click"
Text="Profil Getir" />
         <asp:Button ID="btnProfilKaydet" runat="server"
OnClick="btnProfilKaydet_Click" Text="Profil Kaydet" /></div>
      </form>
```

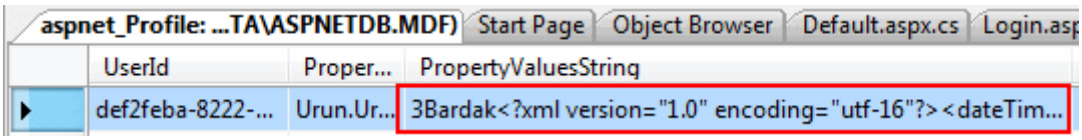
```
   </body>
</html>
```

Default.aspx.cs;

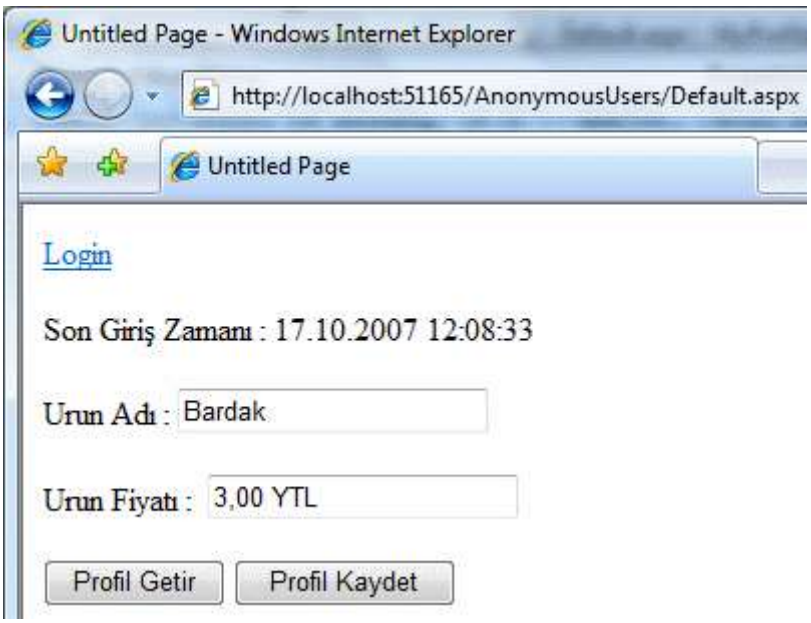
```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
   protected void btnProfilGetir_Click(object sender, EventArgs e)
   {
      try
      {
         lblSonGirisZamani.Text = Profile.SonGirisZamani.ToString();
         txtUrunAdi.Text = Profile.Urun.UrunAdi;
         txtUrunFiyati.Text = Profile.Urun.UrunFiyati.ToString("C2");
      }
      catch (Exception exp)
      {
         Response.Write(exp);
      }
   }
   protected void btnProfilKaydet_Click(object sender, EventArgs e)
   {
      Profile.SonGirisZamani = DateTime.Now;
      Profile.Urun.UrunAdi = txtUrunAdi.Text;
      double urunFiyati = 1;
      Double.TryParse(txtUrunFiyati.Text, out urunFiyati);
      Profile.Urun.UrunFiyati = urunFiyati;
      Profile.Save();
   }
}
```

Uygulamayı çalıştırdığımızda isimsiz bir kullanıcı ile default.aspx sayfasını açabildiğimizi görürüz. Bu aşamada örnek bir kaç veri girip Profil Kaydet başlıklı düğmeye bastığımızda ise bilgilerin başarılı bir şekilde kaydedildiğini görebiliriz. örneğin aşağıdaki veriler girildiğinde veritabanındaki **aspnet_profile** tablosunada bir satır eklendiği görülecektir.

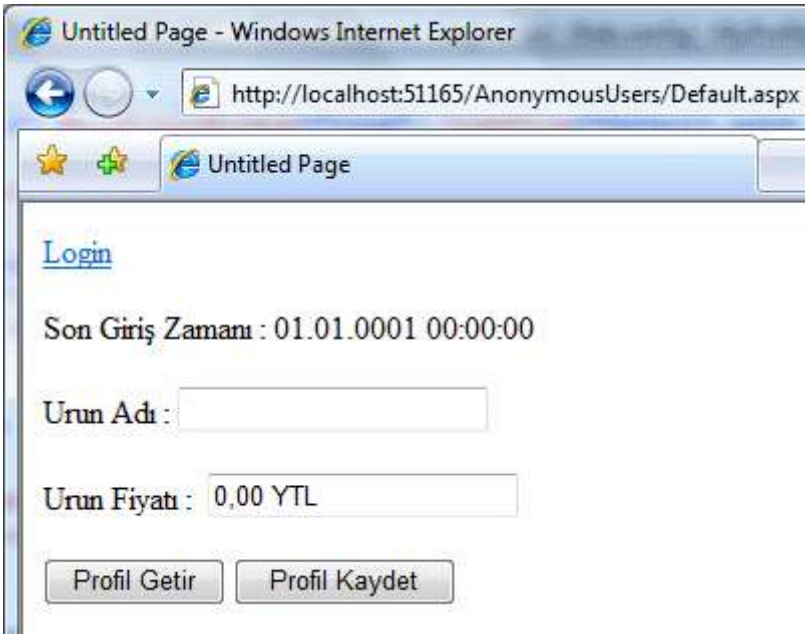




Dahası kullanıcı siteye 3 dakikalık süre zarfı içerisinde yeni bir **talepte(request)** bulunduğunda ve Profil Getir başlıklı düğmeye bastığında ilgili bilgileri elde edebiliecektir.

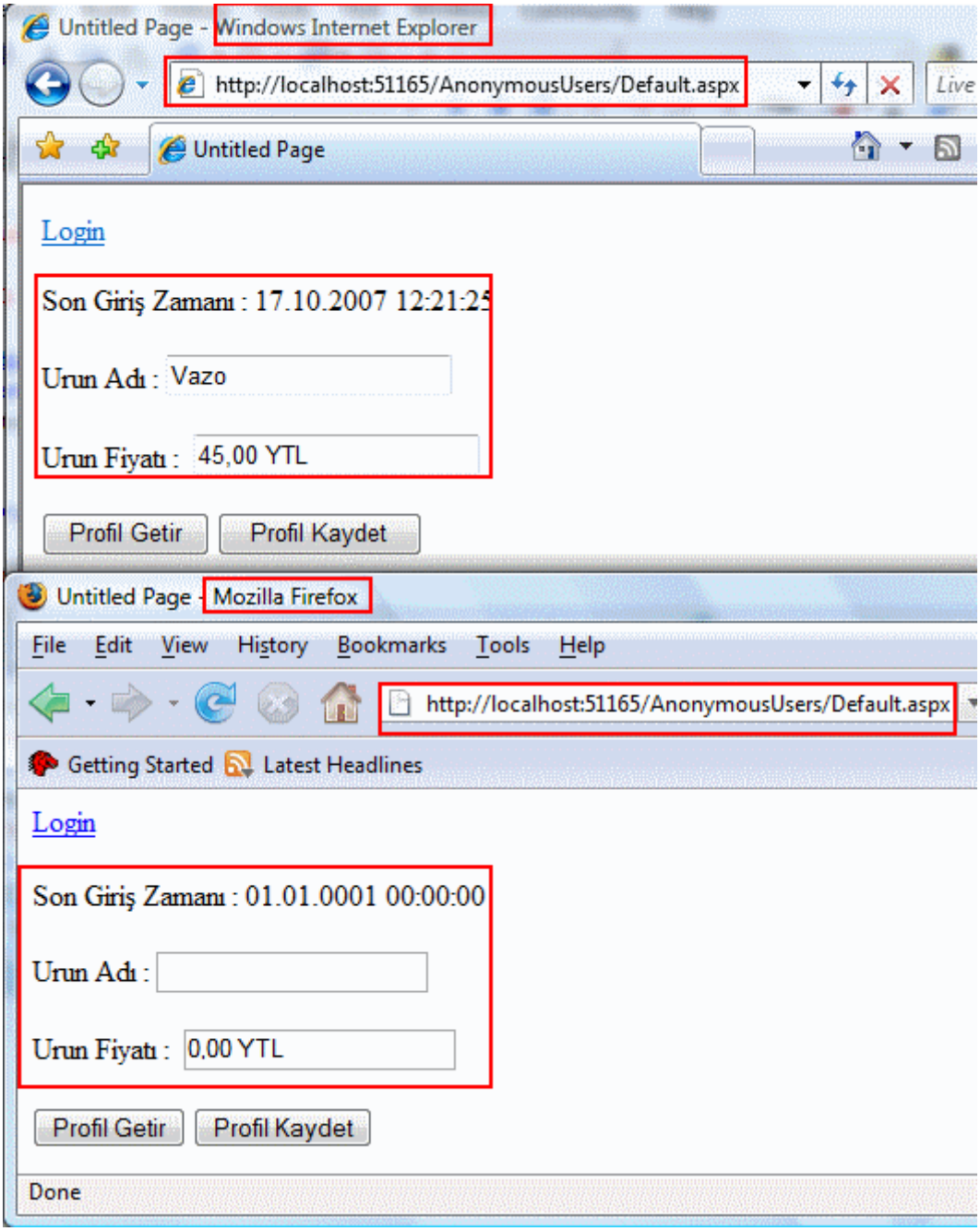


çok doğal olaraktan timeout süresi sona erdikten sonra profil bilgileri getirilmek istenirse aşağıdaki gibi bir ekran görüntüsü alınacaktır.



Bir başka deyişle istemci bilgisayardaki **çerez(cookie)** silindiğinden sunucuda eşleşen bir **GUID** bulunamamaktadır. *(GUID' in bulunamaması isimsiz kullanıcının aspnet_users tablosundan silindiği anlamına gelmemektedir)* Buda istenen bilgilerin elde edilemeyeceği anlamına gelir. Elbetteki veriler **aspnet_profile** tablosunda durmaya devam edecektir. Bu noktada **ProfileManager** sınıfının ilgili metodlarından yararlanarak belirli periyodlarda söz konusu verilerin temizlenmesi sağlanabilir.

Gelelim kullanıcının aynı uygulamaya isimsiz olarak farklı tarayıcılar ile erişmesi durumuna. Bu amaçla örneği önce **Internet Exlplorer** ile açıp profil bilgilerini kaydediyor olacağız. Sonrasında ise 3 dakikalık zaman dilimi sona ermeden **Firefox** ile aynı sayfayı yeniden talep edeceğiz. örnek olarak ürün adını Vazo, ürün fiyatını 4.5 olarak girdiğimizi düşünelim.



Dikkat edileceği üzere **Internet Explorer** kullanılıp kaydedilen profil bilgilerine 3 dakikalık süre zarfı içerisinde başka bir tarayıcı program olan **Firefox Mozilla** içerisinde erişilememiştir. Tersi durumda söz konusudur. Buna göre farklı tarayıcı pencereleri ile gelen taleplerde(request) sunucunun farklı bir **GUID** üreteceğini ve istemci bilgisayara çerez olarak yazacağını göz önüne almalıyız.

İsimsiz kullanıcılar(anonymous users) ile ilgili bir diğer durumda var olan doğrulanmış bir kullanıcı ile birleştirilmeleridir**(Migration)**. Bu durumu daha kolay anlayabilmek için **amazon.com**sitesinin işleyiş şeklini göz önüne alabiliriz. Bu siteye giren kayıtlı bir kullanıcı login olmadan sepete ürünler ekleyebilmektedir. Diğer taraftan kullanıcı alışveriş safhasına geçip sayfaya Login olduğunda, isimsiz kullanıcı olarak sepete attığı bilgiler var olan kullanıcı hesabındaki sepet bilgilerine eklenebilmektedir. Asp.Net 2.0 mimarisinde bu

tarz bir işlemi belirli bir ölçüde kontrollü olarak gerçekleştirebilmek için için **Global.asax.cs** dosyasında **Profile_OnMigrateAnonymous** olay metodunun yüklenmesi yeterlidir. Bu metodun aldığı**ProfileMigrateEventArgs** tipinden parametre sayesinde isimsiz kullanıcı(anonymous user) için üretilen **GUID** değerine erişilebilir ve ilgili değerlerin alınarak, login olan kullanıcıya aktarılması sağlanabilir. Yanlız bu noktada daha öncede login olup profil bilgisi kaydedilmiş bir kullanıcının bilgileri üzerine yazılmamaya çalışılmasına özen gösterilmelidir. Bu durumu analiz edebilmek için **global.asax.cs** dosyasına aşağıdaki kod parçasını eklediğimizi göz önüne alabiliriz.

```
<%@ Application Language="C#" %>

<script runat="server">

   void Profile_OnMigrateAnonymous(object sender, ProfileMigrateEventArgs e)
   {
      /* önce isimsiz kullanıcının profile bilgisi elde ediliri. GetProfile metodunun dönüş değeri ProfileCommon tipinden olacağı için buna göre bir atama yapılır. */
      ProfileCommon isimsizProfil = Profile.GetProfile(e.AnonymousID);
      /* Login olan doğrulanmış kullanıcının halen aktiv olup olmadığı öğrenilir. Eğer aktiv ise profil bilgileri mevcut demektir ve üstüne yazılması istenmez. Bu nedenle LastActivityDate değerine bakılarak bir kontrol yapılması gerekmektedir. */
      if (Profile.LastActivityDate.Date == DateTime.MinValue.Date)
      {
         // İsimsiz kullanıcının profil özelliklerinin değerleri ile login olan doğrulanmış kullanıcının profil özellikleri eşleştirilir
         Profile.SonGirisZamani = isimsizProfil.SonGirisZamani;
         Profile.Urun.UrunAdi = isimsizProfil.Urun.UrunAdi;
         Profile.Urun.UrunFiyati = isimsizProfil.Urun.UrunFiyati;
         Profile.Save(); // Login olan kullanıcı için değiştirilen profil değerleri kaydedilir.
      }
      AnonymousIdentificationModule.ClearAnonymousIdentifier();// isimsiz kullanıcı için üretilen çerezin silinmesi sağlanırki bu olay tekrar tetiklenmesin
      ProfileManager.DeleteProfile(e.AnonymousID); //isimsiz kullanıcıya ait profil değerleri veritabanından silinir.
   }

</script>
```

Yukarıdaki örnek kod parçasında öncelikli olarak isimsiz kullanıcının profil bilgileri elde edilmektedir. Bu amaçla **GetProfile** metodu kullanılmıştır. Sonrasında ise, o an Login olmuş kullanıcının bir profil bilgisi olup olmadığı tespit edilir. Bu kontrolde **LastActivityDate** özelliğinden yararlanılır. Burada amaç isimsiz kullanıcının(anonymous users) profil bilgilerini, login olmuş kullanıcının var olan profil bilgileri üzerine yazmamaktır. *(Burada global bir kontrol yapılıp istemciden bilgilerin üstüne yazmak isteyip istemeyeceği ele alınaraktanda ilerlenebilir. Bu tamamen*

uygulamanın veya sürecin ihtiyaçları doğrultusunda verilebilecek bir karardır.) Sonrasında login olan kullanıcının profile kaydettiği bir bilgi yoksa isimsiz kullanıcı için oluşturulan değerlerin ataması yapılır ve **kaydetme işlemi(save)** gerçekleştirilir. Son olarak olay metodunun tekrardan tetiklenmemesi amacıyla isimsiz kullanıcıya ait bilgi veritabanından silinir. Ayrıca istemcideki çerez bilgiside **AnonymousIdentificationModule** sınıfının **static ClearAnonymousIdentifier** metodu ile geçersiz kılınır. Sonuç olarak isimsiz kullanıcı giriş yapıp profil bilgisini kaydettikten sonra daha önceden profil bilgisi bulunmayan bir kullanıcı gibi Login olursa, var olan bilgileri doğrulanmış kullanıcınınkine aktarılacaktır.

Böylece geldik uzun bir makalemizin daha sonuna. Bu makalemizde profile yönetimini biraz daha güçlü ve esnek hale getirmek için neler yapabileceğimizi incelemeye çalıştık. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

ProfileManagement.rar (27,77 kb)

WPF - Sayfa(Page) Kavramı, Navigasyon İşlemleri ve XBAP (2007-10-04T05:20:00)

wpf,xbap,

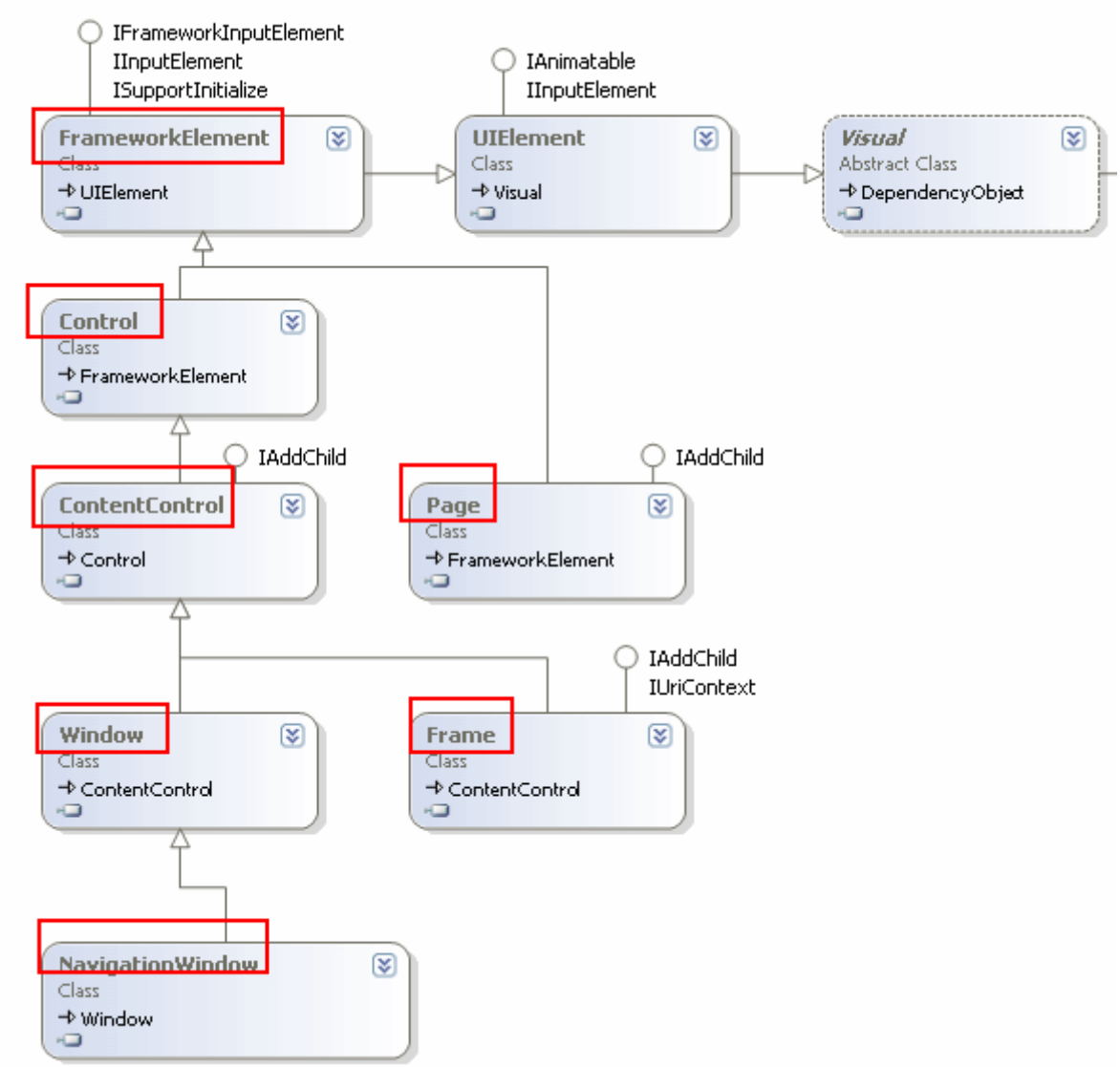
Yıllardır ister büyük çaplı ister küçük çaplı olsun pek çok proje web tabanlı olarak geliştirilmektedir. Projelerde bu tip bir seçime gitmenin en büyük nedenlerinden biriside web uygulamalarındaki **dağıtım modelinin(Deployment Model)** Windows tabanlı olanlara göre çok daha kolay olmasıdır. Her ne kadar .Net 2.0 ile birlikte gelen **ClickOnce** veya daha öncesinden beri var olan **ActiveX** gibi dağıtımı kolaylaştırabilecek teknolojiler var olsada bunlar Web uygulamalarına olan yönelimi azaltamamıştır. Sonuç itibariyle web tabanlı uygulamalarda, yazılan parçaları yorumlayacak bir tarayıcı penceresinin(Browser) olması yeterlidir. Geriye kalan, söz konusu tarayıcı pencerelerinin yorumlayacağı **HTML** içeriklerinin oluşturulmasıdır. Bu amaçlada son derece gelişmiş sunucu**(Server-Side)** veya istemci taraflı(Client-Side) uygulama geliştirme modelleri mevcuttur. Asp.Net bu modellerden yanlızca birisidir. Ancak web tabanlı uygulamalarda tarayıcı tarafından bakıldığında dağıtım dışında sağlanan başka avantajlarda vardır. Örneğin, tarayıcı penceresi yardımıyla uygulama**(Application)** alanı içerisinde bir sayfadan diğerine geçmek bir başka deyişle **navigasyon** işlemleri ile dolaşmak çok kolaydır. Bu tip bir kullanım kolaylığını Windows uygulamalarına kazandırmak ekstra kodlamayı gerekirmektedir.

Microsoft, **.Net Framework 3.0** ile birlikte Windows uygulamalarının tarayıcı pencereleri içerisinde çalıştırılabilmesini sağlayacak bir yenilik getirmektedir. Kısacası **XBAP(XAML Browser Applications)** olarak adlandırılan bu modelde, kısıtlamaları ile birlikte bir **WPF(Windows Prensetation Foundation)** uygulamasını bir tarayıcı penceresinde açmak mümkündür. Yazı dizimizde bu konuyada değiniyor olacağız. Ama öncesinde bunların temelini oluşturan sayfa**(Page)** kavramını anlamak gerekmektedir.

WPF uygulamalarını **sayfa tabanlı(Page-Based)** olacak şekilde tasarlayabilmekteyiz. Burada sayfadan kasıt **Page** tipinden bir nesnedir. WPF mimarisinde sayfa tabanlı uygulamalarda kendi içlerinde iki ana parçaya ayrılmaktadır. Bunlardan birincisi **XBAP** uygulamaları, diğeri ise kendi başına çalışan**(Stand-Alone)** uygulamalardır. Sayfalar(Pages) aslında daha önceki makalelerimizde de incelediğimiz **Window** tipine benzetilebilir. Lakin arada çok önemli bir fark vardır. Window tipi temel olarak bir taşıyıcı(Container) görevini üstelenebilmektedir. Bu sebeptende **ContentControl**tipinden türetilmiştir. Ne varki Page tipi doğrudan **FrameworkElement** tipinden türemektedir. Dolayısıyla Page tiplerinin kullanılabilmesi için bunu servis edecek bir sunucuya(Host) ihtiyaç vardır. Söz konusu özet bilgilere göre sayfa bazlı(Page-Based) uygulamaları aşağıdaki gibi kategorize edebiliriz.

Standalone (Kendi başına çalışan uygulamalar)	**NavigationWindow** içerisinde kullanılabilen sayfalar.
	Bir pencerede**(Window)** yer alan **Frame** veya Frame' ler içerisinde kullanılabilen sayfalar.
	Başka bir sayfa(Page) içerisindeki **Frame** veya Frame' lerde kullanılabilen sayfalar.
XBAP(Xaml Browser Applications) Uygulamaları	Internet Explorer veya destek veren başka bir tarayıcı(Browser) üzerinde çalışabilen sayfalar. **Lightweight** olarakta adlandırılan basit web tabanlı dağıtım modeli için uygun bir yapı sunmaktadır.

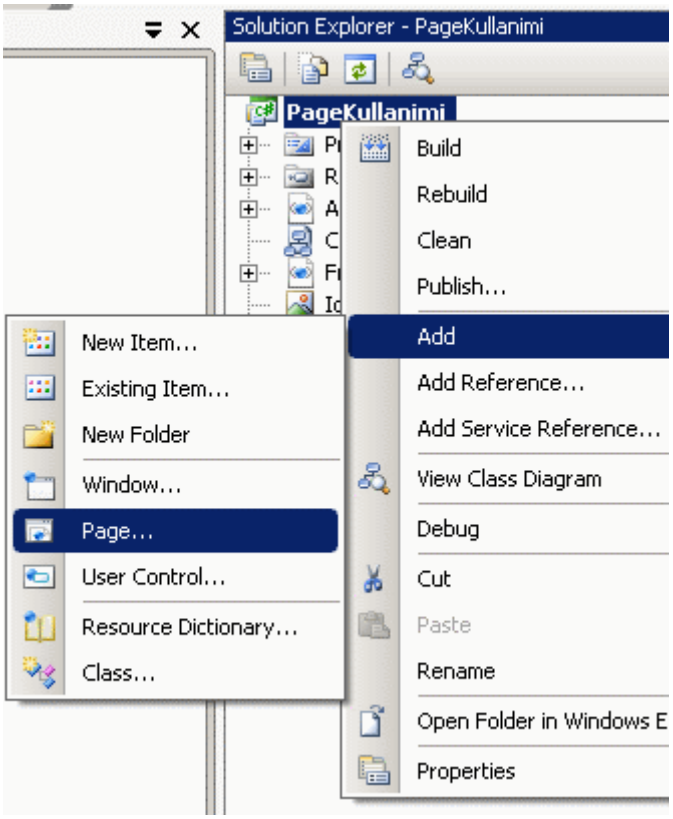
Sayfa tabanlı(Page-Based) uygulamalarda kullanılan genel tipler aşağıdaki sınıf diagramında**(Class Diagram)** görüldüğü gibidir.



Yukarıdaki sınıf diagramında sayfa-tabanlı(Page-Based) uygulamalarda başrol oynayan sınıflardan(class) bazıları yer almaktadır. **Window** sınıfı daha önceki windows programlamada yer alan Form sınıfının karşılığı olarak düşünülebilir. Bir içerik kontrolüdür(**ContentControl**). Bu nedenle kendi içerisinde başka elementleride barındırmaktadır. Söz gelimi Window içerisinde bir **Frame** tanımlanıp bu Frame içerisinde de farklı sayfalar(Page) yer alabilir. Frame tipi aslında bir sayfa içerisinde bağımsız bir parça olaraktanda düşünülebilir. Frame' leri bir **Page** veya **Windows** elementi içerisinde kullanabiliriz. Temel görevleri aslında web uygulamalarından bilinen benzeri ile aynıdır. Bir başka deyişle taşıyıcı kontrol içerisinde başka sayfaların(Page) gösterilebilmesini sağlamaktadır. Bu taşıyıcı özelliği nedeni ilede tahmin edileceği gibi **ContentControl** sınıfından türemektedir.**NavigationWindow**, içerisinde Page elementlerini içerebilen bir tiptir. Varsayılan olarak Page elementi içeren bir XAML içeriği code-behind dosyası ile birlikte çalıştırıldığında çalışma zamanında otomatik olarak bir **NavigationWindow** nesnesi örneklenmektedir. NavigationWindow nesneleri çalışma zamanında dinamik olaraktanda örneklenebilir ve sayfa içeriklerini göstermesi sağlanabilir.

Bu kısa teorik bilgilerden sonra dilerseniz basit örnekler yardımıyla konuyu daha iyi anlamaya çalışalım. Eğer aynı pencere üzerinde yer alacak ve aralarında geçişler yapılabilecek sayfalardan bahsediyorsak doğal olaraktan bunların arasında dolaşabilmek gerekmektedir. Dolaşma işlemleri için kullanılabilecek en basit kontrol **Hyperlink** bileşenidir. Bu bileşenin **NavigateUri**özelliğinden yararlanılarak başka bir sayfaya geçilmesi, aynı sayfa içerisinde veya başka bir sayfa içerisinde yer alan bir noktaya gidilmesi(burada anchor benzeri bir kullanımdan bahsediyoruz), başka bir **NavigationWindow** içerisinde bir sayfaya ve hatta var olan geçerli bir **Url** adresine gidilmesi sağlanabilir. Dolayısıyla ilk örneğimizde bu durumu analiz etmeye çalışıyor olacağız. Bu amaçla **Visual Studio 2008 Beta 2** sürümünde yeni bir WPF uygulaması açıp XAML içerikleri başlangıçta aşağıdaki gibi olan iki sayfa(Page) tasarlamamız yeterlidir.

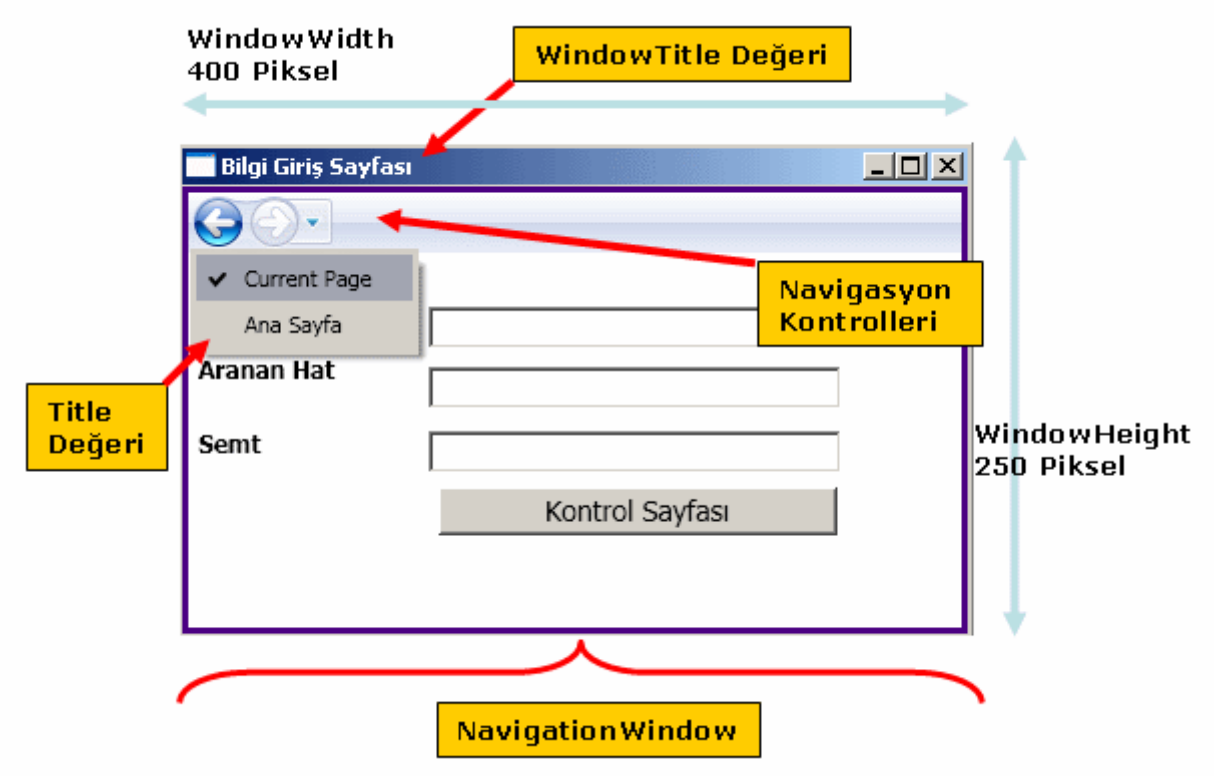
NOT : *Page tiplerini bir WPF uygulamasına eklerken* ***öğelerden(Item)*** *yararlanılabilir. Bunun için projeye sağ tıklayıp* ***Add New Item*** *penceresinden yada, doğrudan sağ tıklayınca çıkan menülerden* ***Add->Page*** *ile gerçekleştirebiliriz.*



MainPage.xaml

```
<Page x:Class="PageKullanimi.MainPage"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Ana
Sayfa" WindowHeight="250"WindowWidth="400" WindowTitle="Azon Giriş
Sayfası" Loaded="Page_Loaded">
  <Page.Background>
    <ImageBrush ImageSource="Iceberg.jpg" Opacity="0.4"/>
  </Page.Background>
  <Grid>
    <Label Foreground="Black" FontSize="15" FontWeight="Bold" Height="30"
HorizontalAlignment="Left" VerticalAlignment="Top" Width="113" Margin="9,21,0,0">
      <Hyperlink NavigateUri="PageX.xaml" ToolTip="Bir sonraki sayfaya
geçmenizi sağlar">Sonraki Sayfa</Hyperlink>
    </Label>
    <Button Name="btnBilgiSayfasi" Background="LightGray" Width="120"
Height="30" FontSize="12" FontWeight="Bold" HorizontalAlignment="Left"
Margin="9,107,0,0" VerticalAlignment="Top">
      <Hyperlink Foreground="Red" NavigateUri="Page2.xaml">Bilgi Giriş
Sayfası</Hyperlink>
    </Button>
    <TextBox Name="txtHosgeldinMesajim" Margin="9,60,33,0" Height="21"
VerticalAlignment="Top" />
  </Grid>
</Page>
```

MainPage içerisinde ilk dikkati çeken noktalardan birisi **Page** elementi ve özellikleridir. **WindowHeight** ve **WindowWidth** özellikleri ile çalışma zamanındaki **NavigationWindow** penceresinin boyutları set edilmektedir. **WindowTitle** özelliği ile sayfanın NavigationWindow içerisinde gösterilirken sahip olacağı başlık değeri verilmektedir. **Title** özelliği ise, navigasyon işlemlerinin yapıldığı Combobox içerisindeki başlık bilgisini belirlemektedir. Aşağıdaki şekil çalışma zamanındaki bir ekran görüntüsü olup bahsedilen özellikleri ifade etmektedir.

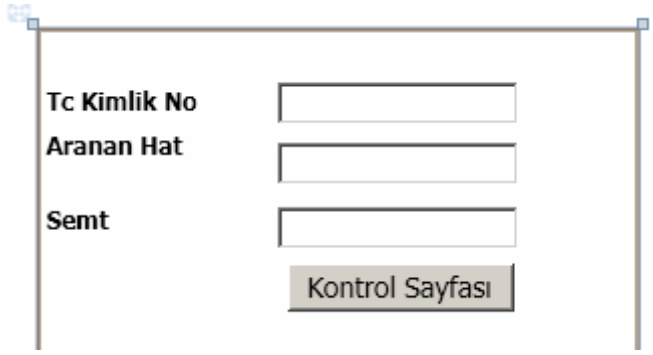


NOT : *Page sınıfının Window sınıfında olduğu gibi* ***Show*** *veya* ***Hide*** *gibi metodları bulunmamaktadır. Bunun en büyük nedeni Page sınıflarına ait nesne örnekleri aralarında gezinirken navigasyon kontrollerinden yararlanılabilmesidir. Dolayısıyla klasik windows programcılığından bildiğimiz Show veya Hide gibi işlemlere gerek kalmamaktadır.*

Bunların dışında MainPage içerisinde Label ve Button kontrollerine ait elementler içerisinde Hyperlink alt elementleri kullanılmıştır. **Hyperlink** bileşeni bağımsız bir element olarak kullanılamamaktadır. Dolayısıyla **Inline-Flow** tipinden bir kontroldür. Diğer taraftan bu elementin **NavigateUri** özelliği ile gidilmek istenen sayfalar belirtilmektedir.

NOT : Hyperlink *elementi* ***Page*** *yerine bir* ***Window*** *içerisinde kullanılmak istendiğinde belirtilen adrese otomatik olarak gidilemeyecektir. Böyle bir durumda Window sınıfının****RequestNavigate*** *olayının bilinçli olarak ele alınması ve yönlendirme işleminin manuel olarak yapılması gerekmektedir.*

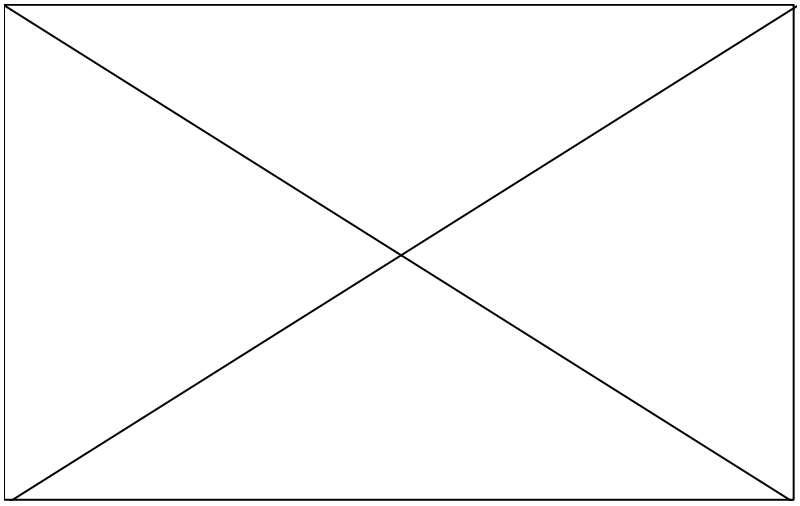
Label kontrolünde yer alan Hyperlink elementinde kasıtlı olarak projede yer almayan PageX.xaml' e gidilmeye çalışılmaktadır. Burada amaç olmayan bir adrese gidilmek istendiğinde ne olacağının analiz edilmesidir. Button kontrolü içerisinde yer alan **Hyperlink** elementinde ise, **XAML** içeriği aşağıda yer alan Page2 isimli sayfaya gidilmektedir.



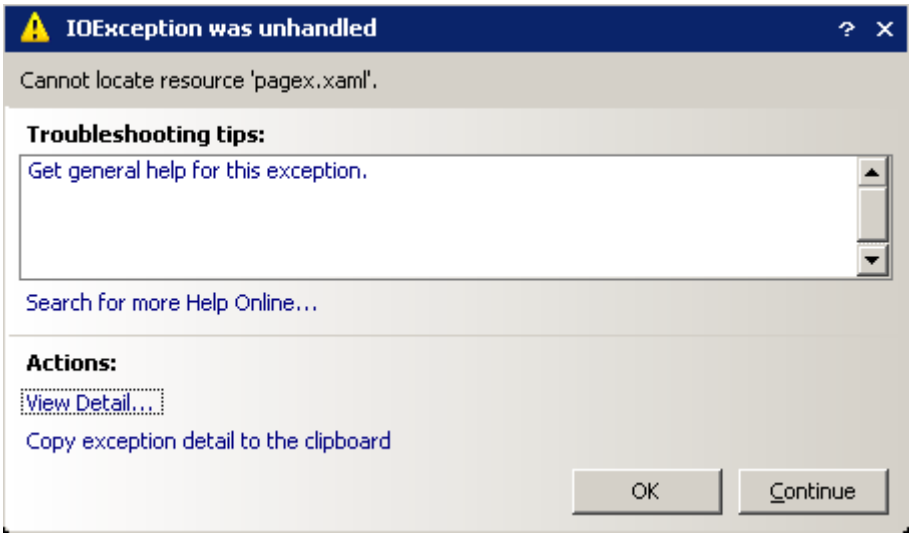
Page2.xaml

```
<Page x:Class="PageKullanimi.Page2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" WindowTitle="Bilgi Giriş
Sayfası" Title="Bilgi Girişi" Loaded="Page_Loaded">
   <Grid>
      <Label FontSize="12" FontWeight="Bold" Height="25" HorizontalAlignment="Left"
VerticalAlignment="Top" Width="85" Margin="0,24,0,0">Tc Kimlik No</Label>
      <Label FontSize="12" FontWeight="Bold" HorizontalAlignment="Left" Width="85"
Margin="0,83,0,0" Height="25" VerticalAlignment="Top">Semt</Label>
      <Label FontSize="12" FontWeight="Bold" Height="25" HorizontalAlignment="Left"
VerticalAlignment="Top" Width="85" Margin="0,46,0,0">Aranan Hat</Label>
      <Button FontSize="14" Margin="125,117,62,0" Name="btnKontrolSayfasi"
Click="btnKontrolSayfasi_Click" Height="24" VerticalAlignment="Top">
         Kontrol Sayfası
      </Button>
      <TextBox Name="txtTcNo" Height="21" Margin="120,27,60,0"
VerticalAlignment="Top" />
      <TextBox Name="txtArananHat" Height ="21" Margin="120,57,60,0"
VerticalAlignment="Top" />
      <TextBox Name="txtSemt" Height ="21" Margin="120,89,60,0"
VerticalAlignment="Top" />
   </Grid>
</Page>
```

İlk olarak örneği çalışma zamanında test ederek işe başlayalım. Uygulamanın yürütülmeye başlaması halinde MainPage.xaml sayfasının örneklenmesi için **App.xaml**' e ait Application elementi içerisindeki **StartupUri** özelliğinin değeri MainPage.xaml olarak ayarlanmıştır. Aşağıdaki Flash görselinde ilk çalışma hali gösterilmektedir.



Dikkat edilecek olursa **çalışma zamanında(Run-Time)** otomatik olarak bir navigasyon çubuğu oluşturulmuştur. İlk aşamada bu çubuk üzerindeki düğmeler pasiftir. Aktif olmaları için sayfalar arasında geçiş yapılması gerekmektedir. Sayfalar arası geçiş yapıldıktan sonra navigasyon düğmeleri ile ileri ve geri yönlü hareketler yapılabilir. Bununla birlikte Combobox kontrolünden yararlanılaraktanda diğer sayfalara daha kolay bir şekilde geçiş yapılmasıda sağlanabilir. Bu örnekte PageX.xaml sayfasına geçiş yapılmasını sağlayan Label kontrolüne basılmamıştır. Bu yapıldığı takdirde söz konusu sayfa olmadığı için aşağıdaki ekran görüntüsünde yer alan bir çalışma zamanı **istisnası(exception)** alınacaktır.



Görüldüğü gibi basit bir **IOException** alınmıştır. Elbetteki programatik olarak uygulamayı tasarlarken olmayan sayfalara gidilmemesini sağlamak geliştiricinin görevidir.
Lakin **NavigateUri** ile var olan bir sayfa dışında geçerli bir URL adresinede gidilebilmesi sağlanabilmektedir. Bir başka deyişle var olan bir web sayfasını çalışma zamanında oluşturulan **NavigationWindow**içerisinde bir sayfa olarak göstermek mümkündür. Bu gibi durumlarda gidilmek istenen **URL** veya Sayfa bilgisinin geçerli olmaması halinde programın istem dışı bir şekilde sonlanmasının önüne geçmek için **Application** nesnesinin **NavigationFailed** olayını ele almak ve içerisinde istisna bilgisini kontrollü bir şekilde yakalamak en doğru yaklaşım olacaktır. Yukarıdaki örnekte

bunu uygulamak istediğimiz App.xaml ve App.xaml.cs içeriklerinin aşağıdaki gibi tasarlanması yeterlidir.

App.xaml;

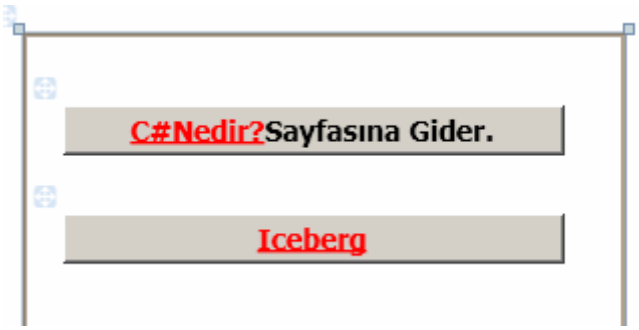
```
<Application x:Class="PageKullanimi.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" StartupUri="MainPage.xaml
" NavigationFailed="Application_NavigationFailed">
   <Application.Resources>
   </Application.Resources>
</Application>
```

App.xaml.cs;

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Windows;

namespace PageKullanimi
{
   public partial class App : Application
   {
      private void Application_NavigationFailed(object
sender, System.Windows.Navigation.NavigationFailedEventArgs e)
      {
         if (e.Exception != null)
         {
            MessageBox.Show(e.Uri.ToString() + " adresi bulunamadı");
            e.Handled = true;
         }
      }
   }
}
```

Örnekte basit olması açısında sadece hataya neden olsan sayfanın **Uri** bilgisi bir **MessageBox** içerisinde gösterilmektedir. Olay metoduna gelen **NavigationFailedEventArgs** tipinden e parametresinin**Handled** özelliğine true değeri atanmasının sebebi hatanın kontrollü bir şekilde ele alındığının belirtilmesidir. Aksi durumda program yine hata sonrası, kullanıcıya oluşan hatanın gönderilip gönderilmeyeceğini soran hepimizin yakından tanıdığı mesaj kutusu ile sonlandırılacaktır. Burada yakalanan hatalar çok doğal olarak başka amaçlarlada değerlendirilebilir. Örneğin Log' lanarak, oluşan hatalar ile ilişkili genel istatistik ve analizlerin yapılması sağlanabilir.

Hyperlink kontrolünü kullanarak web sayfalarınada gidilebildiğinden bahsetmiştik. Bunun dışında bir sayfa içerisinde yer alan herhangibir konuma gidilmeside sağlanabilirki bu durum **parçalı navigasyon(Fragment Navigation)** olarak adlandırılmaktadır. Şimdi bu iki kullanım şeklini ele alacağımız bir örnek üzerinden ilerleyelim. Bu amaçla projemize Page3.xaml ve Page4.xaml sayfalarını aşağıdaki içerikleri ile eklediğimizi düşünebiliriz.



Page3.xaml;

```
<Page x:Class="PageKullanimi.Page3"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" WindowTitle="Doğrulama
Sayfası" Title="Doğrulama" WindowHeight="250" WindowWidth="400">
   <Grid>
      <Button Name="btnCSHarpNedir" FontSize="14" FontWeight="Bold" Height="25"
Margin="20,36,30,0" VerticalAlignment="Top">
         <TextBlock>
            <Hyperlink Foreground="Red"
NavigateUri="http://www.csharpnedir.com">C#Nedir?</Hyperlink>Sayfasına
Gider.
         </TextBlock>
      </Button>
      <Button Name="btnIceberg" FontSize="14" FontWeight="Bold" Height="25"
VerticalAlignment="Top" Margin="20,90,30,0">
         <TextBlock>
            <Hyperlink NavigateUri="Page4.xaml#txtBilgi"
Foreground="Red">Iceberg</Hyperlink>
         </TextBlock>
      </Button>
   </Grid>
</Page>
```

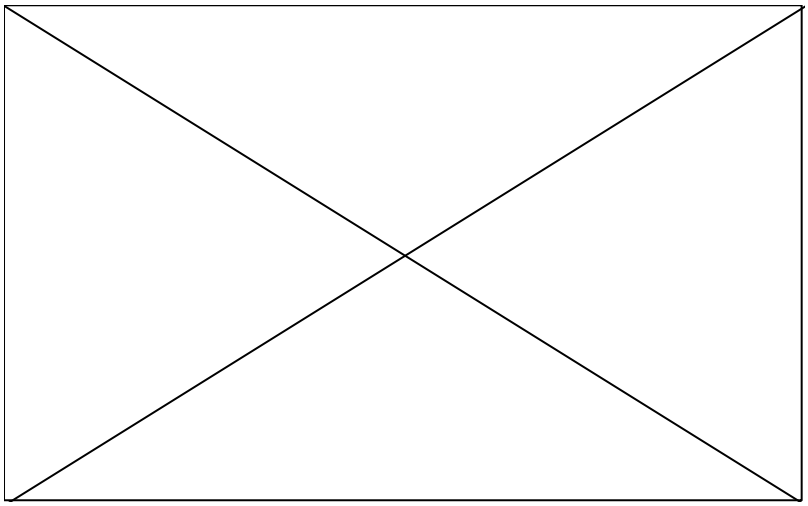
Sayfanın içerisinde kullanılan iki adet **Hyperlink** kontrolü bulunmaktadır. Bunlardan birisi **C#Nedir?** sitesine, diğeri ise Page4.xaml sayfası içerisinde txtBilgi isimli bileşenin olduğu yere yönlendirme yapmaktadır. İkinci navigasyon işlemi aslında parçalı navigasyon(Fragment Navigation) işlemi için bir örnektir. Öyleki Page4.xaml sayfası içerisinde yer alan txtBilgi isimli kontrol sayfanın alt kısmında yer almaktadır. Bu nedenle

sayfa içerisindeki bu yere navigasyon işlemi ile gidilmesi ve odaklanılması mümkün olabilmektedir.

Page4.xaml;

```
<Page x:Class="PageKullanimi.Page4"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Antartika Iceberg"
WindowTitle="Iceberg">
   <Grid>
      <ScrollViewer VerticalScrollBarVisibility="Visible">
         <Canvas Height="299" Width="284">
            <Image Canvas.Left="18" Canvas.Top="51" Height="104" Name="image1"
Width="150" Source="Iceberg.jpg" />
            <Label Canvas.Left="18" Canvas.Top="26" Height="23" Name="label1"
Width="120">Antartika Hakkında</Label>
            <Label Canvas.Left="18" Canvas.Top="166" Height="23" Name="label2"
Width="120">
               <Hyperlink NavigateUri="Page4.xaml#txtBilgi">Genel Bilgi</Hyperlink>
            </Label>
            <TextBox Height="25" Name="txtBilgi" Canvas.Left="18" Canvas.Top="274"
Width="249" />
         </Canvas>
      </ScrollViewer>
   </Grid>
</Page>
```

Page4.xaml içerisinde de bu sayfadaki txtBilgi kontrolüne odaklanılmasını sağlayan bir **Hyperlink** bileşeni bulunmaktadır. Dikkat edilecek olursa parçalı navigasyon işlemlerinde **NavigateUri**özelliğinde **#** işaretinden sonra bir bilgi yer almaktadır. Bu bilgi gidilmek istenen kontrolün **Name** özelliğinin(Property) değeridir. Örneğin çalışma zamanındaki işleyişi aşağıdaki Flash animasyonunda olduğu gibidir.



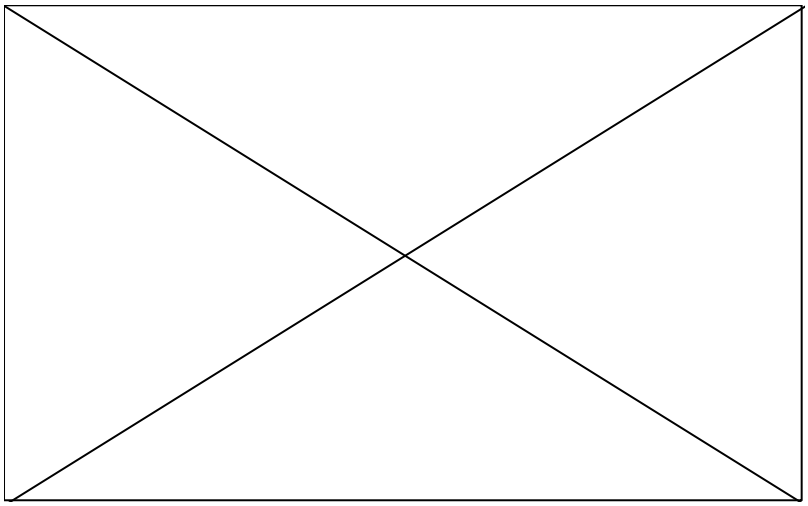
Görüldüğü gibi Page3.xaml içerisinde yer alan düğme kontrollerine basıldığında C#Nedir? sayfasına veya Page4.xaml içerisindeki txtBilgi isimli **TextBox** kontrolünün olduğu yere gidilmektedir. Herhangibir internet veya intranet adresine gidilmesini sağlayan teknikte mutlaka hata kontrolü yapılmalıdır. Diğer taraftan bir web sayfasına gidildiğinde sayfada görünen kısım klasik windows programlamadan tanıdığımız **WebBrowser** kontrolünün sunduğu ortama benzemektedir. Bu sebepten dolayı elde edilen sayfa içerisinde arama yapmak, dinamik kod çalıştırmak gibi işlemler yapılamamaktadır.

NOT : Parçalı navigasyon(Fragment Navigation) *işleminin olabilmesi için, söz konusu sayfa içerisinde aşağı yukarı hareket edilebilmesi bir başka deyişle scrolling olması gerekmektedir. Bu amaçla* ***ScrollViewer*** *kontrolünden yararlanılabilir. Söz konusu kontrolün* ***VerticallScorllBarVisibility*** *ve* ***HorizontalScrollBarVisibility*** *özelliklerine ilgili değerler atanarak dikey veya yatay yönde kaydırma çubuklarının(Scroll Bar) gösterilmesi(yada tam tersi) sağlanabilir.*

Navigasyon işlemleri istenirse manuel olarak kod tarafından da gerçekleştirilebilir. Bu noktada **NavigationWindow** içerisinde üst kısımda görünen navigasyon kontrolleri ve menünün yaptığı işlerin kod yardımıylada gerçekleştirilmesi mümkündür. Bunun için **NavigationService** tipinden ve üyelerinden(Members) yararlanılabilir. Sıradaki örnekte yeni bir sayfaya geçiş işlemini kod ile nasıl yapabileceğimize bakıyor olacağız. Page2.xaml içerisindeki btnKontrolSayfasi isimli Button düğmesine tıklandığında Page3.xaml sayfasına geçilmesini sağlamak için ilgili olay metodunda aşağıdaki kodları yazmak yeterli olacaktır.

```
private void btnKontrolSayfasi_Click(object sender, RoutedEventArgs e)
{
   this.NavigationService.Navigate(new Page3());
}
```

NavigationService referansını Button bileşenine ait Click olay metodu içerisine yakaladıktan sonra **Navigate** fonksiyonuna parametre oalrak Page3 tipinin yeni bir nesne örneği verilmektedir. Böylece Page3 nesne örneği oluşturulup **NavigationWindow** içerisinde gösterilmesi sağlanmaktadır. Buna göre örneğin çalışma zamanındaki durumu aşağıdaki Flash görselindeki gibi olacaktır.

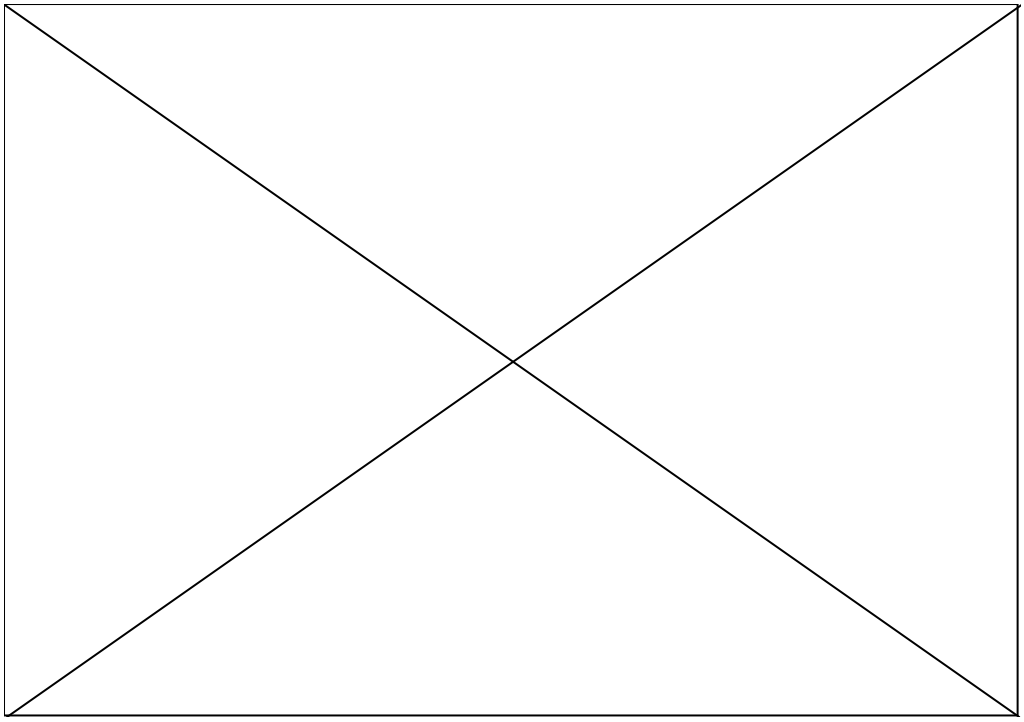


İstenirse çalışma zamanında yeni bir **NavigationWindow** oluşturulması ve bir sayfanın bu örnek üzerinde açılması sağlanabilir. Aşağıdaki örnek kod parçası ile, düğmeye basıldığı zaman Page3 isimli sayfanın yeni bir pencere içerisinde açılması sağlanmaktadır.

```
private void btnKontrolSayfasi_Click(object sender, RoutedEventArgs e)
{
   NavigationWindow nvgWnd = new NavigationWindow();
   Page3 pg3 = new Page3();
   pg3.WindowTitle = "Yeni Pencerede Açılan Doğrulama Sayfası";
   pg3.Title = "Doğrulama(Yeni)";
   pg3.WindowWidth = 300;
   pg3.WindowHeight = 240;
   //pg3.ShowsNavigationUI = false;
   nvgWnd.Content = pg3;
   nvgWnd.Show();
}
```

İlk olarak yeni bir **NavigationWindow** nesne örneği oluşturulmaktadır. Sonrasında ise bu pencerede gösterilmek istenen sayfa örneklenir. Sayfanın **WindowTilte, Title, WindowWidth, WindowHeight** gibi özellikleri set edildikten sonra NavigationWindow nesne örneğinin **Content** özelliğine oluşturulan Page3 nesne örneği atanır. Son

olarak **Show** metodu ile yeni pencerenin gösterilmesi sağlanmaktadır. Buradaki yorum satırı açılırsa eğer, yeni pencerede navigasyon kontrollerinin gösterilmemesi sağlanmış olur. Uygulamayı bu şekilde test ettiğimizde aşağıdaki Flash görselindeki etkiler görülecektir.

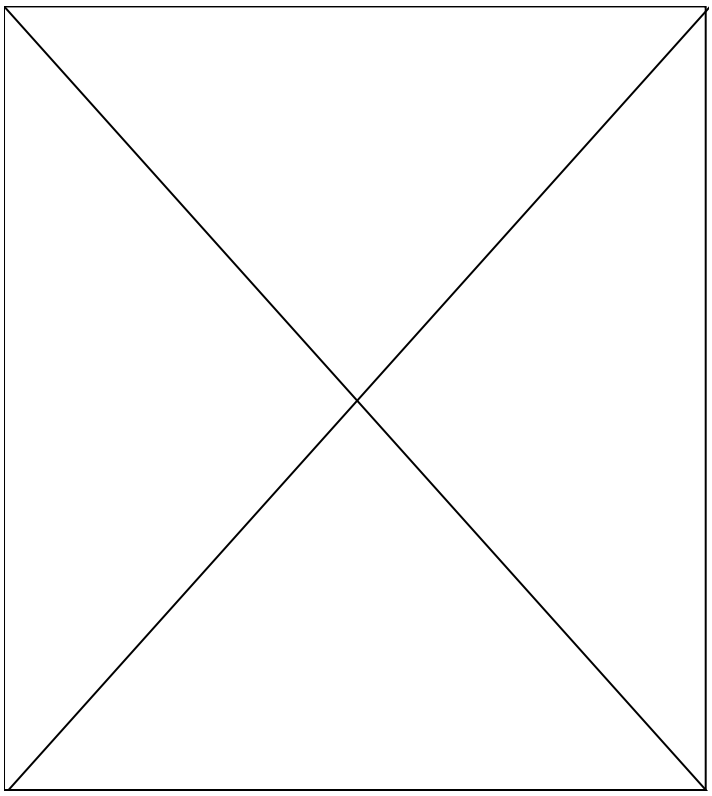


Sayfalar istenirse Frame' ler içerisinde gösterilebilirler. Böylece bir **NavigationWindow** içerisinde birden fazla Frame kullanılarak birden fazla sayfanın aynı anda gösterilmesi sağlanabilir. Örneğin, içerisinde harici bir web sitesini, uygulamanın kendisi, yardım dökümanını barındıracak şekilde bir pencere geliştirilebilir. **Frame** tipi kendi içerisinde çeşitli elementler barındırabilmektedir ancak genel kullanım amacı **Page** tiplerini taşımasıdır. Bu tanımlamalar Frame tipinin webdeki kullanım şeklini tam olarak andırdığınıda göstermektedir. Konuyu daha net anlayabilmek için bir örnek üzerinden ilerlemekte fayda olacağı kanısındayım. Bu amaçla projeye aşağıdaki XAML içeriğine sahip yeni bir pencere(Window) eklediğimizi düşünelim.

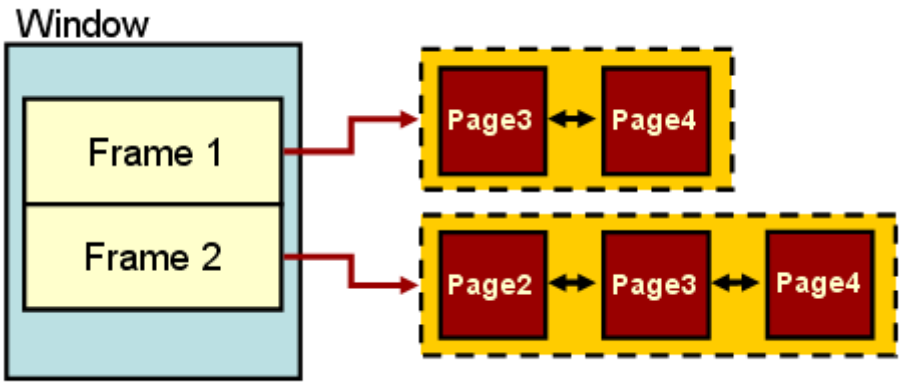
```
<Window x:Class="PageKullanimi.FrameKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Frame Kullanimi"
Height="400" Width="300">
   <Grid Margin="2">
      <Grid.RowDefinitions>
         <RowDefinition/>
         <RowDefinition/>
      </Grid.RowDefinitions>
      <Frame NavigationUIVisibility="Automatic" Grid.Row="0"
BorderBrush="Black" BorderThickness="3" Source="Page3.xaml"/>
      <Frame Grid.Row="1" BorderBrush="Gold" BorderThickness="3"
```

```
Source="Page2.xaml"/>
   </Grid>
</Window>
```

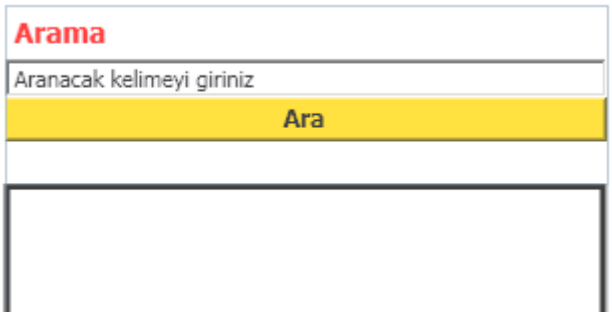
Frame kullanımını kolaylaştırmak için pencere içerisinde iki satırdan oluşan bir **Grid** kontrolü konulmuştur. Bu amaçla Grid kontrolü içerisine **RowDefinition** elementleri ile iki satır eklenmiştir. Hangi **Frame**' in hangi satırda gösterileceğini belirlemek için **Grid.Row** elementlerinden yararlanılmaktadır. Her Frame elementinin **Source** niteliklerine(attribute) atanan değerler ile içlerinde gösterecekleri sayfalar belirlenmektedir. **NavigationUIVisibility** niteliğine atanan değer ile navigasyon kontrolünün Frame içerisinde gösterilip gösterilmeyeceği veya örnekteki gibi bunun otomatik olarak set edilip edilmeyeceği belirlenebilir. Söz konusu Frame' ler dikkat edilecek olursa Page yerine bir **Window** elementi içerisinde kullanılmıştır. Örnek yürütüldüğünde aşağıdaki Flash animasyonunda olduğu gibi bir çalışma zamanı sonucu elde edilir.



Flash animasyonundan görülebileceği gibi ilk etapta navigasyon kontrolleri gösterilmemektedir. Ancak **Frame**' ler içerisinde yer alan sayfalar üzerindeki kontroller yardımıyla hareket edildikten sonra navigasyon kontrolleri görülmektedir. Tabi istenirse **NavigationUIVisibility** özelliğine **Visible** değeri atanarak navigasyon kontrolünün başlangıçta çıkmasıda sağlanabilir. Yukarıdaki örnekte meydana gelen işlemler aşağıdaki grafikle daha net anlaşılabilir.



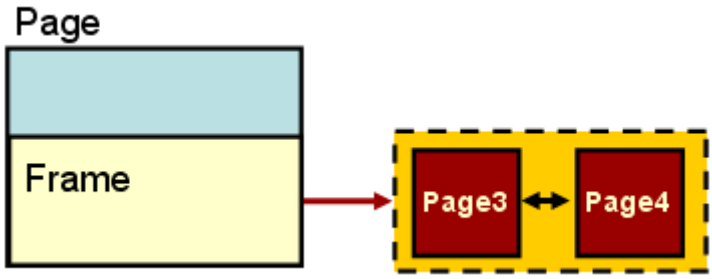
İstenirse sayfalar başka sayfaların içerisinde de kullanılabilir. Böyle bir durumda iç içe sayfalar(**Nested-Page)** söz konusu olmaktadır. Aşağıdaki XAML içeriğinde bir nested-page tasarımı örneği görülmektedir.



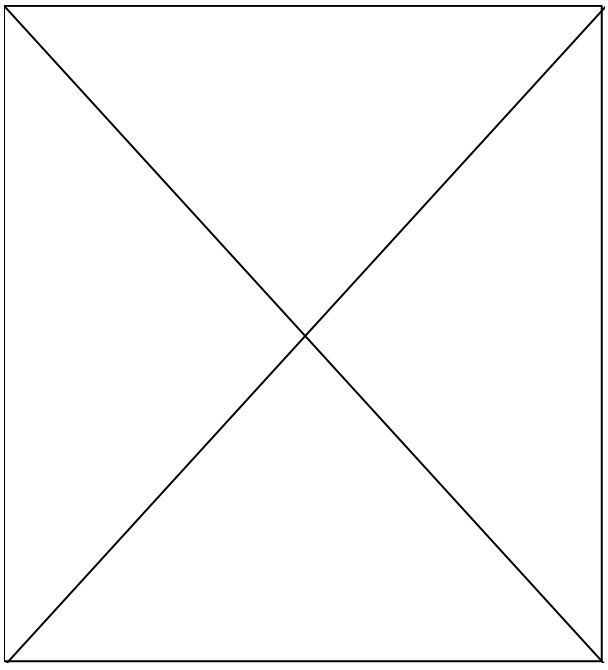
XAML içeriği;

```
<Page x:Class="PageKullanimi.NestedPageKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Nested Page Orneği"
WindowWidth="300" WindowHeight="300">
   <Grid>
      <Grid.RowDefinitions>
         <RowDefinition Height="89" />
         <RowDefinition Height="211" />
      </Grid.RowDefinitions>
      <StackPanel Grid.Row="0">
         <Label Foreground="Red" FontSize="14" FontWeight="Bold">Arama</Label>
         <TextBox Text="Aranacak kelimeyi giriniz"/>
         <Button Name="btnAra" Content="Ara" FontSize="12" FontWeight="Bold"
Background="Gold" Foreground="Black"/>
      </StackPanel>
      <Frame NavigationUIVisibility="Automatic" Grid.Row="1"
BorderBrush="Black" BorderThickness="3" Source="Page3.xaml"/>
   </Grid>
</Page>
```

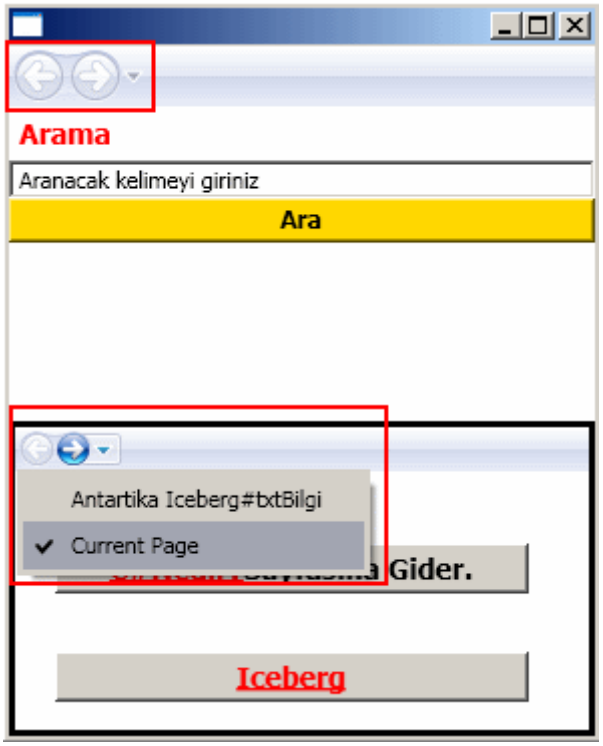
Dikkat edilecek olursa **Page** elementi içerisinde normal elementler dışında birde **Frame** elementi kullanılmaktadır. Bu element içerisinde **Source** niteliği ile gösterilecek olan sayfa belirtilmektedir. Böylece çalışan sayfa içerisinde birbirlerine bağlı başka sayfalarında gösterilebilmesi sağlanmaktadır. Olayı aşağıdaki grafik ile daha net kavrayabiliriz.



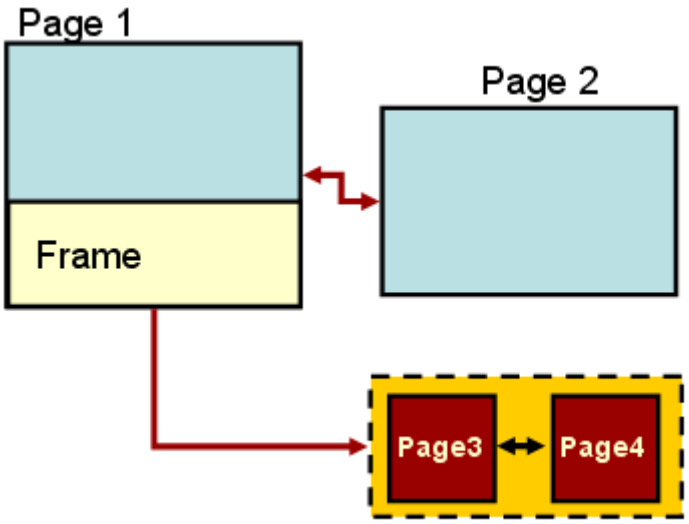
Uygulamanın çalışma zamanındaki görüntüsü aşağıdaki Flash animasyonunda olduğu gibidir.



Burada dikkat çekici noktalardan biriside **Frame** içerisinde sayfalar arasında gezinirken navigasyon kontrolünün sayfanın üst kısmında çıkmış olmasıdır. Bu durumda Frame' lerin kendi navigasyon çubuklarına sahip olması sağlanabilir. Bunun için Frame elementinin **JournalOwnerShip** adı verilen niteliğinden yararlanılır. Bu nitelik **Automatic**, **OwnsJournal** ve **UsesParentJournal**olmak üzere üç farklı değerden birisini alabilir. OwnsJournal değeri seçildiğinde aşağıdaki ekran görüntüsünde olduğu gibi Frame' in navigasyon kontrolü kendi içerisinde çıkacak, sayfanın kendi navigasyon kontrolü ise en üst tarafta yer alacaktır. Doğal olarak bunlar birbirleriylede karışmayacak şekilde çalışmaktadır.



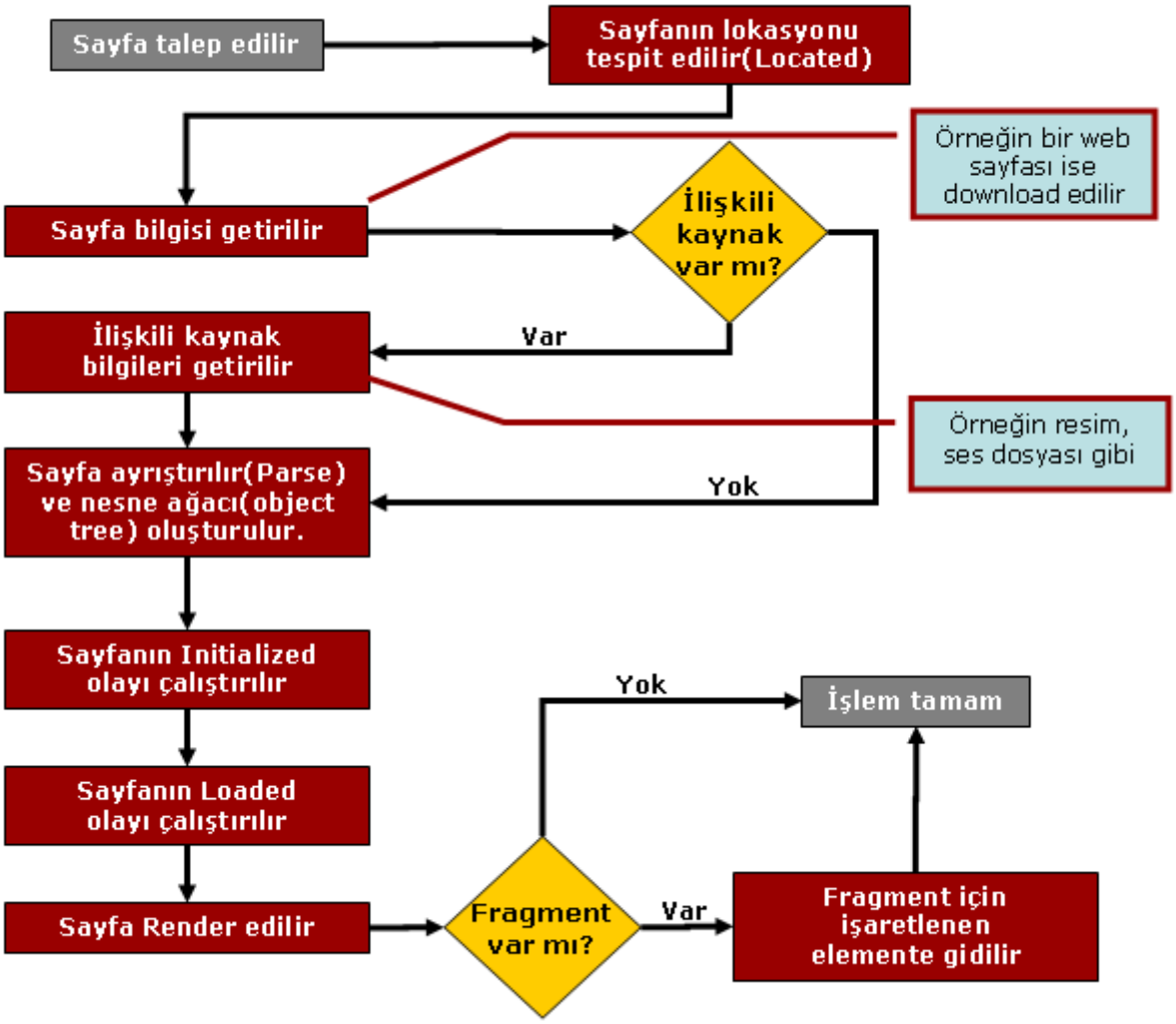
UsesParentJournal değerinin seçilmesi halinde ise, Frame elementinin navigasyon kontrolü bu örneğin ilk versiyonunda olduğu gibi üst tarafta yer alacaktır. Aslında burada analiz edilmesi gereken bir durum daha vardır. Buda sayfanın kendisinin navigasyon işlemlerine sahip olması halidir. Yani aşağıdaki grafikteki gibi bir senaryo olduğunu düşünelim.



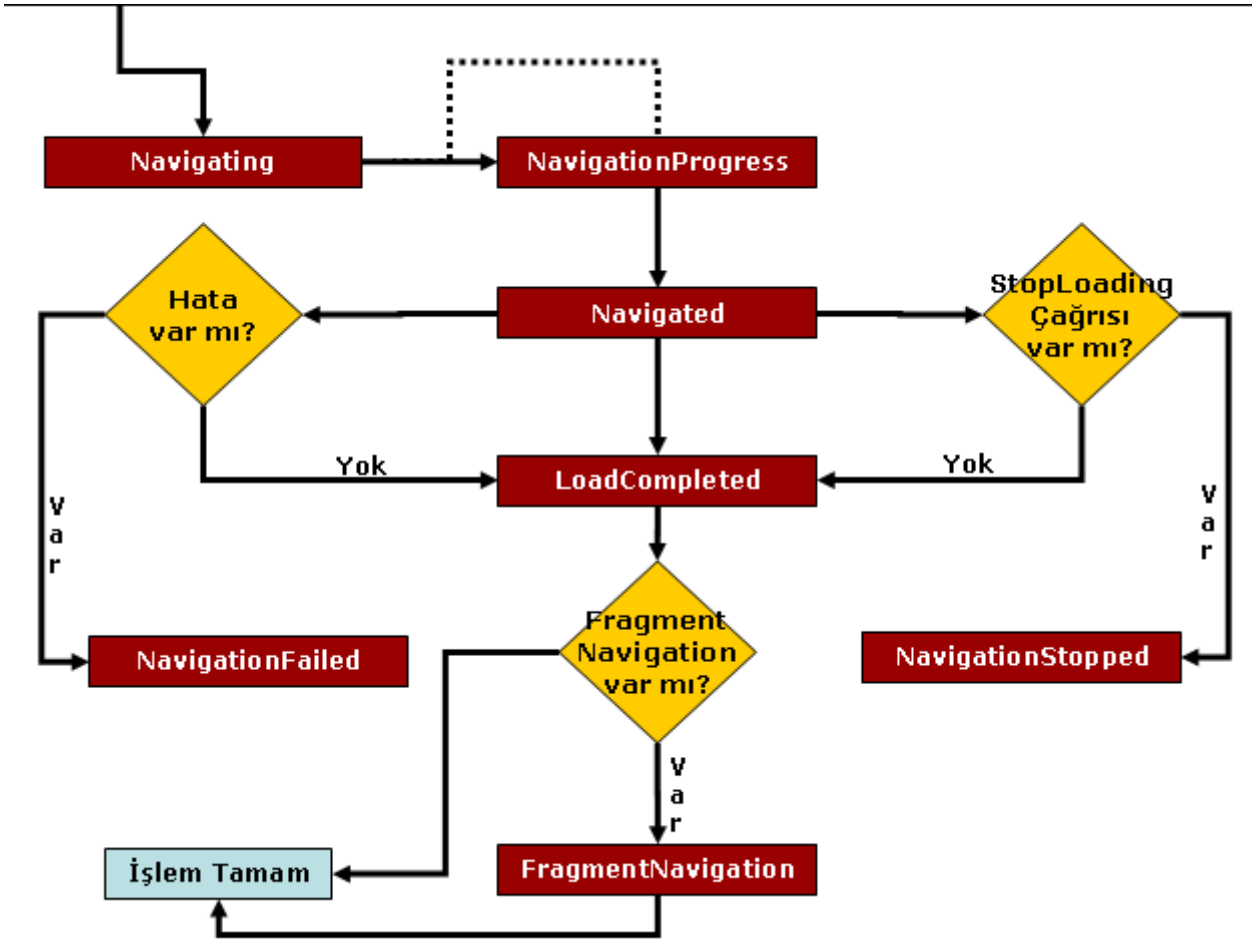
Dikkat edileceği üzere **Frame** içerisinden başka iki sayfaya daha geçilebilmektedir. Diğer taraftan sayfanın kendiside(yani Frame' ide içeren Page) Page2.xaml sayfasına geçiş yapabilmektedir. Bu tip bir durumda **UsesParentJournal** değerinin seçilmesi özellikle **XBAP** uygulamalarında oldukça işe yarayacaktır. Söz konusu kullanımda sayfalar çok fazla parçalandığından zaman zaman takibin zorlaştığıda görülmektedir.

Dolayısıyla **Nested-Page** tekniğinin bir alternatif olarak bilinmesinde ve uygun vakkalar bulunduğunda ele alınmasında yarar vardır.

Yazımızın içerisinde belirttiğimiz gibi navigasyon işlemlerini manuel olarak kod tarafından da yapabiliriz. Burada önemli olan nokta **NavigationService** özelliği ile elde edilecek referans olacaktır.Bununla birlikte bir sayfanın navigasyon süreci içerisinde talep edilmesi halinde neler olduğunun bilinmesinde de yarar vardır. Bir başka deyişle, navigasyon sürecindeki yaşam döngüsünü bilmekte yarar vardır. Temelde süreç bir sayfanın talep edilmesi ile başlar. Bu talep işlemi basit bir **Hyperlink** kontrolü ile olabileceği gibi, **NavigationService** sınıfının **Navigate** metodu ilede olabilir. Sonrasında talep edilen sayfanın yeri tespit edilir. Tahmin edileceği üzere burada bir sorun olması halinde çalışma zamanı istisnası oluşacaktır. Sayfa yeri tespit edildikten sonra bilgileri getirilir. Örneğin talep edilen sayfa bir internet sayfası ise download işlemi gerçekleşir. Bu arada eğer sayfanın indirilmesi sırasında ilişkili kaynaklar var ise bunlarında getirilmeside söz konusudur. Takip eden adımda sayfa için bir **ayrıştırma(parsing)** işlemi uygulanır ve sayfanın **nesne ağacı(Object Tree)** oluşturulur. Bu işlemi takiben sayfaya ait **Initialized** ve **Loaded**olaylarıda sırasıyla tetiklenir. Son aşamada sayfa **Render** işlemine tabi tutulur ve gösterilir. Bu noktada eğer **parçalı navigasyon(Fragment Navigation)** işlemi yapılmışsa ilgili kontrole gidilmesi sağlanır. Aşağıdaki temsili grafik bu işleyişi kısaca özetlemektedir.



Birde bu süreç içerisinde tetiklenen bazı olaylar söz konusudur. Bu olaylar **Application**, **Frame**, **NavigationWindow** yada **NavigationService**' in kendisi tarafından ele alınabilir. Daha çok tercih edilen, söz konusu olayları Application nesnesi seviyesinde ele almaktır. Böylece uygulama içerisinde yer alabilecek tüm sayfalar için ortak bir noktada olayların kontrol edilebilmesi sağlanmış olur. Burada bahsedilen olaylar **Navigating, Navigated, NavigationProgress, LoadCompleted, FragmentNavigation, NavigationStopped** ve **NavigationFailed**' dır. Bu olayların işleyiş sırası aşağıdaki şekildeki gibidir.



İlk olarak süreç bir sayfanın talep edilmesi ile başlar. Sonrasında ise ilgili olaylar tetiklenir. Aşağıdaki tabloda söz konusu olaylar ile ilişkili bilgiler verilmektedir.

Olay(Event)	**Olay Hakkında Kısa Bilgi**
Navigating	Yeni bir navigasyon talebi geldiğinde çalışan olaydır.
Navigated	Navigasyon başlamıştır ve talep edilen sayfa ile ilgili bilgiler gelmektedir. Ancak sayfanın tamamı henüz gelmemiştir.
NavigationProgress	Bu olaya ilişkin metod yazıldığında, sayfanın yüklenmesi tamamlanıncaya kadar bilgilendirme yapılması sağlanabilir. Örneğin sayfanın yüzde olarak kaçının tamamlandığı bilgisi gösterilebilir. Aslında bu olay klasik windows programlamadaki **BackgroundWorker** kontrolünün **ProgressChange** olayınınkine benzer bir görev üstlenmektedir. NavigationProgress olayı, talep edilen sayfa ile ilgili her **1Kb** bilgi geldiğinde tetiklenmektedir.
LoadCompleted	Talep edilen sayfanın **ayrıştırma(Parse)** işlemi tamamlanmıştır. Ancak ilgili sayfanın **Initialized** ve **Loaded** olayları henüz çalışmamıştır.

FragmentNavigation	Eğer **parçalı navigasyon(Fragment Navigation)** söz konusu ise bu olay tetiklenir. Bu olay ilgili kontrole doğru gidilirken çalışmaktadır.
NavigationStopped	**NavigationService** sınıfının static StopLoading metoduna yapılan çağrı sonucu tetiklenen olaydır. Bazı durumlarda uzun süren navigasyon işlemlerinde(örneğin bir web sayfası talebinde) kullanıcı tarafından işlemin durdurulması istenebilir. Böyle bir durumda **StopLoading** metoduna başvurulması halinde bu olay tetiklenmektedir.*(StopLoading' i çalışma zamanında çağırmak son derece kolaydır. Nitekim WPF mimarisinde navigasyon işlemleri asenkron olarak yürütülmektedir.)*
NavigationFailed	Navigasyon işlemi sırasında bir hata oluşursa tetiklenen olaydır. Bu olay içerisinde oluşan hata ile ilişkili detaylı bilgiye ulaşılabilir ve söz konusu bilgiler örneğin loglama amacı ile kayıt altına alınabilir yada kullanıcı farklı bir şekilde yönlendirilerek uygulamanın sağlıklı bir şekilde çalışması sağlanabilir.

Doğal olarak bu olayların tetiklenmesi belirli koşullara bağlıdır. Söz gelimi bir sayfaya doğru navigasyon işleminin başlatılabilmesi için **Hyperlink** kontrolünün **NavigateUri** özelliğinden yada **NavigationService** sınıfının Navigate metodundan yararlanılır. **Navigate** metodunun aşırı yüklenmiş olan versiyonları kullanılarak yukarıda bahsi geçen olaylara veri aktarımıda söz konusu olabilmektedir. Söz gelimi navigasyonun başladığı sürenin gönderilip ilgili olaylarda güncel süre ile arasındaki fark hesap edilerek işlemlerin ne kadar sürdüğü ortalama olarak tespit edilebililir. Yada navigasyon işleminin başlatıldığı sayfanın içerisinde bulunduğu NavigationWindow referansının gönderilmesi sağlanarak, Application seviyesindeki olaylarda ele alınması sağlanabilir.

Burada manuel olarak navigasyon işlemleri adına ismini sıkça duyduğumuz **NavigationService** sınıfının başka güçlü metodlarıda vardır. Söz gelimi navigasyon sürecinde yer alan sayfalar arasında ileri veya geri doğru gidilebilmesini sağlamak amacıyla **GoBack** ve **GoForward** metodlarından yararlanılabilir. Yanlız bu metodlar yardımıyla hareket edilirken herhangibir sebeple hedef sayfa bulunamassa çalışma zamanında **InvalidOperationException** alınır. Bunun önüne geçmek içinse **CanGoBack** ve **CanGoForward** özelliklerinin değerlerine bakılabilir. Bu konu ile ilişkili bir örneği denemenizi şiddetle tavsiye ederim. *(Yazımızı çok fazla uzatacağından bu tarz bir örneği şu an için geliştirmeyeceğiz. Bu konuyu başka bir makalede detaylı bir şekilde incelemeye çalışcağız.)*

XBAP Hakkında

Gelelim sayfalar ile ilgili bir diğer konuya. Yazımızın başında **sayfa bazlı(Page-Based)** uygulamaların geliştirilmesinde kullanılan modellerden bahsederken **XBAP(Xaml Browser Applications)**tekniğinede değinmiştik. XBAP uygulamalarında sayfalar tarayıcı

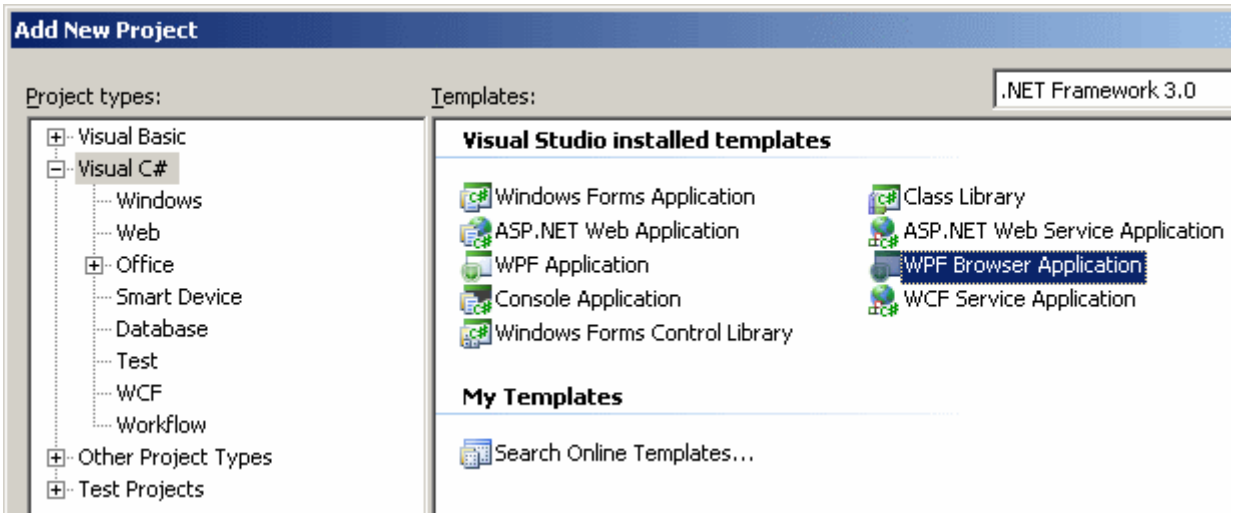
pencerede(browser) gösterilmektedir. Şu an için Internet Exploere 6.0 ve üstü ile Mozilla Firefox' un son sürümünde XBAP uygulamaları çalıştırılabilmektedir. *(En azından makaleyi yazdığım sıralarda bu tarayıcılar ile yapılan testlere göre...)*

Varsayılan olarak XBAP uygulamalarında bazı kısıtlamalar(Restrictions) vardır. Bunlar tahmin edileceği gibi **Code Access Security** kurallarının uygulanmasınında bir sonucudur. Örneğin XBAP uygulamalarında istemci taraflı olaraktan dosyalara yazmak, veritabanlarına bağlanmak, registry işlemleri yapmak yada başka pencereleri tarayıcı penceresi içerisinde popup şekline açmak varsayılan olarak yasaklanmıştır. Bu tarz ihtiyaçların olması durumunda WPF uygulamasını normal bir windows uygulaması olarak yazmak ve **ClickOnce** gibi bir dağıtım modeli ile yaymak daha doğru bir yaklaşımdır. Yinede kısıtlama**(Restriction)** ayarları ile proje özellikleri üzerinden oynanarak istenirse söz konusu işlemlerin yapılması sağlanabilir.

XBAP uygulamaları varsayılan olarak istemci bilgisayardaki tarayıcı penceresine ait tampona**(Cache)** atılırlar. Bir başka deyişle ilk talep edildiklerinde istemci bilgisayara indirilirler**(Download)**. Burada bir yükleme**(install)** işlemi söz konusu değildir. Fakat yine proje ayaları ile oynayaraktan uygulamanın istemciye install edilmesi sağlanabilir. Diğer taraftan install edilmemesinin avantajları vardır. Öyleki, uygulamadaki değişiklikler istemci tarafından otomatik olarak algılanıp son sürümün herhangibir sorgu penceresine gerek kalmadan indirilmesi ve çalıştırılması söz konusudur.

Çok doğal olarak XBAP uygulamalarının çalışabilmesi için indirildikleri bilgisayar sisteminde Microsoft **.Net Framework 3.0** sürümünün yüklü olması gerekmektedir. Eğer yüklü değilse ilgili uygulama çalıştırıldığında .Net Framework 3.0 indirilmeye çalışılacaktır.

Visual Studio 2008 Beta 2 ile bir **XBAP** uygulaması oluşturmak son derece basittir. Tek yapılması gereken proje şablonlarından aşağıdaki resimde görüldüğü gibi **WPF Browser Application** öğesini seçmektir.



Bu işlemin sonucunda içerisinde varsayılan olarak bir sayfa(Page) içeren bir uygulama oluşturulur. Test olması amacıyla uygulamaya ikinci bir sayfa(Page) daha ekleyip bu sayfalar arasında **Hyperlink** ile geçişler yapmaya çalıştığımızı düşünelim. Bu amaçla Page1.xaml ve Page2.xaml içeriklerinin aşağıdaki gibi olduğunu düşünelim.

Page1.xaml;

```
<Page x:Class="MerhabaXBAP.Page1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Ana Sayfa"
WindowTitle="Giriş Sayfası" Background="LightGray" WindowHeight="250"
WindowWidth="250" VerticalAlignment="Top" HorizontalAlignment="Left">
   <Grid>
      <Label Height="33" HorizontalAlignment="Left" Name="label1"
VerticalAlignment="Top" Width="120" FontSize="14" FontWeight="Bold">
         <Hyperlink NavigateUri="Page2.xaml">Sayfa 2</Hyperlink>
      </Label>
   </Grid>
</Page>
```

Page2.xaml;

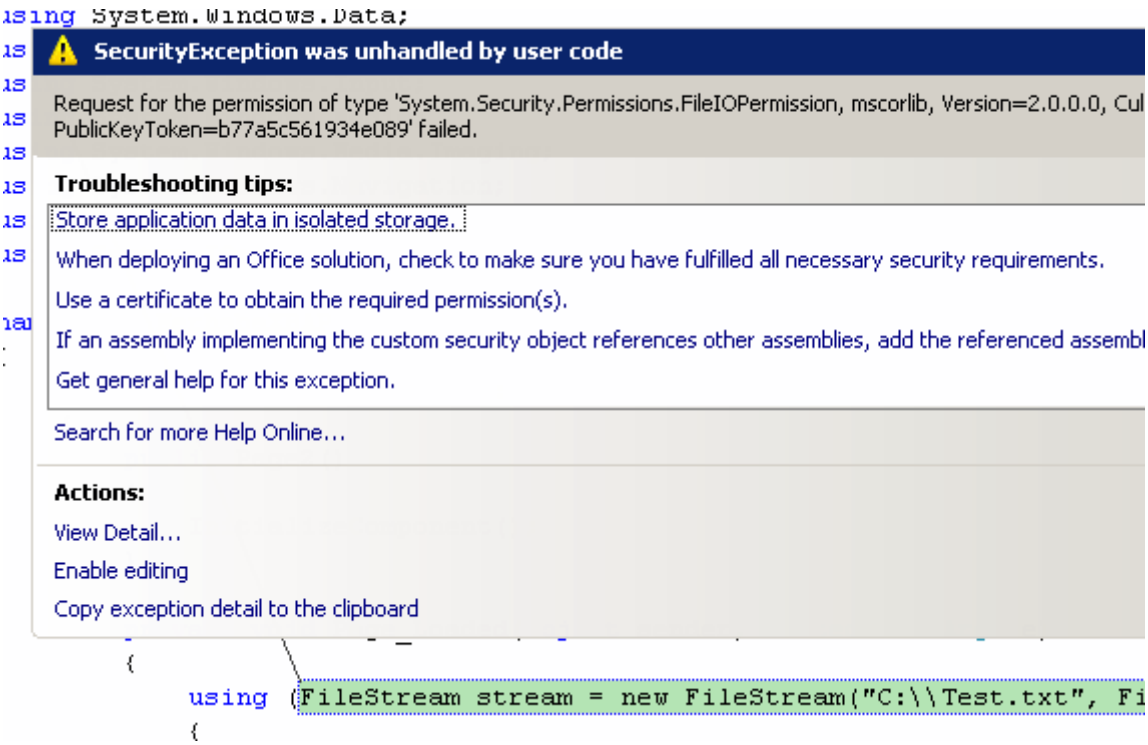
```
<Page x:Class="MerhabaXBAP.Page2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="İkinci Sayfa"
WindowTitle="Sayfa 2"Background="Gold" Loaded="Page_Loaded"
VerticalAlignment="Top" HorizontalAlignment="Left" WindowHeight="250"
WindowWidth="250">
   <Grid>
      <Label Height="35" Name="label1" VerticalAlignment="Top" FontSize="14"
FontWeight="Bold" HorizontalAlignment="Left" Width="120">
         <Hyperlink NavigateUri="Page1.xaml">Sayfa 1</Hyperlink>
      </Label>
   </Grid>
</Page>
```

Uygulama çalıştırıldığında sayfaların **WindowTitle** niteliklerine atanan değerler otomatik olarak **Tab**' da ve tarayıcı penceresinin başlık kısmında görülecektir. Diğer taraftan burada navigasyon kontrolleri oluşturulmayacak, otomatik olarak tarayıcının navigasyon kontrolleri işin içerisine dahil edilecektir. Şimdi güvenlik ile ilişkili kısıtlamaları test etmek amacıyla Page2.xaml sayfasının Loaded olayını yükleyelim ve içerisine aşağıdaki kodları yazdığımızı düşünelim.

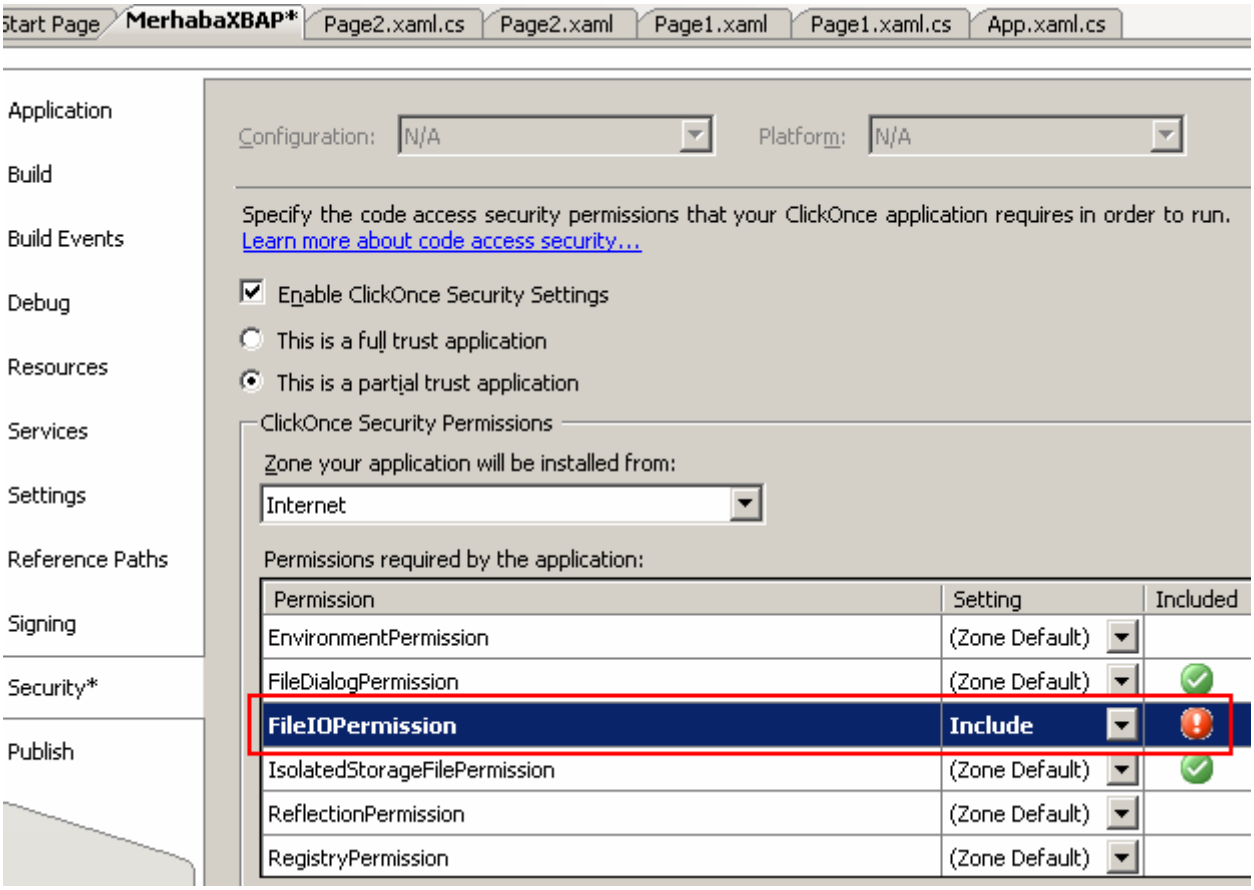
Page2.xaml.cs;

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.IO;

namespace MerhabaXBAP
{
   public partial class Page2 : Page
   {
      public Page2()
      {
         InitializeComponent();
      }

      private void Page_Loaded(object sender, RoutedEventArgs e)
      {
         using (FileStream stream = new FileStream("C:\\Test.txt", FileMode.Append,
FileAccess.Write))
         {
            using (StreamWriter writer = new StreamWriter(stream))
            {
               writer.WriteLine(DateTime.Now.ToString() + " ek bilgi");
            }
         }
      }
   }
}
```

Bu durumda uygulamayı çalıştırıp Page2 isimli sayfaya geçmeye çalıştığımızda **SecurityException** tipinden bir istisna aldığımızı görürüz. Aşağıdaki ekran görüntüsünde bu durum ifade edilmektedir.

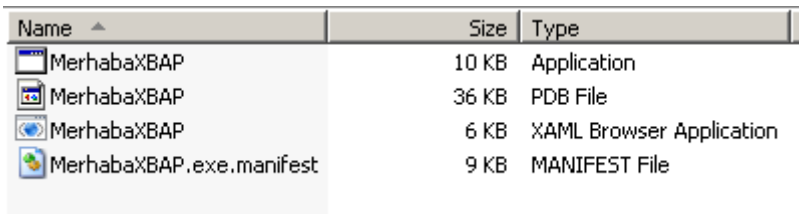


Daha öncedende belirttiğimiz gibi bu istisnanın sebebi varsayılan Code Access Security ayarlarıdır. Hata mesajındanda görüleceği gibi **FileIOPermission** yetkisi olmadığından söz konusu kodlar çalışmayıp istisna vermiştir. Ancak proje özelliklerine girip ilgili izni(Permission) aşağıdaki şekilde görüldüğü gibi vererek dosyaya yazma işleminin gerçekleştirilebilmesi sağlanabilir.



Bu ayarlardan sonra uygulama tekrardan test edilirse Test.txt isimli dosyanın C dizini altında oluşturulduğu ve içerisine bilgi yazıldığı görülür. Buradaki seçeneklerden bir diğeri olan **This is a full trust application** seçilerek, istenirse tüm kısıtlamaların ortadan kaldırılması ve istemci tarafından yürütülebilmesi sağlanabilir. Ancak bu **XBAP** uygulamalarının birincil amacı değildir.

Bir **XBAP** uygulamasını **dağıtırken(Deploye)** uygulamaya ait **exe**, **manifesto** ve **xbap** dosyalarının üçünün birden istemci uygulamaya kopyalanması yeterli bir seçenektir. Elbette söz konusu dosyalar bir intranet sisteminde ortak bir yola(Path) konularaktanda istemcilerin buradan başlatma işlemini gerçekleştirmesi sağlanabilir. Sonuçta varsayılan olarak uygulama, istemci bilgisayarın tarayıcı programının kullandığın ön bellek alanına indirilecektir. Yukarıda geliştirimiş olduğumuz XBAP örneğine baktığımızda **Debug** klasörü altında aşağıdaki şekilde görülen dosyaların oluşturulduğunu görürüz.



Burada yer alan dosyalardan exe uzantılı olan, XBAP uygulamasının derlenmiş halini içermektedir. Bir başka deyişle assembly' ın kendisidir e ILDASM, Metadata, manifesto

gibi bilgileri içermektedir. Ne varki söz konusu exe tek başına çalıştırılabilen bir dosya değildir. Bu sebepten üzerinde çift tıklandığında hiç bir etkileşim olmayacaktır. Asıl çalıştırıcı dosya **XBAP** uzantılı olan **XML** dosyasıdır. Bu dosyanın içeriğinde, uygulamanın giriş noktasını işaret eden bilgi vardır. Açıkçası bu dosya çalıştırıldığında istemci bilgisayardaki varsayılan tarayıcı uygulaması devreye girecek ve exe yürütülmeye başlanacaktır. **XBAP** uzantılı dosyası içerisinde aynı zamanda program yazılırken üretilen dijital imzada yer almaktadır. Bu imza özellikle güncelleme işlemlerinde önem kazanmaktadır. *(Elbette istenirse uygulamanın proje özelliklerine gidilip Signing kısmında başka bir imza üretilip kullanılabilir.)* **Manifesto** dosyasında ise, uygulamanın çalışması için gerekli olan diğer programlara ait bilgiler yer almaktadır. Söz gelimi uygulamanın çalışması için gerekli **.Net Framework** versiyonu, bağlı olan diğer assmebly**(Class Library** gibi)' lar veya uygulamadaki kodların neler yapabileceğini belirten izin**(Permission)** yetkileri gibi bilgiler yer almaktadır. Ne varki dağıtım işlemi özellikle ilgili uygulamalarda güncellemeler yapıldığı takdirde tekrarlanmak zorunda olabilir. Bu vakka **ClickOnce** ile ilgilide olduğundan ve makalemizin konusunu aştığından bu yazı dizimizde ele alınmayacaktır. Umuyorumki ilerleyen zamanlarda ele alabiliriz.

Bu makalemizde WPF(Windows Presentation Foundation) ile birlikte hayatımıza giren yeni kavramlardan birisi olan sayfa-tabanlı(Page-based) uygulamaları tanımaya çalıştık. Konunun detayları için sizlere 1000 sayfalık **Pro WPF: Windows Presentation Foundation in .NET 3.0** kitabı tavsiye ederim.

Böylece geldik uzun bir makalemizin daha sonuna. Bu cümleye kadar sabırla okuduğunu için son teşekkür ederim. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

WPF Temel Animasyon İşlemleri - 2 (2007-10-01T19:35:00)

wpf,

Bir önceki makalemizde **Windows Prensetation Foundation(WPF)** uygulamalarında animasyon işlemlerinin temel animasyon tipleri**(Basic Animation Types)** yardımıyla nasıl gerçekleştirilebileceğini incelemeye başlamıştık. Bu makalemizde animasyon işlemleri üzerindeki yönetimin biraz daha fazla olmasını sağlamak için farklı teknikleri göz önüne alıyor olacağız. Her zaman olduğu gibi konuyu daha iyi kavrayabilmek adına örnekler üzerinden ilerlemekte fayda olduğu kanısındayım. Dilerseniz hiç vakit kaybetmeden ilk örneğimiz ile başalayım. Hatırlanacağı üzere, WPF uygulamalarında bileşenlerin rotasyon işlemleri için **RotateTransform**, **ScaleTransform** ve benzeri tiplerin kullanıldığını görmüştük. Bu tip transformasyon işlemlerini animasyon tipleri ile birlikte kullanmak isteyebiliriz. Söz gelimi bir Button kontrolünün 3 saniye içerisinde kendi ekseninde 360 derece dönmesini ve bu sırada boyutlarınında 2 katına çıkarak tekrardan eski haline dönmesini istediğimizi düşünelim. Bu tip bir animasyon işleminin Buttton kontrolünün üzerine mouse ile gelindiğinde başlatılmasını ve 3 saniyelik zaman dilimi içerisinde mouse ile kontrolün terk edilmesi halinde de durmasınıda sağlayabiliriz. Burada temel animasyon

tiplerinden olan **DoubleAnimation** oldukça işe yarayacaktır. Herşeyden önce animasyon tipinin RotateTransform ve ScaleTransform tiplerindeki uygun özellikleri kontrol edecek şekilde ayarlanması gerekir. Bu senaryoyu gerçekleştirmek için **XAML** içeriği aşağıda verilen bir pencere(Window) oluşturulması yeterli olacaktır.

XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.RotateAnimasyon"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Rotate ve Sacle
Animasyon" Height="220" Width="289">
   <Grid>
      <Button Name="btnMerhaba" Width="75" Height="40" Content="Merhaba"
Background="Gold" Foreground="Black" FontSize="14" FontWeight="Bold">
         <Button.LayoutTransform>
            <TransformGroup>
               <RotateTransform x:Name="RTrans" Angle="0"/>
               <ScaleTransform x:Name="STrans" CenterX="0" CenterY="0"
ScaleX="1" ScaleY="1"/>
            </TransformGroup>
         </Button.LayoutTransform>
         <Button.Triggers>
            <EventTrigger RoutedEvent="Button.MouseEnter">
               <EventTrigger.Actions>
                  <BeginStoryboard Name="strBrd">
                     <Storyboard>
                        <DoubleAnimation Storyboard.TargetName="RTrans"
Storyboard.TargetProperty="Angle" To="360" Duration="0:0:3"/>
                        <DoubleAnimation Storyboard.TargetName="STrans"
Storyboard.TargetProperty="ScaleX" To="2.5" Duration="0:0:1.5"/>
                        <DoubleAnimation Storyboard.TargetName="STrans"
Storyboard.TargetProperty="ScaleY" To="2.5" Duration="0:0:1.5"/>
                        <DoubleAnimation Storyboard.TargetName="STrans"
```

```
Storyboard.TargetProperty="ScaleX" To="1" Duration="0:0:3"/>
                    <DoubleAnimation Storyboard.TargetName="STrans"
Storyboard.TargetProperty="ScaleY" To="1" Duration="0:0:3"/>
                  </Storyboard>
                </BeginStoryboard>
              </EventTrigger.Actions>
            </EventTrigger>
            <EventTrigger RoutedEvent="Button.MouseLeave">
              <EventTrigger.Actions>
                <StopStoryboard BeginStoryboardName="strBrd"/>
              </EventTrigger.Actions>
            </EventTrigger>
          </Button.Triggers>
        </Button>
    </Grid>
</Window>
```

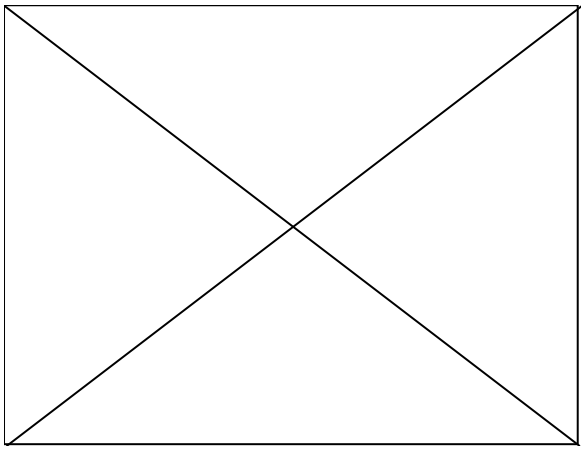
Burada dikkat edilmesi gereken noktalardan birisi, **RotateTransform** ve **ScaleTransform** elementlerinin **DoubleAnimation** tiplerinde kullanılabilmesi için **x:Name** nitelikleri ile isimlendirilmiş olmalarıdır. Diğer taraftan **RotateTransform**, Button bileşenin verilen açı kadar döndürülmesini sağlamaktadır. Buna göre ilk **DoubleAnimation** elementi 3 saniyelik zaman çizgisi(**TimeLine**) içerisinde açının değerinin 360 derece olmasını sağlamaktadır. Sonrasında gelen dört DoubleAnimation tipi ise ilk 1.5 saniyelik zaman dilimi içerisinde **ScaleTransform** elementinin **ScaleX** ve **ScaleY**değerlerini önce iki buçuk katına çıkarmakta sonrasında ise tekrar eski haline döndürmektedir. Söz konusu animasyon işlemleri **MouseEnter** olayı tetiklendiğinde başlatılmaktadır. Diğer taraftan **MouseLeave** olayı tetiklendiğinde, bir başka deyişle mouse ile Button kontrolü üzerinden çıkıldığında animasyon işlemi durdurulmaktadır. Durdurma işlemi için**StopStroyboard** elementi kullanılır. Bu elementin hangi animasyon işlemini durduracağını belirtmek için **BeginStroyboardName** niteliğine(attribute) ilgili değerin atanması gerekir. Örneğimizde bu değer, **BeginStroyboard** elementindeki **Name** niteliğinin değeri olan strBrd' dir. Buna göre istersek birden fazla Storyboard' un olduğu bir senaryoda Name niteliğinden yararlanarak hangisinin durdurulacağını, duraksatılacağını(**Pause**) yada çıkartılacağını(**Remove**) belirleyebiliriz.

NOT: *WPF(Windows Presentation Foundation) mimarisinde farklı Storyboard tipleri vardır.*

- ***BeginStoryboard*** *: Storyboard' un başlatılmasını sağlar.*
- ***PauseStoryboard*** *: Storyboard duraksatılır.*
- ***ResumeStoryboard*** *: Duraksatılan storyboard kaldığı yerden devam eder.*
- ***RemoveStoryboard*** *: Storyboard kaldırılır.*

- ***SetStoryboardSpeedRatio*** *: Storyboard içerisinde animasyonların hız oranı değiştirilebilir.*
- ***SkipStoryboardToFill*** *: Eğer Stroyboard içerisinde tanımlanmış bir Fill periyodu varsa animasyonun otomatik olarak buraya atlaması sağlanır.*

Örneği yürüttüğümüzde çalışma zamanında**(Run-time)** aşağıdaki Flash görselinde yer alan etkileri izleyebiliriz. *(**Flash dosyasının boyutu 264 Kb olduğundan yüklenmesi zaman alabilir**. Dosyanın boyutunun küçük olması için kalitesi düşürülmüştür. Bu nedenle kitap animasyon hareketi sırasında iz bırakmaktadır. Gerçek uygulamada elbetteki böyle bir iz yoktur.)*



Benzer işlemleri kod tarafında yapmakta oldukça kolaydır. Bunun için aşağıdaki kodlara ve XAML içeriğine sahip pencereyi ele alabiliriz.

XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.KodlaRotateAnimasyon"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Kodla Rotate
Animasyon" Height="250" Width="325">
  <Grid>
    <Button Name="btnMerhaba" Width="75" Height="40" Content="Merhaba"
Background="Black" Foreground="Gold" FontSize="14" FontWeight="Bold">
      <Button.LayoutTransform>
        <TransformGroup>
          <RotateTransform x:Name="RTrans" Angle="0"/>
          <ScaleTransform x:Name="STrans" CenterX="0"
CenterY="0" ScaleX="1" ScaleY="1"/>
        </TransformGroup>
      </Button.LayoutTransform>
    </Button>
  </Grid>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Media.Animation; // Animasyon tiplerinin yer aldığı isim alanıdır(Namespace)

namespace AnimasyonIslemleri
{
   public partial class KodlaRotateAnimasyon : Window
   {
      private void AnimasyonuOlustur()
      {
         // Storyboard oluşturulur
         Storyboard strBrd = new Storyboard();

         // Birinci DoubleAnimation tipi oluşturulur.
         // İlk parametre To değeridir ve burada 360 dereceyi işaret etmektedir. İkinci parametre ise Duration değeridir.
         DoubleAnimation dblAni1 = new DoubleAnimation(360, new Duration(TimeSpan.FromSeconds(3)));
         // Animasyon tipinin RTrans adına sahip elementin AngleProperty özelliğine uygulanacağı belirtilir.
         Storyboard.SetTargetName(dblAni1, "RTrans");
         Storyboard.SetTargetProperty(dblAni1, new PropertyPath(RotateTransform.AngleProperty));
         // Animasyon tipi Storyboard' a eklenir.
         strBrd.Children.Add(dblAni1);

         // İkinci DoubleAnimation tipi oluşturulur.
         // İlk parametre To değeridir ve burada 2.5 katını işaret etmektedir. İkinci parametre ise Duration değeridir. Buna göre zaman çizgisinin ilk 1.5 saniyesi içerisinde Button bileşeninin ScaleX değeri 2.5 kat artmaktadır.
         DoubleAnimation dblAni2 = new DoubleAnimation(2.5, new Duration(TimeSpan.FromSeconds(1.5)));
         // Animayonun STrans isimli ScaleTransform elementi içerisindeki ScaleX
```

özelliğine uygulanacağı belirtilir.

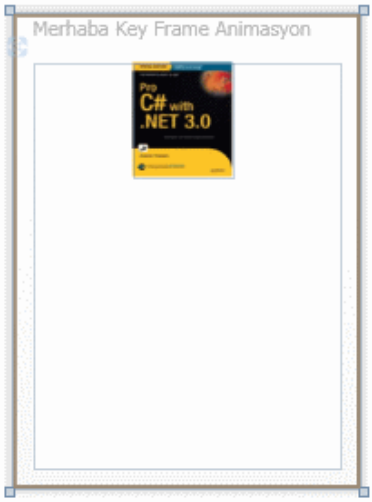
```
        Storyboard.SetTargetName(dblAni2, "STrans");
        Storyboard.SetTargetProperty(dblAni2, new
PropertyPath(ScaleTransform.ScaleXProperty));
        // Animasyon tipi Storyboard' a eklenir.
        strBrd.Children.Add(dblAni2);

        // Üçüncü DoubleAnimation tipi oluşturulur.
        // İlk parametre To değeridir ve burada 2.5 katını işaret etmektedir. İkinci parametre
ise Duration değeridir. Buna göre zaman çizgisinin ilk 1.5 saniyesi içerisinde Button
bileşeninin ScaleY değeri 2.5 kat artmaktadır.
        DoubleAnimation dblAni3 = new DoubleAnimation(2.5, new
Duration(TimeSpan.FromSeconds(1.5)));
        // Animayonun STrans isimli ScaleTransform elementi içerisindeki ScaleY
özelliğine uygulanacağı belirtilir.
        Storyboard.SetTargetName(dblAni3, "STrans");
        Storyboard.SetTargetProperty(dblAni3, new
PropertyPath(ScaleTransform.ScaleYProperty));
        // Animasyon tipi Storyboard' a eklenir.
        strBrd.Children.Add(dblAni3);

        // Zaman çizgisinin 3ncü saniyesine gelindiğinde Button kontrolünün ScaleX değeri
ilk haline döner.
        DoubleAnimation dblAni4 = new DoubleAnimation(1, new
Duration(TimeSpan.FromSeconds(3)));
        Storyboard.SetTargetName(dblAni4, "STrans");
        Storyboard.SetTargetProperty(dblAni4, new
PropertyPath(ScaleTransform.ScaleXProperty));
        // Animasyon tipi Storyboard' a eklenir.
        strBrd.Children.Add(dblAni4);

        // Zaman çizgisinin 3ncü saniyesine gelindiğinde Button kontrolünün ScaleY değeri
ilk haline döner.
        DoubleAnimation dblAni5 = new DoubleAnimation(1, new
Duration(TimeSpan.FromSeconds(3)));
        Storyboard.SetTargetName(dblAni5, "STrans");
        Storyboard.SetTargetProperty(dblAni5, new
PropertyPath(ScaleTransform.ScaleYProperty));
        // Animasyon tipi Storyboard' a eklenir.
        strBrd.Children.Add(dblAni5);

        // Mouse ile Button alanı üzerinde gelindiğinde animasyon başlatılır.
        btnMerhaba.MouseEnter += delegate(object sender, MouseEventArgs e)
                                 {
                                    /* İkinci parametre Storyboard' un kontrol
```

```
edilebileceğini gösterir. Bir başka deyişle programatik olarak animasyonun durdurulması,
duraksatılması ve benzeri işlemlerin yapılabilmesi sağlanır. */
                                        strBrd.Begin(this,true);
                                    };

        btnMerhaba.MouseLeave += delegate(object sender, MouseEventArgs e)
                                    {
                                        // Storyboard' un başlattığı animasyon çıkartılır.
Dolayısıyla Button açısal konum ve büyüklük olarak ilk değerlerine döner.
                                        strBrd.Stop(this);
                                    };
      }

      public KodlaRotateAnimasyon()
      {
         InitializeComponent();
         AnimasyonuOlustur();
      }
   }
}
```

Kod yapısında dikkat edilmesi gereken önemli noktalardan birisi, **MouseLeave** olayının gerçekleşmesi halinde var olan amimasyonun durdurulması**(Stop)** işleminin **Storyboard** nesne örneğine ait Stop metodu ile gerçekleştirimiş olmasıdır. Lakin burada Stop metodunun işe yarayabilmesi için animasyon Begin metodu ile başlatılırken ikinci parametrenin **true** olarak verilmesi şarttır. Bu parametrenin true olarak verilmesi halinde yanlızca **Stop** metodu değil, **Pause**, **Resume**, **Remove** gibi yönetsel fonksiyonelliklerinde kullanılabilir hale gelmesi sağlanmaktadır.

Animasyon işlemlerinde zaman çizgileri (**Timeline**) içerisinde yer alan **KeyFrame**' lerin yönetilmesi ve bu sayede daha güçlü hareketlerin sağlanabilmeside mümkündür. Bu amaçla WPF yapısı içerisine 3 temel **KeyFrame** tipi konulmuştur.
Bunlar **LinearDoubleKeyFrame**, **DiscreteDoubleKeyFrame**, **SplineDoubleKeyFrame** ti pleridir. Söz konusu tiplerin ne şekilde kullanıldıklarını örnekler ile incelediğimizde konuyu daha rahat kavrayabiliriz. Sıradaki örneğimizde LinearDoubleKeyFrame tipini kullanıyor olacağız. Bu amaçla ilk olarak aşağıdaki **XAML** çıktısını içeren bir pencere (Window) hazırladığımızı düşünelim.

XAML içeriği;

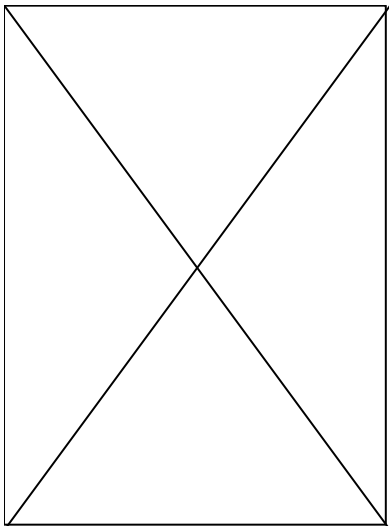



```
<Window x:Class="AnimasyonIslemleri.MerhabaKeyFrameAnimasyon"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Merhaba Key Frame
Animasyon" Height="300" Width="300">
   <Canvas>
      <Rectangle Name="Dortgen" Canvas.Top="0" Canvas.Left ="100" Height="125"
Width="93">
         <Rectangle.Fill>
            <ImageBrush ImageSource="Kitap.jpg"/>
         </Rectangle.Fill>
         <Rectangle.Triggers>
            <EventTrigger RoutedEvent="Rectangle.MouseEnter">
               <EventTrigger.Actions>
                  <BeginStoryboard>
                     <Storyboard RepeatBehavior="Forever" AutoReverse="True">
                        <DoubleAnimationUsingKeyFrames Storyboard.TargetName="Do
rtgen" Storyboard.TargetProperty="(Canvas.Top)">
                           <LinearDoubleKeyFrame Value="160" KeyTime="0:0:2"/>
                           <LinearDoubleKeyFrame Value="0" KeyTime="0:0:4"/>
                           <LinearDoubleKeyFrame Value="100" KeyTime="0:0:6"/>
                           <LinearDoubleKeyFrame Value="40" KeyTime="0:0:8"/>
                           <LinearDoubleKeyFrame Value="80" KeyTime="0:0:10"/>
                        </DoubleAnimationUsingKeyFrames>
                     </Storyboard>
                  </BeginStoryboard>
               </EventTrigger.Actions>
            </EventTrigger>
         </Rectangle.Triggers>
      </Rectangle>
```

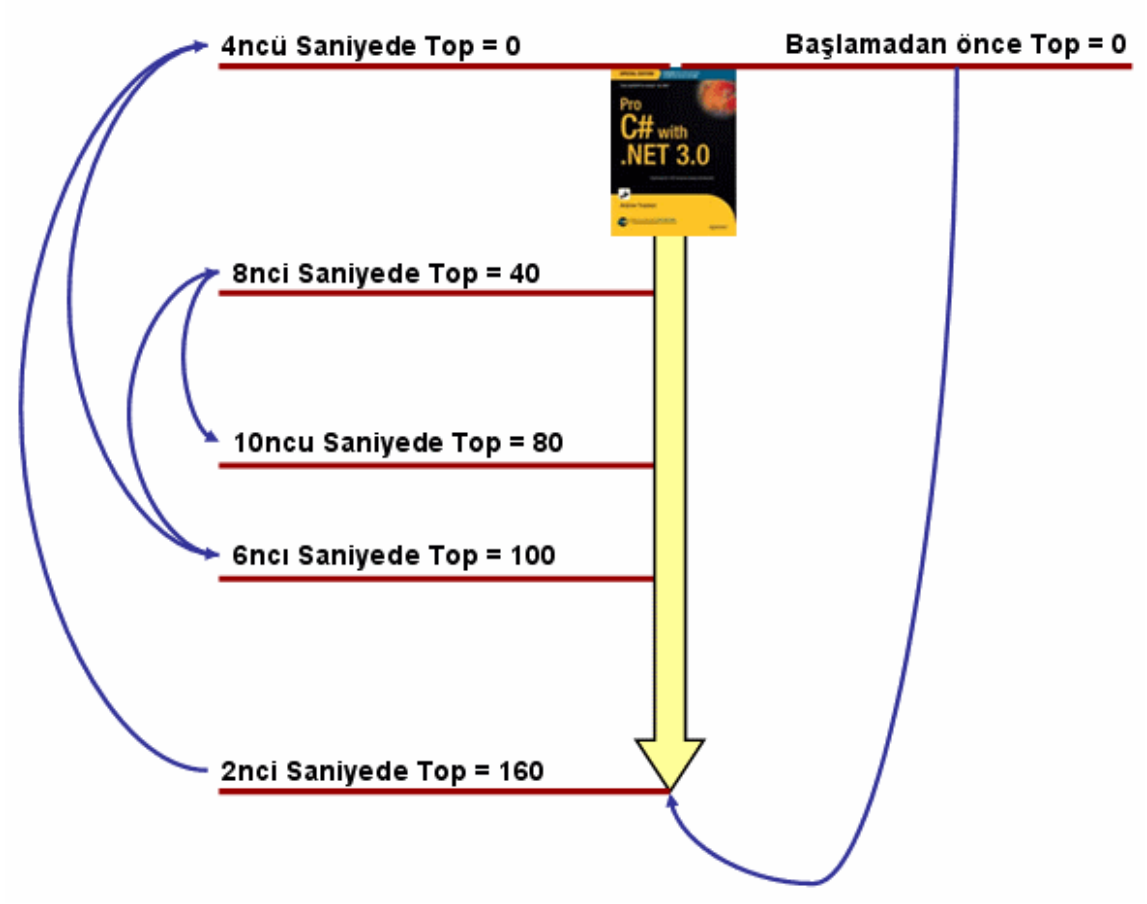
```
    </Canvas>
</Window>
```

Dikkat edileceği üzere örnekte kullanılan **LienarDoubleKeyFrame** elementleri **DoubleAnimationUsingKeyFrames** elementi içerisinde yer almaktadır. Buna göre **KeyFrame** tiplerinin **TipAdıUsingKeyFrames** notasyonu ile tanımlanmış tipler içerisinde olması gerektiği ortadadır. Peki burada yer alan LinearDoubleKeyFrame elementleri ne işe yaramaktadır?

Value özelliği ile tahmin edileceği gibi **DoubleAnimationUsingKeyFrames** elementinde belirtilen kontrol ve ilgili özelliğin yeni değeri belirlenir. **KeyTime** niteliği ilede, özelliğin zaman çizelgesi içerisinde belirtilen anda değerini alması sağlanır. Bir başka deyişle örnekte kullanılan Dortgen isimli **Rectangle** elementinin pencerenin(**Window**) üst kısmından olan uzaklığı, zaman çizelgesi**(timeline)** içerisinde Frame noktalarında farklı değerlere set edilmektedir. Aslında durumu daha iyi kavramak için pencerenin çalışma zamanındaki çıktısına bakmakta fayda vardır. Aşağıdaki Flash animasyonunda bu durum işaret edilmektedir.*(**Flash dosyasının boyutu 322 Kb olduğundan yüklenmesi zaman alabilir.** Dosyanın boyutunun küçük olması için kalitesi düşürülmüştür. Bu nedenle kitap animasyon hareketi sırasında iz bırakmaktadır. Gerçek uygulamada elbetteki böyle bir iz yoktur.)*



Görüldüğü gibi dörtgenimiz zaman çizelgesinin 2nci saniyesinde pencerenin üst kenarının 160 piksel, 4ncü saniyesinde tekrardan 0 piksel, 6ncı saniyesinde 100 piksel, 8nci saniyesinde 40 piksel, 10ncu saniyesinde ise 80 piksel uzağına gelmektedir. Sonrasında ise bu animasyon hareketini tersine doğru yapmaktadır. Bunun için **Storyboard** elementinin **AutoReverse** özelliğine **true** değerinin atanması yeterli olmaktadır. Durumu aşağıdaki şekil ile daha net bir şekilde anlayabiliriz.



Aynı örneği kod yardımıylada gerçekleştirebiliriz. Bunun için aşağıdaki XAML içeriğine ve kodlarına sahip bir pencere(Window) geliştirmemiz yeterli olacaktır.

XAML içeriği;

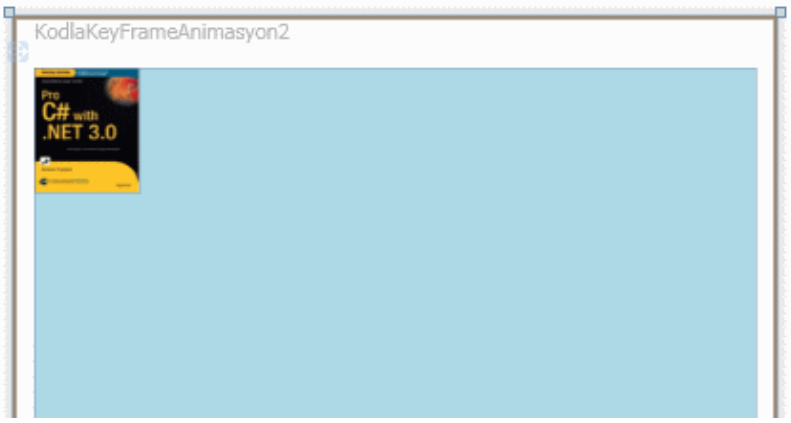
```
<Window x:Class="AnimasyonIslemleri.KodlaMerhabaKeyFrameAnimasyon"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Kodla Merhaba
KeyFrame Animasyon" Height="241" Width="218">
   <Canvas x:Name="Bolge">
      <Rectangle Name="Dortgen" Canvas.Top="0" Canvas.Left="65" Height="65"
Width="55">
         <Rectangle.Fill>
            <ImageBrush ImageSource="Kitap.jpg"/>
         </Rectangle.Fill>
      </Rectangle>
   </Canvas>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Media.Animation;

namespace AnimasyonIslemleri
{
   public partial class KodlaMerhabaKeyFrameAnimasyon : Window
   {
      private void AnimasyonuHazirla()
      {
         // DoubleAnimationKeyFrames nesnesi örneklenir.
         DoubleAnimationUsingKeyFrames dblKeyFrames = new
DoubleAnimationUsingKeyFrames();
         // Animasyon tipi, kontrol ve özelliğini ilişkilendirecek olan StoryBoard oluşturulur.
         Storyboard strBrd = new Storyboard();
         // Animasyon sona erdiğinde aynı rotadan geriye doğru dönmesi için AutoReverse
özelliğine true değeri atanır
         strBrd.AutoReverse = true;
         // Animasyonun sürekli olması için RepeatBehavior özelliğine Forever değeri
atanır.
         strBrd.RepeatBehavior = RepeatBehavior.Forever;
         // Hedef kontrol olarak Rectangle tipinden olan Dortgen seçilir
         Storyboard.SetTargetName(dblKeyFrames, "Dortgen");
         // Animasyon değerlerinin uygulanacağı özellikl olarak Canvas' ın Top özelliği
seçilir.
         Storyboard.SetTargetProperty(dblKeyFrames, new
PropertyPath(Canvas.TopProperty));
         // KeyFrame değerleri belirlenir.
         // İlk 1 saniyede Canvas dikeylemesine 120 pikselinci uzaklığa gelecektir
         dblKeyFrames.KeyFrames.Add(new LinearDoubleKeyFrame(120,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 1))));
         // 2nci saniyede Canvas dikeylemesine 0 pikselinci uzaklığa gelecektir
         dblKeyFrames.KeyFrames.Add(new LinearDoubleKeyFrame(0,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 2))));
         // 3ncü saniyede Canvas dikeylemesine 100 pikselinci uzaklığa gelecektir
         dblKeyFrames.KeyFrames.Add(new LinearDoubleKeyFrame(100,
```

```
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 3)));
        // 4ncü saniyede Canvas dikeylemesine 50 pikselinci uzaklığa gelecektir
        dblKeyFrames.KeyFrames.Add(new LinearDoubleKeyFrame(50,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 4)));
        // 5nci saniyede Canvas dikeylemesine 75 pikselinci uzaklığa gelecektir.
        dblKeyFrames.KeyFrames.Add(new LinearDoubleKeyFrame(75,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 5)));
        // 6nci saniyede Canvas dikeylemesine 25 pikselinci uzaklığa gelecektir.
        dblKeyFrames.KeyFrames.Add(new LinearDoubleKeyFrame(25,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 6)));
        // Animasyon tipi StoryBoard nesnesine eklenir
        strBrd.Children.Add(dblKeyFrames);
        // Animasyonun, Mouse ile Dortgen in üzerine gelindiğinde başlaması sağlanır.
        Dortgen.MouseEnter += delegate(object sender, MouseEventArgs e)
                              {
                                 strBrd.Begin(this);
                              };
    }
    public KodlaMerhabaKeyFrameAnimasyon()
    {
        InitializeComponent();
        AnimasyonuHazirla();
    }
  }
}
```

Kod tarafında dikkat edilmesi gereken noktalardan birisi **LinearDoubleKeyFrame** nesne örneklerinin **DoubleAnimationUsingKeyFrames** tipinin KeyFrames koleksiyonunda tutuluyor olmasıdır. **KeyFrames**özelliği üzerinden çağırılan **Add** metodu ile ilgili nesnelerin koleksiyona eklenmesi sağlanır. Oldukça eğlenceli değil mi? Gelin LinearDoubleKeyKeyFrame nesnelerinin kullanıldığı başka bir örnek ile devam edelim. Bu yeni örnekte bir öncekinden farklı olarak Canvas' ın **Top** ve **Left** özelliklerini rastegele oranlarda değiştiriyor olacağız. İşte örnek XAML içeriği ve kodlarımız.



XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.KodlaKeyFrameAnimasyon2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="KodlaKeyFrameAnimasyon2" Height="400" Width="400"
Background="LightBlue">
   <Canvas x:Name="alan">
      <Rectangle Name="dortgenim" Canvas.Top="0" Canvas.Left="0" Height="65"
Width="55">
         <Rectangle.Fill>
            <ImageBrush ImageSource="Kitap.jpg"/>
         </Rectangle.Fill>
      </Rectangle>
   </Canvas>
</Window>
```

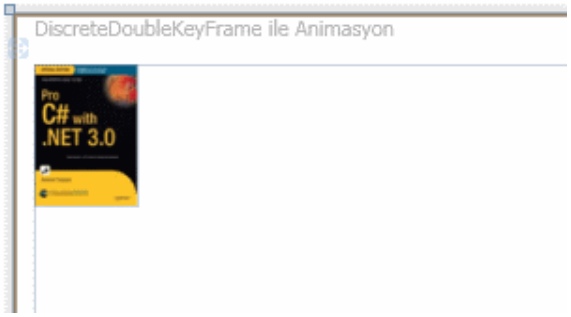
Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Media.Animation;

namespace AnimasyonIslemleri
{
   public partial class KodlaKeyFrameAnimasyon2 : Window
   {
      Random rnd = new Random();
      Storyboard strBrd = new Storyboard();

      public void AnimasyonuYurut()
      {
         double randomT = rnd.Next(1, 250);
         double randomL = rnd.Next(1, 250);
         Title=String.Format("İlk nokta {0}:{1} İkinci Nokta
{2}:{3}",randomT.ToString(),randomL.ToString(),(randomT - 25).ToString(),(randomL -
```

```
25).ToString());

            strBrd.AutoReverse = false;
            strBrd.RepeatBehavior = new RepeatBehavior(1);

            DoubleAnimationUsingKeyFrames keyFrm1 = new
DoubleAnimationUsingKeyFrames();
            keyFrm1.KeyFrames.Add(new LinearDoubleKeyFrame(randomT,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 3))));
            keyFrm1.KeyFrames.Add(new LinearDoubleKeyFrame(randomT-25,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 5))));

            Storyboard.SetTargetName(keyFrm1, "dortgenim");
            Storyboard.SetTargetProperty(keyFrm1, new
PropertyPath(Canvas.TopProperty));
            strBrd.Children.Add(keyFrm1);

            DoubleAnimationUsingKeyFrames keyFrm2 = new
DoubleAnimationUsingKeyFrames();
            keyFrm2.KeyFrames.Add(new LinearDoubleKeyFrame(randomL,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 3))));
            keyFrm2.KeyFrames.Add(new LinearDoubleKeyFrame(randomL-25,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 5))));

            Storyboard.SetTargetName(keyFrm2, "dortgenim");
            Storyboard.SetTargetProperty(keyFrm2, new
PropertyPath(Canvas.LeftProperty));
            strBrd.Children.Add(keyFrm2);

            strBrd.Begin(this);
        }

        void dortgenim_MouseEnter(object sender, MouseEventArgs e)
        {
            AnimasyonuYurut();
        }

        public KodlaKeyFrameAnimasyon2()
        {
            InitializeComponent();
            dortgenim.MouseEnter += new MouseEventHandler(dortgenim_MouseEnter);
        }
    }
}
```

Tahmin edileceği üzere dortgenin **Top** ve **Left** özelliklerini animayon içerisinde kullanmak istediğimizden iki farklı **DoubleAnimationUsingKeyFrames** nesne örneği kullanılması gerekmektedir. Bu nesne örnekleride kendi içlerinde farklı **LinearDoubleKeyFrame** nesnelerine sahiptir. Örneğin çalışma zamanındaki(run-time) çıktısı aşağıdaki Flash görselindeki gibi olacaktır.***(Flash dosyasının boyutu 454 Kb olduğundan yüklenmesi zaman alabilir.*** *Dosyanın boyutunun küçük olması için kalitesi düşürülmüştür. Bu nedenle kitap animasyon hareketi sırasında iz bırakmaktadır. Gerçek uygulamada elbetteki böyle bir iz yoktur.)*

LinearDoubleKeyFrame tipi söz konusu değerlere ulaşılırken akıcı bir görselliğin olmasını sağlar. Bunun dışında **DiscreteDoubleKeyFrame** tipi kullanılaraktan nesnenin belirtilen animasyonda bir sonraki değerine sıçrayarak geçmesi sağlanabilir. Aslında bu durumu anlamanın en iyi yolu örnek bir senaryo üzerinde ilerlemektir. Bu amaçla aşağıdaki **XAML** çıktısına sahip bir **pencere(Window)** tasarladığımızı düşünelim.



XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.DiscreteKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="DiscreteDoubleKeyFrame ile Animasyon" Height="400" Width="400">
   <Canvas>
      <Rectangle Name="Dortgen" Canvas.Top="0" Canvas.Left="0" Height="75"
Width="55">
         <Rectangle.Fill>
            <ImageBrush ImageSource="Kitap.jpg"/>
         </Rectangle.Fill>
         <Rectangle.Triggers>
            <EventTrigger RoutedEvent="Window.Loaded">
               <EventTrigger.Actions>
                  <BeginStoryboard>
                     <Storyboard AutoReverse="True" RepeatBehavior="Forever">
                        <DoubleAnimationUsingKeyFrames
Storyboard.TargetName="Dortgen" Storyboard.TargetProperty="(Canvas.Top)">
                           <DiscreteDoubleKeyFrame KeyTime="0:0:1" Value="150"/>
                           <DiscreteDoubleKeyFrame KeyTime="0:0:2" Value="30"/>
```

```
                        <DiscreteDoubleKeyFrame KeyTime="0:0:3" Value="120"/>
                        <DiscreteDoubleKeyFrame KeyTime="0:0:4" Value="60"/>
                        <DiscreteDoubleKeyFrame KeyTime="0:0:5" Value="90"/>
                        <DiscreteDoubleKeyFrame KeyTime="0:0:6" Value="75"/>
                    </DoubleAnimationUsingKeyFrames>
                    <DoubleAnimationUsingKeyFrames
Storyboard.TargetName="Dortgen" Storyboard.TargetProperty="(Canvas.Left)">
                        <DiscreteDoubleKeyFrame KeyTime="0:0:1.5" Value="150"/>
                        <DiscreteDoubleKeyFrame KeyTime="0:0:2.5" Value="30"/>
                        <DiscreteDoubleKeyFrame KeyTime="0:0:3.5" Value="120"/>
                        <DiscreteDoubleKeyFrame KeyTime="0:0:4.5" Value="60"/>
                        <DiscreteDoubleKeyFrame KeyTime="0:0:5.5" Value="90"/>
                        <DiscreteDoubleKeyFrame KeyTime="0:0:6.5" Value="75"/>
                    </DoubleAnimationUsingKeyFrames>
                </Storyboard>
            </BeginStoryboard>
        </EventTrigger.Actions>
    </EventTrigger>
</Rectangle.Triggers>
</Rectangle>
</Canvas>
</Window>
```

Örnekte iki adet **DoubleAnimationUsingKeyFrames** elementi kullanılmıştır. Bunlardan birisi **Canvas** tipinin **Top** özelliği üzerinde diğeri ise **Left** özelliği üzerinden animasyon işlemlerinin yürütülmesini olanaklı kılmaktadır. Her biri kendi içerisinde birden fazla **DiscreteDoubleKeyFrame** tipi içermektedir. DiscreteDoubleKeyFrame elementleri **KeyTime** niteliğinde belirtilen anda, animasyon uygulanan kontrolün ilgili özelliğinin **value** niteliği ile belirtilen değerde olmasını sağlamaktadır. Söz gelimi animasyondaki zaman çizelgesinin 2nci saniyesinde **Top** değeri 30 piksel, 1.5nci saniyesinde **Left** değeri 150 piksel olacak şekilde belirlenmektedir. Daha öncede belirtildiği gibi kontrolün söz konusu koordinat noktalarına ulaşması sırasında akıcı bir efekt yerine sıçrama(Jump) efekti uygulanmaktadır. Her halde bir yerden bir yere ışınlanmanın WPF' cesinin bu olduğunu söyleyebiliriz. Bu durumu aşağıdaki Flash görselinden daha net bir şekilde analiz edebiliriz.

Elbetteki aynı örneği kod yardımıylada gerçekleştirebiliriz. Bunun için aşağıdaki XAML içeriğine ve kodlara sahip bir pencere hazırlamamız yeterli olacaktır.

XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.KodlaDiscreteKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Kod ile Discrete
Kullanimi" Height="300" Width="300">
```

```
   <Canvas>
      <Rectangle Name="Dortgen" Canvas.Top="0" Canvas.Left="0" Height="75"
Width="55">
         <Rectangle.Fill>
            <ImageBrush ImageSource="Kitap.jpg"/>
         </Rectangle.Fill>
      </Rectangle>
   </Canvas>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Media.Animation; // Animasyon tiplerini içeren isim alanıdır

namespace AnimasyonIslemleri
{
   public partial class KodlaDiscreteKullanimi : Window
   {
      private void AnimasyonuHazirla()
      {
         // Storyboard nesnesi örneklenir
         Storyboard strBrd = new Storyboard();
         // Animasyonun sürekli tekrar edeceği belirtilir
         strBrd.RepeatBehavior = RepeatBehavior.Forever;
         // Animasyonun zaman çizgisini tamamladığında aynı yoldan geriye dönmesi
sağlanır
         strBrd.AutoReverse = true;

         // Dortgen nesnesinde Canvas' ın Top özelliği üzerinde Frame bazlı animasyon
yapılmasını sağlayacak şekilde DoubleAnimationUsingKeyFrames nesnesi örneklenir
         DoubleAnimationUsingKeyFrames dblA1 = new
DoubleAnimationUsingKeyFrames();
         Storyboard.SetTargetName(dblA1, "Dortgen");
         Storyboard.SetTargetProperty(dblA1, new
```

```
PropertyPath(Canvas.TopProperty));

        // Frame' lerdeki Top özelliğinin değerleri belirtilir. Bunun için
DiscreteDoubleKeyFrame nesneleri örneklenerek KeyFrames koleksiyonuna eklenir.
        dblA1.KeyFrames.Add(new DiscreteDoubleKeyFrame(150,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 1))));
        dblA1.KeyFrames.Add(new DiscreteDoubleKeyFrame(30,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 2))));
        dblA1.KeyFrames.Add(new DiscreteDoubleKeyFrame(120,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 3))));
        dblA1.KeyFrames.Add(new DiscreteDoubleKeyFrame(60,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 4))));
        dblA1.KeyFrames.Add(new DiscreteDoubleKeyFrame(90,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 5))));
        dblA1.KeyFrames.Add(new DiscreteDoubleKeyFrame(75,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 6))));
        // DoubleAnimationUsingKeyFrames nesne örneği Storyboard' a alt element olarak
eklenir.
        strBrd.Children.Add(dblA1);

        DoubleAnimationUsingKeyFrames dblA2 = new
DoubleAnimationUsingKeyFrames();
        Storyboard.SetTargetName(dblA2, "Dortgen");
        Storyboard.SetTargetProperty(dblA2, new PropertyPath(Canvas.LeftProperty));

        dblA2.KeyFrames.Add(new
DiscreteDoubleKeyFrame(150, KeyTime.FromTimeSpan(new TimeSpan(0,0, 0,
1,500))));
        dblA2.KeyFrames.Add(new DiscreteDoubleKeyFrame(30,
KeyTime.FromTimeSpan(new TimeSpan(0, 0,0, 2,500))));
        dblA2.KeyFrames.Add(new DiscreteDoubleKeyFrame(120,
KeyTime.FromTimeSpan(new TimeSpan(0, 0,0, 3,500))));
        dblA2.KeyFrames.Add(new DiscreteDoubleKeyFrame(60,
KeyTime.FromTimeSpan(new TimeSpan(0, 0,0, 4,500))));
        dblA2.KeyFrames.Add(new DiscreteDoubleKeyFrame(90,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 0,5,500))));
        dblA2.KeyFrames.Add(new DiscreteDoubleKeyFrame(75,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 0,6,500))));
        strBrd.Children.Add(dblA2);

        // Pencere yüklendikten sonra
        Loaded += delegate(object sender, RoutedEventArgs e)
                {
                   // Animasyon başlatılır
                   strBrd.Begin(this);
```

```
                    };

        }
        public KodlaDiscreteKullanimi()
        {
            InitializeComponent();
            AnimasyonuHazirla();
        }
    }
}
```

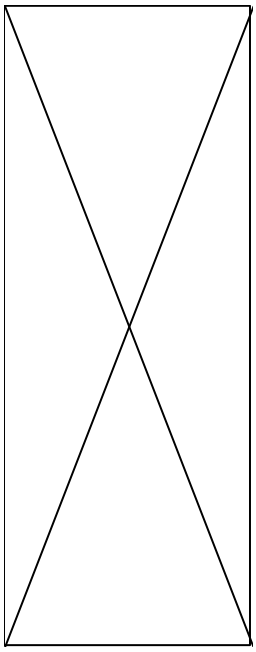
Frame tabanlı animasyon işlemlerinde **LinearDoubleKeyFrame** ve **DiscreteDoubleKeyFrame** tipleri dışında kullanılabilen bir diğer sınıfta **SplineDoubleKeyFrame**' dir. Bu tip sayesinde, animasyon işlemi sırasındaki ileri ve geri yönlü hareketlerde, hızlanma ve yavaşlama oranları daha net bir şekilde belirlenebilir. Söz konusu hızlanma verilerini belirlemek için, sadece **0.0** ile **1.0**arasında değerler alabilen **KeySpline** özelliğinden yararlanılır. Aşağıdaki **XAML** içeriğine sahip örnekte SplienDoubleKeyFrame tipi kullanılmaktadır.

XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.SplineKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Spline Kullanimi"
Height="422" Width="181">
    <Canvas>
        <Rectangle Name="Dortgen" Canvas.Top="0" Canvas.Left="0" Height="75"
Width="55">
            <Rectangle.Fill>
                <ImageBrush ImageSource="Kitap.jpg"/>
            </Rectangle.Fill>
            <Rectangle.Triggers>
                <EventTrigger RoutedEvent="Window.Loaded">
```

```
            <EventTrigger.Actions>
               <BeginStoryboard>
                  <Storyboard AutoReverse="True" RepeatBehavior="Forever">
                     <DoubleAnimationUsingKeyFrames
Storyboard.TargetName="Dortgen" Storyboard.TargetProperty="(Canvas.Top)">
                        <SplineDoubleKeyFrame KeyTime="0:0:3" Value="200"
KeySpline="0.3,0.0 0.9,0.3"/>
                     </DoubleAnimationUsingKeyFrames>
                  </Storyboard>
               </BeginStoryboard>
            </EventTrigger.Actions>
         </EventTrigger>
      </Rectangle.Triggers>
   </Rectangle>
</Canvas>
</Window>
```

Bu örneğin çalışma zamanındaki çıktısı aşağıdaki Flash animasyonunda olduğu gibidir.***(Flash dosyasının boyutu 298 Kb olduğundan yüklenmesi zaman alabilir.)***



Dikkat edilecek olursa düşey düzlemde hareket eden resim 3 saniyelik zaman dilimi içerisinde pencerenin üst noktasından 200 piksel aşağıya gelmektedir. Ancak bu hareketi sırasında**KeySpline** niteliğinde belirtilen değerler nedeni ile düşey olarak hızlanarak ve ters yönde de yavaşlayarak ilerlemektedir. KeySpline niteliğindeki değerler ile oynayarak daha farklı etkiler de elde edilebilir. Hatta bu değerler ile oynayarak farklı etkilerin nasıl oluşturulabileceğini incelemenizi şiddetle tavsiye ederim. Aynı örnek kod yardımıylada aşağıdaki gibi geliştirilebilir.

XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.KodlaSplineKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Kod ile Spline
Kullanimi" Height="397" Width="181">
   <Canvas>
      <Rectangle Name="Dortgen" Canvas.Top="0" Canvas.Left="0" Height="75"
Width="55">
         <Rectangle.Fill>
            <ImageBrush ImageSource="Kitap.jpg"/>
         </Rectangle.Fill>
      </Rectangle>
   </Canvas>
</Window>
```

Kod içeriği;

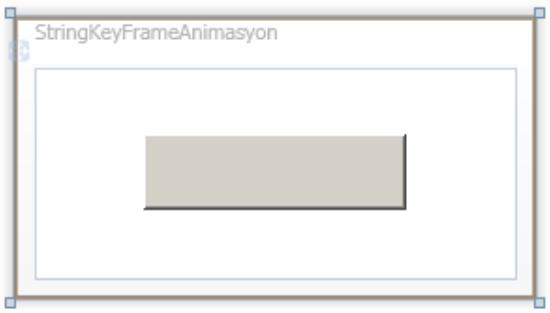
```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Media.Animation; // Animasyon tiplerini içeren isim alanıdır

namespace AnimasyonIslemleri
{
   public partial class KodlaSplineKullanimi : Window
   {
      private void AnimasyonuHazirla()
      {
         // Storyboard nesnesi örneklenir, animasyonun tekrarlı ve geriye dönüşlü olacak
şekilde çalışacağı belirlenir
         Storyboard strBrd = new Storyboard();
         strBrd.RepeatBehavior = RepeatBehavior.Forever;
         strBrd.AutoReverse = true;

         DoubleAnimationUsingKeyFrames dblA = new
DoubleAnimationUsingKeyFrames();

         // SplineDoubleKeyFrame nesnesi eklenir. Üçüncü parametre ile hızlanma ve
```

yavaşlama değerleri belirlenir.

```
        dblA.KeyFrames.Add(new SplineDoubleKeyFrame(200,
KeyTime.FromTimeSpan(new TimeSpan(0, 0, 3)), new KeySpline(0.3, 0, 0.9, 0.3)));
        // Animasyonun Dortgen isimli nesneye ve içinde bulunduğu Canvas' ın Top
özelliğine uygulanacağı belirlenir
        Storyboard.SetTargetName(dblA, "Dortgen");
        Storyboard.SetTargetProperty(dblA, new
PropertyPath(Canvas.TopProperty));

        // Animasyon nesnesi Storyboard' a alt element olarak eklenir
        strBrd.Children.Add(dblA);

        // Pencerenin yüklenmesi tamamlandıktan sonra
        Loaded += delegate(object sender, RoutedEventArgs e)
                    {
                       // Animasyon başlatılır
                       strBrd.Begin(this);
                    };
    }
    public KodlaSplineKullanimi()
    {
       InitializeComponent();
       AnimasyonuHazirla();
    }
  }
}
```

Bu örneklerdende anlaşılacağı üzere farklı veri tipleri için yazılmış olan Frame bazlı animasyon tipleri olduğunu söyleyebiliriz. Geliştirilen örnekler double tipi ile çalışan özellikler üzerinde Frame bazlı animasyonlar geliştirilebilmesi için **DoubleAnimationUsingKeyFrames** tipini kullanmaktadır. Ancak bu tip dışında **ColorAnimationUsingKeyFrames**,**PointAnimationUsingKeyFrames**, **ByteAnimationUsingKeyFrames** vb... sınıflarda yer almaktadır. Tüm bu Frame bazlı animasyon tipleri **LinearTipAdıKeyFrame, DiscreteTipAdıKeyFrame** ve**SplineTipAdıKeyFrame** sınıflarına ait nesne örnekleri ile çalışabilmektedir. Tabiki bu durum her veri türü için geçerli değildir. Söz gelimi **StringAnimationKeyFrames** tipi sadece**DiscreteStringKeyFrame** sınıfına ait nesne örneklerini taşıyabilir. Dolayısıyla enterpoasyon**(Interpolation)** efektleri gerçekleştirilemez.

Sıradaki örnekte bir Button kontrolünün Text özelliğinin içeriğini Frame bazlı animasyon işleminde nasıl ele alabileceğimizi inceleyeceğiz. Bu amaçla XAML içeriği aşağıdaki gibi olan bir pencere tasarladığımızı düşünelim.



XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.StringKeyFrameAnimasyon"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="StringKeyFrameAnimasyon" Height="149" Width="278">
   <Grid>
      <Button Name="btnMerhaba" Width="140" Height="40" FontSize="12"
FontWeight="Bold">
         <Button.Triggers>
            <EventTrigger RoutedEvent="Button.MouseEnter">
               <EventTrigger.Actions>
                  <BeginStoryboard>
                     <Storyboard RepeatBehavior="Forever">
                        <StringAnimationUsingKeyFrames
Storyboard.TargetName="btnMerhaba" Storyboard.TargetProperty="Content"
Duration="0:0:6">
                           <!-- Eğer Duration süresi belirtilmesse DiscreteStringKeyFrame
elementlerinden sonuncusu gösterilmiyor. Sonuncusununda gösterilmesi için Duration
süresi son DiscreteStringKeyFrame' inkinden fazla verilmelidir.-->
                           <DiscreteStringKeyFrame Value="WPF" KeyTime="0:0:1"/>
                           <DiscreteStringKeyFrame Value="ile" KeyTime="0:0:2"/>
                           <DiscreteStringKeyFrame Value="Temel" KeyTime="0:0:3"/>
                           <DiscreteStringKeyFrame Value="Animasyon"
KeyTime="0:0:4"/>
                           <DiscreteStringKeyFrame Value="İşlemleri"
KeyTime="0:0:5"/>
                        </StringAnimationUsingKeyFrames>
                     </Storyboard>
                  </BeginStoryboard>
               </EventTrigger.Actions>
            </EventTrigger>
         </Button.Triggers>
      </Button>
```

```
   </Grid>
</Window>
```

Bu örneğin çalışma zamanındaki işleyişi aşağıdaki Flash görselinde olduğu gibidir. **DiscreteStringKeyFrame** elementlerine ait **Value** nitelikleri tahmin edileceği üzere, animasyonun bağlandığı kontrolün ilgili özelliğindeki string bilginin t anında ne olacağını belirtmekte kullanılmaktadır. **KeyTime** niteliğine verilen değerler ile söz konusu t anları belirtilir.

Elbetteki bu örnekteki animasyon etkisi kod yardımıylada geliştirilebilir. Aşağıdaki kod parçalarında bu duruma bir örnek verilmektedir.

XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.KodlaStringKeyFrameAnimasyon"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="KodlaStringKeyFrameAnimasyon" Height="144" Width="300">
   <Grid>
      <Button Name="btnMerhaba" Width="140" Height="40" FontSize="12"
FontWeight="Bold"/>
   </Grid>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Media.Animation;

namespace AnimasyonIslemleri
{
   public partial class KodlaStringKeyFrameAnimasyon : Window
   {
      string slogan = "CSharp az keyword ile çok iş yapmamızı sağlayan bir dildir";
```

```
        private void AnimasyonuHazirla()
        {
            Storyboard strBrd = new Storyboard();
            strBrd.RepeatBehavior = RepeatBehavior.Forever;

            StringAnimationUsingKeyFrames strA = new
StringAnimationUsingKeyFrames();
            string[] kelimeler = slogan.Split(' ');
            strBrd.Duration = TimeSpan.FromSeconds(kelimeler.Length + 1);
            for(int i=1;i<=kelimeler.Length;i++)
            {
                strA.KeyFrames.Add(new DiscreteStringKeyFrame(kelimeler[i-
1],KeyTime.FromTimeSpan(TimeSpan.FromSeconds(i))));
            }
            Storyboard.SetTargetName(strA, "btnMerhaba");
            Storyboard.SetTargetProperty(strA, new
PropertyPath(Button.ContentProperty));
            strBrd.Children.Add(strA);
            btnMerhaba.MouseEnter += delegate(object sender, MouseEventArgs e)
                                     {
                                         strBrd.Begin(this);
                                     };
        }
        public KodlaStringKeyFrameAnimasyon()
        {
            InitializeComponent();
            AnimasyonuHazirla();
        }
    }
}
```

Bu kez, bir slogan cümle içerisindeki kelimeler boşluk karakterine göre **Split** metodu ile ayrıştırılmakta ve elde edilen string dizisindeki her bir kelime için **DiscreteStringKeyFrame** temelli, saniyede bir artan animasyon uygulanmaktadır. Bu örnek kod parçasının çalışma zamanındaki çıktısı ise aşağıdaki Flash görselindeki gibi olacaktır.

Makalemizde son olarak bir bileşenin belirli bir rota üzerinde hareket etmesinin animasyon teknikleri ile nasıl gerçekleştirilebileceğini incelemeye çalışacağız. Önceki makalelerimizden hatırlayacağınız gibi, **Path** ve **PathGeometry** tiplerini kullanarak birbirlerine bağlı yayların çizilmesi mümkündür. Bu tip bir elementin oluşturacağı rotayı takip eden bir bileşen söz konusu olduğunda, animasyon tiplerinden **TipAdıAnimationUsingPath** isimlendirmesi ile tanımlananlardan yararlanbiliriz. Örneğin bir **PathGeometry** ile tanımlanan rotada hareket edecek bir nesnenin içinde bulunduğu **Canvas**' ın **Top** ve **Left** özellikleri söz konusu

ise **DoubleAnimationUsingPath** tipini kullanabiliriz. Aşağıdaki **XAML** çıktısında bu durum örneklenmeye çalışılmaktadır.

XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.PathFrameKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="PathFrameKullanimi"
Height="300" Width="300">
  <Window.Resources>
    <PathGeometry x:Key="YolGeometrisi">
      <PathFigure>
        <BezierSegment Point1="0,0" Point2="10,120" Point3="70,40"/>
        <ArcSegment Point="150,125" SweepDirection="Clockwise" Size="50,50"
IsLargeArc="True" RotationAngle="60"/>
        <ArcSegment Point="200,200" SweepDirection="Counterclockwise"
Size="50,50" IsLargeArc="True" RotationAngle="60"/>
      </PathFigure>
    </PathGeometry>
  </Window.Resources>
  <Canvas>
    <Path Data="{StaticResource YolGeometrisi}" Stroke="LightGray"
StrokeThickness="2"/>
      <Rectangle Height="65" Name="Dortgen" Width="55">
        <Rectangle.Fill>
          <ImageBrush ImageSource="Kitap.jpg" />
        </Rectangle.Fill>
        <Rectangle.Triggers>
          <EventTrigger RoutedEvent="Rectangle.MouseEnter">
            <BeginStoryboard>
              <Storyboard AutoReverse="True" RepeatBehavior="Forever">
                <DoubleAnimationUsingPath Storyboard.TargetName="Dortgen"
Storyboard.TargetProperty="(Canvas.Top)" PathGeometry="{StaticResource
YolGeometrisi}" Source="Y" Duration="0:0:6" />
                <DoubleAnimationUsingPath Storyboard.TargetName="Dortgen"
Storyboard.TargetProperty="(Canvas.Left)" PathGeometry="{StaticResource
YolGeometrisi}" Source="X" Duration="0:0:6" />
              </Storyboard>
            </BeginStoryboard>
          </EventTrigger>
        </Rectangle.Triggers>
      </Rectangle>
    </Canvas>
</Window>
```

Söz konusu örnekte, **Storyboard** elementi içerisinde **DoubleAnimationUsingPath** alt elementi kullanılmaktadır. Bu elementlerden iki adet tanımlanmasının sebebi, Dortgen isimli nesnenin Top ve Left özelliklerinin ayrı ayrı değiştirilmesi gerekliliğidir. DoubleAnimationUsingPath tipinde yer alan niteliklerden **Storyboard.TargetName** ile animasyonun hangi kontrole uygulanacağı belirtilir ki bu örnekte söz konusu kontrol Dortgen isimli **Rectangle** dır. **Storyboard**.**TargetProperty** nitelikleri ise **Rectangle**' ın içerisinde bulunduğu **Canvas**' ın **Top** ve **Left** özelliklerinin değerleri üzerinde hareketlendirmeler yapılacağını belirtir. **PathGeometry** nitelikleri tahmin edileceği üzere rota bilgisini içeren pencere kaynağını **(Windows Resource)** işaret etmektedir. MSDN kaynakları**Resource** olarak yapılan bu tip tanımlamaların animasyon işlemlerinde performans yönünden olumlu katkı yapacağını belirtmektedir. Static kaynak olarak tanımlanan YolGeometrisi isimli veri kümesi yardımıyla Path elementinin rota bilgiside verilmektedir. Bunun içinde **Path** elementinin **Data** niteliğine değer ataması yapılmıştır. Bu örneğin çalışma zamanındaki çıktısı aşağıdaki Flash animasyonunda olduğu gibidir. ***(Flash dosyasının boyutu 352 kb olduğundan yüklenmesi zaman alabilir.)***

Görüldüğü gibi WPF mimarisinde gerek XAML tarafında gerekse kod tarafında animasyon işlemleri için son derece etkili yollar kullanılabilmektedir. Bence burada önemli olan noktalardan biriside işaretleme dili içerisinde bu tip gelişmiş fonksiyonelliklerin geliştirilebilmesidir. İlerleyen makalelerimizde WPF ile ilgili farklı konularada değinmeye çalışacağız. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

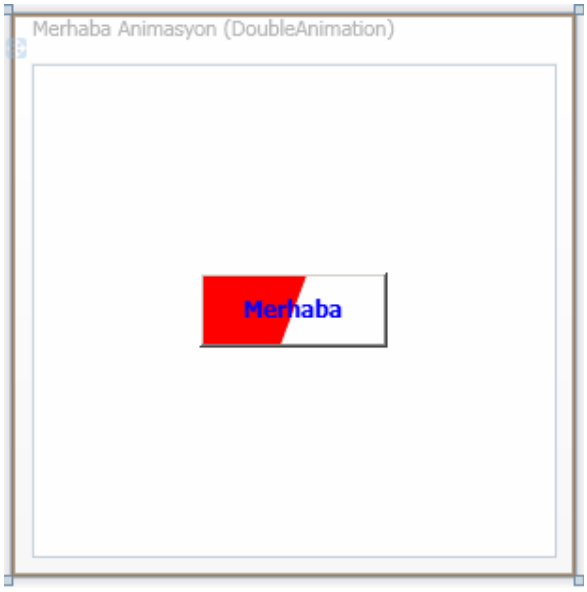
Örnek Uygulama için Tıklayın

WPF Temel Animasyon İşlemleri - 1 (2007-09-26T19:03:00)

wpf,

Windows uygulamalarında kullanıcılara görsel bir şölen sunmak için animasyon işlemlerinden yararlanılır. Ne varki söz konusu animasyonları gerçekleştirebilmek amacıyla zorlu olan bazı süreçlerin aşılması gerekmektedir. Burada aslında, **GDI+(Graphics Device Interface)** alt yapısını(Infrastructure) daha etkili kullanmak adına daha çok ve daha karmaşık kodların yazılması, form üzerinde animasyonun zaman çizelgesine göre **Timer** bileşenleri ile **Interval** özelliği ve **Tick** olayı(event) gibi üyelerlerle uğraşılması ve hatta yeri geldiğinde ana iş parçasının**(Main Thread)** bozabileceği durumların önüne geçmek için özel tedbirler alınması söz konusudur. Hele birde işin içerisine üç boyutlu(3D) nesneler girdiğinde ve bunların üzerinde kullanıcı etkileşimli olayların tetiklenebilmesi de istendiğinde programıcının geliştirme süreci hem zorlaşacak hemde uzayacaktır. Tabiki günümüzde bu tip işlemleri gerçekleştirmek için ele alına pek çok kolaylaştırıcı yazılım vardır. Ancak bir olaya .Net Windows programcısı olarak bakıyor olacağız. Durumun zorluğunu daha net anlayabilmek adına, basit olarak form üzerinde yer alan bir Button kontrolünün **yükseklik(Height)** ve **genişlik(Width)** özelliklerinin sürekli olarak belirli

oranlarda büyüyüp küçüldüğünü düşünelim. Bunu Windows programlamada nasıl yapmamız gerektiğini hayal edin. Her şeyden önce sürenin söz konusu olduğu bu senaryoda bir **Timer** bileşenin var olması ve **Tick** olayının uygun şekilde ele alınması(Handle) gerekmektedir. Oysaki **WPF (Windows Presentation Foundation)** mimarisinde bu ve benzeri animasyon işlemleri çok daha kolay, hızlı ve güçlü bir şekilde gerçekleştirilebilmektedir. İşte bu makalemizde söz konusu durumu incemeleye başlayacak ve WPF mimarisinde temel animasyon işlemlerine değineceğiz. WPF mimarisinin söz konusu animasyon geliştirme işlerini nasıl kolaylaştırdığını görmek adına aşağıda **XAML(eXtensible Application Markup Language)** çıktısı aşağıda görülen örnek ile hızlı bir şekilde başlamakta yarar olacağı kanısındayım.



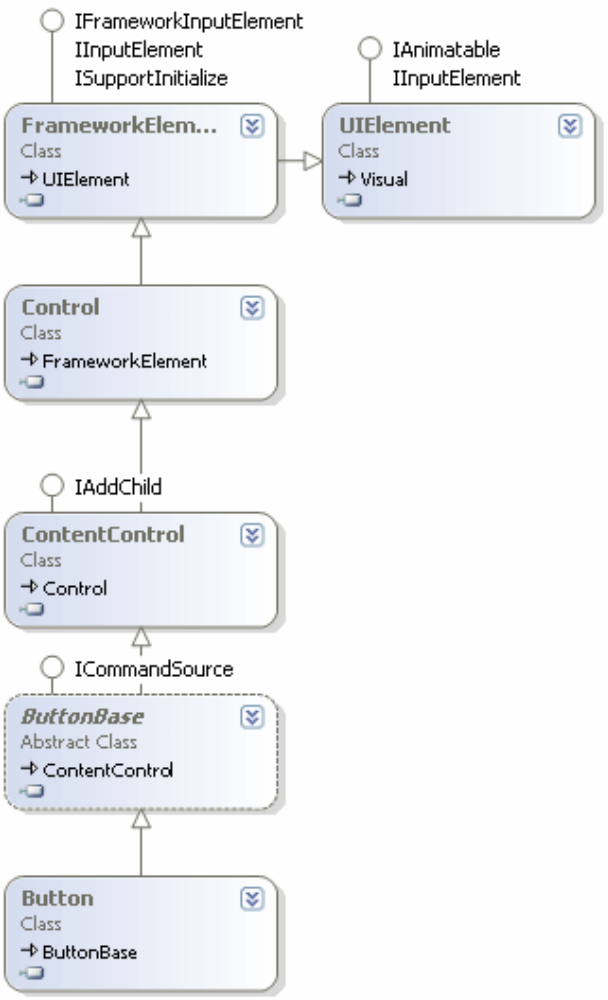

```
<Window x:Class="AnimasyonIslemleri.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Merhaba Animasyon
(DoubleAnimation)" Height="300" Width="300">
   <Grid>
      <Button Name="btnMerhaba" Width="100" Height="40" Content="Merhaba"
FontSize="12" Foreground="Blue" FontWeight="Bold">
         <Button.Background>
            <LinearGradientBrush>
               <LinearGradientBrush.GradientStops>
                  <GradientStop Color="Red" Offset="0.5"/>
                  <GradientStop Color="White" Offset="0.5"/>
               </LinearGradientBrush.GradientStops>
            </LinearGradientBrush>
         </Button.Background>
         <Button.Triggers>
            <EventTrigger RoutedEvent="Button.MouseEnter">
               <EventTrigger.Actions>
```

```
            <BeginStoryboard>
              <Storyboard>
                <DoubleAnimation Storyboard.TargetName="btnMerhaba"
Storyboard.TargetProperty="Width" By="5" RepeatBehavior="Forever"
From="100" To="150" Duration="0:0:1" AutoReverse="True"/>
                <DoubleAnimation Storyboard.TargetName="btnMerhaba"
Storyboard.TargetProperty="Height" From="40" To="60" Duration="0:0:1"
RepeatBehavior="Forever" AutoReverse="True"/>
              </Storyboard>
            </BeginStoryboard>
          </EventTrigger.Actions>
        </EventTrigger>
      </Button.Triggers>
    </Button>
  </Grid>
</Window>
```

XAML içeriğinde yer alan elementler ve **nitelikleri(attributes)** hakkında konuşmadan önce bu pencerenin çalışma zamanına bir bakalım. Aşağıdaki **Flash** animasyonunda çalışma zamanındaki**(run-time)** durum görülmektedir. Dikkat edilecek olursa **pencere(Window)** üzerindeki **Button** kontrolünün **Width** ve **Height** özelliklerinde genişlemeler olmaktadır. Dikkat çekici bir diğer noktada animasyonun, mouse ile Button kontrolü üzerine gelindiğinde başladığıdır. Bununla birlikte söz konusu aminasyon sürekli olarak devam etmektedir.

Gelelim XAML içeriğinde neler yapıldığına. Daha önceki makalelerimizden de hatırlanacağı gibi Button bileşeninin arka planında **LinearGradientBrush** elementi ile geçişli renklerden oluşan bir dolgu efekti kullanılmaktadır. Button elementinin içerisinde animasyon özelliklerinin belirtildiği yer **Button.Triggers** alt elementi ile başlamaktadır. Hemen altta yer alan **EventTrigger**elementinin **RoutedEvent** niteliğine atanan değer ile animasyonun hangi olayın tetiklenmesi sonrası başlatılacağı belirtilmektedir. Örneğimizde **MouseEnter** olayı bu amaçla kullanılmaktadır. Burada olay adı belirtilirken **tipAdı.Özellikadı** (örneğin **Button.MouseEnter** gibi) şeklinde olmasına dikkat edilmelidir. Aksi durumda bir istisna(Exception) alınması muhtemeldir. Dolayısıyla temel anlamda animasyonların başlatılabilmesi için olaylardan yararlanılabildiğini ve bunların ilgili kontrollere birer tetikleyici**(Trigger)** ile bağlandığını söyleyebiliriz.

NOT : *Bir kontrolün animasyon işlemlerinde kullanılabilmesi için* ***IAnimatable*** *arayüzünü uyarlamış(Implement) olması gerekmektedir. Söz gelimi* ***Button*** *bileşeninin sınıf diagramındaki(Class Diagram) görüntüsüne bakıldığında IAnimatable arayüzünü uyarladığı açık bir şekilde görülebilir.*



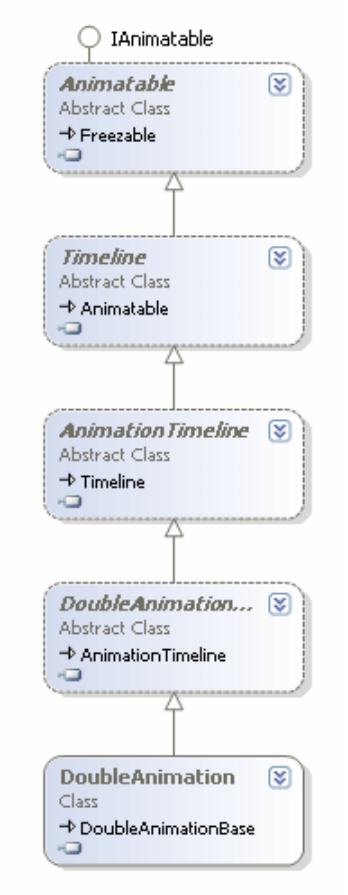
Animasyon işlemlerinde önemli olan noktalardan birisi kontrolün veya şeklin hangi özelliğinin değerinin, ne şekilde değiştirileceğidir. İlk örnekte Button kontrolünün **Width** ve **Height**özelliklerinin değerlerinin her ikisi birden değiştirilmektedir. Peki WPF mimarisinde hangi kontrolün hangi özelliğinin değerinin, ne şekilde değiştirileceği nasıl belirlenmektedir? İşte bu noktada devreye **StoryBoard** tipi ve alt elementleri girmektedir.

NOT : StoryBoard *tipi hangi kontrolün hangi özelliği üzerinde nasıl bir animasyon uygulanacağını belirleyip bunu başlatmak ve belirli bir tetikleyici(Trigger) olaya bağlamak için kullanılan önemli bir sınıftır.*

Örnek **XAML** içeriğinde dikkat edilecek olursa StoryBoard elementi altında iki adet **DoubleAnimation** elementi tanımlanmıştır.

NOT : *WPF içerisinde yer alan* ***temel animasyon tipleri(Basic Animation Types)*** *uygun veri türleri ile çalışacak şekilde tasarlanmışlardır. Bu tiplerin adları* ***DoubleAnimation****,****ColorAnimation****,* ***PointAnimation,*** ***VectorAnimation*** *örneklerinde olduğu gibi* ***TürAdıAnimation*** *kelimelerinden*

*oluşmaktadır. Animasyon tipleride **IAnimatable** arayüzünü uyarlamaktadır. Söz gelimi örnekte kullanılan DoubleAnimation tipinin sınıf diagramındaki(Class Diagram) görüntüsü aşağıdaki gibidir.*



*Ayrıca söz konusu animasyon tipleri **Animatable** isimli **abstract** sınıfdan da türemektedir. Buna göre ilgili abstract sınıfı ve arayüzü kullanarak kendi animasyon tiplerimizde yazabiliriz. WPF içerisinde temel animasyon tipleri dışında yer alan animasyon tipleride vardır. **ColorAnimationUsingKeyFrames** gibi. Bu tipleri ilerleyen makalelerimizde incelemeye çalışıyor olacağız.*

Burada animasyon tipinin kullanımı ve özellikleri hakkında biraz konuşmakta yarar vardır. Örnekte yer alan Button kontrolünün animasyon etkisi oluşturan özellikleri Width ve Height üyeleridir. Bu özellikler **Double** tipinden değerler alabilmektedir. Bu nedenle **StoryBoard** içerisinde tanımlanan animasyon tipleri **DoubleAnimation** tipindendir. Dolayısıyla animasyon işleminde kullanılacak olan kontrol özelliği için, uygun olan animasyon tipinin seçilmesi gerekmektedir. DoubleAnimation tipi içerisinde oldukça önemli **nitelik(attribute)** tanımlamaları yer almaktadır. Bunlardan**StoryBoard.TargetName** niteliği ile animasyonun uygulanacağı kontrol seçilmektedir ki örnekte bu btnMerhaba isimli **Button** kontrolüdür. Diğer taraftan **StoryBoard.TargetProperty** niteliği ile kontrolün hangi özelliğinin animasyon

işleminde kullanılacağı belirtilir. **From** niteliği ile hedef özelliğin hangi değerden başlayacağı, **To** niteliği ilede hedef özelliğin ilgili değerinin hangi değere kadar gelebileceği belirtilir.

Bir başka deyişle Button kontrolünün genişliği 100 değerinden 150 değerine gelecekse From niteliğine 100, To niteliğine ise 150 değerleri atanır. Burada istenirse **By** niteliği kullanılarak değerin artış oranıda belirtilebilir. Çok doğal olarak animasyonun ne kadar süre devam edeceği bilgisinin de belirtilmesi gerekmektedir. Bu amaçla **DoubleAnimation** tipinin **Duration** niteliği kullanılmaktadır. Duration niteliği temelde **saat:dakika:saniye** şeklinde bir değer almaktadır. Buna göre örnek **XAML** içeriğinde yer alan **0:0:1** değeri animasyondaki zaman çizgisinin (Timeline) 1 saniye olacağını belirtmektedir. **RepeatBehavior** niteliği ile animasyonun tekrar sayısı belirtilir. Örnekte verilen **Forever** değerine göre animasyon sürekli olarak devam edecektir. Son olarak**AutoReverse** değerine **true** atanarak, 1 saniyelik animasyon sona erdiğinde işlemlerin tekrar başlangıç haline tersten bir animasyon ile gitmesi sağlanmaktadır. Bu nedenle Flash animasyonunda görüldüğü gibi kontrol büyüdükten sonra aynı animasyon etkisi ile tersten küçülmüş ve ilk haline gelmiştir.

İlk örnekte söz konusu animasyon işlemleri için **XAML** elementlerinden ve niteliklerinden(attributes) yararlanılmıştır. Çok doğal olarak kod ile dinamik olarak aynı etkiler oluşturulabilir. Sıradaki örneğimizde kod tarafında animasyon işlemlerinin nasıl yapılabileceğini incelemeye çalışacağız. Bu amaçla **penceremize(Window)** ait XAML içeriğini ve kodlarını aşağıdaki gibi hazırladığımızı düşünelim.

XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.KodlaMerhabaAnimasyon"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Kodla Merhaba
```

```
Animasyon" Height="300" Width="300">
   <Grid>
      <Button Name="btnMerhaba" Width="100" Height="50" Foreground="Gold"
Background="Black" Content="Merhaba" FontSize="14" FontWeight="Bold">
      </Button>
   </Grid>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Media.Animation; // Animasyon tiplerini barındıran isim alanıdır

namespace AnimasyonIslemleri
{
   public partial class KodlaMerhabaAnimasyon : Window
   {
      private void AnimasyonuHazirla()
      {
         // Button kontrolünün Width özelliği üzerinde animasyon işlemi yaptıracağımızdan
ve Width özelliği Double tipinden değerler aldığından DoubleAnimation tipi
kullanılmaktadır.
         // İlk parametre From
         // İkinci parametre To
         // Üçüncüd parametre animasyonun süresi
         DoubleAnimation dblAnmtr = new DoubleAnimation(100, 150, new
Duration(new TimeSpan(0, 0, 0, 3)));
         // Animasyonun tekrar sayısı belirtilir.
         dblAnmtr.RepeatBehavior = new RepeatBehavior(2);

         // Yeni bir StoryBoard oluşturulur
         Storyboard strBrd = new Storyboard();
         // DoubleAnimation tipi bu storyBoard' a eklenir
         strBrd.Children.Add(dblAnmtr);
         // Hedef kontrol belirlenir
```

```
        Storyboard.SetTargetName(dblAnmtr, btnMerhaba.Name);
        // Animasyon işleminde Button kontrolünün hangi özelliğinin değiştirileceği
belirtilir.
        Storyboard.SetTargetProperty(dblAnmtr,new
PropertyPath(Button.WidthProperty));
        // Animasyon Begin metodu ile Button üzerine Mouse ile gelindiğinde başlatılır
        btnMerhaba.MouseEnter += delegate(object sender, MouseEventArgs e)
                                 {
                                    strBrd.Begin(this);
                                 };
    }

    public KodlaMerhabaAnimasyon()
    {
        InitializeComponent();
        AnimasyonuHazirla();
    }
  }
}
```

Bu kez animasyon işlemleri kod tarafında gerçekleştirilmektedir. Örnekte ilk olarak **DoubleAnimation** tipinden bir örnek oluşturulmuş ve yapıcı metodu içerisinde bazı ilk değerlerin **(From, To, Duration** gibi) verilmesi sağlanmıştır. Sonrasında ise **RepatBehavior** özelliği ile tekrar sayısı belirtilmektedir. Bu örnekte bir öncekinden farklı olarak **Forever** değeri yerine 2 kullanılmıştır. Buna göre söz konusu animasyon sadece iki kere tekrar edecektir. Sonrasında ise her zamanki gibi animasyonu, kontrol ve kontrolün özelliği ile ilişkilendirecek **StoryBoard** nesne örneği oluşturulur. Animasyonun başlatılması için yine kontrolün **MouseEnter** olayı göz önüne alınmıştır. Animasyon işlemi için **StoryBoard** nesne örneğinin **Begin** metodunu kullanmak yeterlidir. Ancak bu metodun uygun olay için tetiklenmesini sağlamak gerekmektedir. Bu amaçlada isimsiz bir metod**(anonymous method)** yazılarak hem Button kontrolünün MouseEnter olayı tanımlanmış hemde gerçekleştiğinde işletilmesi istenen kodlar belirlenmiştir. Örneğin çalışma zamanındaki çıktısı aşağıdaki Flash animasyonunda olduğu gibidir.

Gelelim bir diğer örneğe. Önceki örneklerde **Button** kontrolonün boyutlarını ele alan animasyon tipi kullanılmıştır. Sıradaki örnekte ise **ColorAnimation** tipi başrolü oynamaktadır. Tahmin edileceği gibi bu tip, bir rengin başka bir renge dönüşümünün animasyon olarak gerçekleştirilmesini sağlamaktadır. Örneğin **XAML** içeriği aşağıdaki gibidir.


```
<Window x:Class="AnimasyonIslemleri.RenkliAnimasyon"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Renkli
Animasyon(ColorAnimation)" Height="300" Width="300">
   <Grid>
      <Button Name="btnMerhaba" Foreground="White" Content="Merhaba"
FontSize="12" FontWeight="Bold" Width="100" Height="50">
         <Button.Background>
            <SolidColorBrush x:Name="DolguRengi" Color="Red"/>
         </Button.Background>
         <Button.Triggers>
            <EventTrigger RoutedEvent="Button.MouseEnter">
               <EventTrigger.Actions>
                  <BeginStoryboard>
                     <Storyboard>
                        <ColorAnimation Storyboard.TargetName="DolguRengi"
Storyboard.TargetProperty="(SolidColorBrush.Color)" From="Red" To="Blue"
AutoReverse="True" Duration="0:0:6" RepeatBehavior="Forever"/>
                     </Storyboard>
                  </BeginStoryboard>
               </EventTrigger.Actions>
            </EventTrigger>
         </Button.Triggers>
      </Button>
   </Grid>
</Window>
```

ColorAnimation tip olarak kullanıldığında kontrollerin renk değeri alabilen özelliklerinin animasyon işlemlerinde kullanılabileceği anlaşılmaktadır. Örnekte **Button** kontrolünün

arka plan dolgusu animasyon içerisine katılmaktadır. Bu nedenle ColorAnimation kontrolünün ele alacağı kontrol ve özellik DoubleAnimation tipinin kullanıldığı örneklerdekinden biraz daha farklıdır. Dikkat edilecek olursa **SolidColorBrush** elementi içerisinde **DolguRengi** ismi **x:Name** niteliği yardımıyla tanımlanmıştır. Bu isim **StoryBoard.TargetName** niteliğinde kullanılmaktadır. Bir başka deyişle StroyBoard tipinin **XAML** içeriğindeki SolidColorBrush elementini bulması kolaylaştırılmaktadır. Bunun en büyük nedenlerinden birisi SolidColorBrush elementinin Button elementinin altında kalmış olması ve Button tipine isim ile erişilebiliyor olmasına rağlem aynı işin söz konusu SolidColorBrush için sağlanamıyor olmasıdır. Diğer taraftan animasyonda kullanılacak özellik değeride **StoryBoard.TargetProperty** elementi içerisinde ele alınırken **(SolidColorBrush.Color)** yazım stili kullanılmaktadır. Animasyona göre, Button kontrolünün arka plan rengi kırmızından maviye doğru değişecek ve sonrasında tekrardan maviden kırmızıya dönecektir**(AutoReverse=true** olduğu için). Animasyondaki **zaman çizelgesinin(Timeline)**süresi 6 saniyedir. Aşağıdaki Flash animasyonunda uygulamanın çalışma zamanındaki(Run-time) görüntüsü yer almaktadır.

Aynı etki aşağıdaki kod parçası yardımıyla dinamik olaraktan da gerçekleştirilebilir.

XAML İçeriği;

```
<Window x:Class="AnimasyonIslemleri.KodlaRenkliAnimasyon"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
Title="KodlaRenkliAnimasyon" Height="300" Width="300">
   <Grid>
      <Button Name="btnMerhaba" Width="100" Height="50" Content="Merhaba"
Foreground="White" FontSize="14" FontWeight="Bold">
         <Button.Background>
            <SolidColorBrush x:Name="firca" Color="Red"/>
         </Button.Background>
      </Button>
   </Grid>
</Window>
```

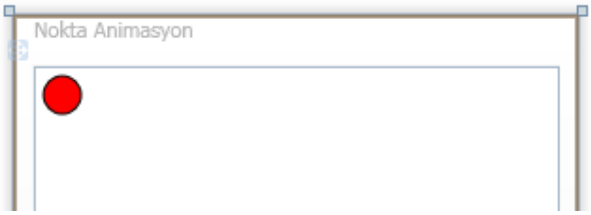
Kod İçeriği;

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Media.Animation; // Animasyon tiplerini içeren isim alanıdır(Namespace)

namespace AnimasyonIslemleri
{
   public partial class KodlaRenkliAnimasyon : Window
   {
      private void AnimasyonuAyarla()
      {
         // Animasyon tipi oluşturulur.
         // İlk parametre başlangıç rengini işaret eder (From)
         // İkinci parametre bitiş rengini işaret eder (To)
         // Üçüncü parametre zaman çizgisinin süresidir
         ColorAnimation clrAnmtr = new ColorAnimation(Colors.Red, Colors.Blue, new Duration(new TimeSpan(0, 0, 6)));
         // Animasyonun sürekli tekrar edeceği belirtilir
         clrAnmtr.RepeatBehavior = RepeatBehavior.Forever;

         // Animasyonu kontrol ve özelliği ile ilişkilendirecek StoryBoard nesnesi örneklenir.
         Storyboard strBrd = new Storyboard();
         strBrd.Children.Add(clrAnmtr);
         Storyboard.SetTargetName(clrAnmtr, "firca"); // SolidColorBrush kontrolünün x:name niteliğinin değeri ile animasyon uygulanacak kontrol belirtilmiş olur.
         Storyboard.SetTargetProperty(clrAnmtr, new PropertyPath(SolidColorBrush.ColorProperty)); // SolidColorBrush kontrolünün Color özelliği üzerinde animasyonun uygulanacağı belirtilir.

         // Animasyon yine MouseEnter olay metodunda Begin metodu ile başlatılır.
         btnMerhaba.MouseEnter += delegate(object sender, MouseEventArgs e)
                                  {
                                     strBrd.Begin(this);
                                  };
      }
      public KodlaRenkliAnimasyon()
      {
         InitializeComponent();
         AnimasyonuAyarla();
      }
   }
}
```

Yukarıdaki kodun yer aldığı pencere çalıştırıldığında, bir önceki örnekteki ile aynı etkinin oluştuğu görülecektir.

Yazımıza **PointAnimation** tipini ele alacağımız son bir örnek ile devam edelim. PointAnimation tipi adındanda anlaşılacağı üzere **Point** tipinden değerler alabilen kontrol özellikleri(Control Property) üzerinde animasyon işlemleri gerçekleştirilebilmesi amacıyla kullanılır. Örnek olarak aşağıdaki **XAML** içeriğini ele alabiliriz.



XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.NoktaAnimasyon"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Nokta Animasyon"
Height="300" Width="300">
   <Grid>
      <Path Name="Daire" Fill="Red" Stroke="Black" StrokeThickness="1">
         <Path.Data>
            <EllipseGeometry x:Name="daireGeo" Center="15,15" RadiusX="10"
RadiusY="10" />
         </Path.Data>
         <Path.Triggers>
            <EventTrigger RoutedEvent="Path.Loaded">
               <BeginStoryboard>
                  <Storyboard>
                     <PointAnimation Storyboard.TargetName="daireGeo"
Storyboard.TargetProperty="Center" From="15,15" To="285,15"
Duration="0:0:1.75" AutoReverse="true" RepeatBehavior="Forever"/>
                  </Storyboard>
               </BeginStoryboard>
            </EventTrigger>
         </Path.Triggers>
      </Path>
   </Grid>
</Window>
```

Burada örnek olarak Path tipinden bir **şekil(Shape)** kullanılmış ve **EllipsGeometry** tipinden yararlanılarak bir daire oluşturulmuştur. Burada **Point** tipinden değer alan üye **Center** özelliğidir. Bu anlamda PointAnimation

tipinin hedef aldığı kontrol aslında **EllipseGeometry** nesnesidir. Bu işaretlemeyi yapabilmek için yine **x:Name** niteliğinden yararlanılmıştır. Diğer taraftan EllipsGeometry nesne örneğinin Center özelliği animasyon işlemine tabi tutulacağından **StoryBoard.TargetProperty** niteliğine **Center** değeri verilmiştir. PointAnimation tipi basit olarak kontrolün bir zaman çizgisi(Timeline) içerisinde belirli bir koordinata doğru hareket etmesini sağlamaktadır. Buna göre örnekteki daire şekli, 15:15 noktasından 285:15 noktasına doğru 1.75 saniyelik zaman dilimi içerisinde hareket edecek ve **AutoReverse** özelliğine **true** değerinin atanması nedeniylede bu süre sonunda tekrardan geriye ilk konumuna gidecektir. Tahmin edileceği üzere **RepeatBehavior**niteliğinin değerinin **true** olması bu işlemi sürekli kılacaktır. Burada diğer örneklerden farklı olarak animasyon işleminin tetiklendiği olay olarak **Path.Loaded** seçilmiştir. Buna göre Path, söz konusu pencere(Window) üzerine yüklendikten sonra animasyon işlemi doğrudan başlayacaktır. Örneğin çalışma zamanındaki görüntüsü aşağıdaki Flash görselinde olduğu gibidir.

Aynı işleyişi kod tarafında gerçekleştirmek istersek aşağıdaki satırları kullanmamız yeterli olacaktır.

XAML içeriği;

```
<Window x:Class="AnimasyonIslemleri.KodlaNoktaAnimasyon"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Kodla Nokta Animasyon
(PointAnimation)" Height="300" Width="300">
   <Grid>
      <Path Name="Daire" Fill="Red" Stroke="Black" StrokeThickness="1">
         <Path.Data>
            <EllipseGeometry x:Name="daireGeo" Center="15,15" RadiusX="10"
RadiusY="10" />
         </Path.Data>
      </Path>
   </Grid>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Windows.Media.Animation; // Animasyon tiplerini içeren isim alanıdır

namespace AnimasyonIslemleri
{
   public partial class KodlaNoktaAnimasyon : Window
   {
      private void AnimasyonuBaslat()
      {
         // Nokta animasyonu için gerekli tip oluşturulur
         // İlk parametre başlangıç noktası koordinatlarıdır From
         // İkinci parametre bitiş noktası koordinatlarıdır To
         // Üçündü parametre zaman çizelgesinin süresidir Duration
         PointAnimation pntAnmtr = new PointAnimation(new Point(15, 15), new
Point(285, 15), new Duration(new TimeSpan(0, 0, 0,1,75)));

         // Animasyonu kontrol ve özelliği ile ilişkilendirecek olan StoryBoard oluşturulur
         Storyboard strBrd = new Storyboard();
         // Animasyon tipi StoryBoard' a eklenir
         strBrd.Children.Add(pntAnmtr);
         // Animasyonun sonundan tekrardan geriye doğru gidileceği belirtilir
         strBrd.AutoReverse = true;
         // Animasyonun sürekli devam edeceği belirtilir
         strBrd.RepeatBehavior = RepeatBehavior.Forever;
         // Animasyonun uygulanacağı EllipsGeometry tipi seçilir. Buradaki ikinci
parametre XAML tarafındaki x:Name niteliğinin değeridir
         Storyboard.SetTargetName(pntAnmtr,"daireGeo");
         // Animasyonun uygulanacağı özellik seçilir.
         Storyboard.SetTargetProperty(pntAnmtr, new
PropertyPath(EllipseGeometry.CenterProperty));
         // Animasyonun, Daire isimli Path yüklendikten sonra başlatılması için Loaded olay
metodu yüklenir.
         Daire.Loaded += delegate(object sender, RoutedEventArgs e)
                         {
                            strBrd.Begin(this);
                         };
      }
      public KodlaNoktaAnimasyon()
      {
         InitializeComponent();
         AnimasyonuBaslat();
      }
   }
}
```

WPF içerisinde kullanılan **temel animasyon tipleri(Basic Animation Types)** sadece yazımızda bahsetiklerimiz ile sınırlı değildir. **System.Windows.Media.Animation** isim alanında(Namespace) yer alan diğer animasyon tiplerinin listesi aşağıdaki gibidir.

- ByteAnimation
- DecimalAnimation
- Int16Animation
- Int32Animation
- Int64Animation
- Point3DAnimation
- QuaternionAnimation
- Rotation3DAnimation
- RectAnimation
- SingleAnimation
- SizeAnimation
- TicknessAnimation
- Vector3DAnimation
- VectorAnimation

Görüldüğü gibi kontrollerin pek çok farklı tipteki özelliği için yazılmış temel animasyon tipleri vardır. Animasyon ile ilgili işlemler bu makalede ele aldıklarımız ile sınırlı değildir elbeteki. 3 boyutlu (3D) animasyon, KeyFrame' lerin kullanımı ve dahası da var. Animasyon işlemleri ile ilgili bu ilk yazımızda temel animasyon tiplerinin tanımaya ve onları anlamaya çalıştık. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

Örnek Uygulama için Tıklayın

WPF - Grafik İşlemlerinde Fırçaların(Brushes) Kullanımı (2007-09-18T20:08:00)

wpf,

Windows Presentation Foundation(WPF) ile ilgili bir önceki makalemizde, **iki boyutlu(2D)** grafiklerin çizilmesi amacıyla kullanılan fırçaları**(Brushes)** incelemeye çalışmıştık. Bu makalemizde ise iki boyutlu şekilleri**(Shapes)** araştırıyor olacağız. Vektörel grafiklerde şekillerin(Shapes) büyük önemi vardır. Nitekim temel şekiller kullanılarak asıl resimler ve görüntüler kolaylıkla elde edilebilir. Bir **CAD** uygulamasının karmaşık çizelgelerinden, eğlenceli çocuk programlarında kullanılan vektörel grafiklere kadar pek çok alanda temel şekiller yeterli olmaktadır. Söz gelimi bir şehrin imar planlamasında kullanılacak bir programda iki boyutlu olarak düşünüldüğünde dörtgenler, daireler, elipsler, poligonlar ve düz çizgiler evlerin, yolların, arsaların, parkların ifade edilmesi için yeterlidir. Senaryolar arttırılabilir ve daha geniş alanlarda düşünülebilir. Ancak temel olarak gereken şekiller bellidir. WPF kendi bünyesinde iki boyutlu çizimlerin gerçekleştirilebilmesi amacıyla aşağıda belirtilen şekilleri(Shapes) sunmaktadır.

- **Ellips** : Bu tip yardımıyla içi dolu veya boş tam daire yada elipslerin çizilmesi mümkündür.
- **Line** : Düz çizgilerin çizilmesini sağlayan tiptir. Başlangıç ve bitiş koordinatları düz çizginin çizilmesi için yeterlidir.
- **Rectangle** : Dört köşeli şekillerin çizilmesinde kullanılan tiptir. İçi boş veya dolu dikdörtgen yada kare gibi şekillerin çizilebilmesini sağlar.
- **Polygon** : N sayıda köşeden oluşan poligonların çizilmesinde kullanılır. Bir üçgen olabileceği gibi bir çokgen de olabilir. Diğer taraftan düzgün köşeli olmayan bir poligonda oluşturulabilir. Ayrıca poligonların içi boş veya dolu olacak şekilde oluşturulabilmesi de mümkündür.
- **Polyline** : Birbirlerine bitiş noktalarından bağlı bir başka deyişle uç uca eklenmiş düz çizgilerin(Line) çizilmesini sağlayan tiptir.
- **Path** : Birbirlerine son noktalarından bağlı olan düz çizgi veya eğri (Curve) gibi toplu şekillerin çizidirilmesini sağlayan tiptir. Farklı şekillerin bir arada kullanılabilmesini sağlamak için geometri**(Geometry)** tiplerinden yararlanır.

WPF, **XAML(eXtensible Application Markup Language)** tabanlı bir ortam sunduğundan, grafiksel şekillerin tasarım zamanında element bazlı olarak geliştirilmeleri ve sonuçlarının görülmesi mümkündür. **GDI+** mimarisinde aynı durum düşünüldüğünde sonuçların ancak çalışma zamanında**(run-time)** elde edilebildiği unutulmamalıdır. Bu nedenle WPF bize büyük avantaj sağlamaktadır.

Bu kısa bilgilerden sonra örneklerimizi geliştirerek şekilleri daha yakından tanımaya çalışalabiliriz. Her zamanki gibi örneklerimizi geliştirirken **Visual Studio 2008 Beta 2** sürümünden yararlanıyor olacağız. Bu nedenle, final sürümünde özellikle IDE bazlı bazı değişiklikler olma ihtimali olduğunu baştan belirtelim. İlk örneğimizde **Ellips** tipinden yararlanıyor olacağız. Bu amaçla Window nesnemizin XAML içeriğini aşağıdaki gibi tasarladığımızı düşünelim.

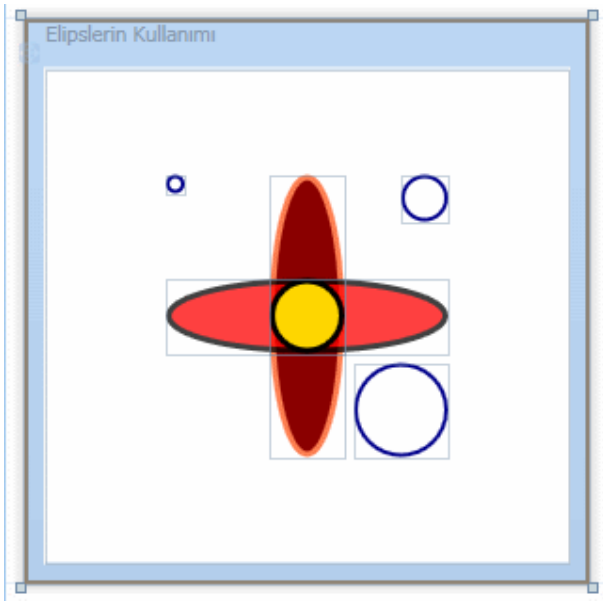
```
<Window x:Class="GrafiklerleCalismak.Elipsler"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Elipslerin Kullanımı" Height="300" Width="300">
   <Grid>
      <Ellipse Fill="DarkRed" Width="40" Height="150" Stroke="Coral" StrokeThickn
ess="3"/>
      <Ellipse Fill="Red" Width="150" Height="40" Stroke="Black"
StrokeThickness="3" Opacity="0.75"/>
      <Ellipse Width="40" Height="40" Fill="Gold" Stroke="Black"
StrokeThickness="3"/>
      <Ellipse Width="10" Height="10" Stroke="DarkBlue" StrokeThickness="2"
HorizontalAlignment="Left" Margin="64,56,0,0" VerticalAlignment="Top" />
      <Ellipse Height="25" HorizontalAlignment="Right" Margin="0,56,64,0"
Stroke="DarkBlue" StrokeThickness="2" VerticalAlignment="Top" Width="25" />
      <Ellipse Height="50" HorizontalAlignment="Right" Margin="0,0,64,56"
```

```
Stroke="DarkBlue" StrokeThickness="2" VerticalAlignment="Bottom" Width="50" />
    </Grid>
</Window>
```

Bu örnekte altı adet elips çizdirilmektedir. Ellips tipinin **Fill** niteliği**(attribute)** yardımıyla dolgu deseni belirtilebilir. Bunun dışında width ve height nitelikleri eşit oldukları takdirde tam bir dairenin çizilmesi söz konusudur. Diğer hallerde ise, yatay doğrultuda veya dikey doğrultuda uzayan bir elips oluşumu söz konusu olmaktadır. Kenar çizgilerini renk ve kalınlık olarak belirlemek amacıyla**Stroke** ve **StrokeTickness** niteliklerine değer atamaları yapılmaktadır. Stroke niteliği tahmin edileceği üzere geçerli bir fırça**(Brush)** ile eşleştirilebilir. Buda çizginin dolu bir renk dışında desenli olabileceği hatta içinde resim barındırabileceği anlamına gelmektedir.

NOT : *Stroke ve StrokeTickness nitelikleri diğer şekillerde de ye almaktadır. Bu nedenle tüm şekillerin çizgilerinin olabileceğini söyleyebiliriz.*

Yukarıdaki XAML içeriğini Visual Studio 2008 Beta 2 ortamındaki çıktısı aşağıdaki gibi olacaktır.



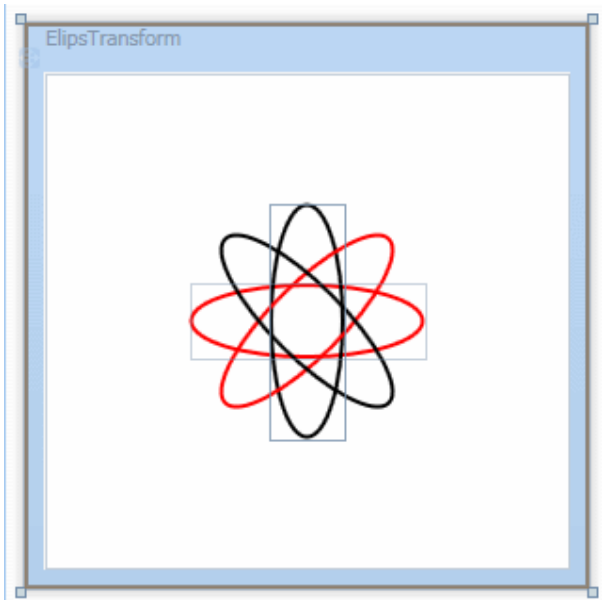
Şekillerin bu biçimde yatay veya dikey düzlemlerde oluşturulması haricinde, açısal olaraktanda yerleştirilmesi istenebilir. Bunu bir elips üzerinde GDI+ ile gerçekleştirmek oldukça zor ve zahmetlidir. Oysaki WPF içerisinde yer alan Transform tipleri kullanılarak bu işlemler son derece kolay bir şekilde gerçekleştirilebilir.

NOT : Transoformâ€™ *dan kasıt, şeklin açısal olarak konumunun değiştirilebilmesi, büyüklüğünün ayarlanabilmesi, kendi ekseni üzerinde veya farklı bir orjine göre döndürülebilmesi, şeklin x veya y düzlemleri üzerinde yer değiştirmesi yada eğilip bükülmesi gibi aksiyonlardır.*

İkinci örneğimizde bu durum incelenmektedir. Yeni penceremizin **XAML** içeriğinin aşağıdaki gibi olduğunu düşünelim.

```
<Window x:Class="GrafiklerleCalismak.ElipsTransform"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="ElipsTransform" Height="300" Width="300" Loaded="Window_Loaded">
   <Grid>
      <Ellipse Width="125" Height="40" Stroke="Red" StrokeThickness="2"/>
      <Ellipse Width="40" Height="125" Stroke="Black" StrokeThickness="2"/>
      <Ellipse Width="40" Height="125" Stroke="Red" StrokeThickness="2">
         <Ellipse.LayoutTransform>
            <RotateTransform Angle="45"/>
         </Ellipse.LayoutTransform>
      </Ellipse>
      <Ellipse Width="40" Height="125" Stroke="Black" StrokeThickness="2">
         <Ellipse.LayoutTransform>
            <RotateTransform Angle="135"/>
         </Ellipse.LayoutTransform>
      </Ellipse>
   </Grid>
</Window>
```

Burada dikkat edilmesi gereken en önemli nokta, **Ellips.LayoutTransform** elementinin içeriğidir. Bu elementin altında yer alan **RotateTransform** elementi içerisinde **Angle** niteliği ile bir açı değeri belirtilmektedir. Bu açı değeri, şeklin x,y eksenine göre farklı bir derecede döndürülmesini sağlamaktadır. Örnek XAML içeriğinde 45 derece ve 135 derecelik açılar ile döndürülmüş iki elips yer almaktadır. Bu içeriğin tasarım zamanındaki çıktısı aşağıdaki gibi olacaktır.



Atomu WPF ile daha kolay çizebildiğimizi söyleyebiliriz. Bu tip dönüştürme(**Transform**) işlemleri sadece **RotateTransform** ile sınırlı değildir. Yazımızın ilerleyen kısımlarında diğer Transform modellerinede kısaca değinmeye çalışacağız. Çok doğal olarak rotasyonların programatik olarak gerçekleştirilmesi gereken vakkalar olacaktır. Yukarıdaki atom çizelgesininin benzerini kod tarafında oluşturmak istersek, element ve niteliklerin karşılığı olan uygun sınıf**(class)** ve özellikleri**(property)** kullanmak yeterli olacaktır. Üçüncü örneğimizde bu durum incelenmektedir. Bu amaçla yeni penceremizin XAML ve kod içeriğini aşağıdaki gibi tasarladığımızı düşünelim.

XAML içeriği;

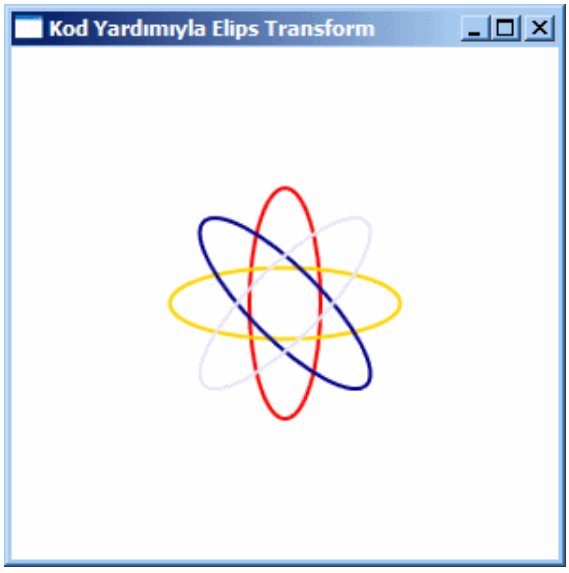
```
<Window x:Class="GrafiklerleCalismak.KodlaElipsTransform"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Kod Yardımıyla Elips Transform" Height="300" Width="300">
   <Grid Name="grdEllips">
   </Grid>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace GrafiklerleCalismak
{
   public partial class KodlaElipsTransform : Window
   {
      private void ElipsCiz(Ellipse elps, int width, int height, int angle,Color color)
      {
         elps.Width = width;
         elps.Height = height;
         elps.Stroke = new SolidColorBrush(color);
         elps.StrokeThickness = 2;
         elps.LayoutTransform = new RotateTransform(angle);
```

```
            grdEllips.Children.Add(elps);
        }
        private void Cizdir()
        {
            ElipsCiz(new Ellipse(), 125, 40, 270,Colors.Red);
            ElipsCiz(new Ellipse(), 40, 125, 90,Colors.Gold);
            ElipsCiz(new Ellipse(), 125, 40, 45, Colors.DarkBlue);
            ElipsCiz(new Ellipse(), 125, 40, 135,Colors.Lavender);
        }

        public KodlaElipsTransform()
        {
            InitializeComponent();
            Cizdir();
        }
    }
}
```

Bir kaçtane elipsi farklı renk, çizgi, çizgi kalınlığı ve açıda çizdirmek istediğimizden yardımcı olacak **ElipsCiz** isimli bir metod tasarlanmıştır. Bu metod, parametre olarak gelen**Ellips** nesne örneğini alıp, genişlik(**Width**), yükseklik(**Height**), Çizgi rengi(**Stroke**), Çizgi kalınlığı(**StrokTickness**) ve rotasyon için gerekli açı(**Angle**) değerlerini set etmektedir. Dikkat edilecek olursa, rotasyon işlemi için **RotateTransform** tipine ait bir nesne örneklenmekte ve yapıcı metoda(**Constructor**) açı değeri verilmektedir. Sonrasında ise bu nesne örneği, Ellips nesne örneğinin **LayoutTransform** özelliğine atanmaktadır. Söz konusu kod parçası yürütüldüğünde çalışma zamanında(run-time) aşağıdakine benzer bir ekran görtüntüsü ile karşılaşılır.



Yine kod yardımıyla elips çizdirmeye örnek olması açısından aşağıdaki pencerede(**Window**) göz önüne alınabilir.

XAML içeriği;

```
<Window x:Class="GrafiklerleCalismak.KodYardimiylaElips"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="KodYardimiylaElips" Height="300" Width="300">
   <Grid>
      <Grid Name="grdTahta" Margin="0,74,0,8" Background="Gold" />
      <Button Height="23" HorizontalAlignment="Left"
Margin="10,20,0,0" Name="btnCiz" VerticalAlignment="Top"
Width="75" Click="btnCiz_Click">Çizdir</Button>
   </Grid>
</Window>
```

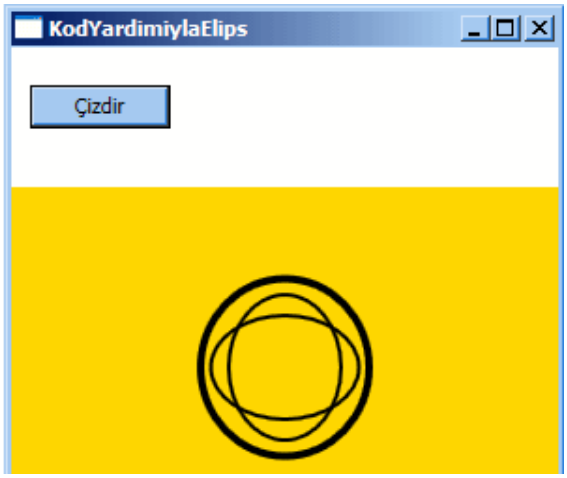
Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace GrafiklerleCalismak
{
   public partial class KodYardimiylaElips : Window
   {
      private void ElipsCiz()
      {
         grdTahta.Children.Clear();
         Random rnd=new Random();
         for (int i = 0; i < 3; i++)
         {
            Ellipse elps = new Ellipse();
            elps.Width = rnd.Next(50, 100);
            elps.Height = rnd.Next(50, 100);
```

```
            elps.Stroke = new SolidColorBrush(Colors.Black);
            elps.StrokeThickness = rnd.Next(1, 5);
            grdTahta.Children.Add(elps);
        }
    }

    public KodYardimiylaElips()
    {
        InitializeComponent();
    }

    private void btnCiz_Click(object sender, RoutedEventArgs e)
    {
        ElipsCiz();
    }
  }
}
```

Bu kez bir düğmeye basılması ile rastgele üretilen değerlere göre elipslerin çizdirilmesi sağlanmaktadır. Bu amaçla 50 ile 100 arasında rastgele genişlik ve yükselik değerleri elde edebilmek için meşhur **Random** sınıfına ait nesne örneğinden ve **Next** metodundan yararlanılmaktadır. Çoğu zaman oygun programlamada şekilsel olarak bazı oyun karakterlerinin saha üzerinde rastgele konumlarda çıkması istenebilir. İşte bu noktada Random sınıfına ait metodlar ve WPF ile gelen yeni şekil çizme teknikleri işimizi oldukça kolaylaştırmaktadır.

Kod yardımıyla gerçekleştirilen örneklerde dikkat edilmesi gereken noktalardan birisi, oluşturulan şekillerin mutlaka bir taşıyıcıya eklenmiş olmalarıdır. Söz gelimi yukarıdaki örneklerde Ellips nesne örnekleri Grid bileşenine alt element olarak, **Children** özelliğinin **Add** metodundan yararlanılarak eklenmektedir. Son pencereye(**Window**) ilişkin olarak aşağıdaki ekran görüntüsünde çalışma zamanında**(run-time)** oluşabilecek bir örnek çıktı yer almaktadır.

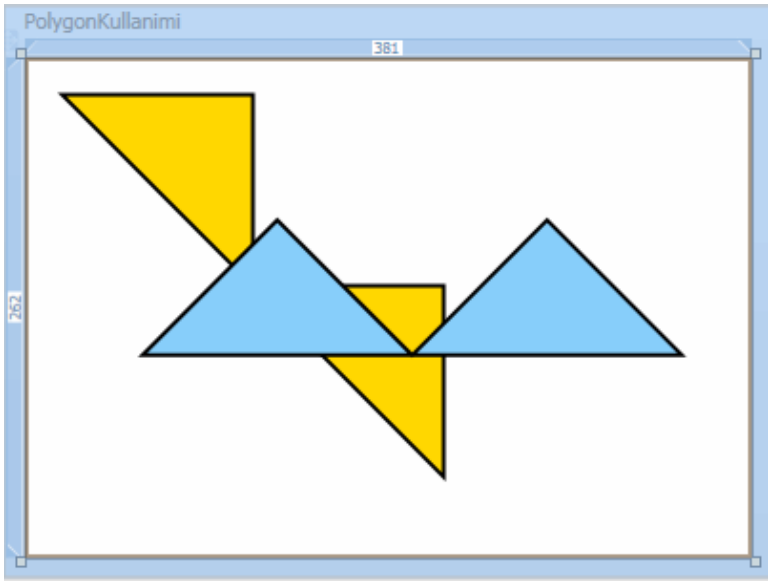


Sıradaki örneğimizde poligonların nasıl çizilebileceğini incelemeye çalışacağız. Poligonlar, çok sayıda köşeden oluşabilen ve kapalı olarak tasarlanabilen şekillerdir. Poligonlar sayesinde basit bir üçgen çizilebileceği gibi bir onaltıgenâ€™ de çizilebilir. Yada düzensiz bir kapalı şekil oluşturulabilir. Burada önemli olan köşe noktalarının belirlenmesidir. Köşe noktalarının belirlenmesinde **Point** tipinden yararlanılır. Point tipi x ve y koordinatlarını bünyesinde taşımaktadır. **Polygon** tipi, köşe noktalarını **Points** isimli bir koleksiyonda taşımaktadır. Şimdi aşağıdaki **XAML** içeriğini göz önüne alalım.

```
<Window x:Class="GrafiklerleCalismak.PolygonKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Polygon Kullanımı" Height="300" Width="403">
  <Grid>
    <Polygon Fill="Gold" FillRule="Nonzero" Stroke="Black"
StrokeThickness="2">
      <Polygon.Points>
        <Point X="20" Y="20" />
        <Point X="120" Y="20" />
        <Point X="120" Y="120" />
        <Point X="220" Y="120" />
        <Point X="220" Y="220" />
      </Polygon.Points>
    </Polygon>
    <Polygon Fill="LightSkyBlue" FillRule="Nonzero" Stroke="Black"
StrokeThickness="2">
      <Polygon.Points>
        <Point X="20" Y="20"/>
        <Point X="120" Y="20"/>
        <Point X="120" Y="120"/>
        <Point X="220" Y="120"/>
        <Point X="220" Y="220"/>
      </Polygon.Points>
      <Polygon.LayoutTransform>
        <RotateTransform Angle="-45"/>
      </Polygon.LayoutTransform>
    </Polygon>
  </Grid>
</Window>
```

Örnekte iki adet **Polygon** elementi tanımlanmıştır. Bunlardan ikincisine -45 derecelik bir açısal döndürme işlemi uygulanmıştır. Her iki Polygon nesne örneğinin köşe noktaları**Polygon.Points** alt elementi**(child element)** içerisinde yer alan **Point** alt elementleri ile belirtilmektedir. Polygon nesnelerinin kenar çizgileri **Stroke** ve **StrokTickness** niteliklerine atanan değerler yardımıyla set edilmiştir. Diğer taraftan Polygonâ€™ların dolgu rengi **Fill** özelliklerine atanan standart

fırça**(Brush)** renkleri ile ayarlanmaktadır. Söz konusu XAML içeriğinin Visual Studio 2008 Beta 2 ortamındaki tasarım zamanı**(design-time)** çıktısı ise aşağıdaki gibi olacaktır.



Bir poligon kod yardımıylada oluşturulabilir. Sıradaki örneğimizde bir üçgenin kod yardımıyla oluşturulması ve düğmeye basılaraktan onbeşer derecelik artan açılar ile döndürülmesi örneklenmektedir. Bu amaçla yeni penceremize(Window) ait XAML ve kod içeriklerini aşağıdaki gibi tasarladığımızı düşünelim.

XAML içeriği;

```
<Window x:Class="GrafiklerleCalismak.KodYardimiylaPolygonKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Kod Yardımıyla Polygon" Height="300" Width="300">
   <Grid Name="grdTahta">
      <Button Height="23" HorizontalAlignment="Left" Margin="8,19,0,0"
Name="button1" VerticalAlignment="Top"
Width="75" Click="button1_Click">Çevir</Button>
   </Grid>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
```

```
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Timers;

namespace GrafiklerleCalismak
{
   public partial class KodYardimiylaPolygonKullanimi : Window
   {
      Polygon plgy;
      int sayac = 1;

      private void Cizdir()
      {
         plgy = new Polygon();
         plgy.Points.Add(new Point(50, 50));
         plgy.Points.Add(new Point(150, 50));
         plgy.Points.Add(new Point(150, 150));
         plgy.Stroke = new SolidColorBrush(Colors.LightSalmon);
         plgy.StrokeThickness = 2;
         grdTahta.Children.Add(plgy);
      }

      public KodYardimiylaPolygonKullanimi()
      {
         InitializeComponent();
         Cizdir();
      }

      private void button1_Click(object sender, RoutedEventArgs e)
      {
         plgy.LayoutTransform = new RotateTransform(sayac*15);
         sayac++;
      }
   }
}
```

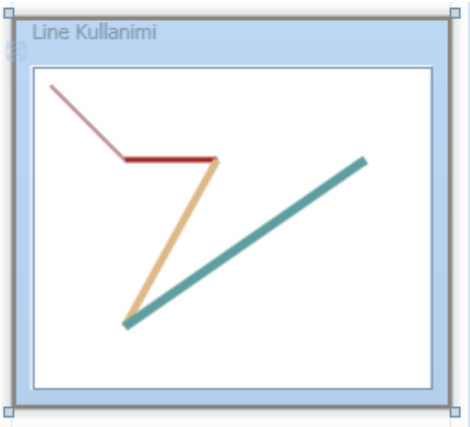
Penceremizin yapıcı metodu(**constructor**) içerisinde Cizdir metodu ile bir **Polygon** nesne örneği oluşturulmakta ve **Grid** kontrolünün alt elementi olarak eklenmektedir. Polygon nesne örneği bir üçgeni temsil edeceğinden **Points** koleksiyonunda sadece üç nokta(**Point**) eklemesi yapılmıştır. Döndürme işleminin gerçekleştirildiği yer düğmenin **Click** olay**(event)** metodudur. Burada dikkat edilecek olursa yine **LayoutTransform** özelliğine bir değer ataması yapılmaktadır. Bu atama sırasında

yeni bir **RotateTransform** nesnesi örneklenmekte ve parametre olarak artan bir açı değer verilmektedir. Uygulama test edildiğinde çalışma zamanında aşağıdaki Flash animasyonunda yer alan çıktı elde edilecektir. *(Flash dosyasını görebilmek için Flasy Playerâ€™ın sisteminizde yüklü olması gerekmektedir.)*

Şimdiki örneğimizde düz çizgileri nasıl çizdirebileceğimizi incelemeye çalışacağız. Düz çizgi için **Line** tipi kullanılmaktadır. Bir çizgi için belkide en önemli özellikler **Stroke**,**StrokeTickness**, **X1, X2, Y1** ve **Y2**â€™ dir. X ve Y özellikleri yardımıyla çizginin başlangıç ve bitiş noktaları belirlenir. Örneğin aşağıdaki XAML içeriğini ele alalım.

```
<Window x:Class="GrafiklerleCalismak.LineKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Line Kullanimi" Height="211" Width="237">
   <Grid>
      <Line Stroke="RosyBrown" StrokeThickness="2" X1="10" Y1="10" X2="50"
Y2="50"/>
      <Line Stroke="Brown" StrokeThickness="3" X1="50" Y1="50" X2="100"
Y2="50"/>
      <Line Stroke="BurlyWood" StrokeThickness="4" X1="100" Y1="50" X2="50"
Y2="140"/>
      <Line Stroke="CadetBlue" StrokeThickness="5" X1="50" Y1="140" X2="180"
Y2="50"/>
   </Grid>
</Window>
```

Bu içeriğin tasarım zamanındaki(design-time) çıktısı aşağıdaki gibi olacaktır.



Dikkat edileceği üzere farklı kalınlık, renk ve lokasyonlarda yer alan çizgiler elde edilmektedir. Çizgiler özellikle kod tarafındada zaman zaman ele alınırlar. Söz gelimi bir harita üzerinden bir şehirden bir şehire doğru olabilecek hava yolu rotalarının belirlenmesi

amacıyla çizgilerden yararlanılabilir. Bunu çok basit olarak sembolize etmek amacıyla aşağıdaki XAML ve kod içeriğini göz önüne alabiliriz.

XAML içeriği;

```
<Window x:Class="GrafiklerleCalismak.KodYardimiylaLineKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Kod Yardımıyla Line Kullanımı" Height="249" Width="422"
WindowStyle="SingleBorderWindow">
   <Grid Name="grdTahta">
      <Image Name="imgHarita" MouseDown="imgHarita_MouseDown"
MouseUp="imgHarita_MouseUp" Source="map_world_destination.gif" />
   </Grid>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace GrafiklerleCalismak
{
   public partial class KodYardimiylaLineKullanimi : Window
   {
      Point basilanNokta;

      public KodYardimiylaLineKullanimi()
      {
         InitializeComponent();
      }

      private void imgHarita_MouseDown(object sender, MouseButtonEventArgs e)
      {
         basilanNokta=e.GetPosition(grdTahta);
```

```
        }

        private void imgHarita_MouseUp(object sender, MouseButtonEventArgs e)
        {
            Line yol = new Line();
            yol.Stroke = new SolidColorBrush(Colors.Red);
            yol.StrokeThickness = 2;
            yol.X1 = basilanNokta.X;
            yol.Y1 = basilanNokta.Y;
            Point bitisNoktasi = e.GetPosition(grdTahta);
            yol.X2 = bitisNoktasi.X;
            yol.Y2 = bitisNoktasi.Y;
            grdTahta.Children.Add(yol);
        }
    }
}
```

Bu örnekte haritayı göstermesi için bir **Image** kontrolü kullanılmaktadır. Image kontrolünün **source** özelliğine atanan değer ile arka plan dünya haritası olarak gösterilmektedir. Kullanıcılar çalışma zamanında mouseâ€™ un tuşuna basıp bir noktadan başka bir noktaya gittiklerinde ve mouseâ€™ un tuşunu bıraktıklarında **Line** nesne örneği oluşturulmaktadır. Bunun için Image kontrolü üzerinde mouse tuşuna basılma ve bırakılma anlarının yakalanması gerekir. Söz konusu anlar aşina olduğumuz **MouseDown** ve **MouseUp** olay metodlarında yakalanırlar. GDI+ ile aynı işlemleri nasıl yaptığımızı hatırlarsak eğer, kontrolün MouseDown olayına gelen parametre ile X ve Y değerlerini ayır ayrı aldığımızı görürüz. Burada durum biraz daha farklıdır. Nitekim MouseUp ve MouseDown olay metodlarında yer alan **MouseButtonEventArgs** parametresi, mouse tuşuna basıldığında o noktadaki X ve Y koordinatlarını yeni bir **Point** tipi olarak geriye döndürmektedir. **GetPosition** metodu X ve Y koordinatları, üzerinden alınmak istenen kontrolüde parametre olarak alır. Örnekte bu parametreye**Grid** kontrolünün referansı verilmiştir. Dolayısıyla Grid kontrolündeki X ve Y değerleri elde edilebilmektedir.

Mouse üzerinde basılan tuş bırakıldığında ise çizginin çizilme işlemi gerçekleştirilmektedir. **Line** nesne örneğinin **X1** ve **Y1** değerleri tahmin edileceği gibi, **MouseDown** içerisinde yer alan **GetPosition** ile elde edilen basilanNokta değişkeninden gelmektedir. Çizginin son noktası ise bu kez **MouseUp** olay metodu içerisindeki **GetPosition** çağrısı ile alınmakta ve **Line** nesne örneğinin **X2** ve **Y2** değerlerine aktarılmaktadır. Son olarak oluşturulan çizgi, **Grid** bileşenine alt element olarak eklenmektedir. Bu ekleme işleminin bize getirdiği önemli bir avantaj vardır. Yine GDI+ ile Windows programlama yaptığımızı düşünecek olursak, aynı senaryoda ekrana çizgiler çizdirmek için **Graphics** nesnesninin **DrawLine** metodundan yararlanıldığını görürüz. Ne varki bu metod hep son çizginin çizilmesine öncekilerin ise kaybolmasına neden olmaktadır. Oysaki WPF mimarisinde şekiller bir taşıyıcının alt elementi olarak eklendiklerinden son çizilen

şekilden öncekiler ekrandan kaybolmamaktadır. Bu durumu **GDI**+ ile Windows programlamada gerçeklemek için WPFâ€™tekine benzer bir mantık ile hareket edilmekte ve ekranda duran çizgilerin sürekli hatırlanması için koleksiyonalardan yararlanılması gerekmektedir. Yukarıdaki kodun çalışma zamanındaki ekran çıktısı aşağıdaki Flash animasyonunda olduğu gibidir.*(Flash dosyasını görebilmek için Flasy Playerâ€™ın sisteminizde yüklü olması gerekmektedir.)*

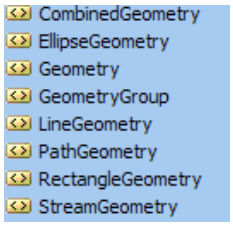
Bu örneği daha da geliştirmeye çalışmanızı öneririm. Oldukça fazla eksiği var. Söz gelimi, mouse tuşuna basıp sürüklerken farklı renkte bir çizginin çıkartılarak gidilen rotanın gösterilmesi sağlanabilir. Mouse bırakıldığında ise asıl rengini alan rota ortaya çıkar. Üstelik örnek kodda sağ tuşa veya sol tuş kontrolü yapılmamıştır. Belkide klavyeden tuş kombinasyonları katarak sadece düz çizgi değil eğrilerin çizdirilmesinide sağlayabiliriz. Bu örneğin geliştirilmesini siz değerli okurlarıma bırakıyorum.

Gelelim **Path** bileşenine. Bu tip aslında kendi içerisinde birden fazla düz çizgi veya eğriyi barındırabilecek şekilde tasarlanmıştır. Temelde şekiller birbirleriyle uç uca eklenecen şekilde yeni bir grafik oluşturmaktadırlar. Dilerseniz örnek üzerinden ilerleyerek konuyu daha iyi anlamaya çalışalım. Bu amaçla aşağıdaki gibi bir XAML içeriği oluşturduğumuzu göz önüne alalım.

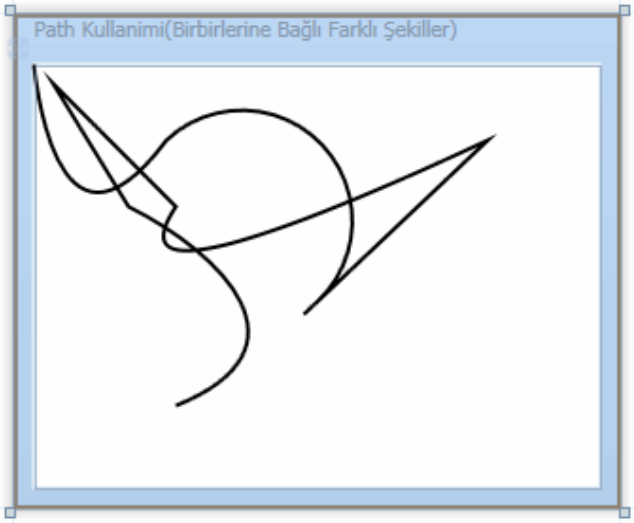
```
<Window x:Class="GrafiklerleCalismak.PathKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Path Kullanimi(Birbirlerine Bağlı Farklı Şekiller)" Height="261" Width="321">
  <Grid>
    <Path Stroke="Black" StrokeThickness="2">
      <Path.Data>
        <PathGeometry>
          <PathFigure>
            <BezierSegment Point1="0,0" Point2="10,120" Point3="70,40"/>
            <ArcSegment SweepDirection="Clockwise" Point="150,125" Size="50,50"
IsLargeArc="True" RotationAngle="60"/>
            <LineSegment Point="240,40"/>
            <QuadraticBezierSegment Point1="35,135" Point2="75,75"/>
            <PolyLineSegment>
              <PolyLineSegment.Points>
                <Point X="10" Y="10"/>
                <Point X="50" Y="75"/>
              </PolyLineSegment.Points>
            </PolyLineSegment>
            <PolyBezierSegment>
              <PolyBezierSegment.Points>
                <Point X="100" Y="100"/>
                <Point X="150" Y="150"/>
                <Point X="75" Y="180"/>
```

```
                </PolyBezierSegment.Points>
            </PolyBezierSegment>
        </PathFigure>
    </PathGeometry>
</Path.Data>
</Path>
</Grid>
</Window>
```

Path elementi içerisinde farklı şekillerin uç uca eklenmesi işini geometri tipleri üstlenmektedir. Bu geometri tipleride kendi içlerinde segmentler halinde şekilleri barındırmaktadır. Kullanılabilecek olan geometri tipleri aşağıdaki gibidir.



Örnekte söz konusu geometri tiplerinden **PathGeometry** kullanılmaktadır. PathGeometry elementinin altında yer alan tüm şekiller **Segment** anahtar kelimesi ile bitmektedir. Buna göre, **BezierSegment** ile başlayan şekiller dizisi sırasıyla, **ArcSegment**, **LineSegment**, **QuadraticBezierSegment**, **PolyLineSegment** ve **PolyBezierSegment** ile devam eder. Tüm bu alt elementleri farklı nitelikleri ile değişik çizgilerin oluşturulması sağlanmaktadır. **BezierSegment** elementinin nitekikleri sayesinde üç noktadan bükülmüş bir eğri çizimi yapılabilmektedir. **ArcSegment** elementinin nitelikleri ile, başlangıç koordiantları, genişlik yükseklik değerleri, eğilme açısı ve saat yönü yada saatin ters yönünde çizilecek yaylar oluşturulabilmektedir. Burada**PolyLineSegment**â€™ i altında belirtilen noktalar uç uca bağlı düz çizgilerin oluşturulmasını sağlarken, **PolyBezierSegment** elementi altındaki noktalarda, uç uca eklenmiş eğrilerin oluşturulmasını sağlamaktadır. Örneğin tasarım zamanındaki çıktısı aşağıdaki gibi olacaktır.

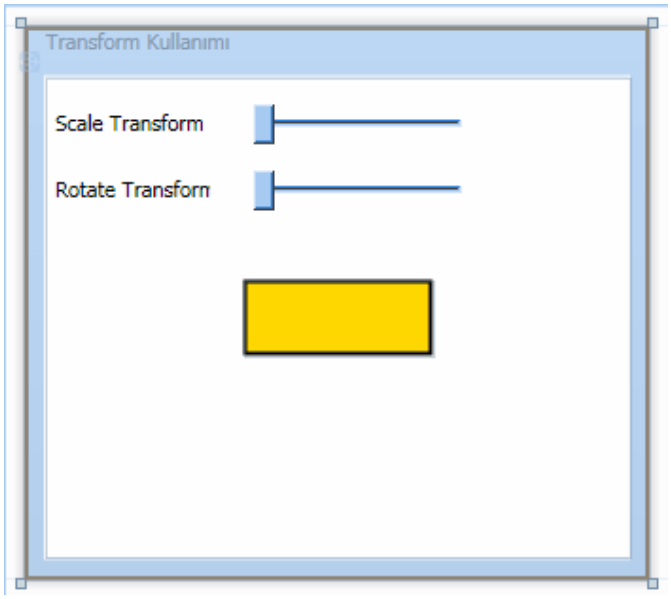


Dikkat edilecek olursa, tüm çizgiler ister eğri ister düz olsunlar, uç uca eklenerekten birleştirilmiştir. Bu nedenle Path tipini örneğin harita gibi arka planlarda yolların birleştirilmesi amacıyla kullanabiliriz.

Bu yazımızda son olarak basit anlamda dönüştürme**(Transform)** işlemlerine bakıyor olacağız. Transform denilince aklımıza gelmesi gerekenler bir şeklin yön, büyüklük, düzlemsel koordinat gibi değerlerinin değiştirilmesidir. Bu sayede bir şekli herhangibir açıda döndürebilir, ebatlarını ayarlayabilir yada herhangibir düzlem üzerinde öteleyebiliriz. Transform işlemlerinde beş farklı tip rol oynamaktadır. Bu tipler aşağıdaki gibidir.

- **RotateTransform** : Şeklin belirtilen bir açıda, kendi ekseninde yada belirtilen orijine göre farklı bir eksende döndürülmesini sağlamak için kullanılır. Söz gelimi bir şeklin farklı açılardan gösterilmesinin sağlanmasında önemli rol oynamaktadır.
- **ScaleTransform** : Şeklin ebatlarının eşit oranda yada farklı oranlarda arttırılması yada azaltılmasında kullanılır. Örneğin Zoom işlemlerinde bu tip çok faydalı olacaktır.
- **SkewTransform** : Şeklin bükülmesini yada eğilmesini sağlamak amacıyla kullanılan tiptir.
- **TranslateTransform** : Şeklin x veya y düzlemleri üzerinde farklı noktalara ötelenmesi amacıyla kullanılan tiptir.
- **MatrixTransform** : Resim işlemede önemli bir yere sahip olan matris algoritmalarının iki boyutlu şekiller üzerinde de uygulanabilmesini sağlayan tiptir. Diğer Transform tipleri ile gerçekleştirilmesi zor olan dönüştürmelerde kullanılmaktadır. Bu tipi ilerleyen yazılarımızda ele almaya çalışacağız.

Şimdi basit bir örnek ile **RotateTransform** ve **ScaleTransform** tiplerinin nasıl kullanılabileceğini incelemeye çalışalım. Bu amaçla XAML ve kod içeriklerini aşağıdaki gibi geliştirdiğimizi düşünelim.



XAML içeriği;

```
<Window x:Class="GrafiklerleCalismak.TransformKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Transform Kullanımı" Height="292" Width="330">
   <Grid Name="grdTahta" Width="309" Height="253">
      <Rectangle Fill="Gold" Stroke="Black" StrokeThickness="2"
Name="dortgen" Width="100" Height="40" />
      <Slider Minimum="1" Maximum="5"
ValueChanged="sldScale_ValueChanged" Height="21"
Margin="110,14,89,0" Name="sldScale" VerticalAlignment="Top" />
      <Label Height="23" HorizontalAlignment="Left" Margin="0,12,0,0" Name="label1"
VerticalAlignment="Top" Width="92">Scale Transform</Label>
      <Slider Height="21" Margin="110,49,89,0" Maximum="360" Minimum="0"
Name="sldRotate" ValueChanged="sldRotate_ValueChanged"
VerticalAlignment="Top" />
      <Label Height="23" HorizontalAlignment="Left" Margin="0,47,0,0" Name="label2"
VerticalAlignment="Top" Width="92">Rotate Transform</Label>
   </Grid>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
```

```
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace GrafiklerleCalismak
{
   public partial class TransformKullanimi : Window
   {
      public TransformKullanimi()
      {
         InitializeComponent();
      }

      private void sldScale_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
      {
         TransformYap();
      }

      private void sldRotate_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
      {
         TransformYap();
      }

      private void TransformYap()
      {
         if (sldRotate != null
            && sldScale != null)
         {
            TransformGroup grp = new TransformGroup();
            grp.Children.Add(new ScaleTransform(sldScale.Value, sldScale.Value));
            grp.Children.Add(new RotateTransform(sldRotate.Value));
            dortgen.LayoutTransform = grp;
         }
      }
   }
}
```

Kullanıcı Slider kontrollerindeki çubuğu hareket ettirdikçe ekran üzerindeki dörtgenin kendi ekseni üzerinde dönmesi veya büyüklüğünün değişmesi amaçlanmaktadır. Bu, aynı

şekle birden fazla dönüştürme işleminin uygulanmasını gerektirmektedir. Bir başka deyişle **Rectangle** nesne örneğinin **LayoutTransform** özelliğine hem **ScaleTransform** hemde**RotateTransform** nesne örneklerinin atanması gerekmektedir. Bunun için **TransformGroup** adı verilen nesne örneklerinden yararlanılır. TransformGroup tipi, sahip olduğu **Children** özelliği ile sunduğu koleksiyon içerisinde farklı **Transform** tiplerini taşıyabilmektedir. Dolayısıyla TransformYap metodu içerisinde bu şekilde bir kodlama yapılmaktadır. Uygulamanın çalışma zamanındaki görüntüsü aşağıdaki Flash animasyonunda olduğu gibidir.*(Flash dosyasını görebilmek için Flasy Playerâ€™ın sisteminizde yüklü olması gerekmektedir. Dosya boyutu 220 Kb olduğundan yüklenmesi zaman alabilir.)*

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde WPF(Windows Presentation Foundation) uygulamalarında iki boyutlu grafik**(2D Graphics)** işlemlerinde kullanılabilecek şekilleri ele almaya çalıştık. Son olarak basit bir transform işleminin örnek bir dörtgen üzerinde nasıl gerçekleştirileceğine değindik. Buradaki bilgilerden yola çıkarak çok daha kolay grafik işlemleri gerçekleştirebileceğimizi ve eski Windows programlamadaki **GDI+** ile zorlandığımız vakkaları daha etkin bir biçimde yapabileceğimizi görmüş bulunuyoruz. İlerleyen makalelerimizde WPF ile ilişkili başka konularada değiniyor olacağız.Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Örnek Uygulama için Tıklayın

WPF - Grafik İşlemlerinde Şekillerin(Shapes) Kullanımı (2007-09-18T20:02:00)

wpf,

Windows Presentation Foundation(WPF) ile ilgili bir önceki makalemizde, **iki boyutlu(2D)** grafiklerin çizilmesi amacıyla kullanılan fırçaları**(Brushes)** incelemeye çalışmıştık. Bu makalemizde ise iki boyutlu şekilleri**(Shapes)** araştırıyor olacağız. Vektörel grafiklerde şekillerin(Shapes) büyük önemi vardır. Nitekim temel şekiller kullanılarak asıl resimler ve görüntüler kolaylıkla elde edilebilir. Bir **CAD** uygulamasının karmaşık çizelgelerinden, eğlenceli çocuk programlarında kullanılan vektörel grafiklere kadar pek çok alanda temel şekiller yeterli olmaktadır. Söz gelimi bir şehrin imar planlamasında kullanılacak bir programda iki boyutlu olarak düşünüldüğünde dörtgenler, daireler, elipsler, poligonlar ve düz çizgiler evlerin, yolların, arsaların, parkların ifade edilmesi için yeterlidir. Senaryolar arttırılabilir ve daha geniş alanlarda düşünülebilir. Ancak temel olarak gereken şekiller bellidir. WPF kendi bünyesinde iki boyutlu çizimlerin gerçekleştirilebilmesi amacıyla aşağıda belirtilen şekilleri(Shapes) sunmaktadır.

- **Ellips** : Bu tip yardımıyla içi dolu veya boş tam daire yada elipslerin çizilmesi mümkündür.
- **Line** : Düz çizgilerin çizilmesini sağlayan tiptir. Başlangıç ve bitiş koordinatları düz çizginin çizilmesi için yeterlidir.

- **Rectangle** : Dört köşeli şekillerin çizilmesinde kullanılan tiptir. İçi boş veya dolu dikdörtgen yada kare gibi şekillerin çizilebilmesini sağlar.
- **Polygon** : N sayıda köşeden oluşan poligonların çizilmesinde kullanılır. Bir üçgen olabileceği gibi bir çokgen de olabilir. Diğer taraftan düzgün köşeli olmayan bir poligonda oluşturulabilir. Ayrıca poligonların içi boş veya dolu olacak şekilde oluşturulabilmesi de mümkündür.
- **Polyline** : Birbirlerine bitiş noktalarından bağlı bir başka deyişle uç uca eklenmiş düz çizgilerin(Line) çizilmesini sağlayan tiptir.
- **Path** : Birbirlerine son noktalarından bağlı olan düz çizgi veya eğri (Curve) gibi toplu şekillerin çizidirilmesini sağlayan tiptir. Farklı şekillerin bir arada kullanılabilmesini sağlamak için geometri**(Geometry)** tiplerinden yararlanır.

WPF, **XAML(eXtensible Application Markup Language)** tabanlı bir ortam sunduğundan, grafiksel şekillerin tasarım zamanında element bazlı olarak geliştirilmeleri ve sonuçlarının görülmesi mümkündür. **GDI+** mimarisinde aynı durum düşünüldüğünde sonuçların ancak çalışma zamanında**(run-time)** elde edilebildiği unutulmamalıdır. Bu nedenle WPF bize büyük avantaj sağlamaktadır.

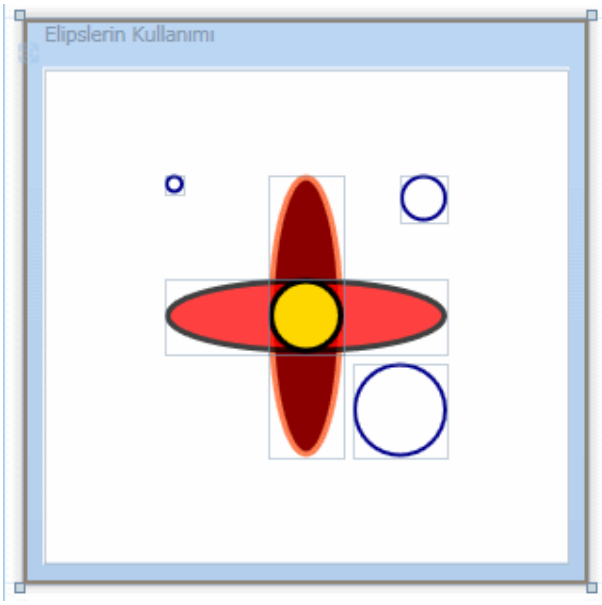
Bu kısa bilgilerden sonra örneklerimizi geliştirerek şekilleri daha yakından tanımaya çalışalabiliriz. Her zamanki gibi örneklerimizi geliştirirken **Visual Studio 2008 Beta 2** sürümünden yararlanıyor olacağız. Bu nedenle, final sürümünde özellikle IDE bazlı bazı değişiklikler olma ihtimali olduğunu baştan belirtelim. İlk örneğimizde **Ellips** tipinden yararlanıyor olacağız. Bu amaçla Window nesnemizin XAML içeriğini aşağıdaki gibi tasarladığımızı düşünelim.

```
<Window x:Class="GrafiklerleCalismak.Elipsler"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Elipslerin Kullanımı" Height="300" Width="300">
  <Grid>
    <Ellipse Fill="DarkRed" Width="40" Height="150" Stroke="Coral" StrokeThickn
ess="3"/>
    <Ellipse Fill="Red" Width="150" Height="40" Stroke="Black"
StrokeThickness="3" Opacity="0.75"/>
    <Ellipse Width="40" Height="40" Fill="Gold" Stroke="Black"
StrokeThickness="3"/>
    <Ellipse Width="10" Height="10" Stroke="DarkBlue" StrokeThickness="2"
HorizontalAlignment="Left" Margin="64,56,0,0" VerticalAlignment="Top" />
    <Ellipse Height="25" HorizontalAlignment="Right" Margin="0,56,64,0"
Stroke="DarkBlue" StrokeThickness="2" VerticalAlignment="Top" Width="25" />
    <Ellipse Height="50" HorizontalAlignment="Right" Margin="0,0,64,56"
Stroke="DarkBlue" StrokeThickness="2" VerticalAlignment="Bottom" Width="50" />
  </Grid>
</Window>
```

Bu örnekte altı adet elips çizdirilmektedir. Ellips tipinin **Fill** niteliği**(attribute)** yardımıyla dolgu deseni belirtilebilir. Bunun dışında width ve height nitelikleri eşit oldukları takdirde tam bir dairenin çizilmesi söz konusudur. Diğer hallerde ise, yatay doğrultuda veya dikey doğrultuda uzayan bir elips oluşumu söz konusu olmaktadır. Kenar çizgilerini renk ve kalınlık olarak belirlemek amacıyla**Stroke** ve **StrokeTickness** niteliklerine değer atamaları yapılmaktadır. Stroke niteliği tahmin edileceği üzere geçerli bir fırça**(Brush)** ile eşleştirilebilir. Buda çizginin dolu bir renk dışında desenli olabileceği hatta içinde resim barındırabileceği anlamına gelmektedir.

NOT : *Stroke ve StrokeTickness nitelikleri diğer şekillerde de ye almaktadır. Bu nedenle tüm şekillerin çizgilerinin olabileceğini söyleyebiliriz.*

Yukarıdaki XAML içeriğini Visual Studio 2008 Beta 2 ortamındaki çıktısı aşağıdaki gibi olacaktır.



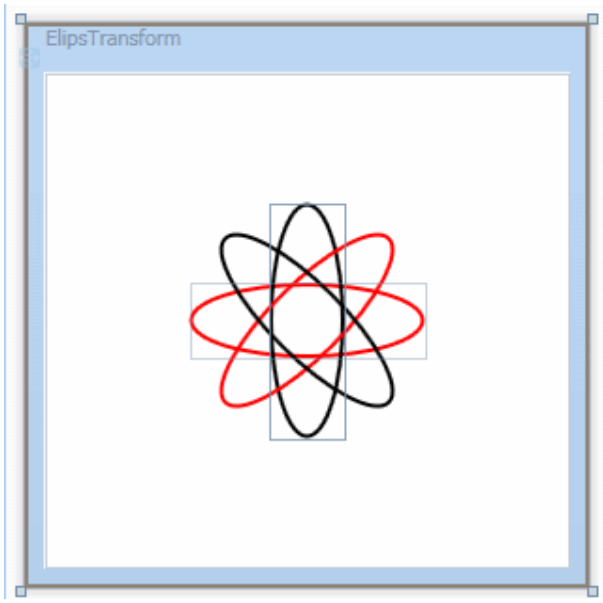
Şekillerin bu biçimde yatay veya dikey düzlemlerde oluşturulması haricinde, açısal olaraktanda yerleştirilmesi istenebilir. Bunu bir elips üzerinde GDI+ ile gerçekleştirmek oldukça zor ve zahmetlidir. Oysaki WPF içerisinde yer alan Transform tipleri kullanılarak bu işlemler son derece kolay bir şekilde gerçekleştirilebilir.

NOT : Transoformâ€™ *dan kasıt, şeklin açısal olarak konumunun değiştirilebilmesi, büyüklüğünün ayarlanabilmesi, kendi ekseni üzerinde veya farklı bir orjine göre döndürülebilmesi, şeklin x veya y düzlemleri üzerinde yer değiştirmesi yada eğilip bükülmesi gibi aksiyonlardır.*

İkinci örneğimizde bu durum incelenmektedir. Yeni penceremizin **XAML** içeriğinin aşağıdaki gibi olduğunu düşünelim.

```
<Window x:Class="GrafiklerleCalismak.ElipsTransform"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="ElipsTransform" Height="300" Width="300" Loaded="Window_Loaded">
   <Grid>
      <Ellipse Width="125" Height="40" Stroke="Red" StrokeThickness="2"/>
      <Ellipse Width="40" Height="125" Stroke="Black" StrokeThickness="2"/>
      <Ellipse Width="40" Height="125" Stroke="Red" StrokeThickness="2">
         <Ellipse.LayoutTransform>
            <RotateTransform Angle="45"/>
         </Ellipse.LayoutTransform>
      </Ellipse>
      <Ellipse Width="40" Height="125" Stroke="Black" StrokeThickness="2">
         <Ellipse.LayoutTransform>
            <RotateTransform Angle="135"/>
         </Ellipse.LayoutTransform>
      </Ellipse>
   </Grid>
</Window>
```

Burada dikkat edilmesi gereken en önemli nokta, **Ellips.LayoutTransform** elementinin içeriğidir. Bu elementin altında yer alan **RotateTransform** elementi içerisinde **Angle** niteliği ile bir açı değeri belirtilmektedir. Bu açı değeri, şeklin x,y eksenine göre farklı bir derecede döndürülmesini sağlamaktadır. Örnek XAML içeriğinde 45 derece ve 135 derecelik açılar ile döndürülmüş iki elips yer almaktadır. Bu içeriğin tasarım zamanındaki çıktısı aşağıdaki gibi olacaktır.



Atomu WPF ile daha kolay çizebildiğimizi söyleyebiliriz. Bu tip dönüştürme(**Transform**) işlemleri sadece **RotateTransform** ile sınırlı değildir. Yazımızın ilerleyen kısımlarında

diğer Transform modellerinede kısaca değinmeye çalışacağız. Çok doğal olarak rotasyonların programatik olarak gerçekleştirilmesi gereken vakkalar olacaktır. Yukarıdaki atom çizelgesininin benzerini kod tarafında oluşturmak istersek, element ve niteliklerin karşılığı olan uygun sınıf**(class)** ve özellikleri**(property)** kullanmak yeterli olacaktır. Üçüncü örneğimizde bu durum incelenmektedir. Bu amaçla yeni penceremizin XAML ve kod içeriğini aşağıdaki gibi tasarladığımızı düşünelim.

XAML içeriği;

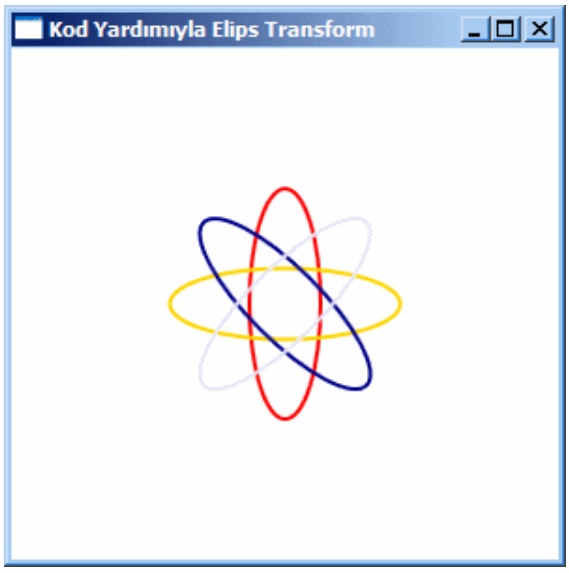
```
<Window x:Class="GrafiklerleCalismak.KodlaElipsTransform"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Kod Yardımıyla Elips Transform" Height="300" Width="300">
   <Grid Name="grdEllips">
   </Grid>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace GrafiklerleCalismak
{
   public partial class KodlaElipsTransform : Window
   {
      private void ElipsCiz(Ellipse elps, int width, int height, int angle,Color color)
      {
         elps.Width = width;
         elps.Height = height;
         elps.Stroke = new SolidColorBrush(color);
         elps.StrokeThickness = 2;
         elps.LayoutTransform = new RotateTransform(angle);
         grdEllips.Children.Add(elps);
      }
```

```
        private void Cizdir()
        {
            ElipsCiz(new Ellipse(), 125, 40, 270,Colors.Red);
            ElipsCiz(new Ellipse(), 40, 125, 90,Colors.Gold);
            ElipsCiz(new Ellipse(), 125, 40, 45, Colors.DarkBlue);
            ElipsCiz(new Ellipse(), 125, 40, 135,Colors.Lavender);
        }

        public KodlaElipsTransform()
        {
            InitializeComponent();
            Cizdir();
        }
    }
}
```

Bir kaçtane elipsi farklı renk, çizgi, çizgi kalınlığı ve açıda çizdirmek istediğimizden yardımcı olacak **ElipsCiz** isimli bir metod tasarlanmıştır. Bu metod, parametre olarak gelen**Ellips** nesne örneğini alıp, genişlik(**Width**), yükseklik(**Height**), Çizgi rengi(**Stroke**), Çizgi kalınlığı(**StrokTickness**) ve rotasyon için gerekli açı(**Angle**) değerlerini set etmektedir. Dikkat edilecek olursa, rotasyon işlemi için **RotateTransform** tipine ait bir nesne örneklenmekte ve yapıcı metoda(**Constructor**) açı değeri verilmektedir. Sonrasında ise bu nesne örneği, Ellips nesne örneğinin **LayoutTransform** özelliğine atanmaktadır. Söz konusu kod parçası yürütüldüğünde çalışma zamanında(run-time) aşağıdakine benzer bir ekran görtüntüsü ile karşılaşılır.



Yine kod yardımıyla elips çizdirmeye örnek olması açısından aşağıdaki pencerede(**Window**) göz önüne alınabilir.

XAML içeriği;

```
<Window x:Class="GrafiklerleCalismak.KodYardimiylaElips"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="KodYardimiylaElips" Height="300" Width="300">
   <Grid>
      <Grid Name="grdTahta" Margin="0,74,0,8" Background="Gold" />
      <Button Height="23" HorizontalAlignment="Left"
Margin="10,20,0,0" Name="btnCiz" VerticalAlignment="Top"
Width="75" Click="btnCiz_Click">Çizdir</Button>
   </Grid>
</Window>
```

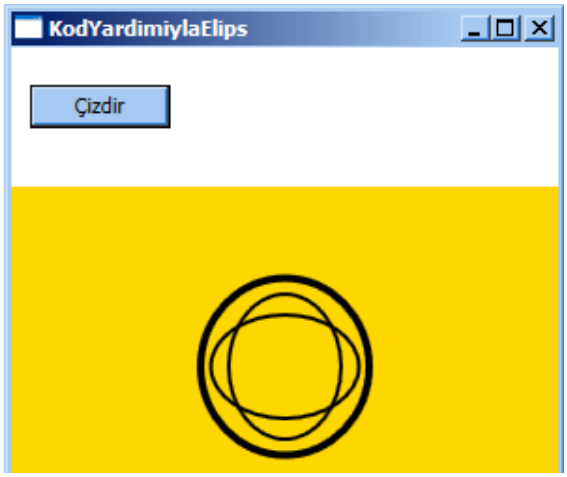
Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace GrafiklerleCalismak
{
   public partial class KodYardimiylaElips : Window
   {
      private void ElipsCiz()
      {
         grdTahta.Children.Clear();
         Random rnd=new Random();
         for (int i = 0; i < 3; i++)
         {
            Ellipse elps = new Ellipse();
            elps.Width = rnd.Next(50, 100);
            elps.Height = rnd.Next(50, 100);
            elps.Stroke = new SolidColorBrush(Colors.Black);
            elps.StrokeThickness = rnd.Next(1, 5);
            grdTahta.Children.Add(elps);
         }
      }
```

```
        public KodYardimiylaElips()
        {
            InitializeComponent();
        }

        private void btnCiz_Click(object sender, RoutedEventArgs e)
        {
            ElipsCiz();
        }
    }
}
```

Bu kez bir düğmeye basılması ile rastgele üretilen değerlere göre elipslerin çizdirilmesi sağlanmaktadır. Bu amaçla 50 ile 100 arasında rastgele genişlik ve yükselik değerleri elde edebilmek için meşhur **Random** sınıfına ait nesne örneğinden ve **Next** metodundan yararlanılmaktadır. Çoğu zaman oygun programlamada şekilsel olarak bazı oyun karakterlerinin saha üzerinde rastgele konumlarda çıkması istenebilir. İşte bu noktada Random sınıfına ait metodlar ve WPF ile gelen yeni şekil çizme teknikleri işimizi oldukça kolaylaştırmaktadır.

Kod yardımıyla gerçekleştirilen örneklerde dikkat edilmesi gereken noktalardan birisi, oluşturulan şekillerin mutlaka bir taşıyıcıya eklenmiş olmalarıdır. Söz gelimi yukarıdaki örneklerde Ellips nesne örnekleri Grid bileşenine alt element olarak, **Children** özelliğinin **Add** metodundan yararlanılarak eklenmektedir. Son pencereye(**Window**) ilişkin olarak aşağıdaki ekran görüntüsünde çalışma zamanında**(run-time)** oluşabilecek bir örnek çıktı yer almaktadır.

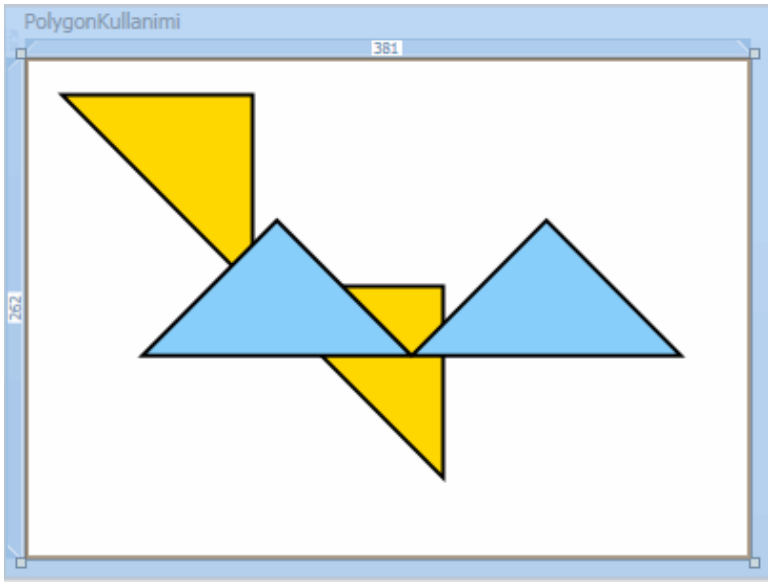


Sıradaki örneğimizde poligonların nasıl çizilebileceğini incelemeye çalışacağız. Poligonlar, çok sayıda köşeden oluşabilen ve kapalı olarak tasarlanabilen şekillerdir. Poligonlar sayesinde basit bir üçgen çizilebileceği gibi bir onaltıgenâ€™ de çizilebilir. Yada düzensiz bir kapalı şekil oluşturulabilir. Burada önemli olan köşe noktalarının belirlenmesidir. Köşe

noktalarının belirlenmesinde **Point** tipinden yararlanılır. Point tipi x ve y koordinatlarını bünyesinde taşımaktadır. **Polygon** tipi, köşe noktalarını **Points** isimli bir koleksiyonda taşımaktadır. Şimdi aşağıdaki **XAML** içeriğini göz önüne alalım.

```
<Window x:Class="GrafiklerleCalismak.PolygonKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Polygon Kullanımı" Height="300" Width="403">
  <Grid>
    <Polygon Fill="Gold" FillRule="Nonzero" Stroke="Black"
StrokeThickness="2">
      <Polygon.Points>
        <Point X="20" Y="20" />
        <Point X="120" Y="20" />
        <Point X="120" Y="120" />
        <Point X="220" Y="120" />
        <Point X="220" Y="220" />
      </Polygon.Points>
    </Polygon>
    <Polygon Fill="LightSkyBlue" FillRule="Nonzero" Stroke="Black"
StrokeThickness="2">
      <Polygon.Points>
        <Point X="20" Y="20"/>
        <Point X="120" Y="20"/>
        <Point X="120" Y="120"/>
        <Point X="220" Y="120"/>
        <Point X="220" Y="220"/>
      </Polygon.Points>
      <Polygon.LayoutTransform>
        <RotateTransform Angle="-45"/>
      </Polygon.LayoutTransform>
    </Polygon>
  </Grid>
</Window>
```

Örnekte iki adet **Polygon** elementi tanımlanmıştır. Bunlardan ikincisine -45 derecelik bir açısal döndürme işlemi uygulanmıştır. Her iki Polygon nesne örneğinin köşe noktaları**Polygon.Points** alt elementi**(child element)** içerisinde yer alan **Point** alt elementleri ile belirtilmektedir. Polygon nesnelerinin kenar çizgileri **Stroke** ve **StrokTickness** niteliklerine atanan değerler yardımıyla set edilmiştir. Diğer taraftan Polygonâ€™ların dolgu rengi **Fill** özelliklerine atanan standart fırça**(Brush)** renkleri ile ayarlanmaktadır. Söz konusu XAML içeriğinin Visual Studio 2008 Beta 2 ortamındaki tasarım zamanı**(design-time)** çıktısı ise aşağıdaki gibi olacaktır.



Bir poligon kod yardımıylada oluşturulabilir. Sıradaki örneğimizde bir üçgenin kod yardımıyla oluşturulması ve düğmeye basılaraktan onbeşer derecelik artan açılar ile döndürülmesi örneklenmektedir. Bu amaçla yeni penceremize(Window) ait XAML ve kod içeriklerini aşağıdaki gibi tasarladığımızı düşünelim.

XAML içeriği;

```
<Window x:Class="GrafiklerleCalismak.KodYardimiylaPolygonKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Kod Yardımıyla Polygon" Height="300" Width="300">
   <Grid Name="grdTahta">
      <Button Height="23" HorizontalAlignment="Left" Margin="8,19,0,0"
Name="button1" VerticalAlignment="Top"
Width="75" Click="button1_Click">Çevir</Button>
   </Grid>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
```

```
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Timers;

namespace GrafiklerleCalismak
{
   public partial class KodYardimiylaPolygonKullanimi : Window
   {
      Polygon plgy;
      int sayac = 1;

      private void Cizdir()
      {
         plgy = new Polygon();
         plgy.Points.Add(new Point(50, 50));
         plgy.Points.Add(new Point(150, 50));
         plgy.Points.Add(new Point(150, 150));
         plgy.Stroke = new SolidColorBrush(Colors.LightSalmon);
         plgy.StrokeThickness = 2;
         grdTahta.Children.Add(plgy);
      }

      public KodYardimiylaPolygonKullanimi()
      {
         InitializeComponent();
         Cizdir();
      }

      private void button1_Click(object sender, RoutedEventArgs e)
      {
         plgy.LayoutTransform = new RotateTransform(sayac*15);
         sayac++;
      }
   }
}
```

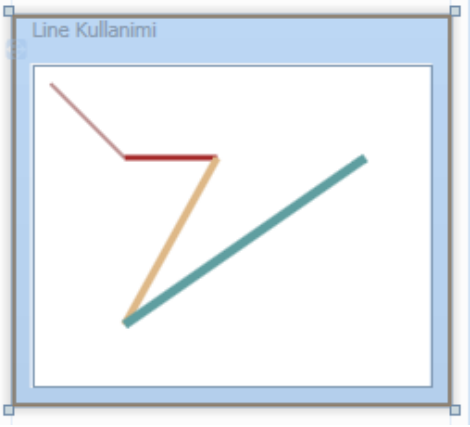
Penceremizin yapıcı metodu(**constructor**) içerisinde Cizdir metodu ile bir **Polygon** nesne örneği oluşturulmakta ve **Grid** kontrolünün alt elementi olarak eklenmektedir. Polygon nesne örneği bir üçgeni temsil edeceğinden **Points** koleksiyonunda sadece üç nokta(**Point**) eklemesi yapılmıştır. Döndürme işleminin gerçekleştirildiği yer düğmenin **Click** olay**(event)** metodudur. Burada dikkat edilecek olursa yine **LayoutTransform** özelliğine bir değer ataması yapılmaktadır. Bu atama sırasında yeni bir **RotateTransform** nesnesi örneklenmekte ve parametre olarak artan bir açı değer verilmektedir. Uygulama test edildiğinde çalışma zamanında aşağıdaki Flash

animasyonunda yer alan çıktı elde edilecektir. *(Flash dosyasını görebilmek için Flasy Playerâ€™ın sisteminizde yüklü olması gerekmektedir.)*

Şimdiki örneğimizde düz çizgileri nasıl çizdirebileceğimizi incelemeye çalışacağız. Düz çizgi için **Line** tipi kullanılmaktadır. Bir çizgi için belkide en önemli özellikler **Stroke**,**StrokeTickness**, **X1, X2, Y1** ve **Y2**â€™ dir. X ve Y özellikleri yardımıyla çizginin başlangıç ve bitiş noktaları belirlenir. Örneğin aşağıdaki XAML içeriğini ele alalım.

```
<Window x:Class="GrafiklerleCalismak.LineKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Line Kullanimi" Height="211" Width="237">
   <Grid>
      <Line Stroke="RosyBrown" StrokeThickness="2" X1="10" Y1="10" X2="50"
Y2="50"/>
      <Line Stroke="Brown" StrokeThickness="3" X1="50" Y1="50" X2="100"
Y2="50"/>
      <Line Stroke="BurlyWood" StrokeThickness="4" X1="100" Y1="50" X2="50"
Y2="140"/>
      <Line Stroke="CadetBlue" StrokeThickness="5" X1="50" Y1="140" X2="180"
Y2="50"/>
   </Grid>
</Window>
```

Bu içeriğin tasarım zamanındaki(design-time) çıktısı aşağıdaki gibi olacaktır.



Dikkat edileceği üzere farklı kalınlık, renk ve lokasyonlarda yer alan çizgiler elde edilmektedir. Çizgiler özellikle kod tarafındada zaman zaman ele alınırlar. Söz gelimi bir harita üzerinden bir şehirden bir şehire doğru olabilecek hava yolu rotalarının belirlenmesi amacıyla çizgilerden yararlanılabilir. Bunu çok basit olarak sembolize etmek amacıyla aşağıdaki XAML ve kod içeriğini göz önüne alabiliriz.

XAML içeriği;

```
<Window x:Class="GrafiklerleCalismak.KodYardimiylaLineKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Kod Yardımıyla Line Kullanımı" Height="249" Width="422"
WindowStyle="SingleBorderWindow">
   <Grid Name="grdTahta">
      <Image Name="imgHarita" MouseDown="imgHarita_MouseDown"
MouseUp="imgHarita_MouseUp" Source="map_world_destination.gif" />
   </Grid>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace GrafiklerleCalismak
{
   public partial class KodYardimiylaLineKullanimi : Window
   {
      Point basilanNokta;

      public KodYardimiylaLineKullanimi()
      {
         InitializeComponent();
      }

      private void imgHarita_MouseDown(object sender, MouseButtonEventArgs e)
      {
         basilanNokta=e.GetPosition(grdTahta);
      }

      private void imgHarita_MouseUp(object sender, MouseButtonEventArgs e)
```

```
        {
            Line yol = new Line();
            yol.Stroke = new SolidColorBrush(Colors.Red);
            yol.StrokeThickness = 2;
            yol.X1 = basilanNokta.X;
            yol.Y1 = basilanNokta.Y;
            Point bitisNoktasi = e.GetPosition(grdTahta);
            yol.X2 = bitisNoktasi.X;
            yol.Y2 = bitisNoktasi.Y;
            grdTahta.Children.Add(yol);
        }
    }
}
```

Bu örnekte haritayı göstermesi için bir **Image** kontrolü kullanılmaktadır. Image kontrolünün **source** özelliğine atanan değer ile arka plan dünya haritası olarak gösterilmektedir. Kullanıcılar çalışma zamanında mouseâ€™ un tuşuna basıp bir noktadan başka bir noktaya gittiklerinde ve mouseâ€™ un tuşunu bıraktıklarında **Line** nesne örneği oluşturulmaktadır. Bunun için Image kontrolü üzerinde mouse tuşuna basılma ve bırakılma anlarının yakalanması gerekir. Söz konusu anlar aşina olduğumuz **MouseDown** ve **MouseUp** olay metodlarında yakalanırlar. GDI+ ile aynı işlemleri nasıl yaptığımızı hatırlarsak eğer, kontrolün MouseDown olayına gelen parametre ile X ve Y değerlerini ayır ayrı aldığımızı görürüz. Burada durum biraz daha farklıdır. Nitekim MouseUp ve MouseDown olay metodlarında yer alan **MouseButtonEventArgs** parametresi, mouse tuşuna basıldığında o noktadaki X ve Y koordinatlarını yeni bir **Point** tipi olarak geriye döndürmektedir. **GetPosition** metodu X ve Y koordinatları, üzerinden alınmak istenen kontrolüde parametre olarak alır. Örnekte bu parametreye**Grid** kontrolünün referansı verilmiştir. Dolayısıyla Grid kontrolündeki X ve Y değerleri elde edilebilmektedir.

Mouse üzerinde basılan tuş bırakıldığında ise çizginin çizilme işlemi gerçekleştirilmektedir. **Line** nesne örneğinin **X1** ve **Y1** değerleri tahmin edileceği gibi, **MouseDown** içerisinde yer alan **GetPosition** ile elde edilen basilanNokta değişkeninden gelmektedir. Çizginin son noktası ise bu kez **MouseUp** olay metodu içerisindeki **GetPosition** çağrısı ile alınmakta ve **Line** nesne örneğinin **X2** ve **Y2** değerlerine aktarılmaktadır. Son olarak oluşturulan çizgi, **Grid** bileşenine alt element olarak eklenmektedir. Bu ekleme işleminin bize getirdiği önemli bir avantaj vardır. Yine GDI+ ile Windows programlama yaptığımızı düşünecek olursak, aynı senaryoda ekrana çizgiler çizdirmek için **Graphics** nesnesninin **DrawLine** metodundan yararlanıldığını görürüz. Ne varki bu metod hep son çizginin çizilmesine öncekilerin ise kaybolmasına neden olmaktadır. Oysaki WPF mimarisinde şekiller bir taşıyıcının alt elementi olarak eklendiklerinden son çizilen şekilden öncekiler ekrandan kaybolmamaktadır. Bu durumu **GDI**+ ile Windows programlamada gerçeklemek için WPFâ€™tekine benzer bir mantık ile hareket edilmekte ve ekranda duran çizgilerin sürekli hatırlanması için koleksiyonalardan yararlanılması

gerekmektedir. Yukarıdaki kodun çalışma zamanındaki ekran çıktısı aşağıdaki Flash animasyonunda olduğu gibidir.*(Flash dosyasını görebilmek için Flasy Playerâ€™ın sisteminizde yüklü olması gerekmektedir.)*

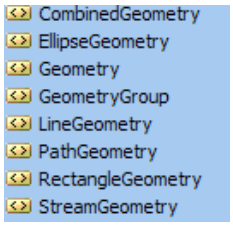
Bu örneği daha da geliştirmeye çalışmanızı öneririm. Oldukça fazla eksiği var. Söz gelimi, mouse tuşuna basıp sürüklerken farklı renkte bir çizginin çıkartılarak gidilen rotanın gösterilmesi sağlanabilir. Mouse bırakıldığında ise asıl rengini alan rota ortaya çıkar. Üstelik örnek kodda sağ tuşa veya sol tuş kontrolü yapılmamıştır. Belkide klavyeden tuş kombinasyonları katarak sadece düz çizgi değil eğrilerin çizdirilmesinide sağlayabiliriz. Bu örneğin geliştirilmesini siz değerli okurlarıma bırakıyorum.

Gelelim **Path** bileşenine. Bu tip aslında kendi içerisinde birden fazla düz çizgi veya eğriyi barındırabilecek şekilde tasarlanmıştır. Temelde şekiller birbirleriyle uç uca eklenecen şekilde yeni bir grafik oluşturmaktadırlar. Dilerseniz örnek üzerinden ilerleyerek konuyu daha iyi anlamaya çalışalım. Bu amaçla aşağıdaki gibi bir XAML içeriği oluşturduğumuzu göz önüne alalım.

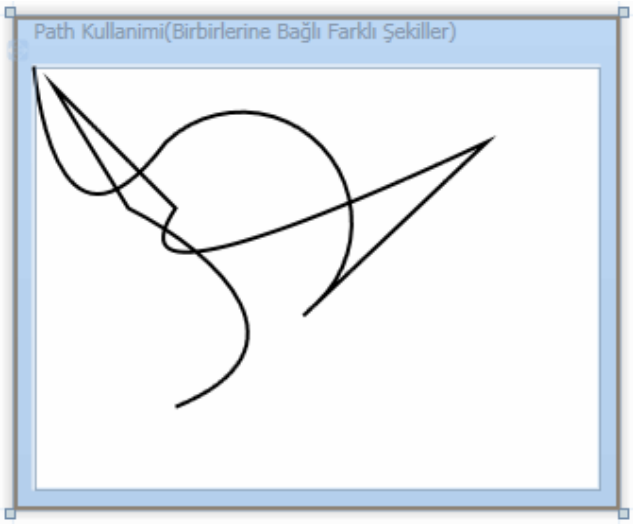
```
<Window x:Class="GrafiklerleCalismak.PathKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Path Kullanimi(Birbirlerine Bağlı Farklı Şekiller)" Height="261" Width="321">
  <Grid>
    <Path Stroke="Black" StrokeThickness="2">
      <Path.Data>
        <PathGeometry>
          <PathFigure>
            <BezierSegment Point1="0,0" Point2="10,120" Point3="70,40"/>
            <ArcSegment SweepDirection="Clockwise" Point="150,125" Size="50,50"
IsLargeArc="True" RotationAngle="60"/>
            <LineSegment Point="240,40"/>
            <QuadraticBezierSegment Point1="35,135" Point2="75,75"/>
            <PolyLineSegment>
              <PolyLineSegment.Points>
                <Point X="10" Y="10"/>
                <Point X="50" Y="75"/>
              </PolyLineSegment.Points>
            </PolyLineSegment>
            <PolyBezierSegment>
              <PolyBezierSegment.Points>
                <Point X="100" Y="100"/>
                <Point X="150" Y="150"/>
                <Point X="75" Y="180"/>
              </PolyBezierSegment.Points>
            </PolyBezierSegment>
          </PathFigure>
```

```
            </PathGeometry>
          </Path.Data>
        </Path>
    </Grid>
</Window>
```

Path elementi içerisinde farklı şekillerin uç uca eklenmesi işini geometri tipleri üstlenmektedir. Bu geometri tipleride kendi içlerinde segmentler halinde şekilleri barındırmaktadır. Kullanılabilecek olan geometri tipleri aşağıdaki gibidir.



Örnekte söz konusu geometri tiplerinden **PathGeometry** kullanılmaktadır. PathGeometry elementinin altında yer alan tüm şekiller **Segment** anahtar kelimesi ile bitmektedir. Buna göre, **BezierSegment** ile başlayan şekiller dizisi sırasıyla, **ArcSegment**, **LineSegment**, **QuadraticBezierSegment**, **PolyLineSegment** ve **PolyBezierSegment** ile devam eder. Tüm bu alt elementleri farklı nitelikleri ile değişik çizgilerin oluşturulması sağlanmaktadır. **BezierSegment** elementinin nitekikleri sayesinde üç noktadan bükülmüş bir eğri çizimi yapılabilmektedir. **ArcSegment** elementinin nitelikleri ile, başlangıç koordiantları, genişlik yükseklik değerleri, eğilme açısı ve saat yönü yada saatin ters yönünde çizilecek yaylar oluşturulabilmektedir. Burada**PolyLineSegment**â€™ i altında belirtilen noktalar uç uca bağlı düz çizgilerin oluşturulmasını sağlarken, **PolyBezierSegment** elementi altındaki noktalarda, uç uca eklenmiş eğrilerin oluşturulmasını sağlamaktadır. Örneğin tasarım zamanındaki çıktısı aşağıdaki gibi olacaktır.

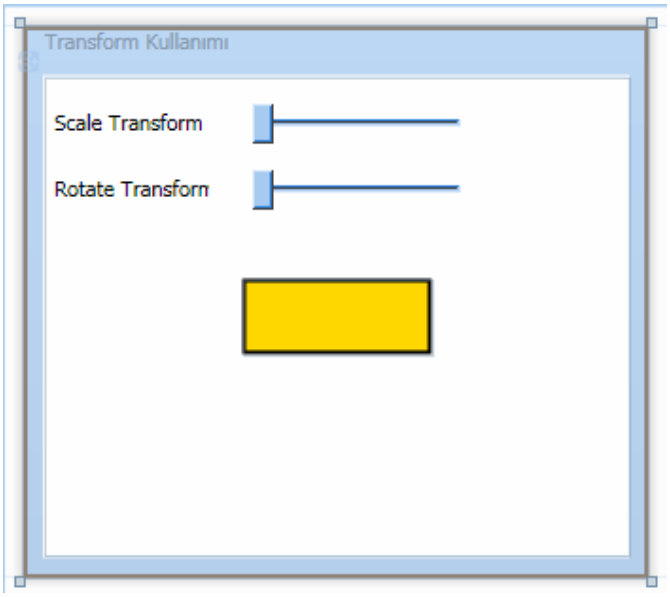


Dikkat edilecek olursa, tüm çizgiler ister eğri ister düz olsunlar, uç uca eklenerekten birleştirilmiştir. Bu nedenle Path tipini örneğin harita gibi arka planlarda yolların birleştirilmesi amacıyla kullanabiliriz.

Bu yazımızda son olarak basit anlamda dönüştürme**(Transform)** işlemlerine bakıyor olacağız. Transform denilince aklımıza gelmesi gerekenler bir şeklin yön, büyüklük, düzlemsel koordinat gibi değerlerinin değiştirilmesidir. Bu sayede bir şekli herhangibir açıda döndürebilir, ebatlarını ayarlayabilir yada herhangibir düzlem üzerinde öteleyebiliriz. Transform işlemlerinde beş farklı tip rol oynamaktadır. Bu tipler aşağıdaki gibidir.

- **RotateTransform** : Şeklin belirtilen bir açıda, kendi ekseninde yada belirtilen orijine göre farklı bir eksende döndürülmesini sağlamak için kullanılır. Söz gelimi bir şeklin farklı açılardan gösterilmesinin sağlanmasında önemli rol oynamaktadır.
- **ScaleTransform** : Şeklin ebatlarının eşit oranda yada farklı oranlarda arttırılması yada azaltılmasında kullanılır. Örneğin Zoom işlemlerinde bu tip çok faydalı olacaktır.
- **SkewTransform** : Şeklin bükülmesini yada eğilmesini sağlamak amacıyla kullanılan tiptir.
- **TranslateTransform** : Şeklin x veya y düzlemleri üzerinde farklı noktalara ötelenmesi amacıyla kullanılan tiptir.
- **MatrixTransform** : Resim işlemede önemli bir yere sahip olan matris algoritmalarının iki boyutlu şekiller üzerinde de uygulanabilmesini sağlayan tiptir. Diğer Transform tipleri ile gerçekleştirilmesi zor olan dönüştürmelerde kullanılmaktadır. Bu tipi ilerleyen yazılarımızda ele almaya çalışacağız.

Şimdi basit bir örnek ile **RotateTransform** ve **ScaleTransform** tiplerinin nasıl kullanılabileceğini incelemeye çalışalım. Bu amaçla XAML ve kod içeriklerini aşağıdaki gibi geliştirdiğimizi düşünelim.



XAML içeriği;

```
<Window x:Class="GrafiklerleCalismak.TransformKullanimi"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Background="White"
Title="Transform Kullanımı" Height="292" Width="330">
   <Grid Name="grdTahta" Width="309" Height="253">
      <Rectangle Fill="Gold" Stroke="Black" StrokeThickness="2"
Name="dortgen" Width="100" Height="40" />
      <Slider Minimum="1" Maximum="5"
ValueChanged="sldScale_ValueChanged" Height="21"
Margin="110,14,89,0" Name="sldScale" VerticalAlignment="Top" />
      <Label Height="23" HorizontalAlignment="Left" Margin="0,12,0,0" Name="label1"
VerticalAlignment="Top" Width="92">Scale Transform</Label>
      <Slider Height="21" Margin="110,49,89,0" Maximum="360" Minimum="0"
Name="sldRotate" ValueChanged="sldRotate_ValueChanged"
VerticalAlignment="Top" />
      <Label Height="23" HorizontalAlignment="Left" Margin="0,47,0,0" Name="label2"
VerticalAlignment="Top" Width="92">Rotate Transform</Label>
   </Grid>
</Window>
```

Kod içeriği;

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;

namespace GrafiklerleCalismak
{
   public partial class TransformKullanimi : Window
   {
      public TransformKullanimi()
      {
         InitializeComponent();
      }
```

```
        private void sldScale_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            TransformYap();
        }

        private void sldRotate_ValueChanged(object sender,
RoutedPropertyChangedEventArgs<double> e)
        {
            TransformYap();
        }

        private void TransformYap()
        {
            if (sldRotate != null
                && sldScale != null)
            {
                TransformGroup grp = new TransformGroup();
                grp.Children.Add(new ScaleTransform(sldScale.Value, sldScale.Value));
                grp.Children.Add(new RotateTransform(sldRotate.Value));
                dortgen.LayoutTransform = grp;
            }
        }
    }
}
```

Kullanıcı Slider kontrollerindeki çubuğu hareket ettirdikçe ekran üzerindeki dörtgenin kendi ekseni üzerinde dönmesi veya büyüklüğünün değişmesi amaçlanmaktadır. Bu, aynı şekle birden fazla dönüştürme işleminin uygulanmasını gerektirmektedir. Bir başka deyişle **Rectangle** nesne örneğinin **LayoutTransform** özelliğine hem **ScaleTransform** hemde**RotateTransform** nesne örneklerinin atanması gerekmektedir. Bunun için **TransformGroup** adı verilen nesne örneklerinden yararlanılır. TransformGroup tipi, sahip olduğu **Children** özelliği ile sunduğu koleksiyon içerisinde farklı **Transform** tiplerini taşıyabilmektedir. Dolayısıyla TransformYap metodu içerisinde bu şekilde bir kodlama yapılmaktadır. Uygulamanın çalışma zamanındaki görüntüsü aşağıdaki Flash animasyonunda olduğu gibidir.*(Flash dosyasını görebilmek için Flasy Playerâ€™ın sisteminizde yüklü olması gerekmektedir. Dosya boyutu 220 Kb olduğundan yüklenmesi zaman alabilir.)*

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde WPF(Windows Presentation Foundation) uygulamalarında iki boyutlu grafik**(2D Graphics)** işlemlerinde kullanılabilecek şekilleri ele almaya çalıştık. Son olarak basit bir transform işleminin örnek bir dörtgen üzerinde nasıl gerçekleştirileceğine değindik. Buradaki bilgilerden yola çıkarak çok daha kolay grafik işlemleri gerçekleştirebileceğimizi ve eski Windows

programlamadaki **GDI+** ile zorlandığımız vakkaları daha etkin bir biçimde yapabileceğimizi görmüş bulunuyoruz. İlerleyen makalelerimizde WPF ile ilişkili başka konularada değiniyor olacağız.Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Örnek Uygulama için Tıklayın

WPF - Veriye Bağlanmak(Data Binding) (2007-09-03T17:42:00)

wpf,

Bu gün geliştirdiğimiz programların çoğu veri**(Data)** ile ilişkili kaynakları kullanmaktadır. özellikle büyük ölçekli pek çok proje tipi içerisinde mutlaka verilerin kullanılması söz konusudur. Veriler kimi zaman müşteri bilgilerini, kimiz zaman ürün bilgilerini, kimi zamanda uygulamaya ait konfigurasyon bilgilerini vb... tutar. Verilerin çoğu zaman veritabanı sistemlerinde, fiziki dosyalarda veya program içerisindeki özel tiplerde saklandıklarını görürüz. çok doğal olarak bu veri depoları içerisinde tutulan bilgilerin son kullanıcılara gösterilmeside söz konusudur. Bu noktada, geliştirilen uygulamaya bakılmaksızın pek çok veri bağlama tekniği olduğunu söyleyebiliriz. Ama bu yazımızda özellike **Windows Presentation Foundation(WPF)** uygulamalarında veri bağlama işlemlerinin nasıl yapılabileceğini basit örneklerden hareket ederek incelemeye çalışıyor olacağız. Windows tabanlı programlamada özellikle **Visual Studio 2005** ile birlikte veri bağlama işlemlerinin dahada genişletildiğine şahit olduk. **Data Source** menüsü bunun en güzel örneklerinden birisidir. Hatta XML ve nesne(object) kaynaklarına daha kolay bağlanılmasını sağlayan **XmlDataSource** ve **ObjectDataSource** gibi kontrolleri gördük. Aslında veri bağlama**(Data Binding)** denildiği zaman akla gelmesi gereken; "bir tipin herhangibir üyesinin başka bir tipin sahip olduğu veriye otomatik olarak erişmesidir" diyebiliriz. Form üzerindeki bir metin kutusunun**(TextBox)** text özelliğinin, veritabanındaki bir alana bağlanması buna verilebilecek basit bir örnektir.

Ne varki WPF mimarisinde durum biraz farklı bir hal almıştır. özellikle .Net Framework 3.0 ve **LINQ**(Language Integrated Query) gibi yenilikler, verilerin işleniş ve ele alınış şekillerinide değiştirmektedir. WPF' e kısaca bakıldığında ilk fark edilen noktalardan birisi **.Net Framework 2.0** formlarındaki kadar çok kontrolün **Toolbox** sekmesine gelmediğidir. Dahası, Visual Studio 2005 uygulama geliştirme ortamından hatırladığımız pek çok veri kontrolü burada yer almamaktadır. üstelik bizleri bekleyen sayısız yeni kontrol bulunmaktadır. Peki bir WPF uygulamasında, pencere**(Window)** üzerindeki kontrollerin çeşitli özelliklerinin verilere olan bağlantılarını nasıl gerçekleştirebiliriz? İşte bu yazımızda araştıracağımız ve örnekleyeceğimiz konular bunlar olacaktır.

WPF uygulamlarında veri kaynaklarına bağlanabilmek amacıyla kullanılan iki temel sağlayıcı bulunmaktadır. Bunlardan birisi XML kaynaklarına bağlanma işlemini gerçekkleştirmemizi sağlayan **XmlDataProvider** bileşenidir. Diğeri ise, herhangibir .Net nesnesine (.Net Object) bağlanmamızı kolaylaştıran **ObjectDataProvider** bileşenidir. *(Bunların dışında özellikle bağlantısız*

*katman nesneleri ilede kullanılan **DataContext** özelliğide veri bağlama işlemlerinde kullanılmaktadır.)* Bu bileşenler sayesinde **XAML** içerisinden ilgili kaynaklara bağlanma, tek yönlü ve çift yönlü olarak veri transfer etme işlemlerini gerçekleştirebiliriz. Bu tiplerin kullanımını gösteren basit örneklerimize geçmeden önce veri bağlama işlemine basit ve genel bir bakış atmakta yarar var. Bu amaçla **Visual Studio 2008 Beta 2** sürümünde yeni bir WPF uygulaması açıyor ve Window1 penceresinin XAML içeriğini aşağıdaki gibi kodluyoruz.

```
<Window x:Class="DataBindIslemleri.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Bir kontrol niteliğini
başka bir kontrodekine bağlamak" Height="169" Width="275"
WindowStartupLocation="CenterScreen" Name="wndBasitBaglama">
   <Grid>
      <Button Background="#FFFFCC66" ClickMode="Release" Height="23"
HorizontalAlignment="Left" Margin="10,16,0,0" Name="btnGiris"
VerticalAlignment="Top" Width="75" Click="btnGiris_Click">Giriş</Button>
      <TextBox Background="Black" Foreground="{Binding ElementName=btnGiris,
Path=Background}" Margin="10,47,44,54" Name="txtSifre" />
   </Grid>
</Window>
```

Bu ilk örneğimizde penceremiz üzerinde bir **Button** ve birde **TextBox** kontrolümüz bulunmaktadır. Dikkat edilmesi gereken nokta TextBox kontrolünün ön plan renginin, Button kontrolünün arka plan rengine bağlanmış olmasıdır. Bunun için TextBox elementi içerisinde **Forground** niteliğine bir değer ataması gerçekleştirilmiştir. **Binding** ile başlayan bu ifade içerisinde**ElementName** isimli özellik verinin kaynağı olan nesneyi temsil etmektedir. Bu örnekte söz konusu nesne btnGiris isimli Button kontrolüdür. Foreground özelliğine bağlanmasını istediğimiz veri içeriği ise **Path** tanımlaması ile belirtilmektedir. Buna göre btnGiris isimli veri kaynağındaki **Background** isimli özelliğin değerinin atanması söz konusudur. Uygulamayı çalıştırıp test ettiğimizde örnek olarak aşağıdakine benzer bir görüntü elde ederiz.

Burada bahsedilen teknik en basit haliyle veri bağlamayı göstermektedir. Şimdi işi biraz daha ilerletip örnek bir **XML** dökümanı içerisinden, WPF kontrollerine veri bağlama işlemlerini nasıl gerçekleştirebileceğimize bakalım. Aşağıdaki gibi bir XML içeriğimiz olduğunu ve projemizde Urunler.xml adıyla kaydedildiğini göz önüne alabiliriz.

```
<?xml version="1.0" encoding="utf-8" ?>
<Depo>
  <Urun id="1000">
    <Ad>Ekran Kartı(VGA)</Ad>
    <BirimFiyat>35</BirimFiyat>
    <StokMiktari>100</StokMiktari>
    <Durum>OK.bmp</Durum>
    <Kategori>Yedek Parça</Kategori>
  </Urun>
  <Urun id="1001">
    <Ad>Intel Core Duo İşlemci (CPU)</Ad>
    <BirimFiyat>90</BirimFiyat>
    <StokMiktari>125</StokMiktari>
    <Durum>OK.bmp</Durum>
    <Kategori>Yedek Parça</Kategori>
  </Urun>
  <Urun id="1002">
    <Ad>17Inch LCD Monitor</Ad>
    <BirimFiyat>150</BirimFiyat>
    <StokMiktari>35</StokMiktari>
    <Durum>Warning.bmp</Durum>
    <Kategori>Ekran</Kategori>
  </Urun>
  <Urun id="1003">
    <Ad>250 GB Usb Harddisk</Ad>
    <BirimFiyat>150</BirimFiyat>
    <StokMiktari>90</StokMiktari>
    <Durum>Serious.bmp</Durum>
    <Kategori>Depolama Aygıtı</Kategori>
  </Urun>
  <Urun id="1004">
```

```
      <Ad>1 GB Usb Flash Bellek</Ad>
      <BirimFiyat>28</BirimFiyat>
      <StokMiktari>300</StokMiktari>
      <Durum>Warning.bmp</Durum>
      <Kategori>Depolama Aygıtı</Kategori>
   </Urun>
</Depo>
```

XML dökümanımız içerisinde çeşitli tipte bilgisayar ürünlerine ait bilgiler yer almaktadır. Temel olarak ürünün adı, birim fiyatı, stok miktarı , kategorisi ve durumuna ait bilgisi yer almaktadır. Söz gelimi bu XML veri kümesi içerisinde yer alan her bir ürünün adlarının bir **ComboBox** kontrolünde gösterilmesini istediğimizi düşünelim. Bu amaçla yine **Visual Studio 2008 Beta 2** üzerinde tasarladığımız WPF projemize yeni bir pencere**(Window)** ekliyor ve **XAML(eXtensible Application Markup Language)** içeriğini aşağıdaki gibi düzenliyoruz.

```
<Window x:Class="DataBindIslemleri.Window2"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="XmlDataProvider ile Xml verilerine basit bağlanmak" Height="150" Width="290" WindowStartupLocation="CenterScreen">
   <Grid>
      <Grid.Resources>
         <XmlDataProvider x:Key="UrunlerProvider" Source="Urunler.xml"/>
      </Grid.Resources>
      <ComboBox Height="28" Margin="23,42,60,0" Name="cmbUrunler"
VerticalAlignment="Top" ItemsSource="{Binding Source={StaticResource UrunlerProvider},XPath=/Depo/Urun/Ad}" FontSize="13" FontWeight="Bold" />
      <Label Height="23" HorizontalAlignment="Left" Margin="18,12,0,0"
Name="label1" VerticalAlignment="Top" Width="120" FontSize="12"
FontWeight="Bold">ürünler</Label>
   </Grid>
</Window>
```

Daha öncedende bahsettiğimiz gibi WPF uygulamalarında veri bağlama işlemleri için **XmlDataProvider** ve **ObjectDataProvider** tipleri kullanılmaktadır. Bu örnekte yer alan **Grid** alanı içerisindeki kontrollerin bir XML veri kaynağını kullanacağını belirtmek

amacıyla **Grid.Resources** elementi içerisinde XmlDataProvider tanımlaması yapılmıştır. XmlDataProvider elementi bu örnekte iki önemli nitelik(attribute) kullanmaktadır. Bunlardan birisi x isim alanı altında bulunan **Key** niteliğidir. Key aslında söz konusu veri kaynağının diğer kontrollerde ele alınmasını sağlamak amacıyla bir isim tanımlaması yapılmasını sağlar. öyleki UrunlerProvider adı, **ComboBox** kontrolünde ele alınan Binding ifadesinde kullanılmaktadır. **XmlDataProvider**elementinin Source niteliğinde ise veri kaynağının yeri işaret edilmektedir. Burada Urunler.xml isimli dosya gösterilmektedir. **Source** özelliğine internet üzerindeki bir URL adresi atanabileceği gibi, başka bir fiziki lokasyondaki XML dosya yeride verilebilir. ComboBox kontrolünde öğelerin**(Items)** içeriklerinin aslında XML dosyasındaki Ad elementlerinden geleceğini belirtmek amacıyla aşağıdaki ifade kullanılmıştır.

```
ItemsSource="{Binding Source={StaticResource
UrunlerProvider},XPath=/Depo/Urun/Ad}"
```

Burada en çok göze çarpan nokta **XPath** niteliğine atanan değerdir. Tahmin edileceği üzere burada bir XPath ifadesi kullanılmış ve Ad elementine kadar geçişler yapılmıştır. Bu anlamda özellikle **XAML** tarafında, XML veri kaynaklarının söz konusu olduğu durumlarda XPath ifadelerinin büyük önem taşıdığını ifade edebiliriz. örneğimizi çalıştırdığımızda aşağıdaki ekran görüntüsü elde edilecektir. Dikkat edilecek olursa Urunler.xml içerisindeki tüm ürünlerin Ad elementlerinin değerleri gelmiştir.

XmlDataProvider tipinin her zaman için harici bir XML veri kümesini işaret etmesine gerek yoktur. **XAML** içeriğinde yer alan gömülü(Embedded) bir XML veri kümeside bu anlamda kullanılabilir. üçüncü örneğimizde bu durumu analiz ederek yazımıza devam edelim. Yeni bir pencereyi(Window) aşağıdaki gibi tasarladığımızı düşünelim. Kahramanımız yine bir**ComboBox** kontrolü olacak.

```
<Window x:Class="DataBindIslemleri.Window3"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Inline Xml kaynağını kontrollere bağlamak" Height="173" Width="280">
   <Grid>
      <Grid.Resources>
         <XmlDataProvider x:Key="SehirVerisi">
            <x:XData>
               <Sehirler xmlns="">
                  <Sehir kod="+90216" ad="Istanbul Anadolu"/>
                  <Sehir kod="202" ad="Kahire"/>
                  <Sehir kod="813" ad="Tokyo"/>
                  <Sehir kod="+44171" ad="Londra"/>
                  <Sehir kod="+1718" ad="New York City"/>
                  <Sehir kod="4989" ad="Münih"/>
               </Sehirler>
            </x:XData>
         </XmlDataProvider>
      </Grid.Resources>
      <ComboBox ItemsSource="{Binding Source={StaticResource SehirVerisi}, XPath=/Sehirler/Sehir/@ad}" Margin="15,57,16,51" FontSize="14" FontWeight="Bold" />
      <Label Height="23" HorizontalAlignment="Left" Margin="15,22,0,0" Name="label1" VerticalAlignment="Top" Width="120">Şehir Telefon Kodları</Label>
   </Grid>
</Window>
```

Bu sefer XML veri kümesi **XAML** dökümanı içerisinde yer alan **Grid**' in kaynağı olarak tanımlanmıştır. Bunun için XmlDataProvider elementi içerisinde **x:XData** isimli bir alt eleman(Child Element) tanımlanmaktadır. Bu elementin içerisinde ise XML veri kümesi bulunmaktadır. Buradaki XML içeriği istenilen şekilde tasarlanabilir. önemli olan noktalardan birisi bir önceki örnekte olduğu gibi yine **x:Key** niteliğinin tanımlanmış olmasıdır. **ComboBox** kontrolünde Sehir elementi içerisindeki ad niteliklerinin**(attributes)** değerleri gösterilmektedir. **ItemsSource** niteliğine atanan ifade

içerisinde bağlanılacak veri kaynağı SehirVerisi olarak belirtildikten sonra ad niteliklerinin değerlerinin elde edilmesi için XPath ifadesinde @ işaret kullanılmıştır.*(Hatırlayalım; XPath ifadelerinde nitelikleri ele alırken **@** işareti kullanılır)* Uygulama bu haliyle çalıştırıldığında aşağıdaki ekran görüntüsü ile karşılaşılacaktır.

Görüldüğü gibi ad niteliklerinin değerleri ComboBox kontrolü içerisine alınmıştır. Geliştirdiğimiz son örneğin bir öncekinden tek farkı, harici bir XML veri kümesi kullanmaktansa, gömülü**(Embeded)** bir XML kaynağının kullanılmasıdır. Pek çok kaynakta özellikle MSDN' de bu tip bir XML içeriği için XML veri adası **(XML Data Island)** tanımlaması yapılmaktadır. XML veri adaları **x:XData** elementleri arasında tutulmak zorundadır.

Sıradaki örneğimizde yine bir XML veri kaynağını ele alacağız. Bu sefer diğer örneklerden farklı olarak ComboBox' ın **ItemTemplate** elementini ele alacağız. Bu sayede ComboBox içerisinde birden fazla veri bağlı kontrolü yan yana göstermemiz mümkün olacaktır. Bu amaçla yeni penceremizi aşağıdaki gibi tasarlamamız yeterlidir.

```
<Window x:Class="DataBindIslemleri.Window4"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="DataTemplete Yardımıyla Xml verisine bağlanma" Height="154" Width="384">
   <Grid>
      <Grid.Resources>
         <XmlDataProvider x:Key="UrunVerileri" Source="Urunler.xml"/>
```

```
    </Grid.Resources>
    <ComboBox Height="40" Margin="21,37,15,0" Name="cmbUrunler"
VerticalAlignment="Top" ItemsSource="{Binding Source={StaticResource
UrunVerileri},XPath=Depo/Urun}" FontSize="14" FontWeight="Bold">
      <ComboBox.ItemTemplate>
        <DataTemplate>
          <TextBlock>
            <Label>
              <Label.Content>
                <Binding XPath="Ad"/>
              </Label.Content>
            </Label>
            <Button>
              <Button.Content>
                <Binding XPath="StokMiktari"/>
              </Button.Content>
            </Button>
            <Image>
              <Image.Source>
                <Binding XPath="Durum"/>
              </Image.Source>
            </Image>
            <Button Name="btnSiparisVer" Content="Sipariş Ver" FontSize="10"
FontWeight="Bold" Background="Black" Foreground="Gold"/>
          </TextBlock>
        </DataTemplate>
      </ComboBox.ItemTemplate>
    </ComboBox>
  </Grid>
</Window>
```

Her zamanki gibi **Grid** içerisindeki kontrollerin bağlanabileceği veri kaynağı sağlayıcısını **XmlDataProvider** yardımıyla **Grid.Resources** elementi içerisinde tanımlamaktayız. Bundan sonra ise ComboBox elementi altında bir **ItemTemplate** elementi açılmaktadır. Bu element içerisinde yer alan **TextBlock** elementi altına **Label**, **Button**, **Image** kontrolleri atılmıştır. Hepsinin ortak özelliği, hangi niteliklerine veri bağlayacaksak, bununla ilgili alt elementin açılması ve **Binding** elementi ile bağlama işleminin gerçekleştirilmesidir. Söz gelimi ürünün Ad elementinin değerini Label kontrolünün **Content** özelliğine atamak istiyorsak aşağıdaki gibi bir bildirim yapılması yeterlidir.

```
<Label>
  <Label.Content>
    <Binding XPath="Ad"/>
```

```
    </Label.Content>
</Label>
```

Burada **Content** elementinin değerinin, Ad isimli elementten alınacağı belirtilmiştir. Yanlız dikkat edilmesi gereken bir nokta vardır. Ad elementi aslında XML ağaç yapısına bakıldığında Depo/Urun üzerinden elde edilebilmektedir. Dolayısıyla burada **XPath** ifadesinde doğrudan Ad değerinin alınabilmesi için Urun elementlerine ulaşılmış olması gerekmektedir. Bunu sağlamak için ComboBox bileşeninin **ItemsSource** niteliğindeki **XPath** ifadesi Depo/Urun şeklinde ayarlanmıştır. Uygulamayı bu haliyle çalıştırdığımızda aşağıdakine benzer bir ekran görüntüsü ile karşılaşırız.

Oldukça etkileyici değil mi? Bir ComboBox' ın her bir öğesi bir taşıyıcı**(Container)** gibi davranıp birden fazla farklı bileşeni içeriyor ve içerikleri bir XML veri kümesinden geliyor. Bence süper.

Şu ana kadar geliştirdiğimiz örneklerimizde **XmlDataProvider** tipinden yararlandık ve XML veri kümelerine bağlandık. XML dışındaki veri kaynakları göz önüne alındığında**ObjectDataProvider** bileşeninin kullanılması söz konusudur. Bu sınıf yardımıyla herhangibir .Net tipine bağlanmak mümkündür. MSDN bu konu ile ilişkili olarak çoğunlukla koleksiyonları örnekleyerek işe başlamaktadır. Bizde dilerseniz geleneği bozmayalım. Şimdiki örneğimizde Urun isimli bir sınıfa ait nesne örneklerini barındıran generic bir **List<T>**koleksiyonunun veri kaynağı olarak kullanılmasını ele alıyor olacağız. İlk olarak aşağıdaki sınıf diagramında **(class diagram)** görülen Urun isimli tipi tasarlayarak başlayalım.

```
public class Urun
{
   public int Id;
   public string Ad;
   public double BirimFiyat;
   public int StokMiktari;
   public bool Durum;

   public override string ToString()
   {
      return Id.ToString() + " " + Ad+" "+BirimFiyat.ToString();
   }
}
```

Urun sınıfı yine bir ürünü tanımlayabilecek bazı public alanlara sahiptir. **ToString** metodunu ezmemizin**(override)** sebebi ise, **ComboBox** kontrolüne bağlandıklarında ne gösterileceğini belirtmektir. Eğer bunu belirtmessek, ComboBox kontrolünde IsimAlaniAdi.TipAdi **(Namespace.TypeName)** notasyonuna uygun olacak şekilde bir görüntü elde edilir. Gelelim ürünlere ait nesne örneklerini taşıyacak koleksiyon sınıfını tasarlamaya. UrunListesi isimli sınıfımız aşağıdaki gibidir.

```
public class UrunListesi:List<Urun>
{
   public UrunListesi()
   {
      Add(new Urun() { Id = 1, Ad = "Grafik Kartı", BirimFiyat = 35, StokMiktari =
100,Durum=true });
      Add(new Urun() { Id = 2, Ad = "Monitor", BirimFiyat = 150, StokMiktari = 50,
Durum = false });
      Add(new Urun() { Id = 3, Ad = "CPU X86", BirimFiyat = 145, StokMiktari = 150,
Durum = true });
      Add(new Urun() { Id = 4, Ad = "USB Bellek", BirimFiyat = 15, StokMiktari = 250,
Durum = true });
      Add(new Urun() { Id = 5, Ad = "HDD 250 Gb", BirimFiyat = 250, StokMiktari = 14,
Durum = false });
   }
}
```

UrunListesi sınıfı veri bağlanmasında kullanılacak kaynak bir tip olarak göz önüne alındığından **List<Urun>** koleksiyonundan türetilmiştir. Daha önceki yazılarımızda da değinildiği gibi **C# 3.0** ile gelen yeniliklerden birisi olan nesne başlatıcıları**(Object Initializers)** kullanılarak Urun nesneleri örneklenmiş bir **List** koleksiyonuna eklenmiştir. Şimdi UrunListesi sınıfını veri kaynağı olarak kullanacak bir pencereyi**(Window)** aşağıdaki gibi tasarlayabiliriz. Kahramanımız her zamanki gibi bir ComboBox bileşenidir.

```
<Window x:Class="DataBindIslemleri.Window5"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://sc
hemas.microsoft.com/winfx/2006/xaml" Title="Basit ObjectDataProvider Kullanımı"
Height="158" Width="290" xmlns:dahili="clr-namespace:DataBindIslemleri">
   <Grid>
      <Grid.Resources>
         <ObjectDataProvider x:Key="UrunVerileri" ObjectType="{x:Type
dahili:UrunListesi}" />
      </Grid.Resources>
      <ComboBox FontSize="12" FontWeight="SemiBold" Height="29"
```

```
Margin="17,44,45,0" Name="comboBox1"
VerticalAlignment="Top" ItemsSource="{Binding Source={StaticResource
UrunVerileri}}" />
   </Grid>
</Window>
```

ObjectDataProvider tipide **x:Key** isimli bir niteliği kullanarak, veriye bağlanacak kontrollerin kullanabilmesi için ortak bir isim tanımlaması yapmaktadır. XmlDataProvider tipinde Source isimli nitelik ile veri kümesi belirtilirken **ObjectDataProvider** tipinde bu iş için **ObjectType** niteliği(attribute) kullanılmaktadır. **ObjectType** niteliğinde ise UrunListesi isimli bir tipin veri kaynağı olarak kullanılacağı ve bunun, dahili kelimesi ile ifade edilen isim alanında olduğu belirtilmektedir. Peki dahili **XML** isim alanı nereden tanımlanmıştır? Bunun için **Window** elementinde bir XML isim alanı **(XML Namespace)** tanımlaması aşağıdaki gibi yapılmıştır. Burada, clr-namespace ifadesini izeleyen ismin bir CLR isim alanı(Namespace) adı olduğu ve XAML dökümanı içerisinde dahili kısa adı ile ifade edileceği belirtilmektedir.

```
xmlns:dahili="clr-namespace:DataBindIslemleri"
```

Dikkat edilecek olursa, tipin içerisinde yer aldığı isim alanı**(Namespace)** işaret edilmektedir. Buradan şu sonucada varabiliriz. Veri bağlama amacıyla kullanılan tip farklı bir assembly içerisinde, dolayısıyla farklı bir isim alanında bulunuyorsada ilgili XAML içeriğinde kullanılabilir. Farklı bir **assembly** söz konusu olduğunda aşağıdakine benzer bir tanımlama yeterli olacaktır.

```
xmlns:dahili="clr-namespace:DataBindIslemleri,assembly=UrunLibrary"
```

Uygulamayı çalıştırdığımızda aşağıdakine benzer bir ekran görüntüsünü elde ederiz.

Dikkat edilecek olursa ToString metodu içeriği ComboBox kontrollerinde birer öğe olarak görülmektedir.

Veri bağlama işlemlerinde tip olarak **DataTable** veya **DataSet** gibi bağlantısız katman nesne örneklerinin kullanılması çok daha yaygındır. Bu tip bir senaryoda DataContext sınıfı ele alınmaktadır. Sıradaki örneğimizde bir DataTable içerisindeki veri kümesinin, WPF kontrollerine nasıl bağlanabileceğini incelemeye çalışacağız. örnek bir senaryo

olarak **SQL Server 2005** ile birlikte gelen veritabanlarından birisi olan **AdventureWorks** ve **Product**, **ProductPhoto**, **ProductProductPhoto** tablolarını göz önüne alabiliriz. Bu tablolar arasındaki ilişki aşağıdaki şekilde olduğu gibidir.

Amacımız resimli ürün bilgilerini bir **ListBox** kontrolünde gösterebilmek. Bu amaçla ilk olarak Window6 isimli penceremizin kodlarını aşağıdaki gibi geliştirelim.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Data;
using System.Data.SqlClient;

namespace DataBindIslemleri
{
```

```
   public partial class Window6 : Window
   {
      string sqlConn = "data source=.;database=AdventureWorks;integrated
security=SSPI";
      string sorgu = @"SELECT PRD.ProductID, PRD.Name AS ProductName,
PRD.SafetyStockLevel , PRD.StandardCost,PRD.ListPrice, PH.ThumbNailPhoto FROM
Production.Product PRD INNER JOIN Production.ProductProductPhoto PPH ON
PRD.ProductID = PPH.ProductID INNER JOIN Production.ProductPhoto PH ON
PPH.ProductPhotoID = PH.ProductPhotoID";

      private void VeriyiCek()
      {
         DataTable dtUrunler = new DataTable();
         using (SqlConnection conn = new SqlConnection(sqlConn))
         {
            SqlDataAdapter da = new SqlDataAdapter(sorgu, conn);
            da.Fill(dtUrunler);
         }
         DataContext = dtUrunler;
      }

      public Window6()
      {
         InitializeComponent();
      }

      private void Window_Loaded(object sender, RoutedEventArgs e)
      {
         lstUrunler.Items.Clear();
         VeriyiCek();
      }
   }
}
```

Bu kod parçasında üzerinde durulacak olan tek nokta **Window** sınıfının **FrameworkElement** sınıfından kalıtımsal olarak devraldığı **DataContext** özelliğine **DataTable** nesne örneğinin atanmış olmasıdır. Böylece bir anlamda **XAML** içerisindeki bileşenlerin bağlanabileceği veri içeriği set edilmiş olur. Buna göre Window6.xaml dosyasının içeriğini aşağıdaki gibi tasarlayabiliriz.

```
<Window x:Class="DataBindIslemleri.Window6"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation" xmlns:x="http://sc
hemas.microsoft.com/winfx/2006/xaml" Title="Window6" Height="273" Width="567"
Loaded="Window_Loaded">
   <Grid>
```

```
    <Grid.Resources>
      <DataTemplate x:Key="Urunler">
        <StackPanel Orientation="Horizontal">
          <Label Content="{Binding Path=ProductID}"/>
          <Label Content="{Binding Path=ProductName}"/>
          <Label Content="{Binding Path=ListPrice}"/>
          <Label Content="{Binding Path=SafetyStockLevel}"/>
          <Image Name="imgPhoto" Source="{Binding Path=ThumbNailPhoto}"/>
        </StackPanel>
      </DataTemplate>
    </Grid.Resources>
    <ListBox Margin="13,28,17,31"
Name="lstUrunler" ItemsSource="{Binding}" ItemTemplate="{StaticResource
Urunler}"/>
  </Grid>
</Window>
```

Şimdi burada neler yaptığımıza kısaca bakalım. öncelikli olarak **Grid.Resources** elementi içerisinde bir **DataTemplete** oluşturulmaktadır. Bu veri şablonu, kullanıldığı yerde nasıl bir içerik sunulacağını belirlemektedir. Söz gelimi, Label kontrollerimizin **Content** özelliklerine yapılan atamalarda **DataContext**' in işaret ettiği veri kümesindeki alanlar belirlenmektedir. Diğer taraftan **Image** kontrolünün **Source** özelliğine yapılan atama ile **ThumbNailPhoto** alanındaki binary içeriğin bağlanması sağlanmıştır. Peki bu veri şablonunu kim kullanacaktır? Bunun için örnek olarak bir ListBox kontrolü ele alınmaktadır. Bu kontrolde **ItemsSource** özelliğine sadece **Binding** atanması, veri kaynağı olarak DataContext içeriğinin ele alınacağını göstermektedir. Diğer taraftan veri şablonunun set edildiği yer **ItemTemplate** özelliğine yapılan atamadır. Burada atamada Urunler isimli veri şablonunun **(Data Template)** kullanılacağı belirtilmektedir. Bu durumda uygulama çalıştırıldığında aşağıdakine benzer bir ekran görüntüsü ile karşılaşılır.

Dikkat edilecek olursa **ListBox** içeriği, **DataTable**' a yüklenen veriler ile dolmuştur. Her halde buradaki en güzel nokta, ürün resimlerininde ListBox içerisinde gösterilebiliyor olmasıdır. Şimdi bu örneği biraz daha farklılaştıralım. örneğin ListBox kontrolünde ürün adları gözüküyor olsun. Bunlardan herhangibiri seçildiğindeyse, ürün ile ilgili diğer bilgiler **TextBox** ve**Image** kontrollerinde görünüyor olsun. Bunun için yeni bir pencere**(Window)** ekleyip aşağıdaki gibi tasarlamamız yeterli olacaktır.

```
<Window x:Class="DataBindIslemleri.Window7"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Window7"
Height="304" Width="632">
   <Grid>
      <Grid.Resources>
         <DataTemplate x:Key="Urunler">
            <StackPanel Orientation="Horizontal">
               <Label Content="{Binding Path=ProductName}"/>
            </StackPanel>
         </DataTemplate>
      </Grid.Resources>
      <ListBox Margin="13,19,0,18" Name="lstUrunler" ItemsSource="{Binding}"
ItemTemplate="{StaticResource Urunler}" HorizontalAlignment="Left"
Width="139"IsSynchronizedWithCurrentItem="True" />
      <Label Height="23" HorizontalAlignment="Left" Margin="176,24,0,0"
Name="label1" VerticalAlignment="Top" Width="61">ürün Adı</Label>
      <TextBox Text="{Binding Path=ProductName}" Height="21"
Margin="270,24,191,0" Name="txtProductName" VerticalAlignment="Top"></TextBox>
      <Label Height="23" HorizontalAlignment="Left" Margin="176,50,0,0"
Name="label2" VerticalAlignment="Top" Width="120">Birim Fiyatı</Label>
      <TextBox Text="{Binding Path=ListPrice}" Height="21" Margin="270,50,193,0"
Name="txtListPrice" VerticalAlignment="Top" />
      <Label Height="23" HorizontalAlignment="Left" Margin="176,77,0,0"
Name="label3" VerticalAlignment="Top" Width="120">Stok Seviyesi</Label>
      <TextBox Text="{Binding Path=SafetyStockLevel}" Margin="270,77,193,0"
Name="txtSafetyStockLevel" Height="20" VerticalAlignment="Top" />
      <Image Source="{Binding Path=ThumbNailPhoto}" Margin="182,113,195,20"
Name="imgPhoto" />
      <Label Content="{Binding Path=ProductID}" Height="49"
HorizontalAlignment="Right" Margin="0,24,57,0" Name="lblProductID"
VerticalAlignment="Top" Width="103" Foreground="Red" FontSize="16"
FontWeight="Bold"/>
   </Grid>
</Window>
```

Bir önceki örnek ile karşılaştırıldığında önemli olan tek fark ListBox kontrolünün **IsSynchronizedWithCurrentItem** özelliğinin değerinin **true** olarak set edilmiş olmasıdır. Eğer bu özelliğe true değerini atamassak, ListBox üzerinde dolaştığımızda, bir başka deyişle başka bir öğeye geçtiğimizde diğer kontrollerin içerikleri **DataContext** nesnesinden dolmayacaktır. Bu durumda ListBox kontrolünde diğer öğelere tıklasakta hep ilk satırın bilgileri görünecektir. Uygulamamızı bu haliyle çalıştırdığımızda aşağıdakine benzer bir ekran görüntüsü elde ederiz.

Görüldüğü gibi örnek kaydın üzerine ListBox ile gidildiğinde, kontrollerin içerikleride o an **DataContext**' te üzerinde bulunulan satıra ait değerler olarak değişmiştir.

Bazı durumlarda birbirleriyle ilişkili olan tabloların kullanılmasıda söz konusudur. özellikle bağlantısız katman**(Disconnected Layer)** nesneleri göz önüne alındığında bu tip vakkaları karşılamak için **DataRelation** nesnelerinden yararlanmaktayız. Peki DataRelation örnekleri ile aralarındaki ilişkiler**(Relations)** ifade edilen tabloları WPF uygulamalarındaki kontrollerimize nasıl bağlayabiliriz? Sıradaki örneğimizde bu durumu ele almaya çalışacağız. Söz gelimi, **AdventureWorks** veritabanında yer alan **ProductSubCategory** ve **Product** tablolarını baz alalım. Bu tablolar aşağıdaki diagramdanda görüleceği gibi birbirlerine **ProductSubCategoryID** alanları üzerinden bağlıdırlar.

Bu ilişkiyi bağlantısız katmanda temsil edebilmek için **DataSet**, **DataTable** ve **DataRelation** nesnelerine ihtiyaç vardır. Window8 örneğinde, alt kategorilerin gösterildiği bir ComboBox kontrolü ve bu kategoriye bağlı ürünlerin gösterildiği bir ListBox kontrolü bulunmaktadır. Senaryomuza göre **ComboBox** kontrolünde değişiklik yapılması halinde, seçilen yeni alt kategoriye bağlı ürünlerinde ListBox kontrolünde gösterilmesi istenmektedir. Bu amaçla ilk olarak verinin çekilmesi ve pencerenin **DataContext** özelliğine gerekli **DataTable** nesne örneğinin atanması gerekmektedir. Bu sebepten Window8.xaml.cs dosyamızın içeriğini aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
using System.Windows.Shapes;
using System.Data;
using System.Data.SqlClient;

namespace DataBindIslemleri
{
```

```
   public partial class Window8 : Window
   {
      private string conStr = "data source=.;database=AdventureWorks;integrated
security=SSPI";
      private string kategoriSorgusu = "Select ProductSubCategoryID,Name From
Production.ProductSubCategory";
      private string urunSorgusu = "Select
ProductID,ProductSubCategoryID,Name,ListPrice From Production.Product";

      private void VerileriCek()
      {
         using (SqlConnection conn = new SqlConnection(conStr))
         {
            SqlDataAdapter daKategori = new SqlDataAdapter(kategoriSorgusu, conn);
            DataTable dtKategori = new DataTable();
            daKategori.Fill(dtKategori);

            SqlDataAdapter daUrun = new SqlDataAdapter(urunSorgusu, conn);
            DataTable dtUrun = new DataTable();
            daUrun.Fill(dtUrun);

            DataSet ds = new DataSet();
            ds.Tables.Add(dtUrun);
            ds.Tables.Add(dtKategori);

            DataRelation iliski = new DataRelation("SubCatToProduct",
dtKategori.Columns["ProductSubCategoryID"],
dtUrun.Columns["ProductSubCategoryID"]);
            ds.Relations.Add(iliski);

            DataContext = dtKategori;
         }
      }
      public Window8()
      {
         InitializeComponent();
         VerileriCek();
      }
   }
}
```

Burada dikkat edilmesi gereken nokta, **DataRelation** nesne örneğinin mutlaka belirtilmesi ve **DataSet** nesne örneğinin **Relations** koleksiyonuna eklenmesi gerektiğidir. Pencerede kullanılacak olan**ComboBox** bileşeninin asıl bağlanacağı veri dtKategori tablosu

olduğundan **DataContext** özelliğine bu tablonun değeri aktarılmıştır. Bundan sonra Window8 penceresinin **XAML**içeriği aşağıdaki gibi hazırlanabilir.

```
<Window x:Class="DataBindIslemleri.Window8"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Window8"
Height="300" Width="440">
   <Grid>
      <Grid.Resources>
         <DataTemplate x:Key="KategoriVerisi">
            <StackPanel>
               <TextBlock Text="{Binding Path=Name}"/>
            </StackPanel>
         </DataTemplate>
         <DataTemplate x:Key="UrunVerisi">
            <StackPanel Orientation="Horizontal">
               <Label Content="{Binding Path=ProductID}"/>
               <Label Content="{Binding Path=Name}"/>
               <Label Content="{Binding Path=ListPrice}"/>
            </StackPanel>
         </DataTemplate>
      </Grid.Resources>
      <ComboBox Height="25" HorizontalAlignment="Left" Margin="12,35,0,0"
Name="cmbAltKategori" VerticalAlignment="Top"
Width="120" ItemsSource="{Binding}" ItemTemplate="{StaticResource
KategoriVerisi}" IsSynchronizedWithCurrentItem="True"/>
      <Label Height="23" HorizontalAlignment="Left" Margin="10,12,0,0"
Name="label1" VerticalAlignment="Top" Width="120">Alt Kategori</Label>
      <ListBox Margin="166,35,10,19" Name="lstUrunler" ItemsSource="{Binding
SubCatToProduct}" ItemTemplate="{StaticResource UrunVerisi}" />
      <Label Height="23" Margin="166,12,132,0" Name="label2"
VerticalAlignment="Top">Urunler</Label>
   </Grid>
</Window>
```

Grid içerisinde iki adet veri şablonu**(Data Templete)** kullanılmaktadır. Bunlardan birisi **ComboBox**, diğeri ise **ListBox** içindir. ComboBox bileşeni kendi içerisinde alt kategori adlarını göstermektedir. Veri kaynağını ItemsSource özelliği ile **{Binding}** olarak belirttiğimizden, **DataContext** içerisinden DataTable kullanılacaktır. Buna göre **ItemTemplate**' in ulaşacağı KategoriVerisi isimli DataTemplete elementinin içeriğine göre alt kategori adları görünecektir. Diğer tarafan, ComboBox üzerinde gezildikçe **DataContext**' in işaret ettiği veri kümesi üzerindede hareket edilebilmesini sağlamak istediğimizden **IsSynchronizedWithCurrentItem** niteliğine true değeri verilmesi şarttır. ListBox kontrolünün ItemsSource özelliğine dikkat edilecek olursa Binding ifadesinden sonra, DataSet' e eklenen DataRelation nesne örneğinin adı

yazılmıştır. İşte bu bizim örneğimizin kilit noktasıdır. Bir başka deyişle ListBox kontrolü, içeriğini oluştururken **Binding** ifadesinde belirtilen ilişki**(Relation)** üzerinden hareket edecek ve buna UrunVerisi isimli DataTemplete' e göre verilerini yükleyecektir. Uygulamamızı çalıştırdığımızda aşağıdaki ekran görüntüsünde olduğu gibi alt kategori ve buna bağlı ürünlerin başarılı bir şekilde ilişkilendirildiği ve kontrollere bu değişimin yansıtıldığı görülür. örnekte Shorts alt kategorisi seçilmiş ve buna bağlı olan ürünlerin bilgiside (DataTemplete ile çektiklerimiz) ListBox içerisinde gösterilmiştir.

Buraya kadar geliştirdiğimiz örneklerimizde, WPF uygulamalarında yer alan kontrollerin çeşitli veri kaynaklarına**(XML, Database, Object** gibi) farklı şekillerde nasıl bağlanabileceklerini incelemeye çalıştık. özellikle XML bazlı veri kaynakları için **XmlDataProvider** tipini, nesne(Object) bazlı kaynaklar için **ObjectDataProvider** tipini kullanmayı, bunlara ek olarak özellikle veritabanı bazlı gerçek bağlantılarda da **DataContext** tipinden yararlanmayı incelemeye çalıştık. önceki Windows mantığına göre XAML getirdiği bir takım yenilikler ile, veri bağlama işleminin epey bir değiştiği sonucunuda çıkartmamız mümkün. Konuyla ilişkili detaylı bilgilere ve çok daha güzel örneklere MSDN' den ulaşabilirsiniz. Ayrıca bir önceki makalemizde tanıttığımız kitaplarında çok faydası olacağını söyleyebilirim. Böylece geldik bir makalemizin daha sonuna. İlerleyen makalelerimizde WPF ile uygulamalar geliştirmeye devam ediyor olacağız. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

örnek Uygulama için Tıklayın

WPF - Application Nesnesi (2007-08-30T17:48:00)

wpf,

Windows Presentation Foundation windows tabanlı uygulama geliştirmeye çok yeni bir yaklaşım getirdi. Tabiri yerindeyse pek çok yenilik ile karşı karşıyayız. İşte bu makalemizde WPF ile geliştirilen windows uygulamalarında çekirdek nesnelerden birisi olan **Application** tipini incelemeye çalışacağız. Application nesnesi, **WPF** uygulamalarının çekirdek nesnesidir. Genel olarak bir windows uygulamasının çalıştırılması işletim sistemi tarafından tetiklenen bir aksiyonla gerçekleşir ve bu uygulama Process dahilinde ayrı bir **AppDomain** içerisinde çalışır. Buna göre her WPF uygulaması çalıştığı yaşam süresi boyunca yanlızca bir Application nesnesine sahip olur. Buda çok doğal olarak, windows uygulaması içerisindeki tüm sayfa**(Page)** veya pencerelerin**(Windows)** ortaklaşa kullanacakları global seviyede bir nesnenin söz konusu olması anlamına gelmektedir. Aşağıdaki şekilde Application nesnesinin WPF uygulamasındaki yeri tasvir edilmeye çalışılmaktadır.

Bu, inanıyorumki web uygulaması geliştirenler için tuhaf değildir. Malum web uygulamalarında da Application nesnesi vardır ve özellikle uygulama seviyesinde değişkenlerin tutulması için kullanılır. Aslında bu kavram .Net Framework 3.0 öncesi windows programlamadada vardır. öyleki hepimiz **Main** metod içerisinde kullanılan Application sınıfını ve üyelerini az çok biliyoruz. Ne varki, WPF ile geliştirilen uygulamalarda Application nesnesinden yararlanarak daha farklı fonksiyonellikler elde edilebilmektedir. Bunları maddeler halinde aşağıdaki gibi sıralayabiliriz.

- Uygulamanın yaşam süresi izlenebilir. **(Application LifeTime)**

- Komut satırından uygulamaya gönderilen parametreler**(Command Arguments)** alınıp işlenebilir.
- Uygulamanın kapatılma süreci ele alınabilir.
- Ele alınamayan istisnalar**(Unhandled Exceptions)** için özel durum yönetimleri gerçekleştirilebilir.
- Uygulama genelinde kullanılabilecek global özellikler**(properties)** ve kaynaklar**(resources)** tanımlanabilir ve kullanılabilir.
- Uygulamanın derleme**(build)** sürecine ait ayarlar yapılabilir.
- XBAP**(Xaml Browser APplication)** uygulamalarında navigasyon süreci izlenebilir.

Application nesnesini işletim sistemi ile uygulama kodu arasındaki bir arayüz olarakda düşünebiliriz. Uygulama içerisindeki tüm pencere(Window) veya sayfaların(Page) aynı Application nesnesine erişebilmesini ve bu nesnenin tekliğini sağlamak için tahmin edileceği üzere **Singleton Pattern**' e uygun bir üretim sistemi söz konusudur. Bu tek nesne üretim işini Application sınıfının**Current** özelliği üstlenmektedir. Application nesneleri, uygulamanın bulunduğu bilgisayarda saklandıklarından özellikle kaynak**(resource)** veya özelliklerin**(properties)** değerlerinin saklanması gibi durumlarda sunucu**(server)** gibi kaynaklara erişime gerek kalmamaktadır. Buda bir avantaj olarak görülebilir.

örneklere başlamadan önce, uygulama yaşam sürecini**(Application LifeTime)** ele almakta ve değerlendirmekte yarar olacağı kanısındayım. Application nesnesi ile birlikte, bir WPF uygulamasının yaşam sürecini daha iyi kontrol edebilmekteyiz. öncelikli olarak aşağıdaki temsili resmi göz önüne alalım. Bu şekilde temel olarak bir WPF uygulamasının standart yaşam döngüsü, süreçteki temel olaylar ve çevresel bazı etkiler irdelenmeye çalışılmaktadır.

Herşeyden önce WPF uygulmasının kullanıcı tarafından tetiklenmesi sonrası işletim sistemi tarafından uygulamanın bir **AppDomain** içerisine açılması söz konusudur. Bundan sonraki süreçte uygulama çeşitli nedenlerle sonlanıncaya kadar ele alabilecek bazı olaylar vardır. Uygulama çalışmaya başladığında Application nesnesinin **Startup** olayı tetiklenir. Bu olay içerisinde uygulama başlatılırken yapılması istenenler yazılabilir. örneğin uygulamanın hangi pencere veya sayfasının yükleneceğine burada karar verilebilir. Startup olayını global seviyedeki özellik ve kaynakların saklanması halinde, yüklenecekleri yer olarakda tasarlayabiliriz. Hatta komut satırı parametrelerininde**(Command Line Arguments)** bu olay içerisinde ele alınması sağlanabilir. Yazımızın ilerleyen kısımlarında bununla ilişkili bir örnek geliştiriyor olacağız.

Gelelim Activated ve Deactivated olaylarına. **Activated** olayı, WPF uygulaması ilk çalıştırıldığında ve ilk penceresi açıldığında otomatik olarak tetiklenir. Bundan sonra uygulama pasif moda(Deactivate) geçinceye kadar çalışmaz. İşletim sistemi üzerinde çalışan başka uygulamalara yapılan geçişlerde, **Deactivated** isimli olay tetiklenmektedir.

Tahmin edileceği üzere tekrardan WPF uygulamasına dönülmesi sonrasında Activated olayı yeniden tetiklenecektir. Bu tetikleme, **taskbar** yardımıyla söz konusu WPF uygulamasındaki formlardan herhangibirinin açılması halinde, **Alt+Tab** tuş kombinasyonu ile yapılan geçişler sonrasında meydana gelmektedir. Deactivated olayında kaynak tüketimi yüksek olan çalışma zamanı nesnelerinin işleyişlerinin duraksatılması yada uyku moduna geçirilmeleri sağlanarak işletim sisteminin daha az yorulması gerçekleştirilebilir. Activated olayı tetiklendiğindede söz konusu kaynakların tekrardan ayağa kaldırılması işlemleri gerçekleştirilebilir. Bu tip bir senaryo elbetteki tartışmaya açık olmalıdır.

NOT : *Activated ve Deactivated olayları* ***XBAP(Xaml Browser APplications)*** *uygulamaları tarafından desteklenmemektedir.*

Gelelim **DispatcherUnhandledException** olayına. Şekildende görüleceği üzere uygulama kodu içerisinden çalışma ortamına fırlayacak bir istisnayı Application nesnesninin bu olayı içerisinde ele alabiliriz. Uygulama içerisindeki kodlarda bir exception oluştuğunda(özellikle ele alınmayan-**unhandled exceptions**) WPF çalışma ortamı standart bir hata dialog penceresi çıkartacak ve hata raporunun gönderilip gönderilmeyeceği sorulacaktır. Ardındanda uygulama sonlandırılacaktır. Ancak, DispatcherUnhandledException olayı içerisine gelen**DispatcherUnhandledExceptionEventArgs** parametresininin **Handled** özelliğini kullanarak uygulamanın kapatılması *(Eğer mümkünse tabi)* engellenebilir.

SessionEnding isimli olay, işletim sisteminden aşağıdaki aktiviteler gerçekleştiğinde tetiklenmektedir.

- LogOff
- Windows Shutdown
- Restart
- Hibernate

Eğer çalşan **WPF** uygulamasında kritik işlemler*(örneğin halen daha devem eden veya sunucuya bağlı halde iken veritabanı üzerinde transaction bazlı gerçekleşen bir işlem vb. söz konusu olabilir)* söz konusu ise, yukarıdaki aktivitelerin gerçekleşmesi halinde sürecin iptal edilmesi arzu edilebilir. Bu nedenle **Application** sınıfının SessionEnding olayı ele alınır. Bu amaçla,**SessionEndingCancelEventArgs** parametresi kullanılır. Eğer SessionEnding içerisinde işlem iptali yapılmassa Application sıfının **Shutdown** metodu çağırılır ve **Exit** isimli olay tetiklenmiş olur.

NOT : *SessionEnding olayı* ***XBAP(Xaml Browser APplications)*** *uygulamları tarafından desteklenmemektedir.*

Bir WPF uygulamasını bilinçli olarak sonlandırmak için Application sınıfının **Shutdown** metodu kullanılabilir. Aslında uygulamanın ana penceresi**(Main Window)** kapatıldığında, tüm pencereleri kapatıldığında veya yukarıdaki gibi işletim sistemince **LogOff, Windows Shutdown, Restart, Hibernate** aksiyonları

gerçekleştirildiğinde otomatik olarak çalışır. İstenirse bir uygulama içerisindeki ShutDown metodunun hangi durumlarda otomatik olarak çalışacağı **ShutdownMode***(ShutDownMode enum sabiti tipinden değerler alır)* özelliği yardımıyla Application elementi içinden veya uygulama kodundan değiştirilebilir. ShutDownMode enum sabitinin **OnLastWindowClose, OnMainWindowClose** ve **OnExplicitShutDown** isminde üç farklı değeri vardır. **OnLastWindowClose** varsayılan değerdir ve uygulama içerisinde birden fazla pencere olması halinde en sonuncusu kapatıldığında uygulamanın kapanması söz konusudur. Bir başka deyişle uygulamanın kapatılabilmesi için açık olan tüm pencerelerin kapatılması gereklidir. Eğer OnMainWindowClose değeri verilirse, uygulama başladığında etklinleştirilen pencere kapatıldığında uygulama sonlanır. **OnExplicitShutDown** seçildiğinde ise uygulamanın kapatılması geliştiriciye bırakılmıştır. ShutDown metodu ile yapılan sonlandırma isteklerinde istenirse işletim sistemine bir çıkış kodu**(ExitCode)** değeri integer tipinden gönderilebilir. çıkış kodunun değeri, bu uygulamaya bağlı başka uygulamalar tarafından değerli olabilir. ExitCode için varsayılan değer sıfırdır.

Shutdown metoduna yapılan bilinçli çağırdan sonra veya **SessionEnding** gerçekleştikten sonra **Exit** isimli olay tetiklenir. Bu olay içerisinde uygulamanın son durumuna ait bilgilerin saklanması gibi işlemler yapılabilir. özellikle global özellik(Global Properties) ve kaynakların(Resources) saklanma işlemlerinin gerçekleştirilmesi için ideal bir lokasyondur. Bu olay içerisine gelindiğinde uygulamanın kapatılma süreci artık iptal edilememektedir. Exit olayı içerisine gelen **ExitEventArgs** parametresi yardımıyla, çıkış kodu(Exit Code) değeride yakalanabilir ve buna göre farklı askiyonlar gerçekleştirilebilir.

NOT : *Exit olayı* ***XBAP(Xaml Browser APplications)*** *uygulamalarında da yazılabilir. Ancak ExitCode değeri XBAP uygulamalarında görmezden gelinir. Exit olayı bir XBAP uygulamasında örneğin Internet Explorer 7 ile açılmışsa ilgili tab kapatıldığında, tarayıcı(Browser) uygulama kapatıldığında veya başka bir yere navigasyon ile gidildiğinde tetiklenmektedir.*

Bu kadar teorik bilgiden sonra birazda pratiğe geçmekte fayda var. Yazımızın bundan sonraki kısımlarında örnekler üzerinden ilerlemeye çalışacağız. Ancak bu kez **Visual Studio 2008 Beta 2** sürümünü kullanıyor olacağız. Bu nedenle yazdığımız kodlarda bir değişiklik olmasada, IDE üzerinde final sürümü çıktığında bazı farklılıklar olabileceğini şimdiden söylemek isterim. İlk olarak Visual Studio 2008 Beta 2 de yeni bir WPF uygulaması açarak işe başlayalım. Uygulama açıldığında Application sınıfından türeyen **App** isimli bir tipin olduğunu göreceğiz. App sınıfının türediği Application sınıfının **.Net Framework 3.0** içerisindeki yeri sınıf diagramdan(class diagram) bakıldığında aşağıdaki gibidir.

Uygulamaya ait ayarların hem element hemde kod bazında yönetimi için aşağıdaki şekildende de görüldüğü gibi **App.xaml** ve **App.xaml.cs** dosyaları otomatik olarak oluşturulacaktır.

Visual Studio 2008 Beta 2 ile bir olayı element seviyesinde yüklemek son derece kolaydır. Burada **intellisense** desteğinin tam olarak verildiğini söyleyebiliriz.
örneğin **Startup** olayını aşağıdaki gibi Application elementi içerisinden seçebiliriz.
Aşağıdaki ekran görüntüsündende görüldüğü gibi, Application elementi içerisinde boşluk tuşuna basıldığında kullanılabilecek tüm üyeler çıkacaktır.

Diğer taraftan bir olay yüklenmek istendiğinde (örneğin Startup) aşağıdaki ekran görüntüsünde olduğu gibi geleneksel olarak tab tuşundan yararlanarak ilgili olay metodun otomatik olarak yüklenmesi sağlanabilir.

Bunun sonucunda **App.xaml.cs** içerisine aşağıdaki olay metodu eklenir.

```
private void Application_Startup(object sender, StartupEventArgs e)
{

}
```

Application elementinin içeriği ise aşağıdaki gibi olacaktır.

```
<Application x:Class="UsingApplicationObjects.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
StartupUri="Window1.xaml" Startup="Application_Startup">
```

```
   <Application.Resources>
   </Application.Resources>
</Application>
```

Visual Studio 2008 Beta 2' nin bu yardımlarına değindikten sonra **Activated** ve **Deactivated** olaylarını inceleyerek devam edelim. Bu amaçla Application elementi içerisinde aşağıdaki yüklemeler yapılmalıdır.

```
<Application x:Class="UsingApplicationObjects.App"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
StartupUri="Window1.xaml" Activated="Application_Activated"
Deactivated="Application_Deactivated">
   <Application.Resources>
   </Application.Resources>
</Application>
```

Sonrasında ise App.xaml.cs dosyası içerisinde açılan olay metodları aşağıdaki gibi düzenlenmelidir.

```
using System;
using System.Collections.Generic;
using System.Configuration;
using System.Data;
using System.Linq;
using System.Windows;
using System.Diagnostics;

namespace UsingApplicationObjects
{
   public partial class App : Application
   {
      private void Application_Activated(object sender, EventArgs e)
      {
         Debug.WriteLine("Activated");
      }

      private void Application_Deactivated(object sender, EventArgs e)
      {
         Debug.WriteLine("DeActivated");
      }
   }
}
```

Test amaçlı bir kod yazıldığı için **System.Diagnostics** isim alanında bulunan **Debug** sınıfın **WriteLine** metodu ile **Output** penceresine çıktılar verilmektedir. Uygulama ilk çalıştırıldığında ve output penceresine bakıldığında Activated yazdığı görülecektir. Eğer **Alt+Tab** tuşları veya mouse ile uygulama dışına çıkılırsa output penceresine DeActivated yazdığı görülecektir. Bir başka deyişle Deactivated olayı tetiklenmiştir. Elbette tekrardan uygulamaya dönülürse Activated olayı yeniden tetiklenecektir.

Sıradaki örneğimizde **Startup** ve **Exit** olaylarını birlikte incelemeye çalışacağız. Bunun için bize örnek bir senaryo gerekmektedir. Application nesnesinin tüm uygulama için geçerli olabiliecek özellik**(Property)** ve kaynakları**(Resources)** saklayabildiğinden bahsetmiştik. çok doğal olarak bunları uygulamadan çıkarken saklamak ve uygulama açıldığında yeniden yüklemek isteyebiliriz. İşte bu noktada saklanan bilgileri okuma işlemini Startup olayında, yazdırma işlemini ise **Exit** olayı içerisinde ele almalıyız. **Application** nesnesi üzerinden bir özellik tanımı yapmak ve değerini vermek son derece kolaydır. Tek yapılması gereken Application sınıfının **Properties** özelliği ve indeksleyicisinden yararlanmaktır. Aynı durum kaynaklar içinde geçerlidir. örneğin aşağıdaki ekran görüntüsünde özellik ekleme adımı gösterilmektedir.

Dikkat edilecek olursa Properties özelliği **Dictionary** bazlı bir koleksiyondur ve object tipinden anahtar-değer(key-value) çiftleri ile çalışmaktadır. Buda kendi tiplerimizi özellik olarak tutabileceğimiz anlamına gelir. Diğer taraftan **Resource** yüklemek içinde aşağıdaki notasyon kullanılır.

Resources özelliğide aslında **ResourceDictionary** tipinden bir koleksiyon döndürmektedir. Bu koleksiyonda **IDictionary** arayüzünü uyarlayan ancak içerisinde **Hashtable** bazlı çalışan hızlı bir koleksiyondur. *(Resource ' ları Application elementi içerisinde ApplicationResource alt elementi içerisindede tanımlayabiliriz. örneğin pencerelerdeki kontroller için ortak stilleri burada belirleyebiliriz. Kaynak yönetimi konusunada ilerleyen yazılarımızda değinmeye çalışacağım.)*

Bu bilgilerden sonra StartUp ve Exit olaylarının bildirimlerini yapıp kodlayarak devam edebiliriz. örnek senaryomuzda kullanıcı sembolik olarak bir pencere(Window) üzerinden ürün bilgisi girecek ve bu bilgiler uygulama seviyesinde yazılmış bir sınıfa ait örnekte

saklanacaktır. Tahmin edileceği üzere tipe ait örnek Application nesnesinin özelliği ile elde edilebilecektir. Urun sınıfını ayrı bir fiziki dosyada projeye ekleyip aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace UsingApplicationObjects
{
   class Urun
   {
      public int Id;
      public string Ad;
      public double BirimFiyat;
   }
}
```

Dikkat edilecek olursa, sınıfın içerisinde özellik(property) veya yapıcı metod(constructor) dahil edilmemiştir. Nitekim burada C# 3.0 ile gelen object initializers tekniğinden yararlanılmak istenmektedir. Bu anlamda **Visual Studio 2008 Beta 2**' nin **C# 3.0** içinde tam bir **intellisense** desteği verdiğini söylemeliyim. örneğin Window1 üzerinde yer alan bir Button kontrolün Click olay metodunda Urun sınıfına ait bir nesneyi **object initializers** tekniği ile oluşturmak istediğimizde bu destek açık bir şekilde görülmektedir.

Window1.xaml.cs dosyasının içeriğini aşağıdaki gibi tasarlayabiliriz.

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Data;
using System.Windows.Documents;
using System.Windows.Input;
using System.Windows.Media;
using System.Windows.Media.Imaging;
```

```
using System.Windows.Navigation;
using System.Windows.Shapes;
using System.Collections;

namespace UsingApplicationObjects
{
   public partial class Window1 : Window
   {
      public Window1()
      {
         InitializeComponent();
      }

      private void button1_Click(object sender, RoutedEventArgs e)
      {
         Urun bilgisayar = new Urun() { Id = 1000, Ad = "Bilgisayar", BirimFiyat =
1000 };
         Application.Current.Properties["SonBakilanUrun"] = bilgisayar;
      }
   }
}
```

Application nesnesi üzerinden o anki uygulama referansına erişmek için **Current** özelliği kullanılır. Sonrasında ise **Properties** özelliği üzerinden indeksleyici(indexer) yardımıyla, oluşturulan Urun nesne örneği SonBakilanUrun adıyla kaydedilir. Artık uygulama çalışma zamanında burada oluşturulan Application seviyesindeki özelliğe heryerden erişilebilir. Söz gelimi uygulamadan çıkarken bu özelliklerin içeriğini saklamak istersek App.xaml.cs içerisinde **Exit** olayını aşağıdaki gibi kodlamamız yeterli olacaktır.

```
private void Application_Exit(object sender, ExitEventArgs e)
{
   try
   {
      Urun sonUrun = (Urun)Application.Current.Properties["SonBakilanUrun"]; //
Properties koleksiyonuna eklenmiş SonBakilanUrun isimli bir anahtar var ise bunun değeri
Urun tipinden elde edilir.
      if (sonUrun != null) // Eğer sonUrun nesne örneği null değilse...
      {
         IsolatedStorageFile storageFile
= IsolatedStorageFile.GetUserStoreForDomain(); // Bu uygulamanın ve assembly' ın
kimlik (Identity) bilgisine göre izole edilmiş kullanıcı odaklı depolama alanının elde
edilmesini sağlar.
         IsolatedStorageFileStream stream = new IsolatedStorageFileStream("GlobalA
ppProperties.txt", FileMode.Create, storageFile); // Bu uygulama için diğerlerinden
ayrılmış olan bir alana path' ten bağımsız olacak şekilde GlobalAppProperties.txt
```

```
dosyasının açılmasını sağlar.
            StreamWriter writer = new StreamWriter(stream); // izole edilmiş alandaki dosya
üzerine yazmak için bir StreamWriter kullanılabilir.
            writer.WriteLine(sonUrun.Id.ToString() + "|" + sonUrun.Ad + "|" +
sonUrun.BirimFiyat.ToString()); // Bilgiler text dosyasına yazdırılır.
            writer.Close();
            stream.Close();
        }
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

Tabiki burada illede **IsolatedStorageFile** kullanılması gerekmemektedir. Bunun yerine ikili(binary) veya xml serileştirmeden yararlanılabilir yada bilinen dosya saklama teknikleri tercih edilebilir. **MSDN**kaynaklarında bu konu işlenirken genel olarak yukarıdaki **IsolatedStorageFile** tipinin kullanıldığını söyleyebilirim.
Gelelim **Application_Startup** olay metodunun içerisine. Bu olay metodunu ise aşağıdaki gibi kodladığımızı düşünelim.

```
private void Application_Startup(object sender, StartupEventArgs e)
{
    try
    {
        IsolatedStorageFile storage = IsolatedStorageFile.GetUserStoreForDomain();
        IsolatedStorageFileStream stream = new
IsolatedStorageFileStream("GlobalAppProperties.txt", FileMode.Open, storage);
        StreamReader reader = new StreamReader(stream);
        while (!reader.EndOfStream)
        {
            string[] keyValue = reader.ReadLine().Split(new char[] { '|' });
            Urun urn = new Urun() { Id = Convert.ToInt32(keyValue[0]), Ad =
keyValue[1].ToString(), BirimFiyat = Convert.ToDouble(keyValue[2].ToString()) };
            Application.Current.Properties["SonBakilanUrun"] = urn;
        }
    }
    catch (Exception exp)
    {
        MessageBox.Show(exp.Message);
    }
}
```

Eğer GlobalAppProperties.txt isimli dosya var ise buradan elde edilen değerlerden yararlanarak Urun nesne örneği oluşturulur ve **Application** nesnesinin özellikler koleksiyonuna eklenir. Söz gelimi yüklenen bu değeri yine bir **Window**' un yüklenmesi sırasında ele aldığımızı düşünebiliriz. Bunun için test olması açısından aşağıdaki gibi bir kod yazdığımızı düşünebiliriz.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
   Urun urn = (Urun)Application.Current.Properties["SonBakilanUrun"];
   if (urn != null)
      Title = urn.Ad.ToString(); // Bu pencerenin başlığında Urun' un adı gösterilir.
}
```

Sonuçta uygulama çalıştırıldığında Window ' un başlık kısmında aşağıdakine benzer bir görüntü elde edilir.

NOT : *Yapılan testlerde bilgisayarın kapatılıp açılmasından sonra da Application özelliklerinin yazıldığı dosyadan başarılı bir şekilde okunabildiği görülmüştür.*

Startup olayını komut satırı parametrelerini almak içinde kullanabiliriz. Burada Startup olay metodundaki **StartupEventArgs** parametresinin args isimli özelliği string tipinden bir dizi döndürmektedir. Bu özellik komut satırından girilen parametre değerlerini taşımaktadır. Ortam Visual Studio 2008 Beta 2, platformda .Net Framework 3.0 olunca, C# 3.0 ile gelen extension metodlar(örneğin **Select, Min, Max, Average, Where, Sum** vb...) ve anahtar kelimelerinde doğrudan desteklendiğini görüyoruz. Aşağıdaki ekran görüntüsünde bu durum açık bir şekilde görülmektedir.

örnek senaryomuzda komut satırında, bir veritabanı bağlantısı açmak için gerekli bazı bilgileri aldığımızı düşünebiliriz. örneğin bu bilgiler sunucu adı, veritabanı adı, kullanıcı adı ve şifre olabilir. Buna göre komut satırından uygulamaya gönderilebilecek 4 farklı parametre söz konusudur. Bu parametre deseninin aşağıdaki gibi olduğunu farz edelim.

ProgramAdi.exe s:LONDON d:AdventureWorks u:Burak p:1234

Buna göre Application sınıfının **Startup** olay metoduna aşağıdaki kodları eklediğimizi düşünelim.

```
if (e.Args.Length == 4)
{
   if (e.Args[0][0] == 's')
      Application.Current.Properties["Sunucu"] = e.Args[0].Substring(2,
e.Args[0].Length-2);
   if (e.Args[1][0] == 'd')
      Application.Current.Properties["Veritabani"] = e.Args[1].Substring(2,
e.Args[1].Length-2);
   if (e.Args[2][0] == 'u')
      Application.Current.Properties["Kullanici"] = e.Args[2].Substring(2,
e.Args[2].Length-2);
   if (e.Args[3][0] == 'p')
      Application.Current.Properties["Sifre"] = e.Args[3].Substring(2, e.Args[3].Length-2);
}
```

Burada çok daha farklı algoritmalar düşünülebilir. Temel amaç, komut satırından gelecek 4 parametreninde elde edilmesi ve bunlardan var olanların uygulama nesnesinin ilgili özelliklerine set edilmesidir. Bundan sonraki adımımızda, herhangibir pencerenin

StatusBar kontrolünde, buradaki bilgilerin içeriğini göstermeye çalışacağız. Söz gelimi Window1 içinde bir StatusBar kontrolünde bu bilgiler gösterilebilir. üzülerek belirtmeliyimki bu **StatusBar** bileşenide WPF' den önce bildiğimiz **StatusStrip** değil. İtiraf etmek gerekirse, StatusBar içerisine kontrolleri atmak için bir süre uğraştım. Sonuçta artık kontrolleri ve içeriklerininde hiyerarşik bir **XML** yapısında düşünmemiz gerektiğini öğrendim. Buna göre StatusBar içerisinde barındırmak istediğimiz her kontrolü bir **StatusBarItem** elementi içerisinde göz önüne almalıyız. Dolayısıyla Window1.xaml içeriğini aşağıdaki gibi geliştirmemiz gerekmektedir.

```
<Window x:Class="UsingApplicationObjects.Window1"
xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml" Title="Window1"
Height="294" Width="452" Loaded="Window_Loaded">
   <Grid>
      <Button Height="23" HorizontalAlignment="Left" Margin="16,57,0,0"
Name="button1" VerticalAlignment="Top" Width="75"
Click="button1_Click">Button</Button>
      <StatusBar Height="23" Name="stbBilgi" VerticalAlignment="Bottom">
         <StatusBarItem>
            <TextBlock Name="txtSunucu" Text="Sunucu:"></TextBlock>
         </StatusBarItem>
         <StatusBarItem>
            <Separator/>
         </StatusBarItem>
         <StatusBarItem>
            <TextBlock Name="txtVeritabani" Text="Veritabanı:"></TextBlock>
         </StatusBarItem>
         <StatusBarItem>
            <Separator/>
         </StatusBarItem>
         <StatusBarItem>
            <TextBlock Name="txtKullanici" Text="Kullanıcı:"></TextBlock>
         </StatusBarItem>
         <StatusBarItem>
            <Separator/>
         </StatusBarItem>
      </StatusBar>
   </Grid>
</Window>
```

Dikkat edilecek olursa, her **StatusBarItem** içerisinde bir kontrol yer almaktadır.**TextBlock** kontrollerinin Text özelliklerinden yararlanarak gerekli bilgileri gösterebiliriz. **Seperator** kontrolü ise bu haliyle basit olarak diğer StatusBarItem kontrolleri içerisindeki bileşenlerin arasında bir ayraç görevi üstlenmektedir. *(**WPF** kontrolleri ile ilişkili olarak ilerleyen yazılarımızda detaylı*

incelemeler yapmayı düşünüyorum.) Window1.xaml.cs dosyasında ise Loaded olay metodunu amacımıza yönelik olarak aşağıdaki gibi kodlayabiliriz.

```
private void Window_Loaded(object sender, RoutedEventArgs e)
{
   Urun urn = (Urun)Application.Current.Properties["SonBakilanUrun"];
   if (urn != null)
      Title = urn.Ad.ToString();

    txtSunucu.Text += Application.Current.Properties["Sunucu"]!=null?
Application.Current.Properties["Sunucu"].ToString():"Tanımlı Değil";
    txtVeritabani.Text += Application.Current.Properties["Veritabani"] != null ?
Application.Current.Properties["Veritabani"].ToString() : "Tanımlı Değil";
    txtKullanici.Text += Application.Current.Properties["Kullanici"] != null ?
Application.Current.Properties["Kullanici"].ToString() : "Tanımlı Değil";
}
```

Artık testimizi gerçekleştirebiliriz. Parametreleri test edebilmek için proje özelliklerine**(Project Properties)** gidip **Command Line Arguments** kısmını aşağıdaki gibi doldurmamız yeterli olacaktır.

Uygulama test edildiğinde aşağıdakine benzer bir sonuç ile karşılaşırız.

Bu son örneğimizde komut satırında **WPF(Windows Presentation Foundation)** uygulamasına gelen parametreleri nasıl ele alabileceğimizi incelemeye çalıştık. Dilerseniz yeni bir örnekle devam edelim. Bu örneğimizde uygulamanın kapatma sürecini kontrol altına almaya çalışacağız. Daha öncedende belirttiğimiz gibi, işletim sistemi seviyesinde gelebilecek olan kapatma taleplerini uygulama içerisinden geri çevirme şansına sahip olduğumuzu söylemiştik. Bunun için **Application** nesnesinin **SessionEnding** olay metodunu ele almamız yeterliydi. SessionEnding olay metodunda kilit nokta **SessionEndingEventArgs** isimli parametredir. Bu parametre üzerinden erişilen **ResonSessionEnding** özelliği ResonSessionEnding enum sabiti tipinden bir değer alır. Bu enum sabitinin değerleri**Logoff** veya **Shutdown**' dır. SessionEndingEventArgs' ın bir diğer önemli özelliği ise **Cancel** üyesidir. Bu özelliğe atanan değere göre sürecin iptal edilmesi bir başka deyişle Shutdown veya Logoff' dan vazgeçilmesi sağlanabilir. Şimdi olay metodumuzu aşağıdaki gibi tasarladığımızı düşünelim. *(Elbette SessionEnding olayının yüklenmesi için Application elementi içerisine ilgili niteliği eklemeyi unutmamalıyız.)*

```
private void Application_SessionEnding(object
sender, SessionEndingCancelEventArgs e)
{
   MessageBoxResult cevap=MessageBox.Show("Bilgisayar " +
e.ReasonSessionEnding.ToString() + " nedeniyle kapatılıyor. İptal etmek ister misiniz?",
"Kapatma Sorusu", MessageBoxButton.YesNo, MessageBoxImage.Question);
   if (cevap == MessageBoxResult.No)
      e.Cancel = true;
}
```

Buradaki kod parçasında sembolik olarak kullanıcıya soru sorulmaktadır. Eğer kullanıcı hayır cevabını verirse işletim sisteminin kapanma süreci iptal edilir. Eğer uzun bir süre cevap verilmesse Windows' un standart End Program penceresi karşımıza çıkacaktır. Bu penceredeki progress tamamlandıktan sonra bile No denildiğinde Windows kapatma süreci yine iptal edilecektir. Aşağıdaki ekran görüntüsünde End Program çıktısınında ele alındığı durum gösterilmektedir.

Session_Ending içerisinde daha önceden de bahsedildiği gibi kritik kaynak kaydetme gibi işlemler yapılabilir. Bir başka deyişle uygulamanın son durumunu **(Application State)** korumak adına çeşitli tedbirler alınabilir.

Bir WPF uygulamasının kapatılması ile ilgili olarak **ShutdownMode** özelliğinin değerlerine bakıldığından bahsetmiştik. Bu özelliği doğrudan Application elementi içerisinde belirtebileceğimiz gibi kod yardımıylada değiştirebiliriz. Varsayılan hali **OnLastWindowsClose** olan bu değeri aşağıdaki gibi **OnExplicitShutdown** olarak değiştirdiğimizde, uygulamanın kapatılabilmesi için **Shutdown** metodunun çağırılması gerekmektedir.

Şimdi bunu test edelim. Bu amaçla uygulamaya ikinci bir Window daha dahil edilmiş ve ana pencereden Window2 penceresinin açılması sağlanmıştır. Window2 penceresini açma işlemini Window1 üzerindeki bir Button kontrolüne ait Click olay metodu içerisinde aşağıdaki gibi gerçekleştirmekteyiz.

```
private void btnDigerForm_Click(object sender, RoutedEventArgs e)
{
   Window2 wnd2 = new Window2();
   wnd2.Show();
}
```

Şimdi uygulamamızı çalıştıralım. Sonradan açılan Window2 kapatıldığında uygulama doğal olarak sonlanmaz. Lakin ana pencere kapatıldığında da uygulama sonlanmaz. Bu iki aksiyonu yaptığımızda ve arka planda çalışan Process' lere baktığımızda uygulamanın gerçektende sonlandırılmadığını görebiliriz. Aşağıdaki ekran görüntüsünde bu durum açık bir şekilde görülmektedir.

Dolayısıyla programdan çıkmak için **Shutdown** metodunun bilinçli olarak çağırılması gerekmektedir. Bunun için Window1 içerisinden başka bir Button yardımıyla aşağıdaki gibi bir metod çağrısında bulunduğumuzu düşünebiliriz. Dikkat edilecek olursa o anki uygulama nesnesi üzerinden Shutdown metodu çağırılabilmektedir.

```
private void btnKapat_Click(object sender, RoutedEventArgs e)
{
   Application.Current.Shutdown();
}
```

Artık uygulamamız başarılı bir şekilde kapatılabilir. Daha önceden de belirtildiği gibi, işletim sistemine bir çıkış koduda gönderilebilir. Bunun için **Shutdown** metodunun aşırı yüklenmiş versiyonunu kullanmamız ve bir integer değeri parametre olarak vermemiz yeterlidir.

Yazımızda son olarak kod içerisinden fırlatılan istisnaların **DispatcherUnhandledException** olayında nasıl yorumlanabileceğini inceleyeceğimiz bir örnek geliştireceğiz. Bu amaçla uygulama içerisinden bilinçli olarak bir istisna(exception) nesne örneği fırlatmamız yeterli olacaktır. Window1 içerisinde bu amaçla iki farklı Button kontrolü yerleştirilmiş ve farklı istisna**(Exception)** nesne örneklerinin fırlatılması sağlanmıştır. Window1.xaml.cs içerisindeki yeni kodlar aşağıdaki gibidir.

```
private void btnArgumentException_Click(object sender, RoutedEventArgs e)
{
   throw new ArgumentException();
}

private void btnStackOverFlow_Click(object sender, RoutedEventArgs e)
{
   throw new StackOverflowException();
}
```

Şimdi uygulamayı test edersek .Net Framework 3.0 çalışma zamanının (Run-time) aşağıdaki standart pencereyi çıkarttığını görürüz. üstelik penceredeki cevabımızdan sonra rapor göndersekte, göndermesekte uygulamanın sonlandığını görürüz.

İstersek bu pencerenin çıkmasını engelleyebiliriz. Bunun için, kod içerisinden ele alınmamış**(Unhandled Exceptions)** istisnaların ele alınacağını belirtmemiz gerekmektedir. Bu amacı gerçekleştirmek için **DispatcherUnhandledException** olayında yer alan **DispatcherUnhandledExceptionEventArgs** parametresinin **Handled** özelliğine **true** değerini atamamız yeterlidir. Ancak bu sadece ele alınmamış istisnalar için geçerlidir. Hatta ele alınamayacak cinsten bir istisna olduğunda da bu kontrol yeterli gelmeyebilir. Ne demek istediğimizi biraz daha net anlayabilmek için olay metodlarımızda bir değişiklik yapalım ve **ArgumentException** istisnasını yakalayacağımız bir try...catch bloğunu aşağıdaki gibi uygulama koduna dahil edelim.

```
private void btnArgumentException_Click(object sender, RoutedEventArgs e)
{
   try
   {
      throw new ArgumentException();
   }
   catch (Exception excp)
```

```
    {
        MessageBox.Show("Bir istisna oluştu");
    }
}
```

Uygulama bu şekilde test edildiğinde ve **ArgumentException** istisnası oluşturulduğunda, **try...catch** bloklarından dolayı .Net Framework' ün çalışma zamanı istisna yönetim mekanizması devreye girerek yukarıda yer alan hata mesaj kutusunu göstermeyecektir. Bunun sebebi hatanın uygulama kodu içerisinde kontrollü bir şekilde try...catch blokları ile ele alınmış**(Handled)** olmasıdır. Ancak ele alınmamış istisnalar varsa ve bunlar oluştuğunda loglama gibi ek işlemler yapılmak isteniyor ve kurtarılabilirse uygulamanın çalışmasına devam etmesi isteniyorsa**DispatcherUnhandledException** olayı aşağıdaki şekilde olduğu gibi değiştirilebilir.

```
private void Application_DispatcherUnhandledException(object sender,
DispatcherUnhandledExceptionEventArgs e)
{
    // Burada EventLog dışından bir dosyaya yazdırma işlemleride gerçekleştirilebilir.
    EventLog.WriteEntry("Application", "X Uygulamasında "+DateTime.Now.ToString()+"
zamanında "+e.Exception.Message+" hatası alınmıştır.", EventLogEntryType.Error); // e
parametresi üzerinden oluşan istisna referansı Exception özelliği ile yakalanabilir.

    e.Handled = true; // Eğer oluşan istisna kurtarılabilecek cinstense, standart hata mesajı
kutusunun çıkartılmaması ve uygulamanın çalışmaya devam etmesi sağlanmış olur. Bu
özelliğin varsayılan değeri false dur.
}
```

Bu makalemizde WPF uygulamalarında çekirdek nesnelerden birisi olan **Application** nesnesini incelemeye çalıştık. Makalemizi sonlandırmadan önce sizlere WPF ile ilişkili faydalı olabilecek bir kaç kitap tavsiye etmek isterim.

	Essential Windows Presentation Foundation Addison Wesley yayınlarının **Essential** serisinde yer alan kitaplar çoğunlukla iç anlamıyla bir başvuru kaynağı olarak düşünülebilir. Zaman zaman okura ağır ge
	Pro WPF : Windows Presentation Foundation in .Net 3.0 Şu sıralarda okumakta olduğum bu kitap oldukşa başarılı. üstelik son çıkan WPF Ancak içeriğinde WPF ile ilgili hemen her bilgiye ulaşabiliyoruz. Bu kitabı ısra

	Professional WPF Programming Diğerlerine göre daha az sayfadan(480 sayfa) oluşan bu kitap içerisinde **Expres** ama bazı yerlerde çok fazla detaya girilmediği için başka kaynaklara bakmayı g
	Programming WPF O'Reilly yayınlarındaki favori yazarım Juval Lowy' ye ait bir kitap olmasada*(O* sayfalık(kabul edilebilir bir sayfa adedi) içeriğinin bulunması bu kitabın okunm öğrenmiş ve en büyük elementlerini kavramış oluyorsunuz.

Application nesnesi ile ilgili olarak değinmediğimiz pek çok nokta var. Söz gelimi bu nesneden yararlanarak uygulamadaki tüm pencere yada formları elde edebiliriz. Application' dan türetilen App sınıfı içerisine, tüm uyguladaki nesnelerin erişebileceği üyeler(Metodlar veya özellikler) koyabiliriz vb. Bu tip konuların ve dahasının araştırılmasını siz değerli okurlarıma bırakıyorum. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

örnek Uygulama için Tıklayın

WCF - Client Callback (2007-08-23T17:54:00)

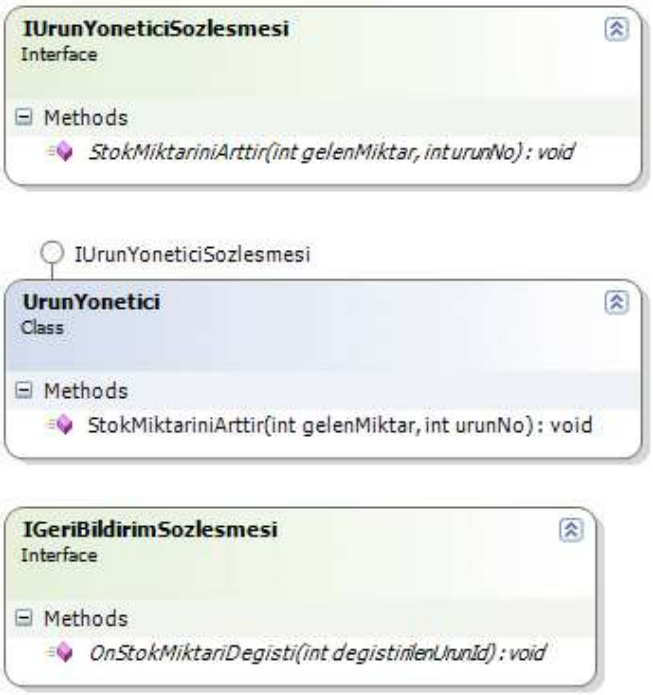
wcf,

Servis yönelimi mimari (Service Oriented Architecture) üzerine geliştirilen sistemler istemci/sunucu**(client/server)** tabanlı bir iletişimi olanaklı kılarlar. Bu sistemlerde süreçler çoğunlukla istemciden sunucuya doğru yapılan operasyon talepleri**(Request)** ve servis tarafından istemciye geri dönen cevaplardan**(Response)** ibarettir. Oysaki bazı **SOA** vakkalarında rollerin tam tersine çevrilmesi gerekebilir. Bir başka deyişle servislerin yeri geldiğinde bir istemci gibi hareket etmesi istenebilir. Söz gelimi bir stok sisteminde yer alan bir servis parçasında, istemcilerin stok üzerinde servis yardımıyla yapacağı hareketleri göz önüne alalım. Bu stok hareketleri istemciden gelecek olan operasyon çağrıları sonucu servis tarafında ele alınıyor olsunlar. Buna göre, servise bağlı diğer istemcilerinde durumdan haberdar olması istenebilir. Bu bir uyarı sistemi olarakda göz önüne alınabilir. Böyle bir sistemde öncelikli olarak servis tarafındaki uygulamanın istemci tarafında yer alan belirli operasyonları çağırabiliyor olması gerekmektedir. Diğer taraftan, servis uygulaması kendisine bağlı istemcilerin hepsinde bir yayınlama yapmak istiyorsa, **olay(event)** yönelimli bir model geliştirilmesi söz konusu olmalıdır. Bu makalemizde **Windows Communication Foundation** sisteminde istemci taraflı geri bildirimlerin **(Client Callback)** nasıl yapılabileceğini çok basit olarak incelemeye çalışacağız.

NOT : WCF (Windows Communication Foundation)*, standart istemci sunucu modeli dışında rollerin tersine dönebileceği iki modeli daha destekler. Bunlar* ***peer-to-peer*** *modeli ve* ***client callback****modelidir. Peer-to-peer modeline göre tüm istemciler aktif ve bağımsız olarak birer servistir ve birbirleriyle mesajlaşabilir. Client Callback modelinde ise servisler, istemci tarafında metod çağrıları gerçekleştirebilecek durumdadır.*

İlk hedef, servis uygulamasının istemci tarafındaki bir metodu çağırabilmesini sağlamak olmalıdır. WCF tarafından bakıldığında, bağlayıcı tiplerin**(binding types)** tamamının geri bildirim işlemlerini desteklemediği görülür. Bunun sebebi kullanılan protokoldür. **TCP** ve **IPC** gibi protokoller istemci geri bildirim işlemlerinde doğaları gereği doğrudan destek vermektedir. Bu sebepten **NetTcpBinding** yada **NetNamedPipeBinding** gibi bağlayıcı tipler istemci geri bildirim işlemlerini doğal olarak desteklemektedirler. Ancak **HTTP** protokolü bağlantısız**(connectless)** olarak çalışan bir model olduğundan geri bildirimi(Callback) doğrudan desteklemez. Bu nedenle **BasicHttpBinding** veya **WsHttpBinding** gibi bağlayıcı tipleri ile Client Callback modeli geliştirilemez. Ancak HTTP protokolününde bu sistemi çift yönlü kanallar açarak gerçekleştirmek mümkündür. Tahmin edileceği gibi bu kanallardan birisi servisden istemciye doğru, diğeri ise ters yönde istemciden servise doğru olmalıdır. Bunu, tek yönlü iki HTTP kanalı olarakta düşünebiliriz. WCF içerisinde yer alan bağlayıcı tiplerinde WsDualHttpBinding tam olarak bu amaçla tasarlanmıştır. Bu nedenle **WsDualHttpBinding** bağlayıcı tipini kullanarak, HTTP protokolünü baz alacak şekilde Client Callback destekli uygulamalar geliştirilebilir.

İlk olarak bir servisin istemci tarafındaki bir operasyonu nasıl çağırabileceğini örnek bir uygulama üzerinden incelemeye çalışalım. Başlamadan önce servis tarafında ve istemci tarafında yapmamız gerekenlerin neler olduğunu vurgulamakta yarar var. Servis tarafında geri bildirim için bir arayüz**(Interface)** tanımlanmalıdır. Bu arayüz istemci tarafında, herhangibir sınıfa**(Class)** uygulanmalıdır. İstemciler, kullanmak istedikleri servise ait bir örnek oluşturduklarında, servis tarafına bir referans verilmelidir. Bu referansı kullanarak servisler, istemciler üzerindeki metodları tetikleyebilirler. Dolayısıyla istemci tarafında geri bildirim tipinin uyguladığı bir arayüzün var olması ve bunun servis tarafında tanımlanmış olması gerekir. Bu cümleler biraz kafa karıştırıcı olabilir. Bu durum daha kısa bir şekilde; istemcideki bir nesne referansının servis tarafından elde edilebilmesi ve onun sayesinde istemci üzerinde, söz konusu arayüzü uygulayan bir sınıfın metodunun servis tarafından çağırılabilmesi şeklinde de ifade edilebilir. öncelikle servis sözleşmesini**(Service Contract)**, geri bildirim sözleşmesini**(Callback Contract)**ve iş yapan servis tipini barındıracak **WCF Class Library** projesini geliştirerek örneğimize başlayalım. WCF sınıf kütüphanemizdeki tiplerin sınıf çizelgesindeki durumu ve kodları aşağıdaki gibidir.



IGeriBildirimSozlesmesi isimli arayüz**(interface)** geri bildirim sözleşmesinin**(Callback Contract)** tanımını yapmaktadır. Bu tanıma göre, servisin istemciler üzerinden tetikleyebileceği metod, int tipinden parametre alan ve geriye değer döndürmeyen bir şekilde tanımlanmıştır. Geri bildirim sözleşmesinin içeriği ise aşağıdaki gibidir.

```
using System;
using System.ServiceModel;

namespace UrunServisi
{
   // Callback Contract tanımlanırken, ServiceContract niteliği kullanılmaz.
   public interface IGeriBildirimSozlesmesi
   {
      // Servisin istemciden çağıracağı geri bildirim metodundan geriye cevap beklenmesine gerek olmadığından IsOneWay özelliğine true değeri atanmıştır.
      [OperationContract(IsOneWay = true)]
      void OnStokMiktariDegisti(int degistirilenUrunId);
   }
}
```

Tanımlanan bu sözleşmede dikkat edilmesi gereken en önemli noktalardan biriside **ServiceContract** niteliğinin uygulanmamış olmasıdır. Bir geri bildirim sözleşmesinde, **OperationContract**niteliği ile, servisin istemciler üzerinden çağırabileceği operasyonların bildirilmesi yeterlidir.

Servis sözleşmesi **IUrunYoneticiSozlesmesi** isimli arayüz(interface) ile tanımlanmaktadır. Bu arayüzün kodları ise aşağıdaki gibidir.

```
using System;
using System.ServiceModel;

namespace UrunServisi
{
   // Servis sözleşmesinde, Callback sözleşmesini bildirmek için CallbackContract özelliği
kullanılır.
   [ServiceContract(CallbackContract=typeof(IGeriBildirimSozlesmesi))]
   public interface IUrunYoneticiSozlesmesi
   {
      [OperationContract()]
      void StokMiktariniArttir(int gelenMiktar, int urunNo);
   }
}
```

Burada dikkat edilmesi gereken
nokta, **ServiceContract** niteliğinde **CallbackContract** tanımlamasının yapılmış olmasıdır. Bu sayede, söz konusu arayüzü uygulayan servis sınıfının, geri bildirim işlemleri sırasında hangi arayüz ile taşınan referansları ele alacağı açık bir şekilde bildirilmiş
olunur. **CallbackContract** özelliği, **type** tipinden bir değer aldığı için **typeof** operatörüne başvurulmuştur ve geri bildirim sözleşmesi olan IGeriBildirimSozlesmesi parametre olarak verilmiştir. Servis sözleşmesini uygulayan sınıfa ait kodlar ise aşağıdaki gibidir.

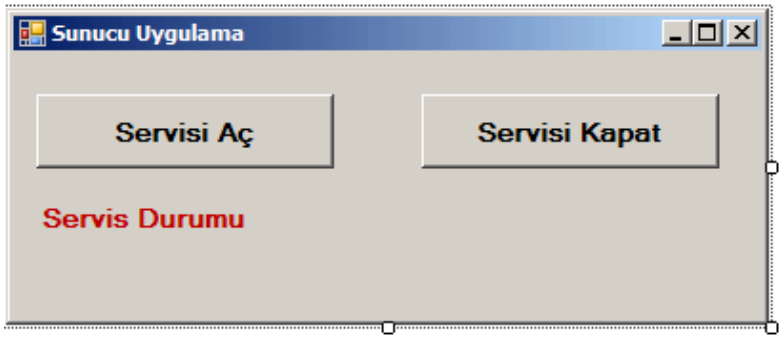
```
using System;
using System.Data;
using System.ServiceModel;
using System.Data.SqlClient;

namespace UrunServisi
{
   public class UrunYonetici:IUrunYoneticiSozlesmesi
   {
      #region IUrunYoneticiSozlesmesi Members

      public void StokMiktariniArttir(int gelenMiktar,int urunNo)
      {
         string sorgu = "Update Products Set UnitsInStock=UnitsInStock+@GelenMiktar
Where ProductID=@PrdId";
         using (SqlConnection conn = new SqlConnection("data
source=.;database=Northwind;integrated security=SSPI"))
         {
            SqlCommand cmd = new SqlCommand(sorgu, conn);
```

```
            cmd.Parameters.AddWithValue("@GelenMiktar", gelenMiktar);
            cmd.Parameters.AddWithValue("@PrdId", urunNo);
            conn.Open();
            int guncellenen=Convert.ToInt32(cmd.ExecuteNonQuery());
            if (guncellenen > 0)
            {
                IGeriBildirimSozlesmesi geriBildirim =
OperationContext.Current.GetCallbackChannel<IGeriBildirimSozlesmesi>();
                if (((ICommunicationObject)geriBildirim).State ==
CommunicationState.Opened)
                {
                    geriBildirim.OnStokMiktariDegisti(urunNo);
                }
            }
        }

        #endregion
    }
}
```

UrunYonetici isimli sınıf içerisinde StokMiktariniArttir isimli bir metod yer almaktadır. Bu metod sembolik olarak, gelen ürün numarasına göre **Northwind** veritabanındaki **Products** tablosunda yer alan**UnitsInStock** alanının değerini gelenMiktar parametresinin değeri kadar arttırmaktadır. İşte burada servisin, istemci tarafında stok miktarının arttırıldığına dair bir geri bildirimde bulunması sağlanmaktadır. Bunun için, istemcinin servis tarafında gönderdiği içerik örneğinin **(InstanceContext)** ele alınması gerekmektedir. İstemciler, servise talepte bulunduklarında WCF çalışma ortamı**(WCF Runtime)**, o anki kapsama ait içerik örneğinin referansınıda mesaj ile birlikte gönderecektir.

OperationContext sınıfının **Current** özelliği üzerinden çağırılan **GetCallbackChannel** generic metodu ile, IGeriBildirimSozlesmesi tarafından taşınabilecek bir istemci referansı elde edilir. İşte bu referans, istemci tarafında geri bildirim sözleşmesini uyarlayan sınıfa aittir. Arayüzlerin polimorfik**(Polymorphic)** yapısı nedeni ile, OnStokMiktariDegisti metoduna yapılacak olan çağrı, aslında istemci tarafındaki metoda yapılmaktadır. özetle servis, istemci tarafındaki bir geri bildirim metodunu tetiklemektedir. İstemci uygulama, servis ile olan bağlantısını herhangibir anda kesebilir. özellikle **OneWay** olarak işaretlenmiş bir callback operasyonunda bu sorun oluşturur. Bu nedenle, geri bildirim kanalının açık olduğundan emin olup geri bildirim metodunu çağırmak amacıyla, **ICommunicationObject** arayüzüne dönüştürme**(cast)** işlemi yapılarak State özelliğinin değerine bakılmaktadır. **State** özelliği, **CommunicationState** enum sabiti tipinden bir değer almaktadır ve örnekte **Opened** olması halinde geri bildirim çağrısının yapılması sağlanmaktadır. Artık servis sözleşmesini yayınlayacak **Host** uygulamanın yazılmasına

geçilebilir. Host uygulama, basit bir Windows uygulaması olarak tasarlanabilir. Bu uygulamanın ekran görüntüsü basit olarak aşağıdaki gibidir.



Şekildende tahmin edileceği üzere, program kullanıcısı servisi açma ve kapatma işlemlerini manuel olarak yapmaktadır. Sunucu uygulamanın kodları aşağıdaki gibi geliştirilebilir.

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.ServiceModel;
using UrunServisi;

namespace Sunucu
{
   public partial class Form1 : Form
   {
      ServiceHost host;

      public Form1()
      {
         InitializeComponent();
         btnServisiKapat.Enabled = false;
      }

      private void ServisiHazirla()
      {
         host = new ServiceHost(typeof(UrunYonetici));
         host.Opening += delegate(object sender, EventArgs e)
                         {
                            lblServisDurumu.Text = "Servis Açılıyor...";
                         };
```

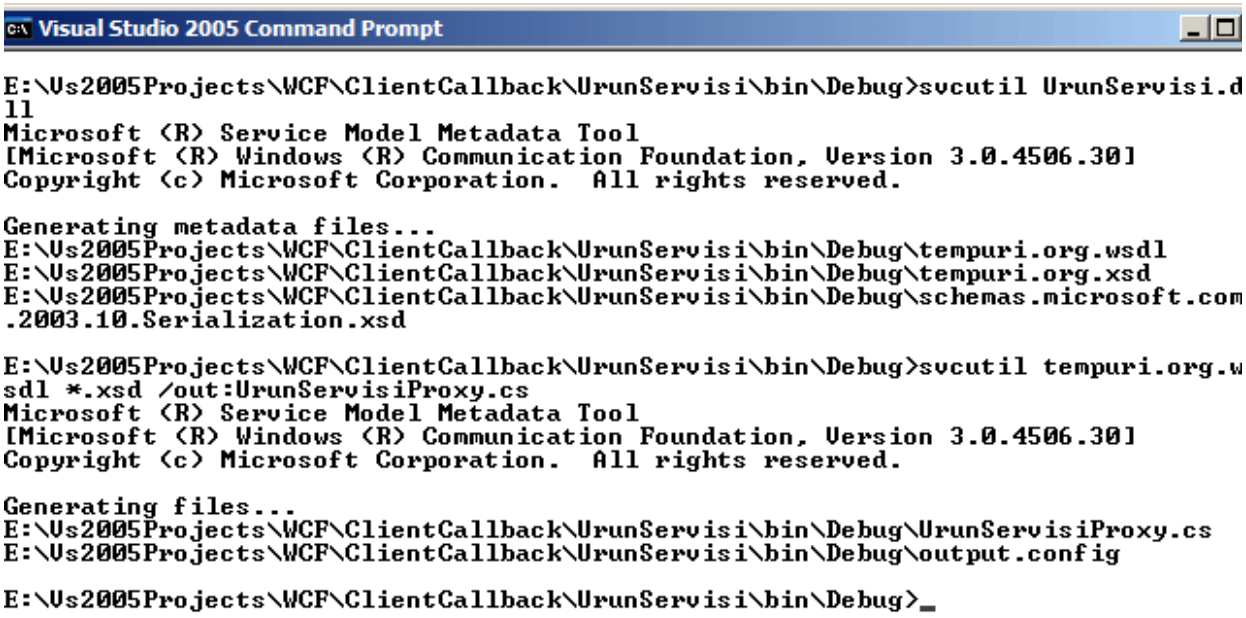
```
        host.Opened += delegate(object sender, EventArgs e)
                    {
                        lblServisDurumu.Text = "Servis Açıldı";
                        btnServisiAc.Enabled = false;
                        btnServisiKapat.Enabled = true;
                    };
        host.Closing += delegate(object sender, EventArgs e)
                    {
                        lblServisDurumu.Text = "Servis Kapatılıyor...";
                    };
        host.Closed += delegate(object sender, EventArgs e)
                    {
                        lblServisDurumu.Text = "Servis Kapatıldı.";
                        btnServisiAc.Enabled = true;
                        btnServisiKapat.Enabled = false;
                    };
    }

    private void btnServisiAc_Click(object sender, EventArgs e)
    {
        ServisiHazirla();
        host.Open();
    }

    private void btnServisiKapat_Click(object sender, EventArgs e)
    {
        host.Close();
    }
  }
}
```

Servis uygulamasında, **ServisHost** örneklemesi yapılmakta ve servisin durumunu daha kolay izleyebilmek açısından gerekli olaylar generic metodlar yardımıyla yüklenmektedir. Böylece program kullanıcıları, servisin açılması ve kapanması durumunu kolay bir şekilde izleyebilecektir. Her zamanki gibi sunucu uygulama, WCF altyapısı için gerekli **System.ServiceModel.dll** ile servis sözleşmesini içeren UrunServisi.dll isimli sınıf kütüphanesini**(WCF Class Library)** referans etmektedir. Bunlara ek olarak sunucu uygulamanın WCF için gerekli konfigurasyon bilgileri ise aşağıdaki gibi **app.config** dosyasında yer almaktadır.

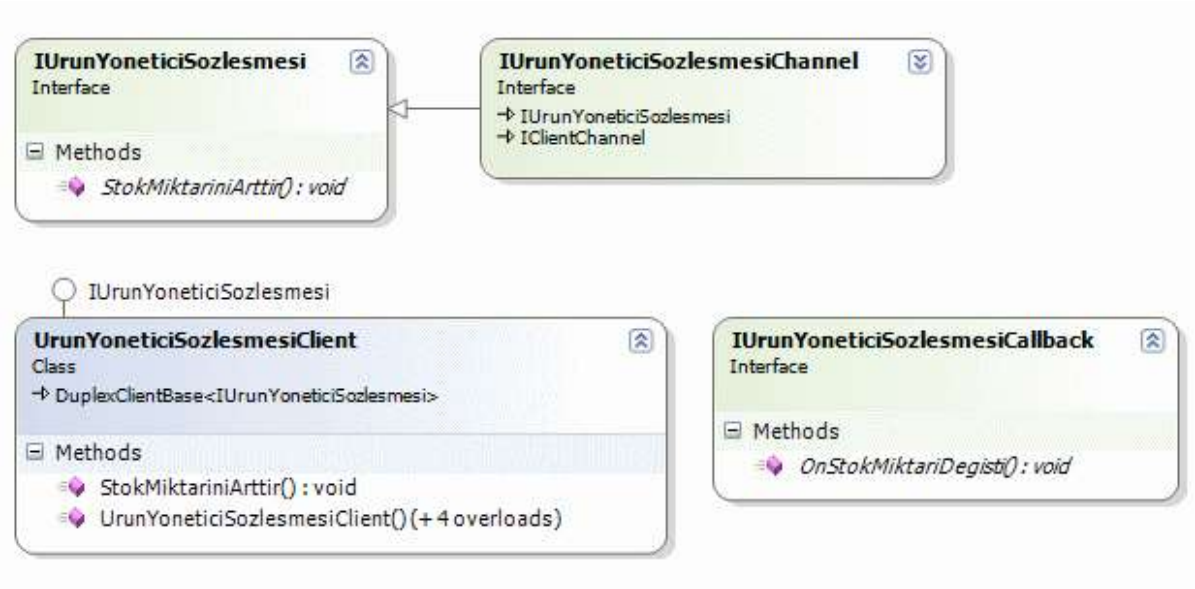
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
```

```
        <service name="UrunServisi.UrunYonetici">
          <endpoint address="net.tcp://localhost:9001/UrunServisi.svc" binding="net
TcpBinding"
name="UrunServisiEndPoint" contract="UrunServisi.IUrunYoneticiSozlesmesi" />
        </service>
      </services>
   </system.serviceModel>
</configuration>
```

Bu örnekte **NetTcpBinding** bağlayıcı tipi kullanılmaktadır. İstemci tarafını yazmadan önce, gerekli proxy sınıfının üretilmesi gerekmektedir. Bu noktada **svcutil.exe** aracından aşağıdaki gibi yararlanılabilir.

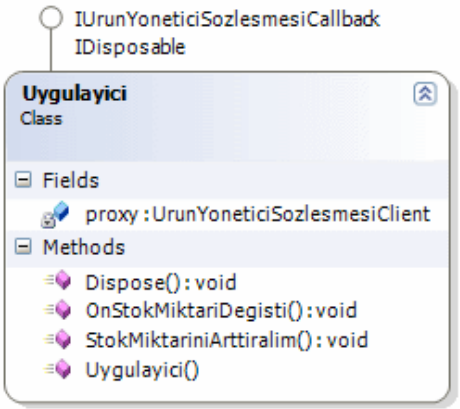


Servis tarafında geri bildirim sözleşmesi var olduğundan, üretilen proxy sınıfı diğer üretimlere göre biraz daha farklıdır. Bu farkı görmek için proxy sınıfı ve beraberinde üretilen tiplerin sınıf diagramına**(class diagram)** bakmak yeterlidir.



Herşeyden önce, istemci uygulamada, servis tarafından çağırılabilecek bir geri bildirim opreasyonun söz konusu olabilmesi için, proxy sınıfının generic **DuplexClientBase** özet sınıfından**(abstract class)** türemiş olması gerekmektedir. Diğer taraftan svcutil.exe bu işlemi otomatik olarak yapmaktadır. örnekte dikkat edilirse, geri bildirim sözleşmesi tipi, DuplexClientBase sınıfının generic parametresi olarak kullanılmaktadır. Bunun dışında, proxy sınıfının aşırı yüklemiş**(overload)** yapıcı metodlarının**(constructor)** tamamında ilk parametreler **InstanceContext** sınıfı tipindendir. Bu son derece doğaldır. Nitekim, servisin istemci üzerindeki bir operasyonu çağırabilmesi için, istemcinin kendisine verdiği içerik referansını kullanması gerekmektedir. Bu nedenle yapıcı metodun ilk parametresi bu referansı istemciden servis tarafına göndermek için kullanılır.

Şimdi gelelim istemci geri bildirim sözleşmesini uyarlayan sınıfa. Bu sınıf geliştirici tarafından yazılmalıdır. örnekte bu amaçla Uygulayici isimli bir sınıf kullanılmıştır.



Uygulayici isimli sınıfın en büyük özelliği IUrunYoneticiSozlesmesiCallback isimli arayüzü**(interface)** uyguluyor olmasıdır. Burada, arayüz adının sonundaki **Callback** kelimesi dikkat çekicidir. Svcutil.exe aracı bunu otomatik olarak

eklemektedir. Bunun dışında Uygulayici sınıfı, **IDisposable** arayüzünüde uygulamaktadır ki burada amaç proxy örneğinin yok edilmesi sırasında servis ile olan bağlantıyı kapatmaya zorlamaktır. Uygulamaci sınıfına ait kodlar aşağıdaki gibidir.

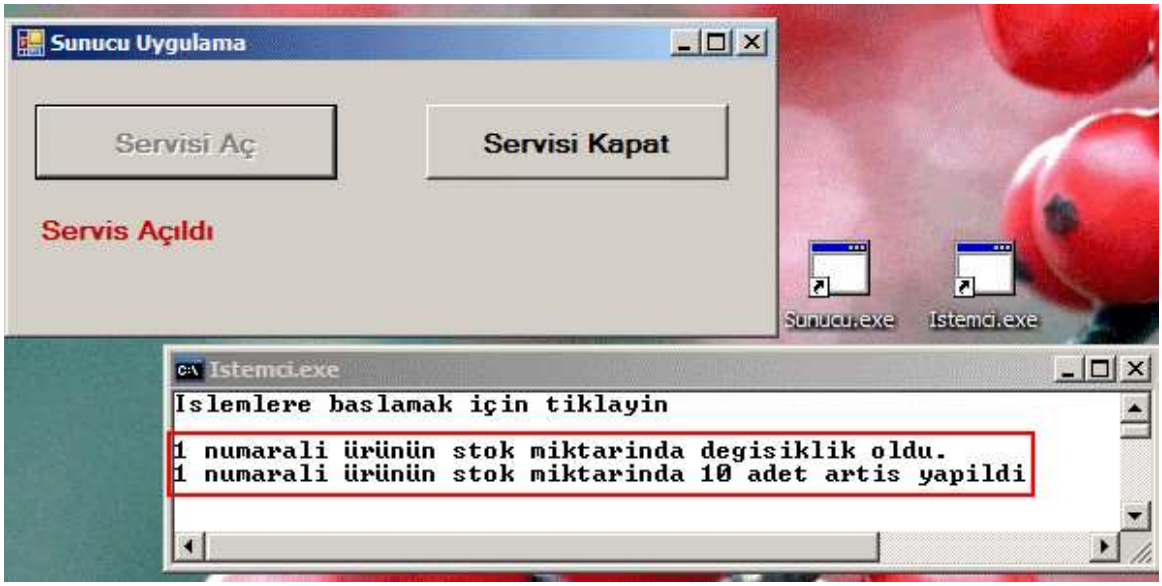
```
using System;
using System.ServiceModel;

namespace Istemci
{
   /* Servisin, istemci tarafında bir geri bildirim operasyonu çağrısı yapabilmesi için geri bildirim sözleşmesini uygulayan bir sınıfın yazılması gerekir. */
   public class Uygulayici:IUrunYoneticiSozlesmesiCallback,IDisposable
   {
      private UrunYoneticiSozlesmesiClient proxy=null;

      public Uygulayici()
      {
         proxy = new UrunYoneticiSozlesmesiClient(new InstanceContext(this), "DefaultBinding_IUrunYoneticiSozlesmesi_IUrunYoneticiSozlesmesi");
      }

      public void StokMiktariniArttiralim(int artisMiktari,int urunNumarasi)
      {
         proxy.StokMiktariniArttir(artisMiktari, urunNumarasi);
         Console.WriteLine("{0} numaralı ürünün stok miktarında {1} adet artış yapıldı",urunNumarasi,artisMiktari);
      }
      #region IUrunYoneticiSozlesmesiCallback Members

      public void OnStokMiktariDegisti(int degisenUrunId)
      {
         Console.WriteLine("{0} numaralı ürünün stok miktarında değişiklik oldu.",degisenUrunId.ToString());
      }

      public void Cikart()
      {
          proxy.AboneligiKaldir(); /* Abonelik çıkartma işlemi yapılmadığ takdirde istemci uygulama istisna mesajları alabilir. Bunun için istemcide açılan bir abonelik var ise bunun bilinçli olarak kapatılmasındada yarar vardır. Cikart metodu buna destek vermesi amacıyla geliştirilmiştir. */
      }

      #endregion
```

```
        #region IDisposable Members

        public void Dispose()
        {
            proxy.Close();
        }

        #endregion
    }
}
```

Sınıfın yapıcı metodu içerisinde proxy sınıfına ait bir nesne örneklenmektedir. Servisin kendisine bağlı istemci üzerinde bir geri bildirim fonksiyonu çağırabilmesi için geri bildirim arayüzünü uygulayan tipe ait referansın, proxy oluşturulurken servise bildirilmesi gerekir. Bunun için proxy sınıfına ait yapıcıların ilk parametreleri daima **InstanceContext** tipinden bir referans alır. Servis tarafında bu içerik örneğinden(InstanceContext) yararlanılarak metod çağrısı yapıldığında, WCF çalışma zamanı bu sınıf içerisindeki ilgili metodu çağıracaktır ki bu örnekte söz konusu metod **OnStokMiktariDegisti** dır. Sınıf aynı zamanda Dispose metodu içerisinde proxy sınıfının Close metodunuda çağırmaktadır. Bunun sebebi şudur; İstemci servis tarafının callback çağrısını beklemeden proxy' yi kapatırsa servis tarafında hatalar oluşabilir. Bu nedenle **Dispose** metodu içerisinden **Close** metodu çağırılır. Artık istemci tarafında Console uygulamasına ait **Main** metodunun içeriğini aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.ServiceModel;

namespace Istemci
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("İşlemlere başlamak için tıklayın");
            Console.ReadLine();

            Uygulayici uyg = new Uygulayici();
            uyg.StokMiktariniArttiralim(10, 1);

            Console.ReadLine();
        }
    }
}
```

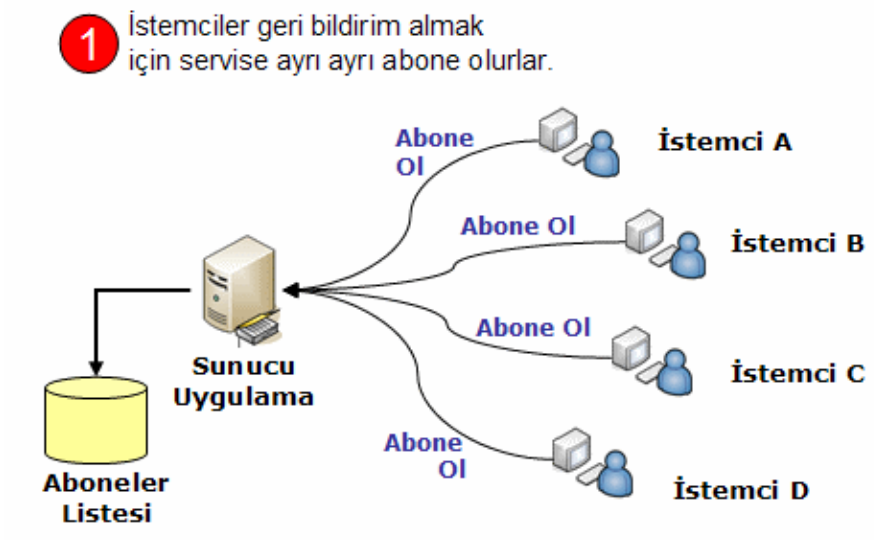
Artık testlere başlanabilir. önce sunucu uygulamanın çalıştırılması ve servisin açılması gerekmektedir. Ardından istemci uygulama çalıştırılarak devam edilirse aşağıdakine benzer ekran görüntüleri ile karşılaşılır.



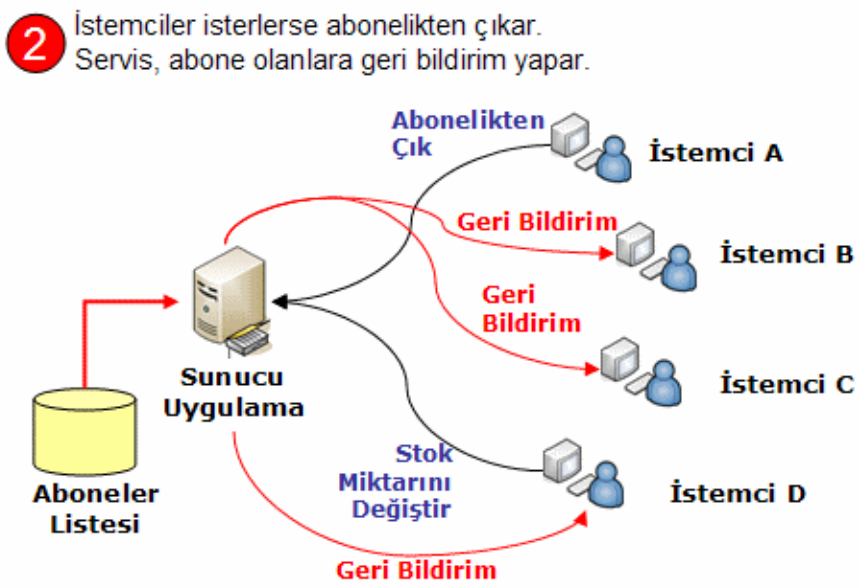
Burada görüldüğü gibi, istemci tarafında servis üzerindeki StokMiktariniArttir metodu çağırldıktan sonra, serviste istemci tarafındaki OnStokMiktariDegisti metodunu çalıştırmıştır.

Buraya kadar geliştirilen örnek sadece servis tarafından istemcideki bir metodun client callback modeli ile nasıl çalıştırabileceğidir. Oysaki gerçek hayat senaryolarında, bir istemcinin aksiyonu sonrası diğer tüm istemcilerinde servis tarafından haberdar edilmesi istenebilir. Bu durumda servisin kendisine bağlı olan istemcileri bilmesi, bunların referanslarını tutması gerekmektedir. Ayrıca istemciler diğer istemcilerin aksiyonlarından servis yoluyla haberdar edilip edilmeyeceklerinide belirleyebilmelidirler. Kısacası, istemcinin kendisini servis tarafına istediği zaman abone edebilmesi**(Subscribe)** ve istediği zamanda abonelikten çıkartabilmesi**(Unsubscribe)** gerekmektedir. Aşağıdaki şekiller ile durum daha net anlaşılabilir.

İlk olarak istemcilere ait örneklerin servis tarafına bildirilmesi bir başka deyişle abone edilmesi söz konusudur.



İstemciler servis üzerinde operayon çağrıları gerçekleştirirler. Bu çağrılardan haberdar olmak isteyenler servise abone olurken istemeyenler abonelikten çıkarlar veya hiç abone olmazlar. Servis ise kendisine abone olanlar üzerinden geri bildirim metodlarını çağırabilir ve böylece bilgilendirme yapabilir. örneğin aşağıdaki senaryoya göre, daha önce abone olmuş olan İstemci A abonelikten çıktıktan sonra, istemci D' nin yapacağı Stok Miktarı Değiştirme işleminden, İstemci B, İstemci C ve İstemci D' nin kendisi haberdar olabilir.



Şimdi bu işlemi adım adım yapmaya çalışalım. İlk olarak servis sözleşmesine, istemcilerin abone olabilmek ve abonelikten çıkabilmek için çağırabilecekleri iki metod tanımı eklenmesi gerekmektedir. Bunun için IUrunYoneticiSozlesmesi arayüzüne**(interface)** aşağıdaki eklemelerin yapılması yeterlidir.

```
using System;
using System.ServiceModel;
```

```
namespace UrunServisi
{
   [ServiceContract(CallbackContract=typeof(IGeriBildirimSozlesmesi))]
   public interface IUrunYoneticiSozlesmesi
   {
      [OperationContract()]
      void StokMiktariniArttir(int gelenMiktar, int urunNo);

      [OperationContract()]
      void AboneOl();

      [OperationContract()]
      void AboneligiKaldir();
   }
}
```

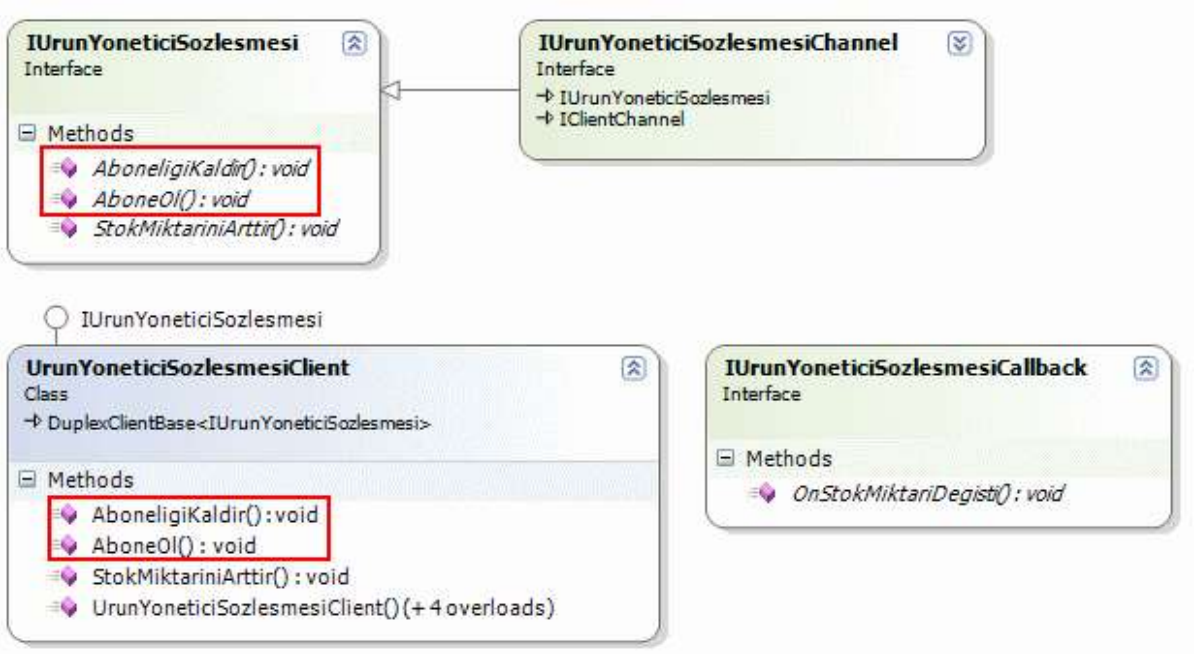
Bu değişiklikler nedeniyle, servis sözleşmesini uygulayan UrunYonetici sınıfı içerisindede AboneOl ve AboneligiKaldir metodlarının yazılması gerekmektedir. İstemcilerin geri bildirim referanslarının servis tarafında saklanabilmesi için generic bir liste koleksiyonu**(List<T>)** kullanılabilir. AboneOl metodu ile bu koleksiyona istemcinin geri bildirim referansı eklenirken, AboneligiKaldir metodu ilede istemciden gelen geri bildirim referansının ilgili koleksiyondan kaldırılması sağlanır. İstemcilere yayınlama yapacak olan ayrı bir metod ise, istenildiğinde servis tarafından koleksiyonda tutulan tüm istemci geri bildirim referanslarına ait metodların çağırılması görevini üstlenir. Bu bilgiler ışığında UrunYonetici sınıfının aşağıdaki gibi dizayn edilmesi yeterlidir.

```
using System;
using System.Data;
using System.ServiceModel;
using System.Data.SqlClient;
using System.Collections.Generic;

namespace UrunServisi
{
   public class UrunYonetici:IUrunYoneticiSozlesmesi
   {
      // İstemcilerin geri bildirim referanslarını tutacak olan List<T> koleksiyonu.
      private static List<IGeriBildirimSozlesmesi> geriBildirimAboneleri = new
List<IGeriBildirimSozlesmesi>();

      #region IUrunYoneticiSozlesmesi Members

      // İstemcilerin servis tarafındaki geri bildirim listesine abone olmasını sağlayan metod.
      public void AboneOl()
```

```
        {
            // Servise o anda bağlı olan istemcideki IGeriBildirimSozlesmesi arayüzünü
uygulamış sınıf referansı alınır.
            IGeriBildirimSozlesmesi geriBildirim =
OperationContext.Current.GetCallbackChannel<IGeriBildirimSozlesmesi>();
            // Eğer List<> kolekisyonunda gelen istemci geri bildirim referansı yoksa eklenir.
            if (!geriBildirimAboneleri.Contains(geriBildirim))
                geriBildirimAboneleri.Add(geriBildirim);
        }

        // İstemcilerin, servis tarafındaki geri bildirim abonleri listesinden çıkartılmasını
sağlayan metod.
        public void AboneligiKaldir()
        {
            IGeriBildirimSozlesmesi geriBildirim =
OperationContext.Current.GetCallbackChannel<IGeriBildirimSozlesmesi>();
            // İstemci geri bildirim referansı, List<> koleksiyonundan çıkartılır.
            geriBildirimAboneleri.Remove(geriBildirim);
        }

        private void AboneleriBilgilendir(int urunNumarasi)
        {
            /* List<> koleksiyonundaki tüm geri bildirim referansları dolaşılır ve sahibi olan
istemci uygulama halen daha çalışıyorsa (ki bu durum State ile tespit edilmektedir), istemci
üzerindeki geri bildirim operasyonu çağırılır. */
            foreach (IGeriBildirimSozlesmesi geriBildirim in geriBildirimAboneleri)
            {
                if (((ICommunicationObject)geriBildirim).State ==
CommunicationState.Opened)
                    geriBildirim.OnStokMiktariDegisti(urunNumarasi);
                else // Eğer istemci canlı değilse, servis tarafındaki List<> koleksiyonundan da
çıkartılır.
                    geriBildirimAboneleri.Remove(geriBildirim);
            }
        }

        public void StokMiktariniArttir(int gelenMiktar,int urunNo)
        {
            string sorgu = "Update Products Set UnitsInStock=UnitsInStock+@GelenMiktar
Where ProductID=@PrdId";
            using (SqlConnection conn = new SqlConnection("data
source=.;database=Northwind;integrated security=SSPI"))
            {
                SqlCommand cmd = new SqlCommand(sorgu, conn);
                cmd.Parameters.AddWithValue("@GelenMiktar", gelenMiktar);
```

```
            cmd.Parameters.AddWithValue("@PrdId", urunNo);
            conn.Open();
            int guncellenen=Convert.ToInt32(cmd.ExecuteNonQuery());
            if (guncellenen > 0)
            {
               AboneleriBilgilendir(urunNo);
            }
         }
      }

      #endregion
   }
}
```

Servis tarafındaki bu değişikliklerin ardından, istemci için gerekli proxy sınıfının **svcutil.exe** aracı yardımıyla yeniden üretilmesi gerekmektedir. Bu üretim işlemi sonrasında istemci tarafına atanan tiplerin son hali aşağıdaki ekran görüntüsündeki gibi olacaktır.



İstemci tarafında geri bildirim sözleşmesini uygulayan Uygulayici isimli sınıfın yapıcı metodu içerisinde AboneOl metodu çalıştırılmaktadır.

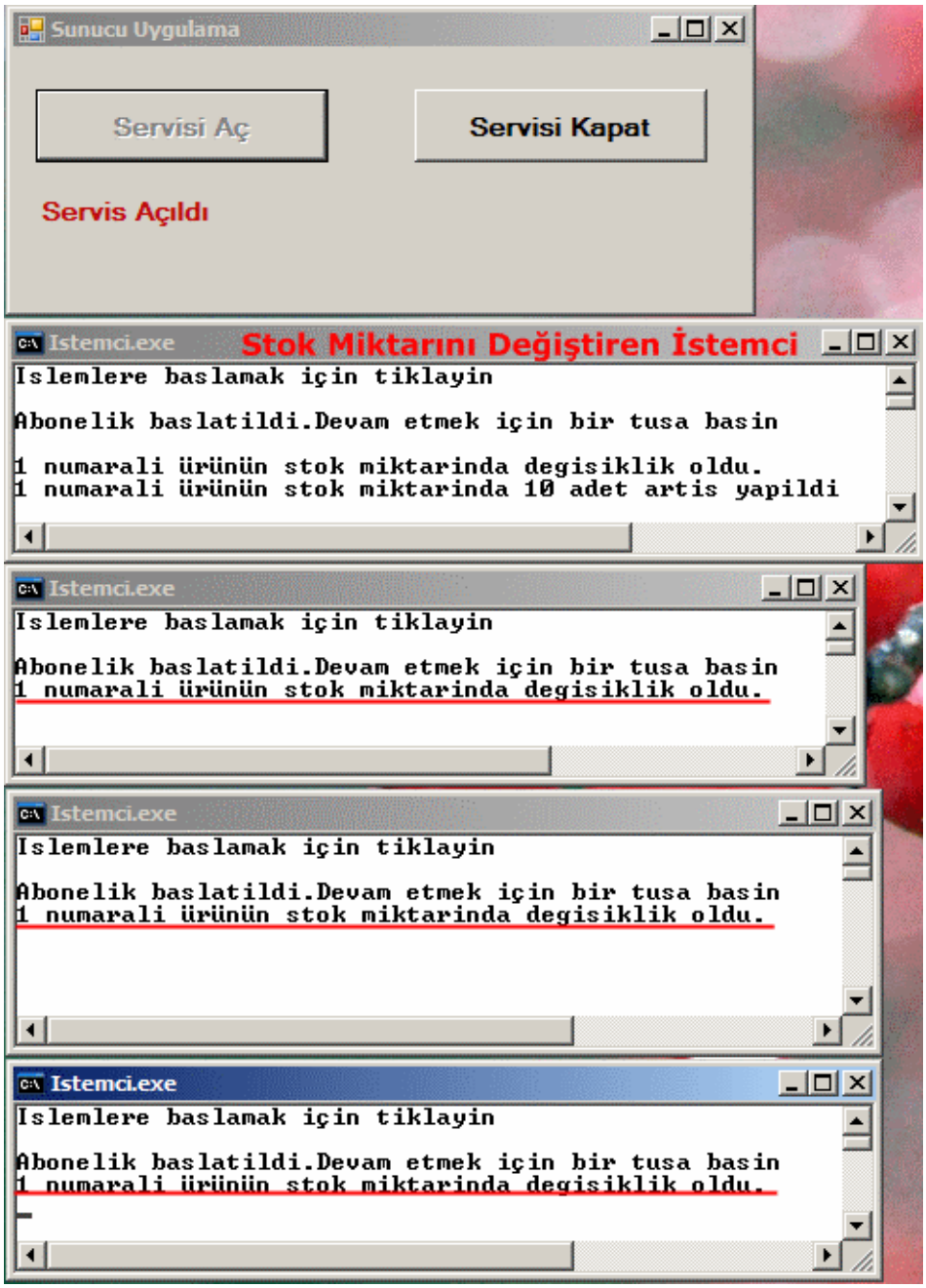
```
using System;
using System.ServiceModel;

namespace Istemci
{
   public class Uygulayici:IUrunYoneticiSozlesmesiCallback,IDisposable
```

```
    {
        private UrunYoneticiSozlesmesiClient proxy=null;

        public Uygulayici()
        {
            proxy = new UrunYoneticiSozlesmesiClient(new InstanceContext(this),
"DefaultBinding_IUrunYoneticiSozlesmesi_IUrunYoneticiSozlesmesi");
            proxy.AboneOl();
        }
        // Diğer kod satırlar
    }
}
```

Main metodu içerisinde ise aşağıdaki test kodları yazılabilir.

```
static void Main(string[] args)
{
    try
    {
        Console.WriteLine("İşlemlere başlamak için tıklayın");
        Console.ReadLine();

        Uygulayici uyg = new Uygulayici();
        Console.WriteLine("Abonelik başlatıldı.Devam etmek için bir tuşa basın");
        Console.ReadLine();
        uyg.StokMiktariniArttiralim(10, 1);
         uyg.Cikart();
    }
    catch (Exception exp)
    {
        Console.WriteLine(exp.Message);
    }
    Console.ReadLine();
}
```

Uygulamayı bu haliyle test etmeye başlayabiliriz. Servis uygulaması çalıştırıldıktan sonra, bir kaçtane istemcinin çalıştırılması ve bunlardan herhangibirinde ürün stok miktarının değiştirilmesi işleminin yapılması yeterlidir. Yapılan testlerde, bir istemcinin urun stok miktarında yaptığı değişiklik sonrası, diğer istemcilerinde ilgili değişiklikten haberdar edildiği görülmektedir. Aşağıdaki ekran görüntüsünde bu testin sonuçları yer almaktadır.



Geliştirilen örnek **TCP** protokolünü kullanan **netTcpBinding** bağlayıcı tipine ele almaktadır. Oysaki HTTP protokolünde varsayılan olaran client callbak desteği olmadığından yazımızın başında bahsetmiştik. Client Callback desteği için çift yönlü **HTTP** kanallarına ihtiyaç vardır. Bir başka deyişle tek yönlü iki adet kanal HTTP protokolü üzerinden kullanılarak istenen sistem kurulabilir. WCF içerisinde bu desteği **WsDualHttpBinding** tipi karşılamaktadır. Yazılan örneği, HTTP destekli hale getirmek için servis ve istemci tarafında yapılması gerekenler vardır. Servis tarafından çift kanallı HTTP desteğini vermesi için WsDualHttpBinding tipini kullanan yeni bir **endPoint** bildirimi yapılmalıdır. Bu amaçla servis uygulamasındaki App.Config dosyasına aşağıdaki bildirim ile yeni endPoint ilave edilir.

<endpoint **address="http://localhost:4500/UrunServisi/UrunServisi.svc"** binding="**wsDualHttpBinding**" name="UrunServisWsEndPoint"
contract="UrunServisi.IUrunYoneticiSozlesmesi" />

İstemci tarafındaki konfigurasyon dosyasında da WsDualHttpBinding tipini kullanan bir endPoint bildirimi yapılmalıdır.

<endpoint address="**http://localhost:4500/UrunServisi/UrunServisi.svc**"
binding="**wsDualHttpBinding**" contract="**IUrunYoneticiSozlesmesi**"
name="WsDualHttpBinding_IUrunYoneticiSozlesmesi" />

Son olarak istemci tarafında yer alan Uygulayici isimli sınıftada bazı önemli değişikliklerin yapılması gerekmektedir. özellikle proxy sınıfına ait nesneyi örneklediğimiz yerde yapılması gereken değişiklikler aşağıdaki gibidir.

```
public Uygulayici()
{
   //proxy = new UrunYoneticiSozlesmesiClient(new InstanceContext(this),
"DefaultBinding_IUrunYoneticiSozlesmesi_IUrunYoneticiSozlesmesi");
   proxy = new UrunYoneticiSozlesmesiClient(new InstanceContext(this),
"WsDualHttpBinding_IUrunYoneticiSozlesmesi");
   WSDualHttpBinding baglayici = (WSDualHttpBinding)proxy.Endpoint.Binding;
   baglayici.ClientBaseAddress = new Uri("http://localhost:4500/UrunServisi/" +
Guid.NewGuid().ToString());
   proxy.AboneOl();
}
```

İstemciler WsDualHttpBinding bağlayıcı tipini kullandıklarında, iki adet tek yönlü **HTTP** kanalı açarlar. WCF çalışma zamanı, servis tarafından gelen talepleri varsayılan olarak **80**numaralı port üzerinden dinleyecektir. Oysaki **80** numaralı portu **IIS(Internet Information Services)** kullanıyor olabilir. Bu durumda istemci uygulama çalışma zamanında hata mesajı verecektir. Nitekim 80 numaralı port başka bir uygulama tarafından kullanılmaktadır. Bunun önüne geçmek için, bağlayıcı tipin üzerinden **ClientBaseAddress** özelliğine geçici bir adres değeri ataması yapılır. Aynı makine üzerinde birden fazla istemci uygulaması çalıştırılabileceğinden, adreslerin benzersizliğinide garanti altına almak adına örnekte **Guid** tipinden yararlanılmıştır. örnek bu son haliyle test edilirse, yine servisin abonelere geri bildirimde bulunabildiği görülecektir. üstelik bu kez durum HTTP üzerinden gerçekleştirilmektedir. Dolayısıyla sistem internet üzerinden kullanılabilir halede gelmiştir.

Elbette bu senaryoda dikkat edilmesi gereken durumlar vardır. Söz gelimi, istemcilerin aboneliklerini kaldırmadıkları durumlar göz önüne alınabilir. örneğin 3 farklı istemci servise abone olsun. Bunlardan birisi aboneliğini kaldırmadan uygulamayı kapatmış olabilir. Oysaki servis tarafında yer alan abone koleksiyonunda 3 adet istemci kayıtlıdır. Dolayısıyla diğer istemcilerden birisinin yaptığı çağrı sonrasında, servis tarafındaki

aboneleri bilgilendirici metod halen daha var olmayan istemci abone koleksiyonundan çıkartılmadığı için istemci tarafında istisna alınmasına neden olabilir. Bunun önüne geçmek için **State** kontrolü yapılsada, yapılan testlerde bazı durumlarda istemcideki proxy ile servis arasındaki bağlantının tam olarak kapanmaması halinde bu tarz bir durum olduğu ortaya çıkmıştır. O halde istemci tarafında yer alan aboneliği kaldırma işleminin rolü son derece önemlidir.

Bu makalemizde, WCF uygulamalarında servislerin istemciler üzerinde nasıl metod çağırabileceğini **Client Callback** tekniği ile incelemeye çalıştık. Bu çağırım işlemini abone bazlı hale getirerek, herhangibir istemcinin bir olay gerçekleştirmesi sonrası servisin diğer istemcileride uyarabilmesini sağladık. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

örnek Uygulama için Tıklayın

Asp.Net Temelleri : Derinlemesine Download/Upload İşlemleri (2007-08-15T17:59:00)

asp.net temelleri,

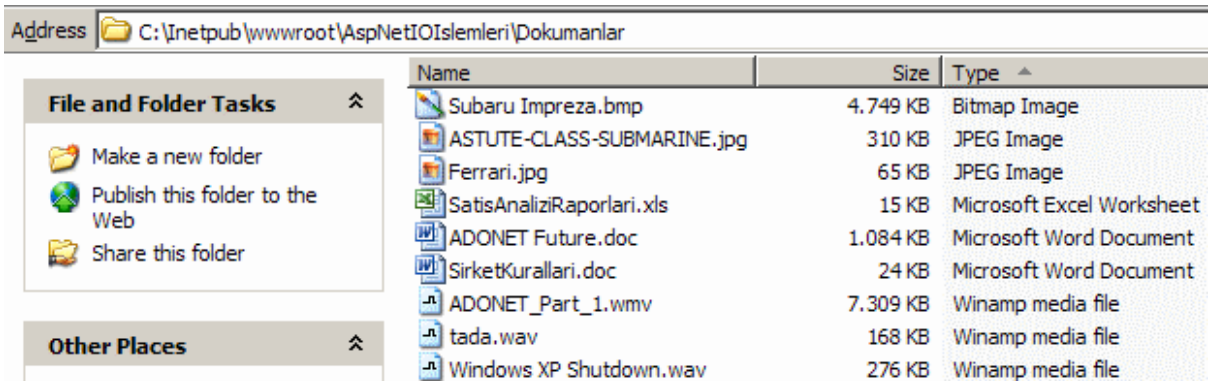
Tatile çıkan herkes, iyi ve dinlendirici geçen günlerin ardından tekrar hayatın akışına kapıldığında kısa süreliğinede olsa adaptasyon problemi yaşar. Tatildeyken hatırlayacağınız gibi hafif ve dinlendirici bir Asp.Net konusu ile ilgilenmeye çalışmıştık. Tatil dönüşündeki adaptasyon sürecinde de benzer nitelikte bir konuyu incelemenin uygun olacağı kanısındayım. Bu yazımızda Asp.Net uygulamalarında sıklıkla başvurduğumuz temel dosya giriş/çıkış **(Input/Output -IO)** işlemlerinden yararlanarak **Download** ve **Upload**işlemlerinin nasıl yapılabileceğini ele almaya çalışacağız.

özellikle web tabanlı içerik yönetim sistemlerinde **(Content Management System)**, kullanıcıların sunucu üzerinde dökümanlar ile etkin bir şekilde çalışabilmeleri sağlanmaktadır. Bu sistemlerde genel olarak kullanıcı kimliği veya rolüne göre istemci bilgisayarlara indirilebilen**(Download)**. Hatta çoğu içerik yönetim sisteminde, istemciler herkesin okuyabileceği yada belirli kişilerin görebileceği şekilde sunucuya döküman aktarma **(Upload)** işlemleride yapabilirler. Söz gelimi bir yazılım şirketinin içerik yönetim sistemi göz önüne alındığında, yazılım departmanındaki geliştiricilerin hazırladıkları teknik dökümantasyonları Upload veya Download edebilecekleri bir ortam hazırlanabilir.

Hangi açıdan bakılırsa bakılsın, web tabanlı olarak yapılan bu işlemler için şirketler büyük ölçekli sistemler tasarlayıp geliştirmiştir. Fakat temel ilke ve yaklaşımlar benzerdir. Dosya indirme veya gönderme işlemleri, web tabanlı bir sistem göz önüne alındığında **HTTP** kurallarına bağlıdır. Dolayısıyla bu kuralların sadece uygulanma ve ele alınma biçimleri programlama ortamları arasında farklılıklar gösterebilir. İşte biz bu makalemizde, Asp.Net 2.0 tarafından olaya bakmaya çalışıyor olacağız. İlk olarak dosya indirme işlemlerini ele alacağız. Sonrasında ise **Asp.Net 2.0** ile gelen **FileUpload** aracı

yardımıyla sunucuya dosya gönderme**(Upload)** işlemlerinin nasıl yapılabileceğini inceleyeceğiz. Ek olarak, upload edilen bir dosyanın kaydedilmeden, sunucu belleği üzerinde canlandırılıp işlenmesinin nasıl gerçekleştirilebileceğine bakacağız. Son olarakta, Upload edilen dosyaların bir veritabanı tablosunda alan(Field) olarak saklanması için gereken adımları göreceğiz. Dilerseniz vakit kaybetmeden dosya indirme süreci ile işe başlayalım.

Dosya indirme (Download) işlemlerinde bilinen **IO** tiplerinden ve **Response** sınıfının ilgili metodlarından yararlanılır. Hatırlanacağı gibi herhangibir resim dosyasını bir web sayfası içerisinde göstermek için üretilen **HTML** içeriği ile oynamak gerektiğinden daha önceki makalemizde bahsetmiştik. Dosya indirme(Download) işlemindede içeriğin tipi**(Content-Type)**, uzunluğu**(Content-Length)** gibi bilgiler önem kazanmaktadır. İlk örneğimizde, IIS üzerinde yayınlanan bir web projesindeki Dokumanlar isimli bir klasörde yer alan dosyaların indirilme işlemlerini gerçekleştirmeye çalışacağız. Bu amaçla web uygulamasına ait dokumanlar klasörü altına aşağıdaki şekildende görüldüğü üzere farklı formatlarda örnek dosyalar atılmasında fayda vardır.



Web uygulamamızın ilk amacı Dokumanlar klasöründeki dosyaların listelenmesini sağlamak olacak. Bu amaçla sayfada bir GridView kontrolü kullanılabilir. Hatta bu kontrolün içeriği **FileInfo**tipinden nesnelerden oluşan generic bir liste koleksiyonundan **(List<FileInfo>)** gelebilir. Böylece istemciler indirebilecekleri dosyalarıda görebilir. Download işleminin gerçekleştirilmesi için GridView kontrolünde bir Select Button' dan faydalanılabilir. İndirme işlemi sırasında indirilmek istenen dosyanın fiziki adresi, uzunluğu gibi bilgiler önemlidir. Bu bilgileri ve fazlasını **FileInfo** sınıfına ait bir nesne örneği yardımıyla elde edebiliriz. Uygulamamıza ait Default.aspx sayfasının içeriği aşağıdaki gibi olacaktır.

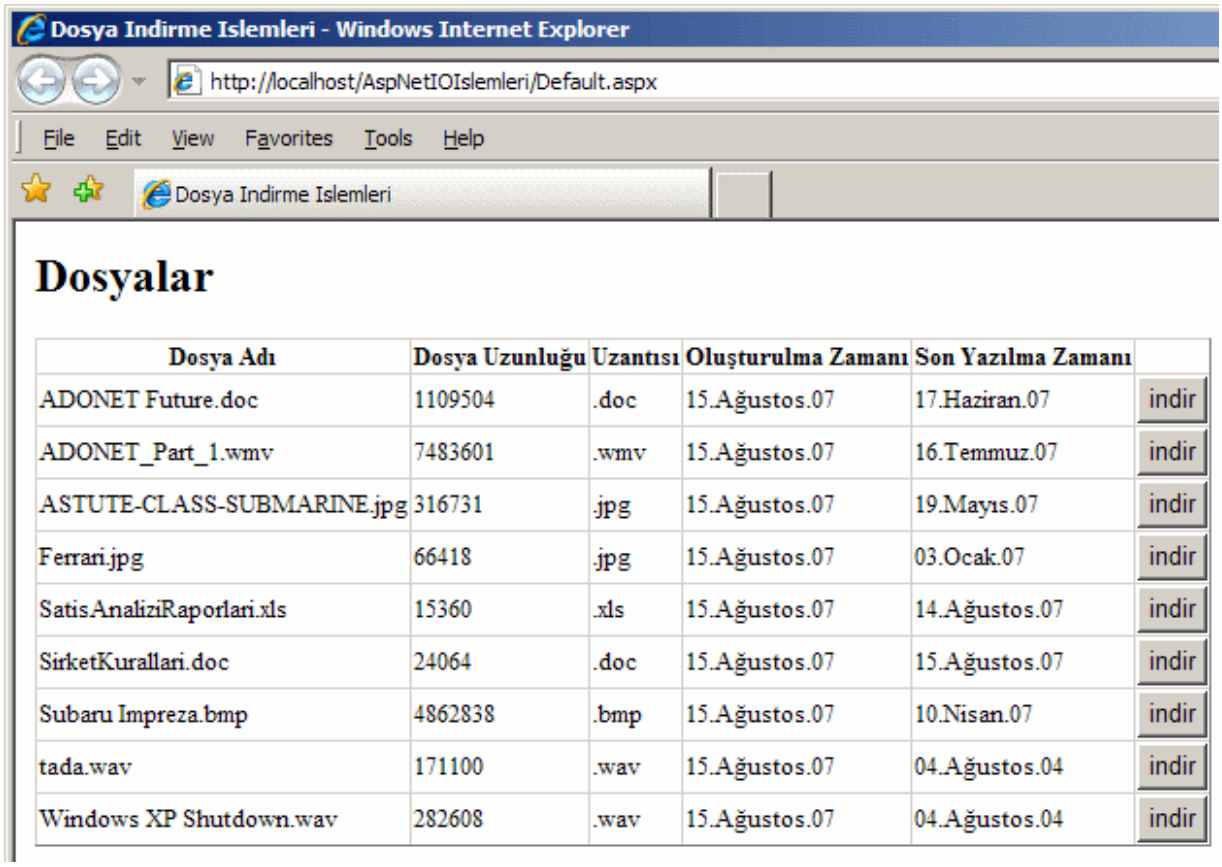
```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Collections.Generic" %>
<%@ Import Namespace="System.IO" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<script runat="server">

protected void Page_Load(object sender, EventArgs e)
{
   if (!Page.IsPostBack)
   {
      string[] dosyalar = Directory.GetFiles(Server.MapPath("Dokumanlar"));
      List<FileInfo> dosyaBilgileri = new List<FileInfo>();

      foreach (string dosya in dosyalar)
      {
         dosyaBilgileri.Add(new FileInfo(dosya));
      }
      grdDosyalar.DataSource = dosyaBilgileri;
      grdDosyalar.DataBind();
   }
}

protected void grdDosyalar_SelectedIndexChanged(object sender, EventArgs e)
{
   string dosyaAdi = Server.MapPath("dokumanlar") + "\\" +
grdDosyalar.SelectedRow.Cells[0].Text;
   FileInfo dosya = new FileInfo(dosyaAdi);

   Response.Clear(); // Her ihtimale karşı Buffer' da kalmış herhangibir veri var ise bunu
silmek için yapıyoruz.
   Response.AddHeader("Content-Disposition","attachment; filename=" +
dosyaAdi); // Bu şekilde tarayıcı penceresinden hangi dosyanın indirileceği belirtilir. Eğer
belirtilmesse bulunulan sayfanın kendisi indirilir. Okunaklı bir formattada olmaz.
   Response.AddHeader("Content-Length",dosya.Length.ToString()); // İndirilecek
dosyanın uzunluğu bildirilir.
   Response.ContentType = "application/octet-stream"; // İçerik tipi belirtilir. Buna
göre dosyalar binary formatta indirilirler.
   Response.WriteFile(dosyaAdi); // Dosya indirme işlemi başlar.
   Response.End(); // Süreç bu noktada sonlandırılır. Bir başka deyişle bu satırdan sonraki
satırlar işletilmez hatta global.asax dosyasında eğer yazılmışsa Application_EndRequest
metodu çağırılır.
}
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>Dosya Indirme Islemleri</title>
   </head>
```

```
<body>
   <form id="form1" runat="server">
      <div>
         <h2>Dosyalar</h2>
         <asp:GridView ID="grdDosyalar" runat="server" AutoGenerateColumns="False"
SelectedRowStyle-
BackColor="Gold" OnSelectedIndexChanged="grdDosyalar_SelectedIndexChanged"
>
            <Columns>
               <asp:BoundField DataField="Name" HeaderText="Dosya Adı" />
               <asp:BoundField DataField="Length" HeaderText="Dosya Uzunluğu" />
               <asp:BoundField DataField="Extension" HeaderText="Uzantısı" />
               <asp:BoundField DataField="CreationTime" HeaderText="Oluşturulma
Zamanı" DataFormatString="{0:dd.MMMM.yy}" HtmlEncode="False" />
               <asp:BoundField DataField="LastWriteTime" HeaderText="Son Yazılma
Zamanı" DataFormatString="{0:dd.MMMM.yy}" HtmlEncode="False" />
               <asp:CommandField ButtonType="Button" SelectText="indir"
ShowSelectButton="True" />
            </Columns>
         </asp:GridView>
      </div>
   </form>
</body>
</html>
```

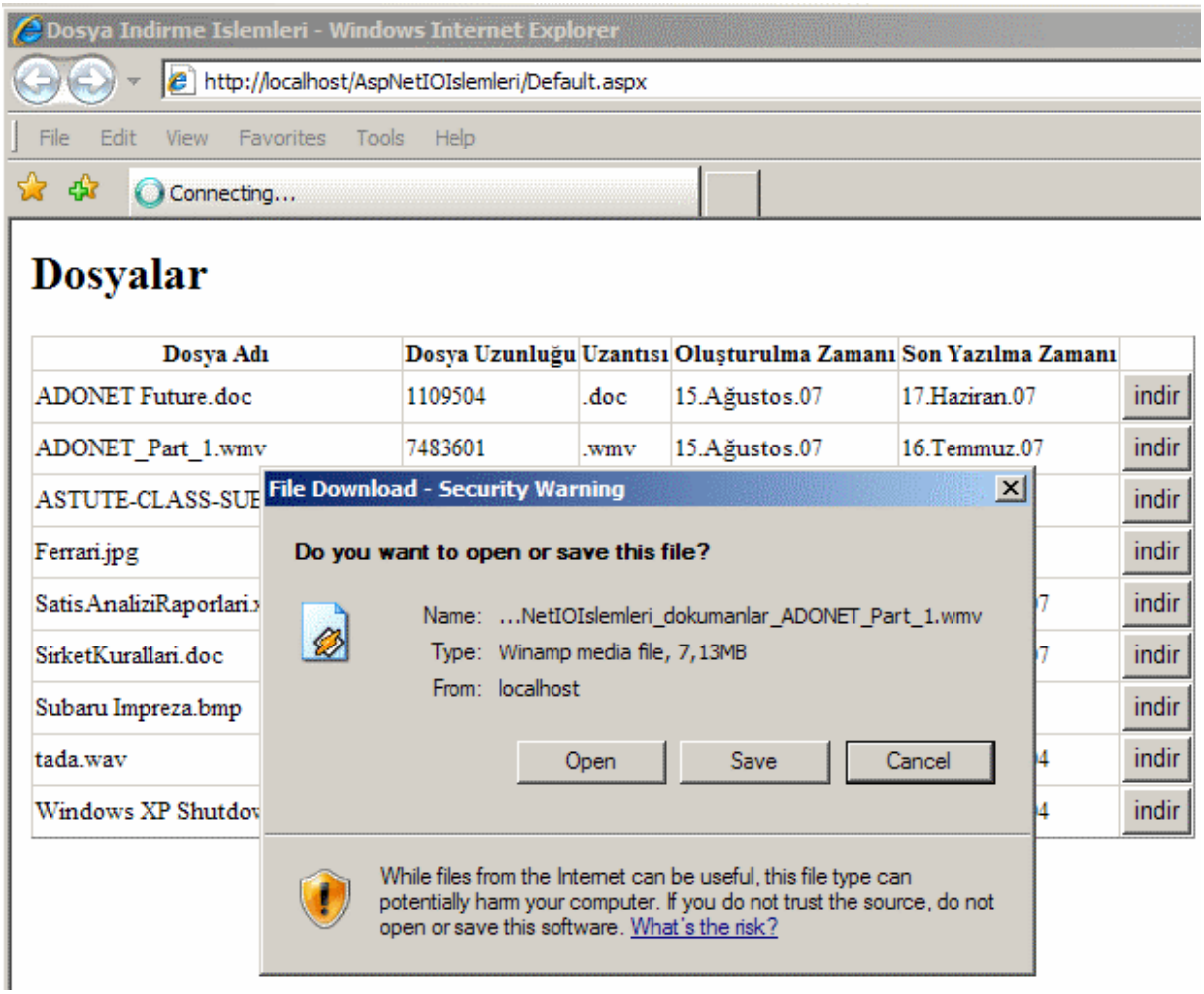
öncelikli olarak sayfamızda neler yaptığımıza kısaca bakalım. Default.aspx sayfası ilk yüklendiğinde (bunu sağlamak için **IsPostBack** özelliği ile kontrol yapılmıştır) Dokumanlar klasöründeki dosyaların adları elde edilmektedir. Bu işlem için **Directory** sınıfının **GetFiles** metodu kullanılmaktadır. Bir web uygulaması söz konusu olduğu için, sanal klasörün karşılık geldiği fiziki adresi bulmak adına **Server.MapPath** metodu ele alınmaktadır. GetFiles metodu parametre olarak belirtilen klasördeki dosya isimlerinin elde edilmesini sağlamaktır. Bu nedenle geriye string tipinden bir dizi döndürür. GridView kontrolü içerisinde, elde edilen bu dosyalara ait bazı temel bilgilerin gösterilmesi hedeflenmiştir. örnekte dosyanın adı**(Name)**, uzunluğu**(Length)**, uzantısı**(Extension)**, oluşturulma**(CreationTime)** ve son yazılma zamanı**(LastWriteTime)** bilgileri ele alınmaktadır. Elde edilen dosya adları aslında fiziki adresleride içermektedir. Bu nedenle ilgili dosya adları **FileInfo** tipinden örneklerin oluşturulmasında kullanılır. Bu nesne örnekleride generic koleksiyonda toplanır. Son olarak GridView kontrolüne veri kaynağı olarak generic liste koleksiyonu atanır. FileInfo ile gelen bilgilerden bazılarını GridView kontrolünde göstermek istediğimizden, **AutoGenerateColumns** özelliğine **false** değeri atanmış ve Columns elementi içerisinde ilgili alanlar açık bir şekilde yazılmıştır. **BoundField** elementlerine ait **DataField** niteliklerinin değerleri FileInfo ile gelen nesne örneklerindeki özellik adları olarak ayarlanmıştır.

İndirilmek istenen dosya için GridView kontrolüne bir adet **CommandField** elementi dahil edilmiştir. Burada seçme işlemi ele alınarak aslında küçük bir hile yapılmaktadır. GridView kontrolünde seçim düğmesine basıldıktan sonra devreye giren **SelectedIndexChanged** olayı içerisinde dosya indirme(Download) işlemi başlatılmaktadır. Teorik olarak Response sınıfının WriteFile metodu ile parametre olarak verilen dosya istemci bilgisayara indirilebilirmektedir. Ancak ön hazırlıklar yapılması gerekmektedir. Bu amaçla, indirilmek istenen dosya adı, GridView kontrolünde seçilen satıra ait ilk hücreden seçildikten sonra, Response sınıfının ilgili metodları ile ön hazırlıklar yapılır. İndirilecek dosya üretilen çıktının **Header** kısmında ele alınmaktadır. Benzer şekilde indirilecek dosyanın uzunluğuda Header kısmına eklenir. Daha sonra içerik tipi belirlenir. **application/octet-stream** değeri, dosyanın ikili**(binary)** formatta indirileceğini belirtmektedir. Bu işlemlerin arkasındanda Response sınıfının **WriteFile** ve **End** metodları sırasıyla çalıştırılır. End metodu, o anki sayfaya ait yaşam sürecinin kesilmesini sağlamaktadır. Bir başka deyişle **Response.End** çağrısından sonra herhangibir kod satırı var ise işletilmeyecektir. Hatta, global.asax dosyasında yer alan**Application_EndRequest** metoduda devreye girecektir. Bu durumu analiz etmeden önceği örneğimizi test edelim. Uygulama çalıştırıldığında aşağıdakine benzer bir ekran görüntüsü ile karşılaşılacaktır.



Burada pek çok ek özellik tasarlanabilir. örneğin uzantıya göre içerik tipi değiştirilebilir. Hatta download işlemi yerine örnek bir dökümanın sayfaya çıktı olarak verilmesi sağlanabilir. PDF içeriklerinin tarayıcıda gösterilmesi buna örnek olarak verilebilir.

Bunların dışında uygulamanın kullanıcıya göre yetkilendirilmesi ve sadece ele alabileceği dosyaları indirebilmesi sağlanabilir. Bu tamamen projenin ihtiyaçlarına ve geliştiricinin kullanıcılara sunmak istediklerine bağlı olarak gelişebilecek bir modeldir. indir başlıklık düğmelerden herhangibirine bastığımızda indirme işleminin aşağıdaki ekran görüntüsünde yer aldığı gibi başladığı görülür.



Şimdi gelelim Response.End metodunun etkisine. Bu durumu analiz etmek için, Response.End sonrasına aşağıdaki gibi örnek bir kod satırı ekleyelim.

```
protected void grdDosyalar_SelectedIndexChanged(object sender, EventArgs e)
{
   string dosyaAdi = Server.MapPath("dokumanlar") + "\\" +
grdDosyalar.SelectedRow.Cells[0].Text;
   FileInfo dosya = new FileInfo(dosyaAdi);

   Response.Clear();
   Response.AddHeader("Content-Disposition","attachment; filename=" + dosyaAdi);
   Response.AddHeader("Content-Length",dosya.Length.ToString());
   Response.ContentType = "application/octet-stream";
```

```
    Response.WriteFile(dosyaAdi);
    Response.End();
    Response.Write("Dosya indirildi");
}
```

Hemen arkasından bilinen yaşam döngüsünü izlemek adına default.aspx sayfasını aşağıdaki gibi değiştirelim.

```
protected void Page_PreInit(object sender, EventArgs e)
{
    Debug.WriteLine("Page_PreInit metodu");
}
protected void Page_Init(object sender, EventArgs e)
{
    Debug.WriteLine("Page_Init metodu");
}
protected void Page_Load(object sender, EventArgs e)
{
    // Diğer kod satırları
    Debug.WriteLine("Page_Load metodu");
}
protected void Page_PreRender(object sender, EventArgs e)
{
    Debug.WriteLine("Page PreRender Metodu");
}
protected void Page_Unload(object sender, EventArgs e)
{
    Debug.WriteLine("Page Unload Metodu");
}

protected void grdDosyalar_SelectedIndexChanged(object sender, EventArgs e)
{
    Debug.WriteLine("Dosya indirme işlemi başlıyor");
    // Diğer kod satırları
    Response.End();
    Debug.WriteLine("Response.End metodu çağırıldı");
    Response.Write("Dosya indirildi");
}
```

Bu değişikliklere ek olarak projeye **global.asax** dosyası ekleyip içerisine **Application_EndRequest** metodunu aşağıdaki gibi dahil edelim.

```
void Application_EndRequest(object sender, EventArgs e)
{
```

```
    Debug.WriteLine("Application EndRequest Metodu çağırıldı");
}
```

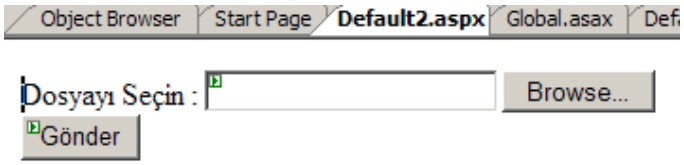
Şimdi uygulamayı **Debug** modda çalıştırıp **output** penceresindeki çıktıları izleyebiliriz. Bir dosya indirme işlemi gerçekleştirildikten sonra sayfanın yaşam döngüsü aşağıdaki gibi çalışacaktır.

```
Page_PreInit metodu
Page_Init metodu
Page_Load metodu
Dosya indirme işlemi başlıyor
  A first chance exception of type 'System.Threading.ThreadAbortException' occurred in mscorlib.dll
  An exception of type 'System.Threading.ThreadAbortException' occurred in mscorlib.dll but was not handled in user code
Page Unload Metodu
Application EndRequest Metodu çağırıldı
```

çok doğal olarak GridView kontrolü üzerindeki düğmeye basıldığında sayfanın sunucuya tekrar gitmesi ve işlenmesi söz konusudur. Bu nedenle süreç **Page_PreInit** ile başlamaktadır. Ancak dikkat edilecek olursa Response.End çağrısından sonraki satırlar devreye girmemiştir. Debug penceresine ve tarayıcıdaki çıktıya herhangibir kod yazılmamıştır. Dahası,
bir **exception(System.Threading.ThreadAbortException)** fırlatılmış ve sayfa yaşam döngüsü **Page_PreRender** metodunu işletmeden doğrudan **Page_Unload** olayını işletmiş ve arkasından **global.asax** dosyasındaki **Application_EndRequest** devreye girmiştir. Elbetteki üretilen istisna**(exception)** Asp.Net çalışma ortamı tarafından görmezden gelinmektedir. Bu nedenle istemci herhangibir şekilde hata mesajı ile karşılaşmaz.

Gelelim makalemizin ikinci konusuna. İndirme işlemleri kadar Upload işlemleride önemlidir. Tabi burada istemcilerin her dosya tipini veya çeşidini sunucuya göndermesi doğru olmayabilir. Kapalı ağ(**intranet)** sistemlerinde bu söz konusu olabilir. Nitekim kimin handi dosyayı Upload ettiğinin belirlenmesi dışında, bu kişiye ulaşılmasıda kolaydır :). Ancak **internet** tabanlı daha geniş sistemlerde her ne kadar kullanıcılar tespit edilebilsede, kötü niyetli istemcilerin varlığı nedeniyle sistemin genelini tehlikeye atmamak adına tedbirler almak doğru bir yaklaşım olacaktır. Biz tabiki basit olarak Upload işlemlerini ele alacağız ve bahsettiğimiz güvenlik konularını şimdilik görmezden geleceğiz.

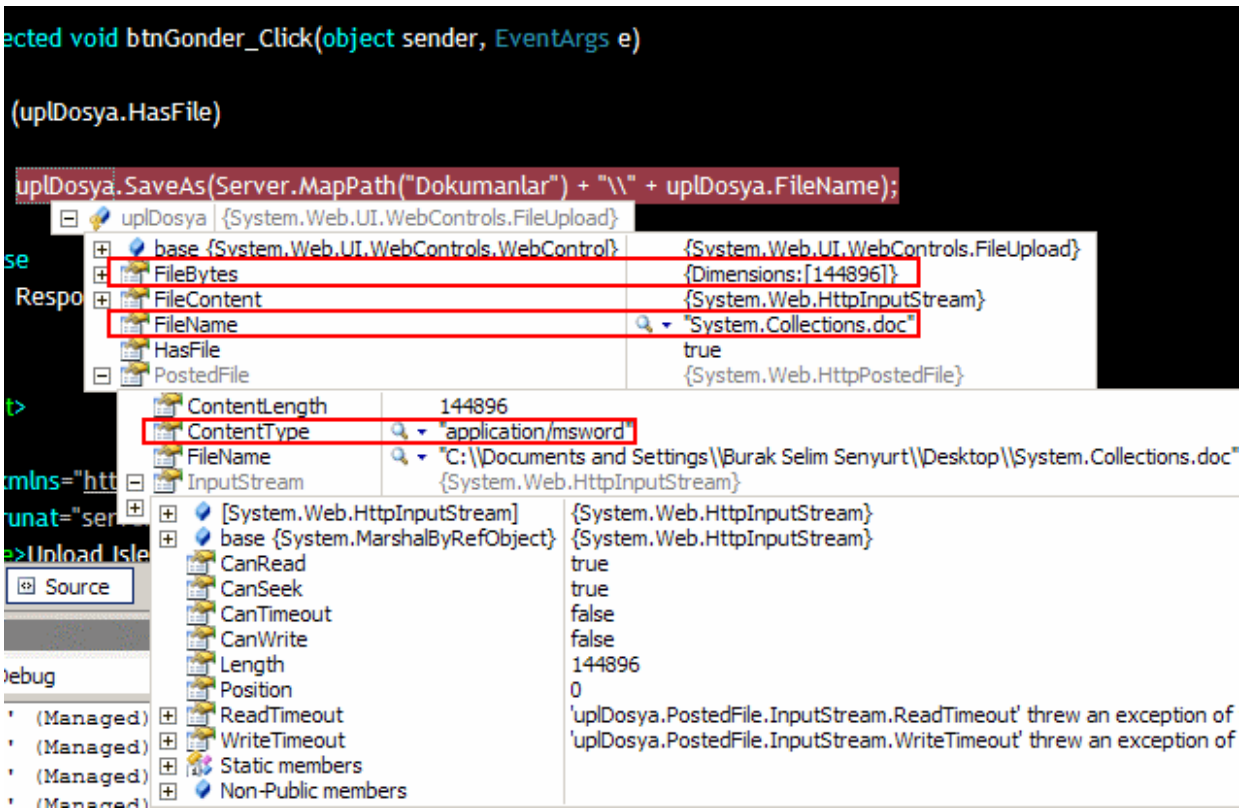
Upload işlemlerini kolaylaştırmak adına Asp.Net 2.0, **FileUpload** isimli bir kontrol getirmektedir. Bu kontrol basit olarak istemcinin Upload etmek istediği dosyayı seçebilmesini sağlamaktadır. Bu seçim işlemi ile birlikte, sunucuya gönderilmek istenen dosyaya ait bir takım bilgilerde FileUpload kontrolünce elde edilir. örneğin içeriğin tipi kontrol edilerek sadece bazı dosyaların gönderilmesine izin verilebilir. İlk olarak web uygulamamıza aşağıdaki gibi Default2.aspx sayfasını ekleyelim.



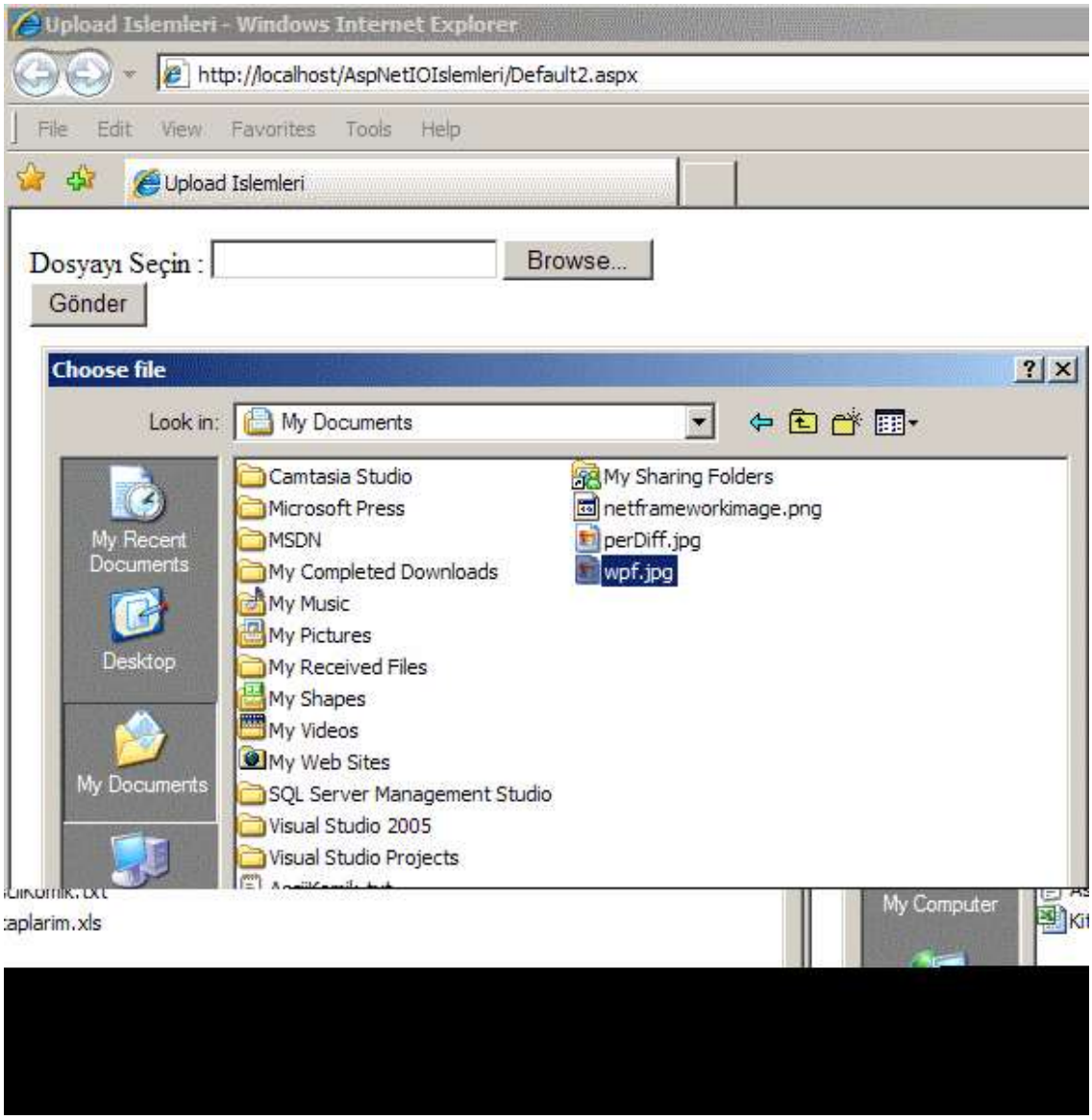

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

   protected void btnGonder_Click(object sender, EventArgs e)
   {
      if (uplDosya.HasFile)
      {
         uplDosya.SaveAs(Server.MapPath("Dokumanlar") + "\\" +
uplDosya.FileName);
      }
      else
         Response.Write("Upload edilecek dosya yok");
   }

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>Upload Islemleri</title>
   </head>
<body>
   <form id="form1" runat="server">
      <div>
         Dosyayı Seçin : <asp:FileUpload ID="uplDosya" runat="server" />
         <br />
         <asp:Button ID="bntGonder" runat="server"
Text="Gönder" OnClick="btnGonder_Click" />
      </div>
   </form>
</body>
</html>
```

öncelikli olarak neler yaptığımıza bir bakalım. FileUpload kontrolü ile dosya seçilmesi, gönderme işlemi için yeterli değildir. Sayfanın sunucuya doğru gönderilmesi gerekmektedir. Bu işi basit olarak bir Button kontrolü üstelenebilir. Button kontrolümüze

ait Click olay metodunda ise öncelikli olarak seçili bir dosya olup olmadığı **HasFile** özelliği ile kontrol edilmektedir. Bu kontrol, dosya seçmeden gönderme işlemi yapıldığı takdirde oluşacak hataların önüne geçilmesini sağlamaktadır. Eğer seçili olan bir dosya var ise basit olarak **FileUpload** sınıfının**SaveAs** metodu çağırılır. SaveAs metoduna dosyanın yol adresinin fiziki olarak verilmesi gerekmektedir. Bu nedenle yine **Server.MapPath** ile Dokumanlar klasörünün fiziki adresi elde edilir. FileUpload kontrolünün **FileName** özelliği ile seçilen dosyanın adı yakalanmakta ve fiziki adresin sonuna eklenmektedir. Eğer kod Debug edilirse, dosya seçildikten sonra Upload işlemi için düğmeye basıldığında FileUpload kontrolü üzerinde, seçilen dosyaya ilişkin çeşitli ek bilgilere ulaşılabildiği görülür.



Söz gelimi dosyanın tipi hakkında fikir elde etmek için **ContentType** özelliğine bakılabilir. Buna göre belirli dosya tiplerinin indirilmesine izin verilmesi istenen durumlara karşı tedbirler alınabilir. Gönderilecek dosyanın boyutu ile ilgili olarak bir kısıtlama getirilmek isteniyorsa içerik uzunluğu **ContentLength** özelliği ile tedarik edilerek gerekli değişiklikler yapılabilir. Şimdi örneği deniyerek devam edelim. Basit olarak bir döküman dosyasını aşağıdaki gibi seçip Upload etmeyi deneyeceğiz.



Görüldüğü üzere Browse işleminden sonra otomatik olarak standart **Choose File** iletişim penceresi (Dialog Box) ile karşılaştık. örnekte bir resim dosyası seçilmiştir. Seçim işleminden sonra Gönder düğmesine basılırsa dosyanın başarılı bir şekilde Dokumanlar klasörü altına yazıldığı görülecektir.

NOT : *Upload işlemleri sırasında yetki problemi nedeni ile IIS altındaki herhangibir klasöre dosya yazma işlemi sırasında hata mesajı alınabilir. Bu durumda* ***ASPNET*** *kullanıcısına ilgili klasöre yazma hakkı verilmesi gerekebilir.*

Upload işlemleri sırasında dikkat edilmesi gereken kritik bir sayı vardır. **4096 byte**. Yani 4 megabyte. Boyutu bu değerin üzerindeki bir dosyayı Upload etmek istediğimizde Asp.Net ortamı bir hata üretir ve dosyanın sunucuya gönderilmesine izin vermez. Ne yazıkki hata üretimi kodların işletilmesinden önce gerçekleşir. Bu nedenle kullanıcıya anlamlı bir mesaj gösterilmeside pek mümkün olmamamaktadır. Eğer 4 megabyte üzerinde dosyaların upload edilebilmesini başka bir deyişle izin verilen limite kadar olan dosyaların gönderilebilmesini

istiyorsak **web.config** dosyası içerisinde **httpRuntime** elementinin ayarlanması gerekmektedir. **system.web** boğumu içerisinde yer alan httpRuntime elementi sayesinde, http çalışma zamanına ait çeşitli ayarlamalar yapılabilmektedir. Bizim ihtiyacımız olan dosya büyüklüğü sınıfı ayarı için örneğimizde **web.config** dosyasını aşağıdaki gibi düzenlememiz yeterlidir.

```
<?xml version="1.0"?>
<configuration>
   <appSettings/>
   <connectionStrings/>
   <system.web>
      <httpRuntime maxRequestLength="51200"/>
      <compilation debug="true"/>
      <authentication mode="Windows"/>
   </system.web>
</configuration>
```

Burada yapılan ayarlamaya göre istemciler 50 megabyte' a kadar dosyaları sunucuya gönderebilecektir. Upload işlemlerini içerik yönetim sistemlerinde resim, döküman veya örnek uygulamaların, programların gönderilmesinde, grafik kütüphanesi tarzındaki sistemlerde çeşitli formatta resim veya akıcı görüntü formatlarının gönderilmesinde ve buna benzer durumlarda kullanmak yaygındır.

Gelelim makalemizin üçüncü konusuna. Yine istemciden sunucuya doğru bir dosya gönderme işlemi gerçekleştirmeyi hedeflediğimizi düşünelim. Ancak bu sefer XML tabanlı bir dosyayı gönderiyor olacağız. Bu dosyanın özelliği, bizim istediğimiz şekilde tasarlanmış olması dışında, sunucu tarafında anında işlenecek olmasıdır. Söz gelimi, XML dosyası içerisinde dinamik olarak sayfaya yüklenmesi istenen kontrollere ait bilgiler yer alabilir. Bu durumda Upload edilen XML dosyasının sunucu tarafında işlenerek bir çıktı üretilmesi gerekmektedir. Bir başka deyişle istemciden sunucuya gönderilen sayfayı, fiziki olarak yazmadan işlemek ve kullanmak istediğimizi göz önüne alıyoruz. Peki bu sistemi nasıl yazabiliriz? İlk olarak istemcinin göndereceği basit XML dökümanını hazırlayarak başlayalım. örnek olarak aşağıdaki gibi bir içerik düşünülebilir.

```
<?xml version="1.0" encoding="utf-8"?>
<Kontroller>
   <Kontrol tip="MetinKutusu" id="metinKutusu1"/>
   <Kontrol tip="Dugme" metin="Gonder" id="gonder1"/>
   <Kontrol tip="Label" metin="Ad" id="ad1"/>
</Kontroller>
```

Olayı basit bir şekilde ele almak adına Xml içeriğini mümkün olduğu kadar basit düşünmeye çalıştık. Elbetteki çok daha karmaşık ve daha çok parametre sunan bir Xml dökümanı söz konusu olabilir. Şimdi bu dosyayı nasıl ele alacağımıza bakalım. Dikkat edilmesi gereken noktalardan birisi, Upload edilecek dökümanın **XML** formatında olması

gerekliliğidir. Bunu sağlamak için, içerik tipine**(ContentType)** bakmak gerekecektir. Sonrasında ise Upload edilen dosyanın Framework içerisinde yer alan XML tipleri yardımıyla ele alınması yeterlidir. Sonuç olarak Default3.aspx dosyamızın içeriği aşağıdaki gibi olacaktır.

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Xml" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

   protected void btnGonder_Click(object sender, EventArgs e)
   {
      if (uplKontroller.HasFile)
      {
         HttpPostedFile dosya = uplKontroller.PostedFile;
         if (dosya.ContentType == "text/xml")
         {
            XmlDocument doc = new XmlDocument();
            doc.Load(dosya.InputStream);
            XmlNodeList kontroller=doc.GetElementsByTagName("Kontrol");
            foreach (XmlNode kontrol in kontroller)
            {
               switch (kontrol.Attributes["tip"].Value)
               {
                  case "Dugme":
                     Button btn = new Button();
                     btn.ID = kontrol.Attributes["id"].Value;
                     btn.Text = kontrol.Attributes["metin"].Value;
                     phlKontroller.Controls.Add(btn);
                     break;
                  case "MetinKutusu":
                     TextBox txt = new TextBox();
                     txt.ID = kontrol.Attributes["id"].Value;
                     phlKontroller.Controls.Add(txt);
                     break;
                  case "Label":
                     Label lbl = new Label();
                     lbl.ID = kontrol.Attributes["id"].Value;
                     lbl.Text = kontrol.Attributes["metin"].Value;
                     phlKontroller.Controls.Add(lbl);
                     break;
               }
```

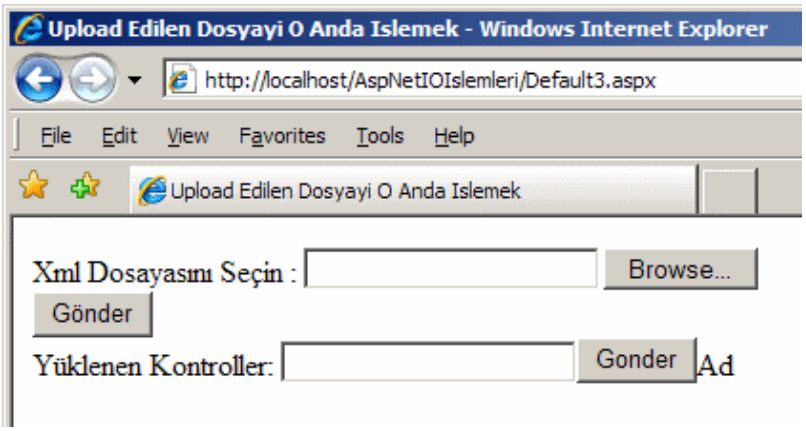
```
            }
         }
         else
            Response.Write("Dosya içeriği XML olmalıdır");
      }
      else
         Response.Write("Dosya bulunamadı");
   }

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>Upload Edilen Dosyayi O Anda Islemek</title>
   </head>
<body>
   <form id="form1" runat="server">
      <div>
         Xml Dosayasını Seçin : <asp:FileUpload ID="uplKontroller" runat="server"
/>
         <br />
         <asp:Button ID="btnGonder" Text="Gönder"
runat="server" OnClick="btnGonder_Click" />
         <br />
         Yüklenen Kontroller:
         <asp:PlaceHolder ID="phlKontroller" runat="server" />
      </div>
   </form>
</body>
</html>
```

Dilerseniz kod içerisinde neler yaptığımıza kısaca bakalım. İlk olarak düğmeye basıldığında HasFile özelliği ile seçilen bir dosya olup olmadığını kontrol ediyoruz. Sonrasında ise FileUpload kontrolünün **PostedFile** özelliğinden yararlanıp seçilen dosyaya ait bazı temel bilgileri taşıyan **HttpPostedFile** tipine ait nesne örneğini elde ediyoruz. Bu nesne örneği üzerinden **ContentType** özelliğine geçerek içeriğin **XML** olup olmadığını **text/xml** eşleştirmesi ile kontrol ediyoruz. Sonrasında ise okunan dosya içeriğini XmlDocument nesnesine yüklüyoruz. **XmlDocument** nesne örneğine ait Load metodunun parametresi olarak bir Stream kullanılabildiğinden, **HttpPostedFile** nesne örneğinin **InputStream** özelliğinden yararlanıyoruz. Son olarak yüklenen XML dökümanı içerisinde dolaşarak ilgili nitelikleri okuyor ve dinamik olarak oluşturulan kontrolleri, sayfadaki **PlaceHolder** kontrolünün Controls koleksiyonuna ekliyoruz.

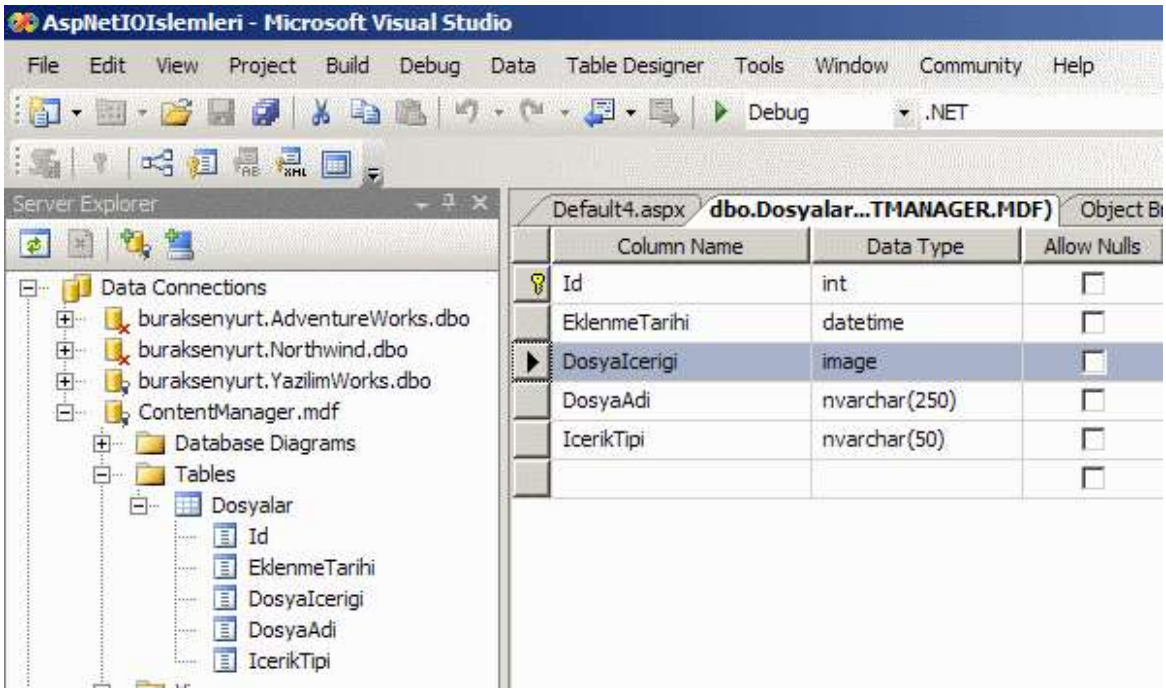
NOT : *İşlemlerin daha sağlıklı olması açısından yüklenen XML içeriğinin belirli kurallara uygun olup olmadığı bir şema dosyası (**XSD** olabilir örneğin) yardımıyla kontrol edilebilir.*

Söz gelimi gelen XML dökümanının yapısı, element adları, nitelik adları veya tipleri bu şema yardımıyla denetlenip, kontrollerin belirli standartlara göre okunabilmesi sağlanmış olunur. Şema kontrolünün nasıl yapılabileceğine dair daha önceki bir makalemizden yararlanabilirsiniz.

Uygulamayı çalıştırdığımızda ve istemci tarafından Kontrollerim.xml dosyasını yüklediğimizde aşağıdaki ekran görüntüsünde olduğu gibi kontrollerin başarılı bir şekilde üretilip sayfaya yüklendiğini görebiliriz.



Sıra geldi makalemizin son konusuna. İçerik yönetimi adına, istemcinin sunucuya gönderdiği dosyaların veritabanı üzerindeki bir tabloda tutulması istenebilir. Eğer ilişkisel veritabanı sistemi**(Relational Database Management System)** söz konusu ise, dosyaların tabloda tutuluyor olması taşıma işlemlerini kolaylaştırabileceği gibi, içerik güvenliğininde de daha etkin bir seviyede yapılabilmesini sağlayacaktır. *(Tabi tersine saklanacak dosya boyutlarına göre veritabanı daha hızlı şişecektir.)* Bu tarz bir ihtiyacın çıkış noktası son derece basittir. Her zaman olduğu gibi bir **FileUpload** kontrolü ile istemciye dosya seçtirilmeli daha sonra ilgili içerik sunucu tarafında işlenerek veritabanındaki ilgili tabloya yazdırılmalıdır. Burada çalıştıralacak komut dışında tablodaki alan tipide önemlidir. **Text** tabanlı bir içerik gönderilecek olsada, tablo tarafında **image** veya **VarBinary** tipinden alanlar tutmak Unicode tutarlılığı açısından daha doğru bir yaklaşım olacaktır. Dilerseniz bir örnek ile bu durumu incelemeye çalışalım. İlk olarak içeriği saklayacağımız basit bir tablo oluşturalım. Bunun için SQL Server 2005 üzerinde aşağıdaki gibi bir tablo göz önüne alınabilir.



Dosyalar isimli tabloda, sunucuya gönderilen dosya içeriğini saklamak için **image** tipinden bir alan kullanılmaktadır. Bunlara ek olarak dosyanın eklenme tarihi,içeriğin tipi ve dosya adı bilgileride yer almaktadır. Söz konusu alanlar dışında, web sitesinde kullanılan **doğrulama(Authentication)** sistemine göre, **Upload** işlemini yapan kullanıcının bilgilerinin saklanması hatta varsa**Membership** gibi kullanıcı tablo sistemleri ile ilişkilendirilmeside mümkün olabilir. Gelelim yükleme işlemini tabloya yazacak kodlarımıza.

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

   protected void btnGonder_Click(object sender, EventArgs e)
   {
      if (uplDosya.HasFile)
      {
         string conStr
= ConfigurationManager.ConnectionStrings["ConStr"].ConnectionString;
         using (SqlConnection conn = new SqlConnection(conStr))
         {
            SqlCommand cmd = new SqlCommand("Insert into Dosyalar
```

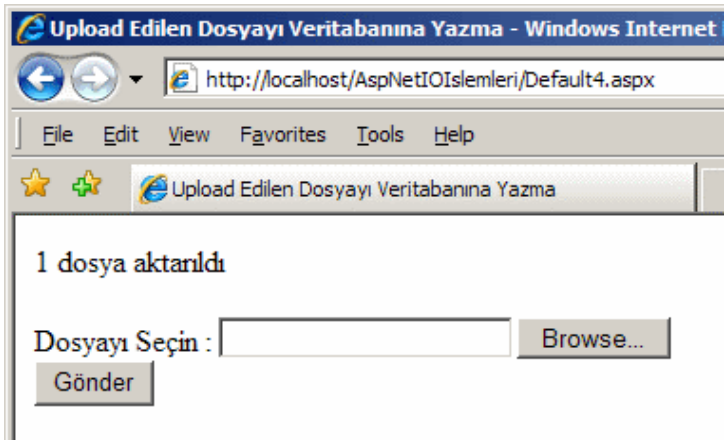
```
(EkleNmeTarihi,DosyaIcerigi,DosyaAdi,IcerikTipi)
Values    (@EklenmeTarihi,@DosyaIcerigi,@DosyaAdi,@IcerikTipi)", conn);
            cmd.Parameters.AddWithValue("@EklenmeTarihi", DateTime.Now);
            cmd.Parameters.AddWithValue("@DosyaIcerigi", uplDosya.FileBytes);
            cmd.Parameters.AddWithValue("@DosyaAdi", uplDosya.FileName);
            cmd.Parameters.AddWithValue("@IcerikTipi",
uplDosya.PostedFile.ContentType);
            conn.Open();
            int sonuc=cmd.ExecuteNonQuery();
            Response.Write(sonuc + " dosya aktarıldı");
         }
      }
      else
         Response.Write("Dosya seçmelisiniz");
   }

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>Upload Edilen Dosyayı Veritabanına Yazma</title>
   </head>
<body>
   <form id="form1" runat="server">
      <div>
         Dosyayı Seçin : <asp:FileUpload ID="uplDosya" runat="server" />
         <br />
         <asp:Button ID="btnGonder" Text="Gönder"
runat="server" OnClick="btnGonder_Click" />
      </div>
   </form>
</body>
</html>
```

Yine kodları kısaca incelemekte fayda var. Her zamanki gibi, istemcinin bir dosya seçtiğinden emin olduktan sonra gerekli işlemleri yapıyoruz. Burada önemli olan nokta çalıştırılacak SQL cümlesinde kullanılan parametrelerin değerlerinin nasıl verildiği. Dikkat edilecek olursa, image tipindeki alanın içeriğini verirken **FileUpload** kontrolünün **FileBytes** özelliğinden yararlanıyoruz. Burada dikkat edilmesi gereken noktalardan biriside çok büyük boyutlu dosyaların aktarılması sırasında yaşanabilecek **timeout** sorunudur. Böyle bir durumda kalındığı takdirde bağlantı için timeout sürelerinin arttırılması yoluna gidilebilir yada dosyanın parçalanarak sunucuya gönderilmesi ve burada o şekilde ele alınması sağlanabilir. Uygulama çeşitli tipteki dosyalar ile test edildiğinde başarılı bir şekilde çalıştığı görülecektir. Aşağıdaki ekran görüntüsünde bir kaç dosya tipinin upload edilmesi sonrasındaki durum vurgulanmaktadır.

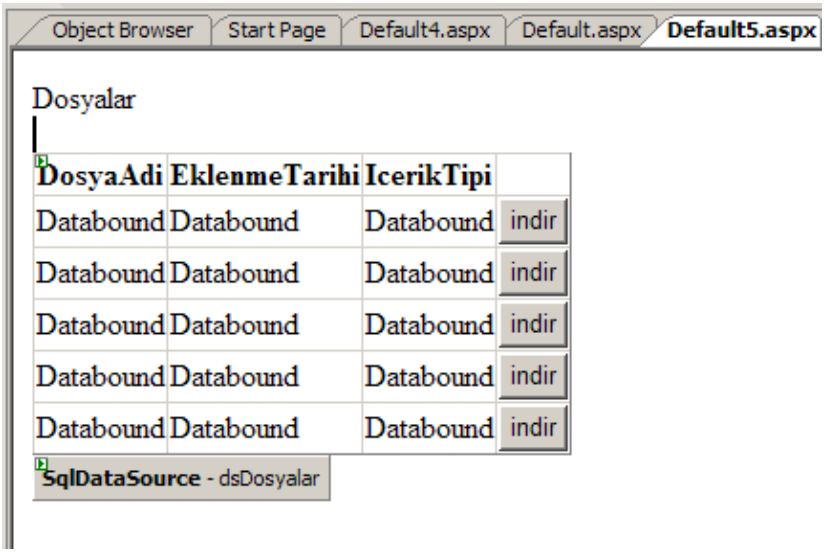
ject Browser | Start Page | **Dosyalar: Qu...TMANAGER.MDF)** | Web.Config | Default4.aspx

Id	EklenmeTarihi	DosyaIcerigi	DosyaAdi	IcerikTipi
2	15.08.2007 13:04:00	<Binary data>	Kontrollerim.xml	text/xml
3	15.08.2007 13:05:07	<Binary data>	System.Collections.doc	application/msword
5	15.08.2007 13:06:52	<Binary data>	wpf.jpg	image/pjpeg
6	15.08.2007 13:06:57	<Binary data>	Kitaplarim.xls	application/vnd.ms-excel
7	15.08.2007 13:07:02	<Binary data>	netframeworkimage.png	image/x-png
8	15.08.2007 13:07:10	<Binary data>	Beethoven's Symphony No. 9 (Scherzo).wma	audio/x-ms-wma
NULL	NULL	NULL	NULL	NULL

Her dosya eklenme işleminden sonrada tarayıcı penceresindeki görüntü aşağıdaki gibi olacaktır.



Elbette Upload edilen içeriklerin, istemciler tarafından indirilmeside gerekecektir. Bu durumda tablo alanındaki dosya içeriğinin stream olarak yazdırılması gerekir. Tabi bunun için **Response** sınıfının **WriteFile** metodu yerine **BinaryWrite** metodunu tercih edeceğiz. *(Alternatif bir yaklaşım olarak, tablodan okunan dosya içeriğinin bir temp dosyaya atılması ve oradanda WriteFile metodu ile yazdırılmasıda düşünülebilir)* Nitekim dosya içerikleri tabloda **binary** olarak tutulmaktadır. öyleyse son olarak bu işlemide nasıl yapabileceğimizi inceleyeceğimiz bir örnek sayfa daha ekleyelim. Default5.aspx sayfamızın içeriği aşağıdaki gibi olacaktır.

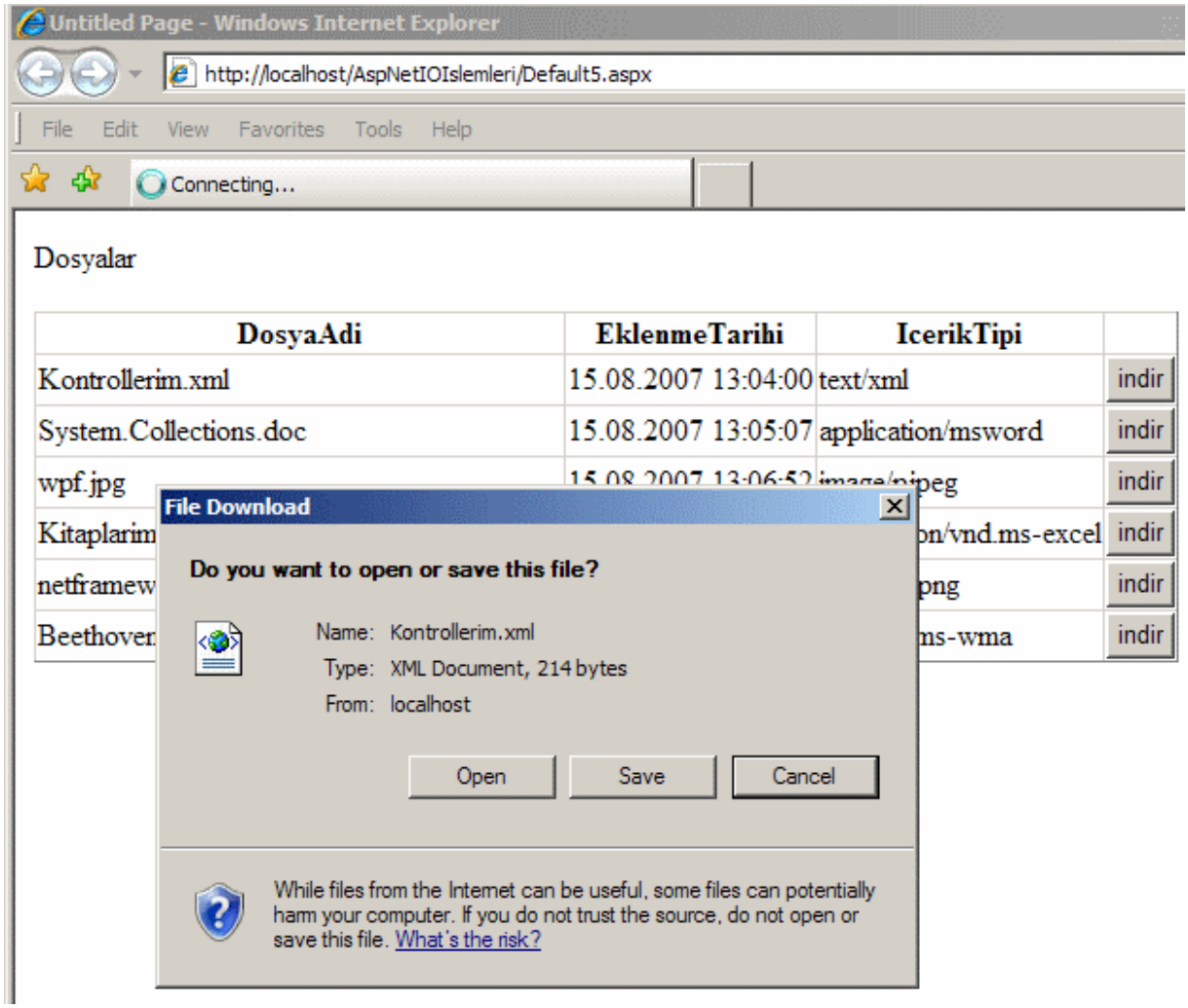



```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.IO" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Configuration" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

   protected void GridView1_SelectedIndexChanged(object sender, EventArgs e)
   {
      using (SqlConnection conn = new
SqlConnection(ConfigurationManager.ConnectionStrings["ConStr"].ConnectionString))
      {
         SqlCommand cmd=new SqlCommand("SELECT
DosyaAdi,DosyaIcerigi,IcerikTipi FROM Dosyalar Where Id=@Id",conn);
         cmd.Parameters.AddWithValue("@Id", GridView1.SelectedValue);
         conn.Open();
         SqlDataReader reader=cmd.ExecuteReader();
         if (reader.Read())
         {
            Response.Clear();
            Response.AddHeader("Content-Disposition", "attachment; filename=" +
reader.GetString(0));
            byte[] dosyaIcerigi = reader.GetSqlBinary(1).Value;
            Response.AddHeader("Content-Length", dosyaIcerigi.Length.ToString());
            Response.ContentType = "application/octet-stream";
            Response.BinaryWrite(dosyaIcerigi);
```

```
        }
        reader.Close();
      }
      Response.End();
   }

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>Untitled Page</title>
   </head>
<body>
   <form id="form1" runat="server">
      <div>
         Dosyalar<br />
         <br />
         <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
DataSourceID="dsDosyalar" OnSelectedIndexChanged="GridView1_SelectedIndexCh
anged" DataKeyNames="Id">
            <Columns>
               <asp:BoundField DataField="DosyaAdi" HeaderText="DosyaAdi"
SortExpression="DosyaAdi" />
               <asp:BoundField DataField="EklenmeTarihi" HeaderText="EklenmeTarihi"
SortExpression="EklenmeTarihi" />
               <asp:BoundField DataField="IcerikTipi" HeaderText="IcerikTipi"
SortExpression="IcerikTipi" />
               <asp:CommandField ButtonType="Button" SelectText="indir"
ShowSelectButton="True" />
            </Columns>
         </asp:GridView>
         <asp:SqlDataSource ID="dsDosyalar" runat="server" ConnectionString="<%$
ConnectionStrings:ConStr %>" SelectCommand="SELECT Id,DosyaAdi,
EklenmeTarihi, IcerikTipi FROM Dosyalar">
         </asp:SqlDataSource>
      </div>
   </form>
</body>
</html>
```

Tasarladığımız sayfayı dilerseniz inceleyelim. Basit olarak Dosyalar tablosunun içeriğini göstermek amacıyla **SqlDataSource** ve **GridView** kontrollerinden faydalanıyoruz. Yine ilk örneğimizde olduğu gibi indirme işlemini başlatmak adına küçük hilemizi yaptık ve bir Select düğmesi kullandık. Burada dosya içeriği tabloda alan olarak tutulduğu için, **SqlCommand** ile verinin çekilmesi gerekiyor. Bunu kolaylaştırmak adına GridView

içerisinde seçilen satıra ait Id alanının değerini almalıyız. Bu amaçla GridView kontrolünün **DataKeyNames** özelliğine Id değerini verdik. Bu değeri **SelectedIndexChanged** metodu içerisinden alarak sorgu cümlesinde parametre olarak kullanıyor ve böylece indirilmek istenen dosyaya ait içeriğin olduğu tablo satırını çekebiliyoruz. Bizim için önemli olan nokta, **binary** içeriği okumak için **SqlDataReader** sınıfının **GetSqlBinary** metodunu kullanıyor olmamız. Bu metod ile dönen tipin **Value**özelliğinden faydalanıp elde edilen **byte[]** dizisini **Response** sınıfının **BinaryWrite** metoduna parametre olarak verdiğimizde yazma işlemi gerçekleştirilmiş oluyor. Sonuç olarak çalışma zamanında istediğimiz sonuca ulaşıyor ve dosya indirme işlemlerini gerçekleştirebiliyoruz.



Bu makalemizde Asp.Net uygulamalarında **Download** ve **Upload** işlemlerini detayları ile incelemeye çalıştık. İlk olarak bir dosyanın indirilme işleminin nasıl yapılabileceğine baktık. Sonrasında ise basit olarak bir Upload işlemi ile sunucuya dosya gönderme olayını ele aldık. Upload işleminin farklı yönlerini ele almak adına, anında sunucu tarafında işleme ve tabloya satır olarak ekleme işlemlerini inceledikten sonra, tablodaki bir binary içeriği indirme sürecine göz attık. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

örnek Uygulama için Tıklayın *(Dosyanın çok yer tutmaması açısından mdf dosyası çıkartılmıştır)*

Asp.Net 2.0 URL Rewriting Hakkında Gerçekler (2007-08-07T18:11:00)

asp.net 2.0,

çok kısa süreliğinede olsa tatilde olduğum şu günlerde yazılım dünyasından kopmak hiç içimden gelmedi. Bu nedenle dinlendiğim zamanlardan arta kalan sürelerde azda olsa bir şeyler karalamak istedim. Sonuç olarak daha hafif ve tatil moduna uygun olacak bir yazı ile yeniden beraberiz. Bu makalemizde Asp.Net 2.0 ile geliştirilen web uygulamalarında, URL eşleştirmelerinin **(Url Mapping)** nasıl düzenlenebileceğini, bir başka deyişle nasıl özelleştirilebileceğini incelemeye çalışacağız. Son kullanıcılar web ortamında, kendi **tarayıcı(browser)**uygulamalarında yer alan adres satırlarında zaman zaman karışık ve uzun URL bilgileri ile karşılaşırlar. Genellikle sorgu katarlarının**(QueryString)** kullanıldığı ve bunların sayılarının çok olduğu durumlarda adres satırlarını okumak gerçekten güçleşebilir. Söz gelimi aşağıdaki URL bilgisini göz önüne alalım.

http://www.azonsitesi.com/urunler.aspx?urunKategori=1&urunAdi=Bilgisayar%20Kitaplari&Sinifi=Ingilizce&BasimYili=2006

Bunun yerine aşağıdaki gibi bir URL bilgisi çok daha kullanışlı ve son kullanıcı açısından okunaklı olabilir.

http://www.azonsitesi.com/Ingilizce/BilgisayarKitaplari/2006Basimi/Goster.aspx

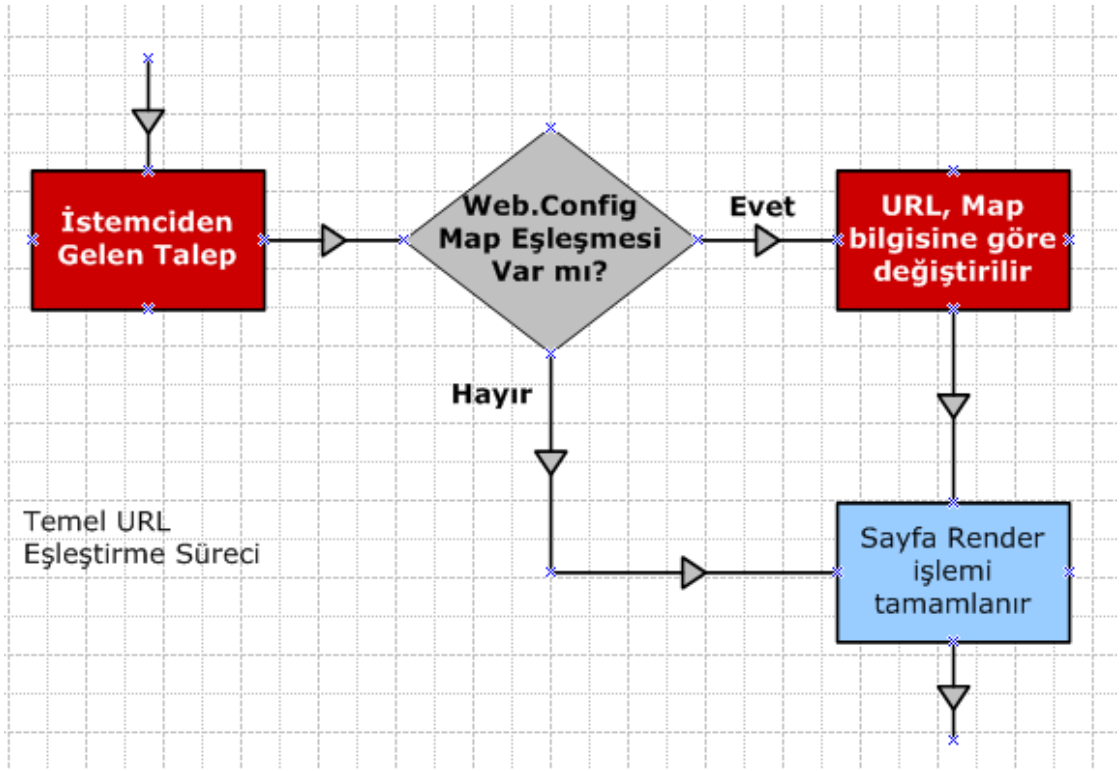
örnekler çoğaltılabilir. Söz gelimi blog sitelerinde, adres satırlarında okunan içeriğe ait bilgilerin QueryString şeklinde durması yerine örneğin**http://buraginblogu/Agustos/7/2007/UrlEslestirme/Oku.aspx** gibi bir formata sahip olması son kullanıcı açısından çok daha cezbedicidir.

Bu ve benzer durumlarda, adres satırındaki bilginin daha kolay anlaşılabileceği hale getirilmesi **son kullanıcı(End User)** için önemli bir hizmettir. Peki bu tarz bir ihtiyaç nasıl karışalanabilir. İlk olarak istemciden gelen adres talebine eş düşecek yeni URL bilgisinin sunucu tarafında ele alınıyor olması gerekir. Sonrasında ise sonuçlar istemci tarayıcı programına istenen formatta gönderilir. **Asp.Net 1.1** kullanıldığı takdirde bu işin çözümü özel **HttpHandler** ve **HttpModule** sınıflarının yazılması ile mümkün olabilir.*(Kendi HttpHandler yada HttpModuler tiplerimizi nasıl yazabileceğimize dair bilgileri daha önceden yayınlanan **makalemizden** takip edebilirsiniz)* **Asp.Net 2.0** mimarisindeyse, sadece URL eşleştirilmelerinin daha kolay yapılabilmesini sağlamak amacıyla **web.config** dosyasında yer alan **system.web** boğumu**(node)** içerisinde ele alınabilecek bir **urlMappings** elementi ve bunun **Configuration API**' sinde karşılığı olan**UrlMappingsSection** sınıfı geliştirilmiştir. Bu sayede konfigurasyon bazında URL

eşleştirmeleri yapılabilmekte ve özel HttpHandler yada HttpModule tipleri yazılmasına gerek kalmamaktadır.

NOT : *Her ne kadar* ***urlMappings*** *elementi veya* ***UrlMappingsSection*** *tipi sayesinde, URL eşleştirmelerinin yapılması kolaylaşmışsada bazı özel durumlarda yine HttpHandler veya HttpModule tipleri geliştirmek gerekebilir. Söz gelimi, urlMappings elementinin doğrudan bir* ***regular expression*** *desteği yoktur. Dolayısıyla benzer yazıma sahip URL bilgileri için ortak eşleştirme yapmak adına element bazında bir hamle yapılması zordur. Bunu sağlamak için HttpModule ve HttpHandler yazmak gerekmektedir. Böyle bir ihtiyaç için kendi HttpModule ve HttpHandler tipinizi yazmaya çalışmanız önerilir.*

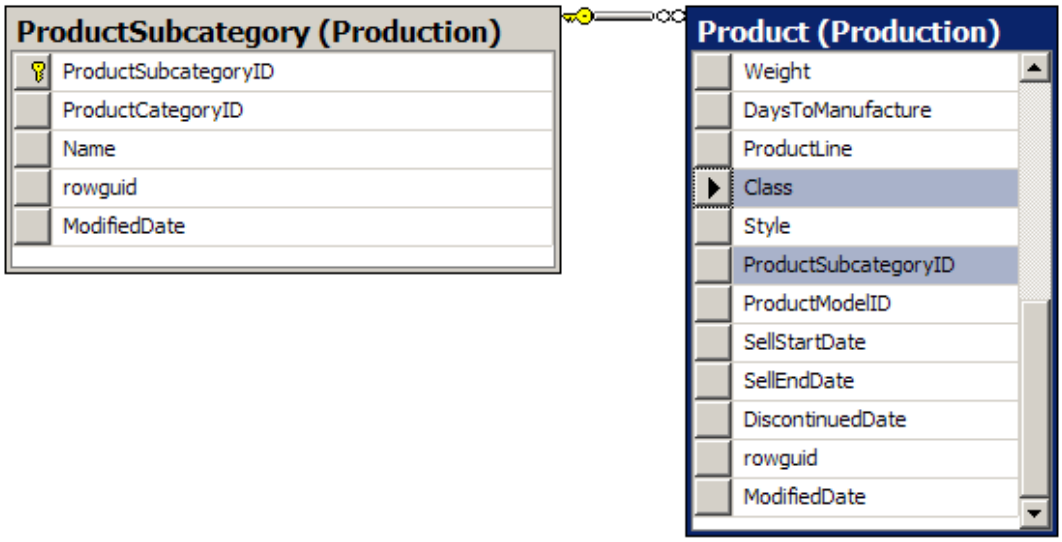
Aslında Asp.Net 2.0 mimarisinde yer alan URL eşleştirme sistemi aşağıdaki grafikte görüldüğü gibi çalışmaktadır.



Buna göre istemciden gelen talepler sonrası, ilgili web uygulaması Asp.Net çalışma zamanı içerisinde normal sürecine devam eder. Taki sayfanın son hali Render işlemine tabi tutulup istemciye gönderilene kadar. Bir başka deyişle Render işleminde önce, **Asp.Net çalışma zamanı(Asp.Net RunTime)** web.config içerisinde herhangibir eşleştirme olup olmadığına bakar. Eğer talep edilen URL için bir eşleştirme varsa buna göre **HttpContext** tipinin **RewritePath** metodu işletilir ve URL adresi değiştirilir. Sonrasında ise sayfa istemciye gönderilir.

Dilerseniz örnek bir senaryo üzerinden hareket ederek URL eşleştirmelerinin Asp.Net 2.0 mimarisinde nasıl yapıldığını yakından incelemeye çalışalım. Senaryo gereği kullanıcının alt kategorisi ve sınıfına göre bazı ürünleri listelediğini düşünebiliriz. Bu amaçla **SQL**

Server 2005 ile
gelen **AdventureWorks** veritabanındaki **ProductSubCategories** ve **Product**tablolarını göz önüne alalım. Bu tablolara arasında aşağıdaki şekilde görülen **bire-çok(one to many)** ilişki vardır.

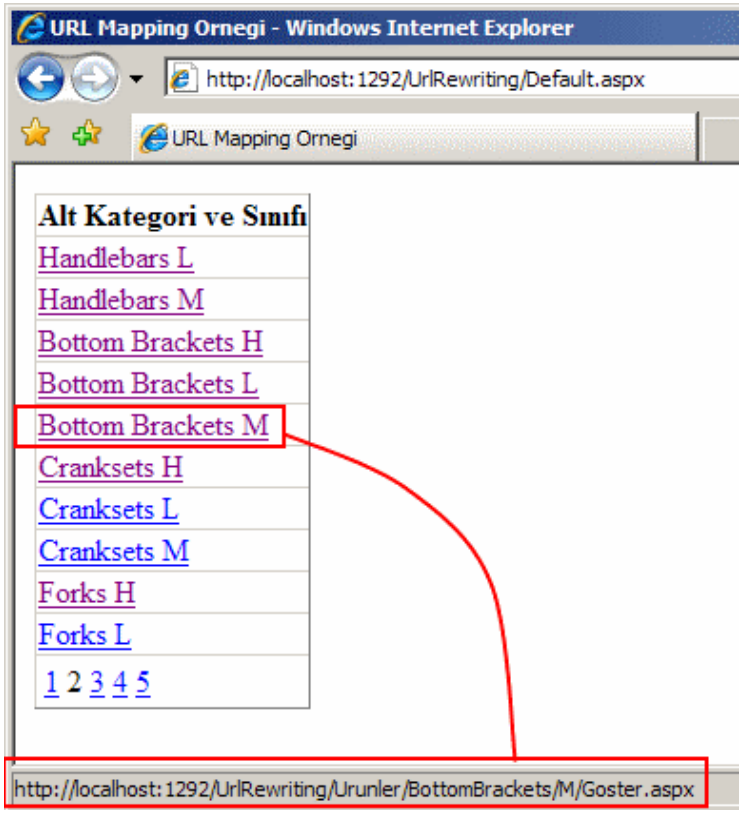


İlk olarak default.aspx sayfamızı aşağıdaki gibi geliştirelim.

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>URL Mapping Ornegi</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:GridView ID="GridView1" runat="server" DataSourceID="dsCategories"
AllowPaging="True" AutoGenerateColumns="False">
         <Columns>
            <asp:HyperLinkField DataNavigateUrlFields="Name,Class" DataNavigate
UrlFormatString="~/Urunler/{0}/{1}/Goster.aspx" DataTextField="Title"
HeaderText="Alt Kategori ve Sınıfı" />
         </Columns>
```

```
        </asp:GridView>
        <asp:SqlDataSource ID="dsCategories" runat="server" ConnectionString="<%$
ConnectionStrings:AdvConStr %>" SelectCommand="SELECT DISTINCT
PSC.ProductSubcategoryID, Replace(PSC.Name,' ','') AS Name, RTRIM(PRD.Class)
as Class, PSC.Name+' '+PRD.Class AS Title FROM Production.ProductSubcategory AS
PSC INNER JOIN Production.Product AS PRD ON PSC.ProductSubcategoryID =
PRD.ProductSubcategoryID WHERE (PRD.Class IS NOT NULL)">
        </asp:SqlDataSource>
    </div>
    </form>
</body>
</html>
```

Default.aspx sayfası içerisinde yer alan GridView kontrolü, Product ve ProductSubCategory tablolarının birleşiminden bir sonuç kümesine ait satırları göstermek üzere tasarlanmıştır. Söz konusu sonuç kümesi elde edilirken Name ve Class alanındaki boşlukların alınması için **Replace** ve **RTrim** isimli T-SQL fonksiyonlarına başvurulmaktadır. GridView bileşenine dikkat edilecek olursa içeride **HyperLinkField** tipinden bir kontrol kullanılmaktadır. Bu kontrolün dikkate değer özelliği ise **DataNavigateUrlFormatString** niteliğidir. Sayfa çalışma zamanında aşağıdakine benzer bir sonuç verecektir.



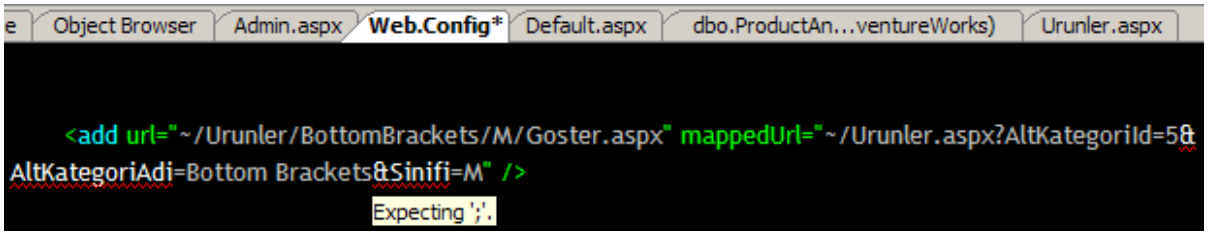
Dikkat edilecek olursa bağlantıların(Links) hedef URL bilgisi, **HyperLinkField** kontrolünün **DataNavigateUrlFormatString** niteliğinin değerine göre şekillenmektedir. Söz gelimi, örnek ekran görüntüsünde yer aldığı gibi M sınıfındaki Bottom Brackets ürünleri için URL bilgisi aşağıdaki gibidir.

http://localhost:1292/UrlRewriting/**Urunler/BottomBrackets/M/Goster.aspx**

Dikkat edilecek olursa bu URL bilgisinin Urunler kelimesinden itibaren olan kısmı çok daha okunaklı ve anlamlıdır. Peki biz bu linke tıkladığımızda Bottom Brackets kategorisinde ve M sınıfında yer alan ürünlerin listesi nasıl elde edilebilir. Bu işlemin Urunler.aspx gibi bir sayfa içerisinde ele alınması düşünüldüğü takdirde, seçilen bağlantı bilgisine göre ilgili kategori ve sınıf bilgilerinin sorgu katarı**(QueryString)** ile diğer sayfaya gönderilmesi gerekmektedir. Buda çok doğal olarak, default.aspx sayfasında seçilen URL bilgisine eş düşecek asıl URL bilgisinin tanımlanması ile mümkün olabilir. İşte bu noktada, Urunler.aspx sayfasını tasarlamadan önce, **web.config** içerisinde aşağıdaki ilaveler yapılmalıdır.

```
<?xml version="1.0"?>
<configuration>
   <appSettings/>
   <connectionStrings>
      <add name="AdvConStr" connectionString="Data Source=.;Initial
Catalog=AdventureWorks;Integrated Security=True"
providerName="System.Data.SqlClient"/>
   </connectionStrings>
   <system.web>
      <urlMappings enabled="true">
         <add url="~/Urunler/BottomBrackets/H/Goster.aspx" mappedUrl="~/Urunler.asp
x?AltKategoriId=5&amp;AltKategoriAdi=Bottom%20Brackets&amp;Sinifi=H"/>
         <add url="~/Urunler/BottomBrackets/L/Goster.aspx"
mappedUrl="~/Urunler.aspx?AltKategoriId=5&amp;AltKategoriAdi=Bottom%20Brackets
&amp;Sinifi=L"/>
         <add url="~/Urunler/BottomBrackets/M/Goster.aspx"
mappedUrl="~/Urunler.aspx?AltKategoriId=5&amp;AltKategoriAdi=Bottom%20Bracket
s&amp;Sinifi=M"/>
         <add url="~/Urunler/Cranksets/H/Goster.aspx"
mappedUrl="~/Urunler.aspx?AltKategoriId=5&amp;AltKategoriAdi=Cranksets&amp;Sini
fi=H"/>
      </urlMappings>
      <compilation debug="true"/>
      <authentication mode="Windows"/>
   </system.web>
</configuration>
```

URL eşleştirmeleri için system.web elementi içerisinde yer alan **urlMappings** boğumu kullanılmaktadır. Bu boğumda, **add** elementi içerisinde yer alan **url** ve **mappedUrl** nitelikleri ilede gereken eşleştirmeler yapılmaktadır. Buna göre, **~/Urunler/BottomBrackets/M/Goster.aspx** gibi bir talep geldiğinde bu, **~/Urunler.aspx?AltKategoriId=5&AltKategoriAdi=Bottom%20Brackets&Sinifi=M** olarak algılanacaktır. & operatörünün ele alınması sırasında & ifadesinin kullanılmasına dikkat etmek gerekir. Eğer & işareti kullanılırsa derleme zamanıda hata mesajı alınır.

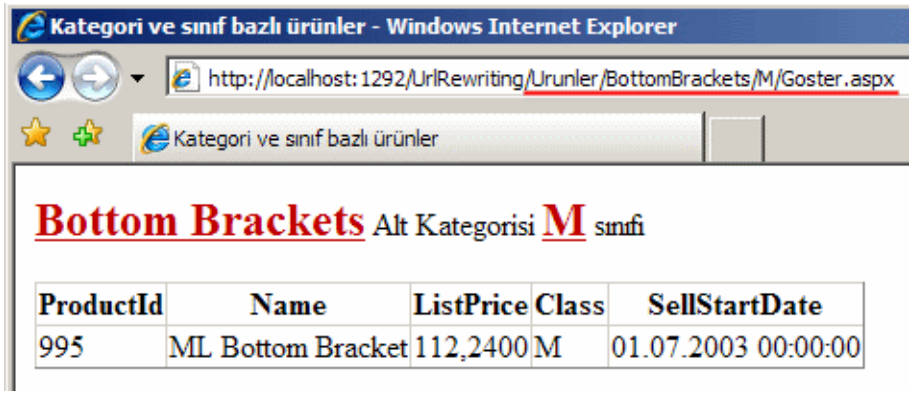


Artık gerekli bildirimler yapıldığına göre Urunler.aspx sayfası aşağıdaki gibi tasarlanabilir.

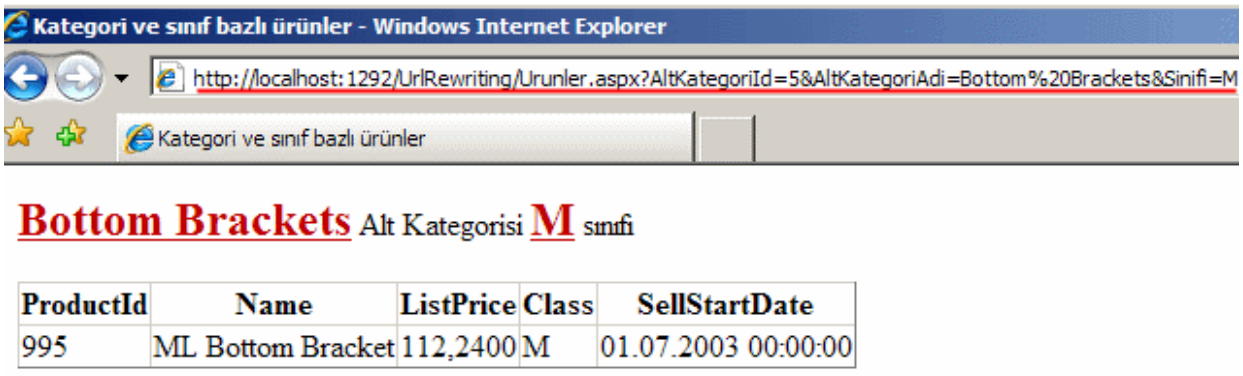
```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">
   protected void Page_Load(object sender, EventArgs e)
   {
      lblAltKategoriAdi.Text
= Request.QueryString["AltKategoriAdi"]!=null?Request.QueryString["AltKategori
Adi"].ToString():"";
      lblSinifi.Text
= Request.QueryString["Sinifi"]!=null?Request.QueryString["Sinifi"].ToString():"";
   }
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kategori ve sınıf bazlı ürünler</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <asp:Label ID="lblAltKategoriAdi" runat="server" Font-Bold="True" Font-Size="X-
Large" Font-Underline="True" ForeColor="#C00000"></asp:Label>
      Alt Kategorisi
      <asp:Label ID="lblSinifi" runat="server" Font-Bold="True" Font-Size="X-Large"
```

```
Font-Underline="True" ForeColor="#C00000"></asp:Label> sınıfı<br />
      <br />
      <asp:GridView ID="grdUrunler" runat="server" AutoGenerateColumns="False"
DataKeyNames="ProductId" DataSourceID="dsProducts">
         <Columns>
            <asp:BoundField DataField="ProductId" HeaderText="ProductId"
InsertVisible="False" ReadOnly="True" SortExpression="ProductId" />
               <asp:BoundField DataField="Name" HeaderText="Name"
SortExpression="Name" />
               <asp:BoundField DataField="ListPrice" HeaderText="ListPrice"
SortExpression="ListPrice" />
               <asp:BoundField DataField="Class" HeaderText="Class"
SortExpression="Class" />
               <asp:BoundField DataField="SellStartDate" HeaderText="SellStartDate"
SortExpression="SellStartDate" />
         </Columns>
      </asp:GridView>
      <asp:SqlDataSource ID="dsProducts" runat="server" ConnectionString="<%$
ConnectionStrings:AdvConStr %>" SelectCommand="Select
ProductId,Name,ListPrice,Class,SellStartDate From Production.Product Where
ProductSubCategoryId=@SubCatId and Class=@Class">
         <SelectParameters>
            <asp:QueryStringParameter DefaultValue="1" Name="SubCatId"
QueryStringField="AltKategoriId" />
            <asp:QueryStringParameter DefaultValue="M" Name="Class"
QueryStringField="Sinifi" />
         </SelectParameters>
      </asp:SqlDataSource>
   </div>
   </form>
</body>
</html>
```

Bu sayfa sadece **QueryString** ile gelen parametreler değerlendirmekte ve buna göre belirli bir alt kategori ve sınıfa ait ürünlerin bazı alanlarının listelenmesini sağlamaktadır. Bu amaçla yine**GridView** ve **SqlDataSource** kontrollerinden yararlanılmış ve uygun sorgu cümleleri kullanılmıştır. örnek senaryoda yer alan Bottom Brockets alt kategorisi ve M sınıfı seçilirse, Urunler.aspx sayfasında aşağıdaki ekran görüntüsünde yer alan çıktılar elde edilecektir.



Yukarıdaki çıktıya dikkat edilecek olursa URL satırı bizim belirlediğimiz şekilde kalmıştır. Bu çıktının aynısını elde etmek için halen daha **sorgu katarı(QueryString)**ifadeleride açıkça kullanılabilir. Aynı web uygulamasında aşağıdaki ekran görüntüsünde yer alan URL talebi, aynı sonuçları verecektir.



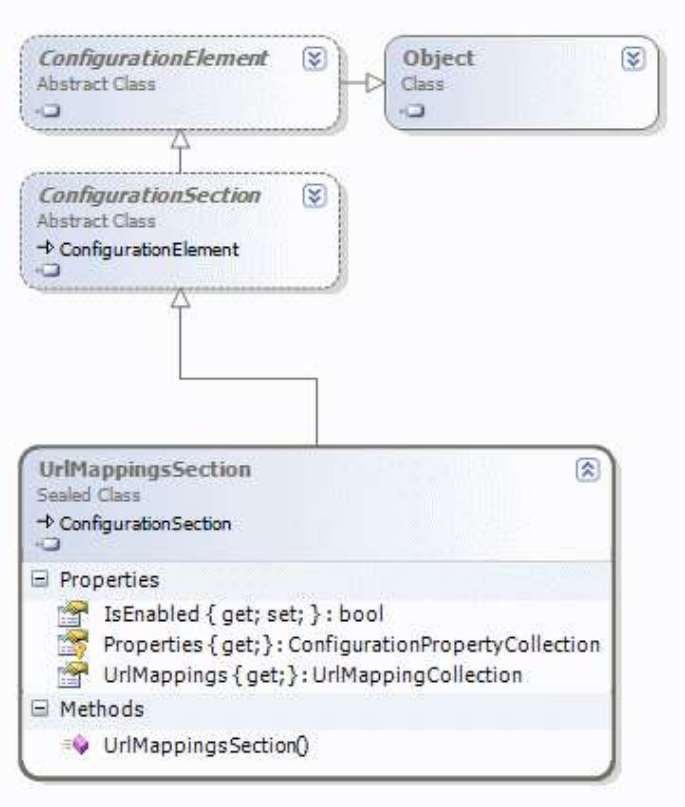
Ne varki halen daha bazı problemler vardır. Herşeyden önce en azından söz konusu senaryo göz önüne alındığında, var olabilecek tüm olasılıklar için web.config dosyasına teker teker URL eşleştirmelerinin eklenmesi gerekmektedir. Bir geliştirici olarak bunun kod içerisinden daha etkili bir şekilde ele alınması çok doğal bir istektir. Neyseki Asp.Net 2.0 ile gelen güçlü Configuration API alt yapısı sayesinde istersek, kod içerisinde **urlMappings** kısmına dinamik olarak yeni **elemanlar(elements)** ekleyebilir, düzenleyebilir ve çıkartabiliriz. Bu amaçla bir admin sayfası veya farklı bir uygulama olacak şekilde bir admin paneli dahi göz önüne alınabilir.(***Configuration API'*** *sinin yönetiminin daha detaylı bir şekilde öğrenmek isterseniz daha önce yazılmış olan bir* ***makaleden*** *yararlanabilirsiniz.*) Bunun için aşağıdaki gibi bir sayfa tasarladığımızı göz önüne alabiliriz.

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>
<%@ Import Namespace="System.Web.Configuration" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

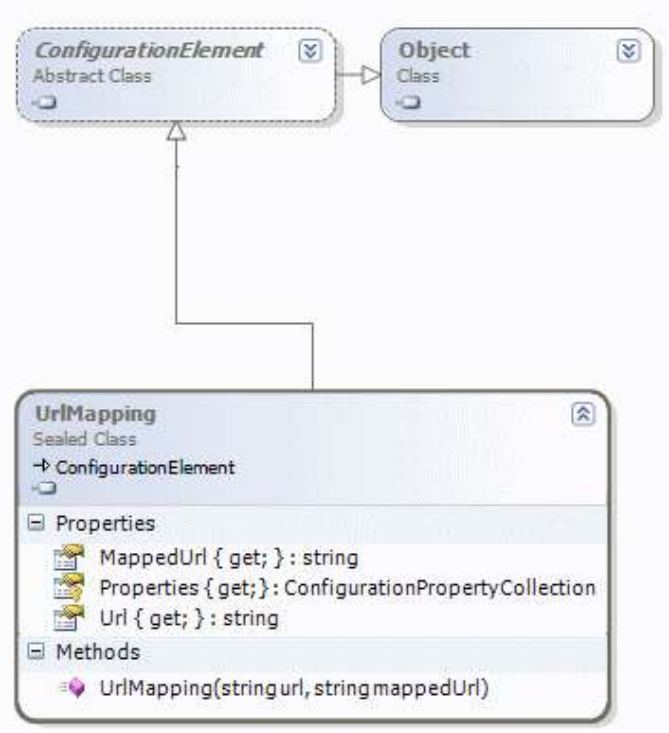
```
<script runat="server">

   private static void AddMappings()
   {
      Configuration cfg = ConfigurationManager.OpenExeConfiguration("");
      UrlMappingsSection urlMapSct =
(UrlMappingsSection)cfg.GetSection("system.web/urlMappings");
      urlMapSct.UrlMappings.Clear();

      string query = "SELECT DISTINCT PSC.ProductSubcategoryID,
PSC.Name,PRD.Class FROM Production.ProductSubcategory AS PSC INNER JOIN
Production.Product AS PRD ON PSC.ProductSubcategoryID =
PRD.ProductSubcategoryID WHERE (PRD.Class IS NOT NULL) ORDER BY
PSC.Name, PRD.Class";

      using (SqlConnection conn = new
SqlConnection(cfg.ConnectionStrings.ConnectionStrings["AdvConStr"].ConnectionString)
)
      {
         SqlCommand cmd = new SqlCommand(query, conn);
         conn.Open();
         string url = "", mappedUrl = "";

         SqlDataReader reader = cmd.ExecuteReader();
         while (reader.Read())
         {
            url = "~/Urunler/" + reader.GetString(1).Replace(" ","") + "/" +
reader.GetString(2).Replace(" ","") + "/Goster.aspx";
            mappedUrl = "~/Urunler.aspx?AltKategoriId=" +
reader["ProductSubCategoryID"].ToString() + "&AltKategoriAdi=" +
reader["Name"].ToString() + "&Sinifi=" + reader["Class"].ToString();
            UrlMapping map = new UrlMapping(url, mappedUrl);
            urlMapSct.UrlMappings.Add(map);
         }
         reader.Close();
      }
      cfg.Save();
   }

   protected void btnMappEkle_Click(object sender, EventArgs e)
   {
      AddMappings();
   }
```

```
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>Yonetici Sayfasi</title>
   </head>
   <body>
      <form id="form1" runat="server">
         <div>
            <asp:Button ID="btnMappEkle" runat="server" Text="Güncel URL
Eşleştirmelerini Ekle" OnClick="btnMappEkle_Click" />
         </div>
      </form>
   </body>
</html>
```

Şimdi burada geliştirdiğimiz kodları kısaca inceleyelim. Configuration alt yapısına göre, web.config dosyası içerisinde bilinen hemen her **boğumun(Node)** birer **yönetimli tip(Managed Type)**karşılığı vardır. Buradaki tipimiz **UrlMappingsSection** sınıfıdır. UrlMappingsSection sınıfıda Asp.Net 2.0 ile birlikte gelmiş bir sınıftır. Sınıf diagramından ele alındığında UrlMappingsSection sınıfının Framework içerisindeki yeri aşağıdaki şekilde görüldüğü gibidir.



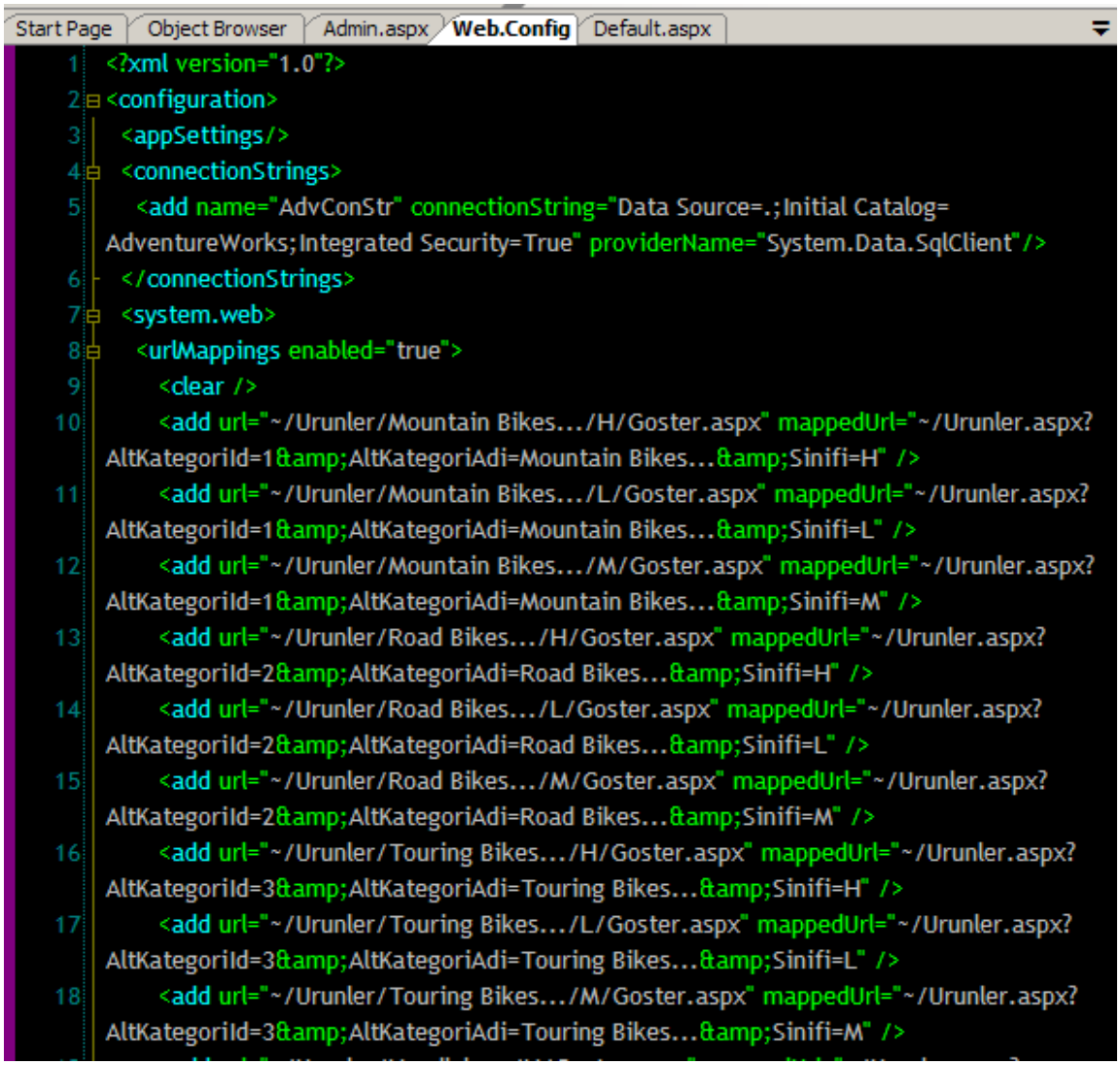
Görüldüğü gibi UrlMappingsSection sınıfının **UrlMappings** özelliği aslında UrlMapping tipinden elemanlar taşıyan özel bir koleksiyonu**(UrlMappingCollection)** işaret etmektedir. Yeni bir **UrlMapping** nesne örneği eklenmek istendiğinde bu koleksiyondan yararlanılır. Bu koleksiyondaki elemanları oluşturan UrlMapping tipinin Framework içerisindeki yeri ise aşağıdaki gibidir.



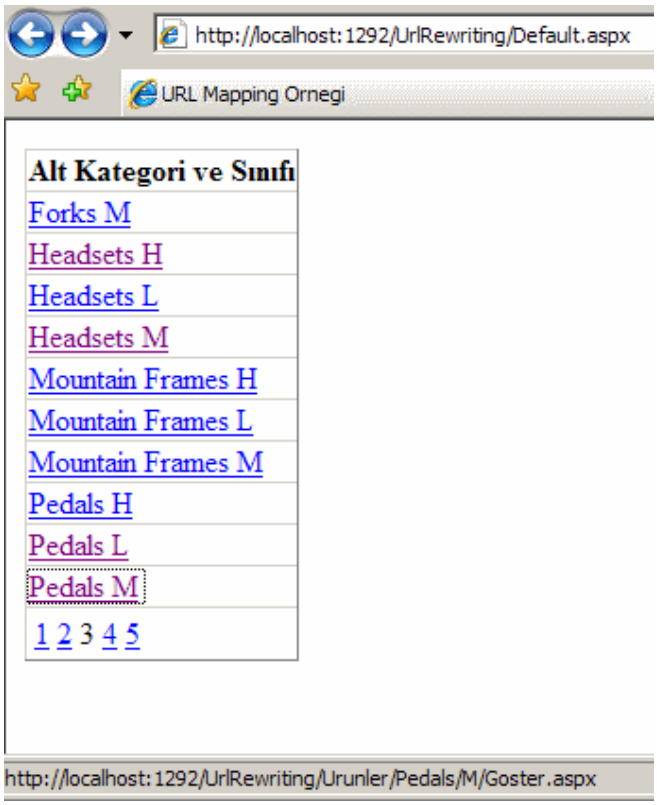
Bu tipi o anki web.config dosyasından elde etmek amacıyla Configuration nesne örneğinin **GetSection** metodu kullanılmaktadır. Bu metod bilindiği gibi **XPath** ifadelerini parametre olarak alır.*(XPath ile ilgili detaylı bilgiyi daha önceki bir **makalemizden** edinebilirsiniz)* Sonrasında ise veritabanından yapılan sorgu sonucu elde edilen veri kümesine göre **url** ve **mappedUrl** değerleri oluşturulur. Bu değişkenler, her bir satır için **web.config** dosyasına eklenecek **UrlMapping** tiplerinin **yapıcı metodlarına(Constructor)** parametre olarak verilmektedir. Son olarak oluşan UrlMapping nesne örneği, **Add** metodu ile web.config/urlMappings elementi içerisindeki yerini almaktadır.

Elbette, bellek üzerinde yapılan bu değişikliklerin kalıcı olması adına **Configuration** tipinin **Save** metodu kullanılmaktadır. Dikkat edilmesi gereken noktalardan biriside URL değeri oluşturulurken **Replace** metodu ile Name ve Class alanlarındaki boşlukların alınmasıdır. Eğer söz konusu boşluklar alınmassa (özellike Class alanlarındakiler alınmassa) bu linklere tıklandığında çalışma zamanında sayfaların bulunamadığına dair hata mesajları alınabilir. Bu aynı zamanda, URL içerisinde geçersiz olabilecek karakterler var ise bunlarında çıkartılması veya değiştirilmesi gerektiği anlamınada gelmektedir. örneğin boşlukar çoğunlukla URL satırına %20 şeklinde aktarılırlar. Ancak örnekteki URL bilgisinde klasör tabanlı bir yaklaşım tercih edildiğinden

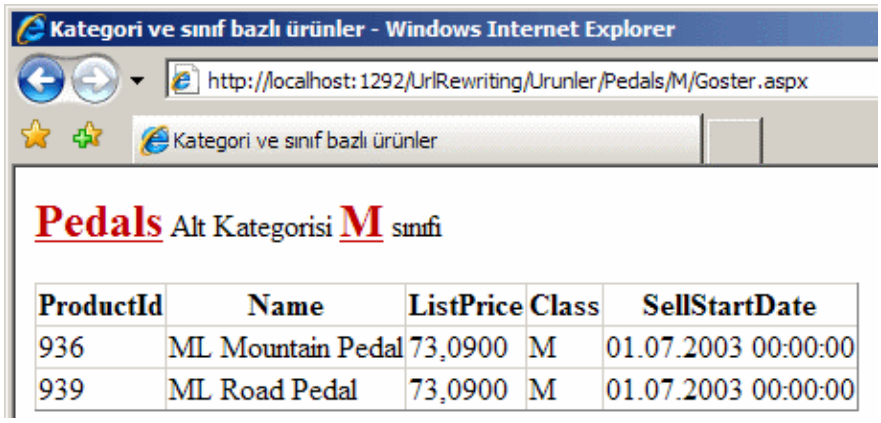
bütün boşlukların çıkartılması yolu tercih edilmiştir. Bu işlemlerin arkasından Admin sayfası çalıştırılır ve düğme tıklanırsa web.config dosyasının aşağıdaki ekran görüntüsünde olduğu gibi değiştiği görülür.



Dikkat edilecek olursa, elde edilen veri kümesine göre, söz konusu olabilecek tüm URL eşleştirmeleri ilave edilmiştir. Configuration API'si sağolsun :) Şimdi default.aspx sayfası çağırılırsa, artık GridView kontrolü içerisindeki her bir bağlantının karşılığının olduğu ve çalıştığı görülür. Default.aspx için örnek görüntü aşağıdaki gibidir.



İlgili bağlantının tıklanması sonrası ise aşağıdaki sonuçlara benzer çıktılar elde edilebilir.



Buraya kadar örnek bir senaryo üzerinden URL eşleştirmesini incelemeye çalıştık. Son teknikte dinamik olarak URL eşleştirmelerini ekledik. Bu teknik her ne kadar göze hoş gelsede dezavantajıda vardır. öyleki, veri kaynağında değişiklikler yapıldığında örneğin Alt kategori adı değiştiğinde yada yenileri eklendiğinde web.config dosyası içerisinde yeni düzenlemelerin yapılması, bir başka deyişle ilgili fonksiyonun tekrardan çağırılması gerekcektir. Bu çeşitli teknikler ile çözülebilir ama yinede düşünülmesi gereken bir adımdır.

Sonuç olarak Asp.Net 2.0 ile birlikte gelen **urlMappings** elementi her ne kadar bazı avantajlar sağlasada çeşitli kısıtlamalarda içermektedir. Regular Expression ifadeleri

kullanılamadığından, her bir URL için gereken eşleştirmeler teker teker web.config dosyası içerisinde yazılmalıdır. Gerçi **Configuration API'** si yardımıyla bu bir nebzede olsa aşılabilmektedir ancak **Regular Expression'**ın yerini tam anlamıyla tutmamaktadır. Regular Expression desteği için geliştiricinin özel **HttpModule** ve **HttpHandler** tiplerini geliştirmesi gerekliliği ise bir zorluktur. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

örnek Uygulama için Tıklayın

Asp.Net Temelleri : Etkili Trace Kullanımı (2007-08-02T18:16:00)

asp.net temelleri,

Web uygulamalarında son kullanıcıların**(End Users)** şikayetçi olabileceği pek çok konu vardır. Bunlar arasında popüler olanlarından biriside sayfaların yavaş açılıyor olmasıdır. Nihayetinde son kullanıcıları her zaman için sabırsız ve acelesi olan kişiler olarak düşünmek doğru bir yaklaşım olacaktır. Sayfaların yavaş açılıyor yada geç cevap veriyor olmasının donanımsal yada çeşitli çevre faktörleri nedeniyle bilinen sebepleri vardır. Söz gelimi bağlantı hızının düşük olması ilk akla gelen nedendir.
Ancak**geliştiriciler(developers)** olarak bizlerinde üzerine düşen önemli görevler vardır. Sonuç itibariyle bir geliştirici, mininum donanım gereksinimleri karşılandığı takdirde hızlı ve yeterli performasta çalışabilen bir web uygulaması geliştiriyorsa, donanım kapasite ve yeteneklerinin daha yüksek olduğu son kullanıcılarda aynı web uygulamasının çok daha iyi sonuçlar vereceği düşünülebilir.

***NOT : Trace** mekanizmasını kullanmak için küçük bir neden; örneğin talep**(request)** edilen sayfanın üretilmesi ve **HTML** çıktısının hazırlanması uzun zaman alıyor olabilir. Sayfanın üretimi sırasında hangi adımların çok zaman aldığını görmek için Trace mekanizmasından yararlanılabilir.*

Elbetteki bir web uygulamasının hızını, performansını ve cevap verebilme yeteneklerini etkiliyecek pek çok faktör vardır. Yine örnek olarak düşünülürse, 10 kullanıcılı bir intranet ortamında gayet iyi çalışan bir web uygulaması, internet' e açıldığında karşılaşabileceği n kullanıcı talebi sonrasında beklenen performansı gösteremeyebilir. Buradaki en büyük etken kullanıcı sayısıdır. Bu nedenle web uygulamalarını ne kadar iyi programladığımızı düşünsekte, sonuçlara bakıldığında gerek test gerekse gerçek ortamlarda istediğimiz sonuçları alamadığımızı görebiliriz. Dolayısıyla test aşamasında uygulamanın genelini izleyebilmek ve sonuçları analiz ederek gerekli tedbirleri alabilmek son derece önemli bir konudur. Bu gibi durumlarda web uygulamasının genelinin yada problemli olabileceği düşünülen sayfaların çalışma zamanı ve sonrasındaki hareketlerinin izlenmesi adına Asp.Net modelinde **Trace** mekanizasından faydalanılmaktadır. İşte bu makalemizde Trace kavramını derinlemesine incelemeye çalışacağız.

Trace mekanizması ile **sayfa(Page)** veya **uygulama seviyesinde(Application Level)**, kullanıcıdan gelen bir talebin işlenmesi sırasında ve sonuçların elde edilmesinin sonrasında gerekli host bilgileri yakalanabilir. Bu bilgiler değerlendirilerek sorunun nerede olduğu daha kolay bir şekilde tespit edilebilir.

NOT : *Trace;* ***HTTP*** *talebi ile ilgili olaraktan* ***sunucu(server)*** *taraflı* ***çalışma zamanı(run-time)*** *ve sonrası detay bilgilerinin alınmasını sağlayan bir Asp.Net mekanizmasıdır.*

Trace mekanizması sadece sayfa ve uygulama seviyesinde değil, **bileşen seviyesindede(Component Level)** ele alınabilir. Nitekim, web uygulamalarında sayfaların çoğu buton arkası programlama yerine bileşenleri ele alır. çok doğal olarak bileşenler içerisinde meydana gelebilecek sorunların ele alınması sırasında yine Trace mekanizması ele alınabilir. Trace ile elde edilen sonuçlar üretilen genellikle sayfa sonlarına **HTML** bilgisi olarak eklenebilirler. Bunun dışında **Trace.axd** isimli özel dosyalardanda uygulama genelinde, talep sırasına göre sayfaların izlenmesi sağlanabilir.

İlk olarak Trace işlemlerini sayfa seviyesinde ele almaya çalışacağız. Bir sayfanın Trace çıktısını yakalamak için iki farklı yöntem vardır. Bunlardan birincisi **Page** direktifinde yer alan**Trace** niteliğine **true** değeri atanmasıdır. Diğer yol ise kod tarafında **Trace** nesne örneği üzerinden **IsEnabled** özelliğine true değerini atamaktır.

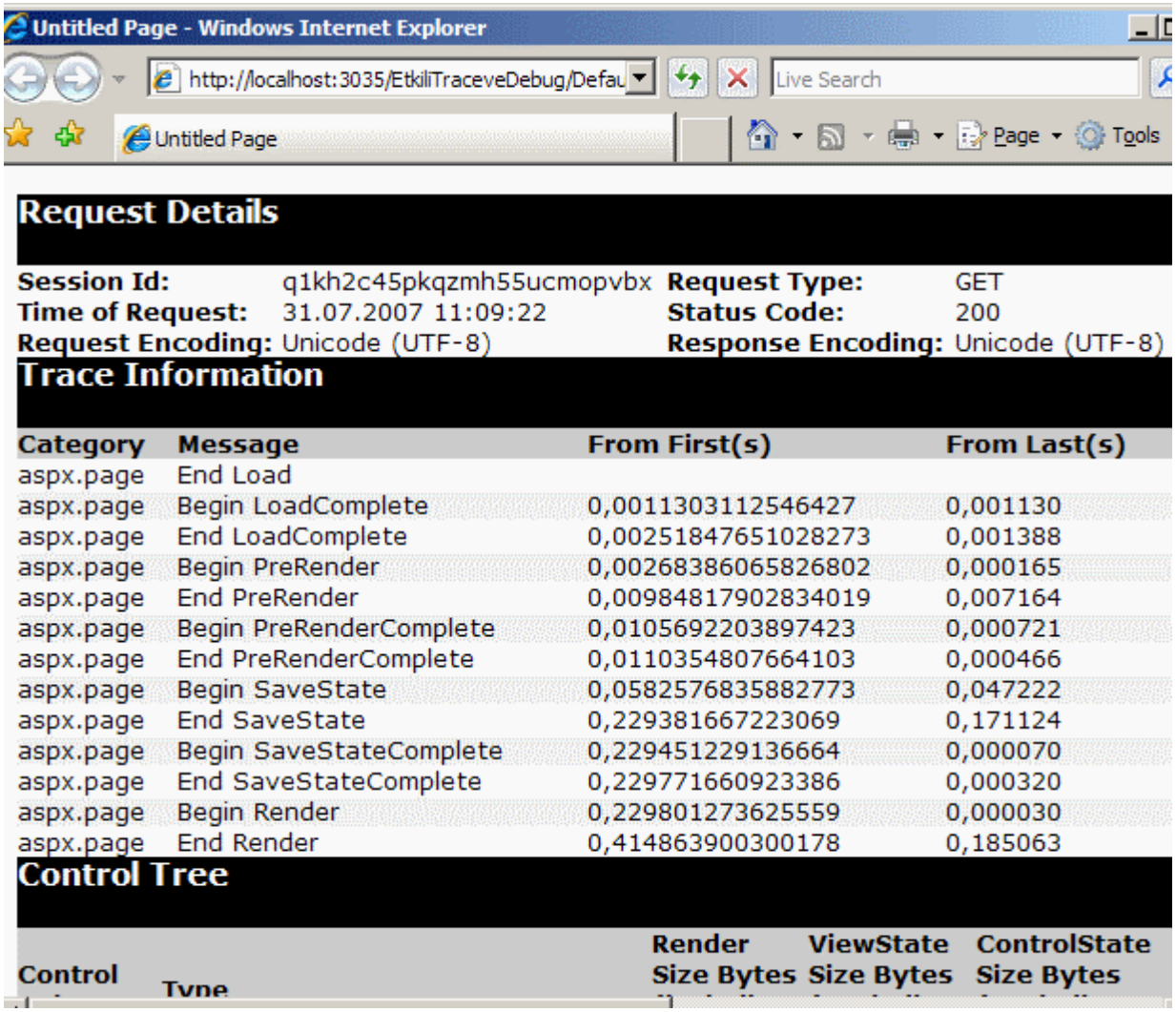
Direktik içerisinden Trace mekanizmasının açılması;

```
<%@ Page Language="C#" Trace="true"%>
```

Kod tarafından Trace mekanizmasının açılması;

```
protected void Page_Load(object sender, EventArgs ea)
{
   Page.Trace.IsEnabled = true;
}
```

Sonuç olarak bu iki kullanım şekline göre sayfanın sonuna Trace bilgileri aşağıdaki ekran görüntüsünde olduğu gibi eklenecektir.



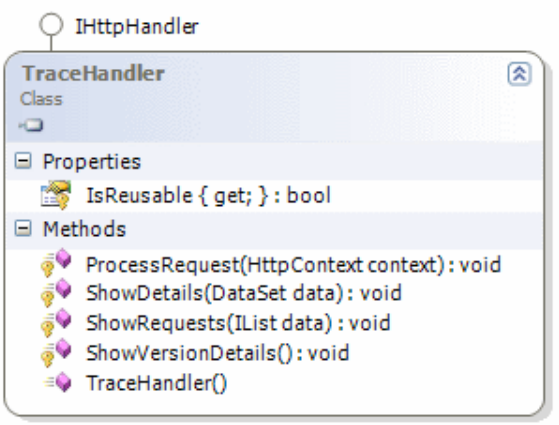
Bu çıktıda tüm Trace bilgisi gösterilmemektedir. Temel olarak Trace çıktısında yer alan kısımlar aşağıdaki tabloda olduğu gibidir.

Trace çıktısında Yer Alan Bölümler	
Request Details	Bu kısımda talep ile ilişkili genel bilgiler yer alır. örneğin **talep tipi(requ** Bir başka örnek detay bilgisi olarakta **Session Identity(SessionId)** değer
Trace Information	Bu kısımda özellikle sayfanın yaşam döngüsü**(Page Life Cycle)** içerisinc sayfanın üretimi sırasında hangi kısımların ne kadar süre harcadığı görüle
Control Tree	Sayfa içerisindeki kontrollerin ağaç yapısını verir. Böylece özellikle form içerisinde yer alan alt kontrollerin elde edilmesi için ne kadar derine inile
Session State	Kullanıcıya ait **oturum(Session)** içerisinde tutulan verilerin değerlerinin
Application State	**Application** nesnesi içerisindeki anahtarlar**(keys)** ve değerlerinin**(values**
Request Cookies	Sayfaya gelen talep sırasındaki çerezlere ait bilgilerin tutulduğu koleksiy

Collection	
Response Cookies Collection	Sayfadan istemciye dönen cevap içerisinde yer alan çerezlere**(Cookies)** a
Headers Collection	İstemciden sunucuya gelen **HTTP** paketindeki başlık bilgileri görülebilir
Response Headers Collection	İstemciye dönen HTTP paketindeki başlık bilgilerini içerir. örneğin üreti
Form Collection	Sunucudan istemciye gönderilen form bilgileri görülebilir. örneğin üretil
Querystring Collection	Sayfadan istenen **query string** bilgileri var ise bunların adları ve o anki d
Server Variables	Sunucu değişkenleri elde edilebilir. örneğin, sunucu uygulamanın fiziki y

Aynı bilgilere ulaşmak için adres satırında **Trace.axd** dosyasıda talep edilebilir.

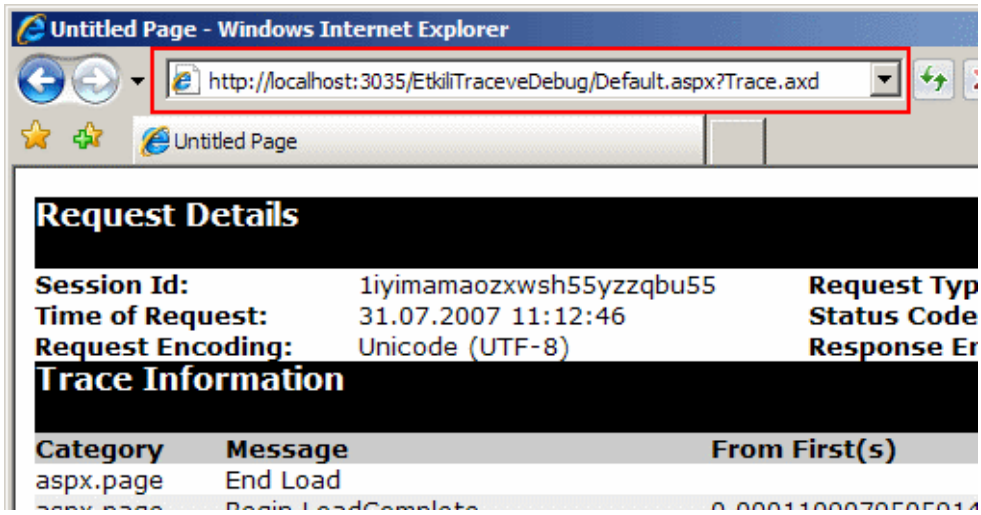
***NOT : Trace.axd**, WebResource.axd benzeri bir dosyadır. Dolayısıyla çalışma zamanında özel şekilde ele alınır. Asp.Net çalışma ortamı Trace.axd taleplerinin **TraceHandler**isimli sınıfa ait nesne örneklerine devredilerek karşılanmasını sağlar.*



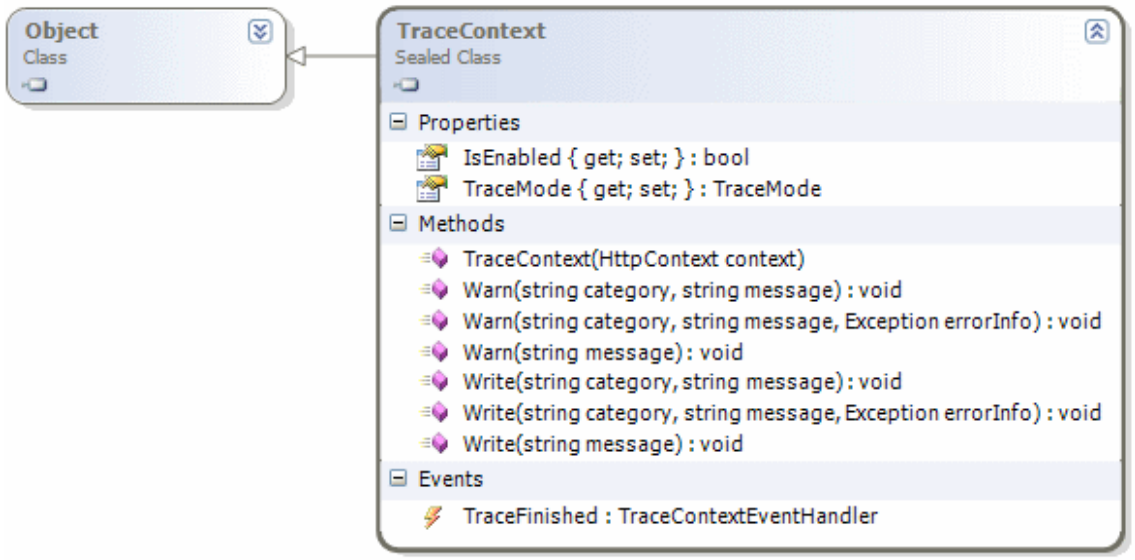
*Dolayısıyla söz konusu talep sonrası oluşan ekran çıktısı **TraceHandler** sınıfı tarafından hazırlanır. Makinedeki ana web.config dosyasının içeriğine bakıldığında bu açıkça görülebilir. Burada dikkat edilmesi gereken noktalardan birisi sadece Trace.axd için böyle bir handler' ın yazılmış olmasıdır. Trace.axd ve **WebResource.axd** dışında gelecek talepler **HttpNotFoundHandler** tarafından ele alınmaktadır.*

```
<httpHandlers>
 <add path="trace.axd" verb="*" type="System.Web.Handlers.TraceHandler"
  validate="true" />
 <add path="WebResource.axd" verb="GET" type="System.Web.Handlers.Asse
  validate="true" />
 <add path="*.axd" verb="*" type="System.Web.HttpNotFoundHandler"
```

Aşağıdaki ekran görüntüsünde **Trace.axd**' nin talep edilmesinin örnek sonuçları gösterilmektedir.

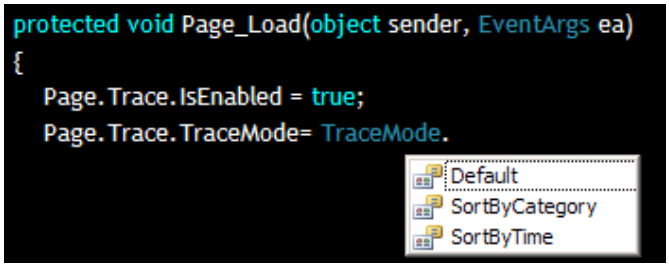


Kod tarafında Trace bilgilerini ele almak için **Page** nesne örneği üzerinden **Trace** özelliği kullanılmaktadır. Aslında Trace özelliği doğrudan **TraceContext** sınıfına ait bir nesne örneği döndürmektedir. TraceContext sınıfı **sealed** olarak işaretlenmiş bir tip olduğundan türetilerek**(inherit)** özelleştirilemez. Bu sınıfın Framework içerisindeki yeri şekilsel olarak aşağıdaki gibidir.



TraceContext sınıfının üyeleri göz önüne alındığında, **Write** ve **Warn** metodları kod içerisinden Trace çıktısına bilgi yazdırmak amacıyla kullanılmaktadır. Bunların arasındaki tek fark Warn metodunun yazıyı **kırmızı punto** ile basıyor olmasıdır. Diğer taraftan her iki metodda 3 farklı aşırı yüklenmiş versiyona sahiptir. Bu versiyonlar yardımıyla yazdırılan bilginin **kategorisi(Category)**, içeriği**(Message)** ve o anda yakalanmış bir **istisna(Exception)** var ise bu istisna nesne örneği Trace çıktısına

gönderilebilir. **IsEnabled**özelliği ile daha öncedende belirtildiği gibi Trace' in etklinleştirilmesi veya pasif moda çekilmesi sağlanabilir. TraceMode özelliği **TraceMode** enum sabiti tipinden değerler alabilmektedir.

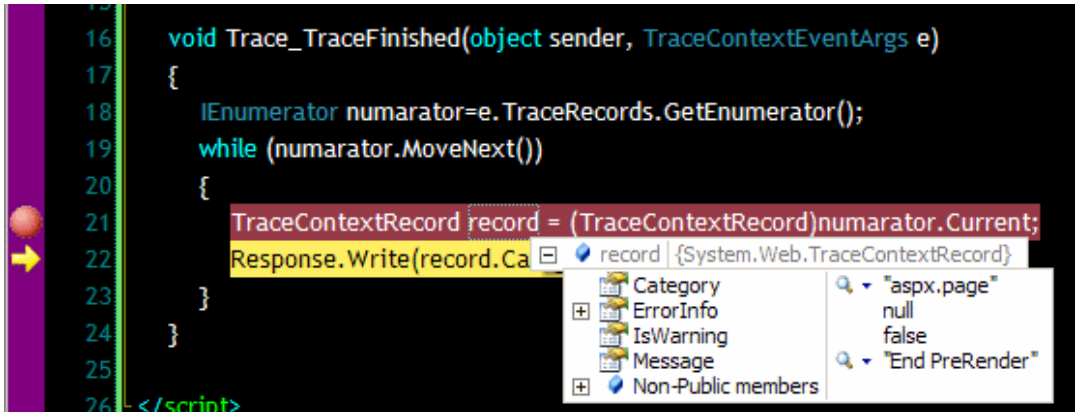


Bu değerler sayesinde Trace çıktısında yer alan **Trace Information** kısmındaki bilgilerin kategori veya süre bazlı olarak sıralanıp sıralanmayacağı belirlenebilir. Bu üyeler dışında TraceContext sınıfı özellikle bileşenlerin içerisinde ele alınmak istendiğinde yapıcı metoddan bir örnek oluşturulması gerekmektedir. Bunun için yapıcı metod **HttpContext**tipinden bir parametre alır. Bunu ilerleyen kısımlarda ele alacağız. TraceContext sınıfının birde **TraceFinished** isimli olayı vardır. **TraceContextEventHandler** temsilcisinin tanımladığı yapıya uygun olay metodu çağırılabilir. Bu olay TraceContext sınıfına .Net 2.0 ile birlikte katılmıştır. TraceFinished olayı, Trace bilgileri toplandıktan sonra TraceContext sınıfının kendisi tarafından tetiklenir. **TraceContextEventArgs** tipinden olan ikinci parametre sayesinde Trace ile ilgili veriler elde edilebilir ve bu sayede farklı veri kaynaklarına yazılmaları sağlanabilir. Aşağıdaki kod parçasında bu olayın kullanımı örneklenmeye çalışılmaktadır.

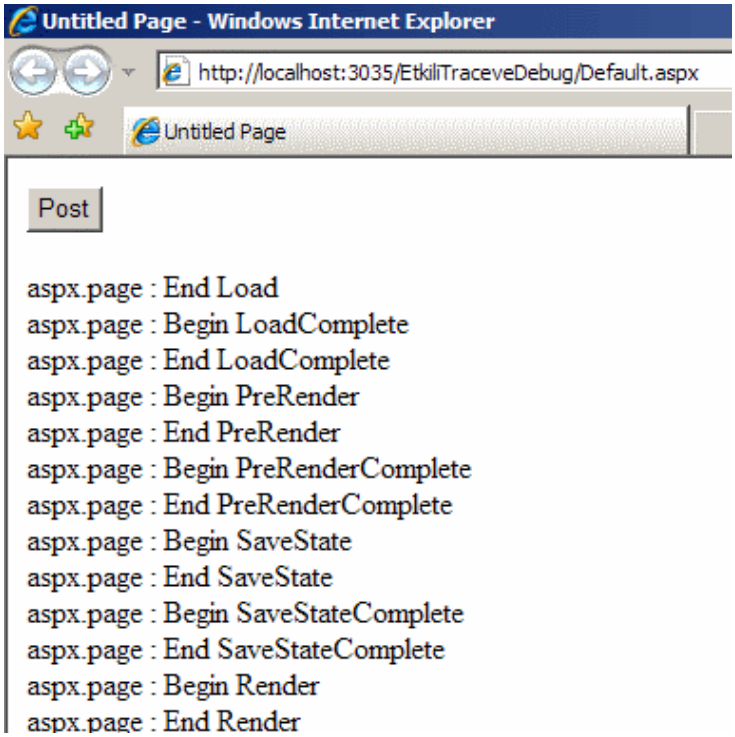
```
protected void Page_Load(object sender, EventArgs ea)
{
   Page.Trace.IsEnabled = true;
   Page.Trace.TraceFinished += new
TraceContextEventHandler(Trace_TraceFinished);
}

void Trace_TraceFinished(object sender, TraceContextEventArgs e)
{
   IEnumerator numarator=e.TraceRecords.GetEnumerator();
   while (numarator.MoveNext())
   {
      TraceContextRecord record = (TraceContextRecord)numarator.Current;
      Response.Write(record.Category + " : " + record.Message + "<br/>");
   }
}
```

TraceContextEventArgs tipinden e değişkeni üzerinden elde edilen koleksiyon içerisindeki her bir eleman **TraceContextRecord** sınıfına ait birer nesne örneğidir. Bu tip yardımıyla Trace Information kısmındaki kategori, mesaj, istisna tipi ve uyarı olup

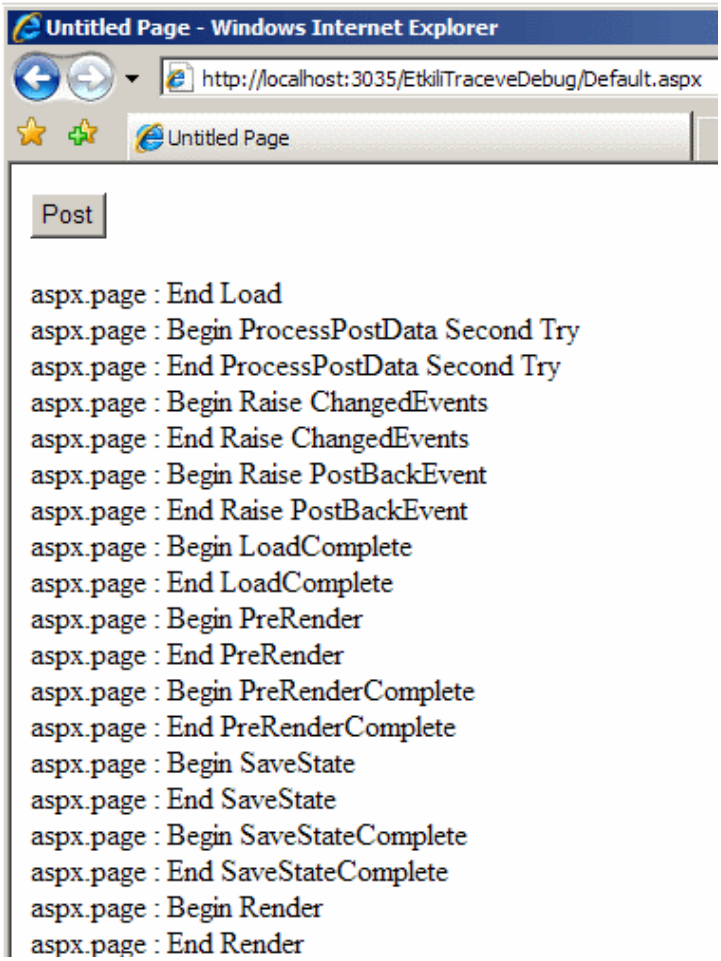
olmadığı**(IsWarning)** bilgilerine erişilebilir. Aşağıdaki ekran görüntüsünde bu bilgiler yer almaktadır.



Bu kodun yer aldığı sayfa ilk talep edildiğinde aşağıdaki ekran çıktısında yer alan içerik elde edilir.



Post işleminden sonra ise **TraceFinished** içerisinden yakalanan mesajların sayısı sayfanın yaşam döngüsü nedeni ile artacak ve aşağıdakine benzer bir ekran çıktısı elde edilecektir.



Söz konusu olay metodu ile her ne kadar **Trace Information** ile ilgili çok az bilgiye ulaşsakta zaten geri kalan verilerin çoğu **HttpResponse** veya **HttpRequest** gibi tipler yardımıyla elde edilebilmektedir. Dolayısıyla bu olay içerisinde bu tipler aracılığıyla elde edilen veriler başka veri ortamlarına da yazdırılabilirler.

Şimdi Trace mekanizmasını farklı yollar ile incelemeye devam edeceğiz. İlk olarak sayfa seviyesinde Trace bilgilerini izlemenin nasıl faydası olabiliceğini görmeye çalışacağız. Bu amaçla aşağıdaki web sayfası ve kodları göz önüne alınabilir. *(Bu ve sonraki asp.net sayfalarında, işlemlerin kolay takip edilmesi amacıyla inline-coding tekniği kullanılmıştır.)*

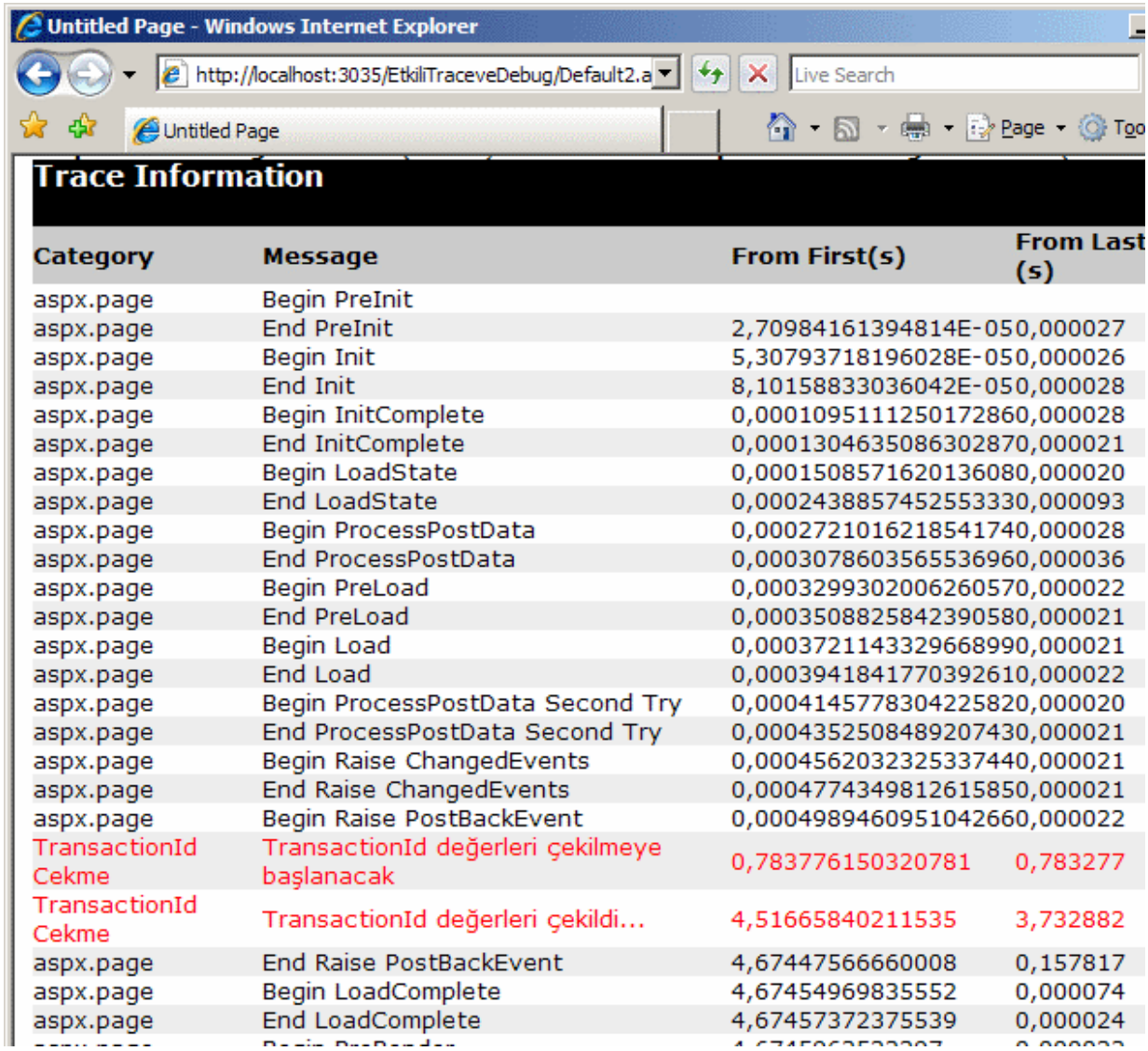
```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

protected void btnCek_Click(object sender, EventArgs e)
```

```
{
   string urunAdlari="";
   using (SqlConnection conn = new SqlConnection("data
source=.;database=AdventureWorks;integrated security=SSPI"))
   {
      SqlCommand cmd = new SqlCommand("Select Top 10000 TransactionId From
Production.TransactionHistory", conn);
      conn.Open();
      SqlDataReader reader = cmd.ExecuteReader();
      while (reader.Read())
      {
         urunAdlari += reader["TransactionId"].ToString()+"|";
      }
      reader.Close();
   }
   Session.Add("TumKategoriler", urunAdlari);
}

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>Untitled Page</title>
   </head>
   <body>
      <form id="form1" runat="server">
         <div>
            <asp:Button ID="btnCek" runat="server" Text="Veriyi çek"
OnClick="btnCek_Click" />
         </div>
      </form>
   </body>
</html>
```

Web sayfamızda yer alan düğmeye basıldığında, **TransactionHistory** tablosundaki ilk 10000 TransactionId değeri elde edilir ve bunlar bir string içerisinde birleştirilerek Session' a atılır. Kod her ne kadar anlamlı gözükmesede (ki gözükmediği ortada :)) ortaya çıkardığı sonuçlar nedeniyle kayda değerdir. Nitekim sayfa talep edildikten sonra düğmeye basıldığında sayfanın uzun bir sürede çıktı verdiği görülecektir. Burada herhangibir hata görünmemektedir. Ancak sayfanın çıktısının üretilmesi ve istemciye gelmesinin neden uzun sürdüğüde incelenmelidir. İşte bu amaçla bu sayfa üzerinde **Trace** işlemi gerçeklenmelidir. Hatırlanacağı üzere bunu kod veya direktif içerisinde yapabileceğimizi belirtmiştik. Bu nedenle **Page** direktifinde yer alan **Trace** niteliğine **true** değerini verelim ve kodlarımızı aşağıdaki gibi değiştirelim.

```
using (SqlConnection conn = new SqlConnection("data
source=.;database=AdventureWorks;integrated security=SSPI"))
{
   SqlCommand cmd = new SqlCommand("Select Top 10000 TransactionId From
Production.TransactionHistory", conn);
   conn.Open();
   Trace.Warn("TransactionId Cekme", "TransactionId değerleri çekilmeye
başlanacak");
   SqlDataReader reader = cmd.ExecuteReader();
   while (reader.Read())
   {
      urunAdlari += reader["TransactionId"].ToString()+"|";
   }
   Trace.Warn("TransactionId Cekme", "TransactionId değerleri çekildi...");
   reader.Close();
}
```

Burada **Warn** metodu kullanılarak urunAdlari toplanmadan önce ve toplandıktan sonra Trace çıktısına kategori bazlı değerler yazdırılmaktadır. Bunun sonucunda sayfa çıktısı aşağıdaki gibi olacaktır.



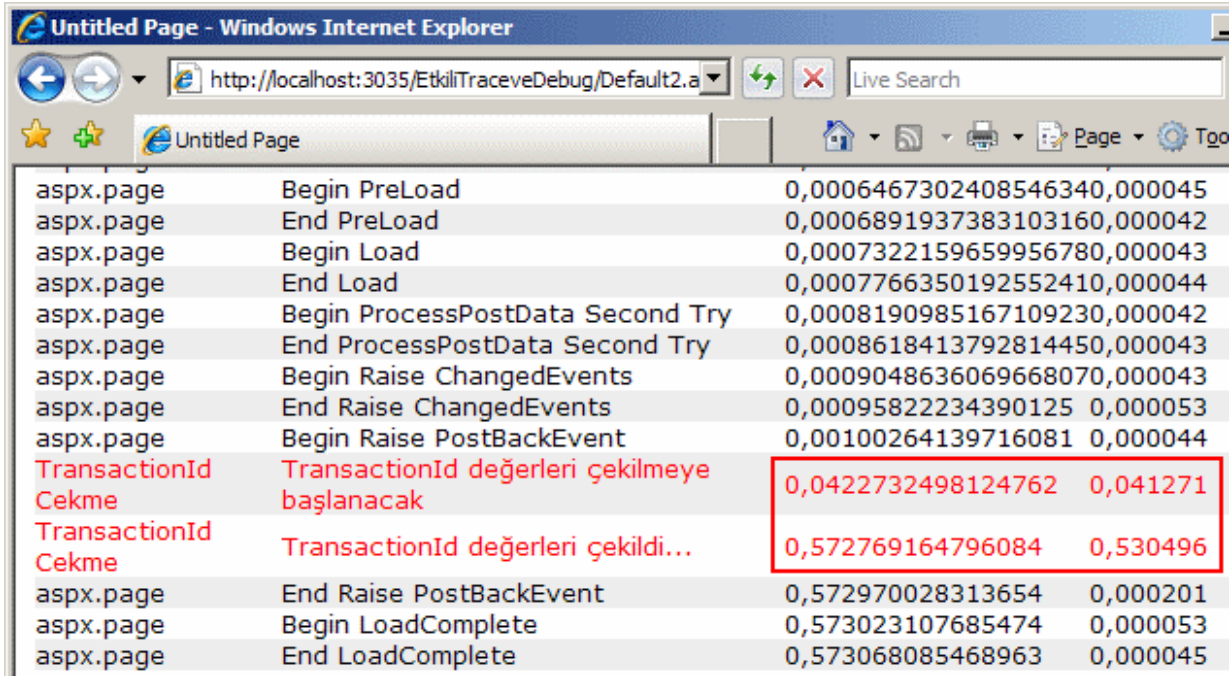
Category	Message	From First(s)	From Last (s)
aspx.page	Begin PreInit		
aspx.page	End PreInit	2,70984161394814E-05	0,000027
aspx.page	Begin Init	5,30793718196028E-05	0,000026
aspx.page	End Init	8,10158833036042E-05	0,000028
aspx.page	Begin InitComplete	0,00010951112501728	60,000028
aspx.page	End InitComplete	0,00013046350863028	70,000021
aspx.page	Begin LoadState	0,00015085716201360	80,000020
aspx.page	End LoadState	0,00024388574525533	30,000093
aspx.page	Begin ProcessPostData	0,00027210162185417	40,000028
aspx.page	End ProcessPostData	0,00030786035655369	60,000036
aspx.page	Begin PreLoad	0,00032993020062605	70,000022
aspx.page	End PreLoad	0,00035088258423905	80,000021
aspx.page	Begin Load	0,00037211433296689	90,000021
aspx.page	End Load	0,00039418417703926	10,000022
aspx.page	Begin ProcessPostData Second Try	0,00041457783042258	20,000020
aspx.page	End ProcessPostData Second Try	0,00043525084892074	30,000021
aspx.page	Begin Raise ChangedEvents	0,00045620323253374	40,000021
aspx.page	End Raise ChangedEvents	0,00047743498126158	50,000021
aspx.page	Begin Raise PostBackEvent	0,00049894609510426	60,000022
TransactionId Cekme	TransactionId değerleri çekilmeye başlanacak	0,783776150320781	0,783277
TransactionId Cekme	TransactionId değerleri çekildi...	4,51665840211535	3,732882
aspx.page	End Raise PostBackEvent	4,67447566660008	0,157817
aspx.page	Begin LoadComplete	4,67454969835552	0,000074
aspx.page	End LoadComplete	4,67457372375539	0,000024

Sonuçları irdelerken **From First(s)** ve **From Last(s)** kısımlarındaki süreler çok önemlidir. From First(s) ile sayfanın talepten sonraki yaşam döngüsü başladığından beri geçen toplam süre ifade edilir. Form Last(s) ise, bir önceki işlem ile son işlem arasındaki geçen süre farkıdır. Böylece kod yardımıyla sayfanın çalışma zamanı çıktısına bakılmış ve süre uzamasının nerede olduğu tespit edilebilmiştir. Görüldüğü gibi string bilgisini oluştururken + operatörü nedeni ile verinin oluşumu son derece fazla zaman almıştır. Aslında burada alınacak tedbir son derece basittir. + operatörü yerine **StringBuilder** kullanmak. Bunun için kod aşağıdaki gibi değiştirilebilir.

```
StringBuilder builder = new StringBuilder();
using (SqlConnection conn = new SqlConnection("data
source=.;database=AdventureWorks;integrated security=SSPI"))
{
   SqlCommand cmd = new SqlCommand("Select Top 10000 TransactionId From
Production.TransactionHistory", conn);
   conn.Open();
```

```
    Trace.Warn("TransactionId Cekme", "TransactionId değerleri çekilmeye başlanacak");
    SqlDataReader reader = cmd.ExecuteReader();
    while (reader.Read())
    {
       builder.Append(reader["TransactionId"].ToString());
       builder.Append("|");
    }
    Trace.Warn("TransactionId Cekme", "TransactionId değerleri çekildi...");
    reader.Close();
}
Session.Add("TumKategoriler", builder.ToString());
```

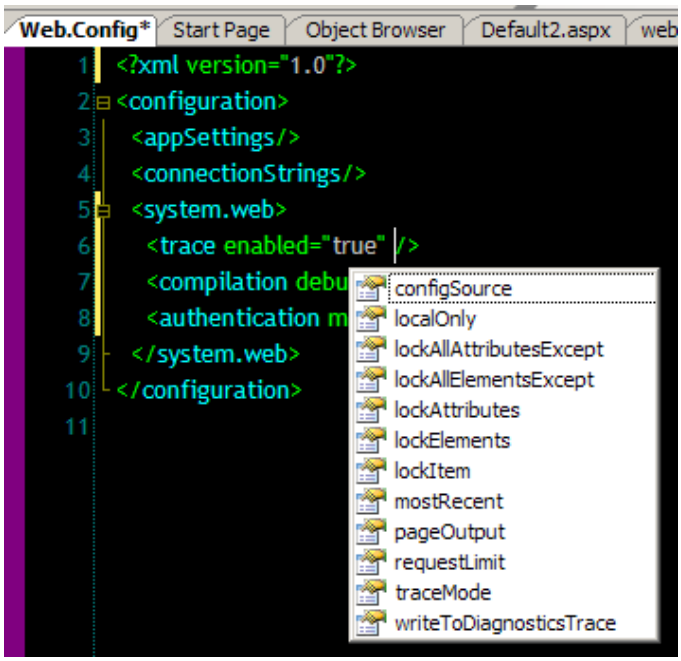
Buna göre sonuçlar aşağıdaki ekran görüntüsündeki gibi olacaktır.



Görüldüğü gibi kod içerisinde şüpheli bulunulan noktalarda uygulanacak teknikler ile sayfanın çalışma zamanındaki hali çok daha kolay bir şekilde izlenebilmektedir.

NOT : *Trace sınıfına ait* ***Write*** *ve* ***Warn*** *metodları uygulama içerisinde pek çok yerde kullanılabilir. Buna göre* ***Trace*** *mekanizmasının geçersiz kılınması haline, söz konusu metodlar görmezden gelineceği için, tüm uygulama kodunu gözden geçirerek bu metodlara ait satırların kaldırılmasına gerek kalmayacaktır.*

Uygulama seviyesinde Trace mekanizmasını aktif kılabilmek için **web.config** dosyasında **trace** elementi kullanılmalı ve **enabled** özelliğine **true** değeri atanmalıdır. Bunun dışında trace elementi içerisinden belirlenebilecek bazı ayarlamalarda yapılabilir. Böylece web uygulaması içerisindeki tüm sayfalar için izleme yapılabilir.



trace elementinin içerisinde kullanılabilecek nitelikler ve anlamları ise aşağıdaki tabloda olduğu gibidir.

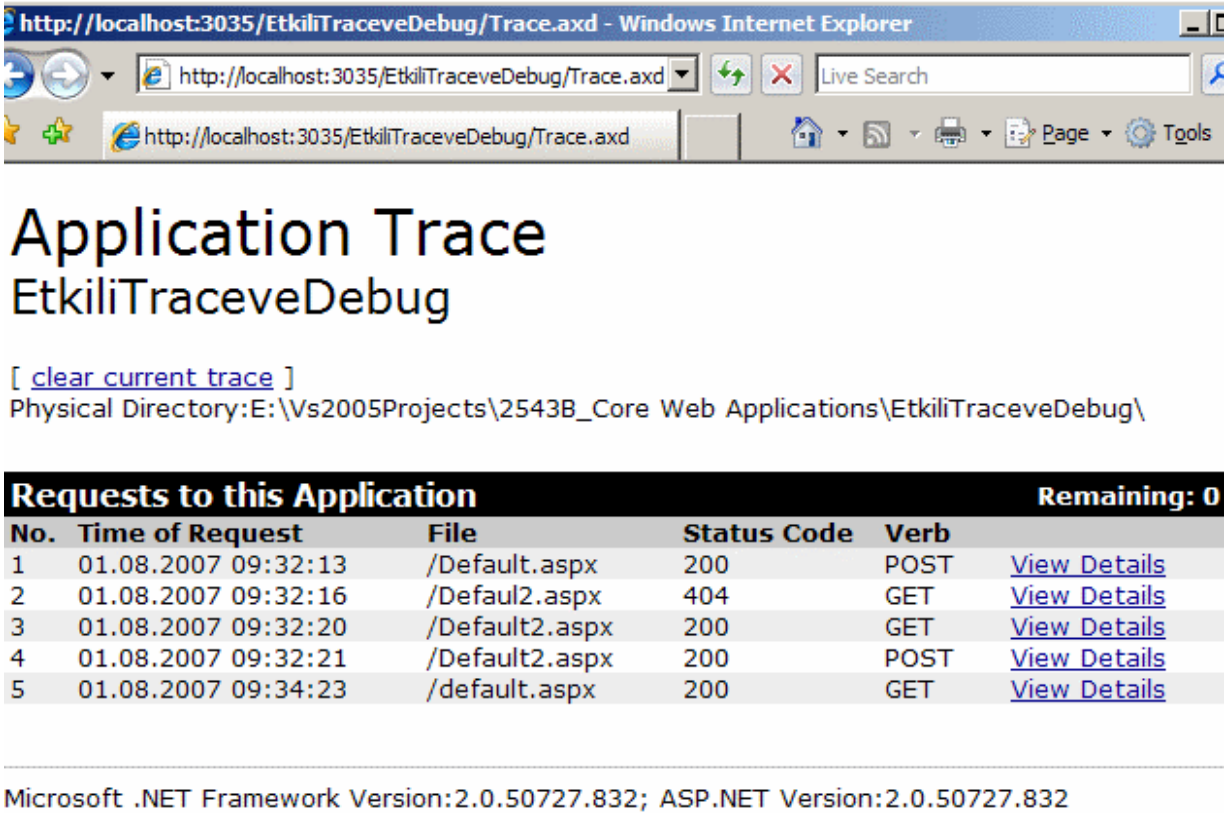
trace Elementine Ait Genel özellikler	
requestLimit	trace log içerisinde uygulamaya ait kaç **talebin(request)** saklanacağı uygulama yeniden başlatılana veya trace log bilinçli bir şekilde temizl olarak **10000** değeri verilebilir. 10000' den büyük bir değer verilmesi
pageOutput	**true** ise Trace bilgileri web uygulaması içerisindeki sayfaların sonuna edilebilir.
localOnly	**true** olmaslı halinde **Trace** çıktısını istemciler göremez. Sadece web sadece geliştiricinin izleyebilmesi anlamınada gelebilir ki güncel proj
enabled	**true** olması haline uygulama bazında**(Application Level)** izleme mod
mostRecent	varsayılan değeri **false** olan bu niteliğe **true** değeri verilirse , **request** Böylece son taleplerin görülebilmesi sağlanmış olur. Bu nitelik**(attrib**
traceMode	Daha öncedende değinildiği gibi, **Trace Information** kısmında yer al
writeToDiagnosticTrace	Varsayılan değeri **false** olan bu özellik Asp.Net 2.0 ile birlikte gelmiş gönderilmeyeceğini belirlemekte kullanılır.

Aşağıda örnek bir trace elementi içeriği yer almaktadır.

```
<?xml version="1.0"?>
<configuration>
   <appSettings/>
```

```
    <connectionStrings/>
    <system.web>
      <trace enabled="true" requestLimit="5" mostRecent="true"
pageOutput="false" localOnly="true" writeToDiagnosticsTrace="true" />
      <compilation debug="true"/>
      <authentication mode="Windows"/>
    </system.web>
</configuration>
```

Buna göre son 5 talebe ait trace bilgileri tutulacak, sayfa çıktısı verilmeyecek bir başka deyişle **trace.axd** ile talep edilebilecek, yanlızca host makinedeki kullanıcı trace.axd' ye bakabilecek ve bilgiler **System.Diagnostics** alt yapısına devredilebilecektir. Dolayısıyla uygulama çalıştırıldıktan sonra söz konusu izleme bilgileri aşağıdaki gibi olacaktır.

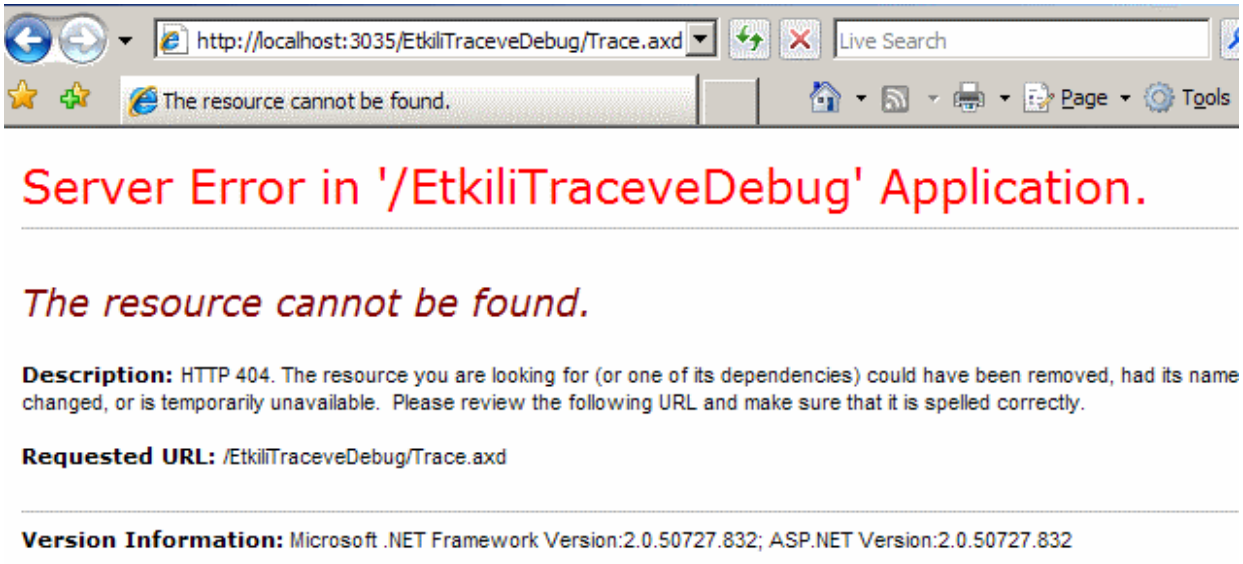


Burada sayfalara ait izleme detaylarına bakmak için **View Details** linki kullanılabilmektedir. çıktıda yer alan bilgilere bakıldığında taleplerin zamanı**(Time of Request)**, durumu-**status code**(örneğin talep edilen sayfa başarılı bir şekilde yüklendiyse 200, olmayan bir sayfa talep edildiyse 404...), HTTP' nin hangi metoduna göre (POST, GET...) talep edildiği**(Verb)** ve talep sırası**(No)**gibi bilgiler yer alır.

Geliştirilen web uygulaması dağıtılırken trace.axd dosyasının hiç bir şekilde talep edilememesi sağlanabilir. Bunun için web.config dosyasında yer alan system.web elementi altında aşağıdaki değişikliği yapmak yeterlidir.

```
<system.web>
   <httpHandlers>
      <remove verb="*" path="trace.axd"/>
   </httpHandlers>
</system.web>
```

Buna göre söz konusu web.config dosyasının içeren uygulamada herhangibir şekilde Trace.axd dosyası talep edilirse aşağıdaki ekran görüntüsü elde edilecektir.



Gelelim izleme ile ilişkili diğer konulara. Bazı durumlarda kod içerisinde kullandığımız Trace ifadelerinin sadece istisnalar oluştuğunda tutulmasını isteyebiliriz. Buna ek olarak, trace içerisine atılacak istisna bilgilerinin sadece host makinedeki kullanıcıya gösterilmesi istenebilir. Trace bilgisini sadece yerel kullanıcıya göstermek için **localOnly** özelliği kullanılabilir. Ancak burada durum biraz daha farklıdır. Nitekim, trace basılmakta ama içeride istisna oluşması halinde gösterilen içerik sadece yerel kullanıcı için oluşturulmak istenmektedir. Bu vakkayı çözmek için Trace ifadelerinin **catch** blokları içerisinde ele alınacağı ortadadır. Diğer taraftan **Write** veya **Warn** metodlarının üçüncü parametreleri burada önemlidir. Nitekim üçüncü parametre oluşan **istisna(Exception)** referansını taşımaktadır. Exception tipini trace çıktısına vermek dışında, talepte bulunan kullanıcınında host makineden geldiğini tespit etmek gerekir. Bu nedenle aşağıdaki adımlar izlenebilir.*(Olayın daha kolay anlaşılabilmesi için konu şema ile desteklenmiştir)*

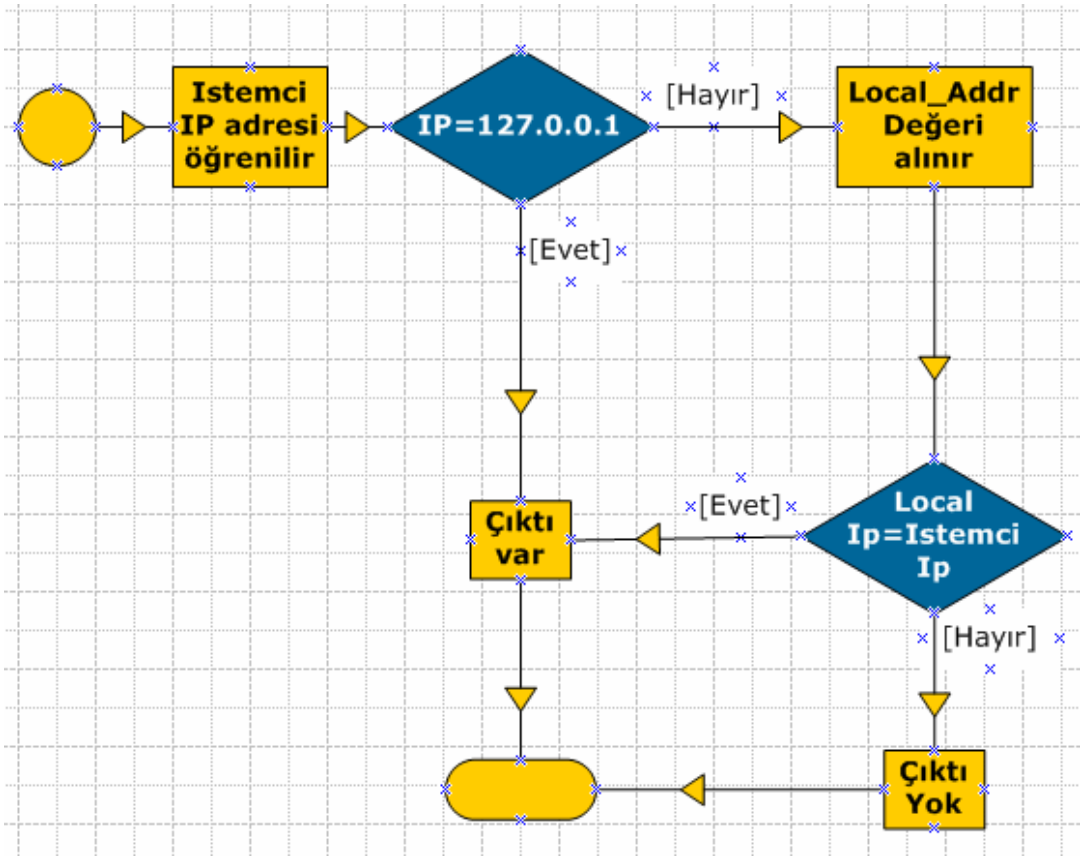
1. Request ile istemcinin **Host** adresi öğrenilir. Burada **Request**.**UserHostAddress**' den faydalanılabilir.

2. Elde edilen adresin **127.0.0.1** olup olmadığına bakılır. öyleyse talep yerel makineden gelmiştir ve trace açılabilir.

3. Talep yerel makineden gelmiyorsa **Requset**.**ServerVariables**("**LOCAL_ADDR**") ile elde edilen ip adresi ve **Request**.**UserHostAddress** ile elde edilen istemci adresi

karşılaştırılır. Bu yerel host adresinin 127.0.0.1' den farklı olmasına karşı alınan bir tedbirdir. Eğer eşitlerse talebin yine yerel makineden geldiği anlaşılabilir.

4. Eğer taleplerin yerelden geldiği anlaşıldıysa çıktı üretilir.



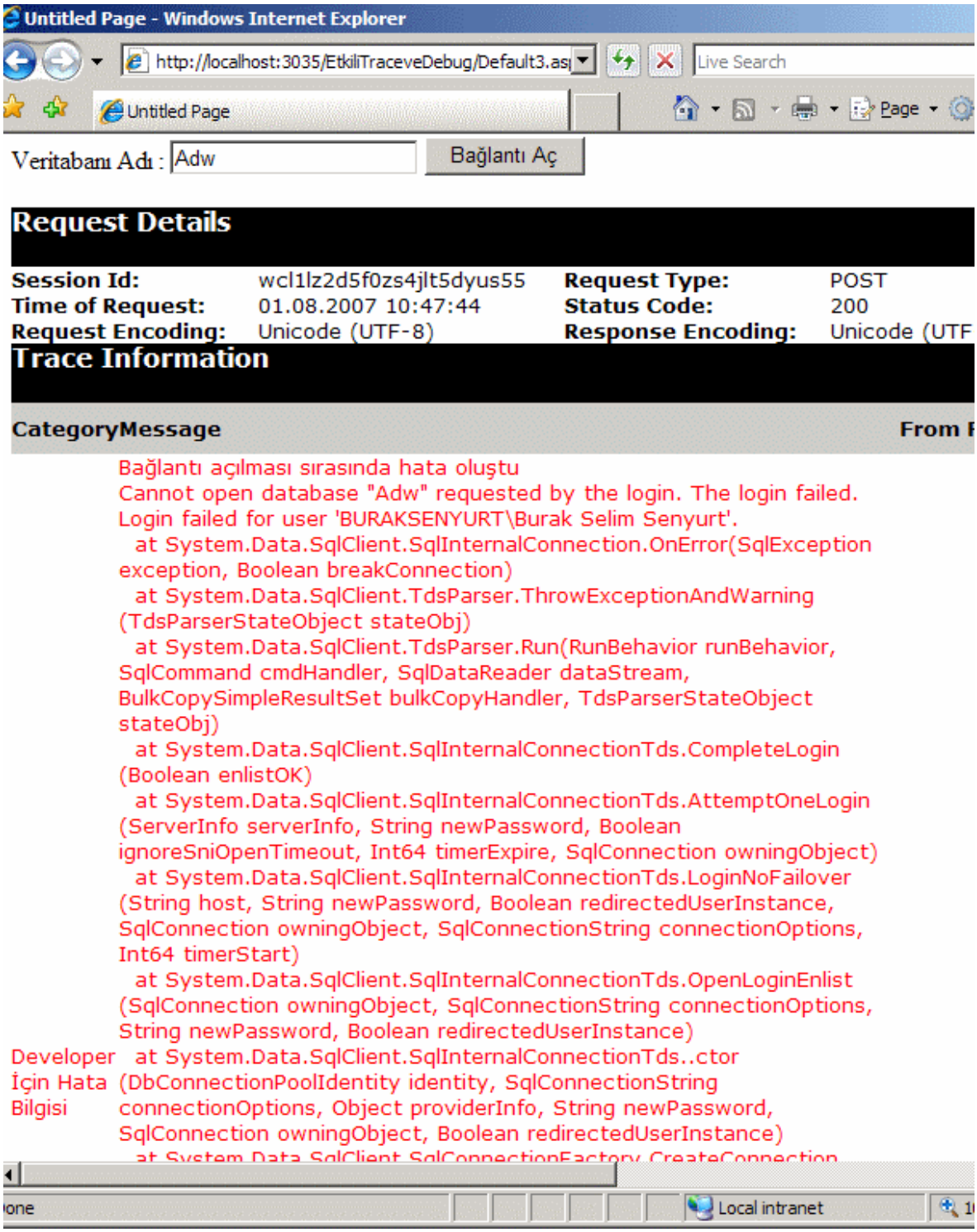
Kod tarafında ise örnek olarak aşağıdaki sayfa düşünülebilir. Burada bilinçli olarak bir **istisna(Exception)** oluşturulmaktadır.

```
<%@ Page Language="C#" %>
<%@ Import Namespace="System.Data" %>
<%@ Import Namespace="System.Data.SqlClient" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

   protected bool TraceYazilsinmi()
   {
      string userHostAddress = Request.UserHostAddress;
      if (userHostAddress == "127.0.0.1")
         return true;
      else
```

```
        {
            string localAddress
= Request.ServerVariables.GetValues("LOCAL_ADDR").ToString();
            if (localAddress == userHostAddress)
                return true;
            else
                return false;
        }
    }

    protected void btnBaglantiAc_Click(object sender, EventArgs e)
    {
        SqlConnection conn = null;
        try
        {
            conn = new SqlConnection("data source=.;database=" + txtVeritabani.Text +
";integrated security=SSPI");
            conn.Open();
            Response.Write("Bağlantı açıldı");
        }
        catch (Exception excp)
        {
            if (TraceYazilsinmi())
            {
                Trace.IsEnabled = true;
                Trace.Warn("Developer İçin Hata Bilgisi", "Bağlantı açılması sırasında hata
oluştu", excp);
            }
        }
        finally
        {
            if (conn != null
                && conn.State == ConnectionState.Open)
                    conn.Close();
        }
    }

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
    <form id="form1" runat="server">
```

```
        <div>
            Veritabanı Adı : <asp:TextBox ID="txtVeritabani" runat="server" />
            <asp:Button ID="btnBaglantiAc" runat="server" Text="Bağlantı Aç"
OnClick="btnBaglantiAc_Click" />
        </div>
    </form>
</body>
</html>
```

Uygulamada kullanıcı textbox kontrolünden girdiği veritabanı adı için bir bağlantı açmaya çalışmaktadır. Eğer bağlantının açılması sırasında bir hata oluşursa ve kullanıcı host makineden talepte bulunmuşsa Trace çıktısının etkinleştirilmesi ve istisna mesajının buraya yazdırılması sağlanır. Burada istemcinin **Ip** adresini tedarik edebilmek için**HttpRequest** sınıfının **UserHostAddress** özelliği kullanılır. **Local_Addr** değeri ilede sunucu değişkenlerinden**(Server Variables)** sunucu ip adresi elde edilmektedir. Nitekim sunucu ip adresi 127.0.0.1' den farklıda olabilir. O halde talep eden istemcinin ip adresi ile yerel ip adresinin eşit olup olmadığına da bakılmalıdır. Uygulamayı test edip, örneğin olmayan bir veritabanı adı girdiğimizde sonuç sayfası aşağıdaki gibi olacaktır.



Dikkat edilecek olursa istisna bilgisi detayaları ile birlikte **Trace Information** kısmında özel kategori adı ve mesajı ile birlikte görülmektedir. Bu tarz bir çalışma zamanı bilgisi elbetteki geliştirici(Developer) açısından önemlidir. Aynı sonuçlar çok doğal olarak Trace mekanizması olmadan da tespit edilebilir. Buradaki temel amaç Trace mekanizmasını kullanarak, sayfalarda oluşabilecek hataların kesin yerlerini ve konumlarını daha kolay tespit edebilmektir.

Gelelim Trace mekanizması ile ilgili diğer bir konuya. çok doğal olarak web uygulamalarında harici bileşenler**(Components)** kullanılır. Bileşenden kastımız çoğunlukla bir sınıf kütüphanesi**(class library)** veya ayrı bir sınıf dosyasıdır. çok doğal olarak bu bileşenler içerisindeki bazı süreçler Trace mekanizması içerisinde ele alınmak istenebilir. örneğin ilk uygulamada gerçekleştirdiğimiz string birleştirme işleminin ayrı bir sınıf içerisinde bir metod olarak ele alındığını düşünelim.

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Text;
using System.Data.SqlClient;

public class VeriBileseni
{
   public string GetTransactionIdString()
   {
      HttpContext ctx = HttpContext.Current;
      StringBuilder builder = new StringBuilder();
      using (SqlConnection conn = new SqlConnection("data
source=.;database=AdventureWorks;integrated security=SSPI"))
      {
         SqlCommand cmd = new SqlCommand("Select Top 10000 TransactionId From
Production.TransactionHistory", conn);
         conn.Open();
         ctx.Trace.Warn("TransactionId Cekme", "TransactionId değerleri çekilmeye
başlanacak");
         SqlDataReader reader = cmd.ExecuteReader();
         while (reader.Read())
         {
            builder.Append(reader["TransactionId"].ToString());
            builder.Append("|");
         }
         ctx.Trace.Warn("TransactionId Cekme", "TransactionId değerleri çekildi...");
         reader.Close();
      }
      return builder.ToString();
   }
   public VeriBileseni()
```

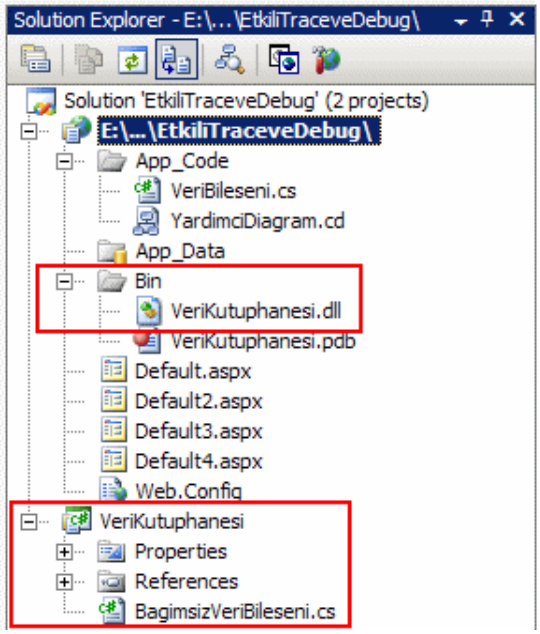
```
    {
    }
}
```

Burada dikkat edilmesi gereken en önemli nokta Trace sınıfını kullanmak için **HttpContext** nesnesinin nasıl elde edildiğidir. Bileşenlerde **Trace** bilgisi ele alınıyorsa, bu bileşenin kullanıldığı HTTP içeriğinin kullanılması gerekmektedir. Bu nedenle **HttpContext** nesne örneği için statik**(static) Current** özelliğinden faydalanılmıştır. Böylece bileşeni o anda kullanan sayfa içeriğine ulaşılmış olunur. Bundan sonra ise **Warn** veya **Write** gibi metodlar kullanılabilir.

NOT : *Bileşenin ayrı bir **sınıf kütüphanesi(class library)** olarak tasarlanması durumunda, **System.Web.dll** assembly' ının referans edilmesi gerekecektir. Bunun ise doğuracağı önemli sonuçlardan birisi şudur; bilşenen içerisindeki izleme alt yapısı web bağımlı hale gelmektedir. Bu nedenle alternatif bir yaklaşım olarak ayrı bir dinleyici mekanizması özel olarak geliştirilebilir.*

Peki söz konusu bileşen sadece web tabanlı kullanılmıyorsa. Bu durumda **HttpContext** gerekli işlevsellikleri sağlamak için uygun olmayacaktır. Dolayısıyla farklı bir yol izlemek gerekmektedir. .Net Framework izleme mesajları için dinleyiciler**(Listener)** kullanır. İstenirse **web.config** aracılığıyla yada **programatik** olarak yeni dinleyiciler eklenebilir. Normal şartlarda **Trace.Write** gibi bir metod çağrısı yapıldığında **TraceListener** koleksiyonundaki tüm dinleyiciler mesajları alıp işlemeye başlarlar. O halde Trace Listener web.config ile açık bir şekilde belirtilirse,**System.Web.dll** assembly' ını projeye referans etmeden ve HttpContext tipini kullanmadan Trace çıktıları web uygulamasına doğru gönderilebilir.

Asp.Net 1.1 ile geliştirme yapıyorsak eğer, bu işlemler için özel bir dinleyici yazmamız gerekecektir. Ne varki **Asp.Net 2.0** ile sadece bu iş için tasarlanmış **WebPageTraceListener** isimli bir sınıf gelmektedir. Bu sınıfın web.config dosyasında belirtilmesi ve bileşen içerisinde **System.Diagnostics** isim alanı altında yer alan **Trace** sınıfının kullanılması yeterlidir. Böylece bileşenimiz trace çıktısı verirken web' e bağımlı olmaktan kurtulmuş olacaktır. örnek olarak az önce geliştirilen VeriBileseni sınıfını ayrı bir sınıf kütüphanesi olarak aşağıdaki gibi tasarladığımızı düşünelim.




```
using System;
using System.Data;
using System.Diagnostics;
using System.Data.SqlClient;
using System.Text;

public class BagimsizVeriBileseni
{
   public string GetTransactionIdString()
   {
      StringBuilder builder = new StringBuilder();
      using (SqlConnection conn = new SqlConnection("data
source=.;database=AdventureWorks;integrated security=SSPI"))
      {
         SqlCommand cmd = new SqlCommand("Select Top 10000 TransactionId From
Production.TransactionHistory", conn);
         conn.Open();
         Trace.Write("TransactionId Cekme", "TransactionId değerleri çekilmeye
başlanacak");
         SqlDataReader reader = cmd.ExecuteReader();
         while (reader.Read())
         {
            builder.Append(reader["TransactionId"].ToString());
            builder.Append("|");
         }
         Trace.Write("TransactionId Cekme", "TransactionId değerleri çekildi...");
         reader.Close();
```

```
        }
        return builder.ToString();
    }
}
```

Burada dikkat edilecek olursa herhangibir şekilde System.Web referansı kullanılmamaktadır. Bunun yerine System.Diagnostics isim alanı ve burada yer alan Trace sınıfı ele alınmaktadır. Bir önceki bileşenden farklı olarak Warn metodu kullanılamamaktadır. Bunun nedeni **System.Diagnostics** isim alanında yer alan Trace sınıfının Warn metodunun olmayışıdır. Diğer önemli noktalardan biriside, **System.Diagnostics.Trace** sınıfındaki Write metodu versiyonlarından belirli bir **Exception** nesne örneğininin fırlatılmasının mümkün olmayışıdır. Ancak istisna mesajı gönderilmesi sağlanabilir. Bu işlemin ardından web uygulamasına ait web.config dosyasında aşağıdaki değişiklikler yapılmalıdır.

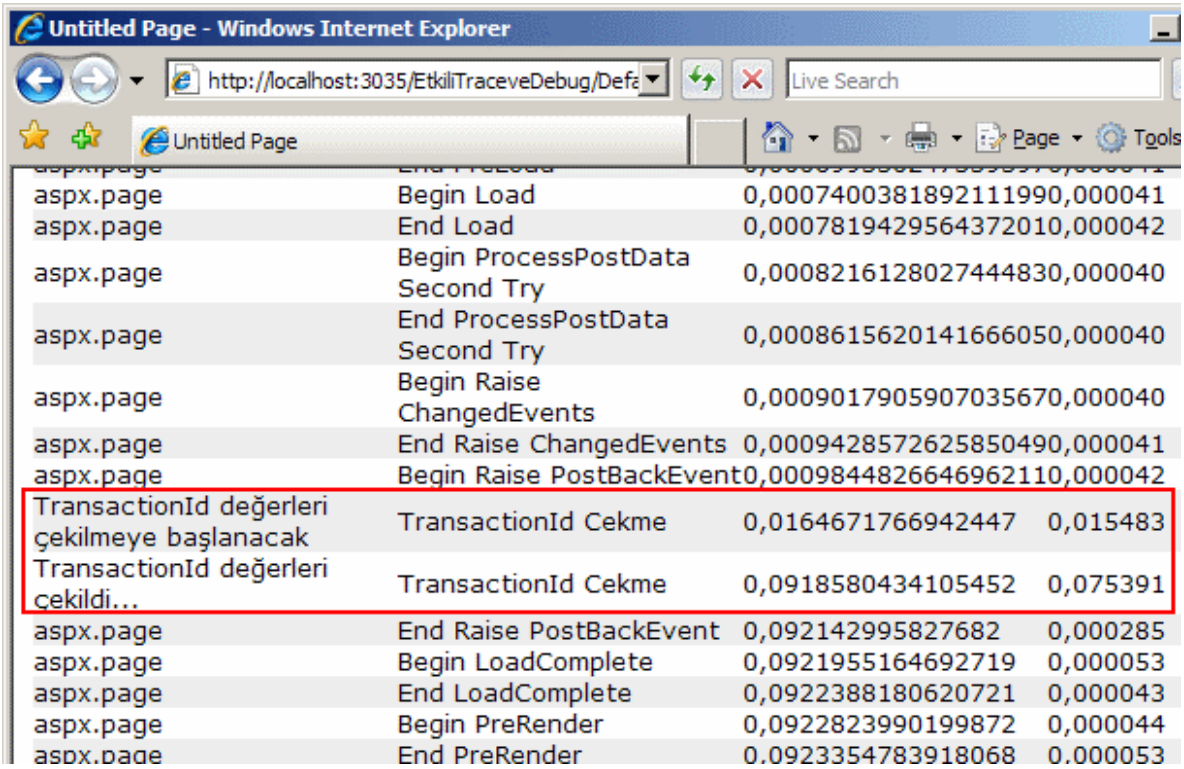
```
<?xml version="1.0"?>
<configuration>
    .
    .
    .
    <system.diagnostics>
        <trace>
            <listeners>
                <add name="WebPageTraceListener" type="System.Web.WebPageTraceListe
ner,System.Web,Version=2.0.3600.0,Culture=neutral,
PublicKeyToken=b03f5f7f11d50a3a" />
            </listeners>
        </trace>
    </system.diagnostics>
    .
    .
    .
</configuration>
```

Burada **listeners** bilgisi eklenirken **system.diagnostics** isim elementi içerisindeki **trace** elementi kullanılmaktadır. add elementi içerisinde yer alan type kımsında WebPageTraceListener sınıfının tam adı (Qualified Name) belirtilmektedir. Bildiğiniz üzere **Qualified Name**' i oluşturan değerler tip adı, assembly adı, versiyon numarası, kültür ve publicKeyToken bilgisidir. Bu durumu test etmek için az önceki örnektekine benzer bir web sayfası aşağıdaki gibi geliştirilebilir.

```
<%@ Page Language="C#" Trace="true" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
```

```
<script runat="server">

   protected void btnCek_Click(object sender, EventArgs e)
   {
      BagimsizVeriBileseni veriBln = new BagimsizVeriBileseni();
      Session.Add("TransactionIDler", veriBln.GetTransactionIdString());
   }

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
<title>Untitled Page</title>
</head>
<body>
   <form id="form1" runat="server">
      <div>
         <asp:Button ID="btnCek" runat="server" Text="Veriyi çek"
OnClick="btnCek_Click" />
      </div>
   </form>
</body>
</html>
```

Trace çıktısı ise aşağıdaki gibi olacaktır.



aspx.page	Begin Load	0,0007400381892111990,000041	
aspx.page	End Load	0,0007819429564372010,000042	
aspx.page	Begin ProcessPostData Second Try	0,0008216128027444830,000040	
aspx.page	End ProcessPostData Second Try	0,0008615620141666050,000040	
aspx.page	Begin Raise ChangedEvents	0,0009017905907035670,000040	
aspx.page	End Raise ChangedEvents	0,0009428572625850490,000041	
aspx.page	Begin Raise PostBackEvent	0,0009844826646962110,000042	
TransactionId değerleri çekilmeye başlanacak	TransactionId Cekme	0,0164671766942447	0,015483
TransactionId değerleri çekildi...	TransactionId Cekme	0,0918580434105452	0,075391
aspx.page	End Raise PostBackEvent	0,092142995827682	0,000285
aspx.page	Begin LoadComplete	0,0921955164692719	0,000053
aspx.page	End LoadComplete	0,0922388180620721	0,000043
aspx.page	Begin PreRender	0,0922823990199872	0,000044
aspx.page	End PreRender	0,0923354783918068	0,000053

Trace mimarisi ile ilgili olaraktan, ele alınabilecek başka konularda bulunmaktadır. örneğin Trace bilgilerini başka ortamlara aktarmak, mail gönderimi gerçekleştirilmesi gibi. Tatile çıktığım şu günlerde bu kadar bilginin yeterli olacağı kanısındayım. Dönüşte Trace mimarisinin ikinci makalesi ile devam edeceğiz. Böylece geldik bir makalemizin daha sonuna. Bu makalemizde trace mimarisini tanımaya, gerekliliklerini vurgulamaya çalıştık. Sayfa seviyesinde ve uygulama seviyesinde trace işlemlerinin nasıl yapılacağını, sadece istisna oluştuğunda yanlız yerel makineye trace bilgisinin nasıl verileceğini, bileşen bazında trace' lerin kullanılmasını ve bileşenin web ortamından bağımsız olabilecek şekilde ele alınabilmesini incelemeye çalıştık. Nihayetinde bu uzun makaleyi buraya kadar sabırla okuduğunuz için teşekkür eder bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

örnek Uygulama için Tıklayın

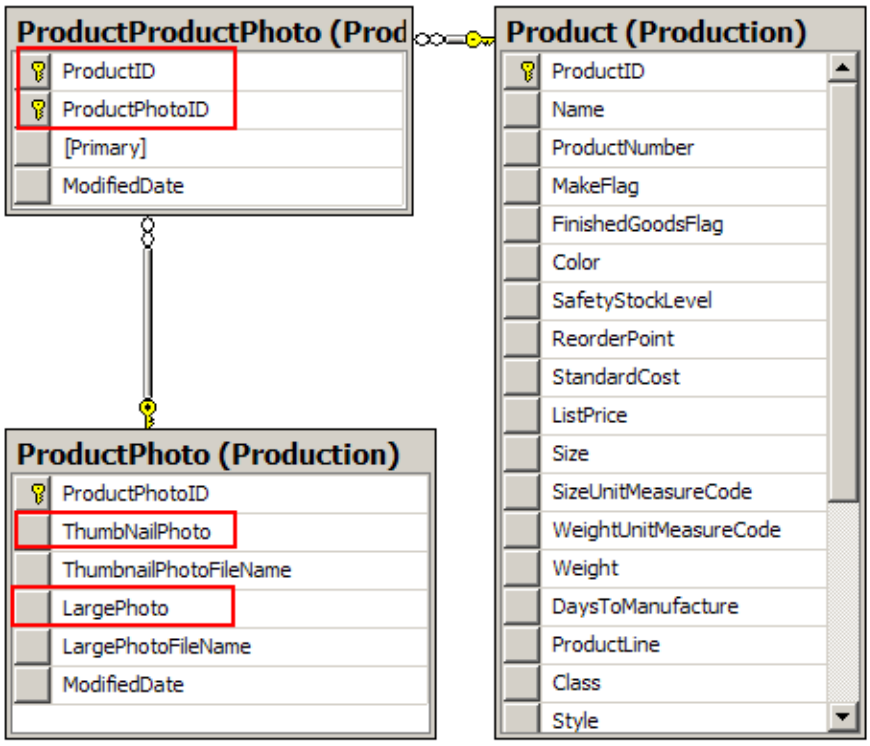
Asp.Net Uygulamalarında Tablo Bazlı Resimleri Ele Almak (2007-07-26T20:25:00)

asp.net temelleri,

Yazın bu sıcak günlerinde daha hafif konularla web maceralarımıza devam ediyoruz. Geçtiğimiz makalemizde Asp.Net uygulamalarında ektin hata yönetiminin nasıl yapılabileceğini incelemeye çalışmıştık. Bu kez veritabanı tablolarında çoğunlukla binary alanlarda saklanan resimlerin, Asp.Net uygulamalarında nasıl ele alınabileceğini örnek projeler üzerinden incelemeye çalışacağız. Bir Windows uygulaması göz önüne alındığında, resimleri gösterebilecek bir **PictureBox** kontrolünün çeşitli özelliklerinden

yararlanarak herhangibir tabloda tutulan binary içeriği kullanmak ve bu içeriğin işaret ettiği resmi göstermek son derece kolaydır. Ne varki Asp.Net uygulamalarında her zaman için, **render** edilerek istemciye gönderilen bir sayfa içeriği mevcuttur. Bu içeriğin tipi**(Content Type)** daha farklıdır. Dolayısıyla binary formatta tutulan resimleri ele almak için farklı bir yaklaşım gerekmektedir.

Tabloda binary formatta tutulabilen resimleri Asp.Net uygulamalarında ele almak amacıyla, gösterilmek istenen resmi tek başına yorumlayan bir Asp.Net sayfası mevcuttur. Bu sayfanın tek bir görevi vardır o da ilgili resmi **image** formatlarından uygun olana göre sayfaya **Render** etmektir. Bunu incemelek için örnek bir senaryo göz önüne almakta fayda olacağı kanısındayım. Bu amaçla**SQL Server 2005** ile birlikte gelen ve Production şemasında**(Schema)** bulunan **Product, ProductPhoto** ve **ProductProductPhoto** tabloları göz önüne alınabilir. Bu tablolar arasındaki ilişki kısaca aşağıdaki şekilde görüldüğü gibidir.



ProductPhoto isimli tabloda yer alan **ThumbNailPhoto** ve **LargePhoto** isimli alanlarda **binary** olarak ürün resimleri saklanmaktadır*(tam olarak varbinary tipinde)*. Buna göre ilk örnek senaryomuzda kullanıcılar ürünlerin listelendiği bir sayfadan detay bilgilerini almak için başka bir sayfaya geçiş yapacaklardır. Detayların verildiği sayfada ürüne ait **ThumbNailPhoto** içeriğide bir resim olarak sayfa gösterilecektir. Başlamadan önce ProductPhoto tablosundaki herhangibir ThumbNailPhoto alanının içeriğini resim olarak nasıl gösterebileceğimize bakalım. Bu amaçla ResimGoster.aspx isimli aşağıdaki gibi bir sayfa tasarlanarak işe başlanabilir.

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ResimGoster.aspx.cs" Inherits="ResimGoster" %>

Eminimki ResimGoster isimli web sayfasının içeriği son derece ilginç gelmiştir. Nitekim herhangibir HTML elementi yer almamaktadır. Aslında bu sayfanın tek amacı yüklenirken*(bir başka deyişle **Page_Load** olay metodu çalışırken)*, ürün resmini ekrana **binary** olarak yazdırmaktır. Burada elbetteki hangi resmin gösterileceğide önemlidir. Bunun için sayfaya bir şekilde ProductPhotoID alanının değerinin gelmesi gerekmektedir. Bunun için en güzel yol **QueryString** kullanımıdır. öyleyse bu sayfanın kodlarını yazarak işe devam edelim.

```
protected void Page_Load(object sender, EventArgs e)
{
   string resimId = Request.QueryString["ResimId"];
   if (!String.IsNullOrEmpty(resimId))
   {
      byte[] resimBytes=null;
      using (SqlConnection conn = new SqlConnection("data
source=localhost;database=AdventureWorks;integrated security=SSPI"))
      {
         SqlCommand cmd = new SqlCommand("Select ThumbNailPhoto From
Production.ProductPhoto Where ProductPhotoId=@PhotoId", conn);
         cmd.Parameters.AddWithValue("@PhotoId", resimId);
         conn.Open();
         SqlDataReader reader =
cmd.ExecuteReader(CommandBehavior.SequentialAccess);
         if(reader.Read())
            resimBytes = reader.GetSqlBytes(0).Value;
         reader.Close();
         if (resimBytes != null)
         {
            Response.ContentType = "image/gif";
            Response.BinaryWrite(resimBytes);
         }
         else
            Response.Write("Resim gösterilemiyor.");
      }
   }
   else
      Response.Write("ResimId parametresi eksik yada hatalı.");
}
```

Page_Load olay metodu içerisinde ilk olarak **HttpRequest** sınıfının static **QueryString** özelliği yardımıyla sayfaya gelen ResimId değeri alınmaktadır. Kullanıcıların bu sayfayı doğrudan talep etme ihtimaline göre ResimId değerinin boş gelme

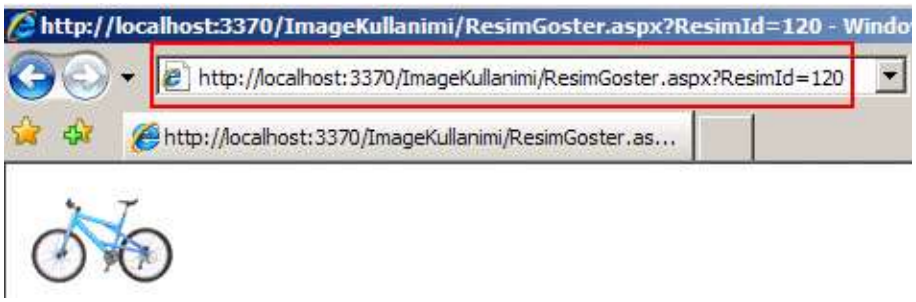
olasılığı bulunmaktadır. Bu nedenle **String** sınıfının **IsNullOrEmpty** metodu ile bir kontrol gerçekleştirilir. Sonrasında ise Production şemasındaki ProductPhoto tablosunda gelen değere göre ilgili ThumbNailPhoto alanı çekilir. Eğer gelen ResimId ile eşleşen bir ProductPhotoId alanı var ise bu satırın ThumbNailPhoto alanının değeri **SqlDataReader** nesne örneğine ait **Read** metodu yardımıyla okunur. Okuma işlemi sırasında **GetSqlBytes** metodu kullanılmakta ve **Value** özelliği ile elde edilen **byte** dizisi resimBytes isimli alana aktarılmaktadır. Diğer taraftan **HttpResponse** sınıfının **ContentType** özelliği ile render edilecek sayfanın içeriği belirlenmektedir. Burada **image/gif**değeri ile basılacak içeriğin gif formatında bir resim olacağı belirtilmektedir.

***NOT: ContentType** özelliğinin varsayılan değeri **text/HTML** dir. Bu değer, tahmin edileceği üzere sayfanın çıktısının HTML olarak üretileceğini işaret etmektedir. Yaygın olarak kullanılan diğer versiyonlar aşağıdaki gibidir.*

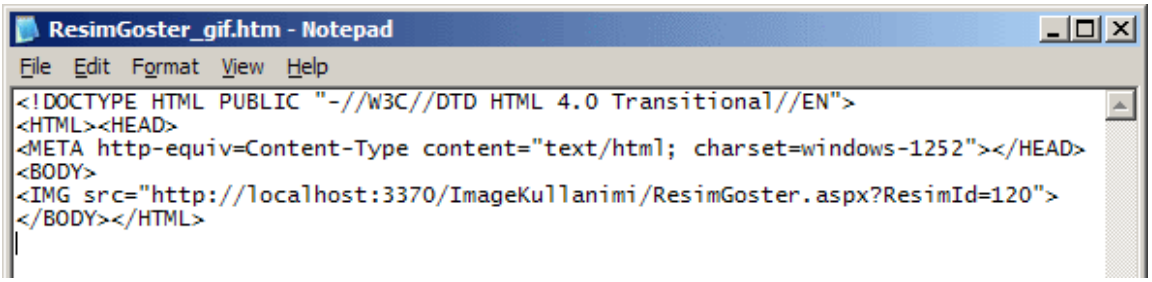
- *image/gif*
- *image/jpeg*
- *text/plain*
- *application/vnd.ms-excel (çıktının excel dökümanı olmasını sağlar)*
- *application/vnd.ms-word (çıktının word dökümanı olmasını sağlar)*

Son olarak, elde edilen byte dizisinin çıktıya aktarılmasını sağlamak için yine **HttpResponse** sınıfının **static** metodlarından **BinaryWrite** çağırılmaktadır.

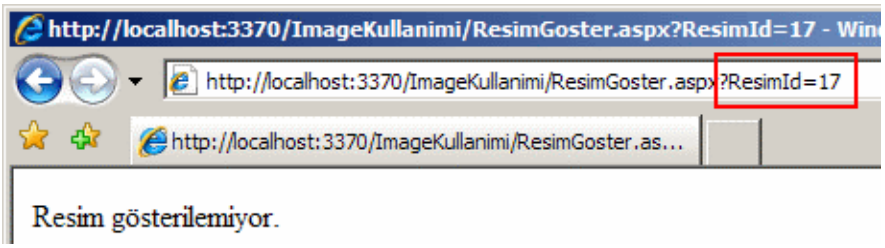
Artık sayfayı test ederek işlemlerimize devam edebiliriz. Elbette doğru sonuçları görebilmek için ResimId parametresini Url satırından göndermekte fayda vardır. Aşağıda örnek olarak**120** numaralı ProductPhotoId değerine sahip satır için elde edilen çıktı görülmektedir.



Burada, üretilen HTML sayfasının kaynak kodlarına bakılmak istenirse tarayıcı buna izin vermeyebilir*(örneğin Microsoft Internet Explorer 7.0 View Source buna izin vermemiştir)*. Bu sebepten çıktıyı **Save As** ile kaydetmek gerekebilir. Kaydedilen çıktının içeriği aşağıdaki ekran görüntüsünde yer aldığı gibi olacaktır. Dikkat edilecek olursa sayfanın çıktısı sonucu oluşturulan içerikte **img** elementi ve **src** niteliği yer almaktadır.



Elbette satır olarak karşılığı olmayan bir ResimId değeri girilirse sayfa çıktısı tarayıcı penceresinde aşağıdaki gibi olacaktır. örneğin ProductPhotoId değeri 17 olan bir satır bulunmamaktadır.



Bununla birlikte kullanıcı bu sayfayı doğrudan talep eder ve ResimId parametresini kullanmassa aşağıdaki ekran çıktısını elde eder.

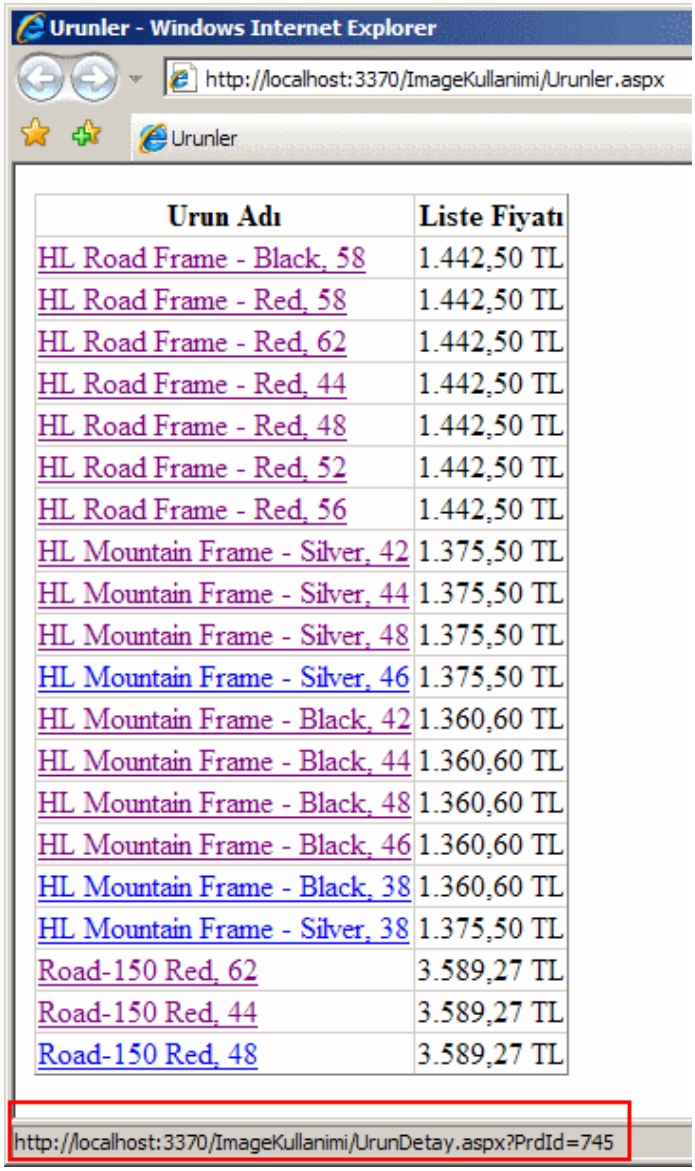
Burada olası bazı hataların önüne geçilmek amacıyla basit tedbirler alınmış ve ekrana bilgi mesajları verilmiştir. Gerçek hayat uygulamalarında son kullanıcıların daha doğru ve etkin bir şekilde uyarılması bir başka deyişle oluşan hatalar konusunda bilgilendirilmesi gerekmektedir.

Artık tek yapılması gereken senaryoyu biraz daha kullanışlı hale getirmektir. Bu amaçla ürünlerin gösterildiği Urunler.aspx isimli basit bir web sayfası tasarlanarak devam edilbilir. Bu sayfada ürünlere ait bir kaç temel bilgi bulunacak ama detayları için başka bir sayfaya yönlendirmede bulunulacaktır. Yönlendirilme yapılan sayfa tahmin edileceği üzere ürüne ait resmide içeren bir detay sayfasıdır. Urunler.aspx sayfasında basit olarak bir **SqlDataSource** kontrolü ve bu kontrolü ele alan bir **GridView** bileşeni düşünülebilir. Buna göre Urunler.aspx sayfasının kaynak kod tarafı aşağıdaki gibi tasarlanabilir.

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Urunler.aspx.cs" Inherits="_Default" %>

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Urunler</title>
</head>
<body>
   <form id="form1" runat="server">
      <div>
         <asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="False"
DataKeyNames="ProductID" DataSourceID="dsProducts">
            <Columns>
               <asp:HyperLinkField DataNavigateUrlFields="ProductId" DataNavigate
UrlFormatString="UrunDetay.aspx?PrdId={0}" DataTextField="Name" HeaderText
="Urun Adı" />
               <asp:BoundField DataField="ListPrice" DataFormatString="{0:C}"
HeaderText="Liste Fiyatı" HtmlEncode="False" SortExpression="ListPrice" />
            </Columns>
         </asp:GridView>
         <asp:SqlDataSource ID="dsProducts" runat="server" ConnectionString="<%$
ConnectionStrings:AdvConStr %>" SelectCommand="SELECT Top 20 ProductID,
Name, ListPrice FROM Production.Product Where ListPrice>=1000">
         </asp:SqlDataSource>
      </div>
   </form>
</body>
</html>
```

Urunler.aspx sayfasında yer alan GridView kontrolünde **HyperLinkField** kontrolü kullanılmaktadır. Bu alan, ürünün adını(Name) göstermekte olup üzerine tıklandığında kullanıcıyı UrunDetay.aspx sayfasına göndermektedir. Bu işlem sırasında da PrdId isimli bir **QueryString** parametresi ProductId alanının değerini detay sayfasına taşımaktadır. Urunler.aspx isimli sayfanın çalışma zamanındaki çıktısı aşağıdaki ekran görüntüsünde yer aldığı gibidir.



Urun Adı	Liste Fiyatı
HL Road Frame - Black, 58	1.442,50 TL
HL Road Frame - Red, 58	1.442,50 TL
HL Road Frame - Red, 62	1.442,50 TL
HL Road Frame - Red, 44	1.442,50 TL
HL Road Frame - Red, 48	1.442,50 TL
HL Road Frame - Red, 52	1.442,50 TL
HL Road Frame - Red, 56	1.442,50 TL
HL Mountain Frame - Silver, 42	1.375,50 TL
HL Mountain Frame - Silver, 44	1.375,50 TL
HL Mountain Frame - Silver, 48	1.375,50 TL
HL Mountain Frame - Silver, 46	1.375,50 TL
HL Mountain Frame - Black, 42	1.360,60 TL
HL Mountain Frame - Black, 44	1.360,60 TL
HL Mountain Frame - Black, 48	1.360,60 TL
HL Mountain Frame - Black, 46	1.360,60 TL
HL Mountain Frame - Black, 38	1.360,60 TL
HL Mountain Frame - Silver, 38	1.375,50 TL
Road-150 Red, 62	3.589,27 TL
Road-150 Red, 44	3.589,27 TL
Road-150 Red, 48	3.589,27 TL

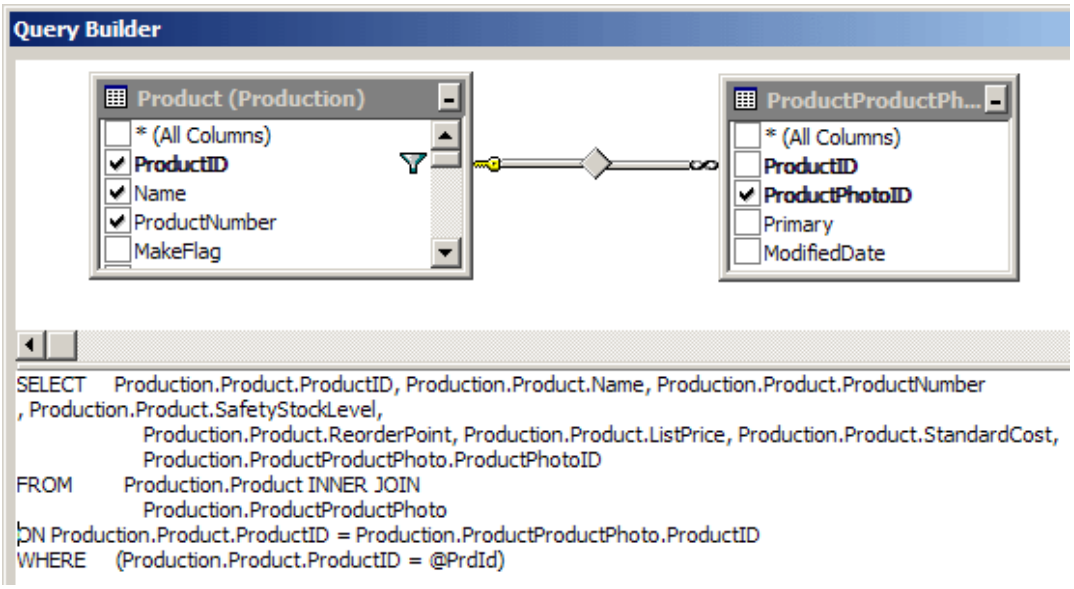
Gelelim UrunDetay.aspx sayfasına. Bu sayfada basit olarak Urunler.aspx sayfasında seçilen ürünlere ait detay bilgileri gösterilecektir. Ancak önemli olan, seçilen ürünün resmininde ProductPhoto tablosundan binary olarak çekilerek ekrana bastırılacak olmasıdır. Bu amaçla tasarlanan Urunler.aspx sayfasında yine bir **SqlDataSource** kontrolü ve detaylar için **DetailsView** bileşeni aşağıdaki gibi kullanılabilir.

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="UrunDetay.aspx.cs"
Inherits="UrunDetay" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
```

```
   <title>Urun Detaylari</title>
</head>
<body>
   <form id="form1" runat="server">
      <div>ürün Detayları :<br />
         <asp:DetailsView ID="DetailsView1" runat="server" AutoGenerateRows="False"
DataSourceID="dsProductDetails" Height="50px" Width="294px">
            <Fields>
               <asp:BoundField DataField="Name" HeaderText="&#220;r&#252;n Adı"
SortExpression="Name" />
               <asp:BoundField DataField="ProductNumber" HeaderText="&#220;r&#252;n
Numarası" SortExpression="ProductNumber" />
               <asp:BoundField DataField="SafetyStockLevel" HeaderText="Stok Seviyesi"
SortExpression="SafetyStockLevel" />
               <asp:BoundField DataField="ReorderPoint" HeaderText="Sipariş Noktası"
SortExpression="ReorderPoint" />
               <asp:BoundField DataField="ListPrice" HeaderText="Liste Fiyatı"
SortExpression="ListPrice" DataFormatString="{0:C}" HtmlEncode="False" />
               <asp:BoundField DataField="StandardCost" HeaderText="Standart Maliyet"
SortExpression="StandardCost" DataFormatString="{0:C}" />
               <asp:TemplateField HeaderText="Urun Resmi">
                  <ItemTemplate>
                     <img alt="ürün Resmi"
runat="server" src='<%#"ResimGoster.aspx?ResimID="+DataBinder.Eval(Container.
DataItem, "ProductPhotoID") %>' id="urunResmi" />
                  </ItemTemplate>
               </asp:TemplateField>
            </Fields>
         </asp:DetailsView>
         <asp:SqlDataSource ID="dsProductDetails" runat="server"
ConnectionString="<%$ ConnectionStrings:AdvConStr %>" SelectCommand="SELECT
Production.Product.ProductID, Production.Product.Name,
Production.Product.ProductNumber, Production.Product.SafetyStockLevel,
Production.Product.ReorderPoint, Production.Product.ListPrice,
Production.Product.StandardCost, Production.ProductProductPhoto.ProductPhotoID
FROM Production.Product INNER JOIN Production.ProductProductPhoto ON
Production.Product.ProductID =Production.ProductProductPhoto.ProductIDWHERE
(Production.Product.ProductID = @PrdId)">
            <SelectParameters>
               <asp:QueryStringParameter DefaultValue="1" Name="PrdId"
QueryStringField="PrdId" />
            </SelectParameters>
         </asp:SqlDataSource>
      </div>
   </form>
```

```
</body>
</html>
```

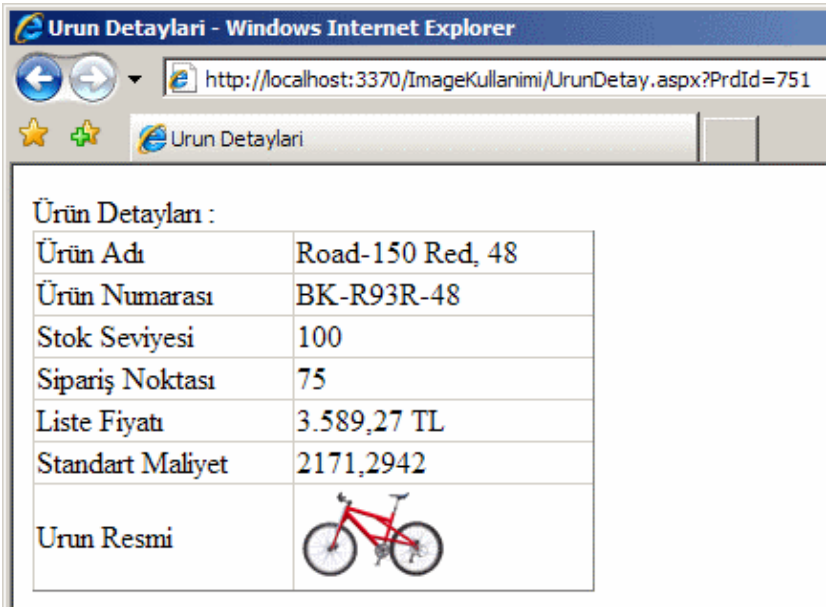
UrunDetay.aspx isimli web sayfasında dikkat edilmesi ve üzerinde durulması gereken bazı noktalar vardır. öncelikli olarak **SqlDataSource** kontrolünde kullanılan sorgu basit olarak aşağıdaki şekilde yer aldığı*(Query Builder ile elde edilmiştir)* gibi Product ve ProductProductPhoto tablolarının **join** ile birleştirilmiş bir halidir ve ProductId değerinin **where** ifadesinde ele almaktadır. Nitekim ürün resminin ProductPhoto tablosundan tedariki için ProductPhotoID değerinin bilinmesi gerekmektedir. Bu nedenle Product ve ProductProductPhoto tabloarı Join ile birleştirilmiştir.



Diğer taraftan **DetailsView** kontrolü içerisindede resmin gösterilebilmesi için bir **TemplateField** kullanılmış ve **ItemTemplate** şablonu içerisinde **img** elementi aşağıdaki gibi kullanılmıştır.

```
<asp:TemplateField HeaderText="Urun Resmi">
                <ItemTemplate>
                    <img alt="ürün Resmi"
runat="server" src='<%#"ResimGoster.aspx?ResimID="+DataBinder.Eval(Container.
DataItem, "ProductPhotoID") %>' id="urunResmi" />
                </ItemTemplate>
            </asp:TemplateField>
```

Burada dikkat edilmesi gereken en önemli nokta **src niteliğine(attribute)** değer atamasının nasıl yapıldığıdır. **DataBinder** sınıfının **Eval** metodunu kullanarak o anki satırın içerisinde yer alan ProdcutPhotoId değeri ResimGoster.aspx sayfasına ResimID adlı parametre ile gönderilmektedir. Buda yazımızın başında tasarladığımız ResimGoster sayfasının çağırılması ve bir resim içeriğinin elde edilerek buradaki img kontrolü içerisinde gösterilmesi anlamına gelmektedir. Sonuç itibariyle UrunDetay.aspx sayfası test edildiğinde aşağıdakine benzer bir ekran çıktısı ile karşılaşılacaktır.



Buraya kadar yaptıklarımızı özetleyecek olursak eğer, binary olarak tutulan resimlerin gösterilmesi için izlenebilecek yollardan birisinin adımları aşağıdaki gibi olacaktır.

- İlk olarak resim içeriğini **binary** olarak tarayıcıya basabilecek bir aspx sayfası tasarlanır.
- Sayfanın amacı gereği aspx kaynağında(Source) sadece **Page** direktifi bırakılır ve diğer içerik silinir. Bu zorunlu değildir. Ancak tavsiye edilen yoldur.
- Sayfanın **Page_Load** olay metodu kodlanır.
- Page_Load olay metodunda gösterilmek istenen resme ait satırın bulunabilmesi için **QueryString**' den yararlanılabilir.
- Resme ait binary içeriğin kod tarafında **byte dizisi(byte[])** şeklinde ele alınması sağlanır.
- Elde edilen byte dizisinin sayfaya resim olarak basılmasını sağlamak için önce **ContentType** özelliğinin değeri **image/gif** veya **image/jpeg** olarak belirlenir.
- Resmi yazdırmak içinse **BinaryWrite** metodu çağırılır.

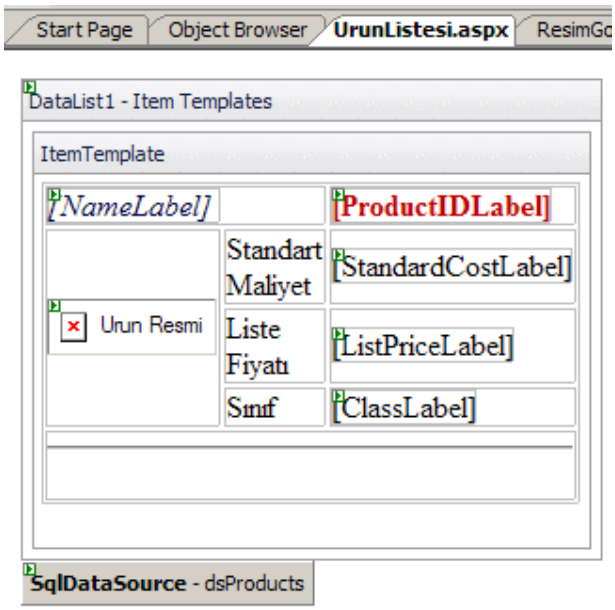
Buradaki örnek göz önüne alındığında istemci sayısının fazla olacağı düşünelecek olursa performansı arttırmak adına parametre bazlı olacak şekilde **ön bellekleme(Caching)** yapılması sağlanabilir. Böylece ResimGoster.aspx sayfasının sürekli olarak **Page_Load** kodlarını çalıştırmasının önüne geçilmiş olunur.

NOT : *Resimlerin çok sık değişmediği düşünülüyorsa ve SQL Server kullanılıyorsa tablo bağımlı bir ön belleklemede yapılabilir****(Sql Cache Dependency)****.*

Söz gelimi ResimGoster.aspx dosyasının içeriği aşağıdaki gibi değiştirilerek sayfanın çıktısının belirli bir süreliğine*(örneğin 60 saniye boyunca)* ön bellekte tutulması sağlanabilir.

<%@ Page Language="C#" AutoEventWireup="true" CodeFile="ResimGoster.aspx.cs" Inherits="ResimGoster" %>
<%@ OutputCache Duration="60" VaryByParam="ResimID" %>

Gelelim resimlerin işlenmesi ile ilgili diğer bir yaklaşıma. örnekte resmi gösteren **img** elementinin **src** niteliğinde bir url adresinden yararlanılmaktadır. Elbetteki bu işlem programatik olarak kod üzerindende geliştirilebilir. örneğin ürün bilgilerini bir DataList kontrolü üzerinde göstermek istediğimizi varsayalım. Bu kez resim alanlarının **DataList** kontrolü içerisindeki **img** elementinin**src** niteliğine bağlanmasını kod tarafından gerçekleştirmeye çalışacağız. Bu amaçla aşağıdaki gibi UrunListesi.aspx isimli bir web sayfası hazırlanarak işlemlere devam edilebilir.




```
<form id="form1" runat="server">
   <div>
      <asp:DataList ID="DataList1"
runat="server" DataSourceID="dsProducts" Width="600px">
         <ItemTemplate>
            <table>
               <tr>
                  <td colspan="2">
                     <asp:Label ID="NameLabel" runat="server" Font-Italic="True"
ForeColor="#000040" Text='<%# Eval("Name") %>'></asp:Label>
                  </td>
                  <td style="width: 100px; text-align: right">
                     <asp:Label ID="ProductIDLabel" runat="server" Font-Bold="True"
ForeColor="#C00000" Text='<%# Eval("ProductID") %>'></asp:Label>
                  </td>
               </tr>
```

```
            <tr>
               <td rowspan="3" style="width: 100px">
                  <img runat="server" id="urunResmi" alt="Urun Resmi" src="" />
               </td>
               <td style="width: 100px">Standart Maliyet</td>
               <td style="width: 100px">
                  <asp:Label ID="StandardCostLabel" runat="server" Text='<%#
Eval("StandardCost", "{0:C}") %>'></asp:Label>
               </td>
            </tr>
            <tr>
               <td style="width: 100px">Liste Fiyatı</td>
               <td style="width: 100px">
                  <asp:Label ID="ListPriceLabel" runat="server" Text='<%#
Eval("ListPrice", "{0:C}") %>'></asp:Label>
               </td>
            </tr>
            <tr>
               <td style="width: 100px">Sınıf</td>
               <td style="width: 100px">
                  <asp:Label ID="ClassLabel" runat="server" Text='<%# Eval("Class")
%>'></asp:Label>
               </td>
            </tr>
            <tr>
               <td colspan="3"><hr /></td>
            </tr>
         </table>
      </ItemTemplate>
   </asp:DataList>
   <asp:SqlDataSource ID="dsProducts" runat="server" ConnectionString="<%$
ConnectionStrings:AdvConStr %>" SelectCommand="SELECT Top 20 P.ProductID,
P.Name, P.StandardCost, P.ListPrice, PP.ProductPhotoID, P.Class FROM
Production.Product P INNER JOIN Production.ProductProductPhoto PP ON
P.ProductID = PP.ProductID WHERE P.ListPrice>1400">
   </asp:SqlDataSource>
  </div>
</form>
```

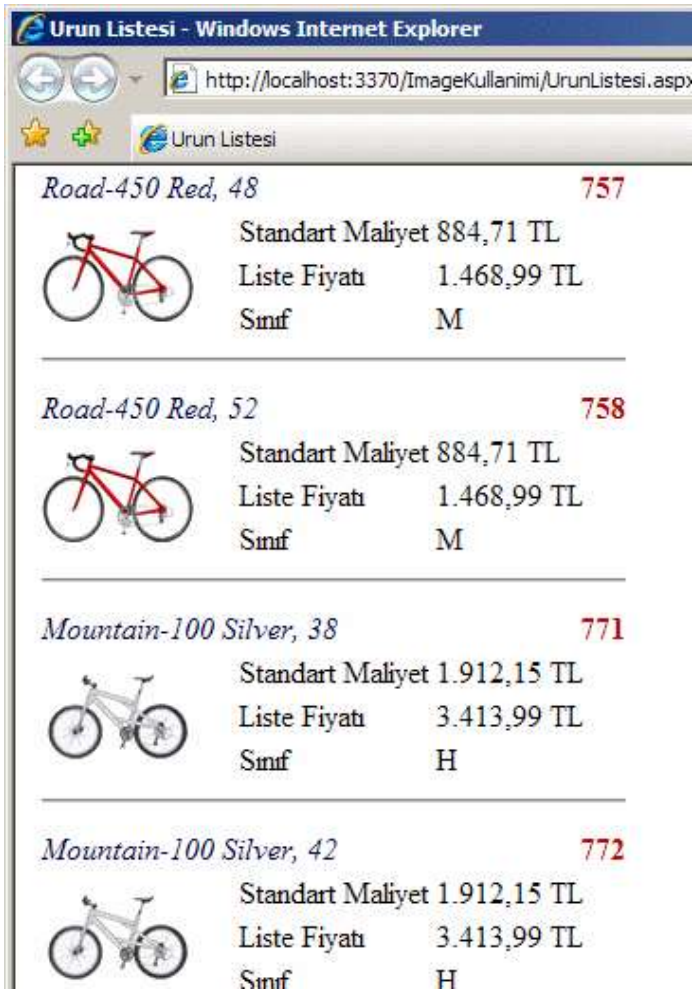
UrunListesi.aspx sayfasında yer alan **DataList** kontrolü Production şemasındaki Product ve ProductProductPhoto tablolarının birleşiminden oluşan sonuç kümesinden ilk 20 satırı göstermek üzere tasarlanmıştır ve hatta ListePrice alanına göre filtreleme eklenmiştir. Burada özellikle üzerinde durmamız gereken nokta, img elementidir. **ItemTemplate** üzerindeki tablo içerisine yerleştirilen **img** elementinin **src** niteliğini kod tarafında ele alabilmek için DataList

kontrolünün **ItemDataBound** olayından yararlanılabilir. Bu olay metodu içerisinde söz konusu img elementi bulunmalı ve src niteliğine o anki satırın ProductPhotoId değeri **QueryString** parametresi olarak aktarılmalıdır. O halde bu amaçla aşağıdaki kod parçasını yazmamız yeterli olacaktır.

```
protected void DataList1_ItemDataBound(object sender, DataListItemEventArgs e)
{
   if (e.Item.ItemType == ListItemType.Item
      || e.Item.ItemType == ListItemType.AlternatingItem)
   {
      // önce ProductPhotoId değeri bulunmalı
      string resimId = DataBinder.Eval(e.Item.DataItem,
"ProductPhotoId").ToString();
      // img elementi bulunmalı ve src niteliğinin değeri değiştirilmeli.
      HtmlImage resimElementi = (HtmlImage)e.Item.FindControl("urunResmi");
      resimElementi.Src = "~/ResimGoster.aspx?ResimId=" + resimId;
   }
}
```

ItemDataBound olayı **DataList** içerisindeki her bir satır için çalışacağından, işlemleri sadece **Item** ve **AlternatingItem** tipindeki satırlarda yapmakta fayda vardır. Bu amaçla**DataListItemEventArgs** tipinden olan e isimli parametrenin özelliklerinden faydalanılmaktadır. Sonrasında ise o anki satır ile gelen ProductPhotoId değerinin elde edilmesi gerekmektedir. Bu amaçlada**DataBinder** sınıfının **Eval** metodu ele alınmaktadır. İlk parametre ile o anki veri satırı yakalanmakta, ikinci parametre ilede söz konusu veri satırındaki **ProductPhotoId** değeri istenmektedir. Eval metodu geriye Object tipinden bir değer döndürdüğü için bilinçli olarak string tipine dönüştürülmüştür. Nitekim url katarında kullanılacak bilgi string' dir.

Bu işlemlerin ardından img kontrolünün elde edilmesi sağlanır. Bunun içinde **e.Item** üzerinden **FindControl** metodu çağırılmıştır. **FindControl** metodunun parametresi kontrolün id değerinin işaret etmektedir. Hatırlanacağı üzere img kontrolünün id niteliğine kaynak tarafında urunResmi adı verilmiştir. **img** kontrolü **HtmlImage** tipinden bir kontroldür ve FindControl metodu geriye **Control** tipinden bir referans döndürdüğünden sonuç referansı bilinçli olarak HtmlImage tipine dönüştürülmüştür. Son olarak elde edilen HtmlImage kontrolüne ait referans üzerinden Src niteliğinin değeri değiştirilir. Burada yapılan işlem tüm yazı boyunca üzerinde durduğumuz konudur. Uygulamayı bu haliyle çalıştırdığımızda aşağıdakine benzer bir ekran görüntüsü elde ederiz.



Bu makalemizde tablolarda **binary** olarak tutulan resim alanlarını web sayfalarında küçük bir hile ile nasıl işleyebileceğimizi incelemeye çalıştık. Görsellik hemen hemen tüm uygulamalarda önemli bir faktör olduğundan resim alanlarının bu şekilde ele alınıyor olmasını bilmek önemli bir avantaj sağlamaktadır. Kullanılan teknikte önemli olan nokta binary alanın içeriğini tek başına ele alıp resim formatında çıktı veren bir sayfanın var olmasıdır. Ayrıca resmi gösteren kontrolün basit bir **img** bileşeni olduğuna ve **src** niteliğinin önemine dikkat edilmelidir. Bu sayfanın çıktısı örneklerden de görüldüğü gibi pek çok farklı biçimde kullanılabilir ve son kullanıcıya görsel olarak daha doyurucu bir içerik sağlanabilir. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

örnek Uygulama için Tıklayın

Asp.Net için Etkili Hata Yönetimi (2007-07-18T20:45:00)

asp.net temelleri,

Uzun süredir Windows Communication Foundation ile ilgili yazılar yayınlıyoruz. Sanıyorumki biraz hava değişimine ihtiyacımız olacak. Bu nedenle bu haftaki yazımızda biraz daha hafif ama önemli olan bir konu üzerinde durmaya çalışacağız. Web uygulamalarında sunucu taraflı hata yönetimi (Server Side State Management). .Net ortamında hataların ele alınmasında kullanılan en bilinen **yol** **try...catch...finally** bloklarıdır. Ne varki uygulama ortamları çeşitlilik göstermektedir. Bir sınıf kütüphanesi içerisinde yapılan hata kontrolü ile dağıtık mimari uygulamaları**(distributed applications)** içerisinde yapılan hata yönetimi farklıdır.*(örneğin WCF içerisindeki Fault Management konusunu hatırlayalım)* Bu sebepten Asp.Net uygulamalarındada farklı bir yaklaşımı ele almak gerekmektedir. Web uygulamalarında oluşan hatalar sonucu çok hoş olmayan hata ekranları ile karşılaşmak mümkün olabilmektedir. Ancak hatalar kontrollü bir şekilde yönetilebilirlerse, son kullanıcıyı bilgilendirebilecek şekilde mesajlar verilip hataların düzeltilmesi yönünde daha sağlam ve güçlü adımlar atılabilir. Bu aynı zamanda uygulamanın tutarlılığı ve güvenilirliği açısındanda önemlidir.

Asp.Net ortamı, hataların yönetimi amacıyla **istisna(Exception)** tiplerini ve hataları yakalayıcı **olay(event)** metodların göz önüne alır. Söz konusu hata yönetimi **metod seviyesinde(Method Level)**, **sayfa seviyesinde(Page Level)** ve **uygulama seviyesinde(Application Level)** gerçekleştirilebilir. Aşağıdaki tabloda söz konusu seviyeler ve aralarındaki temel farklar vurgulanmaya çalışılmaktadır.

Asp.Net Hata Yönetim Seviyeleri (Error Management Levels)	
Metod Seviyesinde	**Sayfa Seviyesinde**
• Toparlanabilir veya bir başka deyişle kurtarılabilir hatalar çoğunlukla metod seviyesinde ele alınır. • Eğer olası hatalar toparlanamayacak cinsten ise bir üst seviyeye yönlendirilir.	• Bir sayfa ile ilgili tüm hataların tek bir merkezd yönetilebilmesi sağlanır. • Olası hatalar sonrasında kullanıcılar çoğunlukla özel sayfalara yönlendirilir. • Bu seviyede sayfaların **Page_Error** olay metod ele alınır. • Hata sayfasına yönlendirilmeden önce sayfa ile ilgili log bilgisi yazdırma, fiziki dosyalara bilgi atma veya adminlere mail gönderme gibi işleml yapılabilir.

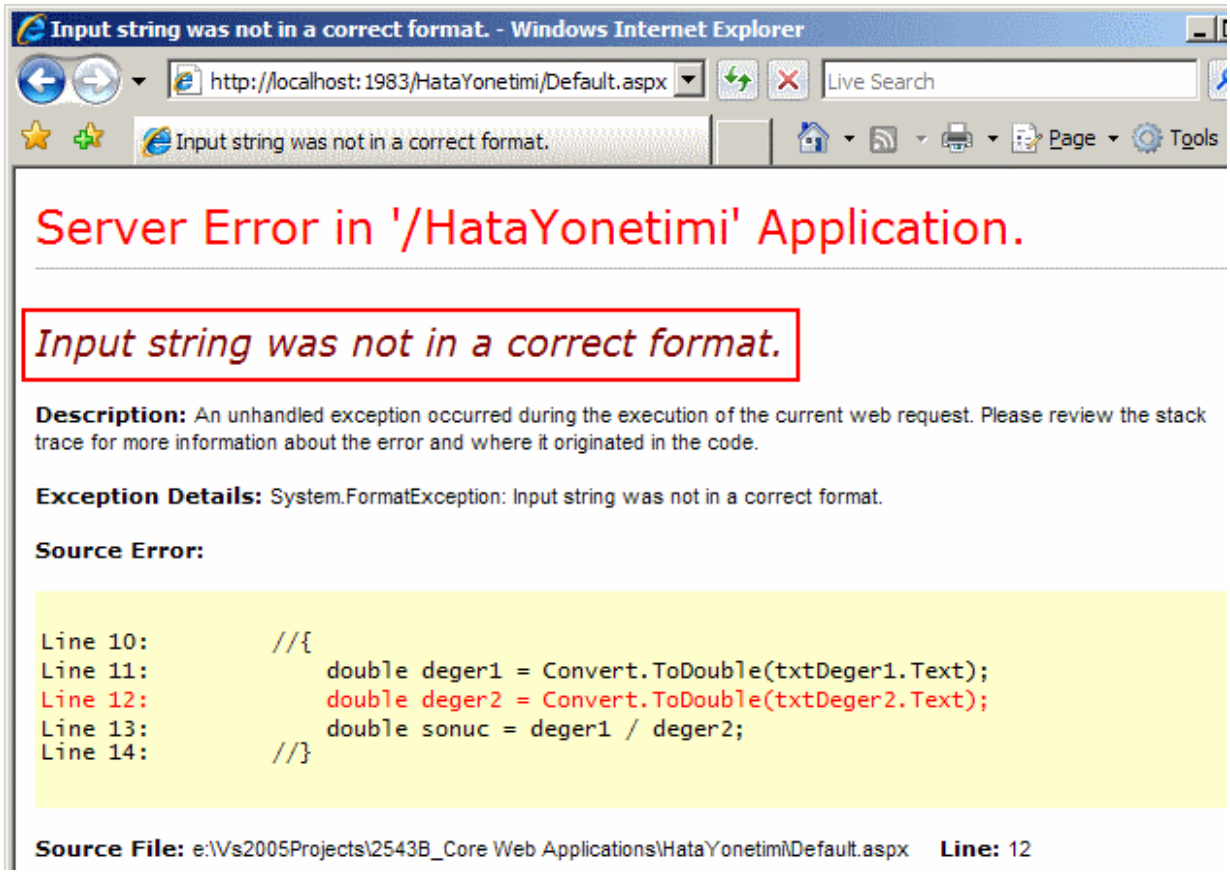
Metod seviyesinden, uygulama seviyesine doğru çıkıldıkça hataların merkezi olarak yönetilmesi ve tek bir merkezden ele alınması dahada kolaylaşmakta ancak, detay bilgilerinden gittikçe uzaklaşılmaktadır. Nitekim bir metod içerisinde meydana gelecek bir hata ile ilişkili yakalanan detayın, sayfa veya uygulama seviyesine aktarılmadığı sürece merkezi olarak ayrıştırılması zor olmaktadır.

Şimdi gelin bu seviyeleri örnekler yardımıyla incelemeye çalışalım. Metod seviyesinde hata yönetiminde **try...catch...finally** blokları büyük önem arz etmektedir. Ancak bu bloklar istenirse try...catch veya try...finally şeklindede yazılabilir. çok doğal olarak bunladan hangisinin kullanılacağının kararını vermek için bazı vakkaların göz önüne alınması gerekmektedir. İlk olarak basit bir örnek ile başlayalım. Bu amaçla kullanıcının bölme işlemi yaptığı aşağıdaki gibi bir aspx sayfası olduğunu göz önüne alabiliriz.

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

   protected void Hesapla_Click(object sender, EventArgs e)
   {
      double deger1 = Convert.ToDouble(txtDeger1.Text);
      double deger2 = Convert.ToDouble(txtDeger2.Text);
      double sonuc = deger1 / deger2;
   }

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>Hata Yonetimi (Metod Seviyesinde)</title>
   </head>
   <body>
      <form id="form1" runat="server">
         <div>
            Birinci Değer :
            <asp:TextBox ID="txtDeger1" runat="server"></asp:TextBox> <br />
            İkinci Değer :
            <asp:TextBox ID="txtDeger2" runat="server"></asp:TextBox>
            <br />
            <asp:Button ID="btnHesapla" runat="server" Text="Hesapla"
OnClick="Hesapla_Click" />
         </div>
      </form>
   </body>
</html>
```

öncelikle hata kontrolü yapmadan sayfayı ele almaya çalışacağız. Bu nedenle örnek olarak ikinci kutucuğu boş bırakıp Hesapla isimli düğmeye basıyoruz. Sonuç olarak kullanıcı

açısından pekte hoş olmayacak aşağıdaki ekran görüntüsü ile karşılaşırız. *(Elbette geliştirme aşamasında bu mesajlar developer açısından daha kıymetli olabilir ;)*

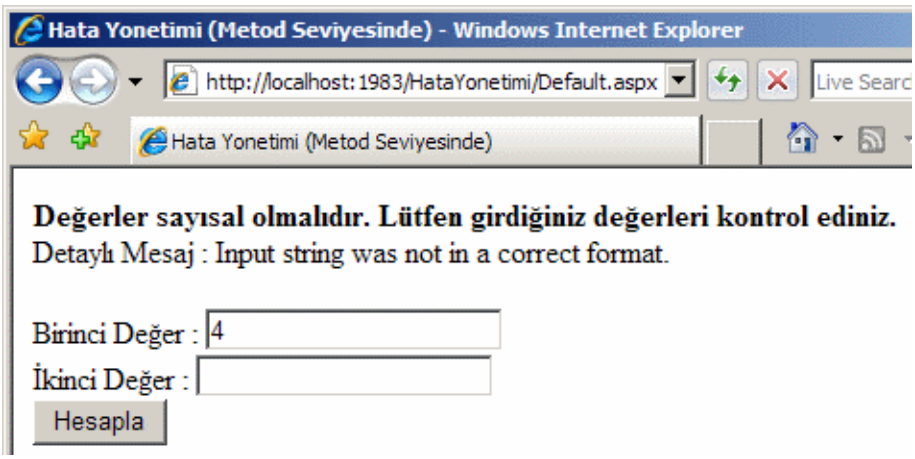


Dolayısıyla try...catch blokları yardımıyla Hesapla_Click isimli olay metoduna ait kodların aşağıdaki hale getirilmesi daha doğru olacaktır.

```
protected void Hesapla_Click(object sender, EventArgs e)
{
   try
   {
      double deger1 = Convert.ToDouble(txtDeger1.Text);
      double deger2 = Convert.ToDouble(txtDeger2.Text);
      double sonuc = deger1 / deger2;
   }
   catch (FormatException err)
   {
      Response.Write("<b>Değerler sayısal olmalıdır. Lütfen girdiğiniz değerleri kontrol
ediniz.</b><br/>Detaylı Mesaj : " + err.Message);
   }
   catch (Exception err)
   {
      Response.Write("<b>Beklenmeyen bir hata oluştu.</b><br/>Detaylı Mesaj : " +
```

```
err.Message);
   }
}
```

Bunun sonucunda aynı hata tekrar edilmeye çalışılırsa bu sefer mantıklı bir ekran ile karşılaşılacak ve kullanıcı hata ile ilişkili olarak daha doğru bir şekilde bilgilendirilebilecektir.



Elbette metod seviyesinde hata yönetimi adına dikkat edilmesi gereken bazı vakkalarda vardır. Eğer oluşan hataların kurtarılabilme(toparlanabilme) ihtimali varsa **try...catch** blokları döngüler (while, for gibi) içerisinde ele alınabilir. Böylece tekrar sayısına göre istenen rutin bir kaç kez üst üste denenebilir. Diğer taraftan oluşan istisnalar ile ilişikili olarak ekstradan verilebilecek yada kullanılabilecek bilgiler varsa bunların bir üst seviyede (**sayfa seviyesinde-page level**) ele alınması için **catch** bloğu içerisinde **throw** anahtar kelimesine başvurulabilir. Burada özellikle **Exception**sınıfının aşırı yüklenmiş**(overload)** yapıcı**(constructor)** metodlarından faydalanılmaktadır.

Diğer bir vakka metod içinde kullanılan dış kaynakları ele alır. örneğin ilgili rutinler içerisinde kaynak temizlenmesi gerekiyorsa *(bağlantıların veya dosyaların kapatılması, yönetimsiz-unmanaged nesnelerin serbest bırakılması gibi)* **finally** bloklarını kullanmak doğru olacaktır. Burada finally bloklarının kullanılması şart değildir.
Nitekim **using** blokları yardımıylada, **IDisposable** arayüzünü uygulayan tipler için blok sonunda **Dispose** çağrıları gerçekleştirilebilir. Son olarak olası hatalar ilgili metod içerisinde ele alınamıyorsa metodu çağıran yerde yakalanmalıdır.

Görüldüğü üzere vakkaların sayısı ve metod içerisindeki hata yönetimi çeşitli şekillerde yapılabilmektedir. Bu sebepten karar verirken aşağıdaki gibi tablodan faydalanmakta yarar vardır.

öneri	Kaynak temizlemesi gerekiyor mu?	Olası hata var mı?
Hiç bir kontrole gerek yok	hayır	yok

	hayır	var
try...finally	evet	yok
	evet	var
try...catch	hayır	var
try...catch...finally	evet	evet

örneğin kaynak temizlenmesi gerekiyorsa, olası hatalar var ise ve hatta olası hatalara eklenebilecek ekstra bilgiler var ise **try..catch...finally** bloklarını kullanmak daha mantıklıdır. Ne varki olası hatalarda, hata mesajına ilave bilgiler eklemek catch blokları içerisinde **throw** kullanmak ile mümkün olabilir. çünkü amaç, bu hatayı ele alan bir üst seviyeye bilgi göndermektir. Bunun ele alınabileceği en güzel yer sayfa seviyesidir **(Page Level)**. öyleyse sayfa seviyesinde hata yönetiminin nasıl yapılacağını inceleyerek devam edelim.

özellikle belirli bir sayfada meydana gelebilecek tüm hataların tek bir merkezden kontrolünün sağlanması gerektiği durumlarda sayfa seviyesinde hata yönetimi gerçekleştirilebilir. Bu teknikte önemli olan nokta, **Page_Error** olay metodunun etkin bir şekilde kullanılmasıdır. Aslında sayfa seviyesinde hatalar ele alınırken izlenen basit bir yol vardır. Aşağıdaki tabloda bu yol gösterilmektedir.

Sayfa Seviyesinde Hata Kontrolü için Tavsiye Edilen Yol	
Madde 0	Bir hata sayfası tasarlanır. :)
Madde 1	Sayfaya **Page_Error** olay metodu eklenir.
Madde 2	Sayfanın **ErrorPage** özelliğine hata sayfasının **Url** bilgisi **Page** direktifi
Madde 3	**Page_Error** olay metodu içerisinde **Server** sınıfının static **GetLastErro**
Madde 4	İstenirse **ErrorPage** özelliğine burada değer ve hatta **querystring** yardın taşınmasıda sağlanmış olur.
Madde 4.5	Gerekirse bu aşamada loglama (özellikle sistemdeki event loglara bilgi y işlemler yapılabilir.

Page_Error isimli olay metodu sayfa seviyesinde ele alınır. Normal şartlarda sayfada ele alınmayan bir hata oluştuğunda bu metod otomatik olarak çağırılacaktır. Biz bu metod içerisinde yönlendirmeler yaparak kullanıcıları daha akıllı hata bilgilendirme sayfalarına yönlendirebilir ve loglama gibi işlemleri gerçekleştirebiliriz. **Page_Error** olay metodu içerisinden ilgili hata sayfasına yönlendirme yaparken Page sınıfının ErrorPage özelliğine değer atamak gerekebilir. Bu daha çok **querystring** yardımıyla hata sayfasına ekstra bilgi gönderileceği durumlarda ele alınır. Aksi durumlarda metod içerisinde değilde Page direktifinde bu özelliğin değerinin belirlenmesi yeterlidir. Ancak burada dikkat edilmesi gereken bir nokta vardır. Eğer metod içerisinde ErrorPage özelliğine hata sayfasını atarken

querystring kullanılmassa Asp.Net, çalışma zamanında **aspxerrorpath** isimli bir anahtarı ve değerini otomatik olarak ekleyecektir. Ki buda hatanın oluştuğu sayfanın yakalanabilmesi ve belkide dinamik bir linkin üretilerek tekrar geri gidilebilmesinide sağlayacaktır.

Şimdi yukarıdaki örneğimizi sayfa seviyesinde ele almaya çalışalım. Madde 0' da değindiğimiz gibi öncelikli olarak bir hata sayfası tasarlamakta fayda var. Bu hata sayfası aşağıdaki gibi tasarlanabilir.

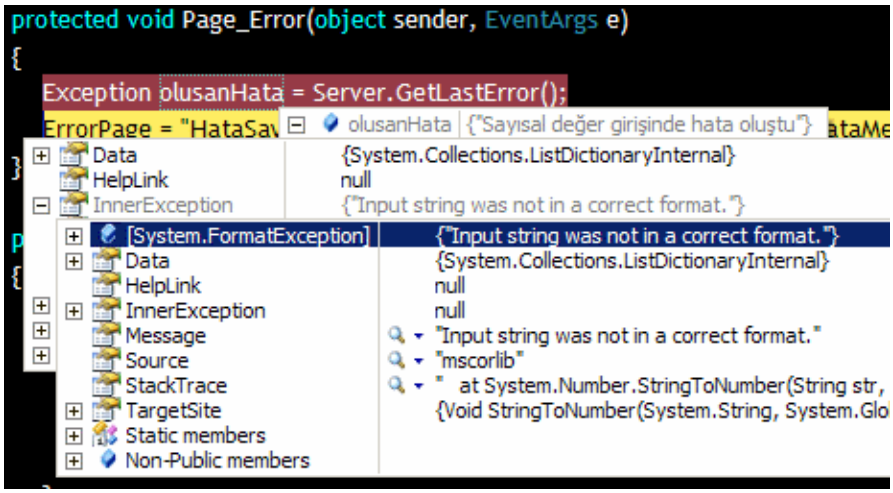
HataSayfasi.aspx;

```
<%@ Page Language="C#" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">

<script runat="server">

   protected void Page_Load(object sender, EventArgs e)
   {
      string ekBilgi = Request.QueryString["EkBilgi"];
      string sayfa=Request.QueryString["Sayfa"];
      string hataMesaji = Request.QueryString["HataMesaji"];
      Response.Write("<b>Hata sayfası : </b>" + sayfa+"<br/>");
      Response.Write("<b>Ek bilgi : </b>" + ekBilgi + "<br/>");
      Response.Write("<b>Hata Mesajı : </b>" + hataMesaji);
   }

</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
   <head runat="server">
      <title>Hata Sayfası</title>
   </head>
   <body>
      <form id="form1" runat="server">
         <div>
         </div>
      </form>
   </body>
</html>
```

Default.aspx sayfasındaki kodları ise aşağıdaki gibi değiştirebiliriz.

```
protected void Page_Error(object sender, EventArgs e)
{
   Exception olusanHata = Server.GetLastError();
   ErrorPage = "HataSayfasi.aspx?EkBilgi=" + olusanHata.Message + "&HataMesaji="
+ olusanHata.InnerException.Message + "&Sayfa="+Page.AppRelativeVirtualPath;
}

protected void Hesapla_Click(object sender, EventArgs e)
{
   try
   {
      double deger1 = Convert.ToDouble(txtDeger1.Text);
      double deger2 = Convert.ToDouble(txtDeger2.Text);
      double sonuc = deger1 / deger2;
   }
   catch (Exception excp)
   {
      throw new Exception("Sayısal değer girişinde hata oluştu", excp);
   }
}
```

Hesapla_Click olay metodu içerisinde yer alan **catch** bloğunda **throw** anahtar sözcüğü kullanılarak bir **Exception** nesnesi daha fırlatılmıştır. Bu istisna nesnesinin yakalanacağı yer sayfanın **Page_Error** isimli olay metodudur. Bu metod içerisinde, sayfada oluşan son hatayı yakalayabilmek için **Server** sınıfının **GetLastError** metodu kullanılmıştır. Dikkat edilmesi gereken noktalardan birisi, throw ile fırlatılan Exception nesnesi örneklenirken ilk parametreye örnek bir ekstra veri konulmasıdır. Eklenen bu bilgi GetLastError ile yakalanan **Exception** nesne örneğinin **Message**özelliği ile elde edilebilir. Bu durumda orjinal istisna mesajınının nereden alınabileceği bir soru işaretidir.
Cevap **InnerException** özelliğidir. InnerException özelliği ile fırlatılan asıl Exception nesne örneği yakalanabilir. Bunu çalışma zamanında test ettiğimizde aşağıdaki ekran görüntüsünde olduğu gibi **FormatException** tipinin **InnerException** özelliğinde saklandığını görebiliriz.



Son olarak **ErrorPage** özelliği ile hata sayfasına gerekli yönlendirme yapılmaktadır. Elbette ErrorPage kullanılmak zorunda değildir. Bunun yerine **Server** sınıfının **Transfer** metodu veya**Response** sınıfının **Redirect** metodlarının kullanımıda tercih edilebilir. özellikle Server sınıfının Transfer metodu gereksiz roundtrip' lerin önüne geçilmesini sağlamakta ama url satırına bakıldığında halen daha aynı sayfada olunduğu izlenimini vermektedir.

örnek geliştirilirken dikkat edilmesi gereken bir nokta vardır. Eğer örneği aynı makine üzerinde **(localhost)** test ediyorsak beklediğimiz yönlendirme sayfasına gidemediğimizi hatta eski sarı ekranın *(orjinal hata mesajının basıldığı sayfadan bahsediyoruz)* üretildiğini görürüz. Bunun nedeni web uygulamasının özel hata modunun aktif olmayışıdır. Bir başka deyişle **web.config** dosyasında yer alan **customErrors** elementinin mode niteliğine **On** değerinin verilmesi gerekir.

NOT : *Aslında mode özelliğinin 3 farklı değeri vardır.* ***RemoteOnly****,* ***On*** *ve* ***Off****. RemoteOnly modu aktif iken özel hata sayfalarını sadece istemciler görebilir. On modunda hem istemciler hemde localhost kullanıcısı özel hata sayfalarını görebilir. Biz örneklerimizdeki sayfalarımızı aynı makine üzerinden test ettiğimiz için bu modu* ***On*** *olarak belirledik. Gerçek bir uygulama ortamına çıkıldığında On yerine RemoteOnly kullanılması tavsiye edilir.*

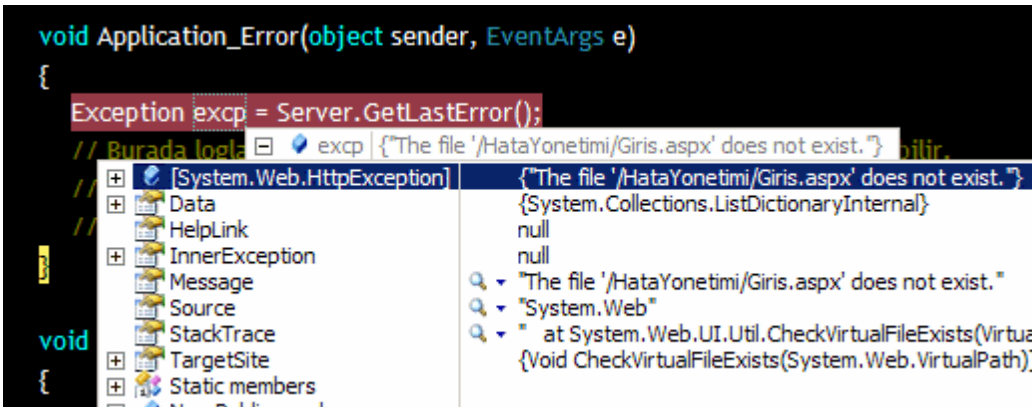
<customErrors **mode="On"**/>

customErrors elementi içerisine **error** isimli alt elementlerde konulabilir. Bu element sayesinde sunucu seviyesinde meydana gelen hatalar var ise bunların sonucunda özel hata sayfalarına yönlendirmeler yapılabilir. Söz gelimi aşağıdaki bildirimleri ele alalım.

```
<customErrors mode="On">
   <error statusCode="404" redirect="SayfaYok.aspx"/>
</customErrors>
```

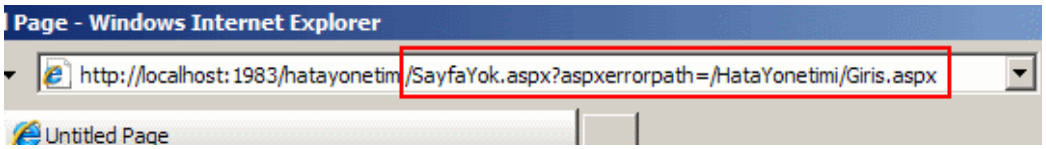
Buna göre sitede olmayan bir sayfa talep edilirse kullanıcılar SayfaYok.aspx' e yönlendirilirer.

NOT : *Yanlız burada dikkat edilmesi gereken bir nokta vardır. Sonradan ele alacağımız gibi* ***global.asax*** *dosyasındaki* ***Application_Error*** *olayı yazılmışsa, uygulama SayfaYok.aspx' e yönlendirilmeden önce buradaki olay metoduna uğrayacaktır ki burada bir yönlendirme yapıyorsak SayfaYok.aspx yerine oraya gidilebilir. Hatta Server.ClearError metodu kullanılmışsa hata sayfasına gidilmeyedebilir. Buda sistemin istediğimiz şekilde çalışmaması anlamına gelmektedir. Gerçi Application_Error içerisinde oluşan hata örneğin sayfa yok hatası yinede yakalanabilir. örneğin Debug modda bu aşağıdaki şekilde olduğu giri görünecektir.*



Dolayısıyla bu noktalara dikkat etmekte fayda vardır.

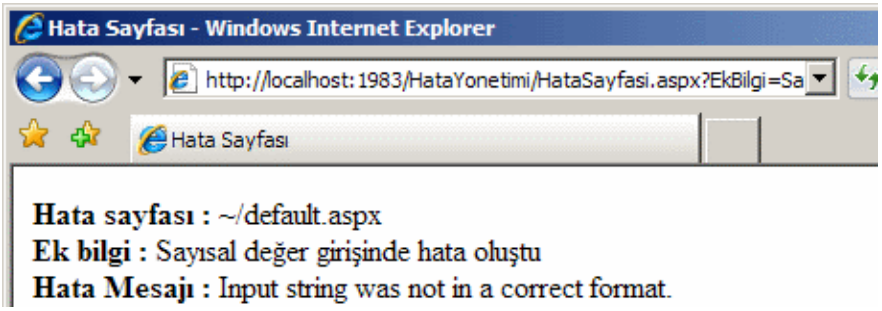
örneğin şu aşamada iken web uygulamasını çalıştırdıkdan sonra Giris.aspx isimli yazmadığımız bir sayfayı talep edersek aşağıdaki ekran görüntüsü ile karşılaşırır.



özellikle SayfaYok.aspx' ten sonra gelen querystring parametresine dikkat edelim. **aspxerrorpath** ile gelen değer alınıp kullanıcıya daha anlamlı bir hata sayfası gösterilebilir. Biz tekrardan konumuza geri dönelim ve **customErrors** elementini aşağıdaki haliyle bırakalım.

<customErrors **mode="On"**/>

Bu değişiklikten sonra uygulama çalışma zamanında test edilirse default.aspx sayfasında hata oluştuktan sonra HataSayfasi.aspx' e gidildiği görülebilir. Tarayıcı penceresindeki url satırına dikkat edilecek olursa querystring parametreleri ve değerleride başarılı bir şekilde aktarılmıştır.

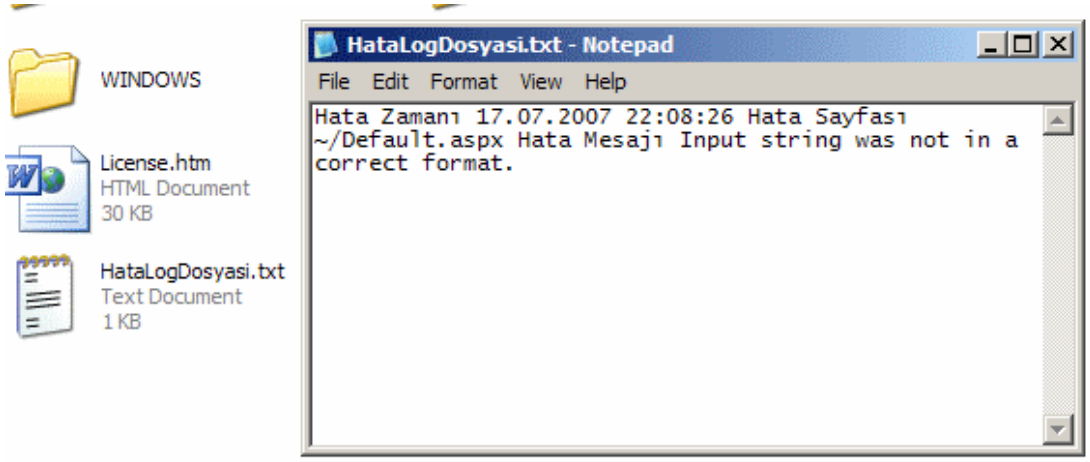


Şunu itiraf etmeliyim ki sarı ekranın görüntüsü buradakinden daha güzeldir. Dolayısıyla hata sayfaları hazırlanırken biraz daha özenilmeli, gerektiğinde projenin sahibi olan şirket standartlarına uygun olaraktan tasarlanmalı ve zengin bir bilgi sunacak hale getirilmelidir.

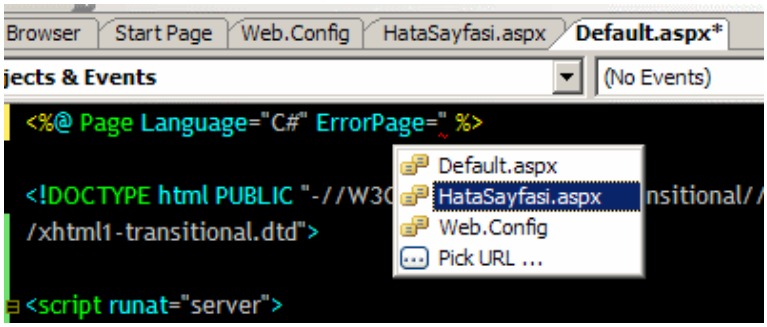
Oluşan hatalara ilişkin kullanıcılara bilgi verilmesi dışında, siteyi tasarlayan veya yönetenlerinde bilgilendirilmesi gerekebilir. Bu bilgilendirme farklı şekillerde yapılabilir. örneğin var olan işletim sistemi loglarına bilgi yazılabilir. örneğin Application loglarına. Ya da daha basit olarak fiziki bir dosyaya hatalar ile ilişkili bazı bilgiler gönderilebilir. Hatta gerektiği yerlerde çok kritik hatalar söz konusu ise ilgili kişilere mail bile gönderilebilir. Söz gelimi aşağıdaki kod parçası ile **Page_Error** metodu içerisinde, oluşan son hataya ait bilgi fiziki bir dosyaya eklenmektedir.

```
protected void Page_Error(object sender, EventArgs e)
{
   Exception olusanHata = Server.GetLastError();
   ErrorPage = "HataSayfasi.aspx?EkBilgi=" + olusanHata.Message + "&HataMesaji=" +
olusanHata.InnerException.Message + "&Sayfa="+Page.AppRelativeVirtualPath;

   using (FileStream stream = new FileStream("C:\\HataLogDosyasi.txt",
FileMode.Append, FileAccess.Write))
   {
      StreamWriter writer = new StreamWriter(stream);
      writer.WriteLine("Hata Zamanı " + DateTime.Now.ToString() + " Hata Sayfası
" + Page.AppRelativeVirtualPath + " Hata Mesajı " +
olusanHata.InnerException.Message);
      writer.Close();
   }
}
```

Burada basit olarak **FileStream** ve **StreamWriter** tiplerinden yararlanılarak hata bilgileri C: klasörü altındaki bir text dosyasına yazdırılmaktadır. Sonuç olarak uygulama test edildiğinden oluşturulan hata sonrasında ilgili dosyaya aşağıdaki ekran görüntüsünde olduğu gibi bazı bilgiler eklenecektir.



Page_Error metodu içerisinde hata sayfasına herhangibir şekilde yönlendirme yapılmamasına rağmen gidilmektedir. Bu metodun bir özelliğidir. Metod sonuna gelindiğinde, **ErrorPage** ile belirlenmiş sayfaya otomatik olarak gidilir. ErrorPage değerinin programatik olarak belirlenmesi haricinde **Page** direktifi içerisinde ayarlanabileceğini daha önceden söylemiştik. Bu aşağıdaki ekran görüntüsünde olduğu gibi düzenlenebilir.



Elbette metod içerisinde ErrorPage özelliği belirtilmişse **Page** direktifinde yapılan tanımlama geçersiz sayılacaktır.

Gelelim uygulama seviyesinde hata yönetimine. Bu durumda web uygulamasında meydana gelecek hataların ele alınabileceği bir merkez söz konusudur. Söz konusu merkez global.asax dosyası içerisinde yer alan **Application_Error** isimli olay metodudur. Bildiğiniz gibi **global.asax** dosyasında, uygulama genelini ilgilendiren bazı olay metodları yer almaktadır. örneğin uygulama çalışmaya başladığında devreye giren **Application_Start**, sonlandığında çağrılan **Application_End** yada kullanıcıların açtıkları **oturumlarda(Session)** devreye giren **Session_Start** gibi. Dolayısıyla ilk yapılması gereken işlem web sitesine, eğer yok ise bir global.asax dosyası eklemek olacaktır. Sayfa seviyesindeki hata yönetiminde olduğu gibi, uygulama seviyesinde yapılacak hata yönetimi içinde tavsiye edilen bir yol haritası vardır ve aşağıdaki tabloda olduğu gibidir.

Uygulama Seviyesinde Hata Kontrolü için Tavsiye Edilen Yol	
Madde 0	Bir hata sayfası tasarlanır. :)
Madde 1	Sayfalara **Page_Error** olay metodları eklenir.
Madde 2	**Page_Error** olay metodlarında, son olarak elde edilen **istisna(Exception** ortama **fırlatılır(throw)**.
Madde 3	global.asax dosyasında yer alan **Application_Error** olay metodu kodlan
Madde 3.5	Gerekirse bu aşamada loglama (özellikle sistemdeki event loglara bilgi y işlemler yapılabilir. *(Sistem loglarına yazma sırasında dikkat edilmesi ge* ***için)*** *kullanıcısının* ***Application***, ***System*** *ve* ***Security*** *loglarına yazma hak olarak* ***Application*** *loglarına yazma hakkı varken System ve Security logl yazma işlemleri sırasında dikkate alınması gerekebilir.)*
Madde 4	Kullanıcı hata sayfasına yönlendirilmeden önce **Server** sınıfının **ClearE**
Madde 5	**Server.Transfer** metodu ile hata sayfasına yönlendirme yapılır.

Buradaki maddelerde dikkat çekici noktalardan biriside son hatanın sayfalara ait Page_Error olay metodları içerisinde tekrardan fırlatılıyor olmasıdır. Bu bir anlamda hatanın bir üst seviyeye aktarılmasıdır. Diğer taraftan bir gerekliliktir. Nitekim, hata bilinçli olarak uygulama seviyesine gönderilmesse Asp.Net çalışma ortamı**(Asp.Net RunTime)**, hatanın ele alınması için**HttpUnhandledException** tipinden bir nesne örneği üretecektir. Biz hatayı kontrollü bir şekilde ele almak istiyorsak bilinçli bir şekilde fırlatma işlemini üstlenmeliyiz.

çok doğal olarak web uygulaması içerisinde birden fazla aspx sayfası olduğu düşünülecek olursa, hepsine bir **Page_Error** metodu eklemek ve kodlamak (en azından **GetLastError** ile elde edilen istisna nesnelerini fırlatmak) uğraştırıcı ve sabrımızı test edici olabilir. Burada nesne yönelimli mimarinin avantajlarından faydalanmak çok daha akılcı bir çözüm olacaktır. Bir başka deyişle tüm sayfaların türediği bir taban sayfa (base page) içerisindeki Page_Error metodu ele alınabilir.

NOT : *Alternatif bir yaklaşım olarak MasterPage kullanımı düşünülebilir. Her ne kadar MasterPage içerisine Page_Error isimli bir metod yazılabiliyor olsada, aslında metod seviyesinden throw ile exception fırlatıldığında bu metod herhangibir şekilde tetiklenmez. Eğer Application_Error olay metodu yazılmışsa doğrudan buraya düşülür ve****HttpUnhandledException*** *tipinden bir istisna nesnesi yakalanır. Bu sebepten bu tip bir durumda MasterPage içerisindeki Page_Error olay metodunun kullanımı şeklinde bir çözüm ne yazıkki söz konusu değildir.*

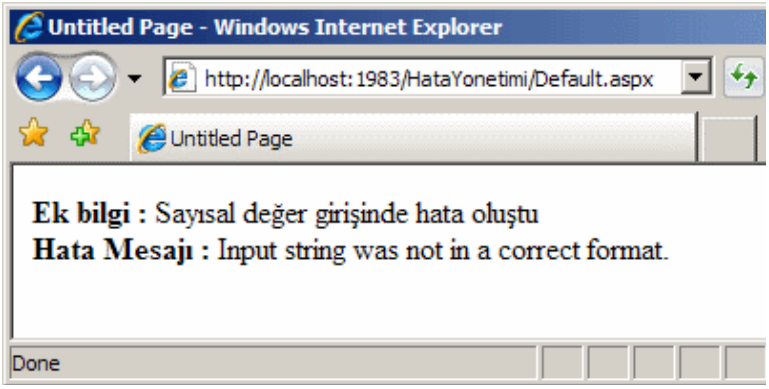
İlk olarak Application_Error olay metodunu nasıl ele alacağımıza bakalım. öncelikle aspx sayfalarındaki **Page_Error** olay metodlarından yine bir üst seviyeye hata fırlatmak gerekir. Bu nedenle örnek olarak default.aspx sayfasındaki **Page_Error** olay metodu aşağıdaki gibi değiştirilmelidir.

```
protected void Page_Error(object sender, EventArgs e)
{
   Exception olusanHata = Server.GetLastError();
   throw olusanHata;
}
```

Uygulama seviyesinde hata kontrolü yapılacağından ilgili olay kodunun **global application class** içerisinde yer alması gerekir. Bu nedenle bir adet **global.asax** dosyası web sitesine dahil edilmelidir. global.asax dosyasında **Application_Error** olay metodu içerisinde ise aşağıdakine benzer kodlamalar yapılmalıdır.

```
void Application_Error(object sender, EventArgs e)
{
   Exception excp = Server.GetLastError();
   // Burada loglama, dosyaya yazma, mail gönderme gibi işlemler yapılabilir.
   Server.ClearError();
   Server.Transfer("GenelHataSayfasi.aspx?EkBilgi="+excp.Message+"&HataMesaj
i="+excp.InnerException.Message);
}
```

Bu sefer global.asax dosyası içerisinden GenelHataSayfasi.aspx isimli web sayfasına querystring yardımıyla bilgi taşınmaktadır. GenelHataSayfasi.aspx içeriği, HataSayfasi.aspx' e benzemekle birlikte tek farkı hatanın meydana geldiği sayfa bilgisini ele almıyor oluşudur. Eğer uygulama bu haliyle test edilir ve yine Default.aspx içerisinde hata yaptırılırsa aşağıdaki ekran görüntüsü ile karşılaşılır.



Görüldüğü gibi default.aspx içerisindeki metodda oluşan hata catch bloğunda ek bilgi ile tekrar throw edilmiş ve bunu sayfanın Page_Error olay metodu yakalamıştır. Sonrasında ise Page_Error olay metodunda hata tekrar throw ile fırlatılmış ve bunun sonucunda Application_Error olay metoduna gidilebilmiştir. Aşağıdaki grafikte bu durumun Debug moddaki karşılığı gösterilmektedir.

önce Metod içi;

```
protected void Hesapla_Click(object sender, EventArgs e)
{
    try
    {
        double deger1 = Convert.ToDouble(txtDeger1.Text);
        double deger2 = Convert.ToDouble(txtDeger2.Text);
        double sonuc = deger1 / deger2;
    }
    catch (Exception excp)
    {
        throw new Exception("Sayısal değer girişinde hata oluştu", excp);
    }
```

1

Sonra Page_Error;

```
<script runat="server">

    protected void Page_Error(object sender, EventArgs e)
    {
        Exception olusanHata = Server.GetLastError();
        throw olusanHata;
    }
```

2

Son olarak Application_Error;

```
    void Application_Error(object sender, EventArgs e)
    {
        Exception excp = Server.GetLastError();
        // Burada loglama, dosyaya yazma, mail gönderme gibi işle
        Server.ClearError();
        Server.Transfer("GenelHataSayfasi.aspx?EkBilgi="+excp.Me
    }
```

3

çalışma sonrasında dikkati çeken noktalardan birisi tarayıcı penceresindeki url satırında Default.aspx' in görünüyor olmasıdır. Halbuki şu anda üzerinde bulunulan sayfa GenelHataYonetimi.aspx' dir. Bunun sebebi **Server.Transfer** metodudur. Bunun yerine Response.Redirect' te tercih edilebilir. Ama daha öncedende belirtildiği üzere Transfer metodu ile sunucuya doğru gidiş gelişler azaldığı için özellikle uygulama seviyesindeki hata yönetiminde tercih edilen bir tekniktir. İlgili hata sayfası dahada geliştirilebilir. örneğin hatanın oluştuğu sayfaya dönülmesi için gerekli bağlantı bilgileri hata mesajı ile birlikte taşınabilir. Bu ve dahası tamamen sizlerin hayal gücüne kalmaktadır.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde Asp.Net uygulamalarında etkin hata yönetimi adına metod, sayfa ve uygulama seviyesinde neler yapabileceğimizi

incelemeye çalıştık. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

örnek Uygulama için Tıklayın

WCF - Windows Tabanlı Doğrulama ve Yetkilendirme (2007-07-12T20:54:00)

wcf,

Servis yönelimli mimaride**(Service Oriented Architecture)**, dağıtık uygulamaların bir kısmı **intranet** tabanlı olaraktan Windows işletim sistemleri üzerinde konuşlandırılmış olarak tasarlanmaktadır. Bu sebepten ağ üzerinde tüm kullanıcıların daha kolay bir şekilde yönetildiği var olan ve bilinen doğrulama ve yetkilendirme materyallerinden yararlanmak sık olarak başvurulan tekniklerdendir. Bir başka deyişle kullanıcı hesaplarının**(Member accounts)** yönetimi hazır olan Windows işletim sistemi unsurları tarafından kolayca ele alınabilmektedir. Buda doğal olarak istemcilerin servis tarafında doğrulanması**(Authentication)** ve yetkilendirilmesi**(Authorization)** için hazır olan kaynakların kullanılabilmesi anlamına gelir.

Sonuç itibariyle servis ve istemci tarafı açısıdan geliştiricinin yükü biraz daha hafiflemektedir. **Windows Communication Foundation** uygulamalarındada intranet tabanlı sistemler için Windows tabanlı doğrulama ve yetkilendirme**(Windows Based Authentication and Authorization)** işlemlerini, bağlayıcı tip (binding type) bazında kolayca gerçekleştirebiliriz. Bu bölümde özellikle**basicHttpBinding** tipi yardımıyla bu işlemlerin nasıl geliştirilebileceğini adım adım incelemeye çalışacağız. Her zamanki gibi servis tarafından hizmete sunulacak olan **WCF Service Library'** sini geliştirmekle işe başlanabilir. Bu kütüphane içerisindeki söz konusu tipler(types) aşağıdaki gibi tasarlanmıştır.

IAritmetik arayüzü(interface);

```
using System;
using System.ServiceModel;

namespace CebirLib
{
   [ServiceContract(Name="AritmetikServisi",Namespace="http://www.bsenyurt.com/AritmetikServisi")]
   public interface IAritmetik
   {
      [OperationContract(Name="ToplamaOperasyonu")]
      int Topla(int x, int y);
   }
}
```

Aritmetik sınıfı(class);

```
using System;
using System.Threading;
using System.Security.Principal;

namespace CebirLib
{
   public class Aritmetik:IAritmetik
   {
      #region IAritmetik Members

      public int Topla(int x, int y)
      {
         IPrincipal principal = Thread.CurrentPrincipal;
         string dogrulamaTipi=principal.Identity.AuthenticationType;
         string dogrulandi = principal.Identity.IsAuthenticated ? "Dogrulandi" : "Dogrulanmadi";
         string ad=principal.Identity.Name;

         Console.WriteLine("Kullanıcı : " + ad + "\n" + "Doğrulama Tipi : " + dogrulamaTipi + "\n" + dogrulandi + "\n");
         return x + y;
      }

      #endregion
   }
}
```

Aritmetik sınıfı içerisinde yer alan Topla isimli metoda başlangıç olarak bazı kod satırlar eklenmiştir. **IPrincipal** arayüzüne ait principal isimli değişkene, **Thread** sınıfının

static**CurrentPrincipal** özelliği yardımıyla atanan değer aslında çalışma zamanında servise talepte bulunan istemci kimliğini işaret etmektedir. Bu özelliğin dönüş değerinden faydalanarak doğrulama tipini**(Authentication Type)**, kullanıcının adını**(Name)** ve hatta sonradanda görüleceği gibi kullanıcının hangi rolde olduğunun tespiti yapılabilmektedir. Daha çok Windows kullanıcıların rollerine**(Role)** bakılarak kod içerisinden yetkilendirme**(Authorization)** yapılmak istendiği durumlarda kullanılmaktadır.

Sırada servis ve istemci tarafındaki uygulamaların tasarlanması var. Her iki tarafıda olayları daha kolay irdelemek açısından birer Console uygulaması olarak tasarlamakta fayda vardır. Servis tarafındaki Console uygulaması, geliştirilen WCF Servis kütüphanesini(CebirLib) ve **System.ServiceModel.dll** assembly' ını referans etmektedir. Servis tarafına ait kodlar başlangıç için aşağıdaki gibidir.

```
using System;
using System.ServiceModel;
using CebirLib;

namespace ServerApp
{
   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(Aritmetik));
         host.Open();
         host.Closing += new EventHandler(host_Closing);
         host.Closed += new EventHandler(host_Closed);
         Console.WriteLine("Sunucu dinlemede...\n Kapatmak için bir tuşa basın...");
         Console.ReadLine();
         host.Close();
      }

      static void host_Closed(object sender, EventArgs e)
      {
         Console.WriteLine("Servis kapatıldı...Yine bekleriz...");
      }

      static void host_Closing(object sender, EventArgs e)
      {
         Console.WriteLine("Servis kapatılıyor. Lütfen bekleyiniz...");
      }
   }
}
```

Servis tarafında yer alan konfigurasyon dosyasının başlangıç ayarları ise aşağıdaki gibidir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="AritmetikBindingHttpCfg">
          <security mode="TransportCredentialOnly">
            <transport clientCredentialType="None" />
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <services>
      <service name="CebirLib.Aritmetik">
        <endpoint address="http://localhost:1600/AritmetikServisi"
binding="basicHttpBinding" bindingConfiguration="AritmetikBindingHttpCfg" na
me="AritmetikServiceHttpEndPoint" contract="CebirLib.IAritmetik" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Servis tarafında **basicHttpBinding** bağlayıcı tipi kullanılmaktadır. Üzerinde durulması gereken ilk noktalardan birisi güvenlik modunun(**security** elementinin **mode** niteliği yardımıyla)**TransportCredentialOnly** olarak belirlenmiş olması ve iletişim seviyesinde istemci yetki belgesi tipi olarak **None** *(**transport** elementi içerisindeki **clientCredentialType** niteliği yardımıyla belirlenmiştir)* değerinin kullanılmış olmasıdır. Buna göre istemcilerin kimlik bilgileri için herhangibir doğrulama yapılmaz. Bir başka deyişle herkes bu servisi kullanabilir.

İstemci tarafındaki Console uygulamasına ait konfigurasyon dosyasının başlangıç içeriği ise aşağıdaki gibi olmalıdır. *(İstemci uygulama için gerekli proxy sınıfı **svcutil.exe**aracı yardımıyla CebirLib.dll' i üzerinden elde edilmiştir. Bu konu daha önceden işlendiğinden burada tekrarlanmamıştır.)*

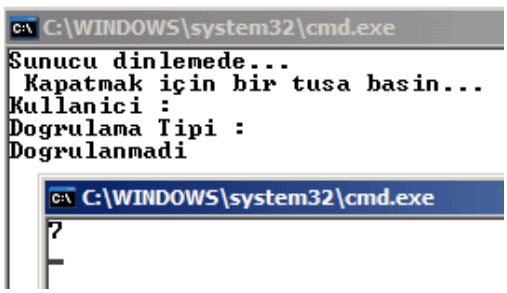
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="CebirClientBindingHttpCfg">
          <security mode="TransportCredentialOnly">
            <transport clientCredentialType="None" />
          </security>
        </binding>
```

```
        </basicHttpBinding>
      </bindings>
      <client>
        <endpoint address="http://localhost:1600/AritmetikServisi"
binding="basicHttpBinding" bindingConfiguration="CebirClientBindingHttpCfg" c
ontract="AritmetikServisi" name="CebirClientHttpEndPoint" />
      </client>
  </system.serviceModel>
</configuration>
```

İstemci tarafındaki console uygulamasına ait kodlar ise aşağıdaki gibidir.

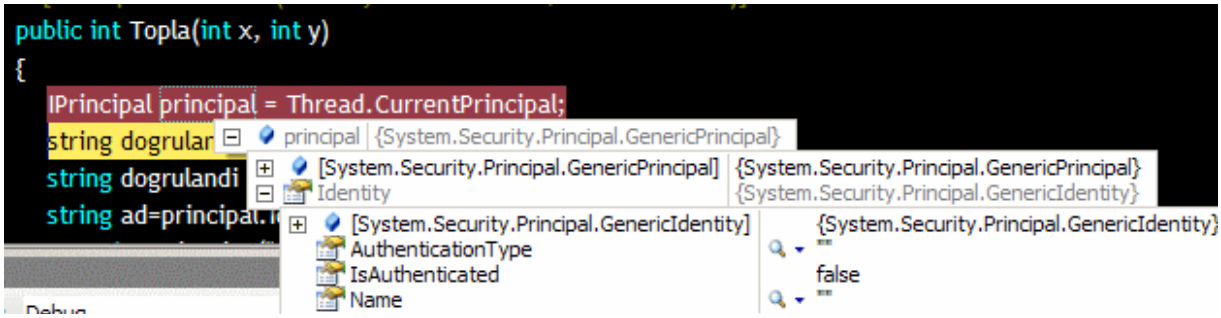
```
using System;
using System.ServiceModel;

namespace ClientApp
{
  class Program
  {
    static void Main(string[] args)
    {
      AritmetikServisiClient client = new
AritmetikServisiClient("CebirClientHttpEndPoint");

      int sonuc = client.ToplamaOperasyonu(3, 4);
      Console.WriteLine(sonuc.ToString());
      Console.ReadLine();
    }
  }
}
```

Uygulama bu haliyle çalıştırıldığında aşağıdakine benzer bir ekran görüntüsü ile karşılaşılır.



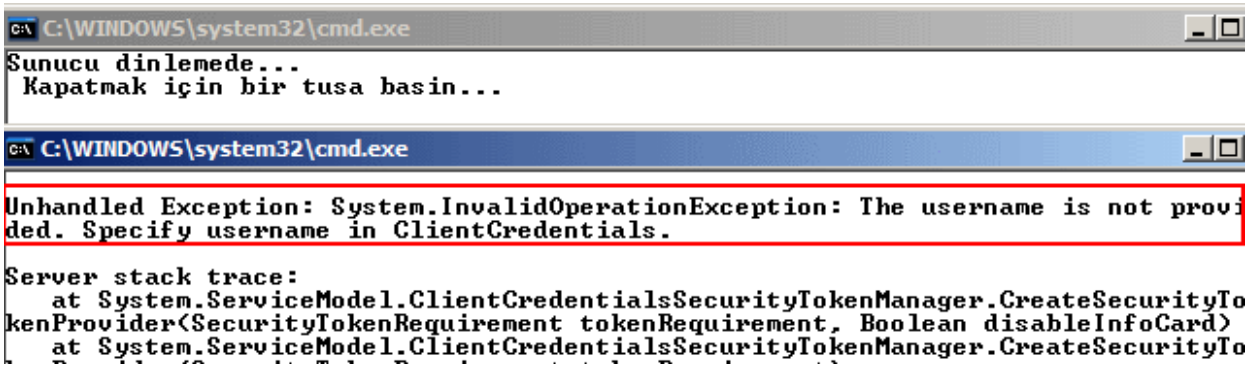
Görüldüğü gibi kullanıcı adı, doğrulama tipi veya kullanıcı bilgisinin herhangibir mekanizma ile doğrulanıp doğrulanmadığı bilgisi bulunmamaktadır. Bunun sebebi güvenlik ayarlarında **mode** niteliğinin değerinin **None** olarak belirlenmiş olmasıdır. Durum

Aritmetik sınıfı içerisindeki Topla metodu debug modda incelendiğinde açık bir şekilde izlenebilir. Aşağıdaki ekran görüntüsünde bu durum yer almaktadır.



Buradan şu sonuca varılabilir. **None** güvenlik modunda hiç bir şekilde istemciden yetki bilgileri gönderilmez. Her istemci isimsiz kullanıcı**(anonymous user)** gibi ele alınır.

Şimdi olayı biraz daha farklı bir hale getirelim. **mode** niteliğinin değerini hem istemci hemde servis tarafında **Basic** olarak değiştirelim. Ardından servis ve istemci uygulamaları tekrar çalıştıralım. Bu durumda aşağıdaki ekran görüntüsü ile karşılaşırız.



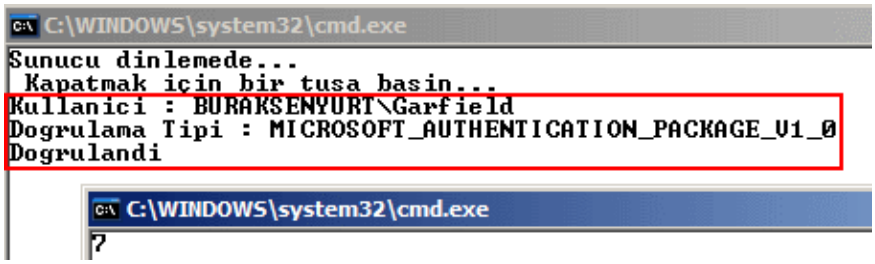
Basic modda iken istemcinin servis tarafına kendisini tanıtması gerekmektedir. Bu nedenle aynen hata mesajında olduğu gibi proxy sınıfı üzerinden **ClientCredentials** özelliği kullanılmalıdır. Söz konusu operasyonda örnek olarak iki test kullanıcısı oluşturulmuştur. Garfield ve Rolfield isimli bu kullanıcılar, daha sonradan iki farklı **Windows Group** altında birleştirilecek ve rol bazlı yetkilendirmelerin**(Role Based Authorization)** nasıl yapılacağı ele alınacaktır. Söz konusu kullanıcılara ait Username ve Password bilgileri aşağıdaki gibidir. Bu kullanıcılar tamamen hayalidir :)

Kullanıcı Adı	Şifre
Garfield	Garfi1234.?
Rolfield	Garfi1234.?

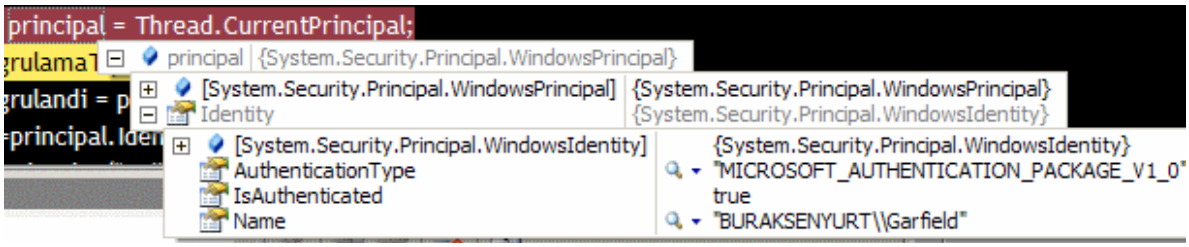
Bu işlemin ardından istemci tarafındaki kodlar aşağıdaki gibi düzenlenebilir.

```
using System;
using System.ServiceModel;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         AritmetikServisiClient client = new
AritmetikServisiClient("CebirClientHttpEndPoint");

         client.ClientCredentials.UserName.UserName = "Garfield";
         client.ClientCredentials.UserName.Password = "Garfi1234.?";

         int sonuc = client.ToplamaOperasyonu(3, 4);
         Console.WriteLine(sonuc.ToString());
         Console.ReadLine();
      }
   }
}
```

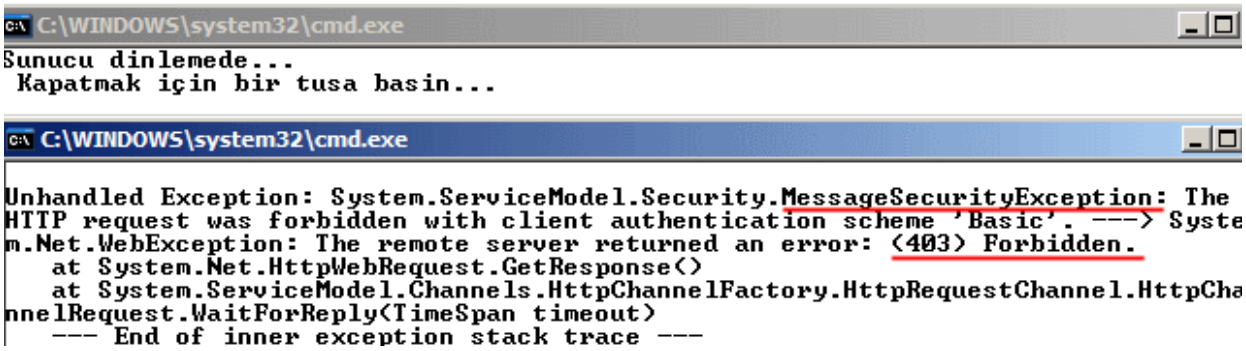
Dikkat edilecek olursa client nesne örneği üzerinden **ClientCredentials** özelliğine geçilmiş ve buradan **UserName** özelliği yardımıylada kullanıcı adı**(UserName)** ve şifre**(Password)**bilgileri belirtilmiştir. Servis ve istemci tarafı tekrar test edilirse aşağıdakine benzer bir ekran görüntüsü ile karşılaşılır.



Yine çalışma zamanında Topla metodu içerisinde **debug** işlemi gerçekleştirilirse, aşağıdaki ekran görüntüsünde yer aldığı gibi **WindowsPrincipal** tipinin örneklendiği ve kullanıcı bilgilerinin **WindowsIdentity** tipi üzerinden elde edildiği görülür. Bu tipik olarak kullanıcının servis uygulamasının çalıştığı Windows işletim sistemindeki kullanıcılardan arandığınında bir göstergesidir.



Eğer hatalı kullanıcı bilgisi veya yanlış şifre girilirse aşağıdakine benzer bir ekran görüntüsü ile karşılaşılır. *(Örneğin kullanıcı adı Garfieldd olarak çift d ile bitecek şekilde girilirse.)*



Dikkat edilecek olursa çalışma zamanında **MessageSecurityException** istisnası alınmış ve **403 Forbidden** mesajı ile karşılaşılmıştır. Şimdi söz konusu kullanıcıların yetkilendirilmesinin nasıl ele alınabileceğini incelemeye çalışalım. Bu amaçla söz konusu Garfield ve Lorfield isimli kullanıcılar Yonetici ve Calisan isimli iki farklı Windows grubunda toplanmıştır.

Kullanıcı Adı	**Windows Grubu**
Garfield	Yonetici, Calisan
Rolfield	Calisan

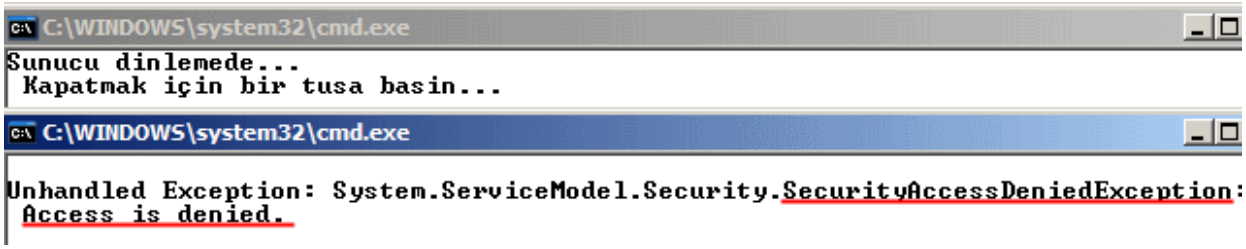
Topla metodunu sadece Yonetici grubundaki kullanıcıların çalıştırması istenirse dekleratif**(declarative)** olarak **PrincipalPermission** niteliği ele alınmalıdır. PrincipalPermission niteliği**(attribute)**, **System.Security.Permissions** isim alanı altında yer almaktadır. Bu sebeple ilgili isim alanının Aritmetik sınıfına eklenmesinde fayda vardır. PrincipalPermission niteliğinin temel kullanımı aşağıdaki gibidir.

```
[PrincipalPermission(SecurityAction.Demand,Role="Yonetici")]
public int Topla(int x, int y)
{
```

PrincipalPermission niteliği metodlar gibi sınıflarada uygulanabilir. Ancak tavsiye edilen metod seviyesinde uygulanmasıdır. Bununla birlikte kendisinden türetilme yapılmasına izin vermeyen**(sealed class)**bir sınıftır. Bu sınıfın .Net içerisindeki içeriği aşağıdaki gibidir. *(Sınıf içeriğinin elde edilmesi için Özcan Değirmenci tarafından geliştirilen Fox Decompiler aracı kullanılmıştır*

```
[ComVisible(true), AttributeUsage(AttributeTargets.Method | AttributeTargets.Class,
AllowMultiple=true, Inherited=false), Serializable]
public sealed class PrincipalPermissionAttribute : CodeAccessSecurityAttribute
{
   // Constructors
   public PrincipalPermissionAttribute (SecurityAction action);

   // Methods
   public override IPermission CreatePermission ();

   // Properties
   public string Name { get; set; }
   public string Role { get; set; }
   public bool Authenticated { get; set; }

   // Instance Fields
   private string m_name;
   private string m_role;
   private bool m_authenticated;
}
```

Böylece Topla metodunu sadece Yonetici rolündeki kullanıcıların çağırabileceği belirtilmiş olur. Eğer istemci tarafından Lorfield isimli kullanıcı ile Topla metodu çağırılırsa aşağıdaki ekran görüntüsü ile karşılaşılır.



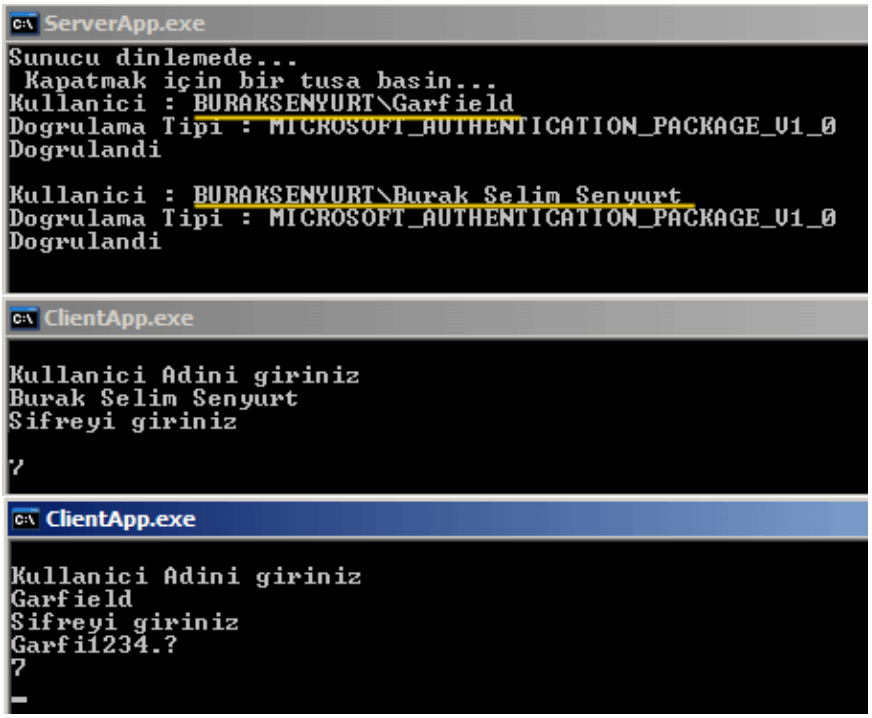
Görüldüğü gibi kullanıcı doğrulanmış**(Authenticate)** ancak Yonetici rolünde olmadığı için yetkisi geçersiz kılınmıştır**(Unauthorized)**. Bu sebepten dolayı **"Access is denied"** hata mesajı ve **SecurityAccessDeniedException** tipinden bir çalışma zamanı istisnası**(runtime exception)** alınmıştır. Oysaki Garfield isimli kullanıcı ile erişilmek istendiğinde bir problem olmadan metod çağrısı gerçekleştirilebilir.

PrincipalPermission niteliği istenirse bir metod için birden fazla kez kullanılabilir. Sadece bu örnekte olduğu gibi rollere yetki vermek amacıyla değil belirli kullanıcıları yetkilendirmek yada birden fazla role izin vermek amacıyla kullanılabilir. Aşağıdaki örnekte bu durum analiz edilmektedir.

```
[PrincipalPermission(SecurityAction.Demand,Role="Yonetici")]
[PrincipalPermission(SecurityAction.Demand,Name="BURAKSENYURT\\Burak Selim Senyurt")]
public int Topla(int x, int y)
{
```

Buradaki tanımlamalara göre BURAKSENYURT alanı içerisinde yer alan Burak Selim Senyurt isimli kullanıcıda Topla metodu için yetkilendirilmiş sayılmaktadır.

Elbette tek bir istemci yerine birden fazla istemci çalıştırıldığında **Thread** sınıfının **CurrentPrincipal** özelliği bağlanan kişiye ait bilgileri içeririr. Söz gelimi yukarıdaki örnek kodlara göre birden fazla istemci çalıştırıldığında aşağıdakine benzer bir ekran görüntüsü elde edilebilir.



Kullanıcılara ait yetki kontrolü istenirse kod içerisindende zorunlu bir şekilde**(imperatively)** gerçekleştirilebilir. Bunun için Aritmetik sınıfı içerisindeki Topla metodu aşağıdaki gibi düzenlemelidir.

```
public int Topla(int x, int y)
{
   IPrincipal principal = Thread.CurrentPrincipal;
```

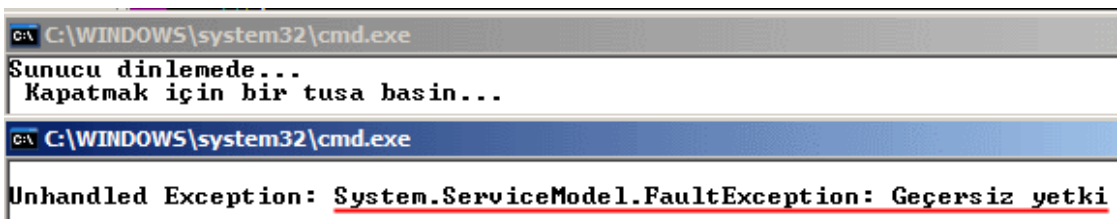
```
    string dogrulamaTipi=principal.Identity.AuthenticationType;
    string dogrulandi = principal.Identity.IsAuthenticated ? "Dogrulandi" : "Dogrulanmadi";
    string ad=principal.Identity.Name;

    #region Kod içerisinden yetki kontrolü

    WindowsPrincipal wp = (WindowsPrincipal)principal;
    if (wp.IsInRole("Yonetici"))
    {
        Console.WriteLine("Kullanıcı : " + ad + "\n" + "Doğrulama Tipi : " + dogrulamaTipi
+ "\n" + dogrulandi + "\n");
        return x + y;
    }
    else
        throw new FaultException("Geçersiz yetki");

    #endregion
}
```

İlk olarak **WindowsPrincipal** tipi yakalanır. Bunun için yine **Thread** sınıfından ve static üyelerinden **CurrentPrincipal** ile elde edilen referanstan faydalanılır. Elde edilen **WindowsPrincipal** nesne örneği üzerinden **IsInRole** metodu yardımıyla talepte bulunan istemci tarafından gelen kullanıcının Yonetici rolünde olup olmadığı kontrol edilebilir. Örnekte kullanıcının Yonetici rolü içerisinde olmaması halinde bir **FaultException***(FaultException kullanımı için* ***System.ServiceModel*** *isim alanının ilave edilmesi gerekir)* istisnası fırlatılmaktadır.

Bu tarz bir kod parçasını kullanmak özellikle rol tabanlı üyelik kontrolü söz konusu olduğunda şart değildir. Nitekim **IsInRole** metodu zaten o anki **Principal** üzerinden kolaylıkla elde edilebilir. Bir başka deyişle aşağıdaki gibi bir kod parçasıda aynı işlemi görecektir.

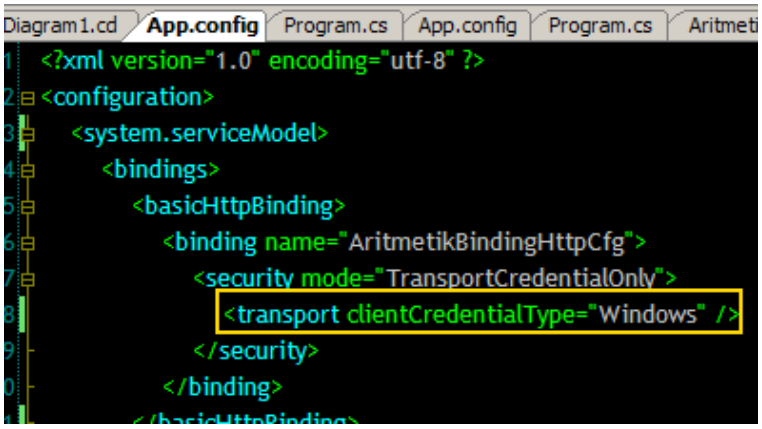
```
if (principal.IsInRole("Yonetici"))
{
```

Yine Garfield isimli kullanıcı ile deneme yapılırsa herhangibir sorun ile karşılaşılmadan Topla metodunun çağırılabildiği görülür. Ancak Lorfield isimli kullanıcı ile Topla metodu çağırılırsa aşağıdaki ekran görüntüleri ile karşılaşılır.

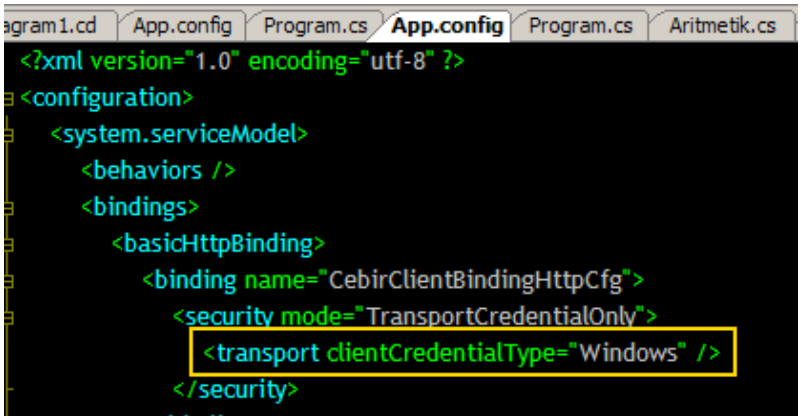


Basic güvenlik modu genellikle istemcilerin servis ile aynı güvenlik alanına**(security domain)** girmediği durumlarda ele alınır. Bu sebepten eğer kullanıcılar zaten güvenlik alanına varsayılan olarak dahil oluyorlarsa Windows Authentication Mode kullanılarak istemcilerin var olan ehliyet bilgilerini belirtmeden servise ulaşmaları sağlanabilir. Tek yapılması gereken istemci ve servis tarafındaki**security** elementlerinde **mode** niteliğini **Windows** olarak ayarlamaktır.

Servis tarafı;



İstemci tarafı;



Eğer servis tarafında **Active Directory** kullanılıyorsa bu durumda rol yönetimi için **Windows Token Role Provider** seçilmelidir. Bu ayarlama sadece servis tarafındaki konfigurasyon dosyasındaaşağıdaki gibi yapılmalıdır.

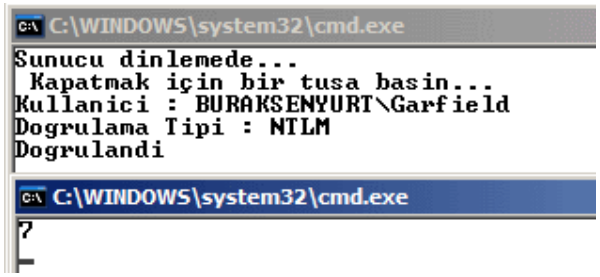
```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="AritmetikServiceBehavior">
          <serviceAuthorization principalPermissionMode="UseWindowsGroups"
```

```
/>
            </behavior>
          </serviceBehaviors>
       </behaviors>
       <bindings>
          <basicHttpBinding>
             <binding name="AritmetikBindingHttpCfg">
                <security mode="TransportCredentialOnly">
                   <transport clientCredentialType="None" />
                </security>
             </binding>
          </basicHttpBinding>
       </bindings>
       <services>
          <service behaviorConfiguration="AritmetikServiceBehavior" name="CebirLib.
Aritmetik">
             <endpoint address="http://localhost:1600/AritmetikServisi"
binding="basicHttpBinding" bindingConfiguration="AritmetikBindingHttpCfg"
name="AritmetikServiceHttpEndPoint" contract="CebirLib.IAritmetik" />
          </service>
       </services>
    </system.serviceModel>
</configuration>
```

Görüldüğü gibi **serviceBehaviors** elementi içerisine **serviceAuthorization** elementi eklenmiştir. Bu element içerisinde yer **principalPermissionMode** niteliğinin değeri ise **UseWindowsGroups**olarak belirlenmiştir. Her iki durumdada istemci tarafından servise kullanıcı bilgileri otomatik olarak gönderilecektir. Elbette domain' e dahil olmuşlarsa. Yanlız **Windows** modu kullanıldığında ve rol tabanlı bir yetkilendirme söz konusu olduğunda istenirse yine istemci tarafında belirli bir kullanıcı için bağlantı gerçekleştirilmesi sağlanabilir. Lakin böyle bir durumda istemci tarafındaki kodların aşağıdaki gibi ele alınması gerekir.

```
AritmetikServisiClient client = new AritmetikServisiClient("CebirClientHttpEndPoint");
client.ClientCredentials.Windows.ClientCredential.UserName = "Garfield";
client.ClientCredentials.Windows.ClientCredential.Password = "Garfi1234.?";
client.ClientCredentials.Windows.ClientCredential.Domain = "BURAKSENYURT"; //
Domain varsayılan ise yazılmak zorunda değildir.

int sonuc = client.ToplamaOperasyonu(3, 4);
Console.WriteLine(sonuc.ToString());
Console.ReadLine();
```

Bu durumda servis ve istemci çalıştırıldığında aşağıdaki ekran görüntüsü elde edilir.



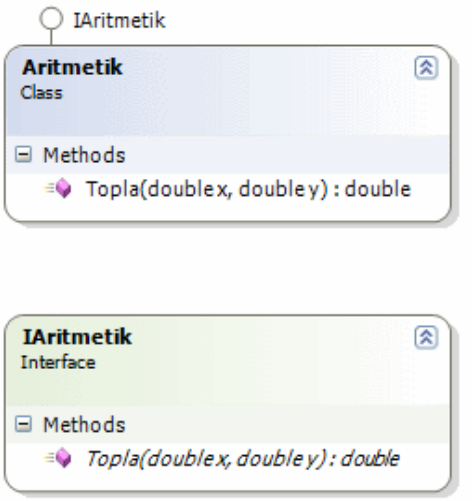
Bu makalemizde basit olarak intranet tabanlı sistemlerde **basicHttpBinding** bağlayıcı tipini kullanarak Windows tabanlı doğrulama ve yetkilendirmelerin**(Windows Based Authentication and Authorization)** nasıl ele alınabileceğini incelemeye çalıştık. **netTcpBinding** veya **wsHttpBinding** bağlayıcı tipleri için iletişim seviyesinde güvenlik modunun varsayılan değeri Windows' dur. Dolayısıyla bu tipleri kullanırken config dosyası içerisinde ekstra bir işlem yapılmasına gerek yoktur. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

Örnek Uygulama için Tıklayın

WCF - Internet Üzerinden Güvenliği Sağlamak - 2 (2007-07-05T21:17:00)

wcf,

Hatırlanacağı gibi bir önceki makalemizde iletişim seviyesinde**(Transport Level)** güvenliğin sağlanabilmesi için gerekli ayarların nasıl yapılabileceğini incelemiştik. Bu makalemizde kaldığımız yerden devam ederek servis tarafındaki doğrulama işlemleri için üyelik ve rol yönetim**(Membership and Role Management)** sistemini devreye alacak ve istemci tarafını yazarak test edeceğiz. İlk olarak önceki yazımızda açmış olduğumuz WCF Service uygulamasına dönelim. Her zaman olduğu gibi basit bir arayüzü, servis sözleşmesi**(Service Contract)** olacak şekilde tasarlayacağız ve bunun uyarlamasını yapacak bir sınıf geliştireceğiz. İşlemlerin kolay bir şekilde anlaşılabilmesi için servis tarafındaki tipler mümkün olduğu kadar basit düşünülmüştür.



IAritmetik arayüzü(interface) ve Aritmetik sınıfına(class) ait kodlar aşağıdaki gibidir.

```
using System;
using System.ServiceModel;

[ServiceContract(Name="Cebirci",Namespace="http://www.bsenyurt.com/Cebirci")]
public interface IAritmetik
{
   [OperationContract]
   double Topla(double x, double y);
}

public class Aritmetik : IAritmetik
{
   #region IAritmetik Members

   public double Topla(double x, double y)
   {
      return x + y;
   }

   #endregion
}
```

Aritmetik sınıfı basit olarak istemcilere Topla isimli bir işlevsellik sunmaktadır. Standart olarak
arayüze **ServiceContract** ve **OperationContract** nitelikleri**(attributes)** uygulanmıştır.
WCF servisinin svc uzantılı dosyasının içeriği ise aşağıdaki gibi olmalıdır.

```
<% @ServiceHost Language=C#
Debug="true" Service="Aritmetik" CodeBehind="~/App_Code/Service.cs" %>
```

Burada önemli olan noktalardan bir tanesi web.config dosyasının içeriğidir. Windows Communication Foundation için gerekli ayarları içerek web.config dosyasında bu sefer, iletişim seviyesinde güvenlik ve ehliyet**(Credential)** kontrol kuralları içinde bazı eklemeler yapılmalıdır. Sonuç itibariyle web.config dosyasının başlangıçtaki içeriği aşağıdaki gibi tasarlanabilir.

```
<?xml version="1.0"?>

<configuration>
<system.serviceModel>
   <bindings>
      <wsHttpBinding>
         <binding name="CebirServiceBindingCfg">
            <security mode="TransportWithMessageCredential">
               <transport clientCredentialType="None" />
               <message clientCredentialType="UserName" />
            </security>
         </binding>
      </wsHttpBinding>
   </bindings>
   <services>
      <service behaviorConfiguration="CebirServiceBehavior" name="Aritmetik">
         <endpoint binding="wsHttpBinding"
bindingConfiguration="CebirServiceBindingCfg" name="CebirServiceEndpoint"
contract="IAritmetik" />
      </service>
   </services>
   <behaviors>
      <serviceBehaviors>
         <behavior name="CebirServiceBehavior">
            <serviceMetadata httpsGetEnabled="true" />
         </behavior>
      </serviceBehaviors>
   </behaviors>
</system.serviceModel>

<system.web>
   <compilation debug="true"/>
</system.web>

</configuration>
```

Servis tarafında **wsHttpBinding** bağlayıcı tipi**(Binding Type)** kullanılmaktadır. Bağlayıcı tipe ait konfigurasyon ayarlarında dikkat edileceği üzere **security** elementi içerisinde

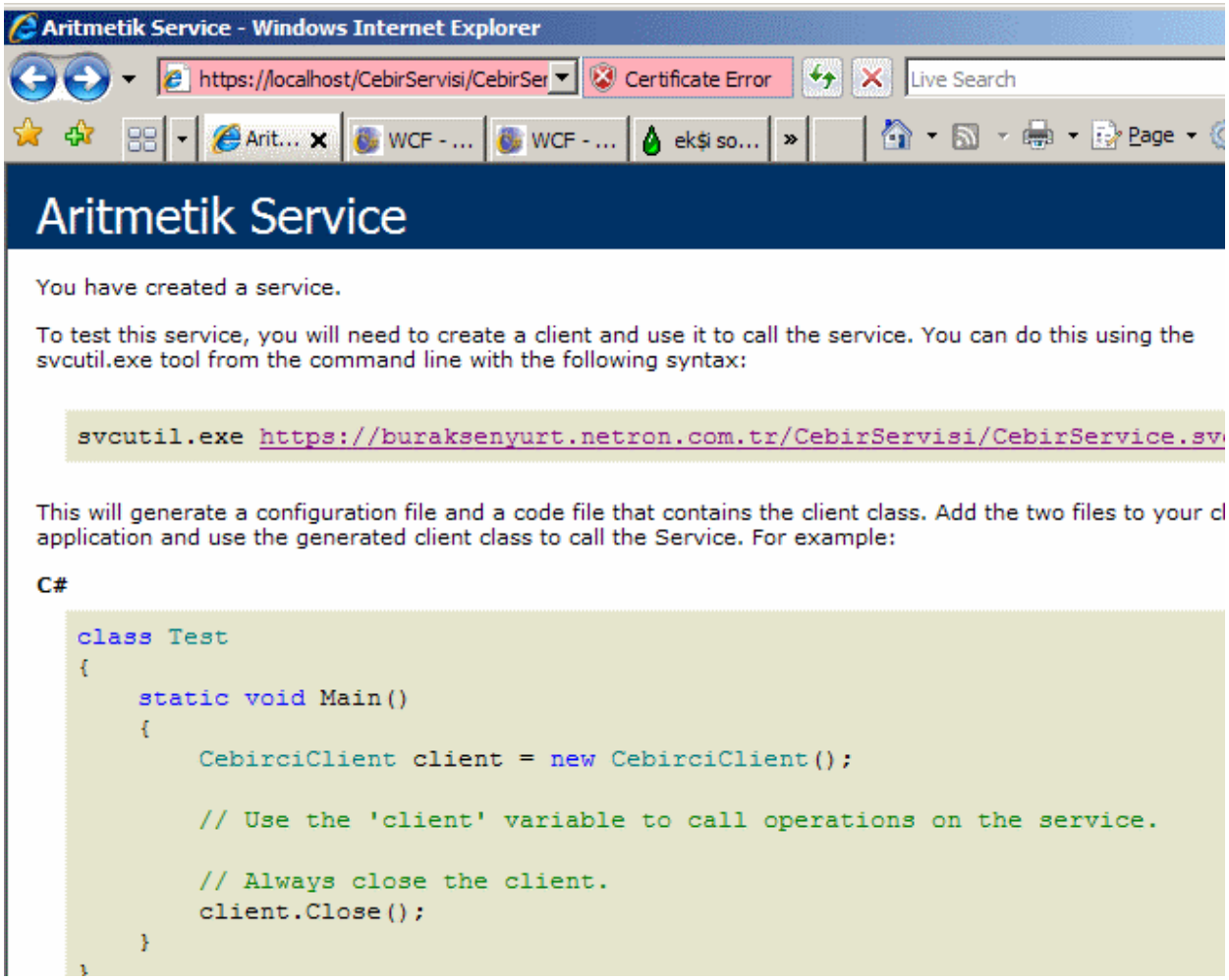
iletişim ve mesaj seviyesi için gerekli istemci ehliyet tipleri **transport** ve **message** alt elementleri yardımıyla tanımlanmaktadır.

security elementi içerisindeki **mode** niteliği **TransportWithMessageCredential** olarak ayarlanmıştır. Buna göre, daha öncedende bahsedilen mesaj bütünlüğü**(Integrity)**, mesaj mahremiyeti**(Privacy)** ve müşterek doğrulama **(Mutual Authentication)** gibi ilkeler HTTPS tarafından sağlanır. Bu nedenlede servis tarafının HTTPS ile hizmet verecek şekilde tasarlanmış olması bir başka deyişle sertifikalandırılmış olması şarttır*(Daha önceden bir sertifika hazırlamamızın nedenide budur).*

Diğer taraftan istemcilerin doğrulanması **SOAP** güvenliğine uygun olacak şekilde yapılır. Bir başka deyişle istemcilerin kendilerini servis tarafına kullanıcı adı, şifre veya sertifika**(Certificate)** yoluyla tanıtması gerekir. Geliştirilecek olan örnekte kullanıcı adı ve şifre kullanımı ele alınacaktır. özellikle **message** elementinde yer alan **clientCredentialType**niteliğinin değeri istemcinin doğrulanması için kullanılacak istemci yetki belgesinin**(Credential)** tipini belirtir.

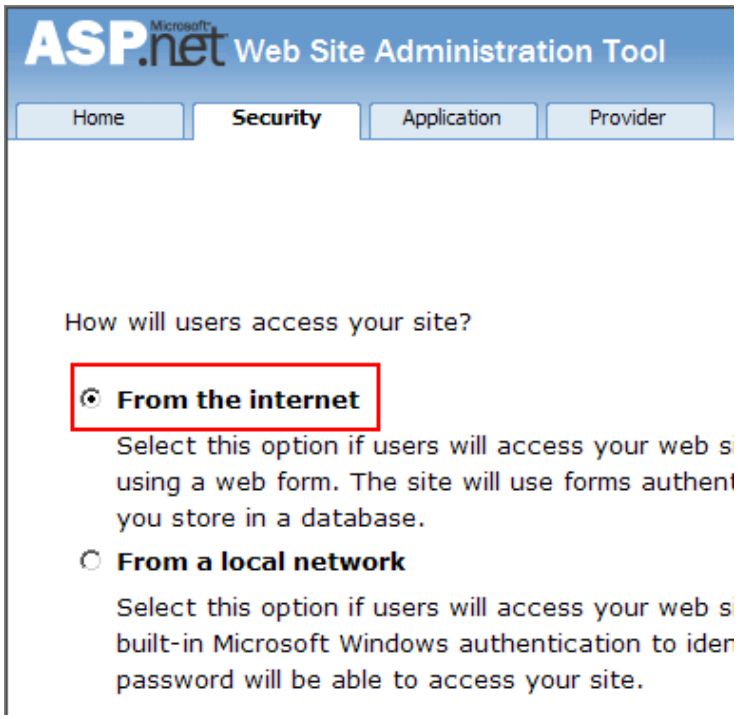
***NOT : Windows SDK** dökümantasyonuna göre **security mode** değeri **TransportWithMessageCredential** olarak ayarlanmışsa, transport alt elementi görmezden gelinir.*

Bu işlemlerin tamamlanması ile birlikte servis herhangibir tarayıcı penceresinden talep edilebilir. Ekran çıktısı aşağıdakine benzer olacaktır.

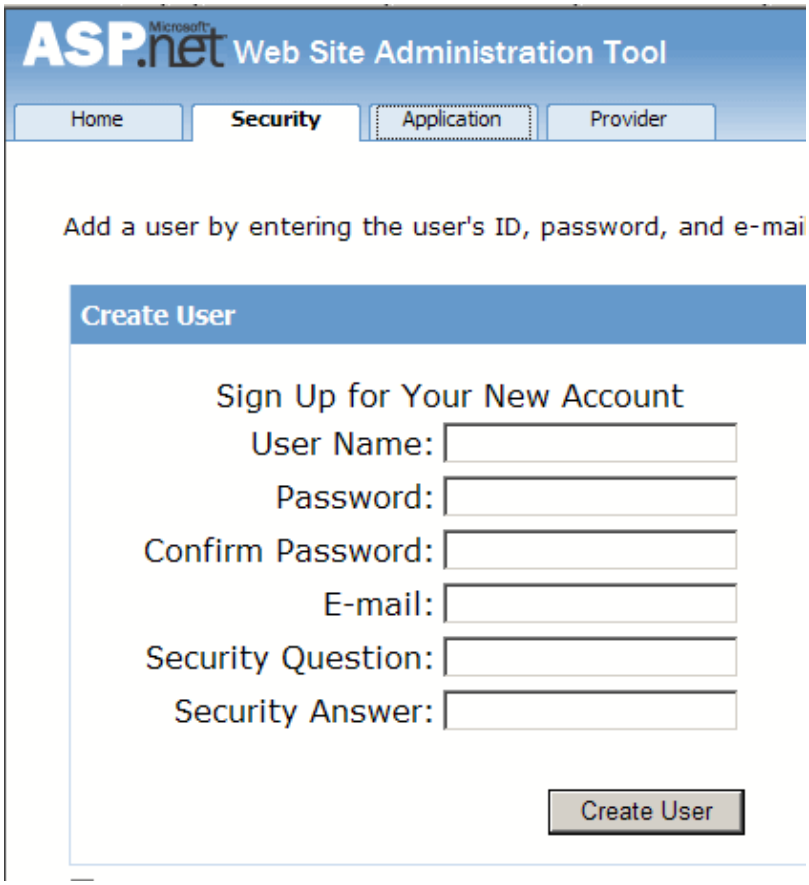


Bu çıktının elde edilmesi için adres alanına **https:**//localhost/CebirServisi/CebirService.svc yazılması gerekmektedir. Dikkat edilecek olursa http yerine https kullanılmaktadır.

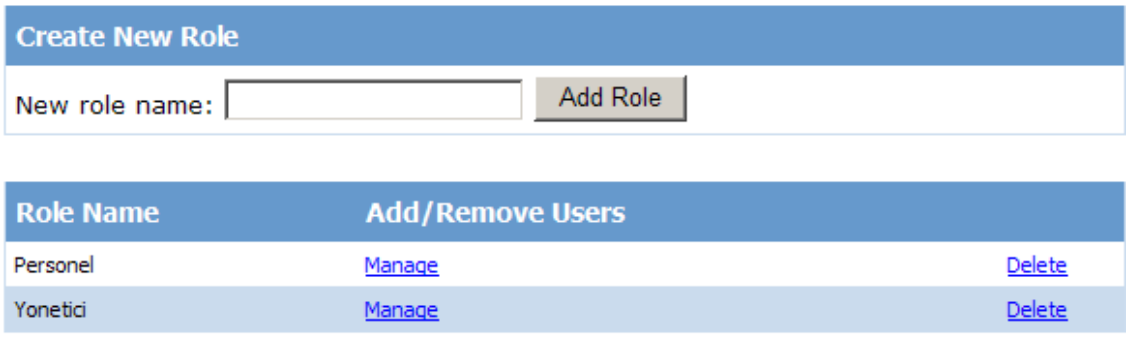
Servis tarafında yapılması gereken işlemlerden biriside kullanıcı hesaplarının saklanması için gerekli Asp.Net üyelik veritabanının oluşturulmasıdır. Bu amaçla **Web Site Administraton Tool** aracından yararlanılabilir. öncelikli olarak **Security** kısmından doğrulama işlemlerinin internet üzerinde yapılacağının bildirilmesi gerekmektedir. Bu nedenle **From the Internet** seçeneği işaretlenir.



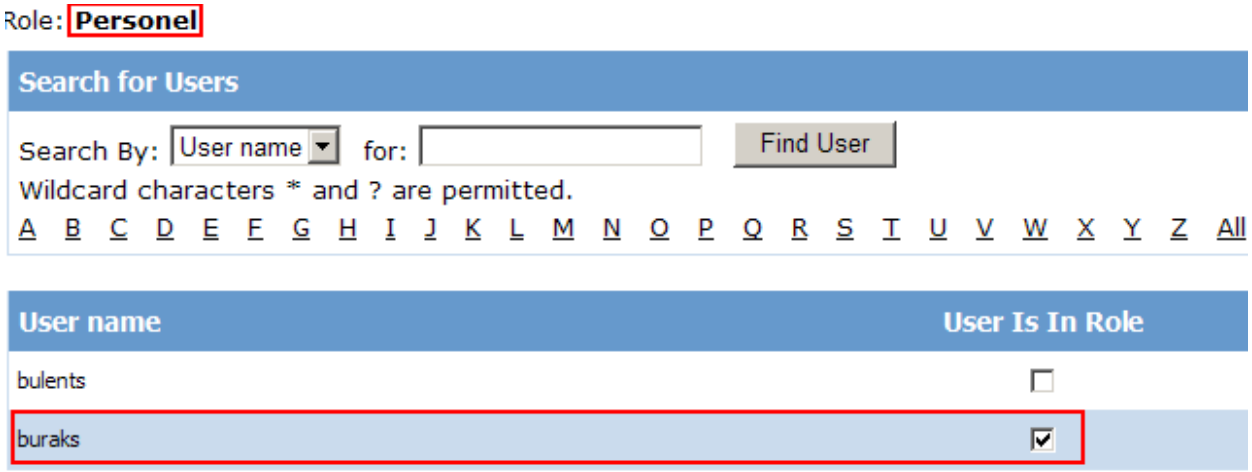
Ardından bir kaç örnek kullanıcı hesabı aşağıdaki ekran görüntüsünde yer alan **Create User** kontrolü ile oluşturulur.



örneklerde kullanılmak üzere daha sonra farklı rollere atanacak olan buraks ve bulents isimli iki kullanıcı oluşturulmuştur. Bu kullanıcıların şifreleride 123456. olarak belirlenmiştir. İlerleyen bölümlerde örnek rol kullanımlarıda ele alınacağından Personel ve Yonetici isimli iki rol **Create New Role** kontrolü yardımıyla tanımlanır *(Rollerin kullanılabilmesi için **Roles** kısmından **Enable Roles** linkine tıklanılmalıdır).*



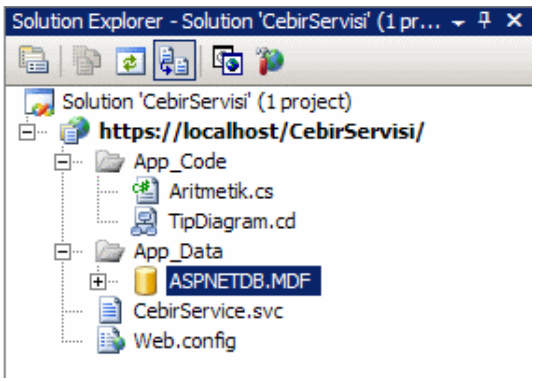
Bu işlemin ardından örnek olarak oluşturulan bulents ve buraks isimli kullanıcılar farklı rollere atanırlar. buraks isimli kullanıcı Personel rolüne, bulents isimli kullanıcı ise Yonetici rolüne atanır. Bunun için **Web Site Administrator Tool** içerisindeyken, rollerin eklendiği kontrolde yer alan **Manage** linki kullanılabilir. örneğin buraks kullanıcısının Personel rolüne atanması sonrası ekran görüntüsü aşağıdaki gibi olacaktır.



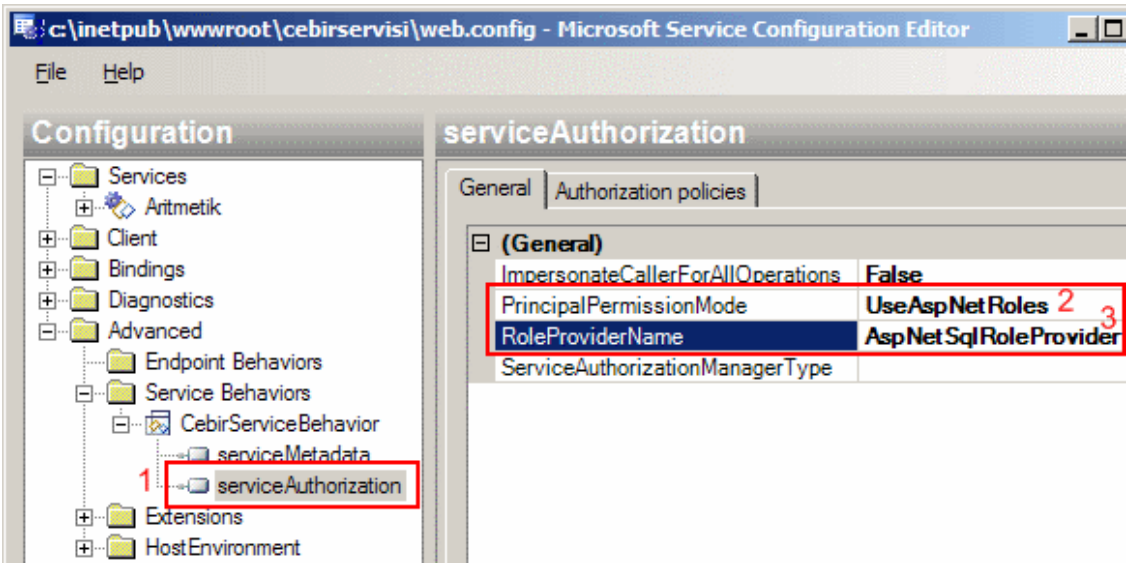
Bu işlemler sonrasında web.config dosyasına **roleManager** ve **authentication** elementleri aşağıdaki gibi eklenecektir.

```
<system.web>
   <roleManager enabled="true" />
   <authentication mode="Forms" />
   <compilation debug="true"/>
</system.web>
```

Yerel**(Local)** veritabanı kullanıldığından aynen web uygulamalarında olduğu gibi ASPNETDB.mdf dosyası App_Data klasörü altına açılır. *(Burada kriter her zamanki gibi root web.config dosyasıdır. Nitekim Asp.Net uygulamalarından bilindiği üzere istenirse ASPNETDB veritabanı SQL Server sunucusu üzerinde de tutulabilir. Varsayılan ayar local olarak kullanılmasını sağlamaktadır.)*



Bu işlemlerde tamamlandıktan sonra yetkilendirme(authorization) ve rol(Role) işlemleri için servise ait davranış**(behavior)** tanımlamaları yapılması gerekmektedir. Bu seferki ayarlamaları**Microsoft Service Configuration Editor** yardımıyla gerçekleştirebiliriz. öncelikli olarak CebirServiceBehavior davranışına **serviceAuthorization** elementi eklenir. Bu işlemin ardından ilk olarak**PrincipalPermissonMode** değeri **UseAspNetRoles** olarak belirlenir. Sonrasında ise **RoleProviderName** değeri **AspNetSqlRoleProvider** olarak belirlenir.

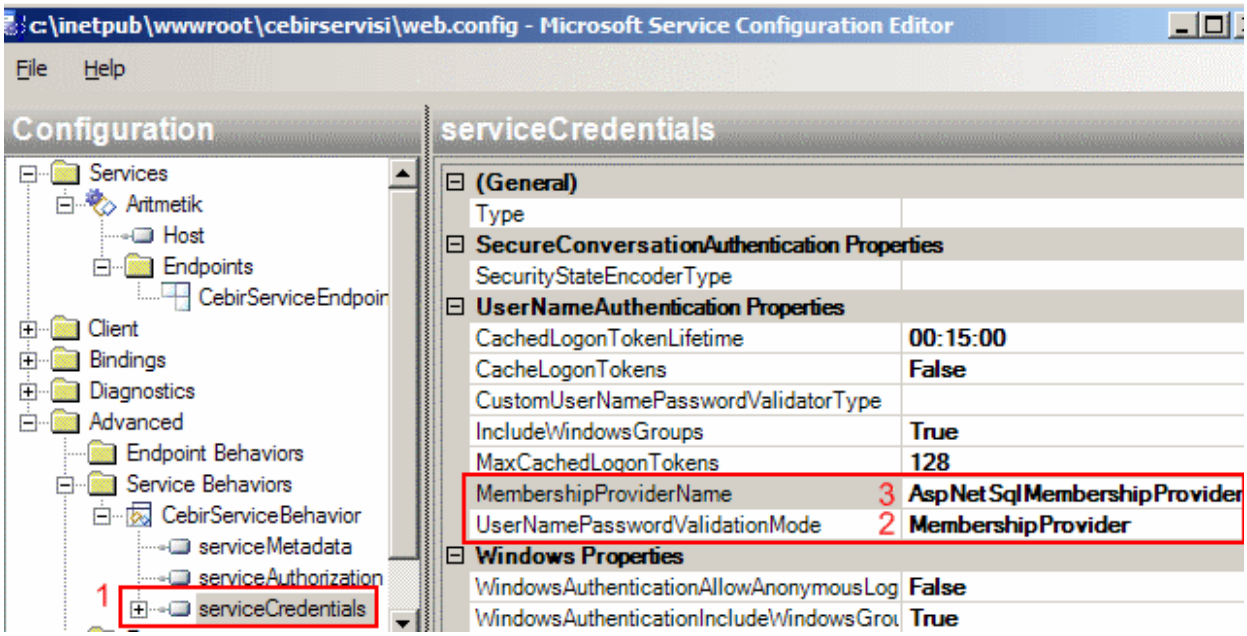


Böylece istemcilerin rol yönetiminin .Net Framework ile hazır olarak gelen **AspNetSqlRoleProvider** tipi yardımıyla gerçekleştirileceği belirtilmiş olur. AspNetSqlRoleProvider tipi **machine.config**dosyası içerisinde tanımlanmış olup rol yönetimini üstlenen hazır .Net sınıflarından birisidir. Söz konusu tip aynı zamanda Asp.Net Web Site Administrator aracının kullandığı varsayılan rol sağlayıcısıdır**(default role**

provider). Temel görevi kullanıcılar ile roller arasındaki ilişkilerin kurulması ve kontrol edilmesidir. Aşağıdaki ekran görüntüsünde söz konusu sağlayıcının machine.config dosyasında bulunduğu yer gösterilmektedir.

```
    <roleManager>
     <providers>
       <add name="AspNetSqlRoleProvider" connectionStringName="LocalSqlServer" applicationName="/" type
="System.Web.Security.SqlRoleProvider, System.Web, Version=2.0.0.0, Culture=neutral, PublicKeyToken=
b03f5f7f11d50a3a" />
       <add name="AspNetWindowsTokenRoleProvider" applicationName="/" type="System.Web.Security.
WindowsTokenRoleProvider, System.Web, Version=2.0.0.0, Culture=neutral, PublicKeyToken=
b03f5f7f11d50a3a" />
     </providers>
    </roleManager>
  </system.web>
</configuration>
```

serviceAuthorization davranışının belirlenmesinden sonra, **serviceCredentials** isimli bir başka davranışın daha servis tarafında eklenmiş olması gerekmektedir. Bu sefer söz konusu davranış ile istemcilere ait hesap bilgilerinin kim tarafından kontrol edileceği belirlenir. Burada **UserNamePasswordValidationMode** özelliğinin değeri **MembershipProvider** ve**MembershipProviderName** özelliğinin değeride **AspNetSqlMembershipProvider** olarak seçilir.



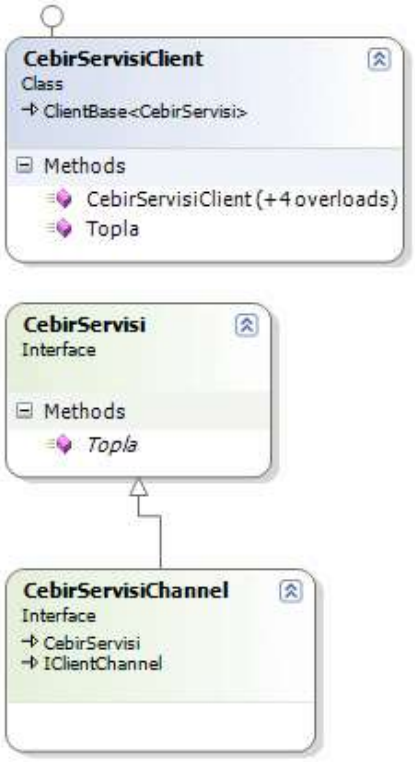
Burada belirtilen **AspNetSqlMembershipProvider**, **machine.config** dosyası içerisinde tanımlanmış olan kullanıcı doğrulama işlemlerini üstlenen varsayılan .Net tipidir. Aşağıdaki ekran görüntüsünde bu tipin machine.config dosyası içerisindeki yeri gösterilmektedir.

```
<membership>
  <providers>
    <add name="AspNetSqlMembershipProvider" type="System.Web.Security.SqlMembershipProvider,
System.Web, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b03f5f7f11d50a3a" connectionStringName=
"LocalSqlServer" enablePasswordRetrieval="false" enablePasswordReset="true" requiresQuestionAndAnswer=
"true" applicationName="/" requiresUniqueEmail="false" passwordFormat="Hashed"
maxInvalidPasswordAttempts="5" minRequiredPasswordLength="7" minRequiredNonalphanumericCharacters
="1" passwordAttemptWindow="10" passwordStrengthRegularExpression="" />
  </providers>
</membership>
```

Tüm bu işlemlerin ardından **web.config** dosyasının içeriğinde yer alan **serviceBehaviors** kısmında aşağıdaki yenilemeler gerçekleşecektir.

```
<serviceBehaviors>
   <behavior name="CebirServiceBehavior">
      <serviceMetadata httpsGetEnabled="true" />
      <serviceAuthorization
principalPermissionMode="UseAspNetRoles" roleProviderName="AspNetSqlRoleProv
ider" />
      <serviceCredentials>
         <userNameAuthentication userNamePasswordValidationMode="MembershipP
rovider" membershipProviderName="AspNetSqlMembershipProvider" />
      </serviceCredentials>
   </behavior>
</serviceBehaviors>
```

Artık istemci tarafı tasarlanmaya başlanabilir. İstemci program, basit bir Console uygulaması olarak tasarlanacaktır. Herşeyden önce istemci için gerekli proxy sınıfının üretilmesi gerekmektedir. Ne varki makaleyi hazırladığım sıralarda gerek **svcutil.exe** aracı gerek **Visual Studio 2005**' in **Add Service Reference** seçeneği https için hazırlanmış olan servise ait proxy sınıfının üretilmesinde hatalar fırlatılmasına neden olmuştur. Burada çözüm olarak servis https' e taşınmadan önce proxy sınıfının üretilmesi sağlanabilir. İkinci bir çözüm yolu ise servisin kullandığı sözleşme ve uyarlama sınıfını içeren ayrı bir sınıf kütüphanesi üzerinden svcutil ile proxy ürettirmektir. Yada üçüncü bir yol olarak aşağıda olduğu gibi proxy sınıfı manuel olarak yazılabilir :)



Gerekli kodlar aşağıdaki gibi olacaktır;

```
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
[System.ServiceModel.ServiceContractAttribute(Namespace =
"http://www.bsenyurt.com/Cebirci")]
public interface CebirServisi
{
   [System.ServiceModel.OperationContractAttribute(Action =
"http://www.bsenyurt.com/Cebirci/Cebirci/Topla", ReplyAction =
"http://www.bsenyurt.com/Cebirci/Cebirci/ToplaResponse")]
   double Topla(double x, double y);
}

[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
public interface CebirServisiChannel : CebirServisi,
System.ServiceModel.IClientChannel
{
}

[System.Diagnostics.DebuggerStepThroughAttribute()]
[System.CodeDom.Compiler.GeneratedCodeAttribute("System.ServiceModel", "3.0.0.0")]
public partial class CebirServisiClient :
System.ServiceModel.ClientBase<CebirServisi>, CebirServisi
```

```
{
   public CebirServisiClient()
   {
   }

   public CebirServisiClient(string endpointConfigurationName) :
      base(endpointConfigurationName)
   {
   }

   public CebirServisiClient(string endpointConfigurationName, string remoteAddress) :
      base(endpointConfigurationName, remoteAddress)
   {
   }

   public CebirServisiClient(string endpointConfigurationName,
System.ServiceModel.EndpointAddress remoteAddress) :
      base(endpointConfigurationName, remoteAddress)
   {
   }

   public CebirServisiClient(System.ServiceModel.Channels.Binding binding,
System.ServiceModel.EndpointAddress remoteAddress) :
      base(binding, remoteAddress)
   {
   }

   public double Topla(double x, double y)
   {
      return base.Channel.Topla(x, y);
   }
}
```

İstemci tarafındaki app.config isimli konfigurasyon dosyasının içeriği ise aşağıdaki gibi olmalıdır.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <bindings>
         <wsHttpBinding>
            <binding name="CebirServiceBindingCfg">
               <security mode="TransportWithMessageCredential">
                  <transport clientCredentialType="None" />
                  <message clientCredentialType="UserName"/>
```

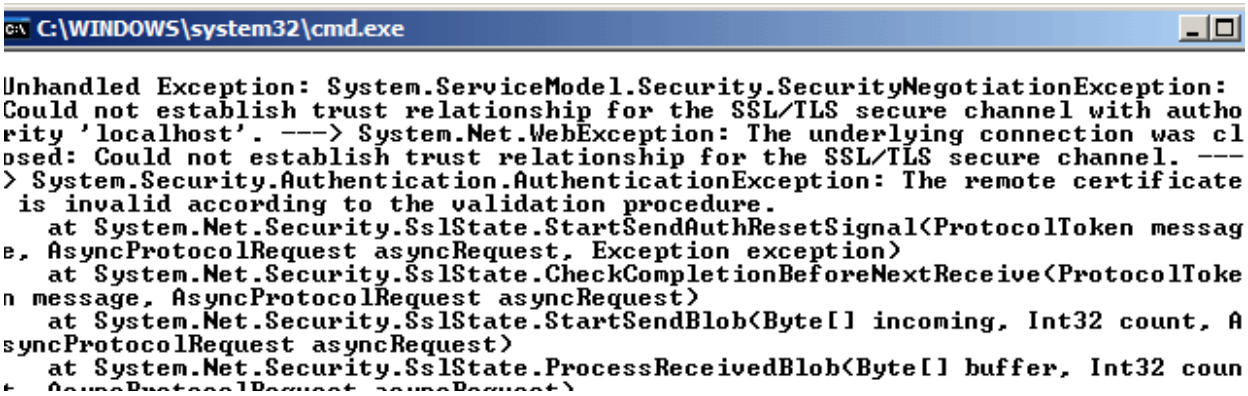
```
            </security>
          </binding>
        </wsHttpBinding>
      </bindings>
      <client>
        <endpoint address="https://localhost/CebirServisi/CebirService.svc" binding="
wsHttpBinding" bindingConfiguration="CebirServiceBindingCfg" contract="CebirServisi"
name="CebirServiceEndpoint">
        </endpoint>
      </client>
  </system.serviceModel>
</configuration>
```

Dikkat edilecek olursa istemci tarafında yer alan bağlayıcı tip**(Binding Type)** ayarlarında da **security** elementi de yer almaktadır ve servis tarafındaki ile aynı değerlere sahip olacak şekilde tasarlanmıştır. İstemci tarafında gerçekleştirilen bu ayarlardan sonra **Main** metodunun içeriği aşağıdaki gibi düzenlenebilir.

```
CebirServisiClient client = new CebirServisiClient("CebirServiceEndpoint");

client.ClientCredentials.UserName.UserName = "buraks";
client.ClientCredentials.UserName.Password = "123456.";

double sonuc=client.Topla(3, 4);
Console.WriteLine("Sonuc {0}",sonuc.ToString());
Console.ReadLine();
```

Dikkat edilecek olursa, istemci tarafına ait yetkinlik belgesini gönderirken kullanıcı adı ve şifre bilgileri verilmektedir. Bunun için proxy sınıfına ait nesne örneği üzerinden erişilen**ClientCrendetials** özelliği kullanılmaktadır. ClientCredentials özelliği **ClientCredentials** sınıfına ait nesne örneklerini işaret eder. Bu nesne örneği üzerinden **UserName**özelliği(property) ile **UserNamePasswordClientCredential** isimli sınıfa erişilebilir. UserNamePasswordClientCredential sınıfının **UserName** ve **Password** isimli iki özelliği bulunmaktadır ve bunlar servis tarafında tanımlı olan istemci hesaplarındakiler ile karşılaştırılmak üzere gönderilir.

Geliştirilen örnekte test amaçlı sertifika kullanılmasının oluşturduğu bazı negatif etkiler vardır. Bu nedenle yukarıdaki kod parçasını içeren istemci uygulama yürütüldüğünde çalışma zamanında(Runtime) **SecurityNegotiationException** tipinden bir istisna mesajı alınması kuvvetle muhtemeldir.



Az öncede belirtildiği gibi bu istisnanın sebebi gerçek sertifikaların kullanılmayışıdır. Yaptığım araştırmalarda bu tip test sertifikalarının kullanıldığı senaryolar için Microsoft geliştiricileri tarafından yazılmış bir sınıf olduğunu tespit ettim. Bu sınıf yardımıyla yukarıdaki istisna mesajını atlatmak ve yerel makinelerde testleri başarılı bir şekilde gerçekleştirmek mümkündür. Sınıfın içeriği Microsoft tarafından aşağıdaki gibi geliştirilmiştir.

PermissiveCertificatePolicy isimli sınıfın içeriği;

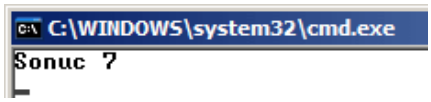
```
using System.Security.Cryptography.X509Certificates;
using System.Net;

// Microsoft' tan alıntıdır.
class PermissiveCertificatePolicy
{
   string subjectName;
   static PermissiveCertificatePolicy currentPolicy;
   PermissiveCertificatePolicy(string subjectName)
   {
      this.subjectName = subjectName;
      ServicePointManager.ServerCertificateValidationCallback +=new
System.Net.Security.RemoteCertificateValidationCallback(RemoteCertValidate);
   }

   public static void Enact(string subjectName)
   {
      currentPolicy = new PermissiveCertificatePolicy(subjectName);
   }

   bool RemoteCertValidate(object sender, X509Certificate cert, X509Chain chain,
System.Net.Security.SslPolicyErrors error)
   {
      if (cert.Subject == subjectName)
      {
```

```
            return true;
        }
        return false;
    }
}
```

Bu sınıfın sadece test sertifikalarının olduğu senaryolarda ele alınmasının, gerçek sertifikaların kullanıldığı senaryolarda kullanılmamasının Microsoft tarafından önerildiğini de hatırlatalım. **PermissiveCertificatePolicy** sınıfı istemci tarafına eklendikten sonra Main metodunda proxy sınıfı örneklenmeden önce ele alınması gerekmektedir.

```
PermissiveCertificatePolicy.Enact("CN=TestSertifika-HTTPS-Server");
CebirServisiClient client = new CebirServisiClient("CebirServiceEndpoint");
```

Burada dikkat edilirse **Enact** metoduna sunucu tarafında oluşturulan test sertifikasının adı parametre olarak verilmiştir. Şimdi uygulama bu haliyle test edilirse aşağıdaki ekran görüntüsünde olduğu gibi başarılı bir şekilde çalıştığı görülür.



Kullanıcı adı ve şifre bilgilerinde hata olması halinde ise uygulama **MessageSecurityException** tipinden bir istisna fırlatacaktır. Bir başka deyişle istemci servis tarafından doğrulanmamış olacaktır.

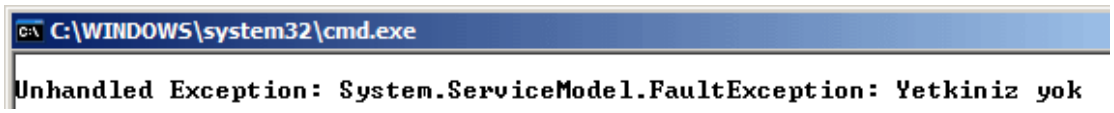
Gelelim rollerin ne şekilde ele alınabileceğine. örneğin Topla isimli metodu sadece Personel rolünde olanların ele alabileceği bir senaryo göz önüne alalım. Buna göre servis tarafındaki Aritmetik sınıfı içerisinde yer alan Topla metodunda aşağıdaki düzenlemeleri yapmak yeterli olacaktır.

```
public double Topla(double x, double y)
{
    IIdentity ulasanKullanici = ServiceSecurityContext.Current.PrimaryIdentity;
    if (Roles.IsUserInRole(ulasanKullanici.Name, "Personel"))
        return x + y;
    else
        throw new FaultException("Yetkiniz yok");
}
```

Buradaki kodlara göre çalışma zamanında metoda çağrıda bulunan istemciye ait kullanıcı bilgileri **ServiceSecurityContext** sınıfı üzerinden ele alınabilir. Elde edilen referans yetki belgesi gönderen kullanıcının ad bilgisinide içerecektir.

***NOT : ServiceSecurityContext** ve **Roles** sınıflarının kullanılabilmesi için **System.Security.Principal** ve **System.Web.Security** isim alanlarının uygulamaya dahil edilmesi gerekir.*

Bundan sonra tek yapılan, **Roles** sınıfının **IsUserInRole** metodunu kullanarak, güncel içerikteki kullanıcının Personel rolünde olup olmadığının tespit edilmesidir. Eğer kullanıcı Personel rolünde ise hesaplama yaptırılır. Aksi halde bir **FaultException** üretilip istemci tarafına doğru fırlatılır. İstemci tarafında bulents isimli kullanıcı ile Topla metoduna erişmek istediğimizde çalışma zamanında FaultException üretildiğini görürüz.



Ancak istemci tarafından, Personel rolündeki bir kullanıcı ile servise ulaşırsak Topla metodu sorunsuz bir şekilde çalışacaktır. Bunu buraks kullanıcısı ile deneyebiliriz. Nitekim buraks isimli kullanıcı Personel rolü içerisinde tanımlanmıştırç

Böylece geldik bir makalemizin daha sonuna. Bu tamamlayıcı makalemizde internet üzerinden **https** ile erişilebilen bir servis üzerinde, Asp.Net rol ve üyelik yönetimini**(Role and Membership Management)** kullanarak doğrulama**(authentication)** ve yetkilendirme**(authorization)** işlemlerinin nasıl yapılabileceğini incelemeye çalıştık. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

SSLClient.zip (25,30 kb) *(Dosya boyutunun küçük olması için veritabanı çıkartılmıştır. örneği denerken sertifika eklemeyi ve veritabanını oluşturup örnek kullanıcılar dahit etmeyi unutmayınız.)*

WCF - Internet Üzerinden Güvenliği Sağlamak - 1 (2007-07-03T21:14:00)

wcf,

Windows Communication Foundation ile geliştirilen dağıtık mimari uygulamalarında istemci(client) ve servis(service) arasındaki güvenliği temel olarak mesaj seviyesinde**(Message Level)** ve iletişim seviyesinde**(Transport Level)** sağlayabileceğimizden daha önceki yazılarımızda bahsetmiştik. Söz konusu seviyelerden hangisi tercih edilirse edilsin, istemcilerin servisi kullanırken doğrulanmaları**(authenticate)** ve gerekli işlemleri yapabilmeleri için yetkilerine bakılmaları**(authorization)** gerekir. Windows Communication Foundation, istemcileri doğrulamak(authenticate) adına altı farklı yol kullanılmasına olanak tanımaktadır. Bunlar aşağıdaki tabloda görüldüğü gibidir.

Windows Communication Foundation Doğrulama(Authenticate) Yolları	
Yol	**Açıklama**

Windows	İstemcilerin doğrulanması için tipik olarak servis tarafında yer alan windows hesaplarından **(Windows Accounts)** faydalanılır. Daha çok **Kerberos** veya **NTLM** gibi sistemler ele alınır. Bu teknik intranet tabanlı dağıtık mimari uygulamalarında oldukça işe yaramakta ve tercih edilmektedir.
Kullanıcı Adı/ Şifre (Username/Password)	İstemciler servis tarafına kullanıcı adı**(Username)** ve şifre**(Password)** bilgisi gönderir. Kullanıcı hesap bilgileri servis tarafında çoğunlukla bir veritabanı sistemi üzerinde tutulur. WCF servislerinin IIS üzerinde tutulabildiği göz önüne alındığında Asp.Net ile gelen Membership veritabanlarını kullanmak yaygın olarak tercih edilebilir. WCF servisinin **HTTP** üzerinden yayınlandığı internet tabanlı senaryolarda sıklıkla kullanılabilir.
X509	İstemciler servis tarafına kendilerini geçerli bir sertifika yardımıyla tanıtırlar.
Özel (Custom)	Doğrulama**(Authenticate)** işlemleri için özelleştirilmiş yapılar kullanılır. **Biometric** buna örnek olarak verilebilir. Söz gelimi istemcilerin parmak izlerine veya göz retinalarına göre doğrulanması gibi mekanizmalar var olan yapıların özelleştirilmesi ile mümkün olabilir.
Issued Token	Bu doğrulama tekniğine verilebilecek en güzel örnek .Net Framework 3.0 ile gelmiş olan **CardSpace** mimarisidir.
Yok (None)	İstemcilerin tamamı doğrulanır. Bir anlamda da servise herkesin erişebilmesi sağlanmış olunur.

İstemcilerin kendilerini servis tarafına doğrulatmaları esnasında kullanıcı bilgilerinin saklandığı bazı ortamlar söz konusudur. Windows hesaplarının**(account)** tutulduğu sistemler bellidir. Ancak bunun dışında özellikle internet tabanlı senaryolarda ele alınabilecek şekilde veritabanı**(database)** kullanımıda mümkündür. WCF mimarisinde servis tarafı IIS üzerinde barındırılabilmektedir. Bu sebepten dolayı kullanıcılara ait hesap bilgileri için Asp.Net 2.0 ile birlikte gelen üyelik yönetim sisteminden **(Membership Management API)** faydalanılabilir. Elbetteki windows veya veritabanı dışında özel depolama sistemleride söz konusu olabilir.

Doğrulanan kullanıcıların yetkilerine bakılmadan işlem yapılması tam olarak güvenliğin sağlanamadığı anlamınada gelir. Dolayısıyla servis tarafında yer alan operasyonlarda doğrulanan kullanıcıların rollerine, başka bir deyişle yetkilerine bakılarak ilerlenilmesinde fayda vardır. WCF mimarisinde güvenlik denince aklan gelenler sadece authentication ve authorization olmamalıdır. Aslında Windows Communication Foundation, maksimum güvenliğin sağlanabilmesi için üç farklı ilkenin var olmasını gerektirmektedir. Bunlar, mesaj bütünlüğü**(Message Integrity)**, mesaj mahremiyeti**(Message Privacy)** ve müşterek doğrulama**(Mutual Authentication)** ilkeleridir.

Maksimum Güvenlik için Sağlanması Gereken İlkeler	
İlke	**Açıklama**
Mesaj Bütünlüğü (Message Integrity)	Mesaj bütünlüğü ilkesine göre istemciden servise doğru gidecek olan mesajın başkaları tarafından kurcalanıp bozulamaması gerekmektedir. Bir başka deyişle bu ilke, kötü niyetli kullanıcıların**(malicious users)** arada hareket eden mesajların bütünlüğünü bozacak şekilde hamlelerde bulunamamasının sağlanmasını gerektirir.
Mesaj Mahremiyeti (Message Privacy)	Bu ilke istemci ve servis arasında hareket eden mesajların gizliliğinin sağlanmasını gerektirir. Bir başka deyişle kötü niyetli kullanıcılar çeşitli 3ncü parti yazılımları kullanarak mesaj içeriklerini okuyamamalıdır. Bu ilke aynı zamanda **Message Integrity** ilkesinin tamamlayıcısı olarak da düşünülebilir.
Müşterek Doğrulama (Mutual Authentication)	İstemcilerin doğru servis ile haberleşmesini, istemciden gelen ehliyet**(Crendential)** bilgilerinin servis tarafında doğrulanmasını ve bunlara ek olarak tekrarlı atakların**(replay attacks)** bertaraf edilebilmesinin sağlanmasını hedefleyen ilkedir.

WCF için söz konusu olan iletişim güvenlik sistemleri yukarıdaki ilkeleri göz önüne alır ve buna göre maksimum güvenliğin sağlanabilmesini kolaylaştırır. Buna göre kendi özel güvenlik sistemlerimizi geliştirmek istediğimizde burada bahsedilen ilkelere uygun olacak şekilde hareket edilmesi gerekir.

WCF mimarisinde iletişim güvenliğini sağlayabilmek adına kullanılabilecek 5 farklı iletişim güvenlik tekniği bulunmaktadır.
Bunlar **None**, **Transport**, **Message**, **Mixed** ve **Both** tekniklerdir. Message seviyesinde iletişim güvenliği tekniğini daha önceki yazımızda ele almıştık. Transport tekniğini ise bu yazımız ile birlikte ele almaya çalışacağız. Gelelim diğer modlara.**Mixed** modda Message Integrity ve Privacy ilkelerini sağlamak için iletişim**(Transport)** seviyesinde güvenlik tekniği kullanılır. İstemci ehliyetlerini(Client Credential) korumak içinse mesaj**(Message)**seviyesinde güvenlik tekniği ele alınır. **Both** iletişim güvenlik tekniğinde mesajlar mesaj seviyesinde güvenlik tekniğine göre şifrelenirken, istemciden servis tarafında gönderilirken **Transport**tekniğine göre aktarılır.

WCF mimarisinin pek çok konusunda olduğu gibi bazı işlemleri gerçekleştirmek için ele alınması gereken oldukça fazla faktör vardır. Güvenlik teknikleri ile var olan bağlayıcı tipler**(binding types)** arasındaki durumda aynıdır. Bu sebepten dolayı aşağıdaki tablonun bilinmesinde ve ele alınmasında fayda vardır.

Bağlayıcı Tipler ve İletişim Güvenlik Teknikleri Arasındaki İlişki					
Bağlayıcı Tip (Binding	**Transport**	**Message**	**Mixe**	**Both**	**None**

Type)			d		
NetTcpBinding	Evet(Varsayılan)	Evet	Evet	Hayır	Evet
NetPeerTcpBinding	Evet(Varsayılan)	Evet	Evet	Hayır	Evet
NetNamedPipeBinding	Evet(Varsayılan)	Hayır	Hayır	Hayır	Evet
NetMsmqBinding	Evet(Varsayılan)	Evet	Hayır	Evet	Evet
WSHttpBinding	Evet	Evet(Varsayılan)	Evet	Hayır	Evet
WSFederationHttpBinding	Hayır	Evet(Varsayılan)	Evet	Hayır	Evet
WSDualHttpBinding	Hayır	Evet(Varsayılan)	Hayır	Hayır	Evet
BasicHttpBinding	Evet	Evet	Evet	Hayır	Evet(Varsayılan)

Bu tabloda hangi bağlayıcı tipin hangi iletişim güvenlik tekniklerini desteklediği belirtilmektedir. Örneğin Both tekniğini sadece **NetMsmqBinding** bağlayıcı tipi desteklerken diğerleri desteklemez. Dolayısıyla istemci ve sunucu arasındaki güvenliğin nasıl sağlanacağına karar verildikten sonra uygun bağlayıcı tiplerin göz önüne alınması için yukarıdaki tabloda yer alan bilgilerden faydalanılabilir.

Bu bölümde geliştirilmeye başlanacak olan örnekte iletişim seviyesinde güvenlik (transport level security) tekniği kullanılacak olup yazının ikinci bölümünde istemcilere ait ehliyet bilgilerini servis tarafında kontrol ederken **Sql Membership Provider** ve **Sql Role Provider API'** leri ele alınacaktır. Örneğe geçmeden önce iletişim seviyesi güvenlik tekniğinde seçilen bağlayıcı tipe göre hangi ehliyet modellerinin kullanılabileceğinin bilinmesinde fayda vardır. Dolayısıyla göz önünde bulundurulması gereken bir tablo daha karşımıza çıkmaktadır.

Bağlayıcı Tipler, İletişim Seviyesinde Güvenlik Tekniği ve Doğrulama Modelleri Arasındaki İlişki				
Bağlayıcı Tip (Binding Type)	**Windows**	**Username/Password**	**X509**	**None**
NetTcpBinding	Evet(Varsayılan)	Hayır	Evet	Evet

NetPeerTcpBinding	Hayır	Evet	Evet	Hayır
NetNamedPipeBinding	Evet(Varsayılan)	Hayır	Hayır	Hayır
NetMsmqBinding	Evet(Varsayılan)	Evet	Hayır	Evet
WSHttpBinding	Evet(Varsayılan)	Evet	Evet	Evet
WSFederationHttpBinding	X			
WSDualHttpBinding				
BasicHttpBinding	Evet	Evet	Evet	Evet(Varsayılan)

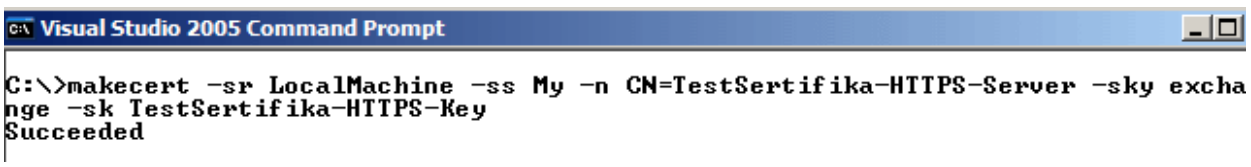
Dikkat edilmesi gereken noktalardan birisi **WSFederationHttpBinding** ve **WSDualHttpBinding** bağlayıcı tiplerinin iletişim seviyesinde güvenlik tekniği söz konusu olduğunda hiç bir doğrulama modelini desteklemediğidir. Bu noktaları açıklığa kavuşturduktan sonra nihayetinde bir örnek geliştirmeye başlayarak internet tabanlı WCF uygulamalarında iletişim seviyesinde güvenliği nasıl sağlayabileceğimizi incelemeye başlayabiliriz.

İletişim seviyesinde güvenlik söz konusu olduğundan WCF servisinin IIS üzerinde barındırılması ve **HTTPS** protokolünü baz alarak hizmet verebilmesinin sağlanması gerekmektedir. Ancak öncesinde güvenli iletişim kanalı kullanımı için(bir başka deyişle https üzerinden hizmet vermek için) hayali bir sertifika oluşturulmalıdır. Hayali sertifikaları oluşturmak için**Makecert.exe** aracı kullanılabilir. Bu araç tamamen test amaçlı **X509** sertifikalarının üretilmesini sağlamaktadır.

NOT : *MakeCert.exe aracı ile ilgili detaylı bilgi için http://msdn2.microsoft.com/en-us/library/bfsktky3(VS.80).aspx adresinden bilgi alınabilir.*

Test sertifikası oluşturmak için **Visual Studio 2005 command prompt** üzerinden aşağıdaki komutun yazılması yeterlidir.

C:\>makecert -sr LocalMachine -ss My -n CN=TestSertifika-HTTPS-Server -sky exchange -sk TestSertifika-HTTPS-Key

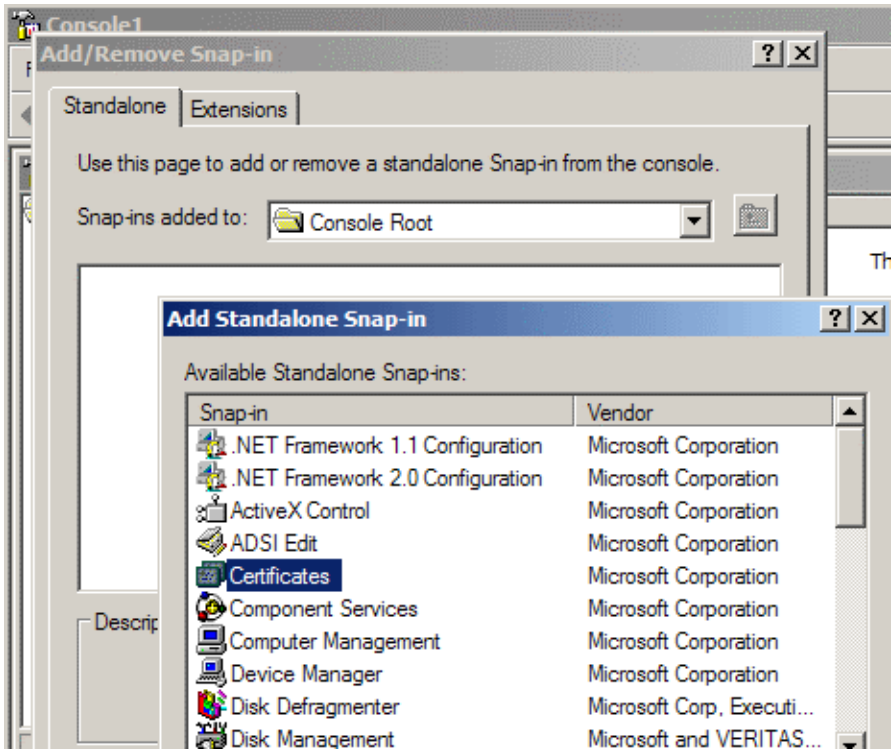


Succeeded mesajı görüldüğü takdirde sertifika başarılı bir şekilde oluşturulmuş demektir. *(Makalemizin amacı Makecert aracını tanımak olmadığından aracın parametre detayları üzerinde durulmayacaktır.)* Oluşturulan sertifikayı görmek için **Microsoft Management Console**' dan faydalanılabilir.

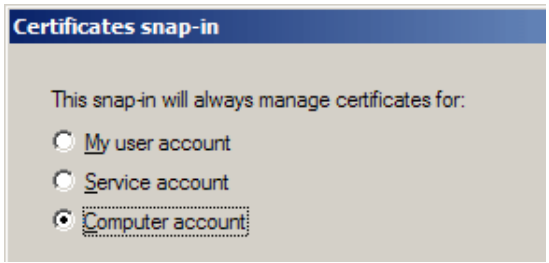
Diğer taraftan oluşturulan sertifikanın iletişim kanalı ile *(ki burada söz konusu olan yerel makinedeki 8000 numaralı Http portudur)* ilişkilendirilmesi gerekir. Bu işlem için **HttpCfg.exe** aracından yararlanılabilinir.

NOT : *Windows XP' de* ***HttpCfg.exe*** *aracını kullanabilmek için* ***Windows XP Service Pack 2 Support Tools*** *' u indirmek gerekebilir.*

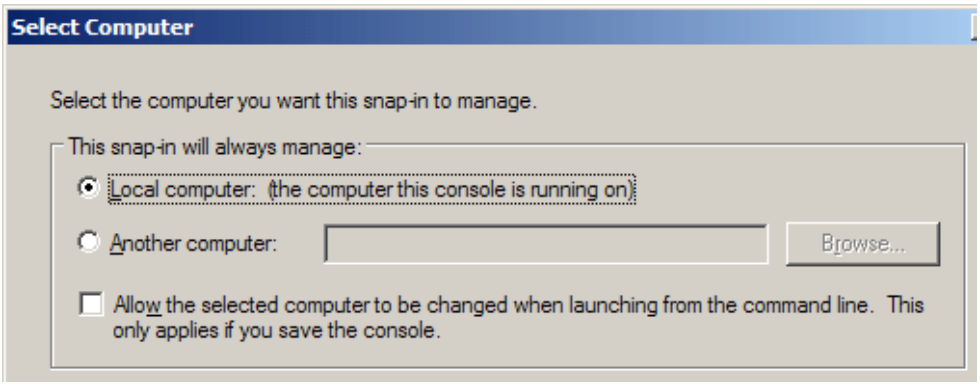
Httpcfg.exe aracı parametre olarak oluşturulan sertifikaya ait parmak damgasını**(thumbprint)** kullanır. Parmak damgasını elde edebilmek için öncelikli olarak **Microsoft Management Console**' da aşağıdaki ekran görüntüsünde yer aldığı gibi **Add/Remove Snap In** seçeneğinden **Certificates** işaretlenir.



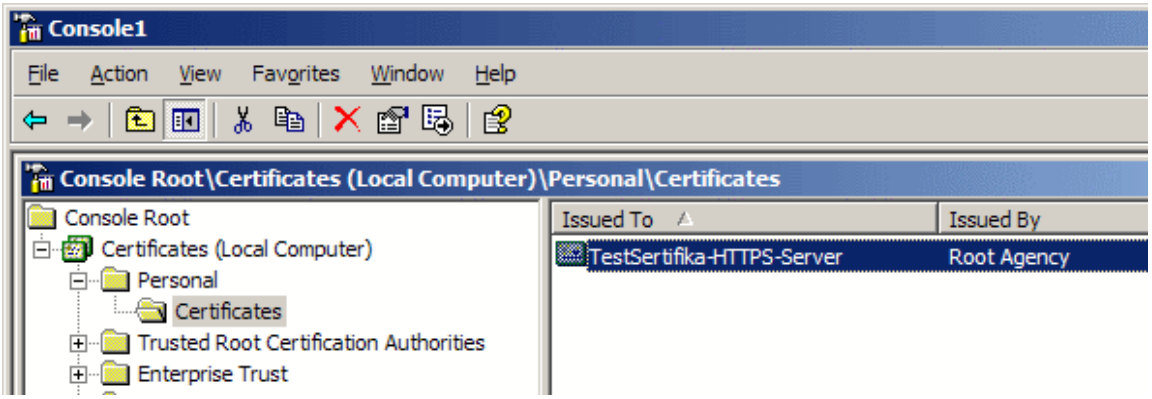
Certificates seçildikten sonra sıradaki adımda **Computer Account** seçilir.



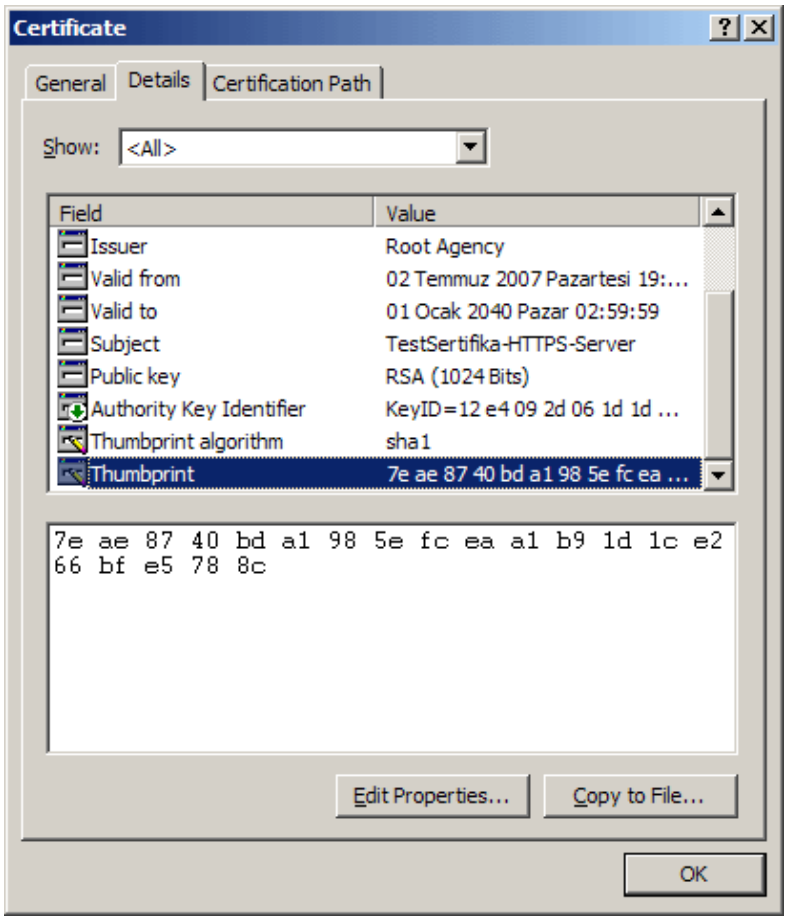
Sonraki adımda snap-in yönetimini üstlenecek olan bilgisayar seçilir. Bu varsayılan olarak yerel bilgisayarı (Local Computer) işaret etmektedir.



Bunun sonucunda oluşturulan test sertifikası aşağıdaki ekran görüntüsünde yer aldığı gibi **Personal->Certificates** klasörü altında görülecektir.



Buradanda sertifikanın detaylarına geçilerek **Thumbprint** alanının değeri öğrenilebilir.



Artık **HttpCfg** aracı yardımıyla sertifikanın port ile ilişkilendirilmesi sağlanabilir. Httpcfg.exe aracı yardımıyla sertifikaya ait parmak damgasının**(thumbprint)**, port ile ilişkilendirilmesini sağlamak için aşağıdaki komut, **Xp Support Tools Command Prompt** üzerinden çalıştırılmalıdır.

C:\>httpcfg set ssl -i 0.0.0.0:8000 h7eae8740bda1985efceaa1b91d1ce266bfe5788c

HttpSetServiceConfiguration Completed with 0 mesajı görüldüğü takdirde operasyonun başarılı bir şekilde tamamlandığı anlaşılabilir.

HttpSetServiceConfiguration completed with 183 gibi bir mesaj alınması hata oluştuğu anlamına gelir. Hata mesajı zaten var olan dosyaya tekrardan yazılmak istenmesinden kaynaklanmaktadır.***(ERROR_ALREADY_EXISTS 183 Cannot create a file when that file already exists.)*** Bu nedenle sertifika **unload** edilebilir. Sertifikanın Unload edilmesi için komut satırından

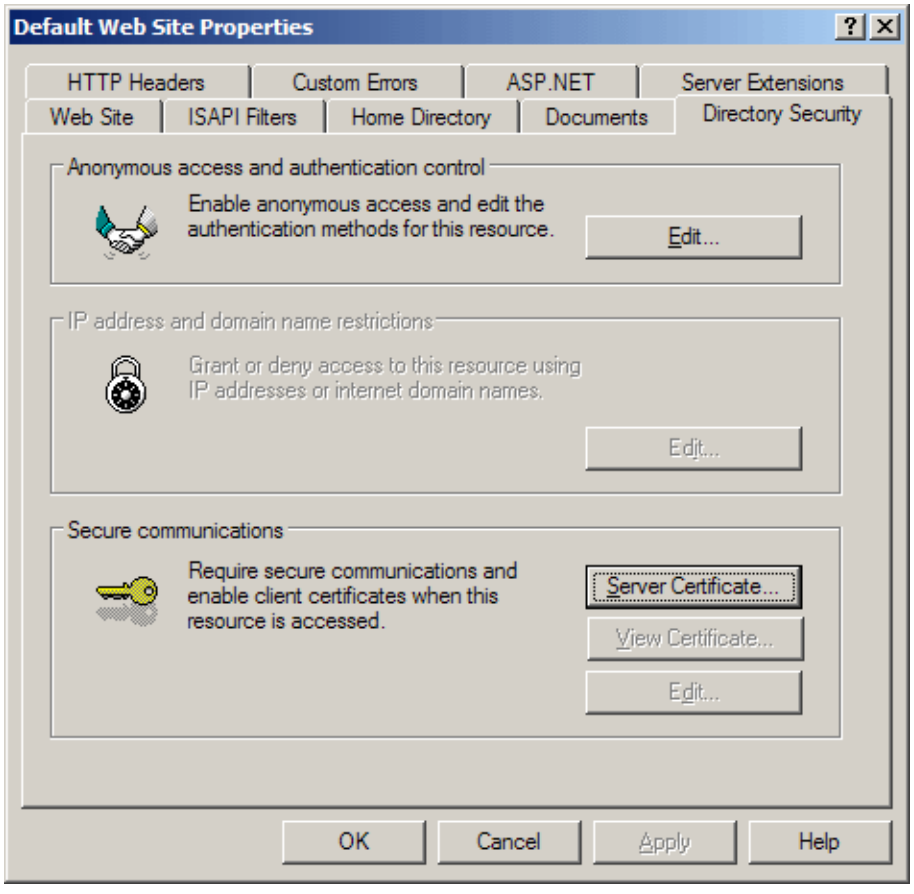
C:\>Httpcfg delete ssl /i 0.0.0.0:8000

yazılması yeterlidir. Buna göre yerel makineye ait 8000 port numarası için tanımlanmış olan ssl sertifikalarına ait bildirimler silinecektir. İstenirse, Httpcfg.exe aracı yardımıyla IIS üzerinde yüklenmiş olan sertifikaları görmek için komut satırından

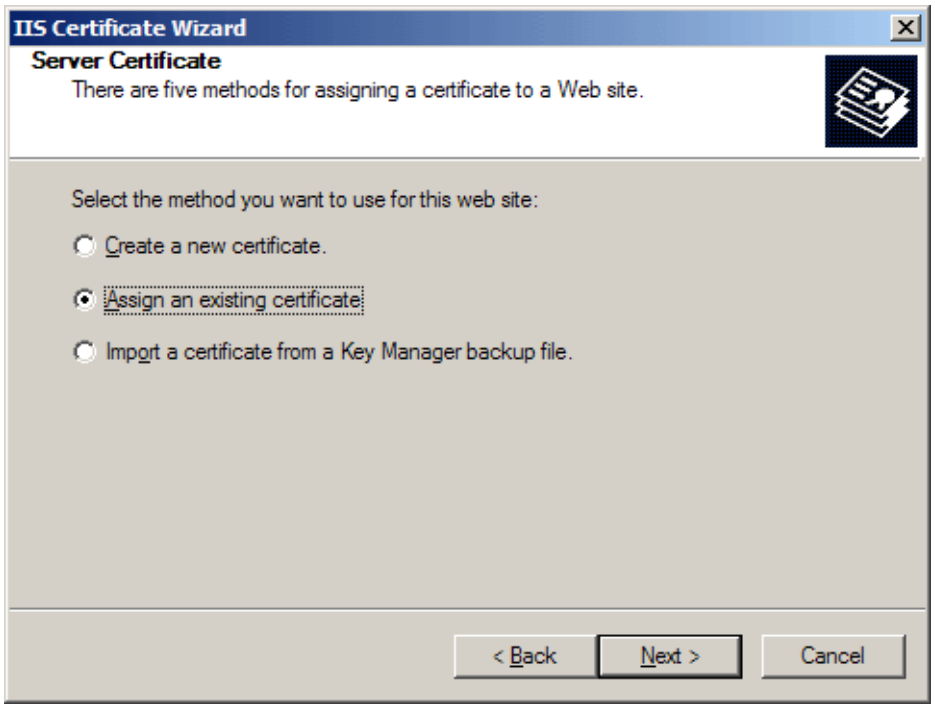
C:\>Httpcfg query ssl

yazılması yeterlidir.

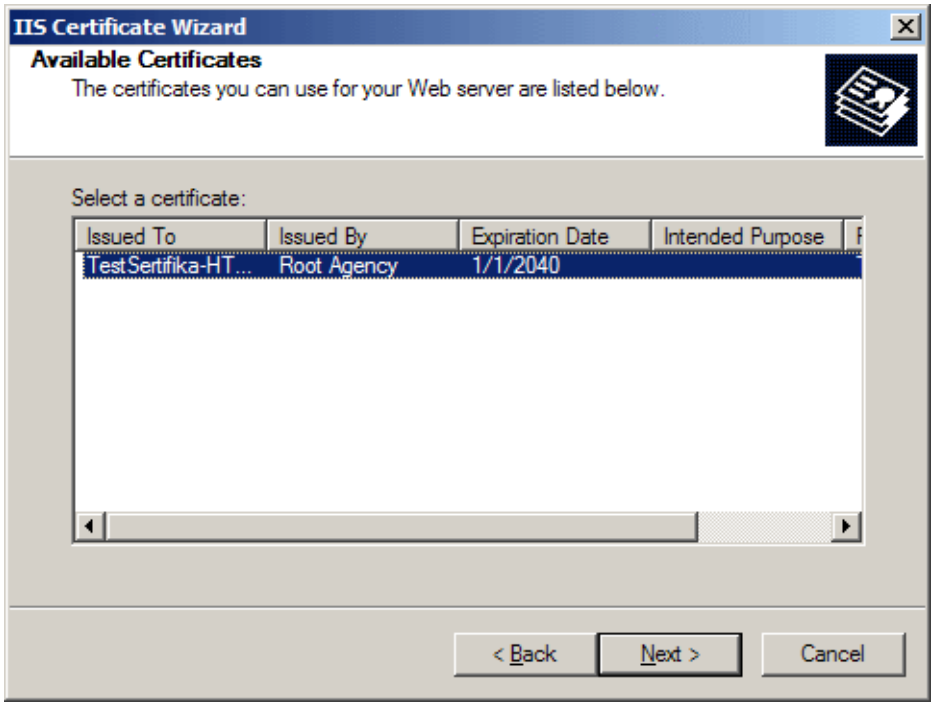
Oluşturulan sertifikanın IIS üzerinde yer alacak WCF uygulaması tarafından kullanılabilmesi için öncelikli olarak bildirilmesi gerekir. İzleyen adımlarda Windows XP işletim sistemi üzerinde yer alan IIS 5.1 için söz konusu bildirim işleminin nasıl yapılacağı ele alınmaktadır. Öncelikli olarak **Internet Information Services** üzerinden **Default Web Site** sağ tıklanıp özelliklerden(Properties) **Directory Security** kısmına geçilir ve buradan **Server Sertificate** düğmesi tıklanır.



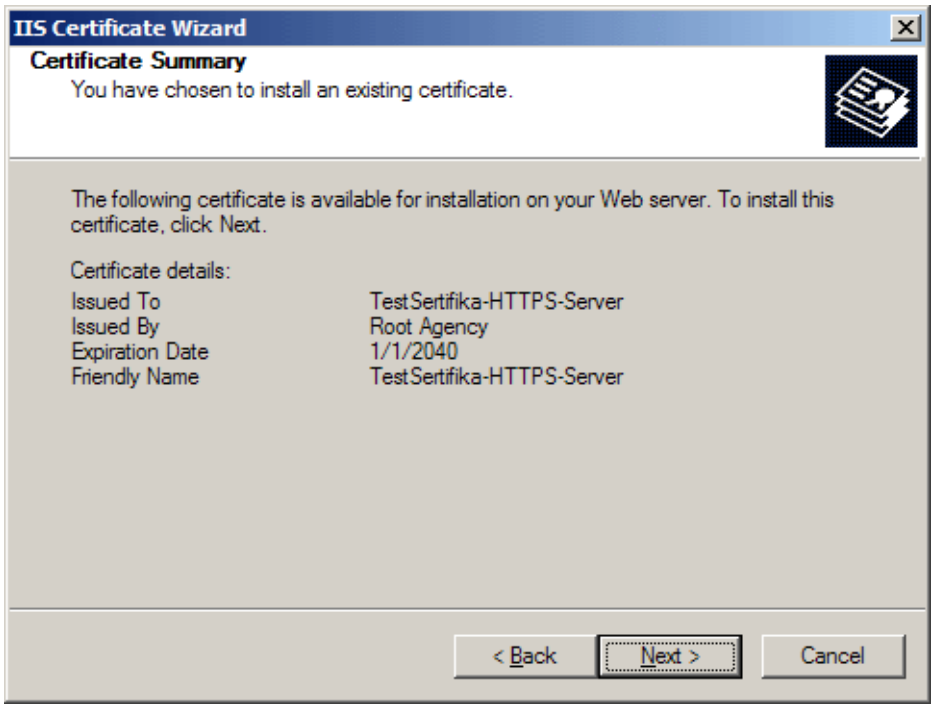
Burada da **Next** ile ilerlenip **Assign an existing certificate** seçeneği işaretlenir ve devam edilir. Böylece daha önceki adımlarda yüklenmiş olan sertifikanın kullanılabilmesi sağlanmış olmaktadır.



İzleyen adımda az önce yüklenmiş olan sertifika görülebilir. Yüklü olan başka sertifikalarda söz konusu olabilir. Uygun olan sertifika seçilerek ilerlemeye devam edilir.

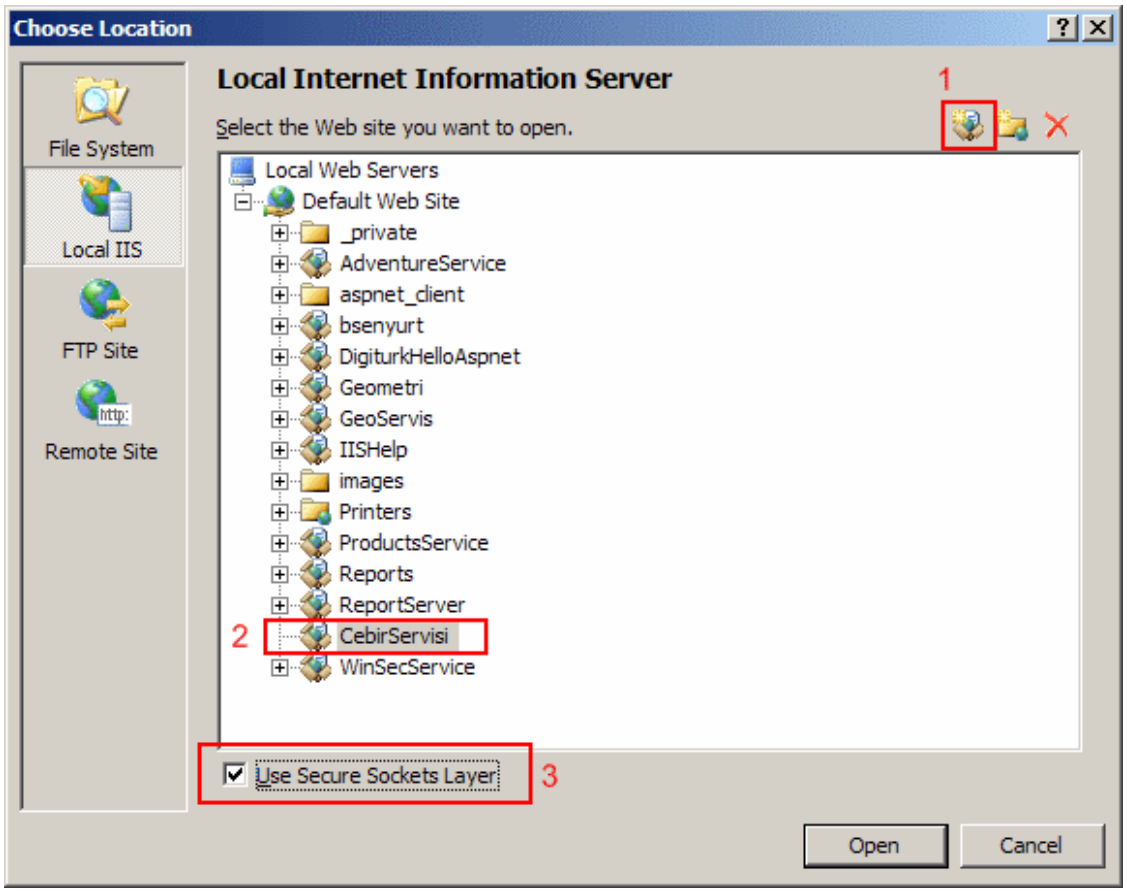


Eğer sertifika yükleme işlemi başarılı bir şekilde gerçekleştirildiyse aşağıdaki ekran görüntüsü elde edilmelidir.

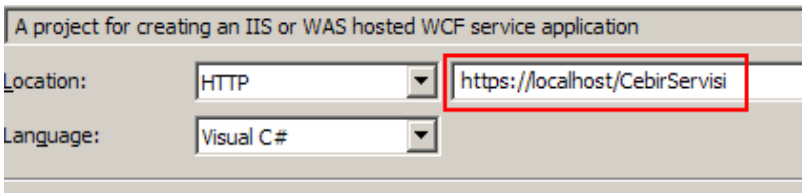


Bundan sonraki tüm adımlar onaylanarak işlemler tamamlanır.

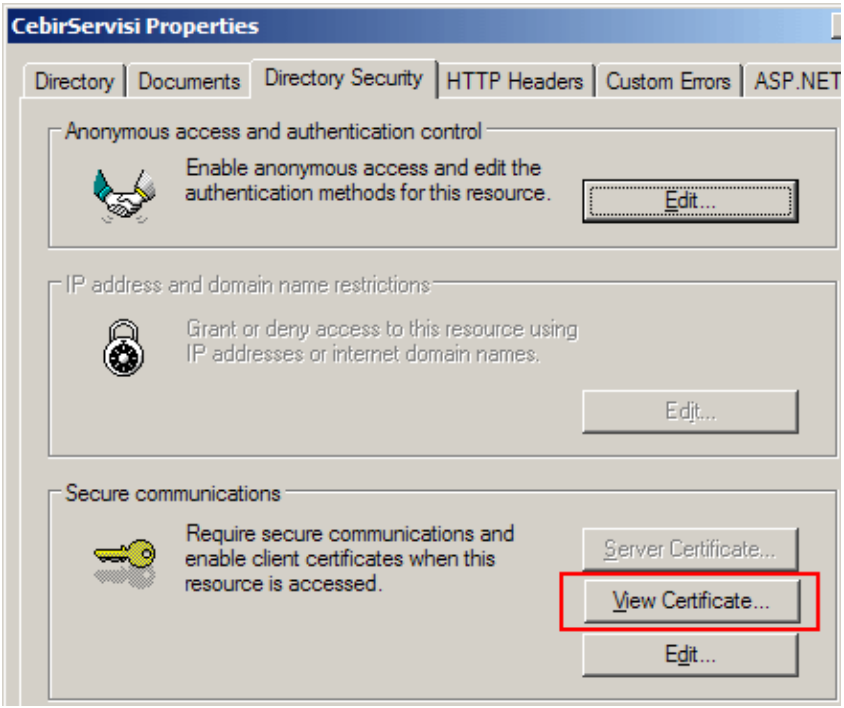
Artık WCF Service uygulamasının yazılmasına başlanabilir. Bu amaçla **Visual Studio 2005** üzerinden **New Web Site** seçeneği ile yeni bir web sitesi açılır ve **WCF Service** şablonu seçilir. Burada önemli olan nokta IIS üzerinde açılacak olan sanal klasör için **Secure Socket Layer** kullanılacağını belirtmektir. Bunun için aşağıdaki ekran görüntüsünde olduğu gibi lokasyon seçilirken **Use Secure Sockets Layer** seçeneğinin işaretlenmesi yeterlidir.



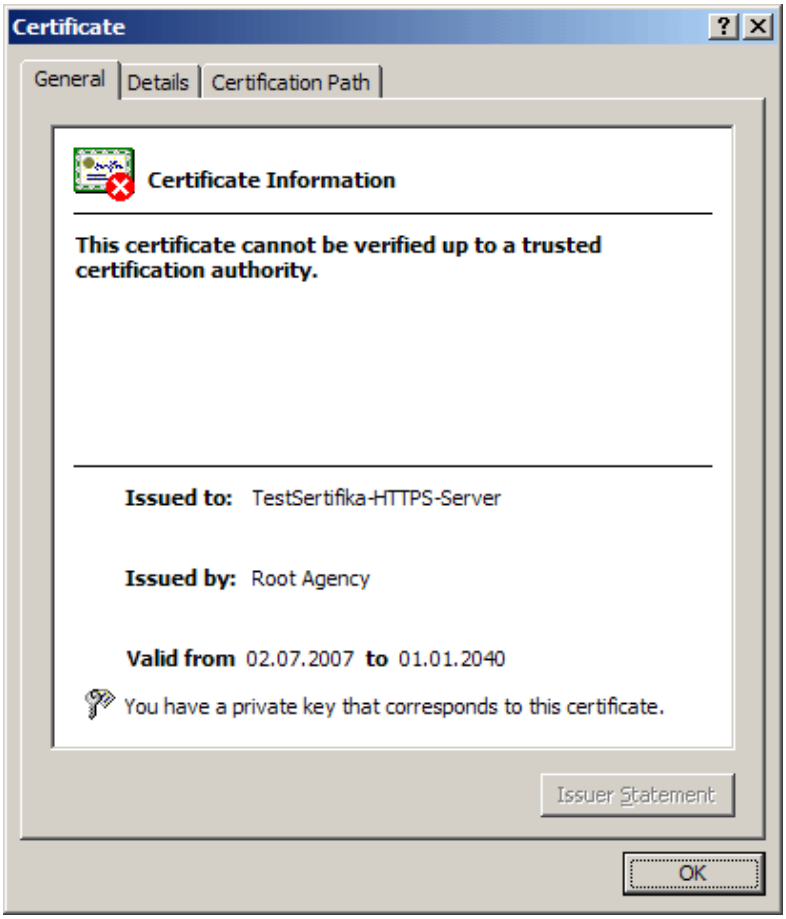
Örnek servis CebirServisi olarak isimlendirilmiştir. Bu işlemin ardından aşağıdaki ekran görüntüsüden yer aldığı gibi ilgili web adresinin başında **HTTP** yerine **HTTPS** yazıldığı görülecektir.



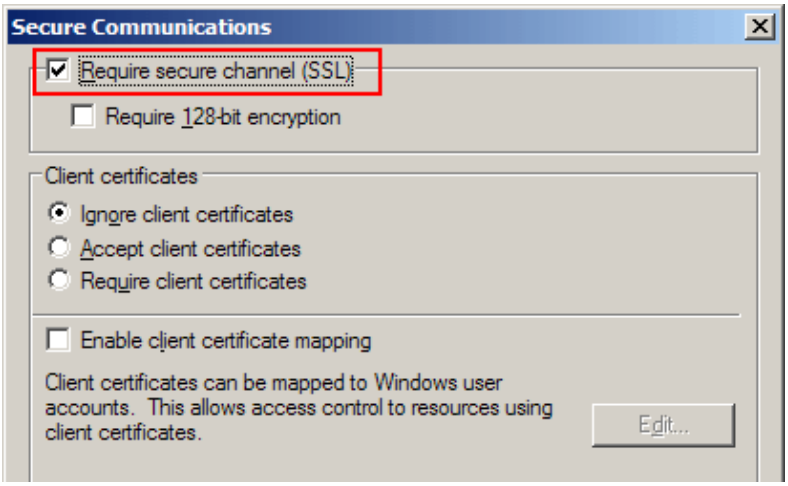
Söz konusu adımlar tamalandıktan sonra oluşturulan CebirServisi isimli servisin iletişim sırasında **HTTPS** protokolüne göre çalışabilmesi için, IIS üzerinden gerekli hazırlıkların yapılması gerekmektedir. Bunun için ilk olarak IIS altında yeni açılmış olan CebirServisi sanal klasörünün özelliklerinden **Directory Security** kısmında gidilir. Yazının başlarında**Default Web Site**' a hazırlanmış olan hayali sertifika tanıtıldığından, **View Certificate** kısmı kullanılabilir haldedir.



View Certificate düğmesi tıklandıktan sonra açılan **Certificate** penceresinde, yüklenmiş olan test sertifikasına ait bilgiler aşağıdaki ekran görüntüsünde olduğu gibi görülebilir.



CebirServisi için yapılması gereken bir diğer işlem ise, **Secure Communications** kısmında yer alan **Edit** düğmesine tıkladıktan sonra çıkan ekrandan, **Require Secure Channel(SSL)**seçeneğini işaretlemektir. Böylece servis ile güvenli iletişimin sağlanması için SSL gerekliliği bildirilmiş olunur.



Son olarak yine **Directory Security** kısmından **Anonymous Access and Authentication Control** içerisinde yer alan **Edit** düğmesine tıklanıp açılan pencerede, **Integrated Windows Authentication** seçeneği kaldırılmalı ve **Basic Authentication** işaretlenmelidir.

Yazı dizisinin bu ilk bölümünde WCF Servisi için IIS üzerinde gerekli sertifika bildirimlerinin nasıl yapılabileceğini incelemeye çalışırken hayali bir sertifika için **Makecert** ve **Httpcfg**araçlarını nasıl kullanabileceğimizide görmeye çalıştık. Bunların dışında iletişim seviyesinde güvenlik söz konusu olduğunda bağlayıcı tiplerin ve doğrulama modlarının nasıl ele alınması gerektiği üzerinde durduk. Böylece geldik bir makalemizin daha sonuna. Sonraki makalemizde geliştirilen örneği tamamlamaya çalışacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

WCF - Transaction Yönetimi - 2 (2007-06-28T18:43:00)

wcf,

Windows Communication Foundation için transaction yönetimi ile ilgili bir önceki makalemizde teorik bilgiler üzerinde durmaya çalışmıştık. Bu makalemizde ise, transaction yönetimi için gerekli materyalleri toplamaya devam edecek ve bir örnek üzerinde konuyu daha net bir şekilde anlamaya çalışacağız. örneği geliştirmeden önce özellikle servis ve metod seviyesinde bilinmesi gerekenler olduğunu belirtelim. özellikle servis nesnesi için çalışma zamanı davranışlarını belirlemek adına **ServiceBehavior** niteliğinin bazı özelliklerini kullanmak gerekmektedir. Benzer şekilde operasyonların transaction ile ilişkili çalışma zamanı davranışlarını belirlemek içinde, **OperationBehavior** niteliğinin özelliklerinden faydalanılmaktadır.

ServiceBehavior ve OperationBehavior dışında transacation yönetimi için kullanılan bir diğer nitelik**(attribute)** ise, **TransactionFlow**' dur. Bu nitelik servis tarafındaki metodlara uygulanabilen türdedir. TransactionFlow niteliği temel olarak bir servisin dış servisler ile olan ilişkilerinde servisler arasında akan transaction' ların operasyon kademesinde kabul edilme seviyelerinin belirlenmesinde kullanılır. Söz konusu seviyeler **TransactionFlowOption** isimli enum sabiti tarafından belirlenebilir. TransactionFlowOption enum sabitinin değerleri ve nasıl bir akışa izin verdikleri aşağıdaki tabloda gösterilmektedir.

TransactionFlowOption Değeri	**Açıklaması**
NotAllowed	WCF çalışma zamanı (runtime) servise akan transact
Allowed	Transaction istemci tarafında (istemci güncel servisin zamanı(runtime) yeni bir tane oluşturur.
Mandatory	Söz konusu operasyonun yürütülebilmesi için istemci ait bilgi SOAP Header (SOAP zarfının başlık kısmın

Bir servisin başka servis veya istemcilerden gelecek transaction akışına izin verip vermemesi durumlarını kontrol etmek için ilgili bağlayıcı tiplerin (binding types) **transactionFlow** elementi kullanılır. Varsayılan olarak transaction flow seçeneği aktif değildir(**disabled**). Dolayısıyla transaction akışının kontrol altına alınmasının istendiği durumlarda bilinçli olarak açılması gerekmektedir. Bunun için transactionFlow niteliğine **true** değerinin atanması yeterlidir.

ServiceBehavior niteliğininde transaction yönetiminde kullanılan önemli özellikleri vardır. Bunlar **TransactionAutoCompleteOnSessionClose**, **RelaseServiceInstanceOnTransactionComplete, TransactionTimeOut** ve **TransactionIsolationLevel** isimli özelliklerdir. TransactionAutoCompleteOnSessionClose özelliği adındanda anlaşılacağı üzere oturum kapatıldığında transaction' a ne olacağının belirlenmesinde kullanılır. Bunu oturum kapatıldığında transaction' ın tamamlanıp tamamlanmayacağının belirtilmesi olarakda düşünebiliriz. Nitekim söz konusu özelliğin değeri true olarak belirlendiğinde hatasız olarak tamamlanan bir oturum sonrasında ilgili transaction otomatik olarak kapatılır. Normal şartlar altında herhangibir problem nedeni ile ortama bir istisna(**exception**) nesnesi fırlatıldığında otomatik olarak geri alma (**rollback**) işlemi gerçekleşir. Bu özelliğin true olarak belirlenmesi bu işlemin daha kontrollü bir şekilde yapılmasınıda sağlar. Ne varki varsayılan değeri false' dur.

RelaseServiceInstanceOnTransactionComplete özelliği, transaction tamamlandığında ilgili servis nesne örneğinin serbest**(relase)** bırakılıp bırakılmayacağının belirlenmesinde kullanılır. Varsayılan değeri true olarak belirlenmiştir.

NOT
: *Eğer* ***OperationBehavior*** *niteliğinin* ***TransactionScopeRequired*** *değeri* ***true*** *ve* ***ServiceBehavior*** *niteliğinin* ***ConcurrencyMode*** *özelliğinin*

*değeri **Reentrant** ise**RelaseServiceInstanceOnTransactionComplete** özelliğinin değeri **false** yapılmalıdır. Aksi takdirde çalışma zamanında bir istisna**(exception)** alınır. Bunun sebebi RelaseServiceInstanceOnTransactionComplete özelliğinin varsayılan değerinin **true** olmasıdır.*

Burada RelaseServiceInstanceOnTransactionComplete özelliğine göre transaction' ları tamamlamanın öncelikli olan 4 farklı yolu vardır.

- OperationBehavior niteliğinde **TransactionAutoComplete** özelliğine **true** değeri verilir.
- Servis tarafında **SetTransactionComplete** metodu çağırılır. Bu özellikle TransactionAutoComplete özelliğinin false olduğu durumlarda kullanılır. Genellikle geliştirme safhasında operasyonun transaction' ı tamamlama ihtiyacının tam olarak kararlaştırılamadığı durumlarda tercih edilebilir.
- İstemcinin (yada diğer bir servisin) dahil olduğu aktif bir transaction içerisindeyken oturumu kapatması veya network hatası alınması halinde eğer **TransactionAutoComplete** özelliğinin değeri **false** ise otomatik olarak geri alma**(rollback)** işlemi gerçekleşir.
- Yukarıdaki durumların dışında oluşabilecek herhangibir neden transaction' ın iptal edilmesi **(abort)** ve o ana kadar yapılmış işlemler var ise bunların geri alınması**(Rollback)**anlamına gelmektedir.

ServiceBehavior niteliğinde transaction yönetimi için kullanılan bir diğer özellikte **TransactionIsolationLevel**' dır. Bu özelliğin değeri, servis içerisinde yeni bir transaction açıldığında veya servise başka bir istemciden (veya bir servisten) bir transaction aktığında (Flow), değişken veriler **(volatile datas)** için söz konusu olacak izolasyon seviyesinin **(Isolation Level)** ne olacağını belirler.

***NOT :** Bir transaction tarafından etkilenen veri, **değişken veri(volatile data)** olarak adlandırılır.*

Izolasyon seviyeleri sayesinde ortak veriler üzerinde diğer transaction' ların yaptığı değişikliklerin nasıl ele alınacağı yada transaction' ın değişikliğe uğrayan veriyi ne kadar süre kilitleyeceği gibi özellikler belirlenebilir. Söz konusu özelliğin değeri **IsolationLevel** enum sabiti türünden olabilir. Temel olarak ele alınabilecek 5 farklı izolasyon seviyesi vardır (Chaos ve Unspecified haricinde). Bunlar aşağıdaki tabloda kısaca özetlenmektedir.

Izolasyon Seviyesi Değeri	**Açıklaması**
ReadUncommited	Transaction boyunca değişken veri **(volatile data)** üzerin Bu nedenle Phantoms, Non-Repeatable Read ve Dirty-Re
ReadCommited	Transaction boyunca değişken veri üzerinde değişiklik ya Ancak Dirty-Read durumları oluşmaz.

Repeatable Read	Transaction boyunca değişken veri üzerinde değişiklik ya veri eklenebilir. Phantoms durumu oluşabilir ancak Non-R
Serializable	En yüksek mertebeli izolasyon seviyesi olarak ele alınır. I ekleme yapılamaz. Bu seviye Phantoms, Non-Repeatable
Snapshot	Bu seviye aslında ReadCommited' ın farklı bir formasyon değişken veri okunabilir. Burada veri değiştirilmeden önc değiştirmediğine bakılır. Eğer böyle bir durum var ise hat yoldur. Bu seviye Phantoms, Non-Repeatable Read ve Di *kullanabilmek için SQL Server üzerinde **Snapshot** desteğ*

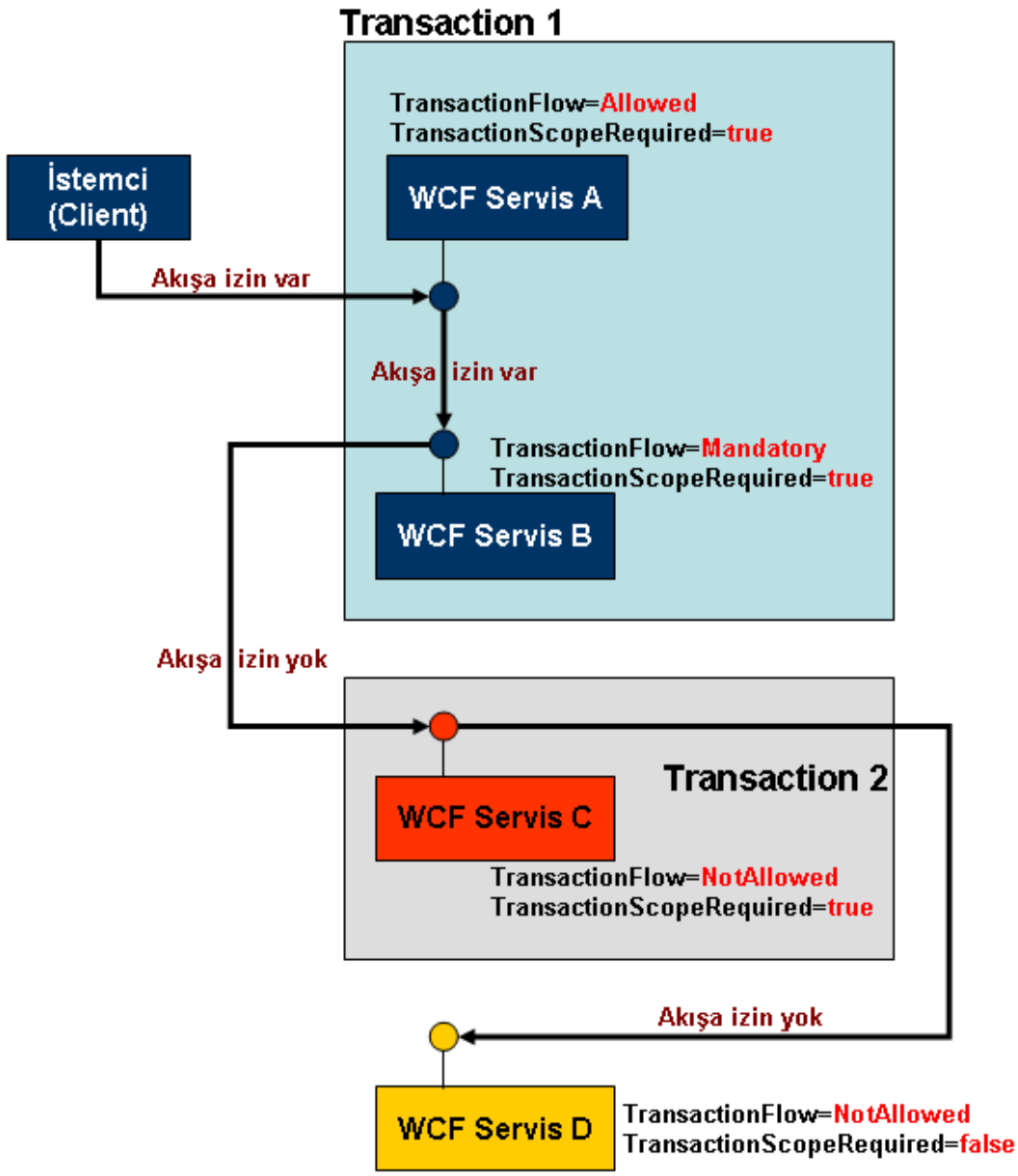
ServiceBehavior tarafında yer alan bir diğer özellik ise **TransactionTimeOut**' dur. Bu özellik tahmin edileceği üzere transaction için zaman aşımı süresini bildirir. Eğer belirtilen sürede transaction tamamlanmamışsa iptal **(abort)** süreci başlatılır.

Gelelim **OperationBehavior** niteliğinin transaction için kullanılan özelliklerine. Söz konusu özellikler, **TransactionAutoComplete** ve **TransactionScopeRequired**' dır. TransactionAutoComplete özelliğinin varsayılan değeri true' dur ve buna göre eğer ele alınamayacak**(unhandled)** bir hata oluşmamışsa söz konusu operasyonun yer aldığı Transaction otomatik olarak tamamlanır (Commit). Ele alınamayan istisnalar otomatik olarak Transaction' ın **iptal(abort)** edilmesini tetiklemektedir.

***NOT :** Transaction' ın kod içerisinden tamamlanması veya iptal edilmesi istendiğinde **TransactionAutoComplete** özelliğinin varsayılan olarak true olan değerinin bilinçli bir şekilde **false**olarak belirtilmesi gerekmektedir.*

TransactionScopeRequired özelliği, uygulandığı metodun çalışması sırasında bir transaction scope gerektirip gerektirmediğinin belirlenmesinde kullanılır. Söz konusu özelliğin varsayılan değeri false' dur. Eğer metodun çalışması sırasında bir Transaction Scope gerekiyorsa true olarak belirlenir. Eğer başka bir yerden metoda akan bir transaction var ise, metod içeriği gelen transaction' a dahil olacaktır. Tam aksine kullanılabilir bir transaction yoksa yeni bir tane oluşturulur ve yürütme yeni açılan bu Transaction içerisinde gerçekleştirilir.

Aslında **TransactionFlow** ve **TransactionScopeRequired** özellikleri göz önüne alındığında bunların sahip olabileceği değerlerin etkisini iyi bir şekilde kavramak gerekir. Bu amaçla örnek bir senaryo olarak aşağıdaki şekli göz önüne alalım.



İlk olarak istemci uygulama Servis A üzerinden bir operasyon çağrısında bulunur. Şekle göre Servis A bağlayıcı tipini (Binding Type) istemcilerden gelecek transaction akışlarını destekleyecek biçimde tasarlamıştır. üstelik çağırılan operasyon için Servis A, Transaction akışına izin vermektedir. Bununla birlikte istemcide açılan bir Transaction mevcut değildir. Lakin Servis A için **TransactionScopeRequired** özelliğinin değeri **true** olarak belirlenmiştir. Buna göre söz konusu operasyon için Servis A otomatik olarak bir Transaction alanı (Scope) açacaktır.

Gelelim Servis B' ye. Servis A, Servis B üzerinden de bir operasyon çağrısı yapmaktadır. Servis B' ye ait bağlayıcı tip(binding type) diğer istemcilerden transaction akışına izin vermektedir. Bununla birlikte Servis B' deki operasyonun **TransactionFlow** özelliği **Mandatory** olarak belirlenmiştir. Bir başka deyişle Servis A' nın Servis B' ye bir Transaction akışı sağlaması şarttır. Diğer taraftan Servis B'

deki TransactionScopeRequired özelliğinin değeri true olduğundan Servis B' deki operasyon, otomatik olarak Servis A' dan gelen **Transaction Scope**' a dahil olacaktır. Buna göre Servis A ve Servis B aynı Transaction Scope içerisinde çalışacaktır.

Servis B, Servis C üzerinden bir operasyon çağırmaktadır. Servis C' deki duruma baktığımızda, bağlayıcı tipin başka servis veya istemcileriden gelen transaction akışına izin vermediği görülmektedir. Bununla birlikte operasyon içinde, transaction akışına izin verilmemektedir. Ancak çağırılan operasyonun bir transaction içerisinde çalıştırılması gereklidir. çünkü **TransactionScopeRequired** özelliğinin değeri **true** olarak ayarlanmıştır. Bu sebepten Servis C kendine ait Transaction alanı içerisinde çalışmaktadır.

Servis C, Servis D üzerinden de bir operasyon çağırmaktadır. Servis C' dekine benzer olarak diğer servis veya istemcilerden gelecek transaction akışına izin vermeyecek şekilde bağlayıcı tipi ayarlanmıştır. Diğer yandan çağırılan operasyon içinde transaction akışına izin verilmemekte ve yeni bir transaction oluşturulması da engellenmektedir. Dolayısıyla buradaki operasyon tamamen transaction haricinde kendi haline çalışmaktadır.

Buradaki bilgiler ışığında pek çok senaryo düşünülebilir. Kullanılan özellikler göz önünde bulundurulduğunda çok doğal olarak olasılıkların sayısı artmakta ve ele alınmaları zorlaşmaktadır. Bu sebepten özelliklerin değerlerinin farklı kombinasyonlarının etkilerini daha kolay bir şekilde görebilmek için, aşağıdaki tablodan yararlanılabilir.

TransactionScopeRequired Değeri	**TransactionFlowOption Değeri**	**transactionFlow Elementi Değeri**	**Transaction'ın Ele Alındığı Taraf**
true	Allowed	false	Servis Taraflı
true	NotAllowed	false	
true	Mandatory	true	İstemci Taraflı
true	Allowed	true	çift Tarafllı
false	Allowed	false	Tarafsız
false	NotAllowed	false	
false	Allowed	true	
false	Mandatory	true	

Tabloda **TransactionScopeRequired**, **TransactionFlowOption** özelliklerinin ve **transactionFlow** elementinin değerlerine göre transaction' ın hangi tarafta ele alınabileceği görülmektedir. Servis taraflı modele göre, servis uygulaması her zaman için

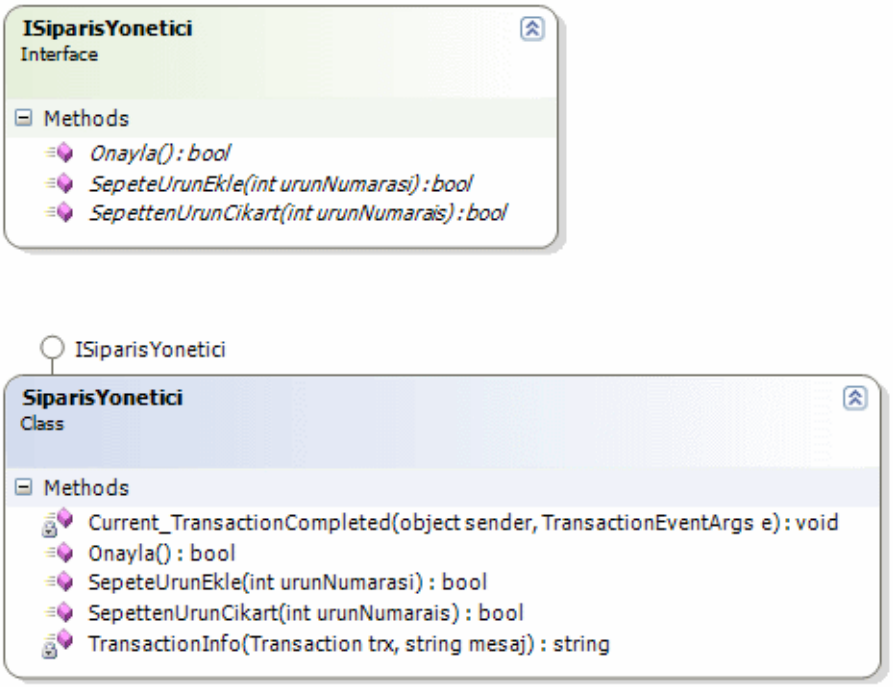
kendine ait bir transaction' a sahiptir. Ayrıca servis tarafı bu transaction' ı her zaman için istemci transaction' ından ayrı olacak şekilde ele alır. Bir başka deyişle servis tarafının her zaman için kendi transaction'ında çalıştığı düşünülebilir. Senaryoda yer alan Servis C bu durumu tam olarak karşılamaktadır.

İstemci taraflı modele göre, servis kendisine istemci tarafından akan transaction' ı kullanır. örnek senaryoda yer alan Servis C bu modele uygun bir şekilde çalışmaktadır. Bu modelde servisin istemciden akan bir transaction'ı alması zorunludur. Daha çok, **deadlock**' ların oluşmasının engellenmesi ve sistem tutarlılığının sağlanması istenilen durumlarda tercih edilebilir. Nitekim istemci tarafında açılan transaction, servis tarafına da dahil edildiğinden tüm işlemler aslında aynı transaction scope içerisinde yer alacaktır. Buda doğal olarak deadlock oluşma ihtimalini azaltmaktadır. Bunun dışında istemci ve servislerin sayısının arttığı bir modelde herkesin aynı transaction' a dahil olması, **atomicity** ilkesinin en iyi biçimde karşılanması anlamınada gelmektedir.

çift taraflı modelde servis istemciden kendisine doğru akan transaction' ı kullanabilir yada akan bir transaction yoksa kendi oluşturduğu transaction scope' u kullanır. Bu yararlı bir modeldir. Eğer istemci tarafından akan bir transaction var ise tüm sürecin atomikliği daha tutarlı bir şekilde sağlanabilir. Yinede istemci tarafından gelen bir transaction yok ise bu durumda servisin bir kök transaction**(root transaction)** açarak tüm süreci üstüne almasıda söz konusu olabilir. Bu açılardan bakıldığında çok fazla tercih edilebilecek bir modeldir.

Son olarak tarafsız modele göre, servis hiç bir durumda transaction' a sahip olmaz. örnek senaryoda yer alan Servis D buna örnek olarak verilebilir. Tarafsız model daha çok istemci tarafındaki transaction' ların, servis tarafından bozulması istenmeyen durumlarda ele alınabilir.

Şimdi basit bir örnek ile transaction yönetimini incelemeye çalışalım. Şu ana kadar anlatılanlar göz önüne alındığında transaction kullanımının değişiklik varyasyonları olabileceği düşünülebilir. örnekte ilk olarak istemci tarafında oluşturulan bir transaction' ın servis tarafına aktarılması ve burada ele alınarak commit işleminin gerçekleştirilmesi incelenecektir. Her zaman olduğu gibi işe servisin sunacağı arayüz sözleşmesi ve uzak nesne sınıfını tasarlayarak başlamak gerekir.



ISiparisYonetici arayüzü;

```
using System;
using System.Transactions;
using System.ServiceModel;

namespace NorthwindYonetici
{
   // IsolationLevel enum sabitinin kullanılabilmesi için System.Transactions.dll' inin
projeye referans edilmesi gerekmektedir.
   [ServiceContract(Name="Siparis_Yonetim_Servisi",
Namespace="http://www.bsenyurt.com/2007/23/6/SiparisYonetimServisi",
SessionMode=SessionMode.Required)]
   public interface ISiparisYonetici
   {
      [OperationContract(Name="SepeteEkle",IsInitiating=true)]
      bool SepeteUrunEkle(int urunNumarasi);

      [OperationContract(Name="SepettenCikar",IsInitiating=false)]
      bool SepettenUrunCikart(int urunNumarais);

      [OperationContract(Name="SepetiOnayla",IsInitiating=false,IsTerminating=true)]
      bool Onayla();
   }
}
```

SiparisYonetici sınıfı;

```
using System;
using System.Data.SqlClient;
using System.Transactions;
using System.ServiceModel;
using System.Text;

namespace NorthwindYonetici
{
   [ServiceBehavior(TransactionIsolationLevel=
IsolationLevel.Serializable, TransactionTimeout="00:02:00",
InstanceContextMode=InstanceContextMode.PerSession)]
   public class SiparisYonetici:ISiparisYonetici
   {
      private string TransactionInfo(Transaction trx,string mesaj)
      {
         StringBuilder builder = new StringBuilder();
         builder.AppendLine(mesaj);
         builder.AppendLine("Güncel Transaction Bilgileri");
         builder.Append("Oluşturulma zamanı (Creation Time): ");
         builder.AppendLine(trx.TransactionInformation.CreationTime.ToLongTimeStri
ng());
         builder.Append("Dağıtık GUID (Distributed GUID): ");
         builder.AppendLine(trx.TransactionInformation.DistributedIdentifier.ToString(
));
         builder.Append("Yered belirleyici (Local Identifier) : ");
         builder.AppendLine(trx.TransactionInformation.LocalIdentifier);
         builder.Append("Durum (Status) : ");
         builder.AppendLine(trx.TransactionInformation.Status.ToString());
         builder.AppendLine();
         return builder.ToString();
      }

      #region ISiparisYonetici Members

      [OperationBehavior(TransactionAutoComplete=false,TransactionScopeRequire
d=true)]
      [TransactionFlow( TransactionFlowOption.Mandatory)]
      public bool SepeteUrunEkle(int urunNumarasi)
      {
         // İstemci tarafında bu metoda yapılan her çağrı sonrasında o anki transaction için
bir TransactionCompleted olayı yüklenir. Bu nedenle kaç metod çağrısı var ise hepsi için
TransactionCompleted olayı birer kez çalışır.
         Transaction.Current.TransactionCompleted += new
```

```
TransactionCompletedEventHandler(Current_TransactionCompleted);
            Console.WriteLine(TransactionInfo(Transaction.Current,"SepeteUrunEkle
metodu çağırıldı"));
            return true;
        }

        void Current_TransactionCompleted(object sender, TransactionEventArgs e)
        {
            Console.WriteLine(TransactionInfo(e.Transaction,"TransactionCompleted Olayı
tetiklendi"));
        }

        [OperationBehavior(TransactionAutoComplete = false,
TransactionScopeRequired = true)]
        [TransactionFlow(TransactionFlowOption.Mandatory)]
        public bool SepettenUrunCikart(int urunNumarais)
        {
            Console.WriteLine(TransactionInfo(Transaction.Current, "SepetenUrunCikart
metodu çağırıldı"));
            return true;
        }

        [OperationBehavior(TransactionAutoComplete = true,
TransactionScopeRequired = true)]
        [TransactionFlow(TransactionFlowOption.Mandatory)]
        public bool Onayla()
        {
            Console.WriteLine(TransactionInfo(Transaction.Current,"Onayla metodu
çağırıldı"));
            /* Yorum satırı haline getirildiğinde TransactionAutoComplete false ise söz konusu
transaction abort edilir. Ancak TransactionScopeAutoComplete true ise aşağıdaki metodu
çağırmaya gerek kalmadan transaction onaylanır. Ayrıca TransactionAutoComplete
özelliği true ise, SetTransactionComplete metodunu çağırmak çalışma zamanı hatasına
neden olur. Bu aktif olan bir transaction var ise bununda iptal edilmesine neden olacaktır.
*/
            //OperationContext.Current.SetTransactionComplete(); // Bu çağrıdan sonra
Transaction.Current null değer döndürecektir.
            return true;
        }

        #endregion
    }
}
```

Söz konusu servis sınıfı içerisindeki metodlarda herhangibir bağlantı ifadesi yer almamaktadır. Sadece bir alış veriş sepetinin hayali bir uyarlaması bulunmaktadır. Ancak önemli olan bazı noktalar vardır. İstemci operasyon çağrılarını gerçekleştirirken ilk olarak **SepeteUrunEkle** metodunu çağırmalıdır. Operasyonların tamamlanması ise **Onayla** metodunun çağırılması ile gerçekleşir. Bu metod sırasını sağlamak için arayüze (interface) ait **OperationContract** niteliği ve **IsInitiating** ile **IsTerminating** özellikleri kullanılmıştır.

SiparisYonetici sınıfındaki metodlarda istemci tarafında oluşturulması zorunlu olan bir transaction akışının gelmesini sağlamak için **TransactionFlow** niteliklerinde **Mandatory** değeri kullanılmıştır. Sınıf içerisinde oluşturulan güncel transaction' ları daha kolay takip edeilmek için TransactionInfo isimli bir metod kullanılmaktadır. çalışma zamanında aktif olan bir transaction' ı elde etmek için Transaction sınıfının **Current** özelliği ele alınabilir.

NOT : *Servis veya istemci uygulamalarda, kod tarafında* ***Transaction***, ***TransactionScope*** *gibi tipleri kullanabilmek için projeye* ***System.Transactions.dll****' inin referans edilmesi gerekmektedir.*

Current özelliği ile çalışma zamanında elde edilen **Transaction** nesne örneği üzerinden **TransactionInfo** özelliğine geçilerek güncel transaction hakkında bazı bilgiler toplanabilir. örneğin oluşturulma zamanı **(Creation Time)**, dağıtık transaction açılmışsa bunun GUID numarası **(Distributed Transaction Identifier)** ve hatta transaction' ın o anki durumu **(status)**gibi. Bu tanımlamalara ek olarak servise ait transaction' ların izolasyon seviyesi **Serializable**, transaction için geçerli zaman aşımı **(timeout)** süresi ise 2 dakika olarak belirlenmiştir.

Servis uygulamasında kullanılacak olan konfigurasyon dosyasının içeriği aşağıdaki gibi tasarlanmalıdır.

Servis tarafındaki App.config dosyası;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <netTcpBinding>
        <binding name="SiparisServisiBindingConfiguration" transactionFlow="true"
transactionProtocol="OleTransactions">
        </binding>
      </netTcpBinding>
    </bindings>
    <services>
      <service name="NorthwindYonetici.SiparisYonetici">
        <endpoint address="net.tcp://localhost:4500/SiparisServisi"
```

```
binding="netTcpBinding" bindingConfiguration="SiparisServisiBindingConfiguration
" name="SiparisServisiEndPoint" contract="NorthwindYonetici.ISiparisYonetici" />
        </service>
      </services>
   </system.serviceModel>
</configuration>
```

Dikkat edilmesi gereken noktalardan birisi, **netTcpBinding** bağlayıcısı için gerekli konfigurasyon
ayarlarında **transactionFlow** niteliğinin **true**, **transactionProtocol** niteliğinin ise **OleTransactions**olarak set edilmiş olmasıdır. Servis tarafı, örneğin daha kolay bir şekilde incelenmesi için Console uygulaması olarak tasarlanmıştır ve kodları aşağıdaki gibidir.

```
ServiceHost host = new ServiceHost(typeof(NorthwindYonetici.SiparisYonetici));
host.Open();
Console.WriteLine("Sunucu dinlemede");
Console.WriteLine("Servisi durdurmak için bir tuşa basın");
Console.ReadLine();
host.Close();
```

Bu adımlardan sonra istemci tarafı için gerekli proxy nesnesinin oluşturulması gerekir. Bunun için yine **svcutil** aracından yararlanılabilir. Dikkat edilmesi gereken noktalardan birisi şudur; servis tarafında **TransactionFlow** niteliklerinde bir değişiklik yapıldığında bunun proxy sınıfı içinde gerçekleştirilmesi bir başka deyişle istemci tarafında güncelleştirilmesi gerekir. Dolayısıyla proxy sınıfının yeniden üretilmesi gerekecektir. Yada manuel olarak istemci tarafındaki proxy sınıfına müdahale edilip bu nitelikler açık bir şekilde değiştirilmelidir.

Söz gelimi SiparisYonetici sınıfındaki **TransactionFlow** niteliklerini **Allowed** olarak belirleyip, istemciler için gerekli proxy sınıfını ürettiğimizi düşünelim. Daha sonradan TransactionFlow niteliğinde **Mandatory** seçeneğini kullanır ve proxy sınıfını güncellenmez ise, WCF çalışma zamanında **ProtocolException** tipinden bir istisna (exception) üretilecektir.

İstemci tarafında yer alacak konfigurasyon dosyasının içeriğinin aşağıdaki gibi olması gerekmektedir.

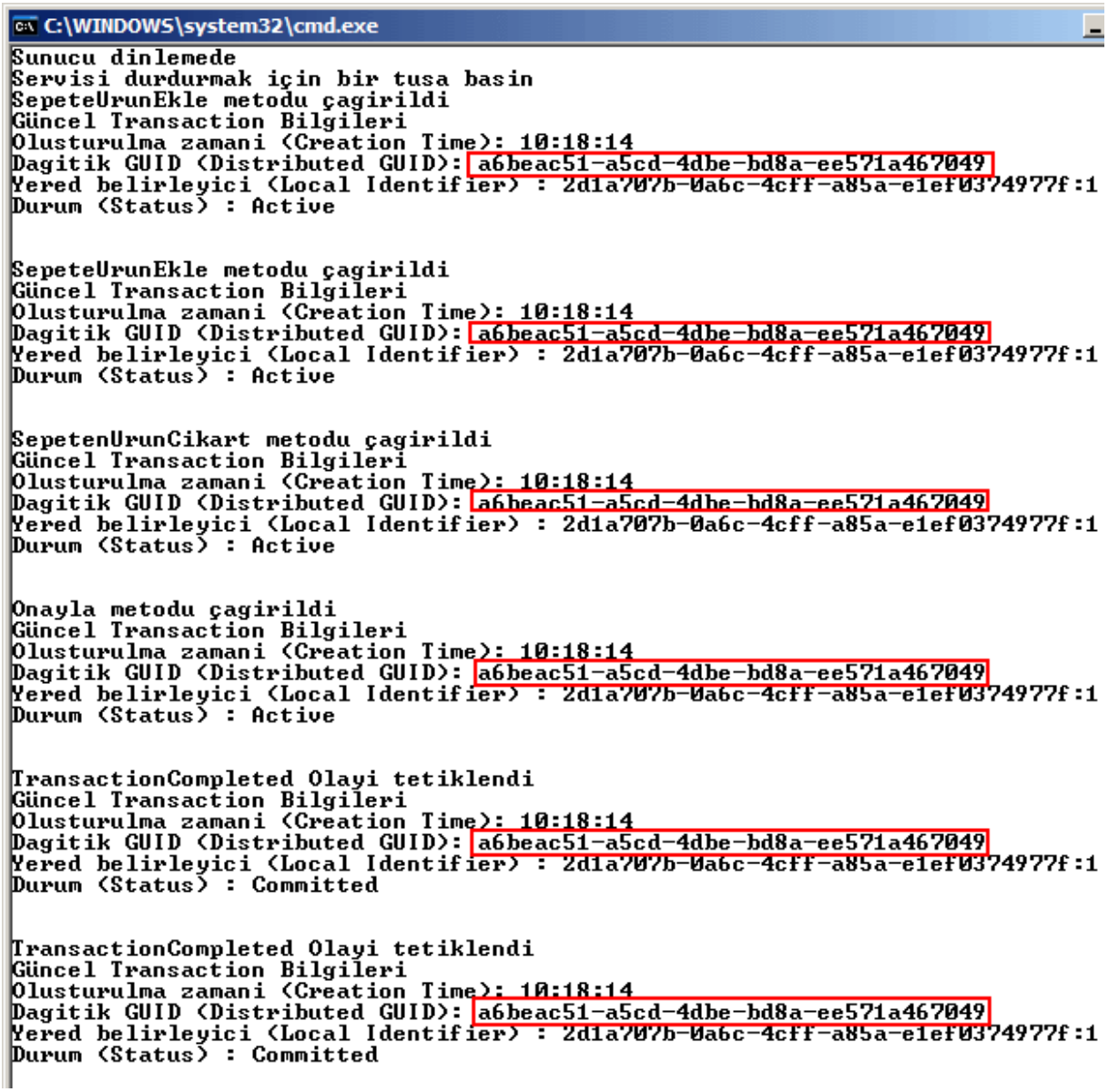
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <system.serviceModel>
      <bindings>
         <netTcpBinding>
            <binding name="SiparisYonetimServisiBinding" transactionFlow="true" tran
sactionProtocol="OleTransactions">
```

```
            </binding>
          </netTcpBinding>
        </bindings>
        <client>
          <endpoint address="net.tcp://localhost:4500/SiparisServisi"
binding="netTcpBinding" bindingConfiguration="SiparisYonetimServisiBinding" cont
ract="Siparis_Yonetim_Servisi" name="Siparis_Yonetim_Servisi" />
        </client>
    </system.serviceModel>
</configuration>
```

İstemci uygulamayıda bir Console olarak tasarladığımızı düşünecek olursak, **Main** metodu içerisinde aşağıdaki kodları yazmamız gerekecektir.

```
Console.WriteLine("Devam etmek için bir tuşa basınız");
Console.ReadLine();
Siparis_Yonetim_ServisiClient client = new
Siparis_Yonetim_ServisiClient("Siparis_Yonetim_Servisi");
using (TransactionScope tx =new
TransactionScope(TransactionScopeOption.RequiresNew))
{
    client.SepeteEkle(11);
    client.SepeteEkle(42);

    client.SepettenCikar(42);

    if(client.SepetiOnayla())
        tx.Complete();
}
Console.ReadLine();
```

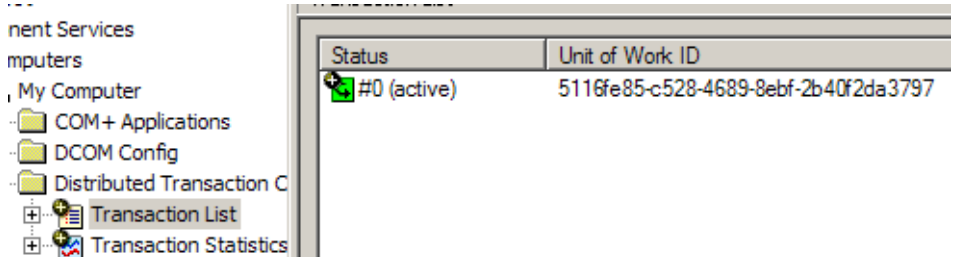
İstemci uygulamada **TransactionScope** tipine ait bir nesne örneği kullanılarak yeni bir transaction alanı açılması sağlanmaktadır. Bundan sonrasında ise istemci tarafından servis üzerindeki bir kaç metod arka arkaya çağırılır. Son olarak using bloğu terk edilmeden önce eğer SepetiOnayla metodunun dönüş değeri true ise tx isimli TransactionScope sınıfı nesne örneğine ait **Complete** metodu çağırılır. önce sunucu ve sonrasında ise istemciyi çalıştırdığımızda servis tarafındaki console uygulamasında aşağıdaki ekran görüntüsü elde edilir.



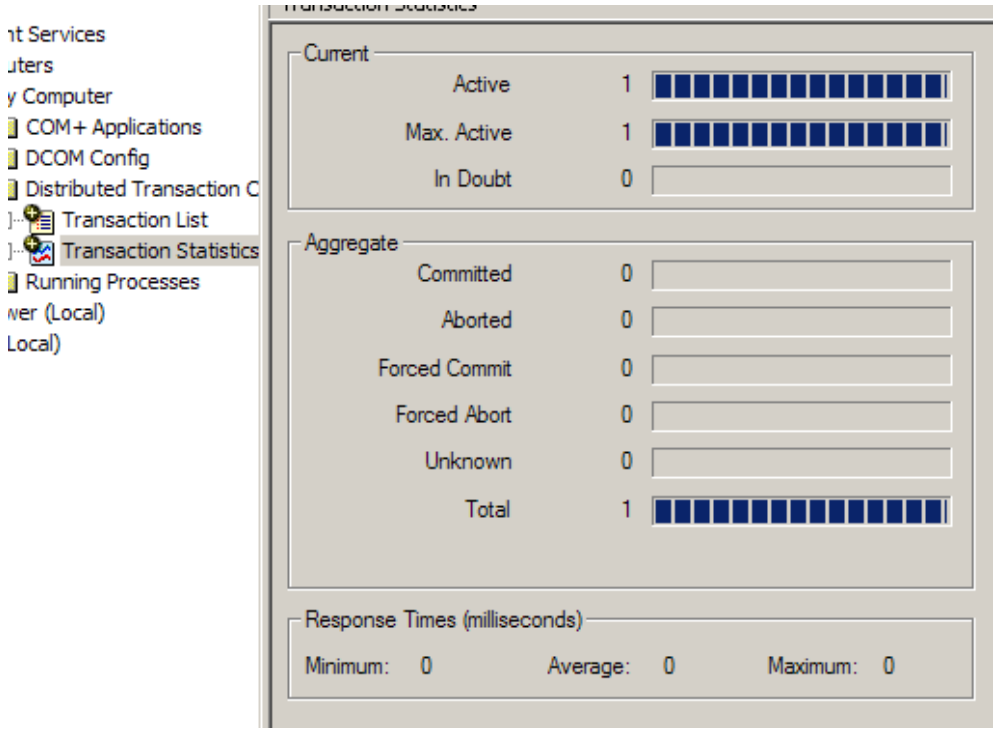
Burada dikkat edilmesi gereken ilk nokta, SepeteUrunEkle metodunun ilk çağırılışıyla beraber elde edilen dağıtık transaction **GUID** numarasının tüm metod çağrılarında aynı olmasıdır. Bu basit olarak, her operasyonun açılan transaction scope' a dahil edildiği anlamına da gelmektedir. SepeteUrunEkle metodu içerisinde Transction için TransactionCompleted olayı yüklenmiştir. İstemci tarafında SepeteUrunEkle metodu iki kez çağırılmıştır. Bu nedenle iki adet **TransactionCompleted** olay metodu yüklenmiş ve güncel transaction' a ait **Commited**durumu iki kez yakalanmıştır. Olayı daha iyi kavrayabilmek için servis tarafındaki SiparisYonetici sınıfı içerisindeki SepeteUrunEkle metodu içerisinde aşağıdaki ekran görüntüsünde olduğu gibi **breakpoint** konulması yerinde bir harekete olacaktır.

```
30
31      [OperationBehavior(TransactionAutoComplete=false,TransactionScopeRequired=true)]
32      [TransactionFlow( TransactionFlowOption.Mandatory)]
33      public bool SepeteUrunEkle(int urunNumarasi)
34      {
35          // İstemci tarafında bu metoda yapılan her çağrı sonrasında TransactionCompleted olay
    nedenle kaç metod çağrısı var ise hepsi için TransactionCompleted olayı birer kez çalışır.
36          Transaction.Current.TransactionCompleted += new TransactionCompletedEventHandler
    (Current_TransactionCompleted);
37          Console.WriteLine(TransactionInfo(Transaction.Current,"SepeteUrunEkle metodu çağır
38          return true;
39      }
```

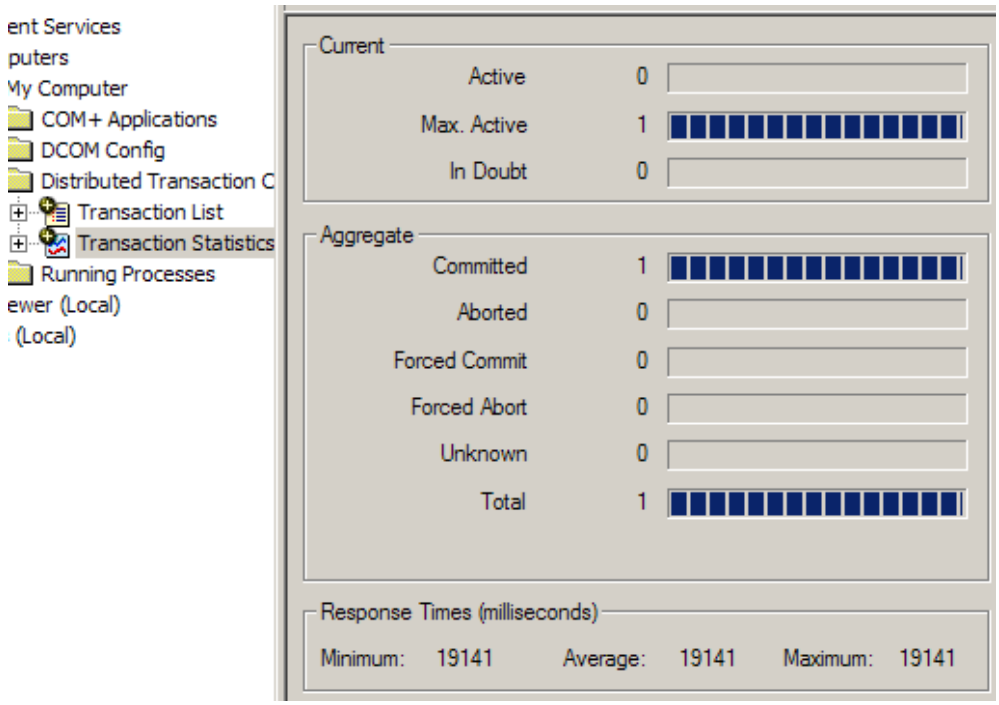
İlk metod çağrısı ile birlikte **Compenent Services** içerisindeki **Distributed Transaction Coordinator** kısmına bakıldığında **Transaction List** içerisinde yeni bir OleTx transaction açıldığı ve **Transaction Statistics** kısmındada bu transaction' ın aktif olarak yer aldığı görülmektedir.



Aynı zamanda **Transaction Statistics** kısmına bakılırsa aşağıdakine benzer bir ekran görüntüsü ile karşılaşılır. Dikkat edilecek olursa aktif olan 1 adet transaction bulunmaktadır.



Eğer işlemler sonuna kadar devam ettirilirse, transaction' ın başarılı bir şekilde tamamlandığı yine istatistik kısmından görülebilir. Burada dikkat edilmesi gereken değer **Commited**' dır.

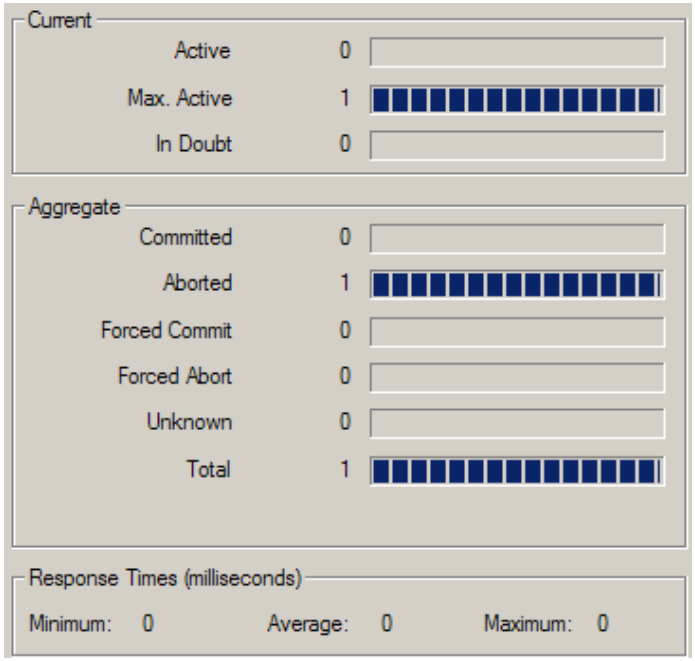


Şimdi Onayla metodundan geriye **false** değer döndürdüğümüzü düşünelim. Bu durumda istemci tarafındaki **TransactionScope** sınıfına ait nesne örneği üzerinden **Complete**metodu tetiklenemiyecektir. Dolayısıyla söz konusu transaction' ın

iptal edilmiş olması gerekir. Gerçektende uygulama çalıştırıldığında aşağıdaki ekran görüntüsünde olduğu gibi **TransactionCompleted** isimli olay metodu tetiklendiğinde, o anki transaction' ın durumunun **Aborted** olduğu gözlemlenebilir.

```
TransactionCompleted Olayi tetiklendi
Güncel Transaction Bilgileri
Olusturulma zamani (Creation Time): 13:41:39
Dagitik GUID (Distributed GUID): 17a19f68-4864-4053-bc19-e27aa32fdff0
Yered belirleyici (Local Identifier) : c5bed0b2-d749-44fc-aaba-210973ce7253:1
Durum (Status) : Aborted


TransactionCompleted Olayi tetiklendi
Güncel Transaction Bilgileri
Olusturulma zamani (Creation Time): 13:41:39
Dagitik GUID (Distributed GUID): 17a19f68-4864-4053-bc19-e27aa32fdff0
Yered belirleyici (Local Identifier) : c5bed0b2-d749-44fc-aaba-210973ce7253:1
Durum (Status) : Aborted
```

Benzer şekilde **DTC** altındaki **Transaction Statistics** bölümüne bakıldığında bir transaction' ın iptal edildiği bilgisi yer alacaktır. *(Bu aşamada olayın daha iyi analiz edilmesi için**Services** kısmından **Distributed Transaction Coordinator** servisi **Restart** edilmiştir. Bu sebepten aşağıdaki ekran görüntüsünde olduğu gibi önceki örnekle birlikte toplam 2 transaction olması gerekirken sadece 1 transaction yer almaktadır.)*



Dikkat etmemiz gereken noktalardan biriside **transactionFlow** elementinin değeridir. Eğer istemci tarafındaki konfigurasyon dosyasında bu özelliğin değeri **false** olarak bildirilirse, **Mandatory** seçeneği nedeni ile WCF çalışma zamanında **InvalidOperationException** istisnası alınır. Benzer durum servis tarafı içinde

geçerlidir. Eğer istemcide özelliğin değeri true ama servis tarafında false ise yine WCF çalışma zamanı servis tarafına bir InvalidOperationException istisnası fırlatacaktır.

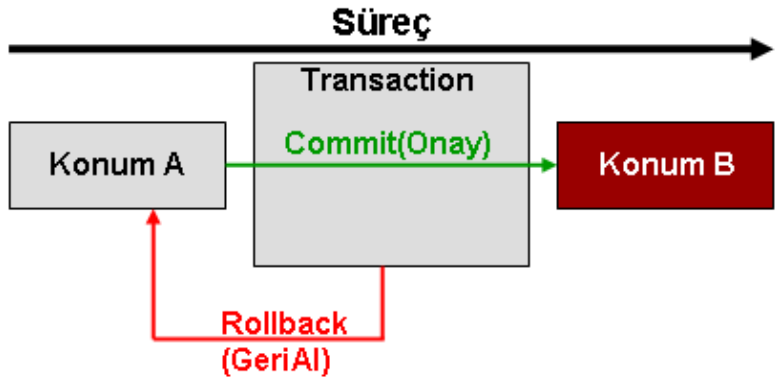
Böylece geldik bir makalemizin daha sonuna. Bu makalemizde WCF için transaction yönetimini biraz daha derinlemesine incelemeye çalıştık. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

örnek uygulama için tıklayın.

WCF - Transaction Yönetimi - 1 (2007-06-19T18:37:00)

wcf,

Transaction (İşlem) yönetimi özellikle veritabanı kaynakları söz konusu olduğunda her sistemde büyük bir önem sahiptir. Basit olarak transaction bir veya daha çok işlem bütününü temsil eder. Bütünü oluşturan söz konusu işlem parçaları çoğunlukla birbirleriyle ilişkilidir ve hepsinin başarılı bir şekilde tamamlanabilmesi sonrasında transaction' ın başarılı olduğu söylenebilir. Bu doğal olarak işlem parçalarından herhangibirinin başarısız olması sonucunda transaction' ınında başarısız olması anlamına gelmektedir. Aşağıdaki şekilde bir transaction' ın süreç içerisinde sistemin belirli bir konumdan başka bir konuma geçişi sırasında üstlendiği rol ifade edilmeye çalışılmaktadır.



Buna göre sistemin belirli bir konumdan başka bir konuma geçmesi sırasında yapılacak işlemlerin tamamı bir bütün olarak ele alınmalıdır. Söz konusu bütün içerisinde meydana gelebilecek aksamalarda sistem ilk konumuna (Konum A) dönebilmelidir. Tam tersine tüm işlemler başarılı olduğunda sistem yeni konumuna (Konum B) geçebilir. Bu şekilde düşünüldüğüne transaction ACID(**A**tomicity**C**onsistency**I**solation**D**urability) adı verilen standartları sağlamak durumundadır. Makalemizin konusu Windows Communication Foundation sistemi üzerinde transaction yönetiminin nasıl sağlanabileceğinin temellerini incelemektir. Yinede **ACID** ilkelerinden kısaca bahsetmek gerektiği en azından hatırlanmasında fayda olacağı kanısındayım.

Atomicity ilkesine göre bir transaction' ın başarılı olması, içerisinde yer alan tüm işlem parçalarının ayrı ayrı başarılı olması anlamına gelmektedir. **Consistency** ilkesi, yapılan

işlemler sonucunda oluşan çıktıların tutarlılığı anlamına gelir. Her zaman verilen bir örnek tutarlılık ilkesini son derece güzel anlatmaktadır. Bankalar arası yapılan bir havale işlemi sırasında bir hesaptan çıkan miktar ne kadar ise diğer hesaba aktarılanında o kadar olması gerekir. Ne bir fazla ne de bir eksik olmamalıdır. İşte bu verinin tutarlılığını korumak olarak düşünülebilir. **Isolation** ilkesine göre bir transaction içerisindeki tüm işlem parçaları diğer transactionlardan izole olacak şekilde çalışmalıdır. Bu olmadığı takdirde tutarsızlıklar oluşabilir.

Elbette transaction' larda söz konusu olabilecek **Dirty-Read**, **Phantom**, **Non-Repeatable Read** gibi durumların hangisinin tercih edileceğine göre çeşitli izolasyon seviyeleri **(Isolation Levels)** pek çok veri tabanı sisteminde ele alınmaktadır. Bunlar .Net tarafındaki Transaction sınıflarıncada kullanılmakta ve **IsolationLevel** enum sabitleri ile işaret edilmektedir. Söz konusu seviyeler **WCF**içerisindede bir servis davranışı (Service Behavior) olarak belirtilebilmektedir. Söz konusu bildirim **ServiceBehavior** niteliğinin **TransactionIsolationLevel** özelliğinin değeri ile sabitlenmektedir.

NOT : Temel Olarak Transaction' larda Düşünelecek Senaryolar;

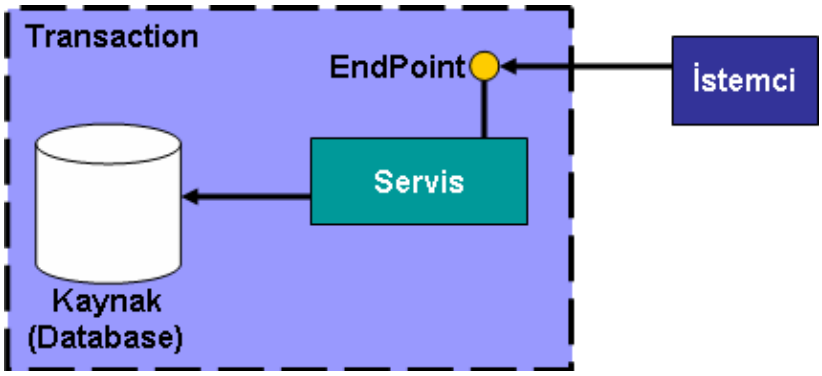
Phantom; *başlatılan iki transaction olduğunu düşünelim. Bu transactionların birisi açık iken yeni bir veri girişi yapıp işlemi onaylıyor (Commit) olsun. Bu durumda halen daha açık olan diğer transaction aynı veri kümesini tekrardan talep ettiğinde daha önce kendisinde var olmayan yeni eklenmiş bir satır ile karşılaşacaktır. Bu satırlar hayalet(Phantom) olarak adlandırılır.*

Non-Repeatable Read; *yine başlatılmış iki transaction olduğunu düşünelim. Bu sefer açık olan transaction' lardan birisi var olan veri kümesinde belirli bir satırda(satırlarda) güncelleme işlemi yapmış olsun. Sonrasında ise yapılan bu değişiklikleri onaylasın(Commit). Açık kalan diğer transaction'ın tekrardan aynı veri kümesini çektiğini düşünelim. Bu durumda açık olan transaction daha önce baktığı verilerin değiştiğini görecektir. Bu Non-Repeatable Read olarak adlandırılan bir durumdur.*

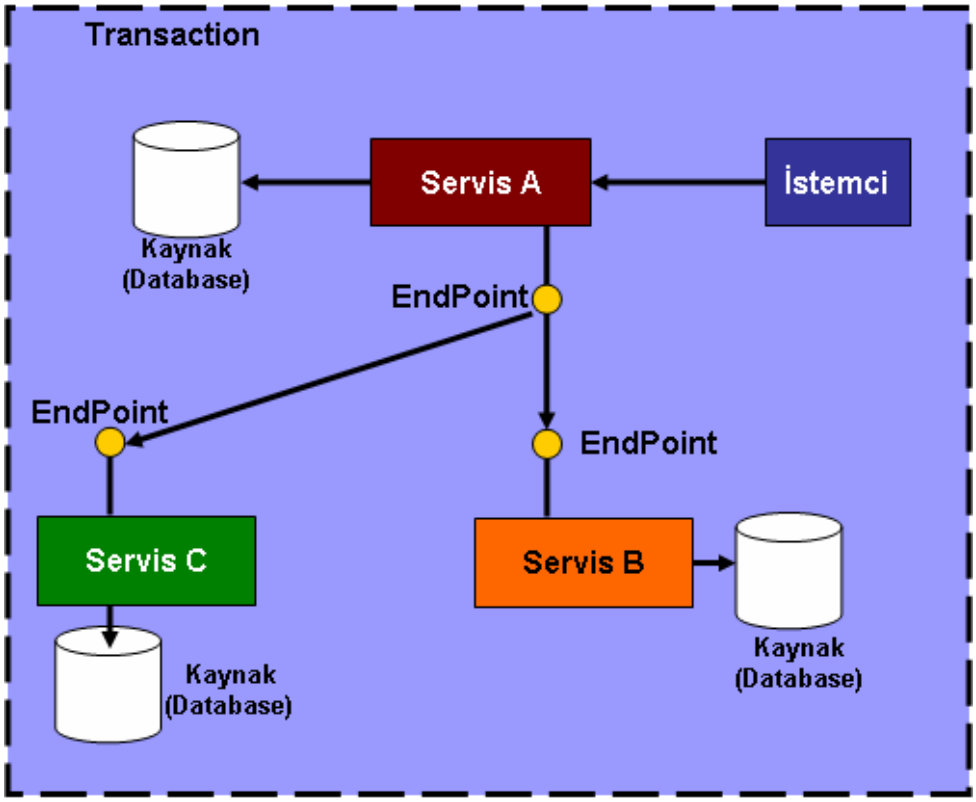
Dirty-Read; *yine başlatılmış iki transaction olduğunu göz önüne alalım. Bunlardan birisi veri kümesine herhangibir satır eklemiş veya güncellemiş olsun. Ancak bu sırada diğer transaction'ın tekrardan veri çektiğini düşünelim. Burada dikkat edilmesi gereken durum şudur; iki transaction açık iken, herhangibiri commit edilmeden önce diğeri veri çekebilmektedirler. Dolayısıyla diğer transaction var olan ekleme ve değişiklikleri o an için görecektir. Ne varki bu senaryoda değişiklikleri yapan transaction işlemleri* ***Commit etmez****. Aksine geri alır****(Rollback)****. Bu durumda daha önce tekrardan veri çeken açık transaction' da aslında geri alınmış güncellemeler ve eklemeler görülmektedir. Bu Dirty-Read durumu olarak adlandırılmaktadır.*

Durability ilkesine göre transaction içerisinde yer alan iş parçalarında bir aksaklık olması halinde sistemin ilk haline dönebilmesi gerekir. Yani taşlar ilk konumlarındaki yerlerinde kalabilmeldir.

Gelelim WCF içerisindeki duruma. Transaction' lar **servis yönelimli mimari (Service Oriented Architecture)** üzerinde ele alındığında işin içerisine giren faktörler transaction yönetimini zorlaştırmaktadır. Nitekim işlem parçalarının farklı makinelerdeki, farklı uygulama alanları **(AppDomain)** içerisinde*(Dolayısıyla farklı süreç-process' lerde)* olmaları söz konusudur. Bu işlem parçaları çeşitli kaynakları **(Resources)** kullanacaktır. Söz konusu kaynaklar farklı veritabanı sistemleride olabilir. örneğin bir servis Microsoft SQL Server üzerinde bir işlem gerçekleştiriyorken, buna bağlı olan diğer bir işlem parçasıda Oracle veritabanı kullanan başka bir servisin parçası olabilir. Dahası farklı uygulama alanları(Application Domain) içerisindeki taraflar basit birer istemci dışında başka servislerde olabilir ve bunlar farklı platformlar üzerinde yer alabilir. En iyimser haliyle düşünüldüğünde tek bir AppDomain içerisinde konuşlandırılmış bir WCF Servis uygulaması, kendisine bağlı bir veya daha çok kaynak **(resource)** üzerinde çalışacak transaction' ları kolay bir şekilde yönetebilir. Aşağıdaki şekilde bu durum analiz edilmeye çalışılmaktadır.



Burada bilinen Ado.Net Transaction tiplerinin kullanılması bile yeterlidir. Ne varki az önce bahsettiğimiz gibi servis yönelimli mimari aslında aşağıdaki şekildekine benzer bir durumuda içermektedir.



Sanıyorumki şimdi durum biraz daha karıştı. Bu şekle göre istemcinin talepte bulunduğu işlemler sonrası Servis A, Servis B ve Servis C üzerindeki kaynaklardaki işlem parçalarının başarılı bir şekilde tamamlanmasını istemektedir. Bir başka deyişle Servis B , Servis C ve Servis A üzerindeki işlem parçalarının hepsinin ayrı ayrı başarılı olmaları gerekmektedir. Burada ortaya bir koordinasyon problemi çıktığıda son derece açıktır. örneğin aşağı maddeler halinde sıralanmış olan soruların söz konusu sistemde nasıl karşılanabileceği düşünüldüğünde, sadece bu işler için neden güçlü koordinasyon uygulamaları yazıldığı daha kolay bir şekilde anlaşılmaktadır.

- Acaba bu sistemde Transaction' ı kim başlatacak?
- Diyelimki Transaction birisi tarafından başlatıldı. Diğer işlem parçalarını açılan bu Transaction' a kim dahil edecek **(auto enlist)**? öyleki dahil olmadan transaction' ın farklı sistemlere yayıldığı nasıl anlaşılacak?
- Geri alma**(Rollback)** veya onaylama**(Commit)** işlemlerini kim üstlenecek?
- Birden fazla servis söz konusu olduğundan bunlar üzerinde toplu bir rollback veya commit kararı nasıl ve neye göre verilecek?
- Eğer Transaction' ı üstlenen, bir başka deyişle süreci başlatıp bitirecek olan servis, diğer servislerdeki işlem parçalarının sonuçlarından nasıl haberdar olacak?
- Bahsedilen bu işlem parçalarını içeren ve yürüten servisler farklı platformlarda hatta farklı iletişim seviyelerinde*(Http, Tcp gibi)* olduğu zaman nasıl kontrol altına alınacak?

Görüldüğü üzere sorular ve sorunlar giderek artmaktadır. çözümün adı **dağıtık transaction (Distributed Transactions)** dır. Gerçektende servis yönelimli mimarilerde(SOA-Service

Oriented Architecture) yer alan işlem parçalarını kontrol altına almanın tek yolü dağıtık transaction' ları kullanmaktadır. Dağıtık transaction' lar iki önemli parçaya sahiptir. İlk olarak bu modelde çift yönlü geçerli kılma **(two phase commit)** adı verilen bir protokol söz konusudur. İkinciside transaction yönetimini üstlenen ve genelde üçüncü parti olarak tedarik edilen bir yönetici uygulamadır **(Dedicated Transaction Manager)**.

çok basit olarak **two phase commit** protokolünü açıklamaya çalışarak devam edelim. Bu protokol iki aşamadan oluşur. İlk aşamada transaction' a katılan işlem parçalarından sorumlu servisler kendi işlemlerinin başarılı olup olmadıklarına dair yöneticiye bilgi gönderirler*(Pek çok kaynakta bu işlem oylama-**voting** olarakta geçmektedir)*. Buradaki yönetici root görevini üstlenen bir servise adanmış da olabilir. İkinci aşamada yönetici, açmış olduğu transaction içerisine katılmış işlem parçalarının oylamalarını değerlendirir. Eğer hepsi geçerli ise tüm bağlı olan işlem parçaları için ilgili servislere onaylayın bilgisini gönderir(Commit aşaması). Elbetteki tam tersi durumda bağlı servislere işlemleri geri almalarına dair başka bir bilgi gönderilecektir(Rollback aşaması).

Söz konusu çift yönlü geçerli kılma protokolünün yanı sıra, Windows Communication Foundation servislerin bulundukları farklı platformlara, aradaki iletişim protokollerinin çeşitliliğine göre üç ayrı transaction protokolü içerir. Bunlar **Lightweight, OleTx** ve **WSAT (WS-Atomic)** transaction protokolleridir. Aslında OleTx ve WS-Atomic kendi içlerinde two phase commit protokolünü zaten ele almaktadırlar.

Lightweight transaction protokolüne göre işlem parçaları yerel olarak servis içerisinde ele alınır. Bir başka deyişle söz konusu transaction servis sınırları dışına çıkartılamaz. Buda ilgili iş parçalarının başka servislerde başlatılacak hizmetlere katılamayacağı anlamına gelir. Tipik olarak bir istemci ve servisin söz konusu olduğu durumlarda ele alınabilir. Servis içerisinde yerel olarak kullanıldıklarından performans yönünden avantajlıdır diğer modellerine göre daha avantajlıdır.

OleTx tipinde, transaction başlatıldığı servis, uygulama alanı ve makine sınırlarını aşabilir. Dolayısıyla dağıtık transaction' ların yönetimi sırasında kolayca ele alınabilir. Ancak bu protokol intranet tabanlı Windows sistemleri üzerine tasarlanmıştır. Bu sebepten firewall gibi sistemlere takılma riski vardır. Ayrıca Windows tabanlı geliştirildiğinden farklı tipteki platformlar tarafından desteklenmeyecektir. Bu nedenle OleTx tipi transaction protokolü Windows tabanlı **intranet** sistemlerinde tercih edilmektedir.

WSAT (Web Service Atomic) tipinden transactionlar aynen OleTx' de oluduğu gibi servis, uygulama alanı ve makine sınırlarını aşabilir. Diğer taraftan WS-Atomic global bir standarttır. özellikle web servislerinde yaygın olarak kullanılmaktadır. Bu nedele açık metin tabanlı olarak çalıştığında farklı platformlar tarafından kullanılabilir ve OleTx tipinde olduğu gibi firewall engeline takılmaz. Bu sebeten **internet** tabanlı sistemlerdeki dağıtık transaction' ların yönetiminde tercih edilen bir protokoldür. Elbetteki intranet tabanlı sistemlerede istenirse uygulanabilir.

Bildiğiniz gibi Windows Communication Foundation içerisinde servisler ile istemciler arasındaki iletişim sağlanırken pek çok ayarlamayı bünyesinde barındıran bağlayıcı tipler **(Binding Type)** kullanılmaktadır. Bağlayıcı tipler iletişim protokolü, güvenlik ayarları dışında, transaction protokollerininde kolay bir şekilde belirlenmesinde önemli görevler üstlenir. Bunların bir kısmının **OleTx** yada **WSAT** transaction protokolleri için desteği bulunmamaktadır. Bir kısmınında hiç bir transaction protokolüne desteği bulunmamaktadır. Aşağıdaki tabloda var olan Windows Communication Foundation bağlayıcı tiplerinin desteklediği **varsayılan** transaction protokolleri yer almaktadır.

Binding Tipi	Transaction Protokolü Desteği
BasicHttpBinding	Destek yok
WSHttpBinding	WSAT
WSDualHttpBinding	WSAT
WSFederationHttpBinding	WSAT
NetTcpBinding	OleTx
NetPeerTcpBinding	Destek yok
NetNamedPipesBinding	OleTx
NetMsmqBinding	Destek yok
MsmqIntegrationBinding	Destek yok

Burada dikkat edilmesi gereken konulardan biriside **NetTcpBinding**, **NetNamedPipesBinding** için varsayılan olan **OleTx** transaction desteğinin **WSAT** olarak da ayarlanabileceğidir. Ancak intranet üzerinde koşan Windows tabanlı bir sistemde bu performası olumsuz etkileyecek bir değişiklik olabilir. Yinede istenirse, intranet üzerinde farklı platformlar söz konusu olduğunda ilgili bağlayıcı tipler için WSAT desteği seçilebilir.

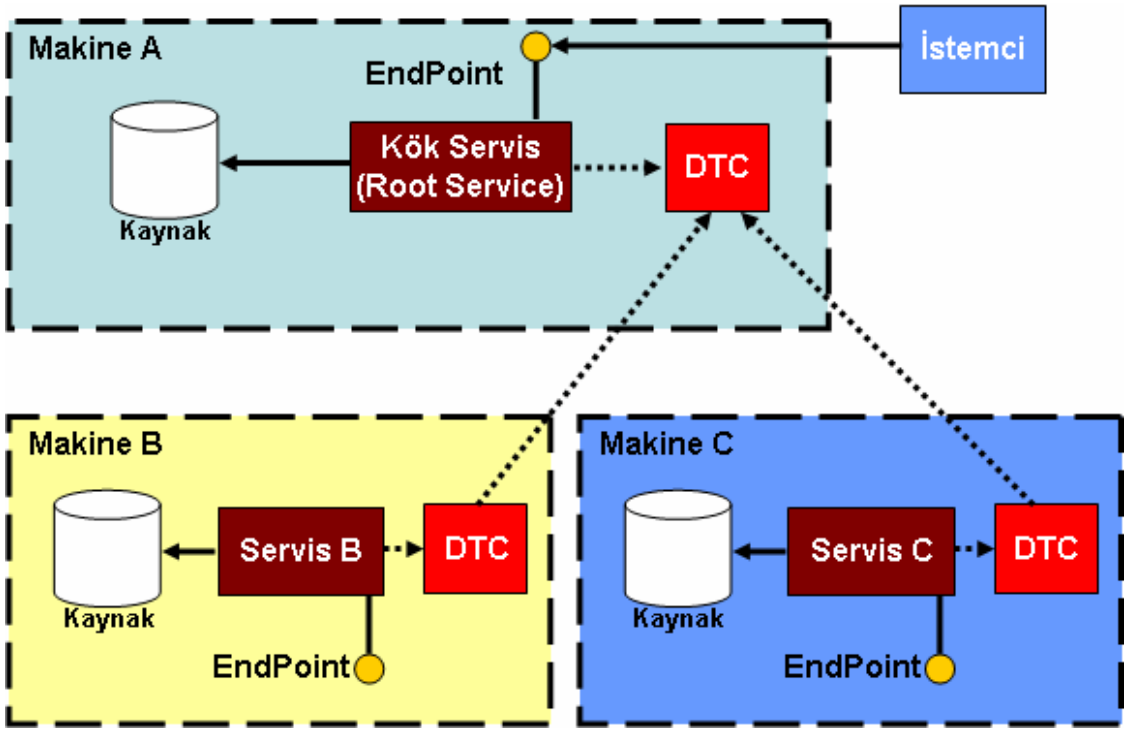
NOT : *Bilindiği gibi Windows Communication Foundation uygulamalarında kendi bağlayıcı tiplerimizi* ***(user defined binding type)*** *yazabiliriz. Böyle bir durumda hangi transaction protokolünün kullanılacağı tam olarak geliştiriciye bağlıdır.*

Artık bağlayıcı tipin ele alabileceği transaction protokolleri bilindiğine göre geriye bir tek transaction yönetimini üstlenecek bir uygulama bulmak kalmaktadır. Bu tip bir uygulamayı geliştirmek oldukça zor olduğundan çoğunlukla üçüncü parti programlar göz önüne alınmaktadır. Windows Communication Foundation düşünüldüğünde kullanılabilecek söz konusu yöneticiler, **LTM (LightWeight Transaction Manager), KTM (Kernel Transaction Manager)** ve **DTC (Distributed Transaction Cooridantor)** tipleridir. Bu tiplerden KTM, **Windows Vista** ile birlikte gelmektedir.

Söz konusu yöneticiler arasında bizi en çok ilgilendirenlerden ve aşina olduklarımızdan birisi DTC' dir. özellikle Ado.Net 2.0 ile birlikte gelen **TransactionScope** nesnesinin otomatik olarak DTC özelliğini aktifleştirebildiği ve dağıtık transaction' ların daha kolay

yönetilebilirliğini sağladığı göz önüne alındığında, servis yönelimli mimari model için ne kadar anlamlı olduğu açık bir şekilde ortadadır. Dağıtık transaction koordinatörü, hem **OleTx** hemde **WSAT** transaction protokollerini destekler. Dolayısıyla internet veya intranet tabanlı sistemlerde yer alabilecek servis uygulamalarında transaction' ların servis sınırları dışarısına yayınlanabilmesine olanak sağlar.

DTC(Distributed Transaction Coordinator) transactionların oluşturulup başlatılmasından, süreç içerisindeki diğer uygulama alanlarında**(AppDomain)** yer alan servislere ait işlem parçalarının var olan transaction' a katılmasından **(enlist)**, kaynak yöneticilerinden**(Resoruce Managers)** gelen onay mesajlarının **(vote)** toplanmasından ve elbetteki tüm işlemlerin onaylanması**(Commit)**veya geri alınmasından**(Rollback)** sorumlu genel bir sistem servisi olarak göz önüne alınabilir. Bu açıdan bakıldığında Windows Communication Foundation ile arasında oldukça sıkı bir ilişki olması kaçınılmazdır. Konuyu daha iyi kavrayabilmek için aşağıdaki şekil göz önüne alınabilir.



İstemci uygulama, Makine A üzerinde yer alan servisten bir işlem için talepte bulunmaktadır. Makine A üzerinde yer alan servis bu senaryoda root olarak görev yapmaktadır. Buna göre transaction' ın başlatılmasından hatta sonlandırılmasında*n (ister commit ister rollback)* kendisi sorumludur. Makine A' da yer alan servis işlemlerin geri kalanı için Makine B ve C üzerinde yer alan servisleride kullanmaktadır. Burada root servis kendi proxy nesnesi yardımıyla B ve C servislerine bir transaction ID değeri gönderir. Bu transaction ID sistem tarafından üretilen benzersiz ve tekrar etmeyen bir değerdir. Bunu programlama ortamında da kullanabildiğimiz GUID (Global Unique IDentifier) değeri olarak düşünebiliriz. Söz konusu ID, B ve C servisleri tarafından kendi makinelerindeki **DTC**hizmetlerinede bildirilir. Bir başka deyişle B ve C makinelerinde yapılmak üzere olan işlemlerin, transaction ID' si belirtilen sürece ait oldukları bildirilir.

Dolayısıyla B ve C servislerinin yürüttükleri işlemler otomatik olarak, root servis tarafından açılan transaction' a dahil edilmiş olurlar.

Bu işlemlerin ardından tahmin edileceği üzere çift yönlü geçerli kılma protokolü**(two phase commit protocol)** başlar. Yani root servis, transaction' a dahil olmuş diğer servislerden oylarını ister. Burada diğer makinelerdeki DTC servisleri devreye girerek oylama**(vote)** sonuçlarını root DTC servisine iletirler. Eğer herkes kabul ediyorsa, bir başka deyişle transaction' a dahil olan tüm servisler yaptıkları işlemlere onay vermişse, root servis ikinci aşamaya geçer. Bu aşamada da transaction' a dahil olan servislere işlemleri onaylamalarına dair bilgi gider. Sonuç olarak işlemler onaylanır **(Commit)**. Elbetteki diğer makinelerdeki DTC' lerden gelecek tek bir iptal isteği, tüm transaction' ın iptal edileceği ve işlemlerin geri alınması için root DTC' den, bağlı olan servislere bilgi gönderileceği anlamına da gelmektedir**(Rollback)**. Burada dikkat edilmesi gereken noktalardan biriside, yönetsel işlemleri üstlenen Dağıtık Transaction Koordinasyon(DTC) servisinin, root makinede olmasıdır. Bir başka deyişle Servis Yönelimli Mimari (SOA) modelindeki tüm DTC' lerin oylarını kontrol edip, kabul edilmeleri veya iptal işlemlerini gerçekleştirebilecek tek bir DTC var olabilir.

Windows Communication Foundation, transaction' lara otomatik olarak bir transaction yöneticisi (transaction manager) atar. Varsayılan olarak tek başına çalışan ve farklı makineler yada süreçlerdeki(process) servislere transaction yayınlamayan sistemler söz konusu olduğunda otomatik olarak ilgili transaction' la **LTM(LightWeight Transaction Manager)** ilişkilendirilir. Ancak servis tarafından başlatılan transaction' ın kendi uygulama alanı sınırları dışına çıkması bir başka deyişle farklı makine veya süreçlere yayınlanması halinde transaction yöneticisinin seviyesi otomatik olarak **DTC(Distributed Transaction Coordinator)** servisine devredilir. KTM göz önüne alındığında çok değişken kaynaklar (Volatile Resource) vardır. Yinede servis sınırları dışına çıkılması halinde DTC' ye terfi edilme söz konusudur.

Burada elbetteki kaynak yöneticisinin**(resource manager)** bahsedilen yöneticilerden hangisine destek verdiğide çok önemlidir. Kaynak yöneticisi olarak SQL Server 2005, Oracle, MSMQ gibi pek çok sistem göz önüne alınabilir. SQL Server 2005 veritabanı servisine ait kaynak yöneticisinin şu an için **KTM(Kernel Transaction Manager)** dışında LTM ve DTC' yi desteklediği bilinmektedir. Aslında DTC değişken olabilecek her türde kaynak yöneticisi tarafından desteklenmektedir.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde teorik olarak Windows Communication Foundation ortamında transaction yönetiminin temellerini incelemeye çalıştık. Görüldüğü gibi yönetimin arka tarafında dikkate alınması gereken pek çok etken bulunmaktadır. Dahasıda vardır. WCF ile ilgili bir sonraki makalemizde transaction yönetimini örnekler ile incelemeye ve diğer temelleride görmeye çalışacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

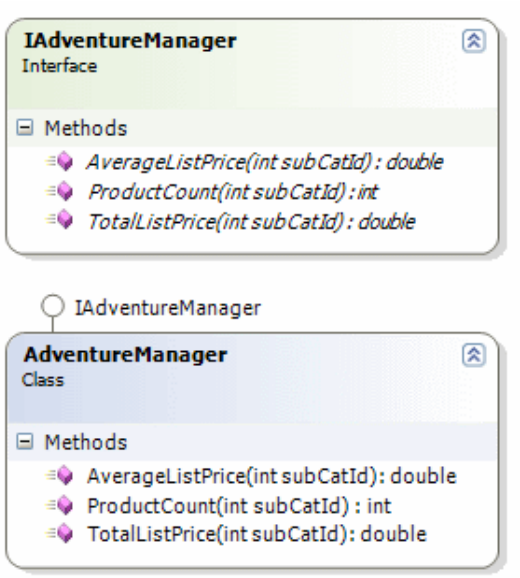
WCF - İstemci Taraflı Asenkron Erişimler (2007-06-13T18:51:00)

wcf,

Windows Communication Foundation ile ilgili bir önceki makalemizde One Way tekniğini uygulayarak istemcilerin asenkron olarak uzak metodları nasıl çağırabileceklerini incelemiştik. One Way tekniğinin elbetteki en büyük dezavantajı geriye değer döndüren metodların ele alınamayışıdır. Oysaki çoğu zaman, işlem süresi uzun zaman alabilecek metodların geriye değer döndürdüğü vakkalarda asenkron erişim tekniklerini kullanmak gerekir. Ancak Windows Communication Foundation göz önüne alındığında asenkron çalıştırma iki farklı şekilde ele alınabilmektedir. Bunlarda birisi istemci taraflı asenkron çağırma **(Client Side Asynchronous Invoking)** modelidir. Diğeri ise servis taraflı asenkron uyarlama modelidir **(Service Side Asynchronous Implementation)**. Bu makalemizde istemci taraflı asenkron çağırma modelini incelemeye çalışacağız.

İstemci taraflı asenkron çağırma modelinde, proxy sınıfının asenkron desene **(Asynchronous Pattern)** uygun olacak şekilde **Begin** ve **End** ile başlayan standart metodları vardır. Bu metodlar temelinde **IAsyncResult** arayüzünü ele almaktadır. İstemci taraflı olarak çalışan bu modelin, doğal olarak farklı uygulanabilme çeşitleri vardır. .Net tarafında asenkron mimarinin ele alınabilen tüm teknikleri Windows Communication Foundation içinde geçerlidir. Burada bahsedilen teknikler **Polling**, **Callback** ve **WaitHandle** modellerini içermektedir. WaitHandle modelininde kendi içerisinde **WaitOne**, **WaitAny**, **WaitAll** gibi farklı kullanım şekilleri vardır. Bu modellerin temel etkilerini ve farklılıklarını yazacağımız örnek kod parçaları üzerinde Windows Communication Foundation açısından incelemeye çalışacağız. Ancak başlamadan önce servis taraflı asenkron uyarlama modelinide açıklamakta fayda olacağı kanısındayım. Servis tarafında gerçekleştirilen asenkron uyarlama tekniği istemcinin metodları asenkron olarak işletmesi anlamına **gelmemektedir**. Bir başka deyişle istemci yine çağırdığı metodu senkronmuş gibi ele alır yani ilerleyebilmek için metodun sonucunun gelmesini bekler. Metodun asenkron olarak çalıştığı yer servis tarafıdır. Aslında servis tarafındaki ilgili sürecin başka bir thread üzerine yıkıldığı düşünülebilir. Ne varki bu model kodlanması zor bir teknik içermektedir. Nitekim servis tarafında asenkron olarak ele alınmak istenen metodların asenkron tasarım desenine göre yazılması gerekmektedir. Az öncede belirttiğimiz üzere söz konusu modeli bir sonraki makalemizde incelemeye çalışacağız.

Dilerseniz örneğimize geçerek işlemlerimize başlayalım. Her zamanki gibi servis tarafında yayınlacak olan fonksiyonellikleri içeren tip ve sözleşme (contract) tanımlamalarını içeren bir**WCF Class Library** projesi geliştirerek işe başlanabilir. Söz konusu kütüphane içerisindeki sözleşme arayüzü (Interface) ve fonksiyonellikleri içeren sınıf(Class) aşağıdaki gibidir.



IAdventureManager isimli arayüz içeriği aşağıdaki gibidir.

```
[ServiceContract(Name="AdventureContract",
Namespace="http://www.bsenyurt.com/2007/6/6/AdventureService")]
public interface IAdventureManager
{
   [OperationContract(Name="AverageListPriceByCategory")]
   double AverageListPrice(int subCatId);

   [OperationContract(Name="TotalListPriceByCategory")]
   double TotalListPrice(int subCatId);

   [OperationContract(Name = "GetProductsCountByCategory")]
   int ProductCount(int subCatId);
}
```

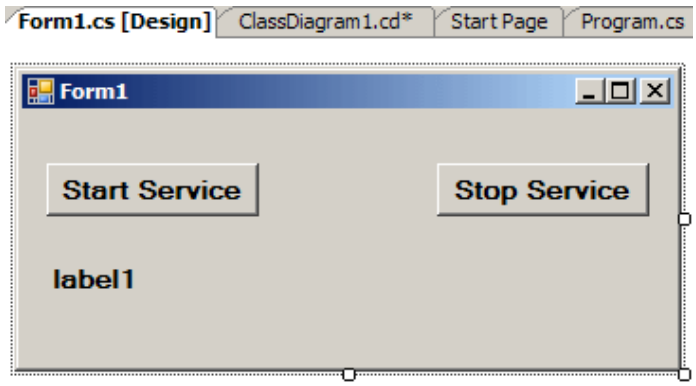
AdventureManager isimli sınıfın içeriği aşağıdaki gibidir.

```
public class AdventureManager:IAdventureManager
{
   #region IAdventureManager Members

   public double AverageListPrice(int subCatId)
   {
      Thread.Sleep(5000);
      return 1000;
   }
   public double TotalListPrice(int subCatId)
   {
```

```
        Thread.Sleep(3000);
        return 4500;
    }
    public int ProductCount(int subCatId)
    {
        Thread.Sleep(7000);
        return 504;
    }

    #endregion
}
```

AdventureManager isimli sınıf içerisinde yer alan metodlarda bilinçli olarak **Thread** sınıfının static **Sleep** fonksiyonundan faydalanılarak farklı sürelerde duraksatmalar yapılmaktadır. Söz konusu metodlardan sembolik olarak double ve int gibi tiplerden değerler döndürülmektedir. Geliştirilen WCF sınıf kütüphanesini kullanacak olan servis tarafını yine bir Windows uygulaması olarak tasarlayabiliriz. *(Windows uygulamasının **System.ServiceModel.dll** ve **AdventureLib** isimli WCF Sınıf kütüphanelerini referans etmesi gerektiğini unutmayalım.)*

Windows uygulamasının form tasarımı basit olarak aşağıdaki gibidir;



Windows uygulamasına ait kodlar aşağıdaki gibidir;

```
ServiceHost host;

private void btnStartService_Click(object sender, EventArgs e)
{
    host = new ServiceHost(typeof(AdventureManager));
    host.Open();
    lblStatus.Text = host.State.ToString();
}

private void btnStopService_Click(object sender, EventArgs e)
```

```
{
   host.Close();
   lblStatus.Text = host.State.ToString();
}
```

Windows uygulamasına ait app.config dosyasının içeriği aşağıdaki gibidir;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <services>
         <service name="AdventureLib.AdventureManager">
            <endpoint address="net.tcp://localhost:9001/AdventureServices.svc"
binding="netTcpBinding" bindingConfiguration="" name="AdventureEndPoint"
contract="AdventureLib.IAdventureManager" />
         </service>
      </services>
   </system.serviceModel>
</configuration>
```

Servis TcpBinding tipine göre geliştirilmiştir. Bu nedenle **net.tcp://localhost:9001/AdventureServices.svc** isimli örnek adres üzerinden sunulmaktadır. Söz konusu konfigurasyon dosyası her zamanki gibi**Microsoft Service Configuration Editor** yardımıyla Visual Studio 2005 ortamında daha kolay bir şekilde yazılabilir.

Gelelim istemci tarafına. Öncelikli olarak **svcutil.exe** aracını kullanarak AdventureLib isimli sınıf kütüphanesinden, istemci için gerekli proxy sınıfı ve konfigurasyon dosyasının üretilmesi gerekmektedir. Burada daha önceki örneklerden farklı olarak asenkron desene uygun olacak şekilde metod üretimlerinin yapılması gerekmektedir. Svcutil aracının **/async** (veya kısaltmalı olarak**/a**) isimli parametresi bu işi otomatik olarak yapmaktadır. Bu nedenle svcutil aracının aşağıdaki şekilde kullanıması gerekmektedir.

Öncelikli olarak proxy üretimi için gerekli olan **wsdl** ve **schema** dosyalarının üretilmesi sağlamak adına aşağıdaki komut kullanılmalıdır.

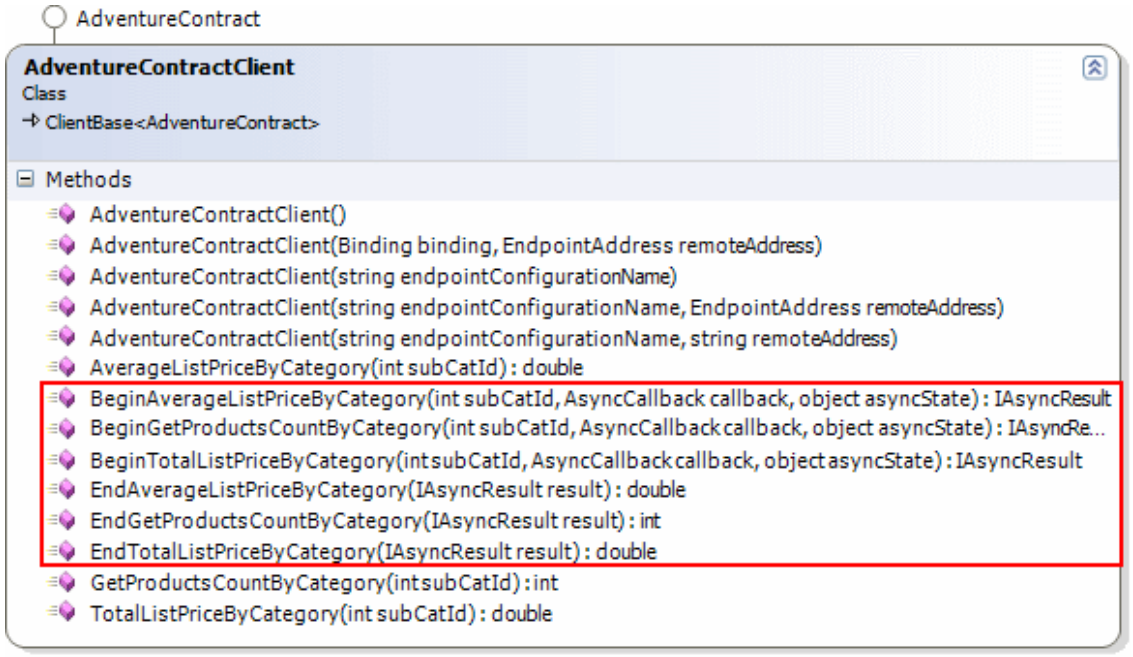
```
svcutil AdventureLib.dll
```

Sonrasında ise svcutil aracı aşağıdaki haliyle çalıştırılmalıdır.

```
svcutil www.bsenyurt.com.2007.6.6.AdventureService.wsdl *.xsd
/out:AdventureProxy.cs /async
```

```
Visual Studio 2005 Command Prompt

E:\Vs2005Projects\WCF Samples\AsenkronErisimler\AdventureLib\bin\Debug>svcutil A
dventureLib.dll
Microsoft (R) Service Model Metadata Tool
[Microsoft (R) Windows (R) Communication Foundation, Version 3.0.4506.30]
Copyright (c) Microsoft Corporation.  All rights reserved.

Generating metadata files...
E:\Vs2005Projects\WCF Samples\AsenkronErisimler\AdventureLib\bin\Debug\www.bseny
urt.com.2007.6.6.AdventureService.wsdl
E:\Vs2005Projects\WCF Samples\AsenkronErisimler\AdventureLib\bin\Debug\www.bseny
urt.com.2007.6.6.AdventureService.xsd
E:\Vs2005Projects\WCF Samples\AsenkronErisimler\AdventureLib\bin\Debug\schemas.m
icrosoft.com.2003.10.Serialization.xsd

E:\Vs2005Projects\WCF Samples\AsenkronErisimler\AdventureLib\bin\Debug>svcutil w
ww.bsenyurt.com.2007.6.6.AdventureService.wsdl *.xsd /out:AdventureProxy.cs /asy
nc
Microsoft (R) Service Model Metadata Tool
[Microsoft (R) Windows (R) Communication Foundation, Version 3.0.4506.30]
Copyright (c) Microsoft Corporation.  All rights reserved.

Generating files...
E:\Vs2005Projects\WCF Samples\AsenkronErisimler\AdventureLib\bin\Debug\Adventure
Proxy.cs
E:\Vs2005Projects\WCF Samples\AsenkronErisimler\AdventureLib\bin\Debug\output.co
nfig

E:\Vs2005Projects\WCF Samples\AsenkronErisimler\AdventureLib\bin\Debug>
```

Sonuç olarak üretilen AdventureProxy.cs isimli sınıfın içeriğine aşağıdaki ekran görüntüsünde olduğu gibi, **Begin** ve **End** ile başlayan standart asenkron metodlar ilave edilmiş olur. Dikkat edilecek olursa AdventureManager sınıfı içerisindeki tüm metodların hem normal hemde Begin ve End ile başlayan versiyonları ilave edilmiştir.



Oluşan bu sınıf içerisinden örnek olarak AverageListPriceByCategory isimli fonksiyon için yazılmış asenkron metodlar göz önüne alınabilir.

```
public System.IAsyncResult BeginAverageListPriceByCategory(int
subCatId, System.AsyncCallback callback, object asyncState)
{
   return base.Channel.BeginAverageListPriceByCategory(subCatId, callback,
asyncState);
}

public double EndAverageListPriceByCategory(System.IAsyncResult result)
{
   return base.Channel.EndAverageListPriceByCategory(result);
}
```

Burada tipik olarak asenkron tasarım desenine uygun metodlar yer almaktadır. **BeginAverageListPriceByCategory** metodu **Polling**, **Callback** ve **WaitHandle** modellerine destek verecek şekilde**IAsyncResult** arayüzünün(Interface) taşıyabileceği bir referansı döndürür. Kullanılan modele göre, asenkron çalışan fonksiyonun sonuçlarını almak için**EndAverageListPriceByCategory** metodu kullanılır. Dikkat edilecek olursa bu metod parametre olarak IAsyncResult arayüzünden bir referans kabul eder ve geriye uygun olan metod çıktısını döndürür. Metod içerikleri asenkron deseni otomatik olarak uygulamaktadır. Bir başka deyişe nesne kullanıcısı **(object user)** söz konusu modelin içerisindeki detaylar ile ilgilenmez. Sadece uygun olan veya istediği asenkron modeli istemci programa uyarlar.

Şimdi tek tek istemci tarafından asenkron çağırma modellerini uygulamaya çalışalım. İstemci basit bir Console uygulaması olarak ele alınmıştır. Söz konusu uygulamanın konfigurasyon dosyasının içeriği aşağıdaki gibi geliştirilebilir. *(İstemci uygulamanında **System.ServiceModel.dll** assembly' ını referans etmesi gerektiğini unutmayalım.)*

İstemci uygulama tarafında ele alınacak app.config içeriği aşağıdaki gibidir.

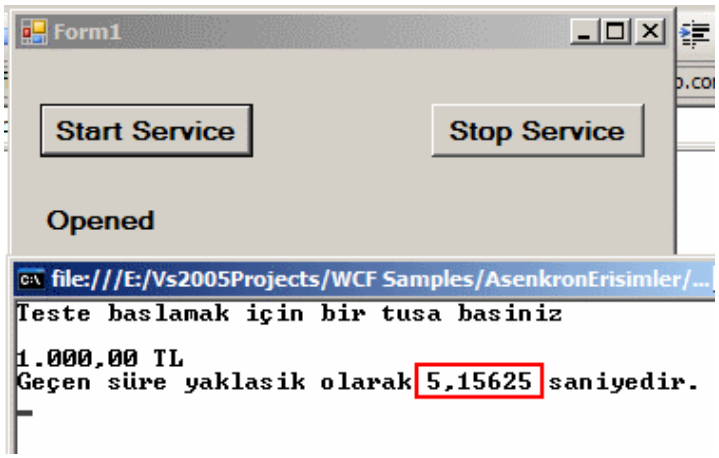
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <system.serviceModel>
      <bindings/>
      <client>
         <endpoint address="net.tcp://localhost:9001/AdventureServices.svc" binding="
netTcpBinding" bindingConfiguration="" contract="AdventureContract"
name="AdventureClientEndPoint" />
      </client>
   </system.serviceModel>
</configuration>
```

Başlamadan önce senkron çalışmanın nasıl bir etkisi olacağını görmekte fayda vardır. Bu amaçla ilk kodlar aşağıdaki gibi geliştirilebilir.

Senkron çalışma durumu;

```
Console.WriteLine("Teste başlamak için bir tuşa basınız");
Console.ReadLine();
AdventureContractClient srv = new
AdventureContractClient("AdventureClientEndPoint");

#region Senkron Çalışma

DateTime baslangic = DateTime.Now;

double ortalamaFiyat=srv.AverageListPriceByCategory(1);

DateTime bitis = DateTime.Now;

Console.WriteLine(ortalamaFiyat.ToString("C2"));
TimeSpan fark = bitis - baslangic;
Console.WriteLine("Geçen süre yaklaşık olarak {0}
saniyedir.",fark.TotalSeconds.ToString());
Console.ReadLine();

#endregion
```

Burada dikkat edilecek olursa AverageListPriceByCategory metodu çağırıldıktan sonra istemci uygulama bir süreliğine beklemede kalacaktır. Bu **TimeSpan** ile elde edilen süre farkından da açıkça görülmektedir. Bir başka deyişle istemci uygulamadaki kod akışı ortalamaFiyat değerinin elde edilmesini bekleyecek ve sonuç alındıktan sonra devam edecektir. Program kodu bu haliyle çalıştırıldığında aşağıdakine benzer bir ekran görüntüsü elde edilir.



Bu tipik olarak senkron çalışma şeklidir. Gelelim diğer tekniklere. İlk olarak **WaitHandle** modelini ele alalım. Bu model temel olarak asenkron olarak çalıştırılan metodların, uygulamanın belirli bir noktasında sonuçları alınıncaya kadar farklı şekillerde

beklenilmesini sağlamaktadır. Dikkat ederseniz metodlar yine asenkron olarak başlatılır ama programın herhangibir noktasında sonuçlarının ortama dönmesi için beklenir. Bu model daha çok asenkron çalışan metodların sonuçlarının uygulamanın belirli bir noktasında girdi olarak kullanılması gerektiği durumlarda işe yaramaktadır. Söz konusu modelin **WaitOne**, **WaitAll** ve **WaitAny** gibi üç farklı uygulanış biçimi vardır. **WaitOne** tekniği adındanda anlaşılacağı üzere sadece tek bir asenkron metodun sonucunun alınması için bir duraksatma gerçekleştirir. Aşağıdaki örnek kod parçasında WaitOne tekniğinin uygulanış biçimi yer almaktadır.

WaitOne tekniği;

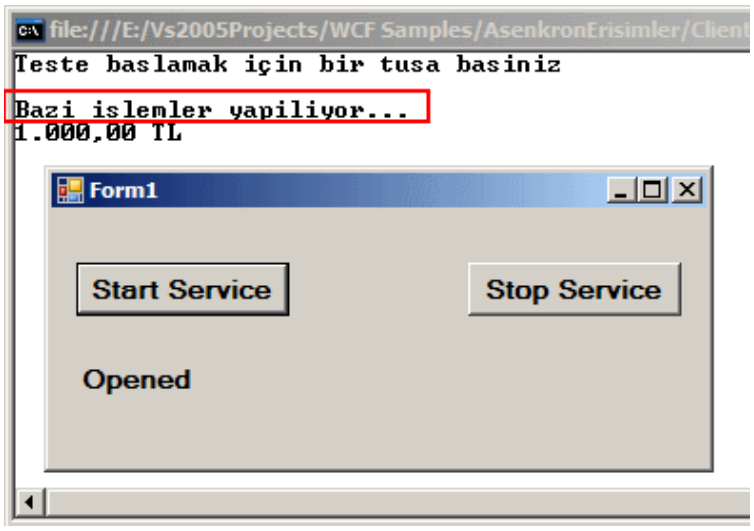
IAsyncResult iar=srv.**BeginAverageListPriceByCategory(1, null, null);**
Console.WriteLine("Bazı işlemler yapılıyor...");

iar.**AsyncWaitHandle.WaitOne();**

double sonuc=srv.EndAverageListPriceByCategory(iar);
Console.WriteLine(sonuc.ToString("C2"));

Console.ReadLine();

Uygulama bu haliyle çalıştırıldığında aşağıdaki ekran görüntüsü elde edilir.



Görüldüğü gibi BeginAverageListPriceByCategory çağrısından sonra uygulama hemen alttaki satırdan çalışmaya devam etmiştir. Burası tamamen sembolik bir kod parçası içerir. Çok doğal olarak burada farklı işlemler gerçekleştirilmesi veya istemci tarafında yer alacak başka fonksiyonelliklerin ele alınması muhtemeldir. Sonrasında ise iar isimli **IAsyncResult** referansının **AsyncWaitHandle** özelliği ile yakalanan **WaitHandle** nesne örneğinin **WaitOne** metodu çağırılır. Bu metod, çalıştırılan asenkron fonksiyonun sonucu alınana kadar uygulamanın duraksatılmasını sağlar. WaitOne satırı aşılır aşılmaz artık ilgili metodun sonuçları uygulama ortamına derhal alınabilir. Bu

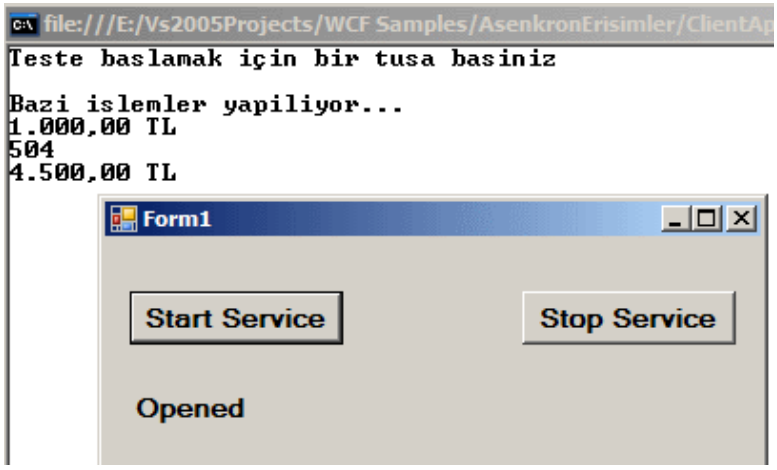
nedenle EndAverageListPriceByCategory metodunun çağırılması ve parametre olarak iar isimli **IAsyncResult** arayüzünün verilmesi yeterlidir.

Her zaman için istemci tarafından çağırılacak tek bir asenkron metod olması söz konusu değildir. Bazı durumlarda birden fazla metod çağrısı asenkron olarak yürütülmek istenebilir. Çok doğal olarak bu metodlarının tamamının, yine uygulamanın belirli bir noktasında girdi olarak kullanılacak dönüş değerleri olabilir. Öyleyse programın bu ilgili noktasında asenkron olarak çalışan tüm metodların tamamının duraksatılması istenebilir. Bunun için **WaitAll** tekniği aşağıdaki gibi kullanılır.

WaitAll tekniği;

```
IAsyncResult iar1 = srv.BeginAverageListPriceByCategory(1, null, null);
IAsyncResult iar2 = srv.BeginGetProductsCountByCategory(2, null, null);
IAsyncResult iar3 = srv.BeginTotalListPriceByCategory(5, null, null);

WaitHandle[] handles = new WaitHandle[] { iar1.AsyncWaitHandle,
iar2.AsyncWaitHandle,iar3.AsyncWaitHandle };

Console.WriteLine("Bazı işlemler yapılıyor...");

WaitHandle.WaitAll(handles);

double sonuc1 = srv.EndAverageListPriceByCategory(iar1);
int sonuc2 = srv.EndGetProductsCountByCategory(iar2);
double sonuc3 = srv.EndTotalListPriceByCategory(iar3);

Console.WriteLine(sonuc1.ToString("C2"));
Console.WriteLine(sonuc2.ToString());
Console.WriteLine(sonuc3.ToString("C2"));

Console.ReadLine();
```

WaitAll tekniğinde asenkron olarak çalıştırılan metodlardan sorumlu **IAsyncResult** referanslarından elde edilen her bir **WaitHandle** örneği bir dizi içerisinde toplanır. Söz konusu dizi WaitHandle sınıfının **static WaitAll** metoduna devredildiği satırda uygulama tüm asenkron metodların sonuçlarının gelmesi için beklemede kalacaktır. Çok doğal olarak bu çağrıya kadar yapılan tüm işlemler, asenkron metodlarınki ile paralel olarak yürütülmektedir. Bu program kodlarının çalışma zamanında üreteceği çıktı aşağıdaki gibi olacaktır.



WaitAny modeli asenkron olarak çalışan metodlardan tamamlanını ortama iade edebilme ilkesine dayalı olarak çalışmaktadır. Söz gelimi örnek servisimizdeki metodlar göz önüne alındığında teorik olarak en kısa sürede biten metodun sonucunun ortama alınabilmesi ve sonrasında diğerleri için beklennmesi gerekir. Metod sonuçları ortama döndükçe asenkron işleyişler tamamlanmış olacaktır.

NOT : *WaitAny modelini WCF içerisinde uyguladığımızda çalışma zamanında* ***ObjectDisposedException*** *istisnası (Exception) alınmaktadır. Bu istisnanın sebebi araştırıldığında, makalenin yazıldığı tarih itibariyle çok kıstılı bilgiye ulaşılmaktadır. Bir çözüm Erwyn van der Meer tarafından geliştirilmiştir ve değerlendirilebilir. Buna göre Dispose edilemeyen servis nesnesi için ekstradan bir sınıf geliştirilmiş ve bu sınıfın proxy sınıfı yerine kullanılması önerilmiştir.*

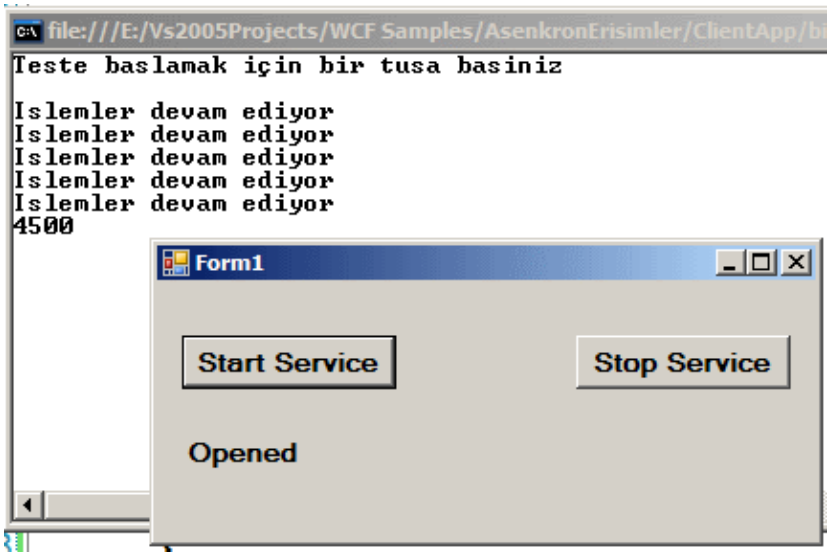
İstemci tarafı için asenkron çağırma tekniklerinden biriside **Polling** ' dir. Bu modelde temel olarak asenkron olarak başlatılan işlemin tamamlanıp tamamlanmadığı kontrol edilir ve bu aralıktaki tüm işlemler paralel olarak işletilir. Polling modelinde, asenkron işleyişin tamamlanmadığını kontrol etmek adına **IAsyncResult** arayüzünün **IsCompleted** isimli özelliğinden yararlanılır.

Polling tekniği;

```
IAsyncResult iar=srv.BeginTotalListPriceByCategory(4, null, null);

while (!iar.IsCompleted)
{
   Console.WriteLine("İşlemler devam ediyor");
   Thread.Sleep(1000);
}

double sonuc = srv.EndTotalListPriceByCategory(iar);
```

```
Console.WriteLine(sonuc.ToString());
Console.ReadLine();
```

Kodlar bu haliyle çalıştırıldığında aşağıdaki ekran görüntüsü elde edilir.



Buna göre while döngüsü içerisinde kodlar asenkron olarak yürütülen metoddan sonuç alınıncaya kadar devam edecektir.

Asenkron erişim teknikleri arasında en popüler olanlarından birisi **Callback**' tir. Bu teknikte asenkron çalışan metodun işleyişi tamamlandığında otomatik olarak bir geri bildirim fonksiyonu devreye girer ve sonuçların uygulama ortamına kolay bir şekilde alınabilmesi sağlanmış olur. Aşağıdaki kod parçasında Callback modelinin uygulanış biçimi gösterilmektedir.

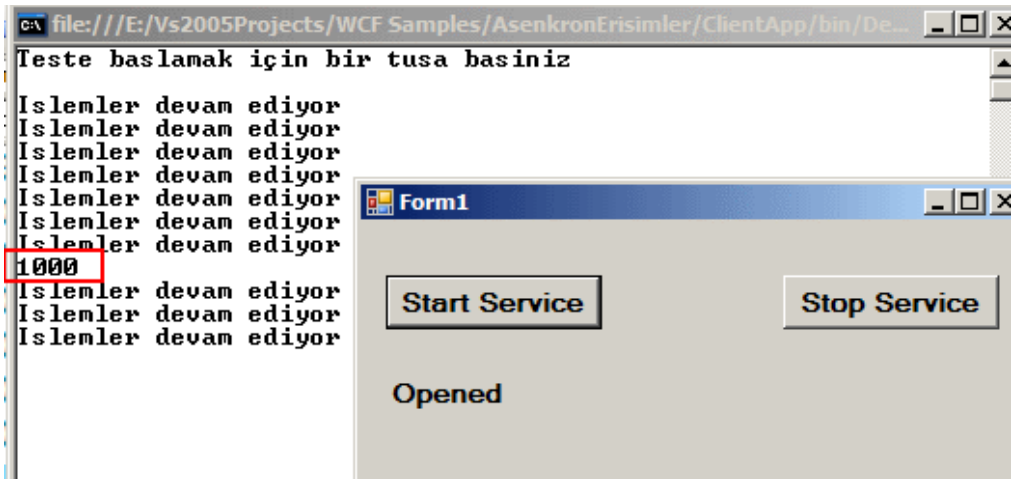
Callback Modeli;

```
class Program
{
   static void Main(string[] args)
   {
      Console.WriteLine("Teste başlamak için bir tuşa basınız");
      Console.ReadLine();
      AdventureContractClient srv = new
AdventureContractClient("AdventureClientEndPoint");

      #region Callback Ornek

      IAsyncResult iar = srv.BeginAverageListPriceByCategory(3, new
AsyncCallback(CallbackMetod), srv);
```

```
        for (int i = 0; i < 10; i++)
        {
            Console.WriteLine("İşlemler devam ediyor");
            Thread.Sleep(1000);
        }

        Console.ReadLine();

        #endregion
    }

    static void CallbackMetod(IAsyncResult iar)
    {
        AdventureContractClient srv = (AdventureContractClient)iar.AsyncState;
        double sonuc=srv.EndAverageListPriceByCategory(iar);
        Console.WriteLine(sonuc.ToString());
    }
}
```

Callback modelinde kritik olan nokta Begin... metodunun aldığı **AsyncCallback** tipindeki parametredir. AsyncCallback .Net içerisinde yer alan bir temsilcidir (**delegate**). Bu temsilcinin temel görevi ise çalışma zamanında otomatik olarak çağırılacak geri bildirim metodunu işaret etmektir. Bir başka deyişle asenkron olarak çağırılan metod işleyişini tamamlandığında bu temsilcinin bildirdiği metod devreye girecektir. Sonuç itibariyle **AsyncCallback** bir temsilci olduğundan tanımında işaret edeceği metodun yapısıda belirtilmektedir. Buna göre geriye değer döndürmeyen ve **IAsyncResult** arayüzü tipinden referanslar alan metodlar işaret edilebilir.

Begin metdunun son parametresi object tipinden bir değer alır. Bu parametre çoğunlukla geri bildirim metoduna referans taşımak amacıyla kullanılır. Söz gelimi yukarıdaki örnek kod parçasında, End... metodunun çağırılabilmesi için iar üzerinden **AsyncState** ile elde edilen referans **AdventureContractClient** sınıfına **cast** edilmektedir. Burada AsyncState özelliğinin Begin... çağrısında kullanılan srv isimli referans olmasını sağlamak amacıyla son parametreye srv örneği verilmiştir. Uygulama çalıştığında paralel olarak yürüyen istemci kodları devam ederken aynen aşağıdaki ekran görüntüsünde olduğu gibi arada bir yerde, tamamlanan asenkron metodun sonucu otomatik olarak ortama alınabilmektedir.



Callback modelinde istenirse C# 2.0 ile birlikte gelen **isimsiz metodlardan (Anonymous Methods)** da yararlanılabilir. Bu sayede ekstradan Callback metodu yazılmasına gerek kalmamakta ve temsilcinin bağlandığı yerde geri bildirim kodları ele alınabilmektedir. Örneğin aşağıdaki kod parçasında bu işlemin nasıl yapılacağı gösterilmektedir.

Callback modelinde isimsiz metod kullanımı;

```
AsyncCallback async = delegate(IAsyncResult ar)
                      {
                         double sonuc = srv.EndAverageListPriceByCategory(ar);
                         Console.WriteLine(sonuc.ToString());
                      };
IAsyncResult iar = srv.BeginAverageListPriceByCategory(3, async, null);

for (int i = 0; i < 10; i++)
{
   Console.WriteLine("İşlemler devam ediyor");
   Thread.Sleep(1000);
}

Console.ReadLine();
```

Bu seferki modelde ekstradan **static***(Console uygulamasındaki static Main metodundan çağırmamız nedeni ile böyle tanımlanmak zorundadır)* olacak şekilde bir geri bildirim metodu yazılmasına gerek kalmamıştır. Bununla birlikte, Begin metodunun son paramertresi ile bir object referansı taşınmasına gerekte yoktur. Bu kod parçası çalıştırıldığında da benzer sonuçlar alınacaktır.

Bu makalemizde WCF için istemci taraflı asenkron çağırma modelini incelemeye çalıştık. Temel olarak kullanabileceğimiz üç modelden bahsettik. Bu modeller ile ilgili olarak kısaca aşağıdaki özet bilgileri söyleyebiliriz.

- **Polling** modelinde asenkron çağırılar sonucu çalışan metodların tamamlanıp tamamlanmadığı sürekli olarak kontrol edilir. Bu amaçla **IsCompleted** özelliği ele alınabilir. Bu kontrol aralığındaki tüm işlemler ilgili asenkron çağrılar ile paralel olarak yürümektedir.
- **Callback** modelinde asenkron olarak yapılan çağrılar ile çalışan metodların sonuçları elde edildiğinde, otomatik olarak bir geri bildirim metodu çalışır. Dolayısıyla asenkron yürüyen metodların tamamlanıp tamamlanmadıklarının sürekli olarak kontrol edilmesine gerek yoktur.
- **WaitHandle** modeli 3 farklı şekilde uygulanabilmekte olup asenkron olarak çalışan metodların uygulamanın belirli bir noktasında girdi olarak kullanılabilecek değerlerinin alınması için bekleme yapılmasını sağlamaktadır. Bu bekleme tek bir metod için **WaitOne**, tüm metodlar için **WaitAll** ve sırayla bitenleri ortama alma ilkesine dayanaraktan **WaitAny** fonksiyonları ile yapılmaktadır.

Böylece geldik bir makalemizin daha sonuna. Yazımızın başındada belirttiğimiz gibi bir sonraki Windows Communication Foundation makalemizde servis tarafında asenkron uyarlamanın (Service Side Asynchronous Implementation) nasıl yapılabileceğini incelemeye çalışacağız. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

Örnek Uygulama İçin Tıklayınız.

C# Temelleri - Olayları(Events) Kavramak (2007-06-06T19:01:00)

c# temelleri,c#,

Olaylar (Events), görsel uygulamalar ile uğraşan her geliştirici tarafından bilinçli veya bilinçsiz bir şekilde kullanılmaktadır. Nesne yönelimli programlama ortamında olayları tanımlamak için klasik olarak verilen bir örnek vardır. Hepinizin bir sonraki cümlede ne diyeceğimi bildiğinizden eminim. Söz konusu örnekte görsel ortamda yer alan bir düğme kontrolü (çoğunlukla Button sınıfına ait bir nesne örneği) ve bu düğmeye kullanıcının mouse ile basması sonucu oluşan Click isimli bir olay mevcuttur. Oysaki olayların her zaman için görsel ortamda olması ve işletim sistemi tarafından algılanacak bir etkileşime karşılık olarak uygulama ortamına fırlatılması şart değildir. Dolayısıyla olayları kavrayabilmenin en güzel yolu, geliştirici tarafından yazılan tiplere özel olarak nasıl yazılacağını ve kullanılacağını bilmekle mümkün olabilir. İşte bu makalemizde kendi tiplerimiz için özel olayları nasıl yazabileceğimizi incelemeye başlayacak ve olayları daha derinlemesine kavramaya çalışacağız.

öncelikli olarak düğme kontrolü örneğinden sıyrılıp farklı vakkalar düşünerek ilerlemeye çalışalım. örnek olarak stoktaki ürün bilgilerini içeren basit bir sınıf göz önüne alınabilir. ürünlerin stoktaki değerlerininde bu tip içerisinde bir özellik yardımıyla sarmalanmış(wrap) olduğunu düşünebiliriz. Buna göre stok miktarının belirli bir değerin altına düşmesi sonrasında başka nesne örnekleri tarafından ele alınabilecek bir olay tanımlaması

yapılabilir. Burada Urun tipi içerisinde tanımlanan ve bu tipe ait nesne örneklerinin ele alabileceği bir olay söz konusudur.

Başka bir örnek daha. Herhangibir bir kargo şirketinin ulaştırma filosundaki araçların programatik ortamda birer nesne ile ifade edilebildiği bir kütüphane(library) olduğunu düşünelim. Bu kütüphane içerisindeki fonksiyonelliklerden biriside, araçların uydu sistemleri yardımıyla düzenli olarak izlenmesi ve güncel koordinat, anlık hız gibi bilgilerinin elde edilmesi olarak göz önüne alınabilir. Bu hizmeti sağlayan kodlar ayrı bir kütüphane olacak şekilde geliştirilmiş olarak ticari bir paket halinde sunulabilir. O halde araçların belirlenen hız limitlerini aşmaları sonrasında oluşacak durumların söz konusu kütüphaneyi kullanan uygulamalar tarafından, istenirse ele alınmalarını sağlamak amacıyla olaylar yazılabilir. Böylece söz konusu program, araç hız limitini aştığı zaman neler yapmak istiyorsa bunları istediği şekilde ele alabilecektir.

Temel olarak kendi tiplerimiz için olay tanımlamak aslında **temsilcileri (delegates)** daha kolay kapsüllenmiş bir halde sunmak şeklinde de yorumlanabilir.
Bir **olayın(event)** tanımlanabilmesi için mutlaka bir temsilci tipi ile eşleştirilmesi gerekmektedir.

NOT : *Temsilciler(delegates)* ***çok kanallı programlamada (multi threading)****,* ***asenkron(asynchronous) mimarilerde(Polling, Callback, WaitHandle gibi)*** *ve son olarak* ***olay tabanlı(event based)****kodlamada kullanılmaktadır.*

Temsilci dışında dikkat edilmesi gereken bir diğer noktada olayın bir şekilde ortama fırlatılmasını sağlamaktır. Düğme örneğini burada göz önüne alabiliriz. Dikkat ederseniz bir düğmeye basıldığında gerçekleştirilmek istenenleri yazmak için tek yapılan oluşan olay metodunun içeriğini doldurmaktan ibarettir. Sistem arka tarafta söz konusu Button nesne örneği için bir olay yüklemsi yapmaktadır. Peki düğmeye basıldığında söz konusu olay metodu nasıl çağırılacaktır? *(Burada temsilcinin rolünün ne kadar önemli olduğu ortadadır.)* Olayın tetiklenmesi işletim sistemi tarafından gerçekleştirilir. Aslında Button nesne örneğinin arka planda yaptıklarından biriside, işletim sistemindeki bu aksiyonu yakalamaktır. Sonuç itibariyle kullanıcının söz konusu Button nesne örneğine yükleme yaptığı olay metodu çağırılır. Bu anlatılanlar bize şunu ifade etmelidir. Kendi olaylarımızı tanımlıyorsak, söz konusu olayın diğer nesneler tarafından ele alınabilmesini sağlamak için manuel olarak tetiklemeliyiz. Manuel olarak yapılan bu tetiklemenin sonucunda çalışma zamanında ele alınacak olay metodunun işaret edilmesini
ise, **temsilciler(delegates)** yardımıyla sağlamalıyız.

NOT : *Kendi tiplerimiz için olay tanımlıyorsak bu olayın çalışma zamanında diğer bir nesne tarafından ele alınabilmesi için bir şekilde tetiklenmesi gerekmektedir.*

Aslında bir olayın tetiklenmesi, bir **istisna(exception)** nesne örneğinin ortama fırlatılmasına**(throw)** benzetilebilir. Tek fark ortama fırlatılan istisnaların **catch** blokları ile yakalanabiliyor olmasıdır. Olaylarda durum farklıdır. Ortada bir catch bloğu yoktur. Bunun yerine bir **abone (subscriber)** vardır. Basit olarak olayın tetiklenmesi sonucu çalıştırılacak

olay metodunun bulunduğu nesne örneğini abone olarak düşünebiliriz. Bir başka deyişle olayı yakalayıp değerlendirecek olan nesne, olayın sahibi olan nesnenin ilgili **olayına(event)** abone olmaktadır. Dolayısıyla olayı tanımlayan ve tetikleyen nesneyi **yayımcı (publisher)** olarakda göz önüne alabiliriz. Aslında bahsettiğimiz kavramları daha kolay anlayabilmek amacıyla aşağıdaki grafiği incelemekte fayda vardır.

Birinci adıma göre Program nesnesi Urun tipine ait bir nesne örneği oluşturur. Program nesne örneğinden kasıt aslında uygulamanın ta kendisi olabilir. örneğin bir konsol uygulamasındaki Program sınıfı veya windows uygulamasındaki bir Form nesne örneği olabilir. Hangisi olursa olsun değişmez gerçek olaya abone olmak isteyen bir nesnenin olmasıdır. Olay tanımlamamızın Urun tipi içerisinde olduğu varsayılırsa, Program nesnesinin ilgili olay metodunu Urun nesnesine abone etmesi gerekmektedir. Bu ikinci adımda sembolize edilmeye çalışılmaktadır.

Nesne kullanıcısı (Object User), Program nesnesi içerisinde olay yüklemesi ile birlikte bir olay metodunuda yazar. Böylece Urun nesnesinin üçüncü adımda yapacağı tetikleme sonucunda Program nesnesi tarafından yazılan olay metodu çağırılabilecektir. Burada söz konusu olan abone etme işlemi aslında Urun sınıfına ait **olayın(event) +=** operatörü yardımıyla Program nesne örneği içerisinde yüklenmesidir. Olaylar tanımlanırken hep bir temsilci(delegate) tipi yardımıyla oluşturulurlar. Dolayısıyla Program sınıfı içerisinde Urun nesnesi için ilgili olay += operatör ile yüklendiğinde, **temsilciye(delegate)** parametre

olarak verilen metod referansı Urun nesnesine bildirilir. Böylece Urun nesne örneği içerisinde ilgili olay tetiklendiğinde hangi metodun çağırılacağı bilinmektedir.

Bu kadar teorik bilgiyle aslında, olayların gerçek anlamda sadece button ve click kelimelerinden ibaret olmadığını göstermeye çalıştık. Artık birazda pratik yaparak bahsedilenleri örneklemekte fayda olacağı kanısındayım. Bu amaçla basit bir console uygulamasını göz önüne alacağız. Konsol uygulamamız içerisinde yer alan Program sınıfımız**abonemiz(subscriber)** olacak. Urun sınıfı içerisinde tanımlayacağımız olay(Event), stoktaki ürün sayısı 10 değerinin altına düştüğünde tetiklenecek şekilde tasarlanacaktır. Bir olay tanımlanırken mutlaka bir temsilcinin olması gerektiğinden bahsetmiştik. Dolayısıyla birde **temsilci (delegate)** tipi geliştirmemiz gerekecektir. Söz konusu temsilci tipini ve Urun sınıfını aşağıdaki gibi tasarlayabiliriz. *(örneklerimizde sadece olay kavramına yoğunlaşmak istediğimizden, yapılması gereken pek çok kontrol ortadan kaldırılmıştır. örneğin ürün adının boş geçilmesini, StokMiktari veya BirimFiyat özelliklerine sıfırın altında değer atanmasını engellemek gibi. Daha pek çok kontrol ve fonksiyonellik düşünülebilir elbette. Siz kendi uygulamalarınızda bu noktaları sakın gözden kaçırmayın ve mutlaka uygulayın.)*

```
using System;

namespace Olaylar
{
   delegate void StokAzaldiEventHandler();
```

```
class Urun
{
   private int id;
   private string ad;
   private double birimFiyat;
   private int stokMiktari;

   public event StokAzaldiEventHandler StokAzaldi;

   public int StokMiktari
   {
      get { return stokMiktari; }
      set {
            stokMiktari = value;
            if (value < 10
               && StokAzaldi != null)
                  StokAzaldi();
         }
   }

   public double BirimFiyat
   {
      get { return birimFiyat; }
      set { birimFiyat = value; }
   }

   public string Ad
   {
      get { return ad; }
      set { ad = value; }
   }

   public int Id
   {
      get { return id; }
      set { id = value; }
   }

   public Urun(int idsi, string adi, double fiyati, int stokSayisi)
   {
      Id = idsi;
      Ad = adi;
      BirimFiyat = fiyati;
      StokMiktari = stokSayisi;
   }
```

```
    }
}
```

Şimdi kodlarımızda neler yaptığımıza kısaca bakalım. İlk olarak **StokAzaldiHandler** tipinden bir **temsilci (delegate)** tanımlıyoruz. Hatırlayacağınız gibi zaman zaman .Net içerisinde var olan isimlendirme standartlarından bahsediyoruz. Niteliklere ait sınıf adlarının **Attribute** kelimesi ile, **istisna(exception)** tiplerinin **Exception** kelimesi ile bittiklerini biliyoruz. özellikle olaylar ile ilişkili temsilcilerinde çoğunlukla **EventHandler** kelimesi ile bittiğini görürüz. Bu nedenle olay ile ilişkili temsilcimizi **StokAzaldiEventHandler** olarak isimlendirdik. StokAzaldiEventHandler isimli temsilci tipi, geriye değer döndürmeyen ve parametre almayan metodları işaret edebilecek şekilde tasarlanmıştır.

NOT : *Temsilcilerin (delegate) çalışma zamanında metodların başlangıç adreslerini işaret ettiklerini ve işaret edebileceği metodun parametrik yapısı ile geri dönüş tipini belirtiklerini hatırlayalım.*

Gelelim Urun sınıfımıza. Urun sınıf içerisinde UrunAzaldi isimli bir olay(event) tanımlanmıştır.

public event StokAzaldiEventHandler StokAzaldi;

Dikkat edilecek olursa **event** anahtar kelimesinden sonra **StokAzaldiEventHandler** isimli temsilci tipi gelmektedir. Son olarakta olayın adı yer alır. Böylece söz konusu olay için çalıştırılabilecek olay metodlarının yapısını StokAzaldiEventHandler isimli temsilcinin söyleyeceğide belirtilmiş olur. Geriye kalan tek pürüz, ilgili olay metodun nasıl ve nerede tetikleneceğidir. örnek olması açısından **StokMiktari** isimli özelliğin **set** bloğunda aşağıdaki kod parçası kullanılmıştır.

```
set {
   stokMiktari = value;
   if (value < 10
      && StokAzaldi != null)
         StokAzaldi();
   }
```

Burada stok miktarı eğer 10 rakamının altındaysa ve StokAzaldi olayı **null** değere eşit değilse StokAzaldi() isimli bir metod çağrısı yapılmaktadır. StokAzaldi olayının **null**olmaması bir şekilde **+=** operatörü ile yüklendiği anlamına gelmektedir. Yani başka bir nesne bu olaya kendisini **abone(subscribe to)** etmiştir. Bu durumda söz konusu olay metodunun buradaki set bloğu içerisinden çağırılması gerekir. Bu iş için yine olayı sanki bir metodmuş gibi çağırmak yeterli olacaktır. Nitekim bu çağrı **+=** operatörü ile bağlanan olay metodunun yürütülmeye başlanması anlamınada gelmektedir. Buraya kadar += operatörü ile olayın yüklenmesi gerektiğinden bahsedip durduk. Peki bu nasıl gerçekleştiriliyor? Cevap aşağıdaki kod parçasında olduğu gibidir.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace Olaylar
{
   class Program
   {
      static void Main(string[] args)
      {
         Urun ciklet = new Urun(10001, "Tipitipitip", 1.20, 35);
         ciklet.StokAzaldi += new StokAzaldiEventHandler(ciklet_StokAzaldi);

         for (int i = 0; i <5; i++)
         {
            ciklet.StokMiktari -= 7;
            Thread.Sleep(600);
            Console.WriteLine(ciklet.Ad + " için stok miktarı " +
ciklet.StokMiktari.ToString());
         }
      }

      static void ciklet_StokAzaldi()
      {
         Console.WriteLine("Stok miktarı 10 değerinin altında...Alarrrmmm!");
      }
   }
}
```

Main metodu içerisinde Urun sınıfına ait bir nesne örneklendikten sonra **StokAzaldi** isimli olay, **+=** operatörü ile yüklenmektedir. Burada Visual Studio kullanılıyorsa += işaretinden sonra iki kez**tab** tuşuna basmak yeterli olacaktır. Bu durumda Visual Studio otomatik olarak bir olay metodu oluşturacaktır. örneğimizde bu olay metodu **ciklet_StokAzaldi** adıyla anılmaktadır.

NOT : *Aslında bir olay tanımlandığında, bu olayın sahibi olan tip için* ***CIL (Common Intermediate Language)*** *kısmınada add_OlayAdı ve remove_OlayAdı isimli iki metod*

eklenir. Bu metodlar içerisinde olay yüklemesi yapıldığında veya çıkartıldığında, gereken temsilci bağlama ve ayırma işlemleri yapılmaktadır.

Program kodu içerisinde test amacıyla StokMiktari özelliğinin değeri 7şer 7şer azaltılmaktadır. Sonuçta ekran çıktısı aşağıdaki gibi olacaktır.

Dikkat edilecek olursa StokMiktari özelliğinin 10 değerinin altında olduğu her durum için otomatik olarak olay metodu tetiklenmiş ve içerisinde yazılan kod parçaları çalıştırılmıştır.

Yazılan olay her ne kadar faydalı görünsede bazı eksiklikleri olduğu ortadadır. örneğin olay metodu içerisinde bir de stoğun o anki miktarının ne olduğunu öğrenebilsek fena olmaz mıydı acaba? Yada birden fazla Urun nesne örneğini aynı olay metoduna bağlayacaksak (ki bu mümkündür), olay metodu içerisinde hangi Urun nesne örneğinin ilgili olayın sahibi

olduğunu tespit edebilsek fena olmaz mıydı? İşte bu iki gereksinime benzer ihtiyaçlar, .Net içerisinde var olan tüm olaylar içinde geçerlidir. O nedenle önceden tanımlanmış olan tüm olaylar aslında standart olarak iki parametre almaktadır. İlk parametre olayı tetikleyen nesne örneğine ait referansın yakalanması için kullanılırken, ikinci parametre olay metodu içerisine bilgi aktarmak maksadıyla ele alınır. Bu standart bir olay temsilcisinin işaret edeceği metodun parametrik yapısıdır. Tahmin edileceği gibi ilk parametre object tipindendir. İkinci parametre ise genellikle **EventArgs** gibi kelimeler ile biten özel bir sınıftır. Kendi örneğimizi göz önüne aldığımızda öncelikli olarak, olay metoduna özel bilgilerin aktarılmasını sağlayacak şekilde bir tipin geliştirilmesi gerekmektedir.

```
class StokAzaldiEventArgs:EventArgs
{
   private int guncelStokMiktari;

   public int GuncelStokMiktari
   {
      get { return guncelStokMiktari; }
      set { guncelStokMiktari = value; }
   }
   public StokAzaldiEventArgs(int gStk)
   {
      GuncelStokMiktari = gStk;
   }
}
```

StokAzaldiEventArgs isimli tipin tek yaptığı, güncel stok miktarını ilgili olay metodu içerisine taşımaktır. Bu tip asıl StokAzaldi olayı için anlamlıdır. Olay argümanlarını taşıyacak kendi tiplerimizi geliştirdiğimizde bunların **EventArgs** tipinden türetilmesi bir zorunluluk değildir ancak bir gelenektir. Amaç aynen isimlendirme kurallarında olduğu gibi kodun standardize edilmesidir. Nitekim .Net içerisindeki tüm olay argüman tipleri, bir şekilde EventArgs sınıfından türemektedir. Bu şekilde olay metoduna bilgi taşıyabileceğimiz bir tip tanımladıktan sonra temsilcininde aşağıdaki gibi değiştirilmesi gerekmektedir.

```
delegate void StokAzaldiEventHandler(object sender,StokAzaldiEventArgs args);
```

Elbette temsilcide yapılan değişikliklerin olayın tetiklendiği yerede adapte edilmesi gerekmektedir. Bu amaçla StokMiktari özelliğinin set bloğu aşağıdaki gibi değiştirilmelidir.

```
public int StokMiktari
{
   get { return stokMiktari; }
   set {
      stokMiktari = value;
      if (value < 10
         && StokAzaldi != null)
            StokAzaldi(this, new StokAzaldiEventArgs(value));
   }
}
```

Dikkat edilecek olursa ilk parametreye **this** anahtar kelimesi getirilmiştir. Hatırlayacağınız gibi ilk parametre için, olayı tetikleyen nesne referansını taşıdığını belirtmiştik. İşte buradaki **this** anahtar kelimesi çalışma zamanındaki(run-time) nesne referansının alınıp ilgili olay metoduna gönderilmesini sağlamaktadır. İkinci parametrede ise **StokAzaldiEventArgs** tipinden bir nesne örneği oluşturulmakta ve güncel stok miktarının değeri **value** anahtar kelimesi ile yapıcı metoduna gönderilmektedir. Tahmin edeceğiniz üzere nesne örneğide, olay metodu içerisinde ele alınabilecektir. Artık program içerisinde kodlarımızıda aşağıdaki gibi düzenlememiz gerekmektedir.

```
class Program
{
   static void Main(string[] args)
   {
      Urun ciklet = new Urun(10001, "Tipitipitip", 1.20, 35);
      ciklet.StokAzaldi += new StokAzaldiEventHandler(ciklet_StokAzaldi);

      for (int i = 0; i <5; i++)
      {
         ciklet.StokMiktari -= 7;
```

```
            Thread.Sleep(600);
            Console.WriteLine(ciklet.Ad + " için stok miktarı "
+ ciklet.StokMiktari.ToString());
        }
    }

    static void ciklet_StokAzaldi(object sender, StokAzaldiEventArgs args)
    {
        Console.WriteLine("Güncel stok değeri {0} . Stokta limit altına inilmiştir.
Alarrmmmm!",args.GuncelStokMiktari.ToString());
    }
}
```

Uygulamayı bu haliyle çalıştırdığımızda aşağıdaki ekran görüntüsünde yer alan sonuçları elde ederiz.

Gelelim olaylar ile ilgili bir başka konuya. Daha öncede birden fazla nesne için aynı olay metodunun ele alınabileceğinden bahsetmiştik. örneğimizi göz önüne aldığımızda, birden fazla Urun nesnesini aynı StokAzaldi olay metoduna yönlendirme şansına sahibiz. Bu şekilde bir ihtiyaç özellikle dinamik olarak oluşturulan kontrollerin bir olay metoduna bağlanarak ele alınması gibi durumlarda kullanılmaktadır. Ki böylece bir den fazla olay metodunu düşünmek yerine tek bir merkez metoddan kontrol ve yönetim işlemlerini gerçekleştirebiliriz. örneğimizdeki kod parçalarını aşağıdaki gibi değiştirelim.

```
class Program
{
    static void Main(string[] args)
    {
        Urun ciklet = new Urun(10001, "Tipitipitip", 1.20, 35);
        Urun cikolata = new Urun(10034, "Marsi", 2.5, 25);
        cikolata.StokAzaldi+=new StokAzaldiEventHandler(urun_StokAzaldi);
        ciklet.StokAzaldi += new StokAzaldiEventHandler(urun_StokAzaldi);

        for (int i = 0; i <5; i++)
        {
            ciklet.StokMiktari -= 7;
            cikolata.StokMiktari -= 5;
            Thread.Sleep(600);
```

```
        Console.WriteLine(ciklet.Ad + " için stok miktarı " +
ciklet.StokMiktari.ToString());
        Console.WriteLine(cikolata.Ad + " için stok miktarı " +
cikolata.StokMiktari.ToString());
      }
   }

   static void urun_StokAzaldi(object sender, StokAzaldiEventArgs args)
   {
      Urun urn = (Urun)sender;
      Console.WriteLine("{0} için güncel stok değeri {1} . Stokta limit altına inilmiştir.
Alarrmmmm!",urn.Ad,args.GuncelStokMiktari.ToString());
   }
}
```

İlk olarak ciklet ve cikolata isimli iki ayrı Urun nesnesi örneklediğimize ama bunların her ikisi içinde aynı olay metodunu kullandığımıza dikkat edelim.

```
cikolata.StokAzaldi+=new StokAzaldiEventHandler(urun_StokAzaldi);
ciklet.StokAzaldi += new StokAzaldiEventHandler(urun_StokAzaldi);
```

Dikkat edilmesi gereken önemli noktalardan biriside olay metodu içerisinde **sender** isimli parametre değişkeninin nasıl kullanıldığıdır. sender isimli değişken **cast** işlemine tabi tutularak bir Urun nesne örneğine dönüştürülmekte ve kullanılmaktadır. Burada elbetteki akla şu soru gelebilir. Urun nesne örneğine dönüştürme işlemi yapıldıktan sonra güncel stok miktarı gibi verilerde elde edebilir bu nedenle **StokAzaldiEventArgs** gibi tipleri geliştirmeye ihtiyacımız var mıdır? Aslında bu bir anlamda doğru olsada bir argüman tipinin var olması, olay metodu içerisine gerçektende ne aktarmak istediğimizi belirten bir kodlama yolu ve standardı sağlamaktadır. Diğer taraftan gereksiz cast işlemlerininde önüne geçilmiş olacaktır. Bunların dışında nesne örneği üzerinden elde edilemeyen ancak argümanlar yardımıyla ele alınabilecek bazı verilerin yayıncı nesne**(publisher object)** içerisinden sadece olay metoduna aktarılmasıda sağlanabilir. Bir gerekçe daha vardırki o da biraz sonra generic bir temsilci ile karşımıza çıkacaktır.

C# 2.0 ile gelen yeniliklerden biriside **isimsiz metodlardır (anonymous methods)**. Bu kavram özellikle temsilcilerin içerisinde rol aldıgı kodlama alanlarında kullanılmaktadır. Dolayısıyla olay metodlarınıda isimsiz olarak tanımlama ve geliştirme şansına sahibiz. Yani yukarıdaki kodlarımızı aşağıdaki gibi geliştirebiliriz.

```
static void Main(string[] args)
{
   Urun ciklet = new Urun(10001, "Tipitipitip", 1.20, 35);
   ciklet.StokAzaldi += delegate(object sender, StokAzaldiEventArgs arg)
                        {
                           Urun urn = (Urun)sender;
                           Console.WriteLine("{0} için güncel stok değeri {1} . Stokta limit
altındayız. Alarrmmmm!", urn.Ad, arg.GuncelStokMiktari.ToString());
                        };

   for (int i = 0; i <5; i++)
   {
      ciklet.StokMiktari -= 7;
      Thread.Sleep(600);
      Console.WriteLine(ciklet.Ad + " için stok miktarı " + ciklet.StokMiktari.ToString());
   }
}
```

Programı çalıştırdığımızda yine olay metodunun başarılı bir şekilde işletildiğini görebiliriz. Burada dikkat edilecek olursa **StokAzaldi** olayı(event) yüklenirken isimsiz metod**(anonymous method)** kullanılmıştır. **delegate** anahtar kelimesi otomatik olarak **StokAzaldiEventHandler** temsilcisine(delegate) bürünmektedir. Sonrasında gelen kod bloğu içerisinde ise olay metodunda yapılması gerekenler yer almaktadır. Ortada bir olay metodu adı olmadığına dikkat edelim. Buda zaten neden isimsiz metod denildiğini açıklamaktadır.

C# 2.0 ile birlikte gelen özelliklerden birisi ve belkide en önemliside **generic** mimaridir. Bildiğiniz gibi generic mimari sayesinde tür bağımsız tipler geliştirebilme şansına sahibiz. .Net içerisinde var olan pek çok tipin bu şekilde tür bağımsız versiyonları geliştirilmiş ve tip güvenli ile performans gibi konularda daha güçlü tipler ortaya çıkmıştır. Olaylarla ilişkili olaraktan,**EventHandler** isimli standart temsilci(delegate) tipinin generic bir versiyonu vardır. Söz konusu temsilcinin prototipi aşağıdaki gibidir.

[SerializableAttribute]
public delegate void EventHandler<**TEventArgs**> (Object sender,**TEventArgs** e) **where TEventArgs : EventArgs**

Buna göre kendi olaylarımız için ayrıca temsilci yazmaya gerek kalmamaktadır. Dikkat edilecek olursa **TEventArgs** isimli generic türün
yazılan **kısıtlama(constraint)** sayesinde **EventArgs**tipinden türemiş bir tip olması beklenmektedir. Buna göre Urun tipi içerisindeki olay tanımlamasını aşağıdaki gibi değiştirebiliriz.

public event **EventHandler<StokAzaldiEventArgs>** StokAzaldi;

EventHandler<TEventArgs> temsilcisi ilk parametre olarak **object** tipinden bir değişken almaktadır. İkinci parametre ise **EventArgs** sınıfından türemiş bir tiptir. Bizim örneğimizde söz konusu tip **StokAzaldiEventArgs** sınıfıdır. (*Sanırım kendi olay argüman tiplerimizi EventArgs sınıfından türetmenin bir faydasını daha görmüş oluyoruz.)* Sonuç olarak uygulamanın çalışması değişmeyecektir. Kazancımız ekstradan temsilciler tasarlanmasına gerek kalmayışıdır. Tabi isimsiz bir metod kullanmıyorsak olayın yükleniş şeklinide aşağıdaki gibi değiştirmemiz gerekecektir.

ciklet.StokAzaldi += new **EventHandler<StokAzaldiEventArgs>**(urun_StokAzaldi);

Olaylar ile ilgili olarak bilmemiz gereken bir diğer noktada; **+=** ile yüklenen olayların **-=** ile kaldırılabildiği ve her iki durumunda **add** ve **remove** isimli bloklar içerisinde kontrol altına alınabildiğidir. Bu konunun incelemesinide siz değerli okurlarıma bırakıyorum. Böylece geldik bir makalemizin daha sonuna. Bu makalemizde event kavramını daha detaylı bir şekilde incelemeye çalıştık ve kendi olaylarımızı nasıl yazabileceğimizi gördük. Temel olarak işlediklerimizi aşağıdaki maddeler ile özetleyebiliriz.

- Olaylar **temsilcilerin(delegates)** özelleştirilmiş bir hali olarak düşünülebilir.
- Bir olay tanımlandığında mutlaka bir temsilci tipi ile eşleştirilir. Nitekim olay meydana geldiğinde çağırılacak metodun, birisi tarafından **çalışma zamanında(runtime)** işaret ediliyor olması gerekir ki bunu temsilciler yapabilir.
- Kendi olaylarımızı geliştirirken isimlendirme standardı açısından temsilcilerimizi **EventHandler**, olay argümanlarını taşıyacak sınıflarımızı **EventArgs** kelimeleri ile bitirmekte fayda vardır.
- Olayın tanımlı olduğu nesne tarafından tetiklenmesi sonrasında yakalanabilmesi için, söz konusu nesneyi kullanan diğer nesnenin**(subscriber)** olaya abone olması gerekmektedir.
- Birden fazla nesne olayını aynı olay metoduna bağlayabiliriz.
- Olay metodlarını işaret edecek temsilciler, ilk parametre olarak olayı meydana getiren nesne referansını taşıyan object bir değişken, ikinci parametre olarakta olay metoduna bilgi taşıyacak bir sınıf örneğini alan metodları işaret edecek biçimde tasarlanırlar.

- Olay metodlarını yazmak zorunlu değildir. Bunun yerine **isimsiz metodlarda(anonymous methods)** kullanılabilir.
- İstenirse kendi olaylarımız için temsilci yazmak yerine **EventHandler<TEventArgs>** generic tipi kullanılabilir.
- Olaylara argüman taşıyacak tiplerimizi hem kod standardı hemde EventHandler<TEventArgs> desteği için **EventArgs** sınıfından türetmekte fayda vardır.
- Olayların += operatörü ile yüklenmesi ve -= operatörü ile kaldırılması durumlarını kontrol altına almak için **add** ve **remove** bloklarından faydalanılabilir.

Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

örnek Uygulama İçin Tıklayınız.

WCF - OneWay Ticket (2007-05-31T19:13:00)

wcf,

One Way Ticket...One Way Ticket... Bu sözleri duyduğumda aklıma bu şarkıyı yapan eruption ve cover versiyonunu söyleyip efsaneleşen Boney M grupları gelir. Ancak One Way ikilisi ne tesadüftürki .Net Remoting mimarisinde de karşımıza çıkmaktadır. Kısaca tek yön olarak çevirebileceğimiz bu iki kelime aslında fırlat ve unut**(fire and forget)** anlamındada düşünülebilir. Yada bir başka deyişle istemci tarafından olaya bakıldığında, "metodu çağırdım gerisi umrumda değil" de denebilir. Aynı kelimelerin **Windows Communication Foundation** içerisinde de yer alması elbetteki şaşırtıcı değildir. Nitekim One Way operasyonlar aslında asenkron istemci-sunucu modelininde önemli bir parçasıdır.

Normal şartlarda Windows Communication Foundation istemcileri servisten bir talepte bulunduklarında, proxy tarafından hazırlanan mesaj sunucuya gönderilir. Servis gelen mesajı alır, çözümler ve gereken yürütme işlemlerini gerçekleştirir. Burada söz konusu yürütmeye dahil olan metodların çalıştırılmasının sonucunda istemciye bu işlemin tamamlandığı bilgisi eğer geri dönüş değeri var ise onunla birlikte döner. Bu, talep/cevap mesajlaşma deseni **(request/response messaging pattern)** olarak adlandırılan klasik çalışma modelidir. Ancak burada istemcinin çağrıda bulunduğu metodun tamamlanışını beklemesi gerekir. Aksi takdirde ilerlemesi söz konusu değildir. Burada bahsi geçen konu kod satırında bir alt ifadeye geçilememesidir. çok doğal olarak servis tarafındaki metodun çalışmasının uzun sürdüğü durumlarda istemci uygulama beklemede olacaktır.

özellikle istemci tarafındaki uygulamaların çağrıda bulundukları metodlar geriye değer döndürmüyorlarsa asenkron programlama (**asynchronous programmin**) adına **One Way** tekniğinden yararlanılabilir. One Way tekniği uygulanması kolay olmasına rağmen dikkat edilmesi gereken noktalara sahiptir. İşte bu makalemizde söz konusu noktalara değinerek One Way tekniğinin WCF açısından detaylarını görmeye çalışacağız.

NOT : *İstemci programların uzak metod çağrımlarında uygulamaları duraksatmasını engellemek adına kullanılabilecek tek yol* ***One Way*** *değildir. Diğer Asenkron erişim modelleride ele alınabilir. Bunlar ilerleyen makalelerimizde ele alınacaktır. One Way ve diğer asenkron modellerinde istemci ve sunucu uygulamaların aynı zaman diliminde çalışıyor olmaları gerekir. Bunun aksi bir durumda ise* ***Message Queue*** *sisteminin kullanılmasında fayda vardır.*

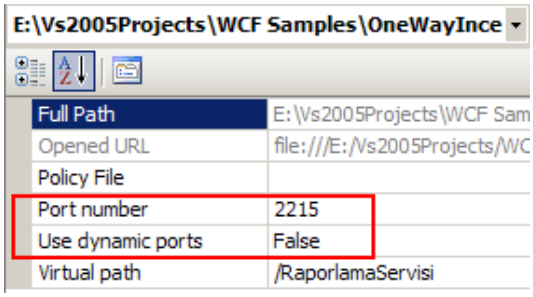
One Way formasyonuna uygun bir çağrım
için **OperationContract** niteliğinin **IsOneWay** özelliğine true değerini aktarmak yeterlidir. OneWay metodlar ile ilişkili bilinmesi gereken öncelikli kurallar vardır. Herşeyden önce OneWay olarak işaretlenenen metodlar geri dönüş değerine sahip olmazlar. Bir başka deyişle bu metodlar **void** olarak tanımlanırlar. Burada metoda parametre olarak**ref** veya **out** tipinden aktarımlar yapılmak istenebilir ancak buda mümkün değildir. Diğer taraftan söz konusu metodlar içerisinde çok doğal olarak istisnalar(Exceptions) olabilir. Burada söz konusu olan SOAP hata mesajları **(SOAP Fault Message)** istemci tarafına gönderilmez. Bu sebeplerden ötürü istemci servis üzerindeki metodun başarılı bir şekilde tamamlanıp tamamlanmadığını asla bilemez.

Ancak en azından istemcinin gönderdiği metod çağrısı mesajının servise ulaşıp ulaşmadığının bilinmesinde yarar vardır. Eğer servis uygulaması çalışmıyorsa çok doğal olarak istemci bir istisna alacaktır ki bu durumda istemciden gönderilen talebin ulaşmadığı zaten doğrudan anlaşılabilir. Ancak bazı iletişim protokollerinde (örneğin **Tcp**) gelen mesajlar tampona alınırarak işlenirler. Arka arkaya gelen bu taleplere ait mesajların toplu olarak bir maksimum sayısı vardır. Sonuçta birden fazla istemci söz konusudur ve bunların gönderecekleri sayısız çağrı bulunmaktadır. Böyle bir durumda eğer servis tarafında kabul edilebilen maksimum talep sayısı aşılırsa gelen yeni talepler var olan yarım kalmışlar tamamlanıncaya kadar beklemeye alınır. Dolayısıyla istemci uygulama her ne kadar OneWay çağrısı yapsada beklemede kalır. Bu durumla başedebilmek için güvenilir oturumlar **(Reliable Session)** açılmasında fayda vardır. Güvenilir oturumlar sayesinde servis, bir çağrı aldığında gelen mesajı işlemeye başladığına dair istemciye bilgi gönderebilir.

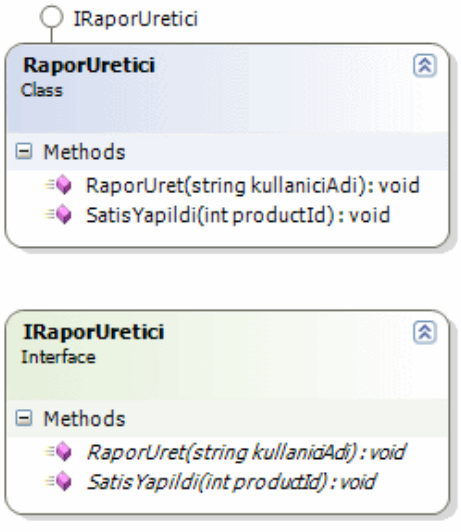
One Way tekniği istemci için önem arz eden **SendTimeout** sürelerini önemsemez. Normal şartlarda istemciler bir metod çağrısında bulunduklarında varsayılan olarak 1 dakikalık bir timeout süreleri vardır. Eğer bu süre içerisinde servisten bir cevap gelmesse otomatik olarak **TimeoutException** tipinden bir çalışma zamanı hatası alınır. One way operasyonlarında çok doğal olarak bu tip bir süre kontrolü önemli değildir. Nitekim istemci tarafı çağrıda bulunduğu metod için bir cevap beklememektedir. Ama varsayılan olarak bazı aksilikler olacaktır ki ilerleyen kısımlarda buna değinmeye çalışacağız.

Şimdi örnek bir uygulama üzerinden ilerleyerek One Way tekniğini nasıl ele alabileceğimizi inclemeye çalışacak ve dikkat edilmesi gereken durumları analiz edeceğiz. Bu makalemizdeki örneğimizde, file-based olarak web üzerinde barındırılan bir servis

uygulaması yer alacaktır. İstemciyi olayları kolay bir şekilde takip edebilmek adına bir konsol uygulaması olarak tasarlayabiliriz. Burada diğer makalelerimizden farklı olarak **wsHttpBinding** bağlayıcı tipini ele alacağız. Bu tip aslında **BasicHttpBinding** bağlayıcı tipine benzerdir ancak güvenilir oturumlara **(reliable sessions)** sahip, **transcation** yönetimine izin veren **Http** ve **Https** iletişim protokollerinin kullanılabildiği, **WS-Adressing** işlemlerin yapılabildiği bir ortam sunarak ekstra imkanlar sağlar. Geliştirilecek servis uygulaması için **WCF Service** şablonu kullanılabilir. Bu amaçla ilk olarak Visual Studio 2005 ile aşağıdaki gibi bir servis uygulaması açarak işe başlanmalıdır. WCF Service uygulaması f**ile-based** olarak host edileceğinden ve istemcilerin tek bir adresten ilgili servise erişmeleri istendiğinden projenin Properties penceresindeki özelliklerden port numarası sabit bir değere ayarlanabilir. Bu işlem için öncelikle **Dynamic Ports** özelliğine **false** değer atamak gerekir.



Servisimiz için gerekli tiplerimizi ise aşağıdaki gibi geliştirebiliriz. Söz konusu tipler **App_Code** klasörü içerisinde yer almaktadır.



IRaporUretici Arayüzü İçeriği;

```
[ServiceContract(Name = "RaporlamaServisi", Namespace =
"http://www.bsenyurt.com/RaporlamaServisi/2007/30/05")]
public interface IRaporUretici
```

```
{
   [OperationContract(IsOneWay = true)]
   void RaporUret(string kullaniciAdi);
   [OperationContract(IsOneWay = true)]
   void SatisYapildi(int productId);
}
```

RaporUretici sınıfı içeriği;

```
public class RaporUretici
            : IRaporUretici
{
   #region IRaporUretici Members

   public void RaporUret(string kullaniciAdi)
   {
      Thread.Sleep(65000); // 1 dakika 5 saniye duraksatma
      FileStream fs = new FileStream("C:\\Izleme.txt", FileMode.Append,
FileAccess.Write);
      StreamWriter writer = new StreamWriter(fs);
      writer.WriteLine(kullaniciAdi + " İçİN RAPOR üRETME EMRİ VERİLDİ");
      writer.Close();
      fs.Close();
   }

   public void SatisYapildi(int productId)
   {
      Thread.Sleep(65000); // 1 dakika 5 saniye duraksatma
      FileStream fs = new FileStream("C:\\Izleme.txt", FileMode.Append,
FileAccess.Write);
      StreamWriter writer = new StreamWriter(fs);
      writer.WriteLine(productId + " üRüNü İçİN SATIŞ YAPILDI");
      writer.Close();
      fs.Close();
   }

   #endregion
}
```

Arayüz tanımlaması içerisinde dikkat edilecek
olursa **OperationContract** niteliğinin **IsOneWay** özelliğine **true** değeri atanmıştır. Servis sözleşmesini uygulayan sınıf içerisinde sembolik olarak iki metod yer almaktadır. özellikle bu metodların içerisinde **Thread** sınıfının static **Sleep** metodu yardımıyla uygulama 65 saniye duraksatılmaktadır. Burada 1 dakikalık TimeOut sınırının aşılmasının doğuracağı

bazı sonuçlar olacaktır. Bu sonuçlar ilerleyen kısımlarda ele alınacaktır. Servis tarafında yer alan web.config dosyasının içeriği aşağıdaki gibi tasarlanabilir.

web.config içeriği;

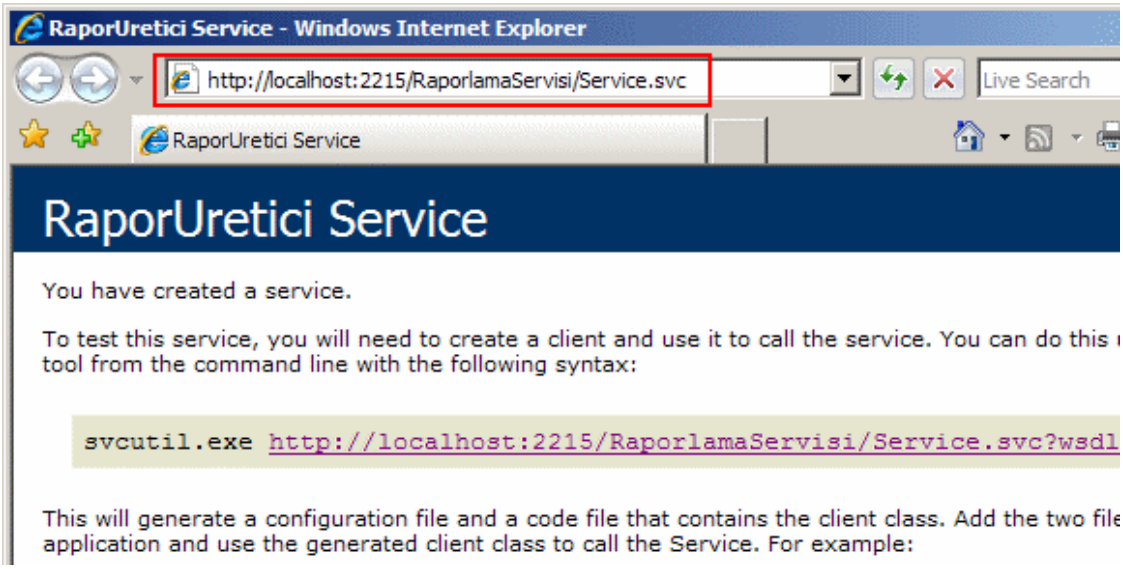
```
<?xml version="1.0"?>
<configuration>
   <system.serviceModel>
      <services>
         <service behaviorConfiguration="RaporlamaServisiBehavior"
name="RaporlamaServisi.RaporUretici">
            <endpoint address="http://localhost:2215/RaporlamaServisi/Service.svc" bin
ding="wsHttpBinding" bindingConfiguration="" name="RaporlamaServisiEndPoint"
contract="RaporlamaServisi.IRaporUretici" />
         </service>
      </services>
      <behaviors>
         <serviceBehaviors>
            <behavior name="RaporlamaServisiBehavior">
               <serviceMetadata httpGetEnabled="true"/>
            </behavior>
         </serviceBehaviors>
      </behaviors>
   </system.serviceModel>
   <system.web>
      <compilation debug="true"/>
   </system.web>
</configuration>
```

Servis web tabanlı olarak host edildiğinden **svc** uzantılı bir dosyanında var olması gerekmektedir. Bu dosyanın içeriğinde servis sözleşmesini uygulayan sınıfın adı ve fiziki dosya adı aşağıdaki gibi yer alır.

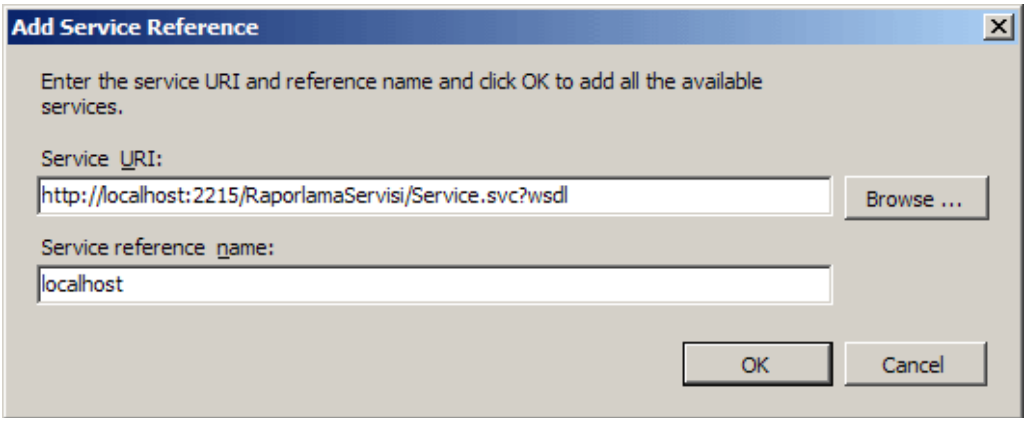
Service.svc;

```
<% @ServiceHost Language=C#
Debug="true" Service="RaporlamaServisi.RaporUretici" CodeBehind="~/App_Code/
Service.cs" %>
```

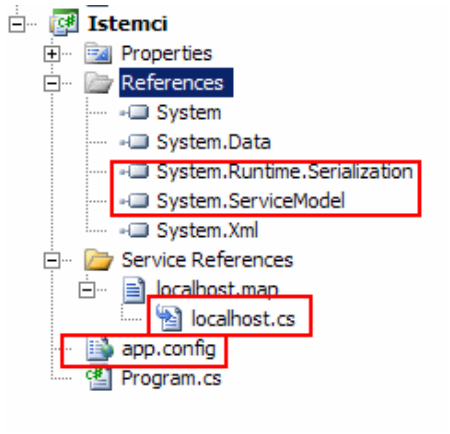
Servis ile ilgili işlemlere geçmeden önce çalıştığından emin olmakta fayda vardır. Servis başarılı bir şekilde hazırlanmışsa aşağıdaki ekran görüntüsünün elde edilmesi gerekir.



Artık istemci için gerekli kodlar yazılıp testlere başlanabilir. öncelikli olarak istemci uygulamalar için gerekli proxy tipinin üretilmesi gerekir. Bunun için komut satırında (Visual Studio 2005 Command Prompt) **svcutil.exe** aracı kullanılabilir yada Visual Studio 2005 ortamında konsol uygulamasında **Add Service Reference** seçeneği ile aşağıdaki ekran görüntüsünde olduğu gibi eklenebilir.



Bu işlemin ardından istemci uygulamasına ait proje içerisinde gerekli proxy tipi ve **App.config** konfigurasyon dosyası üretilmiş olacaktır.



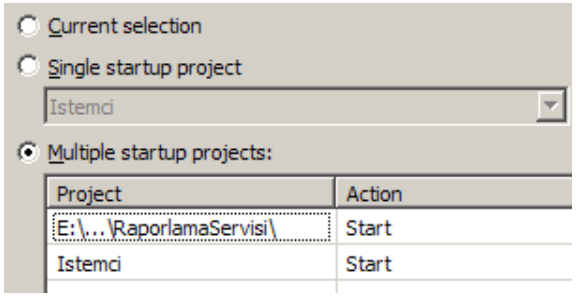
İstemci uygulamaya ait kodlar aşağıdaki gibi geliştirilebilir.

```
using System;
using System.Text;
using System.ServiceModel;
using System.Collections.Generic;
using Istemci.localhost;

namespace Istemci
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            RaporlamaServisiClient srv = new
RaporlamaServisiClient("RaporlamaServisiEndPoint");
            Console.WriteLine("Rapor üretimini başlat..." + DateTime.Now.ToString());
            srv.RaporUret("Mayk");
            Console.WriteLine("Rapor üretimi başlatıldı..." + DateTime.Now.ToString());
            Console.WriteLine("ürün satışı yapıldı bilgisini gönder..." +
DateTime.Now.ToString());
            srv.SatisYapildi(1001);
            Console.WriteLine("ürün satışı yapıldı bilgisi gönderildi..." +
DateTime.Now.ToString());

            srv.Close();
         }
         catch (Exception hata)
         {
            Console.WriteLine(hata.Message);
         }
```

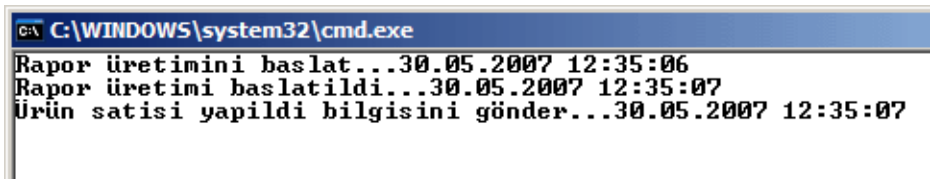
```
        }
    }
}
```

RaporlamaServisiClient nesnesi örneklendikten sonra RaporUret ve SatisYapildi isimli metodlar çağırılmaktadır. Bu metodları işaret eden servis sözleşmesinde **OneWay** oldukları belirtilmiştir. Buna göre istemci uygulamanın, bu metodlar içerisindeki 65 saniyelik sürelerden etkilenmeden diğer kod satırlarını işletmesi gerekmektedir. Bakalım gerçekten böyle mi olacak?

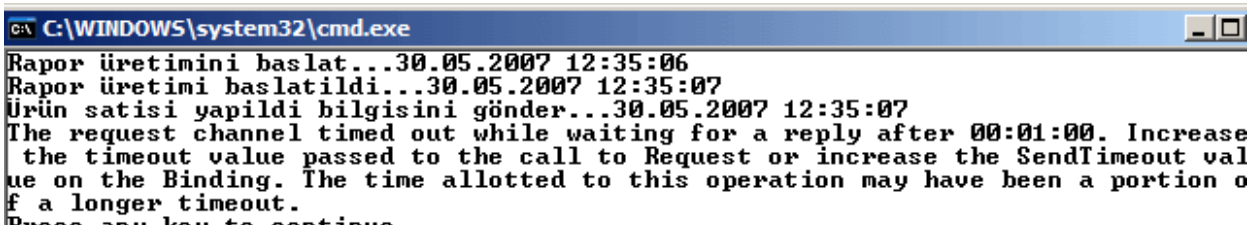
Testleri yapabilmek için hem sunucu uygulamanın hemde istemci uygulamanın aynı zaman diliminde çalışıyor olmaları gerekmektedir. Dolayısıyla **Solution** özelliklerinde **Multiple Startup Projects** seçilmeli ve önce servis uygulaması ardından istemci uygulama çalışacak şekilde ayarlanmalıdır.



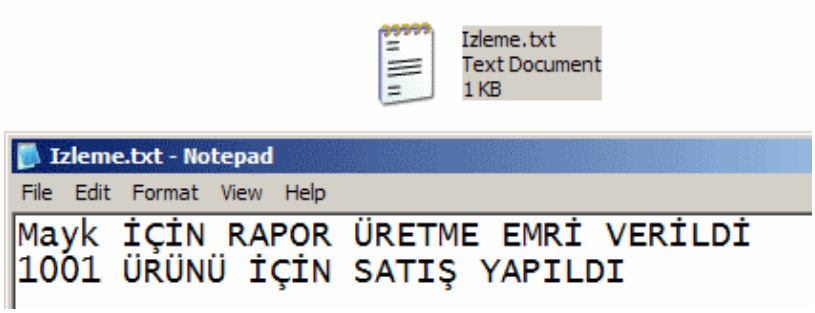
Bu aşamadan sonra uygulama çalıştırılırsa ilk etapta aşağıdaki gibi bir ekran görüntüsü ile karşılaşılır.



Görüldüğü gibi RaporUret çağrısı kolayca aşılmış ancak, **SatisYapildi** çağrısından sonra istemci uygulama ekranı duraksamıştır. Oysaki ikinci metodda **OneWay**olarak çağrılmaktadır. Dolayısıyla aynen ilk metod çağrısında olduğu gibi buradada istemci uygulamanın diğer kod satırlarından işlemlerine devam etmesi beklenir. Ne varki böyle olmamıştır. Sonuç olarak bir süre bekledikten sonra aşağıdaki ekranda görülen hata mesajı ile karşılaşılır.



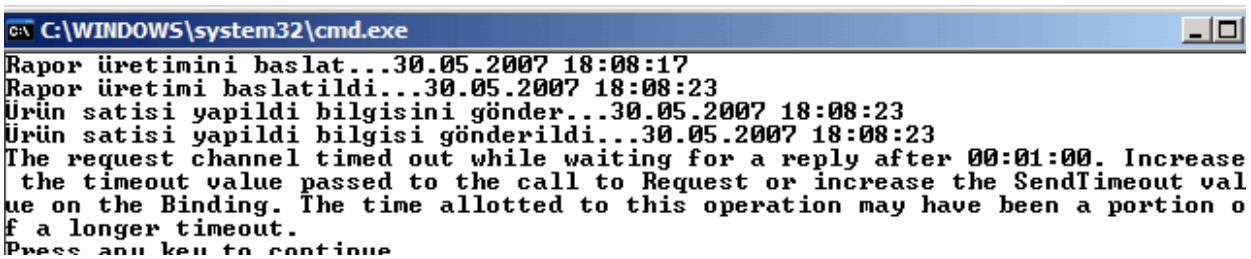
Peki sorun nedir? İstemci uygulama neden çalışma zamamında bir istisna alarak sonlanmıştır? Hatta yeterli süre beklenildiğinde sunucu tarafında **Izleme.txt** dosyasının içeriğine bakıldığında metod içeriklerinin çalıştığıda gözlemlenebilir.



Ne varki istemci açısından bazı sorunlar olduğu ortadadır. Problemin nedeni oturumlardır **(Sessions)**. Aynı oturum içerisinde gelen arka arkaya iki çağrı söz konusudur. Bu çağrılar doğal olarak aynı andalık ilkesine **(Concurrency)** göre değerlendirilir. Dolayısıyla ikinci çağrıdan sonra istemci uygulama, ilk çağrı sonucunun tamamlanmasını beklemek durumunda kalmıştır. Bu sebepten servisin özellikle aynı oturum içerisinden gelecek eş zamanlı metod çağrılarına cevap verebilecek şekilde özelleştirilmesi gerekir. Burada elebetteki Session moddan vazgeçilmesi tercih edilebilir. örneğin **PerCall** mode seçilebilir. Ancak oturumların **(Sessions)** kullanım amaçları göz önüne alındığında bir çözüm yolu bulunmasında fayda vardır. Yapılması gereken servis sözleşmesini uygulayan RaporUretici sınıfına, **ServiceBehavior** niteliğini uygulamak
ve **ConcurrencyMode** özelliğine **Multiple** değerini aşağıdaki gibi vermektir.

```
namespace RaporlamaServisi
{
   [ServiceBehavior(ConcurrencyMode= ConcurrencyMode.Multiple)]
   public class RaporUretici
                  : IRaporUretici
   {
```

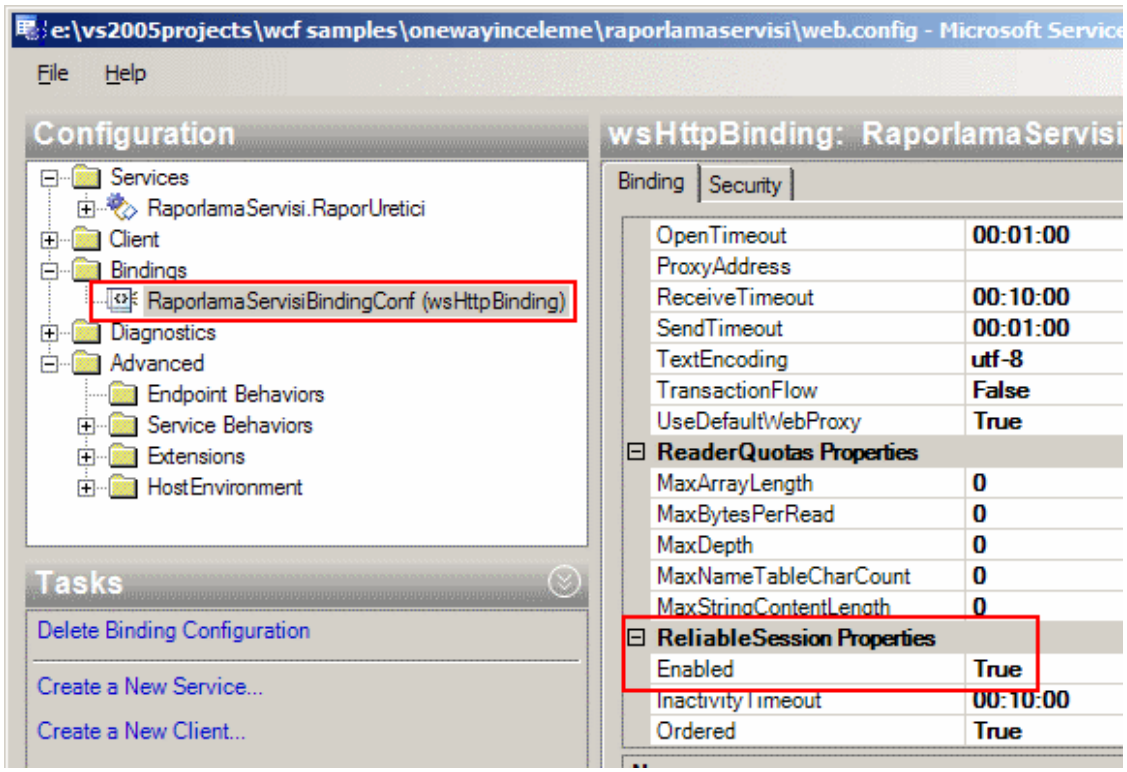
Uygulamamızı bu haliyle test ettiğimizde ise aşağıdaki ekran görüntüsü ile karşılaşırız.



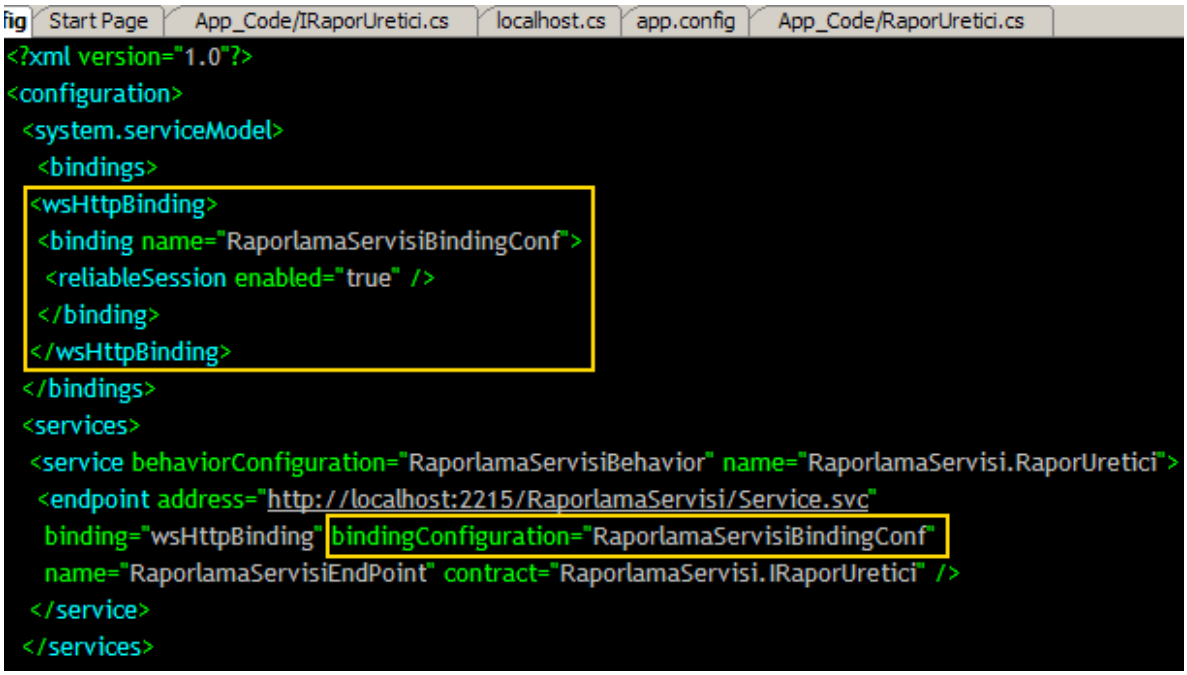
Herşey ilk etapta yolunda gibidir. Nitekim SatisYapildi metod çağrısı sonrasındaki kodlarda çalışmıştır. Ancak uygulama kapanırken yine bir hata mesajı alınmıştır. Bu hata mesajının üretildiği nokta istemci tarafında servis için **Close** metodunun çağrıldığı yerdir. Bu çağrıya gelindiğinde istemci uygulama uzun bir süre beklemede kalmıştır. Sonrasında

ise bir istisna mesajı fırlatarak sonlanmıştır. Bunun sebebi kullanılan WsHttpBinding tipinin özellikleridir. **WsHttpBinding** oturumları kullanan ve varsayılan olarak mesaj seviyesinde güvenlik **(Message-Level Security)** kullanan bir bağlayıcı **(Binding)** tipidir. İstemci Close metodunu çağırdığında servis tarafında kendisi için açılan oturumu(Session) kapatmak ister. Buna karşılık servis ilgili oturuma ait işlemlerin bitmesini bekler ve tamamlandıklarından emin olduktan sonra bunu istemciye bir mesaj ile bildirir. örnekte çok doğal olarak servis tarafındaki işlemler 1 dakika sınırını aştığından, söz konusu mesaj istemciye iletilemez. Buda doğal olarak istemci tarafında bir istisna mesajı fırlamasına neden olur. çözüm olarak güvenlik ile ilgili ayarlar kapatılabilir ki bu önerilen bir yöntem değildir. Diğer bir yöntem ise iletişim seviyesinde güvenlik **(transport levet security)** kullanmaktır. Ancak bu yöntemde sertifika kullanılması gerekir nitekim Https politikası söz konusudur.

Ancak mesaj seviyesinde yapılabilecek bir seçenek daha vardır. Buda güvenilir bir oturum **(Reliable Session)** açmaktır. Nitekim bu sayede istemcinin yaptığı çağrılar sonucu servisten bir bilgi beklenmesine gerek kalmayacaktır. Bunun için servis tarafında ve istemci tarafında **wsHttpBinding** tipi için konfigurasyon ayarları yapılmalı ve **ReliableSession** için **Enabled** özelliklerine **true**değeri atanmalıdır. Aşağıdaki ekran görüntüsünde servis tarafı için ilgili değişikliklerin **Microsoft Service Configuration Editor** yardımıyla nasıl yapıldığı gösterilmektedir.



Bu değişiklikler sonucu servis tarafı için konfigurasyon dosyasının içeriği aşağıdaki gibi olur.



Benzer değişikliklerin istemci tarafı içinde yapılması gerekir. Bunun sonucu olarak istemci tarafındaki konfigurasyon dosyasındada aşağıdaki değişiklikler olacaktır.

```
fig | Start Page | App_Code/IRaporUretici.cs | localhost.cs | app.config | App_Code/RaporUretici.cs
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <wsHttpBinding>
        <binding name="RaporlamaServisBindingConfiguration" closeTimeout="00:01:00"
            openTimeout="00:01:00" receiveTimeout="00:10:00" sendTimeout="00:01:00"
            bypassProxyOnLocal="false" transactionFlow="false" hostNameComparisonMode=
"StrongWildcard"
            maxBufferPoolSize="524288" maxReceivedMessageSize="65536"
            messageEncoding="Text" textEncoding="utf-8" useDefaultWebProxy="true"
            allowCookies="false">
            <readerQuotas maxDepth="32" maxStringContentLength="8192" maxArrayLength="16384"
              maxBytesPerRead="4096" maxNameTableCharCount="16384" />
            <reliableSession ordered="true" inactivityTimeout="00:10:00"
              enabled="true" />
            <security mode="Message">
              <transport clientCredentialType="Windows" proxyCredentialType="None"
                realm="" />
              <message clientCredentialType="Windows" negotiateServiceCredential="true"
                algorithmSuite="Default" establishSecurityContext="true" />
            </security>
          </binding>
        </wsHttpBinding>
      </bindings>
      <client>
        <endpoint address="http://localhost:2215/RaporlamaServisi/Service.svc"
          binding="wsHttpBinding" bindingConfiguration="RaporlamaServisBindingConfiguration"
          contract="Istemci.localhost.RaporlamaServisi" name="RaporlamaServisiEndPoint">
```

Artık uygulamımızı bu haliyle test edebiliriz. Aşağıda uygulamanın son halinin test sonuçlarına ait ekran görüntüsü yer almaktadır.

```
C:\WINDOWS\system32\cmd.exe
Rapor üretimini baslat...30.05.2007 18:14:21
Rapor üretimi baslatildi...30.05.2007 18:14:21
Ürün satisi yapildi bilgisini gönder...30.05.2007 18:14:21
Ürün satisi yapildi bilgisi gönderildi...30.05.2007 18:14:21
Press any key to continue . . .
```

Sonuç itibariyle basit bir **One Way Ticket** şarkısından oldukça farklı noktalara geldik. Bu ana kadar gördüklerimizden aşağıdaki sonuçları çıkartabiliriz.

- OneWay tekniği ile Windows Communication Foundation uygulamalarında asenkron çağırımlar gerçekleştirilebilir ve **performans** artışı sağlanır. çünkü istemci uygulama, servis tarafındaki metod sonuçlarını beklemeden yoluna devam eder.

- **Günvenilir oturumlara (Reliable Sessions)** izin verilmesi halinde istemci tarafında **Close** metodu uygulandığında servis ile olan **bağlantı(Connection)**, servisten cevap beklenmeden sonlandırılabilir.
- özellikle **oturum(Session)** kullanıldığı durumlarda servis üzerinde aynı oturuma ait arka arkaya metod çağırımlarında, istemci uygulamanın duraksamasını engellemek adına**ConcurrencyMode** özelliğinin değeri **Multiple** olarak set edilebilir.
- ConcurrencyMode özelliğinin Multiple olarak belirlenmesi halinde servisin gerçektende **thread-safe** olup olmadığına dikkat edilmelidir. Nitekim Multiple değeri ile, servis tarafındaki uygulama**Multi-Thread** hale gelmektedir.

Bu makalemizde asenkron programlama çeşitlerinden birisi olan One Way metod çağırım tekniğini incelemeye çalıştık. Asenkron programlamaya ilişkin başka örnekleri ilerleyen makalelerimizde incelemeye çalışacağız. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

örnek Uygulama İçin Tıklayınız.

WCF - InstanceContextMode (2007-05-24T19:17:00)

wcf,

Windows Communication Foundation uygulamalarında istemciler başvurdukları servisler üzerindeki nesne örneklerini kullanırlar. özellikle kullanılan bağlayıcının (binding) tipine göre servis üzerindeki nesne örneklerinin farklı şekillerde oluşturulup ele alınması söz konusudur. **.Net Remoting** ile uygulama yazan geliştiriciler, istemcilerin talepte bulunacağı uzak nesne örneklerinin farklı modellerde örneklendiklerini bilirler. Burada bahsi geçen modeller **Server Activated Object** için **Singleton** ve **SingleCall** ile **Client Activated Object**' dir. örneğin CAO modeline göre istemciler, uzak nesneyi örneklediklerinde sunucu üzerinde bir referans oluşturulur ve istemciler bunu kullanır. Singleton ve SingleCall modelleri metod çağrıları sonucu referans oluşturulmasını sağlar. Ama Singleton modelinde her istemci için(dolayısıyla her metod çağrısı için) aynı nesne örneği, SingleCall' da ise her metod çağrısı için ayrı bir nesne örneği sunucu üzerinde oluşturulmaktadır.

Benzer durumlar WCF servislerindeki nesne örnekleri içinde geçerlidir. İstemci uygulama ile servis örneği arasındaki ilişkiyi kontrol altına alabilmek
için **ServiceBehavior** niteliğinin**InstanceContextMode** özelliğinden yararlanılır. InstanceContextMode özelliği InstanceContextMode enum sabiti tipinden bir değer almaktadır ve alabileceği değerler **PerSession**, **PerCall** ve**Single**' dır. Bu modlardan hangisinin aktif olduğu özellikle WCF uygulamalarında **oturum yönetimi (Session Management)** açısından da önemlidir.

Varsayılan olarak her istemci için sunucuda bir adet nesne örneği oluşturulur. Servis örneği istemciden gelen ilk operasyon çağrısında (yani ilk metod çağrısında) oluşur. İstemci

sunucu ile olan bağlantısını kesene kadarda servis örneği sunucuda kalmaya devam eder. Ancak binlerce kullanıcının servise bağlandığı durumlarda da her istemci için bir servis örneğinin sunucuda oluşturulması ve ne zaman kalkacaklarının istemcinin hareketine bağlı olması sunucu belleğinin gereksiz yere şişirilmesi anlamına gelir. Dolayısıla diğer modeller göz önüne alınabilir.

NOT : *Binding Tipine göre Instance Mode türlerinden hangisinin uygulanabileceği değişir. örneğin* ***BasicHttpBinding****, Http protokolünün doğası gereği iletişim seviyesinde (transport level) PerSession mode türünü desteklemez. Bir başka deyişle* ***InstanceContectMode*** *hangi değeri alırsa alsın varsayılan olarak* ***PerCall*** *modda çalışır. Buda gelecek her istemci operasyon çağrısı için servis tarafında bir nesne örneğinin oluşturulması ve operasyon tamamlandığında kaldırılmak üzere Garbage Collector' e devredilmesi anlamına gelir.*

İlk olarak PerSession, PerCall ve Single modellerinin nasıl çalıştıklarını teorik olarak bilmekte fayda vardır.

PerSession Modeli;

PerSession modunda istemciler ilk operasyon çağrısında bulunduklarında servis örneği oluşturulur ve istemci **Close** metodunu kullanana kadar yada uygulamayı kapatana kadar söz konusu örnek sunucuda kalır. İstemci, bir servis örneğini elde ettikten sonra bu örnek sadece ilgili istemciye ait olacak şekilde tahsis edilir. Bir anlamda o istemci için bir **oturum(Session)** açılmış olur. Farklı istemcilerin aynı oturumu kullanmaları mümkün değildir. İstemci uygulamalarda **çok kanallı (multithread)** kod parçaları olabileceği düşünüldüğünde eşzamanlı olarak aynı oturuma ait metod çağrıları söz konusu olabilir. Varsayılan olarak bir talep tamamlanmadan başka bir talep gelirse öncekinin tamamlanması beklenmektedir. Ama istenirse bu davranış biçimide**ServiceBehavior** niteliğinin **ConcurrencyMode** özelliği ile değiştirilebilir. Yani istenirse eş zamanlı olarak çağrılara cevap verilmesi de sağlanabilir.

Bu mod kullanılırken, istemci tarafından çalıştırılabilecek operasyonlar için bir sıralama belirtilmek istenebilir. örneğin hangi metod çağrısı ile servis örneğinin oluşturulacağının belirtilmesi veya hangi metod çağrısından sonra servis örneğinin yok edileceğinin belirtilmesi gibi. Bunun için de **OperationContract** niteliğinin **IsInitiating** ve **IsTerminating** özellikleri kullanılır. IsInitiating özelliğine true değeri atandığında, ilgili metoda bir çağrı geldiği zaman servis nesnesi örneklenecektir. Elbetteki aynı metoda oturum süresince yeni bir çağrı gelebilir. Bu durumda yeni bir servis örneği oluşturulmaz. Eğer false değeri verilirse söz konusu metoda yapılan çağrı sonrasında servise ait bir örnek oluşturulmaz. **IsTerminating** özelliğine true değeri atandığı takdirde, ilgili metoda dair söz konusu operasyon tamamlandığında servis örneği otomatik olarak yok edilir. Yani istemcinin Close metodunu çağırmasına gerek kalmaz.

PerCall Modeli;

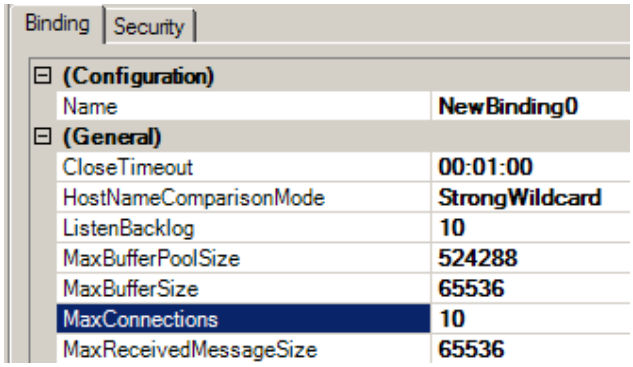
PerCall modunda, istemcinin yaptığı her operasyon çağrısında servis uygulamasının çalıştığı sistem üzerinde bir nesne örneği oluşturulur ve operasyon tamamlandığında(bir başka deyişle metod çağrısı sonlandığında) bu örnek yok edilir. Bu sebepten oturum yönetimi **PerSession** moduna göre daha zordur. Bu nedenle oturum yönetimi adına istemcinin kendisini her operasyon çağrısında servise tanıtabilmesini sağlayacak bir sistem gerekebilir. Ne varki sunucu kaynaklarının idareli kullanılması adına verimli bir modeldir.

Single Modeli;

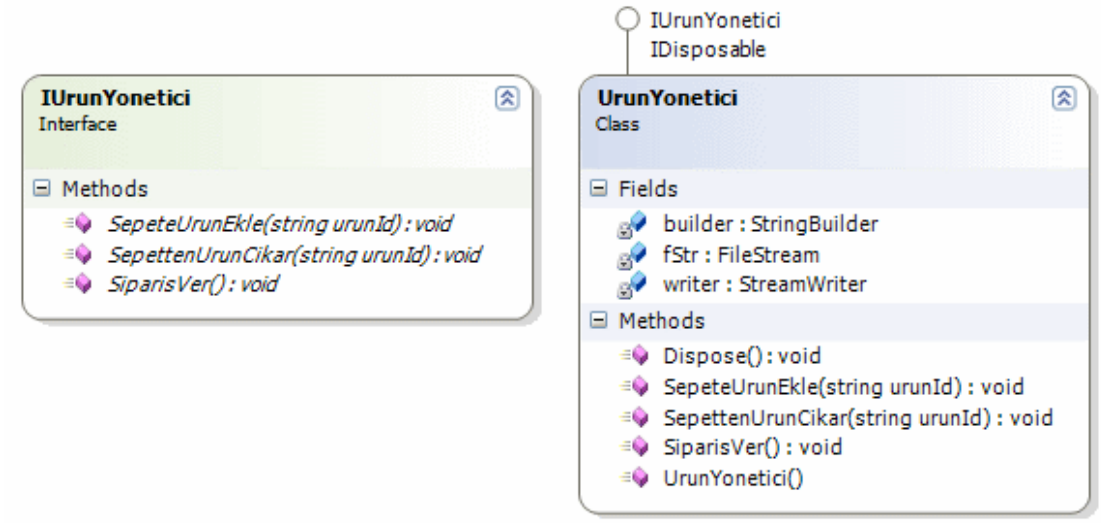
Single modunda, servis örneği yine istemci tarafından gelecek ilk operasyon çağrısında oluşturulur. Ne var ki **PerSession** modelinden farklı olarak var olan istemcinin ve diğer istemci uygulamaların metod çağrıları için servis uygulaması üzerindeki aynı nesne örneği kullanılır. Söz konusu servis örneğinin yok edilmesi ise sadece host uygulamanın kapatılması ile gerçekleşebilir. Bu teknik kaynak yönetimi adına maksimum faydayı sağlar. Ayrıca tüm kullanıcıların aynı veriyi paylaşması çok daha kolaylaşır. Lakin burada servisin **single thread** olup olmadığına dikkat etmek gerekir. Eğer öyleyse istemcilerden gelecek talepler sonrası **zaman aşımları (timeout)** söz konusu olabilir. Bunun için **ConcurrencyMode** özelliğine**Multiple** değeri atanarak **thread-safe** bir ortam sağlanabilir.

Şimdi örnek bir uygulama üzerinden bu modelleri incelemeye çalışalım. örnek uygulamada Tcp protokolü baz alınmaktadır ve bu nedenle **NetTcpBinding** bağlayıcı tipi kullanılmaktadır.

NOT : *Tcp protokolü için varsayılan olarak eş zamanlı bağlantı sayısı* ***maksimum 10*** *dur. Ancak istenirse binding configuration kısmından* ***MaxConfiguration*** *özelliği ile bu değiştirilebilir.*



İlk olarak WCF Library şablonunda bir sınıf kütüphanesi geliştirelim. Bu sınıf kütüphanesi içerisinde yer alan tipler ve içerikleri aşağıdaki gibidir.



Servis sözleşmesini tanımlayan IUrunYonetici interface(arayüz) kodları aşağıdaki gibidir;

```
using System;
using System.ServiceModel;

namespace UrunYonetimKutuphanesi
{
   [ServiceContract(Name="UrunYonetimServisi",
Namespace="http://www.bsenyurt.com/UrunYonetimServisi")]
   public interface IUrunYonetici
   {
      [OperationContract()]
      void SepeteUrunEkle(string urunId);
      [OperationContract()]
      void SepettenUrunCikar(string urunId);
      [OperationContract()]
      void SiparisVer();
   }
}
```

Arayüzü uygulayan UrunYonetici sınıfının kodları aşağıdaki gibidir;

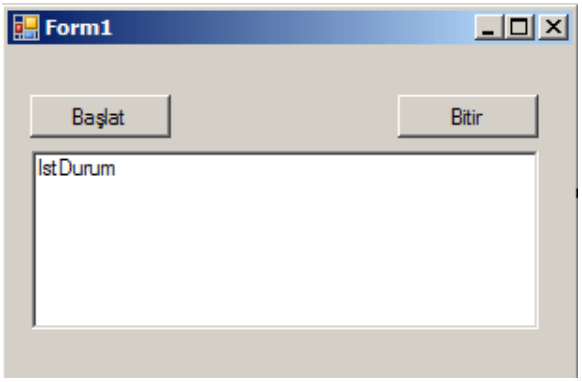
```
using System;
using System.IO;
using System.Text;
using System.Threading;
using System.ServiceModel;

namespace UrunYonetimKutuphanesi
{
```

```
[ServiceBehavior()]
public class UrunYonetici:IUrunYonetici,IDisposable
{
    private FileStream fStr;
    private StreamWriter writer;
    private StringBuilder builder;

    public UrunYonetici()
    {
        Thread.Sleep(500);
        builder = new StringBuilder();
        builder.AppendLine(DateTime.Now.ToLongTimeString() + ": UrunYonetici
nesnesi oluşturuldu.");
    }
    #region IUrunYonetici Members

    public void SepeteUrunEkle(string urunId)
    {
        Thread.Sleep(500);
        builder.AppendLine(DateTime.Now.ToLongTimeString() + ": Urun ekle metodu
çağırıldı.");
    }

    public void SepettenUrunCikar(string urunId)
    {
        Thread.Sleep(500);
        builder.AppendLine(DateTime.Now.ToLongTimeString() + ": Urun çıkart metodu
çağırıldı.");
    }

    public void SiparisVer()
    {
        Thread.Sleep(500);
        builder.AppendLine(DateTime.Now.ToLongTimeString() + ": Sipariş ver metodu
çağırıldı.");
    }

    #endregion

    #region IDisposable Members

    public void Dispose()
    {
        Thread.Sleep(500);
        builder.AppendLine(DateTime.Now.ToLongTimeString() + ": UrunYonetici
```

```
nesnesi için Dispose metodu çalıştırıldı.");
            builder.AppendLine("");
            fStr = new FileStream("Izleyici.txt", FileMode.Append, FileAccess.Write);
            writer = new StreamWriter(fStr);
            writer.Write(builder.ToString());
            writer.Close();
            fStr.Close();
        }

        #endregion
    }
}
```

UrunYonetici sınıfı içerisinde bir alışveriş sepetine ürün ekleme, ürün çıkarma veya sipariş verme gibi örnek fonksiyonellikler vardır. özellikle nesneye ait örneklerin ne zaman oluşturulduğunu incelemek adına **FileStream, StreamWriter ve StringBuilder** tiplerinden yararlanılmakta ve **Izleyici.txt** isimli dosyaya log bilgileri aktarılmaktadır. Burada dikkat edilmesi gereken noktalardan biriside sınıfa **IDisposable** arayüzünün uygulanmış olmasıdır. Bunun tek sebebi **Garbaga Collector**' un nesneyi dispose etmeden önce log dosyasına gereken bilgileri aktarmaktır. ServiceBehavior niteliğinde özel olarak **InstanceContextMode** değeri verilmemiştir. Nitekim varsayılan değeri **PerSession**' dır. Şimdi örnek sunucu ve istemci uygulamalarını geliştirerek devam edelim. Servis uygulaması bir windows uygulamasıdır ve aşağıdaki kodlardan oluşmaktadır. *(Servis uygulamasının **System.ServiceModel** ve UrunYonetimKutuphanesi assembly' larını referans etmesi gerektiğini hatırlayalım.)*

Servis tarafındaki windows formu;



Servis tarafındaki kodlar;

```
namespace ServerApp
{
    public partial class Form1 : Form
    {
```

```
        public Form1()
        {
            InitializeComponent();
        }

        ServiceHost host;

        private void btnBaslat_Click(object sender, EventArgs e)
        {
            host = new ServiceHost(typeof(UrunYonetici));
            host.Open();
            lstDurum.Items.Add(DateTime.Now.ToLongTimeString() + host.State);
        }

        private void btnBitir_Click(object sender, EventArgs e)
        {
            host.Close();
            lstDurum.Items.Add(DateTime.Now.ToLongTimeString() + host.State);
        }
    }
}
```

Host uygulamada servisi çalıştırmak için **Open**, kapatmak içinse **Close** metodları kullanılmaktadır. Bununla birlikte servis uygulaması UrunYonetici tipini hizmete sunar. Bağlantı için gereken tüm konfigurasyon ayarları aşağıda çıktısı görülen App.config dosyasından elde edilmektedir.

Servis tarafı için **App.config** içeriği;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.serviceModel>
        <behaviors />
        <services>
            <service behaviorConfiguration=""
name="UrunYonetimKutuphanesi.UrunYonetici">
                <endpoint address="net.tcp://localhost:4560/UrunYonetim.svc"
binding="netTcpBinding" bindingConfiguration=""
name="UrunYonetimServiceEndPoint"
contract="UrunYonetimKutuphanesi.IUrunYonetici" />
            </service>
        </services>
    </system.serviceModel>
</configuration>
```

Baştada belirtildiği gibi Http yerine Tcp iletişim protokolünü kullanan bir sistem ele alınmaktadır. Bu nedenle NetTcpBinding bağlayıcı tipi kullanılmaktadır.

İstemci tarafı basit bir Console uygulaması olarak düşünülebilir. İstemci tarafında, uzak nesnenin kullanılabilmesi için fiziki proxy sınıfınında üretilmiş olması gerekir. Bu amaçla yine **svcutil.exe** aracı kullanılmaktadır. Servis Http üzerinden **metadata publishing** yapmadığından, önce kütüphane üzerinden gerekli **wsdl** ve **schema** dosyaları elde edilmeli ve sonrasında bu bilgilerden faydalanarak gereken **proxy** sınıfı oluşturulmalıdır.

önce komut satırından UrunYonetimKutuphanesi.dll bulunur ve

\> svcutil UrunYonetimKutuphanesi.dll

çalıştırılır. Ardından

\>svcutil www.bsenyurt.com.UrunYonetimServisi.wsdl *.xsd /out:UrunYonetimClient.cs

komutları çalıştırılmalı ve üretilen UrunYonetimClient.cs ve output.config dosyaları istemci uygulamaya dahil edilmelidir. Output.config dosyasını **App.config** adıyla kaydetmekte yarar vardır. Nitekim istemci uygulama konfigurasyon ayaları için bu dosyayı arayacaktır. İstemci uygulama konfigurasyon dosyası aşağıdaki şekilde daha kısa olarak modifiye edilebilir.

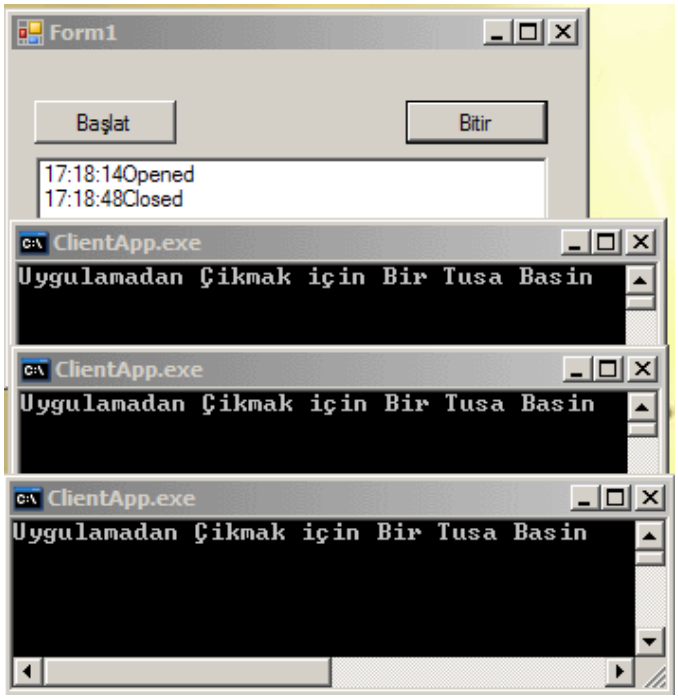
```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
   <system.serviceModel>
      <client>
         <endpoint address="net.tcp://localhost:4560/UrunYonetim.svc"
binding="netTcpBinding" bindingConfiguration="" contract="UrunYonetimServisi"
name="DefaultBinding_UrunYonetimServisi_UrunYonetimServisi" />
      </client>
   </system.serviceModel>
</configuration>
```

İstemci uygulama kodları aşağıdaki gibidir;

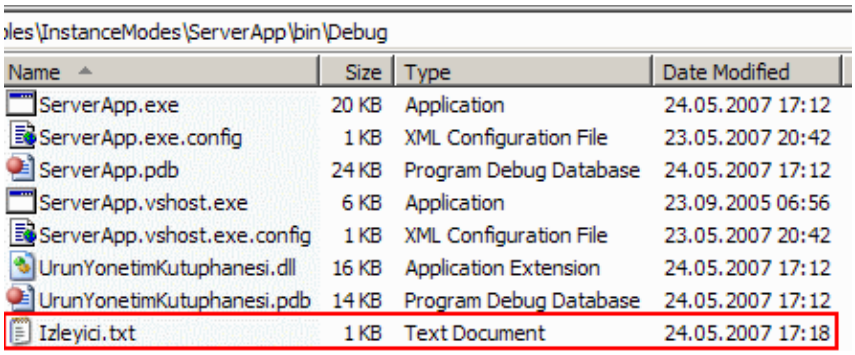
```
using System;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
```

```
            UrunYonetimServisiClient srv = new
UrunYonetimServisiClient("DefaultBinding_UrunYonetimServisi_UrunYonetimServisi");
            srv.SepeteUrunEkle("1000");
            srv.SepeteUrunEkle("1004");
            srv.SepettenUrunCikar("1000");
            srv.SiparisVer();
            Console.WriteLine("Uygulamadan çıkmak için Bir Tuşa Basın");
            Console.ReadLine();
        }
    }
}
```

İstemci uygulamada nesne örneği oluşturulduktan sonra bir dizi metod çağırımı gerçekleştirilmektedir. Şimdi uygulamayı bu haliyele test edelim. öncelikli olarak servis uygulaması çalıştırılmalı ve Başlat düğmesine basılarak servis açılmalıdır. Sonrasında istemci uygulamadan bir kaç tane örnek çalıştırmakta fayda vardır. Böylece farklı istemciler için servis nesnesinin ne şekilde örneklendiği daha kolay takip edilebilir.



Bu deneme sonrasında servis uygulamasının bulunduğu klasörde **Izleyici.txt** isimli bir dosya oluşacak ve içeriği aşağıdaki gibi olacaktır.



17:31:53: UrunYonetici nesnesi oluşturuldu.
17:31:53: Urun ekle metodu çağırıldı.
17:31:54: Urun ekle metodu çağırıldı.
17:31:55: Urun çıkart metodu çağırıldı.
17:31:55: Sipariş ver metodu çağırıldı.
17:31:57: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

17:31:51: UrunYonetici nesnesi oluşturuldu.
17:31:51: Urun ekle metodu çağırıldı.
17:31:52: Urun ekle metodu çağırıldı.
17:31:52: Urun çıkart metodu çağırıldı.
17:31:54: Sipariş ver metodu çağırıldı.
17:31:58: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

17:31:48: UrunYonetici nesnesi oluşturuldu.
17:31:48: Urun ekle metodu çağırıldı.
17:31:49: Urun ekle metodu çağırıldı.
17:31:49: Urun çıkart metodu çağırıldı.
17:31:50: Sipariş ver metodu çağırıldı.
17:31:59: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

Dikkat edilecek olursa çalıştırılan 3 istemci için 3 ayrı UrunYonetici nesne örneği oluşturulmuş ve bu istemcilerin yapmış olduğu metod çağrılarının tamamı sona erdiğinde nesneler **dispose** edilmek üzere Garbage Collector' e devredilmeye başlanmıştır. Bu durum **NetTcpBinding** için varsayılan davranıştır ve **PerSession** modunun karşılığıdır. Eğer servis nesneleri PerCall veya Single modeline göre çalıştırmak istenirse yapılması gereken **ServiceBehavior** niteliğini aşağıdaki gibi değiştirmektir. İlk olarak PerCall yapıldığındaki durumu analiz edelim.

UrunYonetici sınıfındaki değişiklik aşağıdaki gibidir.

```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.PerCall)]
public class UrunYonetici:IUrunYonetici,IDisposable
```

Buna göre yine örnek olarak 3 istemci çalıştırıp denersek Istemci.txt dosyasının içeriği aşağıdaki gibi olacaktır.

17:37:09: UrunYonetici nesnesi oluşturuldu.
17:37:09: Urun ekle metodu çağırıldı.
17:37:10: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

17:37:10: UrunYonetici nesnesi oluşturuldu.
17:37:11: Urun ekle metodu çağırıldı.
17:37:11: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

17:37:13: UrunYonetici nesnesi oluşturuldu.
17:37:14: Urun ekle metodu çağırıldı.
17:37:14: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

17:37:15: UrunYonetici nesnesi oluşturuldu.
17:37:15: Urun ekle metodu çağırıldı.
17:37:16: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

17:37:16: UrunYonetici nesnesi oluşturuldu.
17:37:17: Urun çıkart metodu çağırıldı.
17:37:17: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

17:37:18: UrunYonetici nesnesi oluşturuldu.
17:37:18: Urun ekle metodu çağırıldı.
17:37:19: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

17:37:19: UrunYonetici nesnesi oluşturuldu.
17:37:20: Urun çıkart metodu çağırıldı.
17:37:20: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

17:37:21: UrunYonetici nesnesi oluşturuldu.
17:37:21: Sipariş ver metodu çağırıldı.
17:37:22: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

17:37:22: UrunYonetici nesnesi oluşturuldu.
17:37:23: Urun çıkart metodu çağırıldı.
17:37:23: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

17:37:24: UrunYonetici nesnesi oluşturuldu.
17:37:24: Sipariş ver metodu çağırıldı.
17:37:25: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.

Dikkat edilecek olursa, istemcilerden gelen her metod çağrısında UrunYonetici sınıfına ait bir örnek oluşturulmuş ve metod işleyişi servis tarafında tamamlandıktan sonra söz konusu örnekler toplanmak üzere Garbage Collector' e devredilmiştir.

Single modunda test işlemini yapmak için yine UrunYonetici sınıfına uygulanan **ServiceBehavior** niteliğinde aşağıdaki değişiklik yapılmalıdır.

```
[ServiceBehavior(InstanceContextMode=InstanceContextMode.Single)]
public class UrunYonetici:IUrunYonetici,IDisposable
```

Tekrar, örnek olarak 3 istemci çalıştırılıp test edilirse Izleyici.txt dosyasına aşağıdaki bilgilerin yazıldığı görülür.

```
17:45:13: UrunYonetici nesnesi oluşturuldu.
17:45:16: Urun ekle metodu çağırıldı.
17:45:16: Urun ekle metodu çağırıldı.
17:45:17: Urun çıkart metodu çağırıldı.
17:45:17: Urun ekle metodu çağırıldı.
17:45:18: Sipariş ver metodu çağırıldı.
17:45:18: Urun ekle metodu çağırıldı.
17:45:19: Urun ekle metodu çağırıldı.
17:45:19: Urun çıkart metodu çağırıldı.
17:45:20: Urun ekle metodu çağırıldı.
17:45:20: Sipariş ver metodu çağırıldı.
17:45:21: Urun çıkart metodu çağırıldı.
17:45:22: Sipariş ver metodu çağırıldı.
17:45:29: UrunYonetici nesnesi için Dispose metodu çalıştırıldı.
```

Görüldüğü gibi kaç istemci olursa olsun hepsi için tek bir UrunYonetici nesne örneği oluşturulmuştur.

Makalemizin başında özellikle PerSession modunda hangi metod çağrısı ile nesne örneğinin oluşturulabileceğini veya hangi metod çağrısından sonra servise ait nesne örneğinin yok edilebileceğini **IsInitiating** ve **IsTerminating** özellikleri ile belirtebileceğimizi söylemiştik. Bu modların kullanılabilmesi için ayrıca **SessionMode** özelliğinin değerinin **Required** olarak işaretlenmiş olması gerekmektedir. Nitekim söz konusu operasyonların takibi için bir oturumun var olması gerekir. Aşağıdaki kod parçasında söz konusu sistemin IUrunYonetici arayüzüne uygulanış şekli gösterilmektedir.

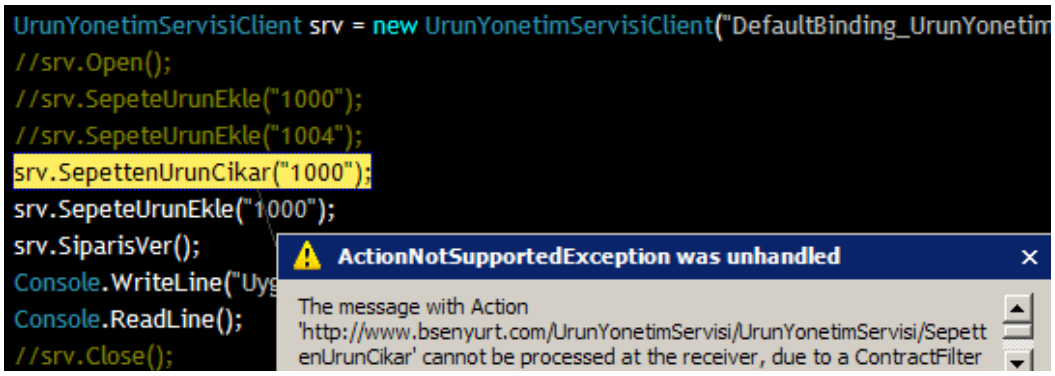
```
[ServiceContract(Name="UrunYonetimServisi",Namespace="http://www.bsenyurt.com/
UrunYonetimServisi"
,SessionMode=SessionMode.Required)]
public interface IUrunYonetici
{
```

```
    [OperationContract(IsInitiating=true)]
    void SepeteUrunEkle(string urunId);
    [OperationContract(IsInitiating=false)]
    void SepettenUrunCikar(string urunId);
    [OperationContract(IsInitiating=false,IsTerminating=true)]
    void SiparisVer();
}
```

Buna göre nesne örneği sadece SepeteUrunEkle metodu ile oluşturulabilir. Bir başka deyişle SepettenUrunCikar veya SiparisVer metodlarının SepeteUrunEkle metodundan önce çalıştırılamaması garanti altına alınmış olunur. Buna göre istemci uygulama kodlarının aşağıdaki gibi değiştirildiği göz önüne alınsın.

```
UrunYonetimServisiClient srv = new
UrunYonetimServisiClient("DefaultBinding_UrunYonetimServisi_UrunYonetimServisi");
srv.SepettenUrunCikar("1000");
srv.SepeteUrunEkle("1000");
srv.SiparisVer();
Console.WriteLine("Uygulamadan çıkmak için Bir Tuşa Basın");
Console.ReadLine();
```

Dikkat edilecek olursa önce SepettenUrunCikar metodu çağırılmaktadır. Sonrasında ise SepeteUrunEkle ve SiparisVer isimli metodlar çalıştırılmak istenir. Ancak kod bu şekilde denendiğinde çalışma zamanında aşağıdaki ekran görüntüsünde yer alan istisna alınır.

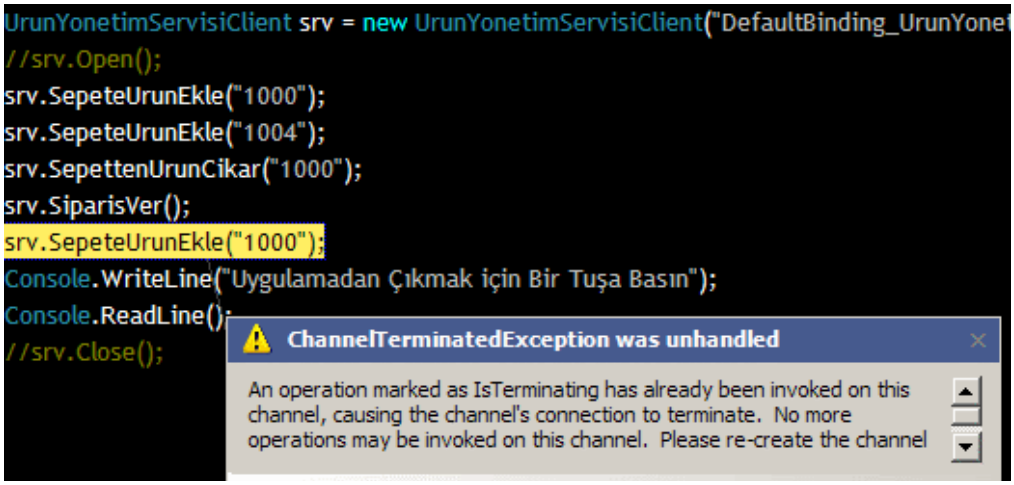


Dolayısıyla metodların çalışma sırasıda söz konusu özellikler yardımıyla garanti altına alınmıştır. Elbette **IsInitiating** özelliğinin **true** olarak atandığı SepeteUrunEkle metodu bir kere çalıştırıldıktan sonra diğer metodlar istenildiği gibi çalıştırılabilir. Diğer taraftan aşağıdaki gibi bir kod parçasında yine dikkatli olunmalıdır.

```
UrunYonetimServisiClient srv = new
UrunYonetimServisiClient("DefaultBinding_UrunYonetimServisi_UrunYonetimServisi");
srv.SepeteUrunEkle("1000");
srv.SepeteUrunEkle("1004");
srv.SepettenUrunCikar("1000");
```

srv.**SiparisVer();**
srv.**SepeteUrunEkle("1000");**
Console.WriteLine("Uygulamadan çıkmak için Bir Tuşa Basın");
Console.ReadLine();

Dikkat edilecek olursa SiparisVer metodundan sonra SepeteUrunEkle metodu çalıştırılmıştır. Oysaki SiparisVer metodu için **IsTerminating** özelliğinin değeri true olarak atanmıştır. Dolayısıyla bu metod çağrısından sonra söz konusu servis nesne örneği yok edilecektir. Buna göre istemci tarafında aşağıdaki ekranda görüldüğü gibi**ChannelTerminatedException** sınıfı tipinden bir istisna mesajı alınır.



özetle daha öncedende değindiğimiz gibi istemci tarafında istisna yönetimi adına bazı hazırlıklar yapmak **(Fault Management)** son derece isabetli bir hareket olacaktır.

Bu makalemizde servis tarafındaki nesne örnekleri ile istemci arasındaki ilişkilerin ele alınması adına **InstanceContextMode**, **IsInitiating**, **IsTerminating** gibi özelliklerden nasıl yararlanabileceğimizi ele almaya çalıştık. Oturum yönetimi adına WCF içerisinde ele alınması gereken daha pek çok konu vardır. örneğin **PerCall** modda oturum yönetiminin sağlanması gibi. Bu gibi konulara ilerleyen makalelerimizde değinmeye çalışacağız. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

örnek Uygulama İçin Tıklayınız.

WCF - Hata Yönetimi(Fault Management) (2007-05-09T19:26:00)

wcf,

Hata yönetimi her programlama dili ve geliştirme ortamı içerisinde yer alan önemli konulardan birisidir. özellikle kullanıcıların yapmış olduğu işlemler sonucunda oluşan veya sistem üzerinde beklenmeyen durumlardan doğan hataların önüne geçmek amacıyla çeşitli mekanizmalara başvurulmaktadır. Bunlardan birisi ve aynı zamanda etkili olanıda istisna

yönetimidir**(Exception Handling)**. Microsoft .Net ortamında istisna yönetimi **CLR** (Common Language Runtime - Ortak Dil çalışmaZamanı) tarafından gerçekleştirilen bir unsurdur. Ne varki, basit bir windows veya web uygulamasında etkili bir şekilde ele alınabilen istisnalar, dağıtık mimari uygulamaları **(Distributed Applications)** göz önüne alındığında daha farklı yaklaşılmak durumundadır. Bu durumu daha iyi anlayabilmek için, dağıtık mimari uygulamalarının taraflarını göz önüne almakta yarar vardır. Biz her ne kadar sunucu tarafında .Net ortamını kullanıyor olsakta, istemci açısından durum aynen geçerli değildir. örneğin dağıtık mimari uyarlamalarından birisi olan Xml Web Servisleri düşünüldüğünde istemcinin farklı platformda yer alan bir uygulama olması muhtemeldir. öyleyse düşünülmesi gereken iki önemli nokta vardır. Bunlardan birincisi, istemcilerin farklı bir platformda olabileceğidir. Bu CLR tarafınan ele alınabilecek bir istisnanın istemci tarafında ele alınamaması için yeter bir sebeptir. çünkü istemci tarafında bir CLR ortamı bulunmayabilir. İkinci önemli nokta ise sunucu üzerinde bir istisna oluşursa bunun istemci tarafına nasıl bildirileceğidir. Bir başka deyişle sunucu tarafında yakalanan hatanın, farklı bir makinedeki farklı bir bellek bölgesinde çalışan bir uygulamanın yakalayabileceği şekilde gönderilebilmesi gerekmektedir. Bu iki noktanın oluşturduğu sorunu çözmenin tek yolu herkesin kabul ettiği ortak bir standarda göre istisna bilgisi yayınlamaktır. İşte bu noktada devreye **SOAP (Simple Object Access Protocol)** girer. Bu protokol örnek olarak web servislerinde oluşacak istisnaları istemci tarafına **Soap Fault** mesajları olarak gönderir ki bu mesaj çeşidi tüm platformlar tarafında kabul edilmiştir.

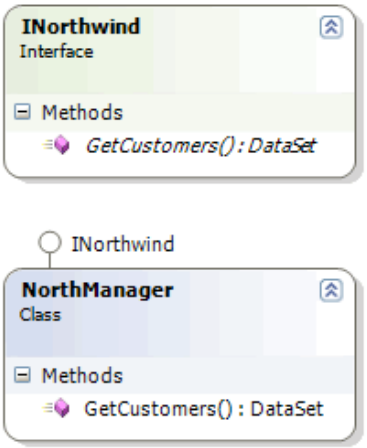
NOT : *Soap Fault hakkında daha fazla bilgi için* ***http://www.w3.org/TR/soap12-part1/#soapfault*** *adresinden faydalanabilirsiniz.*

SoapFault, sunucu tarafından fırlatılabilecek istisnaların nasıl bir formata sahip olması gerektiğine dair kuralları içerir. örneğin tüm SoapFault mesajlarında **Code** (Hata kodu bilgisi),**Reason**(Hatanın sebebi) gibi hataya ait detaylı bilgi alınmasını sağlayan özellikler yer almaktadır. .Net Framework, J2EE gibi örnek plaftormlar bu kurallara uygun olacak şekilde tasarlanmış sınıflar sunarlar. Böylece, yazacağımız servisleri kullanacak istemcilerin tamamı için ele alınabilecek istisna mesajları oluşturma şansına sahip oluruz. Peki Windows Communication Foundation için durum nasıldır? WCF içerisindede istemcilerin ele alabileceği şekilde hata mesajları tanımlayabilmek için **FaultException** isimli sınıf geliştirilmiştir. Aslında bir WCF servisi içerisinde istemci uygulamalara fırlatılabilecek istisna mesajları, karşı tarafa geçtiklerinde her zaman bir **FaultException** örneği olarak ele alınırlar. Ancak önemli olan bir nokta vardır ki buda, servis üzerinde oluşan istisnanın mutlaka karşı tarafa (istemci uygulamaya) fırlatılması(throw) gerektiğidir.

çoğu zaman servis tarafında meydana gelen istisnalar için, istemciye daha fazla bilgi taşıyabilecek şekilde kendi istisna nesnelerimizi organize etmek isteyebiliriz. .Net tarafından yaklaştığımızda bunun yolu **ApplicationException** sınıfından bir tip türetmektir. Lakin WCF için durum biraz daha farklıdır. Nitekim oluşturulan istisna nesne örneğinin Soap Fault standartlarına uygun olacak şekilde istemci tarafına bir mesaj olarak gönderilebilmesi ve orada referansının ele alınabilmesi (handle) gerekmektedir. WCF

mimarisini tanımaya çalıştığımız ilk makalemizde dört çeşit sözleşme (contract) olduğundan bahsetmiştik. Bunlardan biriside **hata sözleşmesidir(Fault Contract)**. Hata sözleşmeleri basit olarak, servis tarafında üretilen kullanıcı tanımlı hataların istemciye nasıl taşınması gerektiğini bildiren bir sözleşme çeşidir. Geliştirici olarak kendi hata bildirimlerimizi yapmak için öncelikli olarak hataya ait bilgileri taşıyacak bir veri sözleşmesininde (data contract) tanımlanması gerekmektedir. Makalemizin ilerleyen kısımlarında bu işlemlerin nasıl yapılacağını adım adım inceleyeceğiz.

WCF için, hata yönetimi ile ilişkili olarak dikkat edilmesi gereken bir diğer noktada, servisin açılması ve kapanması arasında meydana gelebilecek bazı beklenmedik tepkilere karşı nasıl tedbir alınması gerektiğidir. Her servis çalışmadan önce nesne olarak örneklenmektedir. Bunun programatik olarak **ServiceHost** sınıfına ait bir nesne örneğinin oluşturulması olarak düşünebiliriz. Sonrasında servis açılır ve istemcilerden gelecek olan talepler karşılanmaya başlar. Ne varki servis açılırken, açıldığında ve çalışırken servis tarafında bazı beklenmedik hatalar oluşabilir. Bu gibi durumlarda servise ait **Faulted** olayı tetiklenmektedir. Dolayısıyla bu olay içerisinde gereken hazırlıklar yapılarak servisin tekrar ayağa kaldırılması denenebilir. Burada servisin yaşam döngüsününde bilinmesinde fayda vardır. Bu konuyuda makalemizin sonunda ele almaya çalışacağız. Şimdi vakit kaybetmeden örnek bir senaryo üzerinden gidelim. Basit olarak Tcp bağlayıcısını kullanan Console tabanlı sunucu ve istemci uygulamaların yer aldığı bir WCF senaryosu geliştireceğiz. öncelikli olarak servis sözleşmesi ve uyarlamayı yapan tipi içeriside barındıran RemoteLib isimli WCF Servis kütüphanemizi **(WCF Service Library)** aşağıdaki gibi tasarlayalım.



INorthwind arayüzümüz tahmin edeceğiniz gibi servis sözleşmemizi**(Service Contract)** ifade etmektedir.

```
using System;
using System.Data;
using System.ServiceModel;
```

```
namespace RemoteLib
{
   [ServiceContract(Name="NorthManagerService",
Namespace="http://www.bsenyurt.com/NorthService")]
   public interface INorthwind
   {
      [OperationContract(Name="GetCustomers")]
      DataSet GetCustomers();
   }
}
```

Servis sözleşmemizi uygulayan NorthManager isimli sınıfımızın içeriği ise şu an için aşağıdaki gibidir.

```
using System;
using System.Data;
using System.ServiceModel;
using System.Data.SqlClient;

namespace RemoteLib
{
   public class NorthManager:INorthwind
   {
      #region INorthwind Members

      public DataSet GetCustomers()
      {
         DataSet ds = null;
         SqlConnection conn=null;
         try
         {
            conn= new SqlConnection("data source=.;database=AdventureWorks;integrated
security=SSPI");
            SqlDataAdapter da = new SqlDataAdapter("Select
CustomerID,CompanyName,ContactName,ContactTitle From Customers", conn);
            ds = new DataSet("CustomersSet");
            da.Fill(ds);
         }
         catch (SqlException excp)
         {
            Console.WriteLine(excp.Message);
         }
         finally
         {
            conn.Close();
```

```
            }
            return ds;
        }

        #endregion
    }
}
```

NorthManager sınıfımız içerisinde yer alan GetCustomers isimli metodumuz basit olarak Northwind veritabanında yer alan Customers isimli tablodan veri çekip sonuç kümesini bir DataSet olarak geriye döndürmektedir. Ancak SqlConnection nesne örneği oluşturulurken veritabanı adı olarak Northwind yerine AdventureWorks adı verilmiştir. Bu, çalışma zamanında çok doğal olarak bir istisnaya neden olacaktır. İstisna servis tarafı açısından düşünüldüğünde GetCustomers metodu içerisindeki catch bloğu tarafından yakalanacaktır. Peki ya sonrasında ne olacaktır? Gerçekten istemci uygulama burada **SqlException** istisna nesne örneğini yakalayabilecek midir? Bu sorulara net bir cevap verebilmek için önce sunucu uygulamamızı, ardından istemci uygulamamızı geliştirerek incelemelerimize devam edelim. Servis uygulamamız bir konsol programı olarak tasarlanabilir. WCF için gerekli ayarları konfigurasyon dosyasında tutacağız. Konfigurasyon dosyamızın içeriği şu an için aşağıdaki gibi yazılabilir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <services>
      <service name="RemoteLib.NorthManager">
        <endpoint address="net.tcp://localhost:65002/NorthService"
binding="netTcpBinding" bindingConfiguration="" name="NorthServiceEndPoint"
contract="RemoteLib.INorthwind" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

NetTcpBinding bağlayıcı tipini kullandığımız için TCP protokolü üzerinden binary formatlama gerçekleşek şekilde iletişim kurabiliriz. Servisimize istemcilerin ulaşabilmesi içinde net.tcp://localhost:65002/NorthService adresi kullanılmaktadır. Elbette siz örneği denerken istediğiniz port numarasını ele alabilirsiniz. Servis sözleşmemizde contract niteliği içerisinde bildirilmiştir. Sunucu uygulamaya ait program kodlarımız ise aşağıdaki gibi olacaktır.

```
using System;
using System.ServiceModel;

namespace ServerApp
```

```
{
   class Program
   {
      static void Main(string[] args)
      {
         ServiceHost host = new ServiceHost(typeof(RemoteLib.NorthManager));
         host.Open();
         Console.WriteLine("Sunucu  dinlemede...");
         Console.ReadLine();
         host.Close();
      }
   }
}
```

Servis uygulaması önce ServiceHost tipinden bir nesne örneği oluşturmakta ve sonrasında servisi kullanıma açmaktadır. Kullanıcı uygulamayı tuşa basarak kapattıktan sonra açılan servis içinde kapatma emri Close metodu ile verilmektedir. Gelelim istemci tarafına. HTTP üzerinden metadata yayınlaması yapmadığımız için bir önceki WCF makalesinde yaptığımız gibi **svcutil.exe** aracını RemoteLib.dll assembly' ı üzerinde kullanmamız ve istemci için gerekli proxy sınıfı ile konfigurasyon dosyasını üretmemiz gerekmektedir. Bu amaçla komut satırından svcutil aracını aşağıdaki gibi kullanalım.

önce;

```
svcutil RemoteLib.dll
```

Sonrasında ise;

```
svcutil /namespace:*,RemoteLib.NorthManager www.bsenyurt.com.NorthService.wsdl
*.xsd
```

Tahmin edeceğiniz gibi komut satırından yaptığımız bu işlemlerin sonrasında istemci için gereken proxy sınıfı ve config dosyaları başarılı bir şekilde oluşturulacaktır. İstemci uygulamamızın konfigurasyon dosyasını isterseniz daha basit olması açısından aşağıdaki gibi oluşturabilirsiniz. Nitekim svcutil ile oluşturulan standart **output.config** dosyası oldukça fazla element bilgisi içermektedir. Tek dikkat edilmesi gereken nokta contract niteliğine doğru tip bilgisini girmektir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <client>
         <endpoint address="net.tcp://localhost:65002/NorthService" binding="netTcp
Binding" bindingConfiguration="" contract="RemoteLib.NorthManager.NorthManag
erService"name="NorthClientEndPoint" />
```

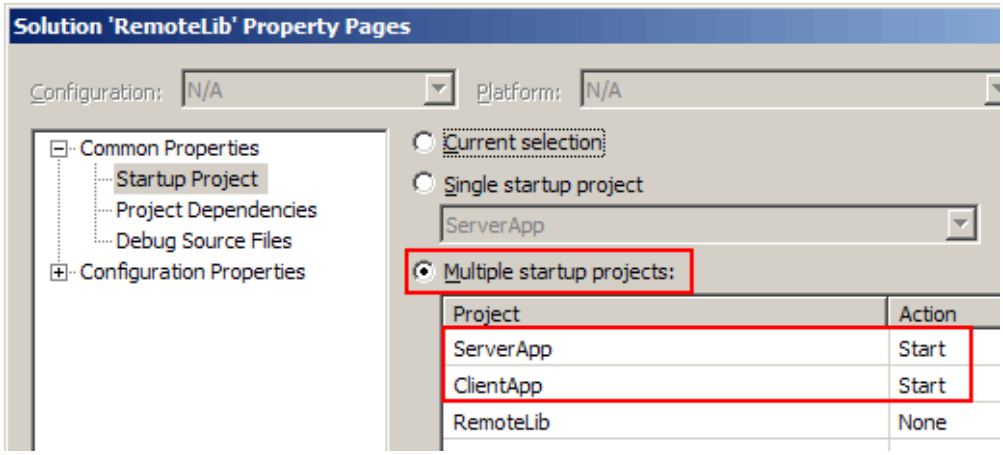
```
      </client>
   </system.serviceModel>
</configuration>
```

Bundan sonra istemci uygulamamızın kod satırlarınıda aşağıdaki gibi geliştirelim.

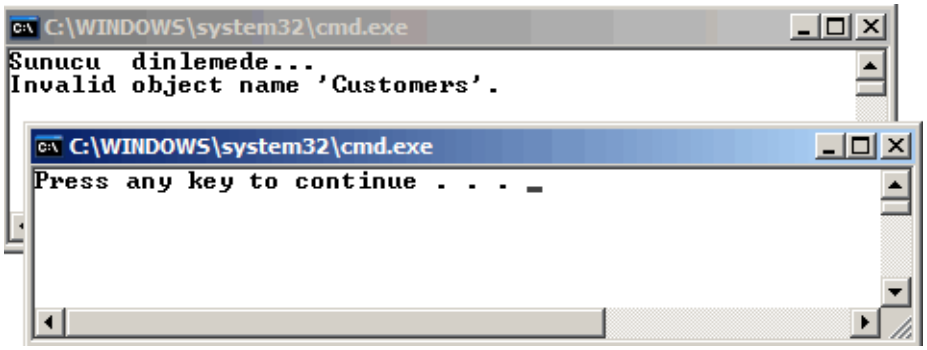
```
using System;
using System.Data;
using System.ServiceModel;
using RemoteLib.NorthManager;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         try
         {
            NorthManagerServiceClient srvClient = new
NorthManagerServiceClient("NorthClientEndPoint");
            DataSet customers = srvClient.GetCustomers();
            if(customers.Tables.Count!=0)
               Console.WriteLine(customers.Tables[0].Rows.Count.ToString());
         }
         catch (FaultException excp)
         {
            Console.WriteLine(excp.Code.Name+ "\n" + excp.Reason.ToString());
         }
      }
   }
}
```

İstemci tarafındaki kod parçamızıda dikkat ederseniz try...catch blokları arasına aldık. Artık testimize başlayabiliriz.

NOT : *İstemcinin gereken taleplerde bulunabilmesi için sunucu uygulamanın çalışıyor ve açık olması gerekecektir. Visual Studio içerisinde debug işlemleri yapmak isteyebileceğimizden* ***Solution*** *özelliklerinden* ***Multiple Startup Projects*** *seçeneğini aşağıdaki gibi set ederek F5 (Start) veya Ctrl+F5 (Start without debuging) sonrasında önce sunucunun sonrasında ise istemcinin sırayla çalıştırılmalarını sağlayabiliriz.*



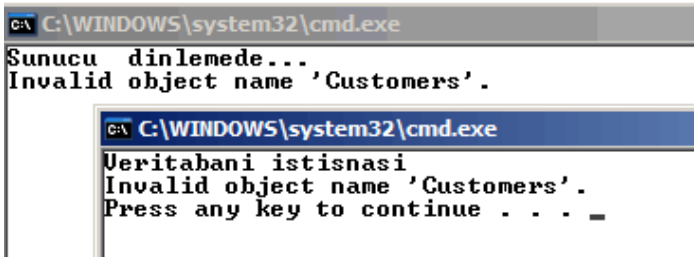
Sonuç olarak aşağıdaki ekran çıktılarını alırız.



Dikkat ederseniz, sunucu tarafındaki catch bloğuna girilmiş ve oluşan istisnaya uygun bir biçimde ekrana mesaj çıktısı verilmiştir. Ne varki bu istisna mesajını istemci tarafına gönderebilmiş değiliz. Bunun için baştada belirttiğimiz gibi servis uygulamasında bilinçli bir şekilde **FaultException** nesne örneğinin oluşturulması gerekmektedir. Bu amaçla NorthManager sınıfımızın GetCustomers metodundaki catch bloğunu aşağıdaki gibi düzenleyelim.

```
catch (SqlException excp)
{
   Console.WriteLine(excp.Message);
   throw new FaultException(excp.Message, new FaultCode("Veritabani istisnası"));
```

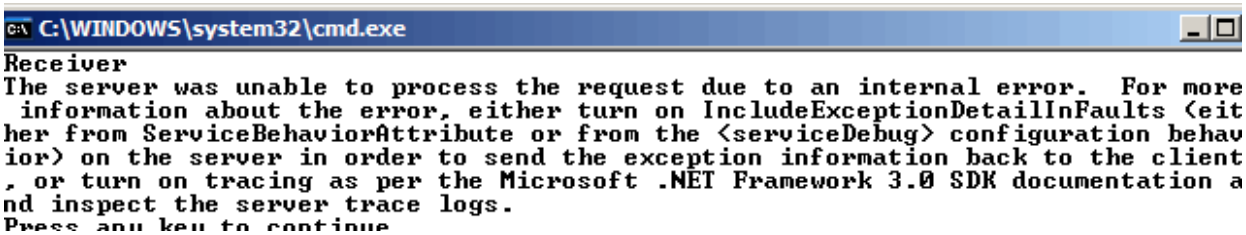
FaultException sınıfının farklı şekilde yüklenmiş yapıcı metodlar(constructors) vardır. Burada kullanılan versiyonda ilk parametre olarak hatanın sebebi (**Reason**) ikinci parametre olarakta hata kodu (**FaultCode**) verilmektedir. Oluşan istisna sonrasında üretilecek ve istemciye gidecek olan **SoapFault** mesajı içerisinde, yapıcı metod içerisinde kullandığımız parametre bilgileri yer alacaktır. Buna göre uygulamalarımızı tekrar çalıştırırsak aşağıdaki ekran görüntülerini elde ederiz.



Gördüğünüz gibi artık istemci tarafında, sunucudan fırlatılan **FaultException** nesnesinin içeriğini yakalayabilmekteyiz. Servis tarafında yer alan NorthManager sınıfına ait GetCustomers metodunda istisna oluştuğunda, istemci uygulamalara FaultException tipinden bir nesne örneği fırlatmaktansa, normal bir **Excpetion** nesnesi fırlatılmasıda denenebilir *(yada başka tipte bir istisna nesne örneği).* Bu durumda istemci uygulama çok farklı bir mesaj alacaktır. Durumu analiz etmek için GetCustomers metodu içerisinde aşağıdaki değişiklikleri yapalım.

throw new **Exception**("Veritabanı adı yanlış");

Testi tekrar yaptığımızda istemci uygulamamız için aşağıdaki ekran görüntüsünü elde ederiz.

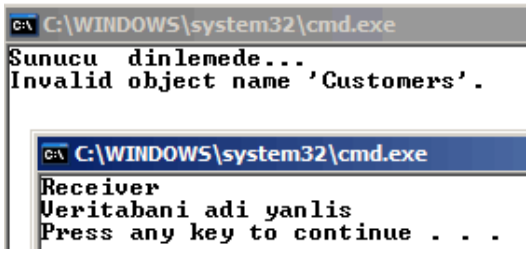


Dikkat ederseniz istemci tarafına son derece enteresan bir çıktı gelmiştir. Asıl hata mesajının içeriğini alabilmek için servis tarafındaki konfigurasyon bilgilerinde biraz değişiklik yapmak gerekecektir. öncelikle olarak bir servis davranışı **(service behavior)** tanımlanmalıdır. Servis davranışının (Service Behavior) **serviceDebug** özelliğinin **includeExceptionDetailInFaults**isimli niteliğinede **true** değeri atanmalı ve son olarak bu davranış servise bildirilmelidir. Söz konusu değişiklikleri servis uygulamasının **App.config** isimli konfigurasyon dosyasına aşağıdaki gibi ekleyebiliriz.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="NorthServiceBehavior">
          <serviceDebug includeExceptionDetailInFaults="true" />
        </behavior>
      </serviceBehaviors>
```

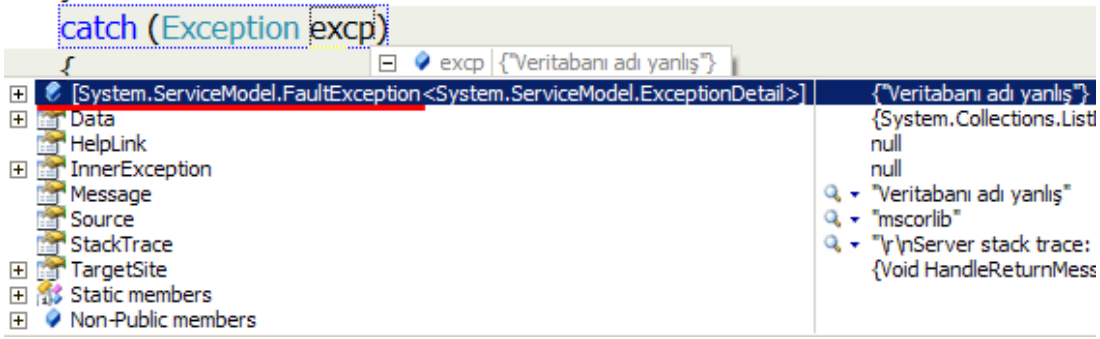
```
    </behaviors>
    <services>
      <service behaviorConfiguration="NorthServiceBehavior" name="RemoteLib.N
orthManager">
        <endpoint address="net.tcp://localhost:65002/NorthService"
binding="netTcpBinding" bindingConfiguration="" name="NorthServiceEndPoint"
contract="RemoteLib.INorthwind" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Böylece servis tarafında ürettiğimiz FaultException dışındaki bir istisnanın istemci tarafına taşınması mümkün olabilir ki istemci uygulamayı tekrar test ettiğimizde aşağıdaki ekran görüntüsünü elde edebiliriz.



Burada dikkat edilmesi gereken önemli bir nokta daha vardır. Servis uygulaması tarafında her nekadar bir Exception sınıfına ait nesne örneği fırlatsakta, istemci tarafında bu yine **FaultException**sınıfına ait bir örnek olarak ele alınacaktır. Bunu görebilmek için istemci uygulamada FaultException yerine **Exception** nesnesi yakalanmaya çalışabilir. Uygulamayı debug ettiğimizde aynen aşağıdaki ekran görüntüsünde olduğu gibi istisna tipinin aslında FaultException olduğu görülecektir.

```
static void Main(string[] args)
{
    try
    {
        NorthManagerServiceClient srvClient = new NorthManagerServiceClien
        DataSet customers = srvClient.GetCustomers();
        if(customers.Tables.Count!=0)
            Console.WriteLine(customers.Tables[0].Rows.Count.ToString());
    }
    catch (Exception excp)
    {
```



Gelelim daha güçlü hata istisna nesnelerinin nasıl oluşturabileceğimize**(Strongly Typed Faults)**. Yazımızın başındada belirttiğimiz gibi, öncelikle istemci tarafına gönderilecek olan mesajın bir veri sözleşme olarak tanımlanması gerekmektedir. Bu amaçla aşağıdaki gibi bir sınıfı RemoteLib içerisinde tanımlayabiliriz.

TabloAdiFault isimli sınıfımız tipik olarak bir veri sözleşmesi (data contract) tanımlamaktadır.

```
using System;
using System.Runtime.Serialization;

namespace RemoteLib
{
```

```
    [DataContract]
    public class TabloAdiFault
    {
        private string _faultCode;
        private string _Message;
        private string _Reason;

        [DataMember]
        public string FaultCode
        {
            get { return _faultCode; }
            set { _faultCode = value; }
        }
        [DataMember]
        public string Message
        {
            get { return _Message; }
            set { _Message = value; }
        }
        [DataMember]
        public string Reason
        {
            get { return _Reason; }
            set { _Reason = value; }
        }
        public TabloAdiFault(string faultCode,string message,string reason)
        {
            Reason = reason;
            Message = message;
            FaultCode = faultCode;
        }
    }
}
```

Burada dikkat edilmesi gereken noktalardan biriside **DataContract** ve **DataMember** niteliklerinin kullanılabilmesi için **System.Runtime.Serialization.dll** assembly' ının projeye dahil edilmiş olması gerekmektedir. *(Biz uygulamamızı WCF Service Library şablonundan tasarladığımız için bu assembly' lar otomatik olarak referans edilmiş olacaktır.)* Artık tanımlamış olduğumuz bu sınıfın, istemcilere FaultException olarak gönderilmesi için gereken hazırlıkları yapabiliriz. öncelikli olarak bu verinin hangi metodlardan döndürülecekse **FaultContract** niteliği yardımıyla bildirilmesi gerekmektedir. Bu nedenle servis sözleşmemizde gerekli tanımlamayı aşağıdaki gibi yapmamız yeterli olacaktır.

```
[ServiceContract(Name="NorthManagerService",Namespace="http://www.bsenyurt.com/
NorthService")]
public interface INorthwind
{
   [FaultContract(typeof(TabloAdiFault))]
   [OperationContract(Name="GetCustomers")]
   DataSet GetCustomers();
}
```

FaultContract niteliği parametre olarak metoddan döndürülebilecek tipe ait bir bilgi içermektedir. Bu bilginin istemci tarafına serileştirilerek(Serializable) gitmesi gerektiği için, zaten veri sözleşmesi olarak tanımlanmıştır. Elbette istisnaları fırlatırkende yapmamız gereken bazı işlemler olacaktır. Bu amaçla GetCustomers metodumuzun içeriğini aşağıdaki gibi değiştirelim.

```
public DataSet GetCustomers()
{
   DataSet ds = null;
   SqlConnection conn=null;
   try
   {
      conn= new SqlConnection("data source=.;database=AdventureWorks;integrated
security=SSPI");
      SqlDataAdapter da = new SqlDataAdapter("Select
CustomerID,CompanyName,ContactName,ContactTitle From Customers", conn);
      ds = new DataSet("CustomersSet");
      da.Fill(ds);
   }
   catch (SqlException excp)
   {
      Console.WriteLine(excp.Message);
      TabloAdiFault tblFault = new TabloAdiFault("TabloAdi", excp.Message,
"Tablo adı yanlış");
      throw new FaultException<TabloAdiFault>(tblFault);
   }
   finally
   {
      conn.Close();
   }
   return ds;
}
```

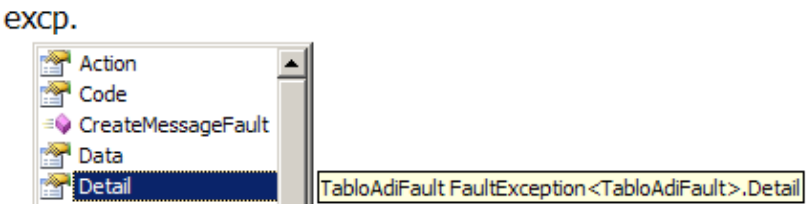
öncelikle TabloAdiFault sınıfımızı örnekliyoruz. Bu bir exception sınıfı olmadığından istemci tarafına fırlatılabilmesi için **FaultException** sınıfının ilgili yapıcı metoduna parametre olarak verilmesi gerekmektedir. Bu işlem için Framework içerisinde

FaultException sınıfının generic bir versiyonu kullanılmaktadır. Artık istemci tarafını aşağıdaki gibi kodlayabiliriz. Ancak bu işlemlerin ardından istemciler için gerekli **proxy** sınıflarını yeniden oluşturmamız gerekecektir. Dolayısıyla **svcutil** aracını tekrardan ele almalıyız. Svcutil aracını kullandığımız takdirde oluşan proxy sınıfı içerisinede **TabloAdiFault** sınıfının getirildiğini görebiliriz.

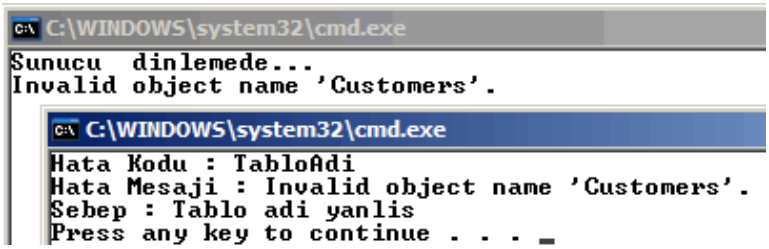
çok doğal olarak sadece public üyeler ve bunlara ilişkin alanlar bu sınıf içerisine dahil edilmiştir. Ama aynı zamanda **ExtensionDataObject** tipinden değerler ile çalışan bir özellikte ilave edilmiştir ki bu özellik IExtensibleDataObject arayüzünden uygulanmaktadır. Artık istemci tarafındaki catch bloğumuzu aşağıdaki gibi kodlayabiliriz.

```
catch (FaultException<TabloAdiFault> excp)
{
   Console.WriteLine("Hata Kodu : "+excp.Detail.FaultCode.ToString() + "\nHata Mesajı
: " + excp.Detail.Message + "\nSebep : " + excp.Detail.Reason);
}
```

Detail özelliği aslında generic tipimize ait referansı bir başka deyişle **TabloAdiFault** nesne örneğini ele almaktadır.



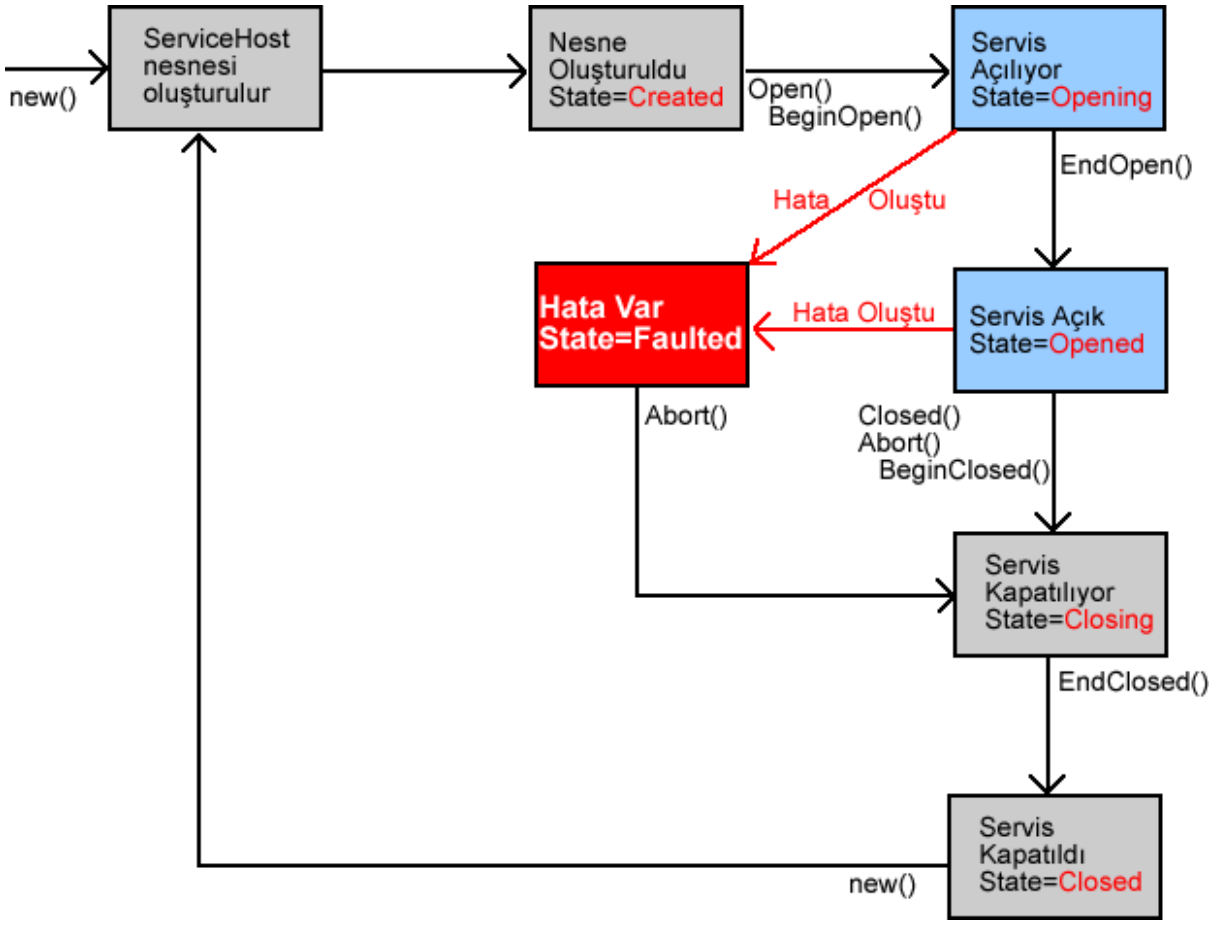
Dolayısıyla Detail üzerinden, TabloAdiFault sınıfı içerisinde tanımlanmış özelliklere erişebiliriz. öyleki çalışma zamanında bu özelliklerin değerleri servis uygulaması tarafında üretilip istemci tarafına taşınacaktır. O halde uygulamayı deneyip çalışmanın sonuçlarını görebiliriz. Aşağıdaki ekran görüntüsünde bu sonuçlar yer almaktadır.



WCF uygulamalarında hata yönetimi adına sunucu tarafındada yapılması gerekenler olabilir. özellikle servisin açılması, açıldıktan sonra kapatılana kadar geçen süre içerisinde bazı beklenmeyen sistem hataları meydana gelebilir. Bu gibi durumlarda istenirse **ServiceHost** sınıfına ait **Faulted** olay metodu kullanılabilir ve hataların ele alınması sağlanabilir. Faulted olayını daha iyi kavrayabilmek için aslında bir servisin yaşam çemberini **(life cycle)** incelemekte ve hangi durumlarda Faulted olayının tetikleneceğini bilmekte fayda vardır. Temel olarak bir servis uygulamasının oluşturulması (**Create**), açılması (**Open**), iptal edilmesi (**Abort**) veya durdurulması (**Stop**) gibi durumlar söz konusudur. Bu durumlar arasında geçişler yapılabilmesi içinde bazı metodların çağırılması gerekmektedir. Söz gelimi servis nesnesi oluşturulduktan sonra açmak için **Open** metodu çağırılır. Bu gibi metod çağrıları sonrasında servisin durumu (**state**) sürekli değişecektir. İşte bu geçişler (**transitions**) sırasında oluşabilecek bazı beklenmedik hatalara karşılık Faulted olayı ele alınabilir.

NOT : *ServiceHost sınıfının bir servisin durumunu öğrenebilmek amacıyla* ***CommunicationState*** *isimli enum sabitinden değerler döndüren* ***State*** *isimli sadece okunabilir (read-only) bir özelliği vardır. Bu enum sabitinin alabileceği değerler* ***Created, Opening, Opened, Faulted, Closing, Closed*** *dır.*

Aşağıdaki çizelgede bir servisin alabileceği durumlar, bu durumlar arasında geçiş yapılması için gereken metodlar ve Faulted olayının devreye girebileceği zamanlar ifade edilmeye çalışılmaktadır.



Şekildende göreceğiniz gibi, Faulted olayının tetiklenmesi sonrasında ele alınabilecek senaryolardan birisi servisin o ana kadar olan tüm işlemlerini iptal etmek (Abort) ve servisi tekrardan oluşturup açmayı denemek olacaktır. Abort metodu çağırıldığında eğer askıda bekleyen talepler varsa bunlara cevap verilmesi beklenmez. Oysaki Close metodu çağrıldığında askıda bekleyen talepler(request) var ise bunlar cevaplanır ama emir verildikten sonra istemciden yeni talepler alınmaz. Yukarıdaki senaryoyu uygulamak istediğimizde servis tarafındaki kodlarımızı aşağıda olduğu gibi geliştirebiliriz.

```
using System;
using System.ServiceModel;

namespace ServerApp
{
   class Program
   {
      static ServiceHost host;

      static void Main(string[] args)
      {
         host = new ServiceHost(typeof(RemoteLib.NorthManager));
         host.Faulted += delegate(object sender, EventArgs e)
```

```
        {
            host.Abort();
            host = new ServiceHost(typeof(RemoteLib.NorthManager));
            host.Open();
        };
        host.Open();
        Console.WriteLine("Sunucu dinlemede...");
        Console.ReadLine();
        host.Close();
    }
  }
}
```

Dikkat ederseniz Faulted olay metodu içerisinde önce servis **Abort** metodu ile iptal edilmekte, sonrasında servis nesne örneği oluşturulmakta ve servis tekrar açılmaya çalışılmaktadır. Bu olay metodu içerisinde loglama işlemleride yapılarak hataların daha kolay izlenmesi sağlanabilir.

Bu makalemizde WCF uygulamalarında hata yönetimini farklı şekillerde ele almaya çalıştık. özellikle hata yönetiminin basit bir istisna yönetiminden çok daha farklı olmamakla birlikte farklı platformların söz konusu olduğu dağıtık bir ortamda daha titiz bir biçimde ele alınması gerektiğini öğrendik. Bunların dışında istemci tarafında oluşacak hataların sunucuya gönderilmesi istenebilir. Ancak istemcilerinin Java gibi farklı platformlar olabileceği göz önüne alınırsa istemcinin sunucuya göndereceği mesajların Soap Fault' a uygun olması gerekecektir. Bu konuyu inceleyip ilerleyen makalelerimizde ele almaya çalışacağız. Böylece geldik bir makalemizin daha sonuna. İlerleyen makalelerimizde WCF mimarisinin detaylarını incelemeye devam ediyor olacağız. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

örnek Uygulama İçin Tıklayınız.

WCF - Mesaj Seviyesinde Güvenlik (2007-05-09T19:22:00)

wcf,

Dağıtık mimariye yönelik olarak geliştirilen uygulamalarda güvenlik son derece önemli bir faktördür. özellikle farklı süreçler (process) içerisinde yer alan uygulamalar, birbirleriyle haberleşirken aradaki veri trafiği mesajlar üzerine kuruludur. Bu mesajların ağ ortamları üzerinden (Internet-Intranet) hareket etmeside güvenlik ile ilgili olarak dikkat edilmesi gereken noktaların sayısını arttırır. Temel olarak dağıtık mimarilerde güvenlik düşünüldüğünde, kullanıcıların sunucu tarafından doğrulanması **(authentication)**, doğrulanan kullanıcıların hangi fonksiyonellikleri kullanabileceğine bakılması **(authorization)**, arada hareket etmekte olan mesajların ne şekilde şifreleneceğinin**(encryption)** veya çözümleneceğinin**(decryption)**belirlenmesi gibi konular

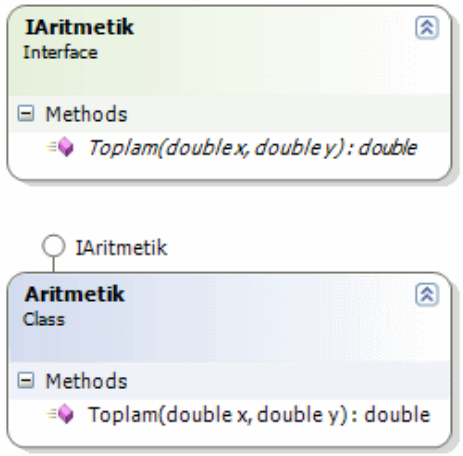
yer almaktadır. Bu tip işlemlere ihtiyaç duyulmasının bilinen pek çok nedeni vardır. Bunlardan bir kaçı aşağıda maddeler halinde listelenmiştir.

- İyi geliştiriciler veya sistem bakımından sorumlu olanlar, genellikle ağ üzerindeki trafiği kontrol etmek, performans kayıplarını tespit etmek amacıyla çeşitli programlar kullanırlar.*(örneğin biz makalemizde istemci ile sunucu arasındaki mesajları görmek adına **Microsoft Service Trace Viewer** aracından faydalanacağız.)* Bu programlar sayesinde ağ üzerinde istemciler ve sunucu arasında hareket eden mesajlar görülebilir. Ancak kötü niyetli kişilerde bu paketleri takip edebilirler. Eğer paketler içerisinde hassas bilgiler var ise söz konusu bilgilerin görülmesi istenmeyen bir durumdur. Dolayısıyla mesajların çeşitli algoritmalar ile (**TripleDes, SHA** vb...) şifrelenmesi çok doğrudur.
- Yine olayların baş kahramanı olan kötü niyetli kullanıcılar istemciler ve sunucu arasındaki mesajları yakalayıp değiştirebilirler. Bu bir önceki durumdan biraz daha farklıdır. Nitekim mesajın orjinal haliyle gitmesi yerine bozulmuş haliyle taşınması söz konusudur. Bu yaklaşım elbetteki veri bütünlüğünü tamamen bozan bir etki yapar. Sertifikalandırma ve dijital imza gibi tekniklerin kullanılması tercih edilerek gereken tedbirler alınabilir.
- Bazı durumlarda istemciler gerçek sunucu yerine, araya alınmış başka bir yalancı sunucuya başvuruda bulunuyor olabilir. özellikle internet tabanlı bankacılık uygulamalarında zaman zaman duyduğumuz bu senaryo dağıtık mimari uygulamaları içinde söz konusu olabilir. önlem olarak çift taraflı doğrulama modeli ele alınabilir.
- Kötü niyetli kişilerin yakaladığı mesajlar, sadece bozulmakla kalmaz defalarca sunucuya gönderilebilir. Dolayısıyla sunucunun doğru bir şekilde çalışması engellenmişte olur. Bu gibi bir duruma önlem olarak güvenli bir iletişim ortamı sağlanması gerekmektedir.

Dikkat edilecek olursa bu basit senaryolar bile, bir dağıtık mimari sisteminin çökmesi için yeterlidir. WCF mimarisinde kullanıcıların doğrulanması sırasında veya doğrulama işlemleri sonrasında arada hareket edecek mesajlar söz konusu olduğunda taşınan hassas bilgiler söz konusudur. Söz konusu bilgilerin güvenliğini iletişim seviyesinde **(transport level)** ve mesaj seviyesinde**(message level)** olmak üzere iki şekilde sağlayabiliriz. İletişim seviyesinde güvenliği sağlamanın bilinen yollarından birisi **HTTPS**' dir. Dolayısıyla iletişim seviyesinde sağlanan güvenliğin işletim sistemi ve donanıma bağlı olaraktan daha etkili ve performanslı olduğunu düşünebiliriz. Mesaj seviyesinde sağlanan güvenlik göz önüne alındığında sorumluluk servisin üzerindedir. Diğer taraftan servis ve istemci arasında gidecek bilgilerin şifrelenmesi hem sunucuyu hemde istemciyi ilgilendirmektedir.

özellikle Web servisleri üzerinde uygulanabilen güvenlik seçenekleri düşünüldüğünde **(Web Service Enhancements)**, Windows Communication Foundation içerisinde benzer imkanları sağlamak çok daha kolaydır. Bu makalemizde ilk olarak mesaj seviyesinde güvenliğin nasıl sağlanabileceğine dair adım adım ilerleyeceğimiz bir örnek üzerinde durmaya çalışacağız. Her zaman olduğu gibi basit bir sınıf kütüphanesi ile işe

başlamak gerekiyor. Sınıf kütüphanesi (Class Library) bir **WCF Library** olarak tasarlanabilir ve içerisinde aşağıdaki servis sözleşmesi ile tip yer alabilir.



Servis sözleşmemize ait arayüz (Interface) aşağıdaki gibidir;

```
using System;
using System.ServiceModel;

namespace AritmetikLib
{
   [ServiceContract(Name="AritmetikServisi",Namespace="http://www.bsenyurt.com/AritmetikServisi")]
   public interface IAritmetik
   {
      [OperationContract]
      double Toplam(double x, double y);
   }
}
```

Arayüzü uyarlayan sınıf ise aşağıdaki gibidir.

```
using System;

namespace AritmetikLib
{
   public class Aritmetik:IAritmetik
   {
      #region IAritmetik Members
      public double Toplam(double x, double y)
      {
         return x + y;
      }
```

```
      #endregion
   }
}
```

Bu noktada istenirse üretilen assembly' dan faydalanarak istemci için gerekli **proxy** sınıfının yazdırılması sağlanabilir. Daha önceki makalelerden hatırlayacağınız gibi bu amaçla **svcutil**aracı kullanılabilir.

Artık servis uygulamasını tasarlamaya başlayabiliriz. Amacımız güvenlik konusuna değinmek olduğundan servis ve istemci uygulamayı basit birer Console uygulaması olarak ele alacağız. İlk olarak servis uygulaması ile başlayalım. Her zaman olduğu gibi bu uygulama için kritik olan referans **System.ServiceModel.dll** isimli assmebly' dir. Servis uygulaması için gereken konfigurasyon bilgileri başlangıç aşamasında aşağıdaki gibi olmalıdır. Bu bilgileri kolay bir şekilde **Edit WCF Configuration** seçeneği ile açılan **Microsoft Service Configuration Editor** arabirimi yardımıylada hazırlayabiliriz.

Servis uygulaması için konfigurasyon dosyası;

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <services>
         <service name="AritmetikLib.Aritmetik">
            <endpoint address="net.tcp://localhost:9002/AritmetikServisi" binding="netTc
pBinding" name="AritmetikServerEndPoint" contract="AritmetikLib.IAritmetik" />
         </service>
      </services>
   </system.serviceModel>
</configuration>
```

Servis tarafında TCP protokolünü baz alacak şekilde bir ayarlama yapılmıştır. Buna göre istemciler söz konusu endPoint erişimi için **net.tcp://localhost:9002/AritmetikServisi** adresini kullanacaktır. Diğer taraftan bağlayıcı tip olarak **NetTcpBinding** tipi ele alınmaktadır. Bu noktadan sonra sunucu uygulamanın kodları aşağıdaki gibi tasarlanabilir.

```
using System;
using System.ServiceModel;
using AritmetikLib;

namespace Server
{
   class Program
   {
      static void Main(string[] args)
```

```
        {
            ServiceHost host = new ServiceHost(typeof(Aritmetik), new
Uri("net.tcp://localhost:9002/AritmetikServisi"));
            host.Open();
            Console.WriteLine("Host state " + host.State);
            Console.ReadLine();
            host.Close();
        }
    }
}
```

Sunucu uygulamanın çalıştığı süre boyunca istemcilere hizmet verebilmesi ve söz konusu endPoint' e ait nesne referanslarını kullandırtması için her zaman olduğu gibi ServiceHost tipinden bir örnek kullanılmaktadır. Open metodu ile servis açlmakta, uygulama kapatılırken Close metodu ile söz konusu servis sonlandırılmaktadır.

İstemci uygulamamıza ait konfigurasyon dosyasıda başlangıç için aşağıdaki gibi tasarlanabilir.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
    <system.serviceModel>
        <client>
            <endpoint address="net.tcp://localhost:9002/AritmetikServisi" binding="netTcpB
inding" contract="AritmetikServisi" name="TemelMatClientEndPoint" />
        </client>
    </system.serviceModel>
</configuration>
```

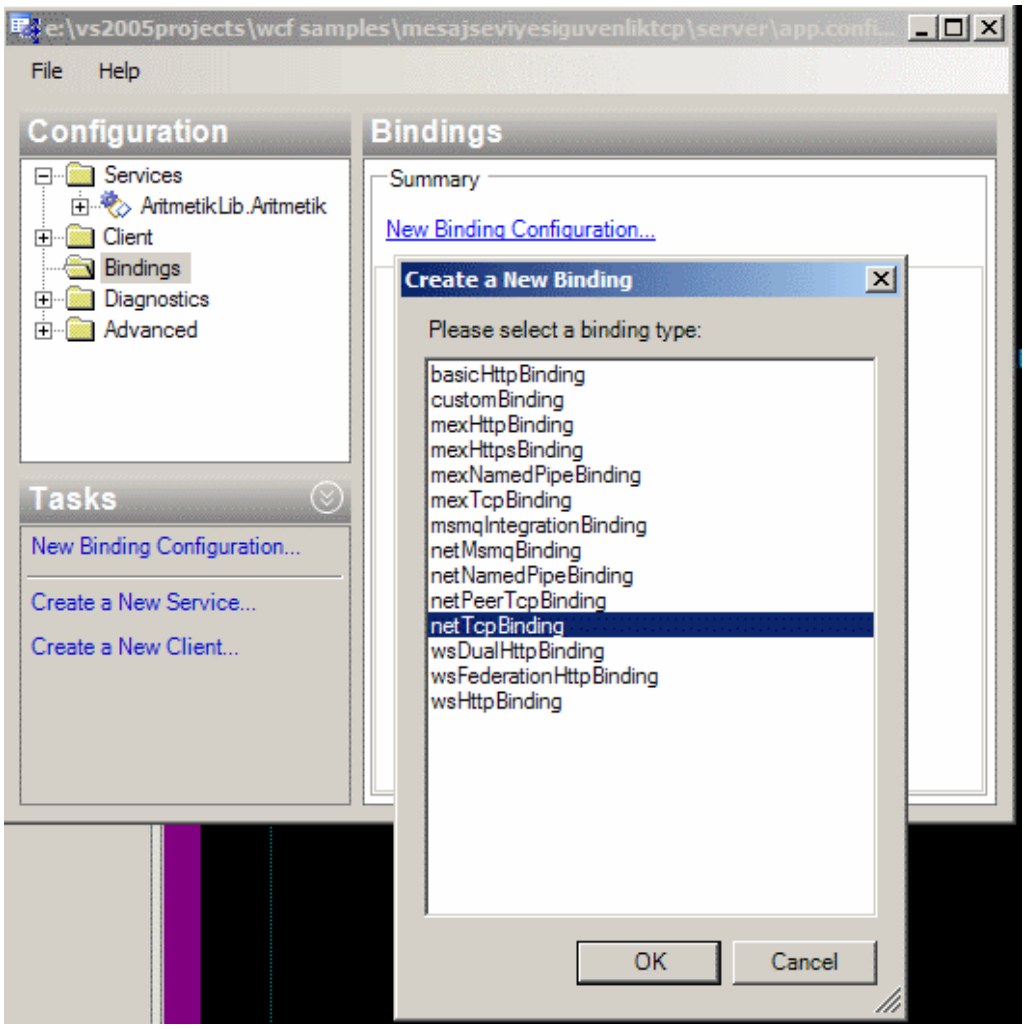
İstemci uygulamaya ait kodlar ise aşağıdaki gibidir.

```
using System;
using System.ServiceModel;

namespace Client
{
    class Program
    {
        static void Main(string[] args)
        {
            try
            {
                AritmetikServisiClient client = new
AritmetikServisiClient("TemelMatClientEndPoint");
                double sonuc = client.Toplam(3, 5);
```

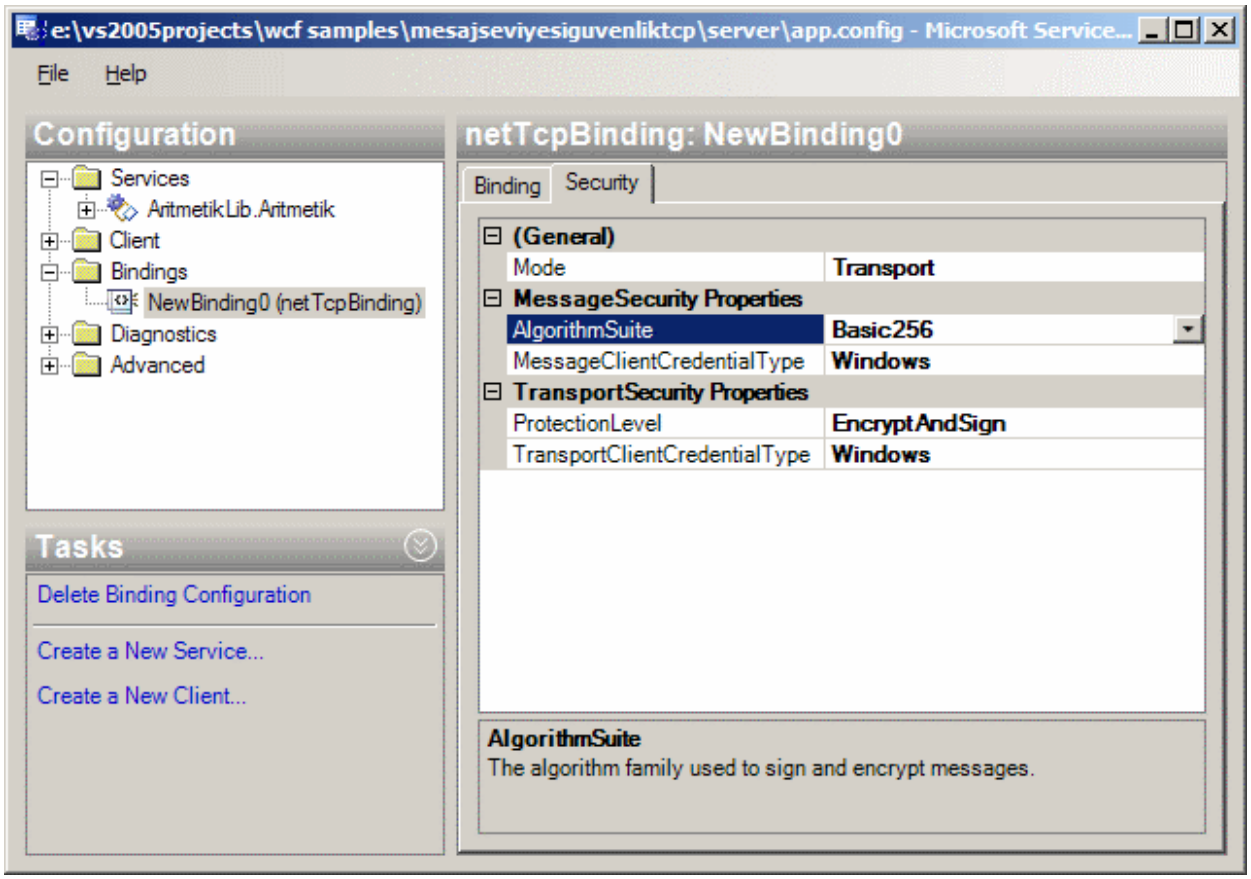
```
            Console.WriteLine(sonuc.ToString());
            Console.ReadLine();
        }
        catch (FaultException excp)
        {
            Console.WriteLine(excp.Message);
        }
    }
  }
}
```

İstemci uygulamada yer alan **AritmetikServisiClient** isimli sınıf, svcutil aracı yardımıyla AritmetikLib.dll isimli assembly üzerinden elde edilmektedir. *(Buraya kadarki adımlarımız daha önceki makalelerimizde incelendiğinden detaylı bir şekilde açıklanmamıştır.)*

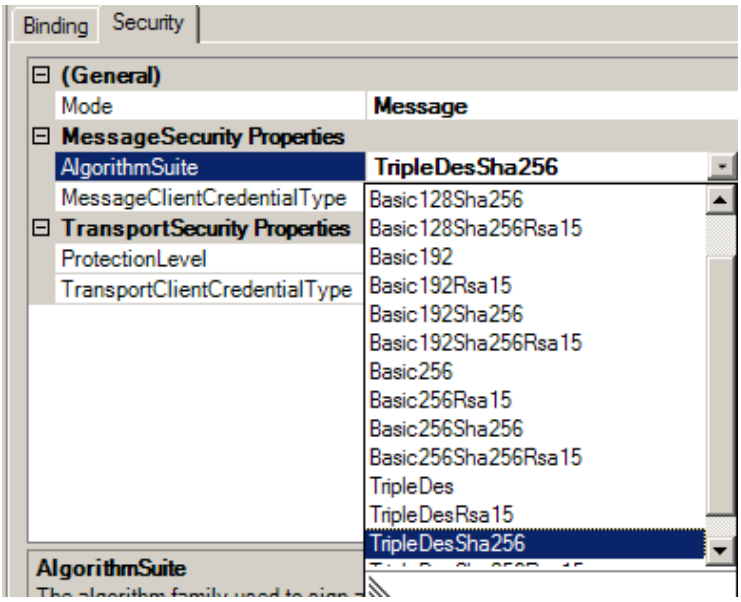
Bu makalemizde bizim için önemli olan mesaj seviyesinde güvenliğin **(Message Level Security)** nasıl sağlanacağıdır. İlk olarak servis tarafında bazı ayarlamaların yapılması gerekmektedir. örneğin, mesaj seviyesinde güvenlik uygulanacağının, mesajların belirtilen ve bilinen bir algoritmaya göre şifreleneceğinin belirtilmesi vb... Bu amaçla servis tarafında yer alan App.config dosyası **Microsoft Service Configuration Editor** yardımıyla açılıp yeni bir bağlayıcı konfigurasyon **(Binding Configuration)** eklenmeli ve **EndPoint** ile ilişkilendirilmelidir. öncelikle**New Binding Configuration** linkine tıklanarak **netTcpBinding** tipi seçilir ve eklenir. Bir başka deyişle servis tarafında kullanılan Binding tipi için gereken ayarlamaların yapılması için yeni bir element eklenmektedir.



Uygulama netTcpBinding bağlayıcı tipini kullandığı için, bağlayıcı konfigurasyon ayarları buna göre yapılmalıdır. Yeni eklenen elementin **Security** sekmesine geçildiği takdirde gereken güvenlik ayarları ile ilişkili özellikler olduğu görülebilir.



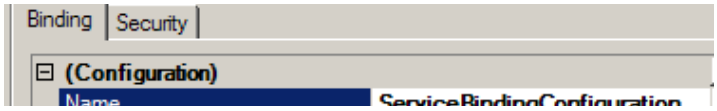
Mode özelliğinde güveinlik seviyesinin mesaj, iletişim veya bunların kombinasyonu olup olmadığı belirlenir. örneğimizde mesaj seviyesinde güvenlik gerçekleştirileceğinden Mode özelliğine **Message** değeri verilmiştir. **AlgorithmSuite** özelliğinde, mesajların hangi modele göre şifreleneceği belirlenir. Burada oldukça fazla ve yeterli seviyede şifreleme algoritmasına ait tanımlamalar yer almaktadır. Dikkat edilmesi gereken nokta burada belirtilen şifreleme modelinin istemci içinde aynı olması gerektiğidir. Bu durum Mode özelliğinin değeri içinde geçerlidir. Yani servis tarafında mesaj seviyesinde güvenlik kullanılacağı belirtiliyorsa, istemci uygulamada aynı model kullanılmalıdır.



Biz örnek olarak **TripleDesSha256** modelini göz önüne alabiliriz. AlgorithmSuite için varsayılan değer Basic256' dır. Security kısmında yer alan özelliklerden bir diğeri olan**MessageClientCredentialType** ile, istemcilerin doğrulamasının (Authentication) nasıl yapılacağı belirlenmektedir.

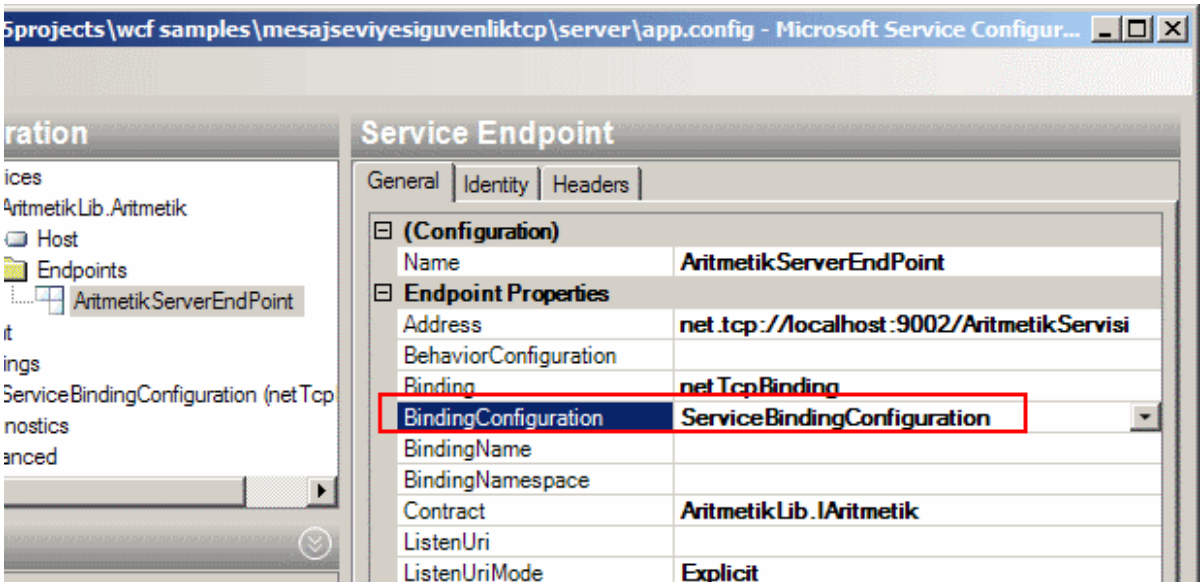
Ekran görüntüsündende izlenebileceği gibi söz konusu özelliğe, **Windows, UserName, Certificate, IssuedToken ve None** değerlerinden birisi verilebilir. örnekte Windows değeri verilmiştir. Bir başka deyişle istemcilerin servise gönderecekleri ehliyet bilgileri (**Credentials**) windows tabanlı doğrulama modeline göre aktarılacaktır. örnekte yer alan istemci ve sunucu uygulamalar aynı makine üzerinde yer aldıklarında varsayılan olarak makineyi açan kullanıcı bilgileri ele alınacaktır.

TransportSecurity kısmındaki özellikler iletişim seviyesinde güvenlik modeli için gerektiğinden şu aşamada varsayılan halleri ile bırakılabilirler. Bu işlemlerin ardından bağlayıcı konfigurasyon bilgileri için **ServiceBindingConfiguration** adı belirtilerek gereken değişiklikler kaydedilebilir.



Artık tek yapılması gereken servisten sunulan ilgili endPoint için **BindingConfiguration** özelliğine hazırlanan ServiceBindingConfiguration değerini

vermektir. Böylece endPoint içerisinde kullanılan binding tipinin belirttiği güvenlik ayarları aktif olarak set edilmiş olunur.



Bu işlemlerin ardından servis tarafı için konfigurasyon dosyasının içeriği aşağıdaki gibi yenilenecektir. Dikkat edilecek olursa görsel tarafta yapılan tüm eklentiler buraya element ve nitelikler olarak geçirilmiştir. Dikkat edilmesi gereken nokta, endPoint elementi içerisindeki bindingConfiguration elementinin değerinin, netTcpBinding alt elementindeki name özelliğinin değeri oluşudur.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <bindings>
      <netTcpBinding>
        <binding name="ServiceBindingConfiguration">
          <security mode="Message">
            <message algorithmSuite="TripleDesSha256" />
          </security>
        </binding>
      </netTcpBinding>
    </bindings>
    <services>
      <service name="AritmetikLib.Aritmetik">
        <endpoint
address="net.tcp://localhost:9002/AritmetikServisi" bindingConfiguration="ServiceBind
ingConfiguration" binding="netTcpBinding" name="AritmetikServerEndPoint"
contract="AritmetikLib.IAritmetik" />
      </service>
    </services>
```

```
   </system.serviceModel>
</configuration>
```

Sırada istemci tarafı için yapılması gereken ayarlar var. Yine istemci tarafındaki konfigurasyon dosyasında, aynen servis tarafındaki konfigurasyon dosyasındakine benzer güvenlik ayarlamaların yapılması gerekmektedir. İlk olarak bir **bindingConfiguration** elementi oluşturulmalıdır. Bu elementin **security** ile ilişkili özelliklerinde mesaj seviyesi için **AlgorithmSuite** değeri servis tarafındaki ile aynı olacak şekilde **TripleDesSha256** olarak belirlenmelidir. Diğer taraftan aynı sunucu tarafında yapıldığı gibi güvenlik modu mesaj seviyesine çekilmelidir. Bu değer, konfigurasyon dosyasında security elementi içerisinde yer alan mode niteliği ile set edilmektedir ve Message olarak ayarlanmıştır. Bundan sonra oluşturulan bindingConfiguration elementi istemci tarafındaki endPoint ile bindingConfiguration niteliği yardımıyla ilişkilendirilmelidir. Bunun sonucu olarak istemci uygulama için konfigurasyon dosyasının son hali aşağıdaki gibi olacaktır.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
   <system.serviceModel>
      <bindings>
         <netTcpBinding>
            <binding name="ClientBindingConfiguration">
               <security mode="Message">
                  <message algorithmSuite="TripleDesSha256" />
               </security>
            </binding>
         </netTcpBinding>
      </bindings>
      <client>
         <endpoint address="net.tcp://localhost:9002/AritmetikServisi"
binding="netTcpBinding" bindingConfiguration="ClientBindingConfiguration" contra
ct="AritmetikServisi" name="TemelMatClientEndPoint" />
      </client>
   </system.serviceModel>
</configuration>
```

Artık istemci ve sunucu tarafı için mesaj seviyesinde güvenlik ayarları hazırdır. Ancak bunu test ederek analiz etmek gerekmektedir. Microsft Windows SDK tam bu amaç için tasarlanmış ve Windows Communication Foundation uygulamalarında istemci ile sunucu arasındaki mesaj trafiğini izlememizi sağlayan **Service Trace Viewer** isimli bir araç ile birlikte gelmektedir.

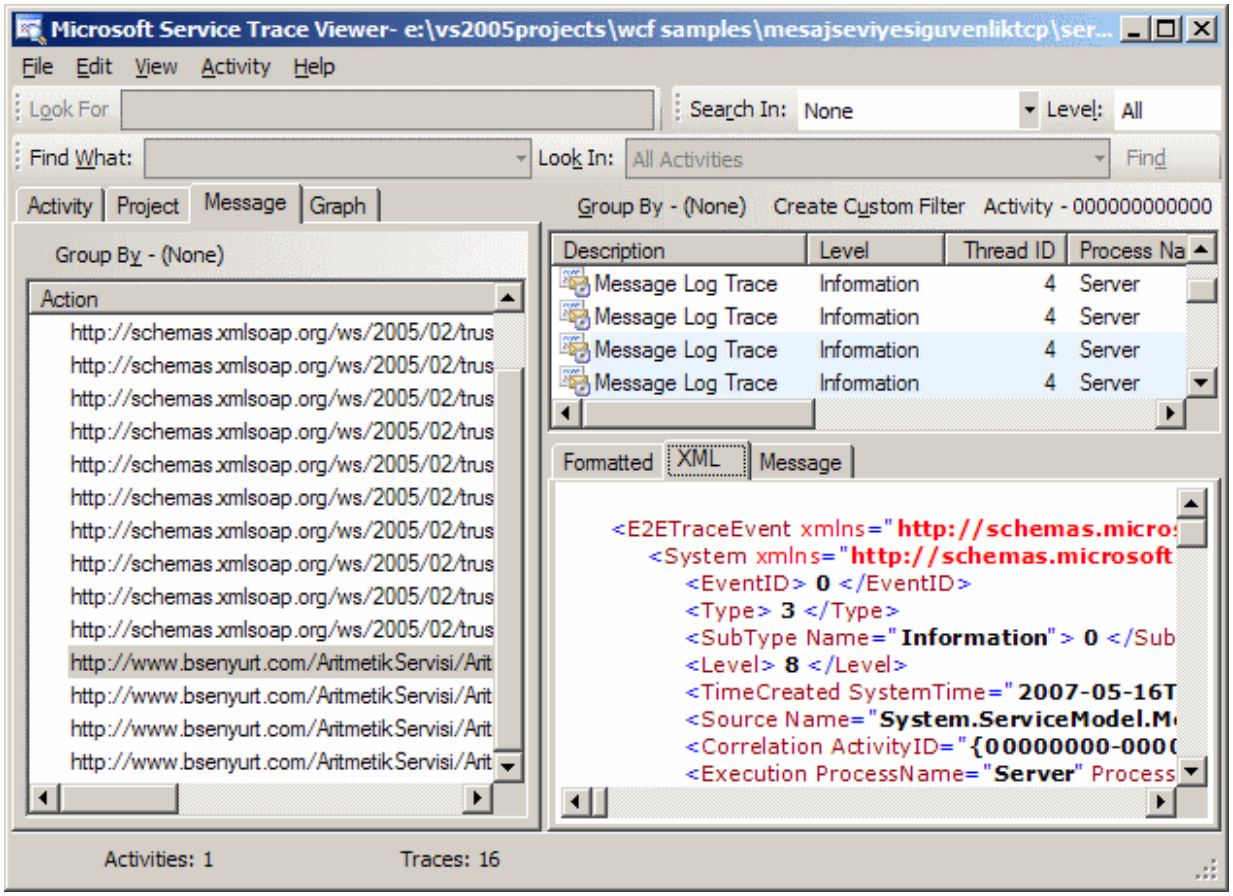
Ancak söz konusu aracın geliştirilen WCF uygulamasını izleyebilmesi içinde servis tarafında **Diagnostics** ayarlarının tesis edilmesi gerekir. Bu ayarlarıda yine Microsoft Service Configuration Editor yardımıyla kolayca gerçekleştirebiliriz. Burada söz konusu

ayarlar üzerinde şu an için çok fazla durmayacağız. öncelikli olarak amacımız servis ve istemciler arasındaki mesajlaşmayı izlemektir. İzleme**(Trace)** işlemi için yapılan ayarlardan sonra servis tarafındaki konfigurasyon dosyasının içeriği aşağıdaki gibi olacaktır. Dikkat edilmesi gereken noktalardan birisi**sharedListeners** elementi içerisinde yer alan **initializeData** niteliğinin değeridir. Burada uygulamanın yazıldığı klasör altında **svclog(Service Log)** uzantılı bir fiziki dosya bildirimi yapılmıştır. Bu bildirim Service Trace Viewer uygulaması tarafından ele alınacak dosyayı işaret etmektedir ve log bilgilerinin tamamı burada tutulmaktadır.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.diagnostics>
    <sources>
      <source name="System.ServiceModel.MessageLogging"
switchValue="Verbose,ActivityTracing">
        <listeners>
          <add type="System.Diagnostics.DefaultTraceListener" name="Default">
            <filter type="" />
          </add>
          <add name="MessageListener">
            <filter type="" />
          </add>
        </listeners>
      </source>
    </sources>
    <sharedListeners>
      <add initializeData="E:\Vs2005Projects\WCF
Samples\MesajSeviyesiGuvenlikTcp\Server\app_tracelog.svclog" type="System.Diagn
ostics.XmlWriterTraceListener, System, Version=2.0.0.0, Culture=neutral,
PublicKeyToken=b77a5c561934e089" name="MessageListener"
traceOutputOptions="None">
        <filter type="" />
      </add>
    </sharedListeners>
  </system.diagnostics>
  <system.serviceModel>
    <diagnostics>
      <messageLogging logEntireMessage="true"
logMessagesAtServiceLevel="true" logMessagesAtTransportLevel="true" />
    </diagnostics>
    <bindings>
      <netTcpBinding>
        <binding name="ServiceBindingConfiguration">
          <security mode="Message">
            <message algorithmSuite="TripleDesSha256" />
```

```
            </security>
          </binding>
        </netTcpBinding>
      </bindings>
      <services>
        <service name="AritmetikLib.Aritmetik">
          <endpoint address="net.tcp://localhost:9002/AritmetikServisi"
binding="netTcpBinding" bindingConfiguration="ServiceBindingConfiguration"
name="AritmetikServerEndPoint" contract="AritmetikLib.IAritmetik" />
        </service>
      </services>
    </system.serviceModel>
</configuration>
```

Bu noktadan sonra istemci ve sunucu uygulamalar test edilebilirler. çalışma zamanında istemci ve sunucu aktif iken Service Trace Viewer programı çalıştırılıp söz konusu svclog dosyası açılırsa ilk başta aşağıdakine benzer bir ekran ile karşılaşılacaktır.

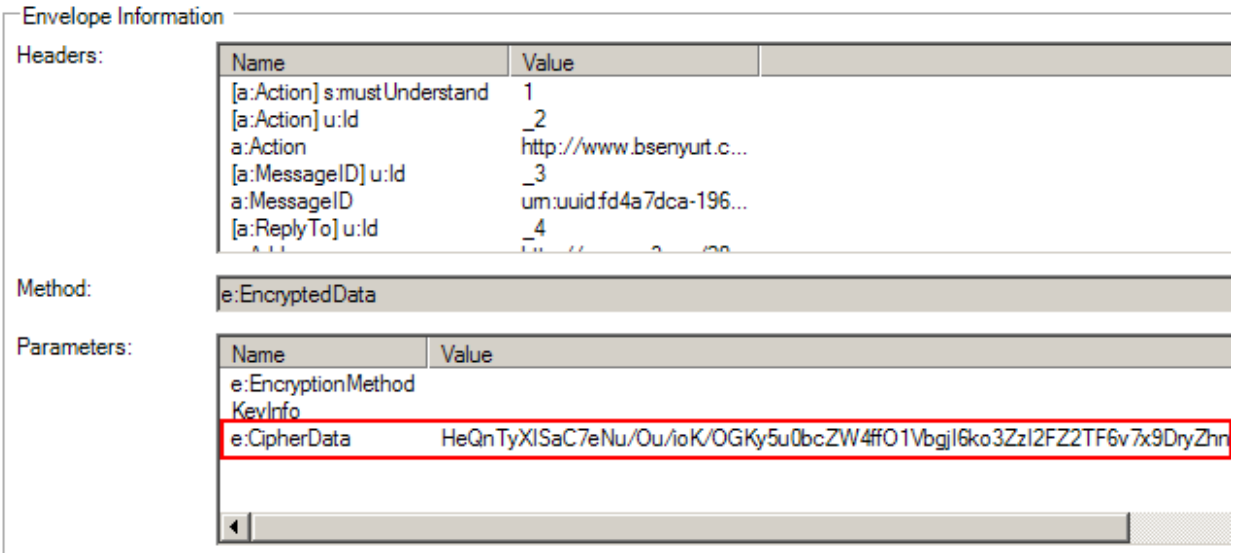


Bu ekranda özellikle dikkat edilmesi gereken kısımlar sol taraftaki **Message** sekmesinde yer alan son dört mesajdır. Bu mesajlar aslında istemciden sunucuya gelen talepleri ve sunucudan istemciye dönen cevapları içeren mesajlardır. İstemci uygulamanın kodları hatırlanacak olursa burada Toplam isimli metoda bir çağır yapılmaktadır. Topla metodu

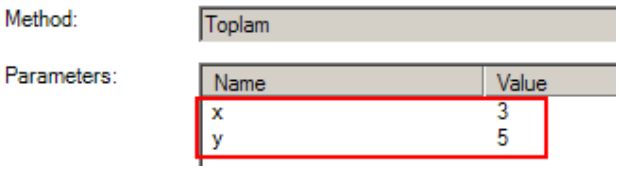
servis tarafında çalıştırılıp sonucu istemci tarafına gelmektedir. Dolayısıyla istemci Topla metoduna çağrı yaptığında aktarılan parametre ve diğer bilgiler servise gönderilecek, serviste ilgili fonksiyon çalıştırılacak ve sonucu istemci tarafına geri bildirilecektir.

http://www.bsenyurt.com/AritmetikServisi/AritmetikServisi/Toplam	net.tcp://localhost:9002/AritmetikServisi
http://www.bsenyurt.com/AritmetikServisi/AritmetikServisi/Toplam	net.tcp://localhost:9002/AritmetikServisi
http://www.bsenyurt.com/AritmetikServisi/AritmetikServisi/ToplamResponse	
http://www.bsenyurt.com/AritmetikServisi/AritmetikServisi/ToplamResponse	http://www.w3.org/2005/08/addressing/anonymous

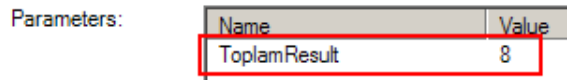
Bu mesajlardan ilki istemciden sunucya gelen bilginin şifreli olarak nasıl geldiğini göstermektedir. Şifrelenmiş veriyi görmek için sağ alt tarafta yer alan **Formatter**, **XML** yada **Message** kısımları kullanılabilir. Formatter kısmına baktığımızda **Envelope Information** bölümündeki **e:ChiperData** kısmında, istemciden sunucuya gelen parametrelerin **TripleDesSha256** algoritmasına göre şifrelenmiş halinin yer aldığı görülebilir.



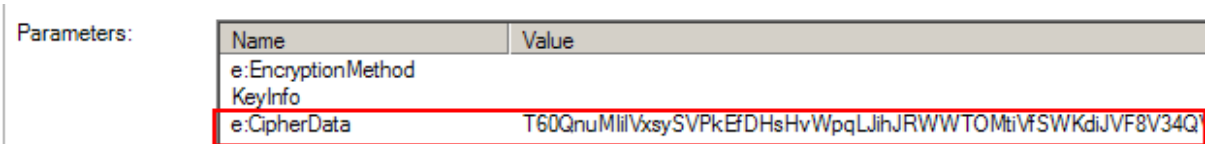
Eğer takip eden mesaj için aynı kısma tekrar bakarsak verinin servis tarafından çözümlenmiş olan hali görülmektedir. Dikkat edilecek olursa söz konusu değerler, istemcideki Toplam metodunun aldığı parametre içerikleridir.



Bir sonraki mesaj servisin bu metod çağırısına karşılık vereceği cevap hakkında bilgiler içermektedir. Yine **Parameters** kısmına bakılacak olursa aşağıdaki ekran görüntüsünde olduğu gibi 8 değerinin yer aldığını farkedilir.



Ancak son mesaj istemciye gönderilecek verinin şifrelenmiş halini işaret etmektedir. Bir başka deyişle, 8 değeri aslında servisten istemciye gönderilmeden önce TripleDesSha256 modeline göre şifrelenecek ondan sonra iletilecektir.



Eğer aynı örnek güvenlik ayarları yapılmadan test edilirse ve yine **Service Trace Viewer** yardımıyla istemci ile sunucu arasındaki mesajlaşmalar izlenirse verilerin herhangibir şekilde şifrelenmediği görülebilir. Bu amaçla istemci ve sunucu tarafındaki konfigurasyon dosyalarında security elementlerinin içerisinde yer alan **Mode** niteliklerinin değerini **None** olarak belirlemek yeterlidir.

Bu makalemizde özellikle mesaj seviyesinde güvenliği nasıl sağlayabileceğimizi incelemeye çalıştık. İlerleyen makalelerimizde windows tabanlı ve iletişim seviyesinde güvenlik işlemlerini nasıl yapabileceğimizi incelemeye devam edeceğiz. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

örnek Uygulama İçin Tıklayınız.

WCF - Windows ve Windows Service Hosting (2007-05-04T20:20:00)

wcf,

Windows Communication Foundation ile ilişikili önceki makalelerimizde mimarinin temellerinden ve bir WCF sevisinin **Internet Information Services (IIS)** üzerinden nasıl yayımlanabileceğini incelemiştik. Bu makalemizde ise Host uygulama olarak windows uygulamalarını ve windows servislerini ele almaya çalışacağız. Geliştireceğimiz uygulamalarda öncekilerden farklı olarak konfigurasyon dosyalarını kullanmayıp, programatik kod parçalarından yararlanacağız. Ayrıca, HTTP yerine **TCP** protokolü üzerinden haberleşmeyi kullanıyor olacağız.

Bildiğiniz gibi WCF mimarisinde sözleşmeleri(**Contracts**) sunan uygulamalar, sadece IIS üzerinden yayınlanmak zorunda değildir. Bu şablon dışında windows uygulamalarını (Hatta **WPF**uygulamalarını) ve windows servislerinide kullanabiliriz. Özellikle windows servisleri (**Windows Service**), bulundukları makine üzerinde otomatik olarak başlatılabildiklerinden, yönetilebilirlikleri daha kolaydır ve özellikle windows tabanlı

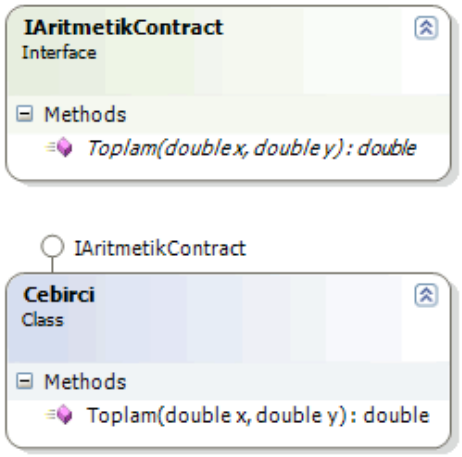
intranet sistemlerinde tercih edilirler. Özellikle .Net Remoting üzerine yazılan sistemlerde windows servisleri yaygın olarak kullanılmaktadır.

WCF servislerinin ABC' sinden (**AddressBindingContract**) hatırlayacağınız üzere bağlayıcılar (Binding) bizim için pek çok zorluğun üstesinden gelmektedir. Daha önceki makalelerimizde geliştirdiğimiz WCF örnek uygulamalarında sadece **BasicHttpBinding** tipinden faydalandık. Oysaki projenin ihtiyacına ve şartlara göre diğer bağlayıcı tipleride göz önüne alabilir ve kullanabiliriz. Bu noktada bağlayıcı tipleri **(Binding Types)** biraz daha tanımakta fayda olacağı kanısındayım. Bağlayıcı tipler temel olarak istemcilerin, bir servis ile iletişime nasıl geçebileceği konusunda bazı esasları otomatik olarak belirlemekte rol alan varlıklar olarak tanımlanabilir. Söz gelimi, istemcilerden kabul edilebilecek mesajların neler olacağını bağlayıcı tipler yardımıyla belirleyebiliriz. İstemciler, servislere erişirken çeşitli iletişim protokollerini kullanacaktır. Örneğin HTTP veya TCP. HTTP intranet dışında internet üzerindende kullanılabilecek bir ortam sağlarken, TCP özellikle intranet ortamında maksimum performansı sağlayacak nitelikte bir alt yapı sağlamaktadır. Bağlayıcı tipler bu protokolün ne olacağının belirlenmesinde etkin rol oynar. Söz gelimi **BasicHttpBinding** tipi HTTP protokolü üzerinden iletişime geçilmesi için gerekli olan alt yapıyı sağlamaktadır.

NOT : *Bazı WCF servis uygulamalarında, birden fazla* ***endPoint*** *tanımlaması sıklıkla uygulanan bir tekniktir. Söz gelimi intranet tabanlı sistemler için TCP tabanlı bir bağlayıcı tipi(Binding) içerecek bir endPoint ile birlikte, internet üzerinden gelen isteklere yönelik olarak , HTTP tabanlı ve* ***XML*** *formatında çözümleme yapabilecek bir bağlayıcı tipi kullanacak****endPoint*** *tanımlamaları bir arada ele alınıp kullanılabilir.*

Diğer taraftan istemciler ve servis arasında taşınacak olan verinin nasıl formatlanacağınada bağlayıcı tipler karar vermektedir. Burada söz konusu olan formatlama, verinin **XML** olarak mı yoksa **binary** olarak mı serileştirileceğidir. Yine binary formatta serileştirme, özellikle intranet gibi sistemlerde hız ve performans sağlayacaktır. Özellikle image gibi büyük boyutlu veri yapılarının binary formatta transfer edilmesi performans açısından önemli bir kriterdir. Bağlayıcıların etkin olarak rol aldıkları diğer noktalar servisin güvenilirliğinin nasıl sağlanacağı**(reliability)** ve servisin **transaction** içeren bir operasyonda görev alıp almayacağıdır. Bu ve benzeri kriterler göz önüne alındığında, **System.ServiceModel** isim alanı altında yer alan önceden tanımlı bağlayıcı tipler hazır çözümler sunmaktadır. Bu elbetteki kendi bağlayıcı tiplerimizi yazamayacağımız anlamına gelmemelidir. Dilersek bunuda yapabiliriz. Örneklerimizde TCP prokolünü ve binary serileştirmeyi tercih edeceğimizden, bununla ilgili olan bağlayıcı tipi kullanabiliriz.

Bu kısa bilgilerden sonra dilerseniz örneklerimizi adım adım geliştirmeye başlayalım. İlk olarak servis sözleşmemizi **(Service Contract)** ve bu sözleşmeyi uygulayacak tipimizi yazmamız gerekiyor. Amacımız kod tarafında servisi tesis etmek ve kullanmak olduğundan mümkün olduğu kadar basit bir sözleşme ve tip tanımlaması yapacağız. Bu amaçla aşağıdaki tipleri içerecek AritmetikLib isimli bir sınıf kütüphanesi **(class library)** geliştirerek işe başlayabiliriz.



Servis Sözleşmemiz;

```
using System;
using System.ServiceModel;

namespace AritmetikLib
{
   [ServiceContract(Name="CebirciServisi",
        Namespace="http://www.bsenyurt.com/CebirciServisi")]
   public interface IAritmetikContract
   {
      [OperationContract]
      double Toplam(double x, double y);
   }
}
```

Servis sözleşmemiz herzamanki gibi bir arayüzdür(**Interface**). Diğer tarafan istemcilere sunulabilecek olan operasyonumuz basit olarak bir Toplama işlemi yapmaktadır. Özel olarak **ServiceContract** isimli niteliğimizin **Name** ve **Namespace** özelliklerine bazı değerler atadık. Name ve Namespace özellikleri, sözleşmeye ait **WSDL** (Web Service Description Language) elementi içerisindeki ilgili değerleri belirlemekte kullanılmaktadır ve daha sonradan bu sözleşmenin istemciler için gerekli olan proxy sınıfını üretilirken ele alınacaktır.

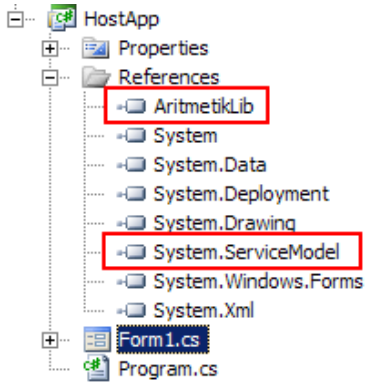
Sözleşmeyi uyarlayan tipimiz;

```
using System;

namespace AritmetikLib
{
   public class Cebirci:IAritmetikContract
   {
```

```
        #region IAritmetikContract Members
        public double Toplam(double x, double y)
        {
            return x + y;
        }
        #endregion
    }
}
```

Cebirci isimli sınıfımız basit olarak servis sözleşmesi olan **IAritmetikContract** isimli arayüzü (interface) implemente etmektedir. Servis sözleşmemizi ve tipimizi tanımladıktan sonra artık Windows tabanlı Host uygulamamızı geliştirmeye başalayabiliriz. Söz konusu uygulamamızın ön yüzünü aşağıdaki gibi tasarlayabiliriz.

Başlat başlıklı düğmeye basıldığında, windows uygulamamız **ServiceHost** sınıfına ait bir nesneyi örnekleyerek istemcilerden(**Clients**) gelecek olan talepleri dinlemeye başlayacaktır. Tam tersine Durdur başlıklı düpmeye bastığımızda ServiceHost nesne örneği ile açılan kanallar kapatılacak ve Host uygulama artık istemcilere cevap vermeyecektir. Bu işlemler sırasında servisin durumuda label kontrolü içerisinde gösterilecektir. Başlamadan önce WCF tiplerini uygulamamız içerisinde ele alabilmek ve az önce geliştirdiğimiz servis sözleşmesi ve tipimizi kullanabilmek için sırasıyla **System.ServiceModel.dll** ve AritmetikLib.dll isimli assemblylerımızı uygulamamıza referans etmemiz gerekmektedir.



Artık uygulamamızın kodlarını aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.ServiceModel;
using System.Windows.Forms;
using AritmetikLib;

namespace HostApp
{
   public partial class Form1 : Form
   {
      private ServiceHost srvHost;

      public Form1()
      {
         InitializeComponent();
      }

      private void btnBaslat_Click(object sender, EventArgs e)
      {
         srvHost = new ServiceHost(typeof(Cebirci));
         srvHost.AddServiceEndpoint(typeof(IAritmetikContract), new
NetTcpBinding(), "net.tcp://localhost:4501/CebirciServisi");
         srvHost.Open();
         lblServisDurumu.Text = srvHost.State.ToString();
         btnBaslat.Enabled = false;
         btnDurdur.Enabled = true;
      }

      private void btnDurdur_Click(object sender, EventArgs e)
      {
         srvHost.Close();
         lblServisDurumu.Text = srvHost.State.ToString();
         btnDurdur.Enabled = false;
         btnBaslat.Enabled = true;
      }
   }
}
```

btnBaslat düğmesine ait Click olay metodu içerisinde ilk olarak **ServiceHost** nesnemiz örneklenmektedir. Yapıcı metoda(Constructor), Cebirci sınıfın tipi verilmiştir. Bir başka deyişle servis sözleşmesini uyarlayan tip bildirilmektedir. Servislerin, istemcilerden gelen talepleri **endPoint**' ler yardımıyla aldığını biliyoruz. Bir endPoint içerisinde **adres** , **bağlayıcı tip** ve **sözleşme**bilgilerininde yer alması gerekiyor. Bu nedenle, ServiceHost sınıfımıza ait nesne örneğimize **AddServiceEndPoint** metodu

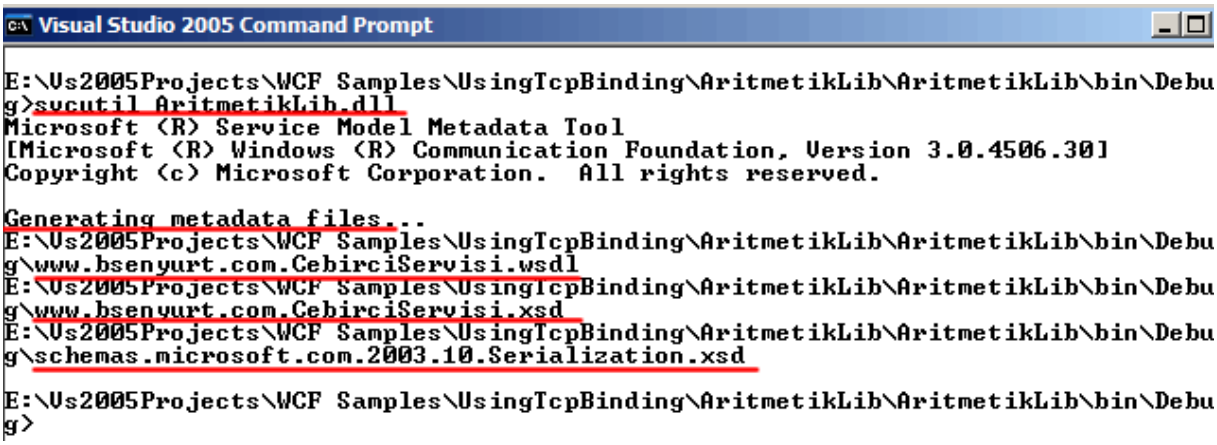
yardımıyla bir endPoint eklemekteyiz. Bu metodun ilk parametresi servis sözleşmemize ait tipi temsil etmektedir. İkinci parametrede dikkat ederseniz **NetTcpBinding** tipinden bir nesne örneği verilmektedir. Birbaşka deyişle TCP protokolü üzerinden, binary formatta çözümleme kullanılarak iletişime geçileceğini bildirmiş oluyoruz. Son parametre ise servisimizdeki bu endPoint' e nasıl ulaşılabileceği bilgisini içermektedir. Bu parametrede makine adı, iletişim protokolü, port numarası ve nesne takma adı gibi bilgiler yer almaktadır.

Bu adımlardan sonra tek yapılması gereken servisin dinlemeye başlamasını sağlamaktır. Bu amaçla ServiceHost tipinden nesne örneğimizin **Open** metodunu kullanıyoruz. btnDurdur isimli düğmemize ait Click olay metodunda ise **Close** metodu çağırılarak servisin kapatılması sağlanmaktadır. İstemciyi yazmadan önce windows uygulamasını başlatarak test edebilirsiniz. Eğer sistemde yüklü bir Firewall var ise, 4501 numaralı portun açılmasına karşın bir uyarı mesajı verecektir. Bunu kabul ederek devam etmek gerekmektedir. Aksi takdirde portun kullanılması engelleneceğinden, istemcilere cevap verilemez ve servis çalışmaz.

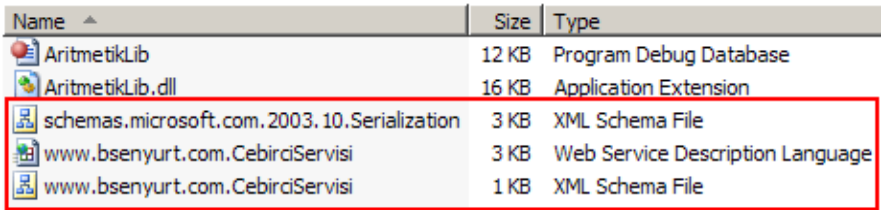
NOT : *Bir Host uygulama herhangibir iletişim portunu kullanarak hizmet vermeye başladığında, başka bir Host uygulama aynı makine üzerindeki aynı iletişim portunu kullanamaz. Böyle bir durumuda çalışma zamanı istisnası (Runtime Exception) alınacaktır.*

Artık istemci tarafını programlayabiliriz. Bildiğiniz gibi WCF sisteminde yer alan istemcilerimiz, servislere ait endPoint' ler ile haberleşirken **proxy** sınıflarını kullanmaktadırlar. Şuanki senaryomuzda servisimizi IIS üzerinden host etmediğimiz için HTTP tabanlı olacak şekilde metadata publishing yapamamaktayız. Ancak **svcutil.exe** aracı ile AritmetikLib isimli dll' imizden yararlanarak istemci için gereken proxy sınıfı ve konfigurasyon dosyalarını ürettirebiliriz. Bu amaçla ArtimetikLib.dll ' ini herhangibir yerden svcutil aracı ile aşağıdaki gibi ele almalıyız.

svcUtil AritmetikLib.dll

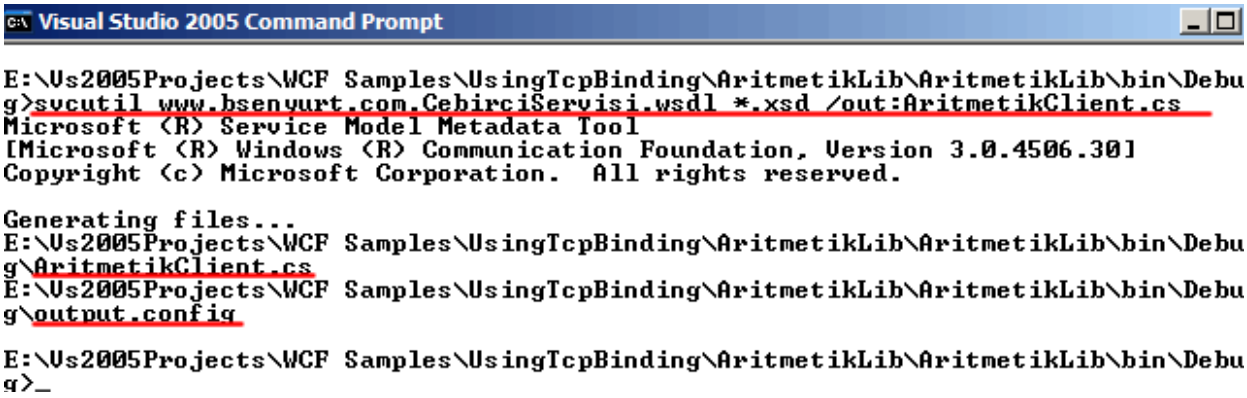


SvcUtil aracını bu şekilde kullandığımızda library içerisinde yer alan servis sözleşmesi ve tiplerden yararlanılaraktan bir takım metadata ve şema dosyaları üretilecektir.

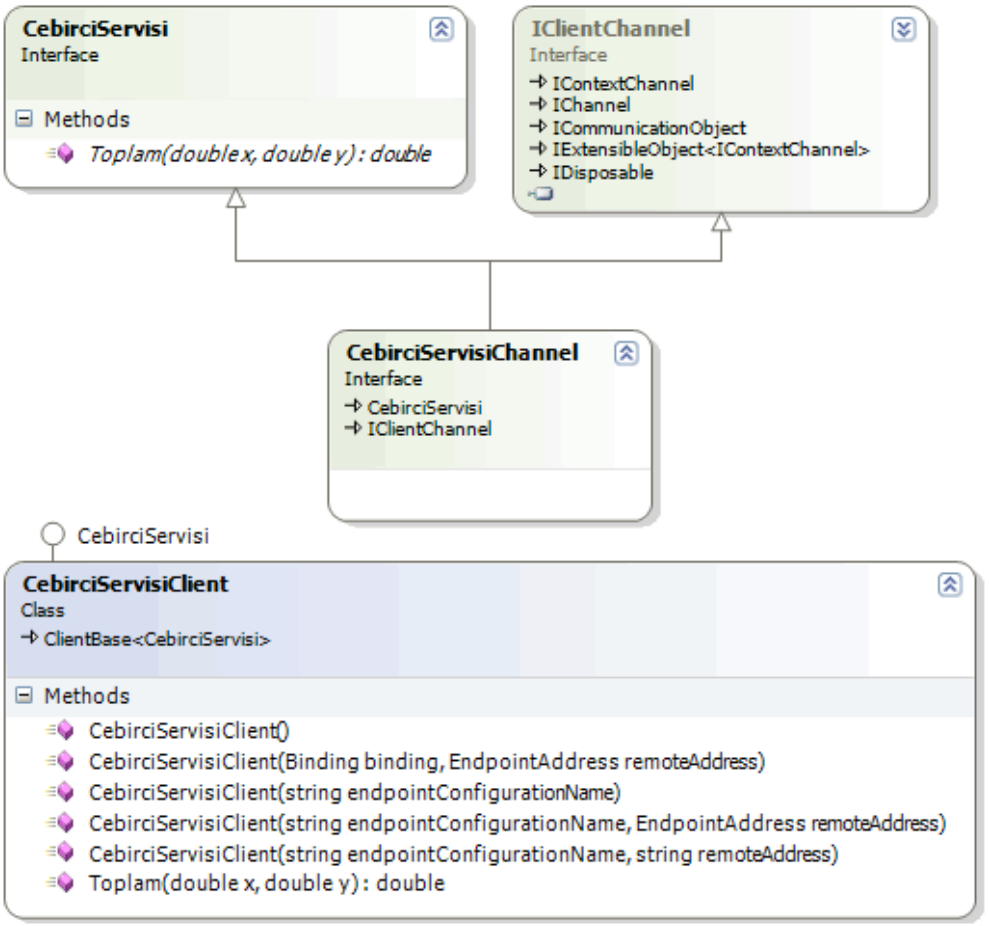


Çok basit olarak düşünecek olursak, bu dosyaların istemciler için gereken proxy sınıflarını üretmek amacıyla kullanılabileceğini düşünebiliriz. Sonuç itibariyle IIS üzerinden yapılan Host işlemlerindede bir **WSDL** dökümanı gerekmektedir ki burada bu işi **www.bsenyurt.com.CebirciServisi.wsdl** isimli dosya üstlenmektedir. Bu işlemin ardından yine svcutil aracını aşağıdaki gibi kullanarak istemciler için gerekli proxy sınıfı ve konfigurasyon dosyasını üretebiliriz. *(Buradaki dosya isimlerinin tesadüf olmadığını söyleyelim. Hatırlarsanız**ServiceContract** niteliğimizin Namespace özelliğine http://www.bsenyurt.com/CebirciServisi değerini vermiştik.)*

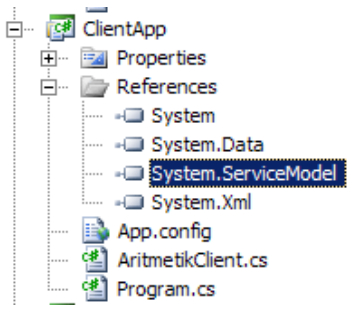
svcutil www.bsenyurt.com.CebirciServisi.**wsdl *.xsd /out:**AritmetikClient.cs



Komut başarılı bir şekilde çalıştığı takdirde istemciler için AritmetikClient.cs ve output.config isimli fiziki dosyalar üretilecektir. Oluşturulan proxy dosyası içerisinde CebirciServisi, CebirciServisiChannel isimli iki arayüz yer almaktadır. İstemci tarafından örneklenerek kullanılabilecek olan CebirciServisiClient isimli sınıf CebirciServisi isimli arayüzü uygulamakatadır. Servis ile iletişimde önemli bir rol üstelenen kanalı temsil eden CebirciServisiChannel isimli tipimiz ise hem CebirciServisini hemde **IClientChannel** arayüzünü uygulamaktadır. IClientChannel arayüzü temel olarak istemciler için taban arayüzdür(base interface) ve WCF iletişimi için gerekli çalışma zamanı fonkisyonellikleri sağlamaktadır. Örneğin kanalın çalışma zamanında açılması veya dispose edilmesi gibi işlemlerin yapılmasını sağlayan fonksiyonellikler içerir. *(Bunun gibi mimari detayları ilerleyen makalelerimizde ele almaya çalışacağız.)*



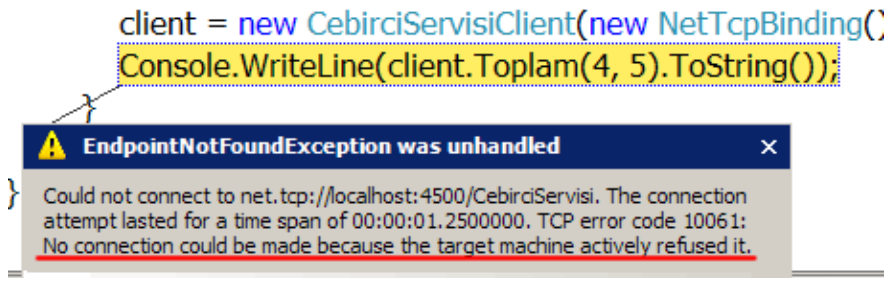
Artık istemci uygulamamızı geliştirmeye başlayabilir ve az önce üretilen proxy tipimizi burada kullanarak sunucu ile iletişime geçebiliriz. İşlemlerimizi basit bir şekilde ele almak amacıyla istemcimizi sıradan bir Console uygulaması olarak tasarlayacağız. Uygulamamızın yine **System.ServiceModel** assembly' ını referans etmesi ve komut satırından **SvcUtil** aracı yardımıyla ürettiğimiz proxy sınıfını içermesi gerekmektedir.



Output.config dosyasını istemci uygulamaya eklemek zorunda değiliz. Şu aşamada istemci tarafından servisi çağırmak için gerekli ayarları programatik olarak yapıyor olacağız. İlgili referans ve proxy tipini ekledikten sonra kodlarımızı aşağıdaki gibi geliştirebiliriz.

```
using System;
using System.ServiceModel;

namespace ClientApp
{
   class Program
   {
      static void Main(string[] args)
      {
         Console.WriteLine("Devam etmek için tuşa basın");
         Console.ReadLine();
         CebirciServisiClient client;
         client = new CebirciServisiClient(new NetTcpBinding(), new
EndpointAddress("net.tcp://localhost:4500/CebirciServisi"));
         Console.WriteLine(client.Toplam(4, 5).ToString());
         Console.ReadLine();
      }
   }
}
```

İlk olarak CebirciServisiClient isimli sınıfımıza ait bir nesne örneği oluşturuyoruz. Bunu gerçekleştirirken yapıcı metodumuza ilk parametre olarak **NetTcpBinding** tipinden bir nesne örneği atanmaktadır. Sunucu tarafında NetTcpBinding tipini kullandığımız için istemci tarafındada uygun bir bağlayıcı tipin kullanılması gerekmektedir. İkinci parametre olarak bir**EndpointAddress** nesne örneği oluşturulmaktadır. Bu sınıfın yapıcı metodunda verdiğimiz adres ise, servis tarafında belirttiğimiz erişim adresidir. Son olarak uzak nesne metodumuzu çalıştırmaktayız. Artık istemcimizi test edebiliriz. Elbetteki önce sunucu uygulamanın çalıştırılması ve servisin açılması gerekmektedir. Eğer sunucu uygulama üzerinden servis açılmadan istemci çalıştırılırsa aşağıdaki gibi çalışma zamanı hatası(Runtime Exception) alırız.



Exception mesajı özellikle .Net Remoting uygulamaları geliştirenler için tanıdık bir cümle içermektedir. **No Connection could be made because the target machine actively refused it.** :) Bu aynı zamanda istemcimizin gerçekten belirtilen adreste kendisini dinleyecek bir WCF servis uygulaması aradığınında bir göstergesidir. Aşağıdaki flash animasyonunda basit olarak sistemimizin nasıl çalıştığı gösterilmektedir

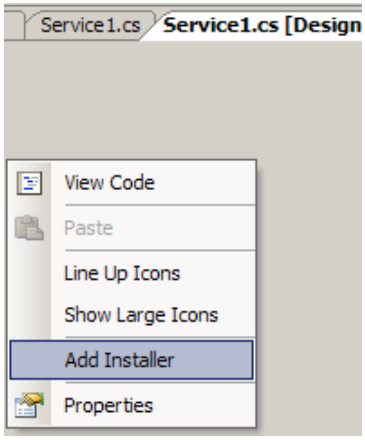
Gelelim Windows uygulamasından sunmuş olduğumuz hizmeti bir windows servisi üzerinden nasıl sunacağımıza. Aslında WCF tarafından baktığımızda sadece windows servisi ortamına ayak uydurmamız yeterli olacaktır. Gelin örneğimiz üzerinden devam edelim. İlk olarak solution' ımıza bir **Windows Service** projesi eklememiz gerekiyor. Projemizi oluşturduktan sonra, her zaman olduğu gibi **System.ServiceModel.dll** isimli assembly' ın ve servis sözleşemesi ile uzak nesne sınıfını barındıran sınıf kütüphanemizin (**AritmetikLib.dll**) referans edilmesi gerekmektedir.

Windows servisleri, web servislerine benzer olarak herhangibir görsel arabirime sahip değildir. İş yapan metodları, olay tabanlı çalışmaktadır. Dolayısıyla ilgili servis, windows işletim sisteminin servislerine eklendikten sonra, başlatılması, durudurulması yada hata oluşması gibi durumlarda yapılması gereken kodlar ilgili olay metodlarına yazılmaktadır. Bizim windows servisimiz başlatıldığında uzak nesneleri istemci uygulamaların hizmetine sunacak şekilde dinlemede olmalıdır. Diğer taraftan servis kapatıldığında ise, Host uygulamanın kullandığı serviceHost nesne örneği artık istemcilere cevap vermeyecek şekilde kapatılmalıdır. Bu duruma göre servis sınıfımızın kodları aşağıdaki gibi olmalıdır.

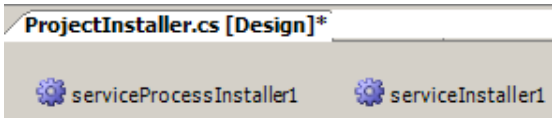
```
using System;
using System.Collections.Generic;
using System.ServiceProcess;
using System.ServiceModel;
using AritmetikLib;

namespace AritmetikWinServis
{
   public partial class Service1 : ServiceBase
   {
      private ServiceHost srvHost;

      public Service1()
      {
         InitializeComponent();
      }

      protected override void OnStart(string[] args)
      {
         srvHost = new ServiceHost(typeof(Cebirci));
         srvHost.AddServiceEndpoint(typeof(IAritmetikContract), new
NetTcpBinding(), "net.tcp://localhost:65001/CebirciServisi");
         srvHost.Open();
      }

      protected override void OnStop()
      {
         srvHost.Close();
```

```
        }
    }
}
```

Dikkat ederseniz makalemizin başında geliştirdiğimiz Windows uygulmasındaki kodlarımızdan çokda farklı bir işlem gerçekleştirmiyoruz. Servis **OnStart** olay metodu içerisinde oluşturulurken, endPoint' in eklenmesi ve açılmasıda burada yapılmaktadır. Benzer şekilde **OnStop** olay metodu içerisinde servisimiz kapatılmaktadır. Bundan sonraki adımlarımızda servisimizi windows sistemine **install** etmek için gerekli işlemleri yapmamız gerekmektedir. Söz konusu adımlar makalemizin konusu dışında olduğundan yüzeysel olarak ele alınacaktır. Öncelikli olarak servisimize **Add Installer** seçeneği ile gereken yükleyici tipleri ekliyoruz.



Bu işlemin sonucunda servise **ServiceProcessInstaller** ve **ServiceInstaller** tiplerinden iki adet bileşen eklenecektir.

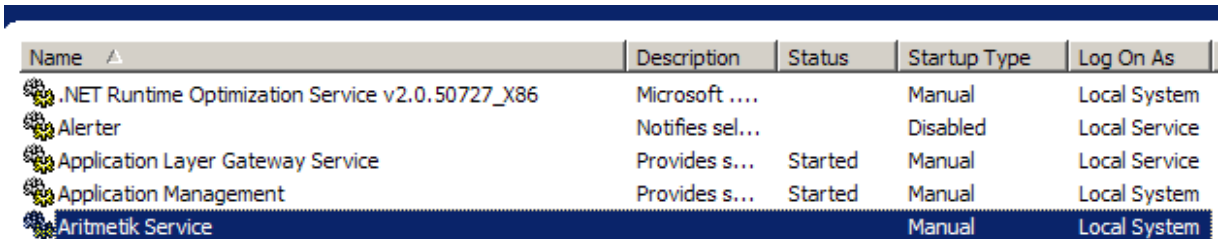


ServiceProcessInstaller bileşeninin üyelerinden **Account** özelliğinin değerini **Local System** olarak ayarlayalım. ServiceInstaller bileşenimizin, **Display Name** özelliğine Aritmetik Servis değerini atayalım. Bu sistem servislerine baktığımızda, geliştirdiğimiz windows servisini bulmamızı kolaylaştıracaktır. Uygulamamız derlendikten sonra işletim sistemine yüklenmeye hazır hale gelecektir. Yükleme işlemi için Visual Studio 2005 Command Prompt üzerindeyken **installUtil** aracını aşağıdaki gibi -i parametresi ile kullanmamız gerekmektedir.

```
Visual Studio 2005 Command Prompt

E:\Vs2005Projects\WCF Samples\UsingTcpBinding\AritmetikWinServis\bin\Debug>installutil -i AritmetikWinServis.exe
Microsoft (R) .NET Framework Installation utility Version 2.0.50727.42
Copyright (c) Microsoft Corporation.  All rights reserved.


Running a transacted installation.

Beginning the Install phase of the installation.
See the contents of the log file for the E:\Vs2005Projects\WCF Samples\UsingTcpB
inding\AritmetikWinServis\bin\Debug\AritmetikWinServis.exe assembly's progress.
The file is located at E:\Vs2005Projects\WCF Samples\UsingTcpBinding\AritmetikWi
nServis\bin\Debug\AritmetikWinServis.InstallLog.
Installing assembly 'E:\Vs2005Projects\WCF Samples\UsingTcpBinding\AritmetikWinS
ervis\bin\Debug\AritmetikWinServis.exe'.
Affected parameters are:
   logtoconsole =
   assemblypath = E:\Vs2005Projects\WCF Samples\UsingTcpBinding\AritmetikWinServ
is\bin\Debug\AritmetikWinServis.exe
   i =
   logfile = E:\Vs2005Projects\WCF Samples\UsingTcpBinding\AritmetikWinServis\bi
n\Debug\AritmetikWinServis.InstallLog
Installing service Service1...
Service Service1 has been successfully installed.
Creating EventLog source Service1 in log Application...

The Install phase completed successfully, and the Commit phase is beginning.
See the contents of the log file for the E:\Vs2005Projects\WCF Samples\UsingTcpB
inding\AritmetikWinServis\bin\Debug\AritmetikWinServis.exe assembly's progress.
The file is located at E:\Vs2005Projects\WCF Samples\UsingTcpBinding\AritmetikWi
nServis\bin\Debug\AritmetikWinServis.InstallLog.
Committing assembly 'E:\Vs2005Projects\WCF Samples\UsingTcpBinding\AritmetikWinS
ervis\bin\Debug\AritmetikWinServis.exe'.
Affected parameters are:
   logtoconsole =
   assemblypath = E:\Vs2005Projects\WCF Samples\UsingTcpBinding\AritmetikWinServ
is\bin\Debug\AritmetikWinServis.exe
   i =
   logfile = E:\Vs2005Projects\WCF Samples\UsingTcpBinding\AritmetikWinServis\bi
n\Debug\AritmetikWinServis.InstallLog

The Commit phase completed successfully.

The transacted install has completed.

E:\Vs2005Projects\WCF Samples\UsingTcpBinding\AritmetikWinServis\bin\Debug>
```

InstallUtil aracı işlemleri başarılı bir şekilde tamamladıktan sonra servislerde, eklemiş olduğumuz windows servisi görülebilir.

NOT : *Sisteme yüklediğimiz servisi kaldımak için* ***installUtil*** *aracının* ***-u*** *parametresini kullanmak gerekmektedir.*



Servisimizi buradan çalıştırdığımızda artık istemciler tarafından kullanılabilir hale gelecektir. İstemci tarafında tek yapmamız gereken **port** bilgisini değiştirmek olacaktır.

Örneğin bu makalede geliştirdiğimiz Console uygulamasının kodlarını aşağıdaki gibi değiştirmek yeterlidir.

```
CebirciServisiClient client;
client = new CebirciServisiClient(new NetTcpBinding(), new
EndpointAddress("net.tcp://localhost:65001/CebirciServisi"));
Console.WriteLine(client.Toplam(4, 5).ToString());
```

Mimarinin yine doğru bir şekilde tesis edildiğini kontrol etmek adına, geliştirilen servis kapalı iken istemci uygulama çalıştırılabilir. Eğer **"No Connection could be made because the target machine actively refused it"** hatası alınabiliyorsa, istemcinin gerçektende sunucu servisi aradığını anlayabiliriz. Elbette servis çalıştırıldıktan sonrada istemciyi test edip herşeyin yolunda gittiğinden emin olmakta fayda vardır. Aşağıdaki Flash animasyonunda bu durum gösterilmektedir.

Bu makalemizde WCF mimarisinde yer alan Host uygulama modellerinden **Windows** ve **Windows Servislerini** kısaca incelemeye çalıştık. Aynı zamanda, konfigurasyon dosyası kullanımı yerine kod tarafında neler yapabileceğimize değindik. Özellikle TCP tabanlı bir binding tipi kullandığımızda**(NetTcpBinding)**, bir başka deyişle HTTP üzerinden **Metadata** yayınlanması söz konusu olmadığında, istemciler için gerekli olan proxy sınıflarının **svcutil** aracı yardımıyla nasıl kolay bir şekilde üretilebileceğine bakmaya çalıştık ve böylece bir makalemizin daha sonuna geldik. İlerleyen makalelerimizde WCF mimarisinin detaylarını incelemeye devam ediyor olacağız. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Örnek Uygulama İçin Tıklayınız.

WCF - Adım Adım IIS Hosting (2007-04-28T20:29:00)

wcf,

Windows Communication Foundation ile ilişikili bir önceki makalemizde mimarinin detaylarına yakından bakmaya çalışmış ve örnek bir uygulama geliştirmiştik. Bu makalemizde ise bir **WCF Service** projesi geliştirmeye çalışacağız. Bir başka deyişle **HTTP** protokolünü kullanarak, **IIS** üzerinden bir .Net kütüphanesini servis olarak yayınlayacağız.

NOT : *WCF servislerini **Self Hosting** ve **IIS Hosting** olmak üzere iki farklı şekilde yayınlayabiliyoruz. Bununla birlikte Self Hosting modeli kendi içerisinde **Console**, **Windows**, **Windows Service** ve Vista ile gelen **Windows Activation Service (WAS)** olmak üzere dört farklı seçenekten oluşmaktadır.*

Ayrıca kendi tiplerimizi döndüren metodlar söz konusu olduğunda veri sözleşmelerini(**DataContract**) nasıl kullanabileceğimize yakından bakma fırsatı bulacağız. Bunlara ek olarak özellikle **.Net Remoting** mimarisinde bizi her zaman zorlayan

konfigurasyon ayarlarının*(Çoğu zaman ezberlemek zorunda olduğumuz ve hatırlamakta güçlük çektiğimiz)*, WCF içerisinde **Edit WCF Configuration** seçeneği yardımıyla Visual Studio 2005 içerisinden görsel olarak nasıl daha kolay hazırlanabileceğinide incelemeye çalışacağız.

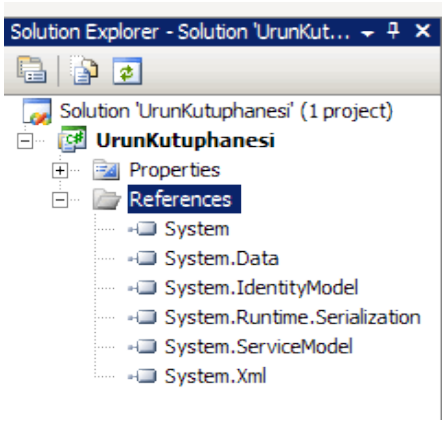
Aslında IIS(Internet Informatin Service) üzerinden çeşitli tipte istemcilerin (hangi platformda olduklarına bakılmaksızın) belirli fonksiyonellikleri kullanabilmesi amacıyla **Xml Web Servisleri** (Xml Web Services) oldukça fazla kullanılan bir **SOA (Service Oriented Architecture)** modelidir. Dolayısıyla WCF içerisinde de buna benzer bir model vardır ve **WCF Service** olarak adlandırılmaktadır. Visual Studio 2005 için, Framework 3.0' a yönelik eklentileri yüklediğinizde Web Site şablonları arasında **WCF Service** seçeneğide gelmektedir. Ancak biz bunu kullanmayıp biraz daha elle ilerleyeceğiz ve alt yapı içerisinde yer alan parçaları daha net olarak görmeye çalışacağız. Konuyu daha iyi kavrayabilmek için örnek bir senaryo üzerinden gideceğiz ve bazı basit veritabanı işlemlerini ele alacağız. Gelin hiç vakit kaybetmeden işe başlayalım ve adım adım ilerleyelim.

Bildiğiniz gibi WCF tamamen nitelik(**attribute**) tabanlı bir mimariye dayanmaktadır. Bununla birlikte istemci tarafına yayınlanacak olan, başka bir deyişle **metadata** içeriği gönderilecek olan tip(type) aslında ilgili nitelikler ile imzalanmış bir sözleşme(**contract**) dir. Sözleşmeler genellikle bir arayüz şeklinde tasarlanmakta olup asıl işlevsellikleri yerine getirecek olan uzak nesne modellerinin(sınıflar) türetilmesinde de kullanılır.

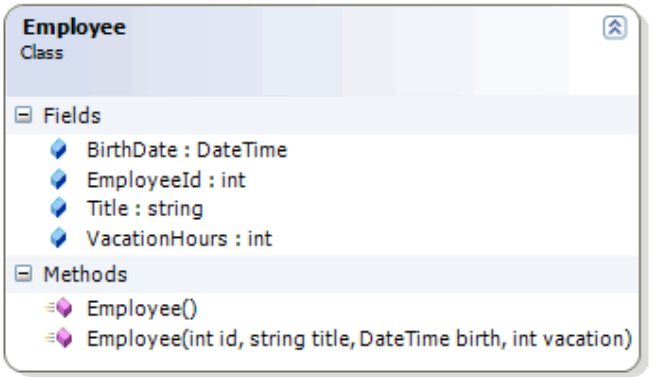
NOT : *.Net Remoting mimarisinden hatırlarsanız eğer; uzak nesneleri* ***(Remote Objects)*** *yayınlamada ele alınan modellerden birisi arayüz(Interface) kullanımını içermektedir. Burada temel amaç istemci tarafında, sunucuda örneklenen uzak nesne referansını taşıyabilecek bir tip olmasıdır. Bununla birlikte istemci tarafının kesin olarak uzak nesne fonksiyonelliklerinin nasıl çalıştığını görmesi gibi bir şansı yoktur. (Elbette bu durum* ***MarshalByReference*** *nesneler için geçerlidir. Bildiğiniz gibi* ***MarshalByValue*** *tipler zaten sunucuda örneklenip istemci tarafına tüm varlığı ile serileştirilerek(Serialization) geçerler). Ayrıca, sunucu tarafındaki uzak nesne metodlarının iç yapılarında meydana gelecek değişiklilikler sonucu, istemcilere yeniden dağıtım(****deployement****) işlemi uygulanmasına gerek kalmamaktadır.*

Öyleyse ilk yapmamız gereken içerisinde servis sözleşmesini**(Service Contract)** ve gerekli fonksiyonelliklere ait tanımlamaları barındıracak bir arayüz**(interface)** tasarlamaktır. Örneğimizde, WCF servisimizi IIS üzerinden sunacağımızdan bahsetmiştik. Bu sebepten ilk olarak bir sınıf kütüphanesi**(class library)** geliştirmeli ve bu kütüphane içerisine servis sözleşmemizi(Service Contract), uzak nesnemizin modelini(class), veri sözleşmesine(Data Contract) sahip tipimizi katmalıyız. Sonrasında IIS üzerinden ilgili kütüphanemizi işaret edecek bir sanal klasör (virtual path) oluşturmalıyız. Yayınlama ortamı IIS olduğundan, WCF için gerekli konfigurasyon ayarlarını **web.config** dosyası içerisinde tutmamız gerekmektedir. Bu noktada konfigurasyon ayarlarını görsel bir arayüzü kullanarak kolay bir şekilde oluşturabiliriz. Diğer taraftan aynı web servislerinde olduğu gibi asmx uzantılı dosyaların görevini üstlenen svc uzantılı bir dosya oluşturmalıyız.

İlk olarak bilgisayarımızın herhangibir yerinde konuşlandırılacak bir sınıf kütüphanesi (class library) projesi oluşturalım. UrunKutuphanesi isimli projemizi yine **.Net Framework 3.0** sonrası Visual Studio 2005 için gelen eklentilerden **WCF Service Library** olarak açabiliriz. Ama bu şablon yoksada önemli değildir. Çünkü yapmamız gereken **System.ServiceModel,System.Runtime.Serialization** ve **System.IdentityModel** assembly' larını referans edecek olan bir proje geliştirmektir. *(Bu assembly' ların .Net Framework 3.0 ile geldiklerine dikkat edelim.)*



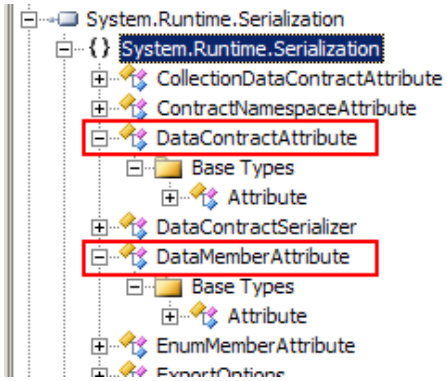
Sınıf kütüphanemiz içerisinden, **AdventureWorks** içerisindeki tablolara ilişkin bazı hizmetler vereceğiz. Bunlardan
biriside **HumanResources** şemasındaki **Employee** tablosundan herhangibir çalışan bilgisini döndürecek bir metod olacaktır. Lakin bu metodun dönüşünde biz kendi geliştireceğimiz sınıfımıza ait bir nesne örneğini döndüreceğiz. Bu nedenle ilk olarak aşağıdaki gibi bir sınıfı, kütüpanemiz içerisine katarak işe başlayabiliriz.




```
[DataContract]
public class Employee
{
   [DataMember]
   public int EmployeeId;

   [DataMember]
```

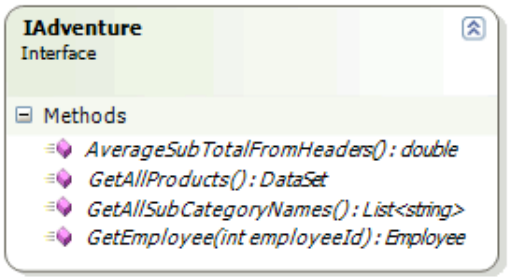
```
    public string Title;

    [DataMember]
    public DateTime BirthDate;

    [DataMember]
    public int VacationHours;

    public Employee()
    {
    }
    public Employee(int id, string title, DateTime birth, int vacation)
    {
        EmployeeId = id;
        Title = title;
        BirthDate = birth;
        VacationHours = vacation;
    }
}
```

Employee isim sınıfımız çok basit olarak bir kaç public alan(**Field**) ile, aşırı yüklenmiş(**overload**) ve varsayılan yapıcı**(default constructor)** metodları içermektedir. Ancak burada asıl dikkat edilmesi gereken sınıfın ve üyelerin nitelikleridir. Dikkat ederseniz sınıfımız **DataContract** niteliği(attribute) ile, public olan alanlarımız ise **DataMember** nitelikleri ile işaretlemiş durumdadır. DataContract ve DataMember isimli niteliklerimiz **System.Runtime.Serialization** isim alanı altında yer alan sınıflardır. Böylece servisimizin döndüreceği özel bir tip için gerekli veri sözleşmesini bildirmiş bulunuyoruz.



Eğer servis metodlarımız geriye ilkel tipler döndürmüyorsa (int, double, string gibi), özellikle Employee gibi geliştirici tarafından tasarlanmış tipler söz konusu ise mutlaka bir veri sözleşmesinin**(data contract)** tanımlanması gerekmektedir. *(İlerleyen makalelerimizde veri sözleşmelerine daha detaylı bir şekilde bakmaya çalışacağız.)* Artık kendi veri sözleşmemizide(Data Contract) tanımladığımıza, bir başka deyişle servisteki ilgili

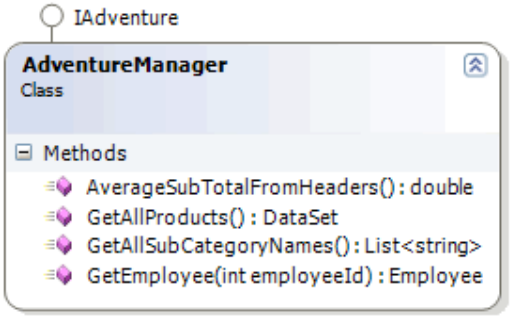
metoddan geriye nasıl bir tip döndürüleceğini bildirdiğimize göre, iş yapan metodlarımızı içerecek tip için gerekli servis sözleşmesini **(service contract)** yazarak işlemlerimize devam edebiliriz. IAdventure isimli arayüzümün içeriği aşağıda görüldüğü gibidir.




```
[ServiceContract(Name="Adventure Works Services")]
public interface IAdventure
{
   [OperationContract]
   Employee GetEmployee(int employeeId);

   [OperationContract]
   DataSet GetAllProducts();

   [OperationContract]
   List<string> GetAllSubCategoryNames();

   [OperationContract]
   double AverageSubTotalFromHeaders();
}
```

IAdventure arayüzümüze(interface) ait üyelerde özellikle çeşitli tiplerden dönüş değerleri kullanmaya çalıştık. Bildiğiniz gibi, yazacağımız servisin istemciler tarafından kullanılacak bir sözleşmesi olması gerekmektedir. Herhanbiri arayüzün veya sınıfın bir servis sözleşmesi (Service Contract) sağlayacağını bildirmek için **ServiceContract** niteliği ile imzalamak gerekmektedir. Benzer şekilde servisin sunabileceği metodları bildirmek içinse **OperationContract** isimli nitelik kullanılmaktadır. ServiceContract ve OperationContract isimli nitelikler(attributes),**System.ServiceModel** isim alanı(namespace) altında yer almaktadır.

Servis sözleşmemizi tanımladıktan sonra, uzak nesne olarak asıl iş yapan metodları taşıyacak ve istemcilerin başvuruda bulanabileceği sınıfı tasarlayabiliriz. **AdventureManager** isimli sınıfımız herşeyden önce **IAdventure** arayüzünü uyarlamalıdır(Implementation).

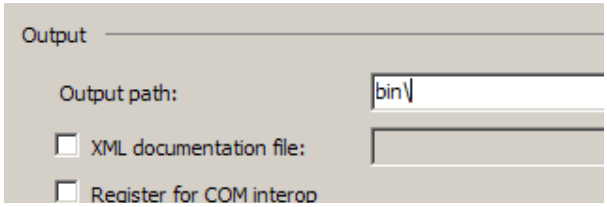



```
class AdventureManager:IAdventure
{
   #region IAdventure Members

   public Employee GetEmployee(int employeeId)
   {
      Employee emp=null;
      using (SqlConnection conn = new SqlConnection("data
source=localhost;database=AdventureWorks;integrated security=SSPI"))
      {
         SqlCommand cmd = new SqlCommand("Select
EmployeeId,Title,BirthDate,VacationHours From HumanResources.Employee Where
EmployeeId=@EmpId", conn);
         cmd.Parameters.AddWithValue("@EmpId", employeeId);
         conn.Open();
         SqlDataReader dr = cmd.ExecuteReader();
         if (dr.Read())
         {
            emp = new Employee();
            emp.EmployeeId = Convert.ToInt32(dr["EmployeeId"]);
            emp.Title = dr["Title"].ToString();
            emp.VacationHours = Convert.ToInt32(dr["VacationHours"]);
            emp.BirthDate = Convert.ToDateTime(dr["BirthDate"]);
         }
         dr.Close();
      }
      return emp;
   }

   public DataSet GetAllProducts()
   {
      DataSet ds=null;
      using (SqlConnection conn = new SqlConnection("data
source=localhost;database=AdventureWorks;integrated security=SSPI"))
```

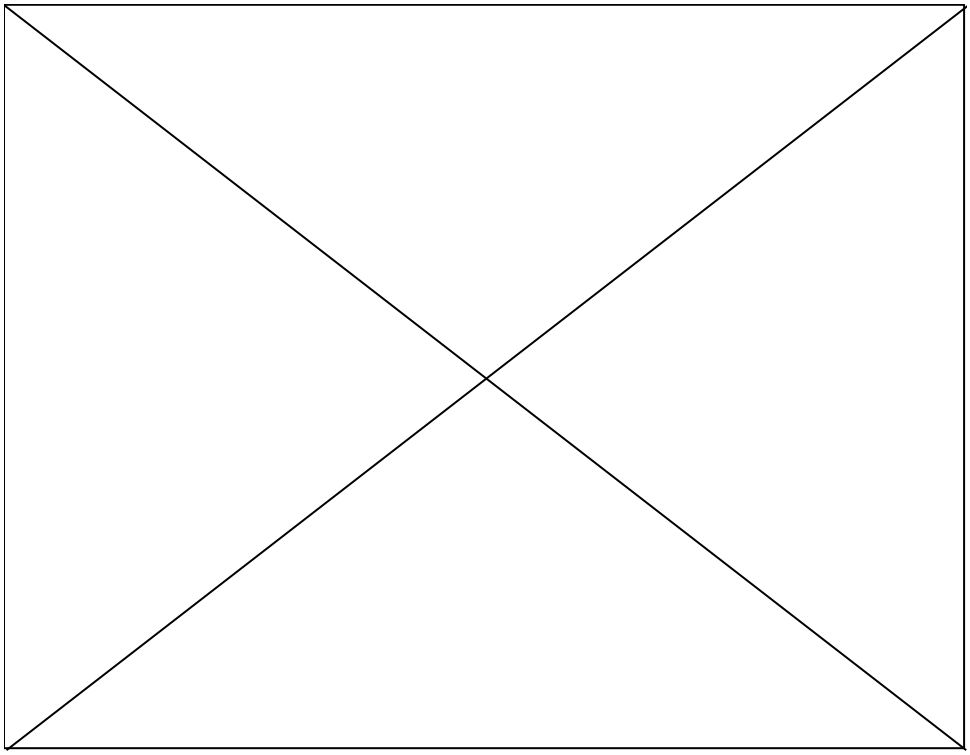
```
        {
            SqlDataAdapter da = new SqlDataAdapter("Select
ProductId,Name,ListPrice,Class,SellStartDate From Production.Product", conn);
            ds = new DataSet();
            da.Fill(ds);
        }
        return ds;
    }

    public List<string> GetAllSubCategoryNames()
    {
        List<string> categoryNames = null;
        using (SqlConnection conn = new SqlConnection("data
source=localhost;database=AdventureWorks;integrated security=SSPI"))
        {
            SqlCommand cmd = new SqlCommand("Select ProductSubCategoryId,Name From
Production.ProductSubCategory Order By Name", conn);
            conn.Open();
            SqlDataReader dr = cmd.ExecuteReader();
            categoryNames = new List<string>();
            while (dr.Read())
            {
                categoryNames.Add(dr["Name"].ToString());
            }
            dr.Close();
        }
        return categoryNames;
    }

    public double AverageSubTotalFromHeaders()
    {
        double average=0;
        using (SqlConnection conn = new SqlConnection("data
source=localhost;database=AdventureWorks;integrated security=SSPI"))
        {
            SqlCommand cmd = new SqlCommand("Select Avg(SubTotal) From
Sales.SalesOrderHeader", conn);
            conn.Open();
            average = Convert.ToDouble(cmd.ExecuteScalar());
        }
        return average;
    }

    #endregion
}
```

AdventureManager isimli sınıfımız içerisinde kullandığımız metodlar basit Ado.Net fonksiyonelliklerini ele alarak farklı tiplerde sonuçlar üretmektedir. Konumuz WCF olduğu için metod içerisindeki kod parçalarının işleyiş şekline çok fazla değinmeyeceğiz. Sırada servis için gerekli sanal klasörün, svc dosyasının ve web.config dosyasının oluşturulması adımları yer almaktadır.

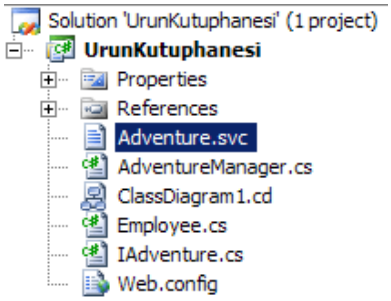
İlk olarak IIS üzerinde sanal klasörümüzü oluşturalım. Özellikle IIS üzerinden host edilen assembly' lar göz önüne alındığında, dll' in **Bin** klasörü altında yer alması ve Bin klasörü ile aynı seviyede bir **web.config** dosyası olması gerekmektedir. Söz konusu web.config dosyası uzak nesne kullanımı için gerekli alt yapı ayarlarını barındırmaktadır. Örneğimizde IIS altına açacağımız sanal klasör(Virtual Directory) içinde aynı durumu ele almak istediğimizden proje özelliklerine girip **Build** sekmesinde yer alan **Output Path** özelliğinin değerinin **bin\debug** adresinden **bin** adresine çekilmesi yeterli olacaktır. Böylece, kodda yapacağımız değişikliker anında canlı ortamada yansıyacaktır.



Bu tarz bir işlem zorunlu değildir. Sanal Klasörün(Virtual Directory) işaret edeceği başka bir fiziki adreste belirlenebilir ancak yine **Bin** klasörünün olması, dll' in kopyasının burada yer alması ve bin ile aynı seviyede bir **web.config** dosyasının alt yapı ayarlarını içerecek şekilde olması şarttır. Örneğimizde, yapacağımız derlemelerin anında yansımalarını test edebilmek amacıyla bu tarz bir ön hazırlık yapmayı tercih ediyoruz. Artık IIS altında sanal klasörümüzü oluşturabiliriz. Aşağıdaki flash animasyonunda basit olarak IIS üzerinde yapmamız gereken adımlar gösterilmektedir. *(Flash animasyonunun boyutu 187 kb olup yüklenmesi zaman alabilir.)*



Makalemizde ilerlemeden önce Web servisleri ile HTTP üzerinden iletişime geçerken devreye giren asmx uzantılı dosyaları göz önüne almamızda fayda olacaktır. Sonuç itibariyle WCF servisimizi IIS üzerinden host ederken **asmx** ile aynı amaca yönelik bir dosya var olmalıdır ki bu sayede istemciler URL üzerinden servis metadata içeriğini talep edebilsinler. İşte WCF için bu görevi yerine getiren **svc** uzantılı bir servis dosyası olacaktır. Bu dosya aynı zamanda **EndPoint** için gerekli adres tanımlanırkende ele alınacaktır. *(Bildiğiniz gibi bir WCF servisi içerisinde birden fazla EndPoint yer alabilir ve bunların ayrıt edici özelliklerinden biriside adresleridir.)* Bu nedenle projemize aşağıdaki içeriğe sahip olan svc uzantılı bir dosya eklememiz gerekmektedir. Bu dosya tipini öğeler arasından bulamayabilirsiniz. Bu nedenle boş bir text dosyasını svc uzantısı ile kaydederek sorunu aşabiliriz.

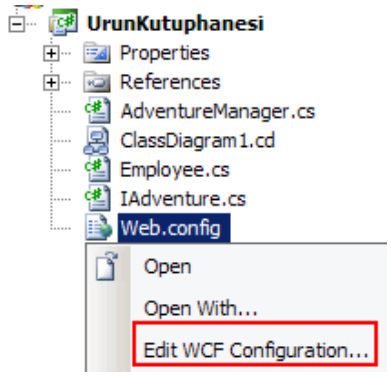


<%@**ServiceHost Service**="UrunKutuphanesi.AdventureManager"%>
<%@**Assembly Name**="UrunKutuphanesi"%>

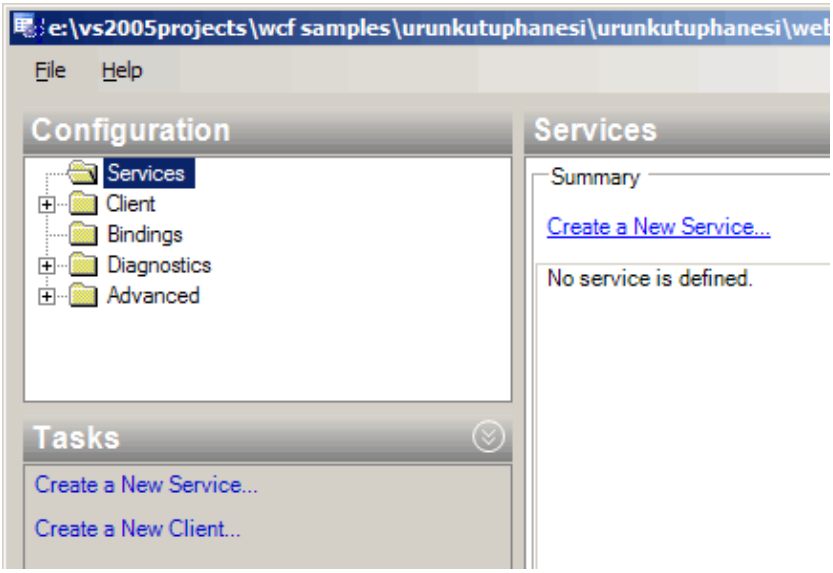
Svc dosyamız içerisinde iki direktif yer almaktadır. **ServiceHost** direktifi içerisinde Service' e ait tip bilgisi yer alır. Bir başka deyişle uzak nesne görevini üstlenecek olan

tipimizi bildiririz. Dikkat ederseniz notasyon olarak isimAlanı.tipAdı formatı kullanılmıştır. **Assembly** direktifinde ise söz konusu tipin, içerisinde yer aldığı assembly adı bildirilmektedir.

Artık web.config dosyamız için gerekli konfigurasyon tanımlamalarını yapabiliriz. (*Yada görsel aracımız yardımıyla kolayca inşa edebiliriz :))* Bu amaçla projemize yeni bir **Application Configuration File** öğesi (Item) ekleyelim ve ismini **web.config** olarak değiştirelim. Konfigurasyon ayarlarımızı yapmak için web.config dosyamıza sağ tıklayıp**, Edit WCF Configuration**seçeneğini işaretlememiz yeterli olacaktır.



Edit WCF Configuration vasıtasıyla açılan yardımcı arabirim aslında, IIS üzerinde yer alan Asp.Net sekmesindeki Edit Configuration linki yardımıyla açılana benzemektedir. Ancak kullanım amacı tamamıyla WCF' a yöneliktir ve daha da önemlisi IDE dışına çıkmadan gerekli ön hazırlıkların yapılabilmesini sağlamaktadır.



Elbette buradaki ayarları yapmadan önce bir WCF servisi için gereken temel konfigurasyon ayarlarının ne olduğunu bilmek gerekmektedir. Bu konu ile ilgili bir önceki makalemizden de hatırlayacağınız gibi, WCF' nin **ABC**' si önemli bir rol oynamaktadır. Yani **AddressBindingContract** üçlemesinden söz ediyoruz. Bu kuralları

konfigurasyon dosyamızda da göz önüne almalıyız. WCF Configuration arabirimi, service ve istemci**(Client)** tarafı için ayrı ayrı konfigurasyon dosyaları hazırlanabilmesine olanak tanımaktadır. Şu aşamadan biz servis**(Service)** tarafı için gerekli konfigurasyon ayarlarını yapacağız. Bu amaçla **Create a New Service** linkinden faydalanabilir yada manuel olarak **Configuration** sekmesini kullanabiliriz. Biz örneğimizde Create a New Service linkinden faydalanacağız.

Bu linke tıklandıktan sonra sihirbaz bizden uzak nesne görevini üstlenecek tipi seçmemizi isteyecektir. Örneğimizde söz konusu olan tip **AdventureManager** sınıfıdır. Bu seçimin arkasından sihirbaz, servis sözleşmesini**(Service Contract)** içeren tipi belirtmemizi isteyecektir. Örneğimizde bu yükü üstlenen **IAdventure** isimli arayüzdür(Interface). WCF Configuration IDE' si eğer ilk adımda seçilen tipin uyarladığı ve **ServiceContract** niteliğini taşıyan bir arayüz var ise bunu otomatik olarak bulacaktır. Bir sonraki adımda bağlantı tipi seçilir. Burada **HTTP, TCP, MSMQ, Peer To Peer** gibi seçenekler yer almaktadır. Örneğimizde HTTP tipini seçerek ilerleyeceğiz. Takip eden adımımızda **WSI (Web Service Interoperability)** ayarları yapılabilir veya varsayılan hali ile bırakılabilir. Biz bu adımı değiştirmeden devam edebiliriz. Son adımda ise servisimiz için bir **endpoint** adresi bildirilmesi gerekmektedir.

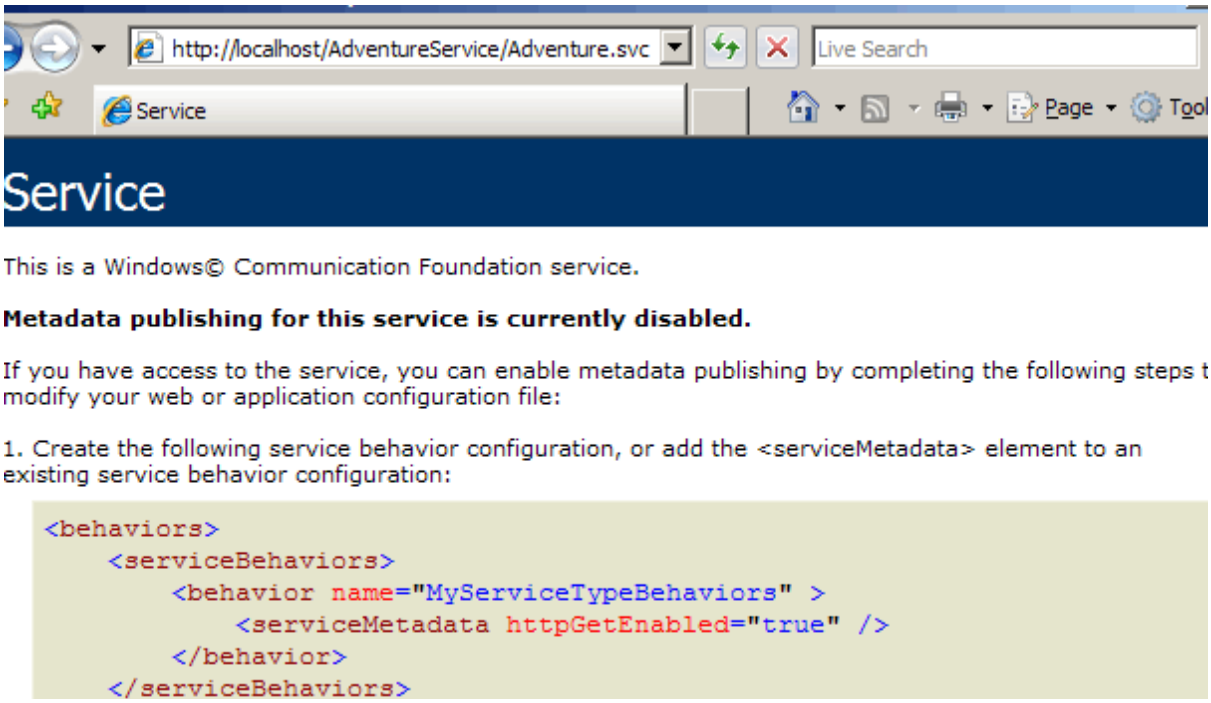
WCF mimarisinde istemciler(Clients), WCF Servisleri ile haberleşirken taleplerini **proxy** sınıflarına iletirler. Proxy sınıfları ise servis üzerinde yer alan endpoint' ler ile haberleşebilirler. Bu sebepten servis tarafında endpoint tanımlamalarının yapılması gerekmektedir. Bir endpoint tanımlaması içerisinde servisin adres bilgiside bulunur. Bu sebepten bu adımda **HTTP** protokolüne uygun bir adres girilmesi gerekir. Zaten örneğimizi IIS üzerinden sunduğumuz ve sanal klasörümüzü buna göre oluşturduğumuz için vereceğimiz adres bellidir.http://localhost/AdventureService/Adventure.svc .Dikkat ederseniz adresimizde daha önceden oluşturduğumuz Adventure.svc isimli dosyamızda yer almaktadır. Böylece şimdilik gerekli WCF ayarlarını tamamlamış bulunuyoruz. Aşağıdaki Flash animasyonuda gerekli ayarların nasıl yapıldığını izleyebilirsiniz. *(Flash dosyasının boyutu 229 kb olduğu için yüklenmesi zaman alabilir)*

Bu işlemlerin ardından web.config dosyamızın içeriğide aşağıdaki gibi olacaktır.

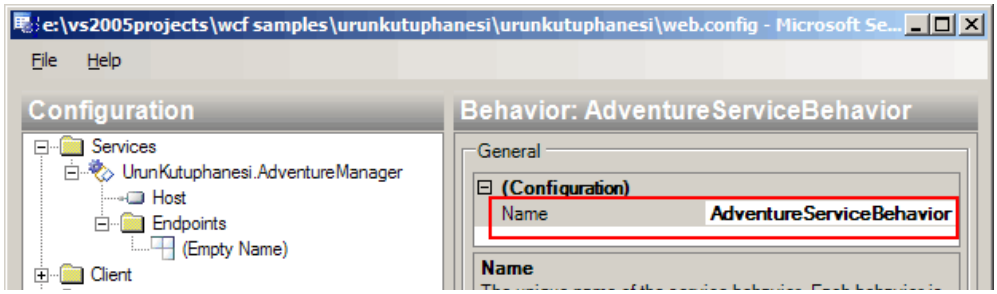
```
<?xml version="1.0" encoding="utf-8" ?>
  <configuration>
    <system.serviceModel>
      <services>
        <service name="UrunKutuphanesi.AdventureManager">
          <endpoint address="http://localhost/AdventureService/Adventure.svc" bindi
ng="basicHttpBinding"
bindingConfiguration="" contract="UrunKutuphanesi.IAdventure" />
        </service>
      </services>
    </system.serviceModel>
</configuration>
```

Artık servisimizin çalışıp çalışmadığını tarayıcı penceremiz üzerinden test edebiliriz. Tek yapmamız gereken http://localhost/AdventureService/Adventure.svc adresini talep etmek olacaktır. Bu talep sonucunda eğer herşey yolunda ise aşağıdakine benzer bir ekran görüntüsü ile karşılaşırız.

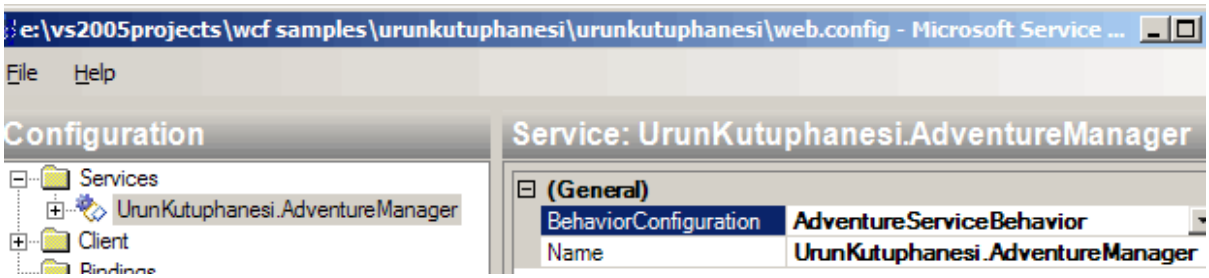
Yine dikkat edilecek olursa servisin metadata yayınlamasının**(MetaData Publishing)** aktif olmadığını görürüz. Bildiğiniz gibi metadata bilgisi istemcilerin proxy sınıflarını oluşturabilmeleri için gereklidir. Bunu sağlamak maksadıyla yeni bir servis davranışı**(Service Behavior)** oluşturulmalı ve servise bildirilmelidir. Bu amaçla ilk olarak **Advanced** sekmesinde yer alan**Service Behaviors** kısmından yeni bir davranış nesnesi eklenmeli ve aşağıdaki ekran görüntüsünde yer alan ayarlar yapılmalıdır. Bu ayarlara göre eklenen servis davranışına bir isim verilmiş (Name özelliği ile) ve **serviceMetadata** elementi dahil edilmiştir.



ServiceMetadata elementinin içeriğininde aşağıdaki gibi düzenlenmesi gerekmektedir. Burada **HttpGetEnabled** özelliğinin değerini **true** olarak belirlemeliyiz ki HTTP üzerinden servis metadata içeriğini çekebilelim.



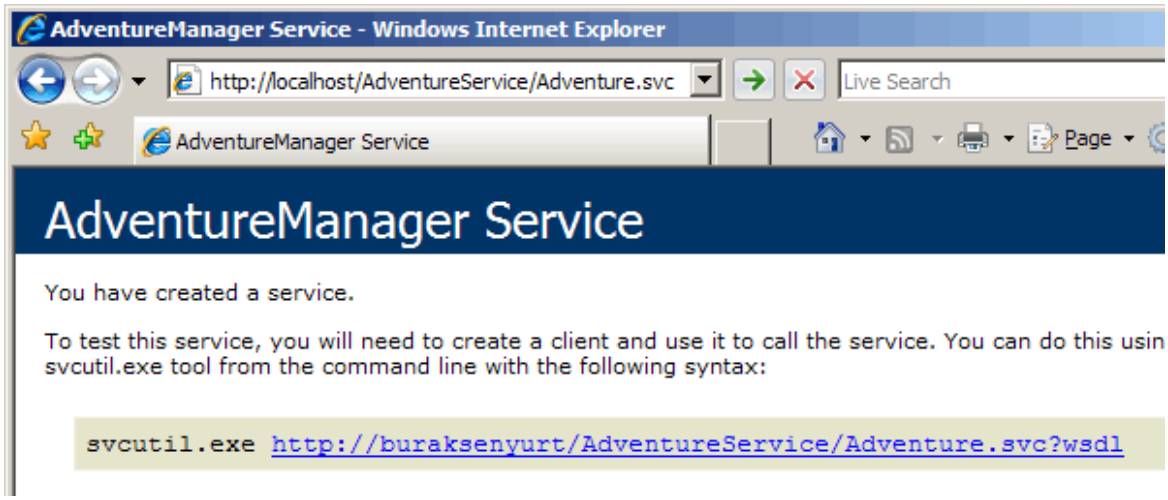
Sonrasında ise servisimizin belirtilen davranışı kullanacağını bildirmemiz gerekmektedir. Bunun içinde servisin, **BehaviorConfiguration** elementine aşağıdaki ekran görüntüsünde olduğu gibi az önce oluşturulan Service Behavior nesnesinin adını vermeliyiz.



Bu işlemlerin ardından servisimize ait web.config dosyasının son halide aşağıdaki gibi olacaktır.

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <behaviors>
      <serviceBehaviors>
        <behavior name="AdventureServiceBehavior">
          <serviceMetadata httpGetEnabled="true" />
        </behavior>
      </serviceBehaviors>
    </behaviors>
    <services>
      <service
behaviorConfiguration="AdventureServiceBehavior" name="UrunKutuphanesi.Adven
tureManager">
        <endpoint address="http://localhost/AdventureService/Adventure.svc"
binding="basicHttpBinding" bindingConfiguration=""
contract="UrunKutuphanesi.IAdventure" />
      </service>
    </services>
  </system.serviceModel>
</configuration>
```

Gördüğünüz gibi servisimiz için berlilediğimiz davranış ve servis ile olan ilişkisi otomatik olarak konfigurasyon dosyasına element ve nitelikler yardımıyla yansıtılmıştır. Servisimizi tarayıcı penceresi üzerinden yeniden talep edersek aşağıdaki ekran görüntüsü ile karşılaşırız. Dikkat edersek, servise ait Wsdl dökümanını elde edebilmemizi sağlayacak bir link bilgiside yer almaktadır.

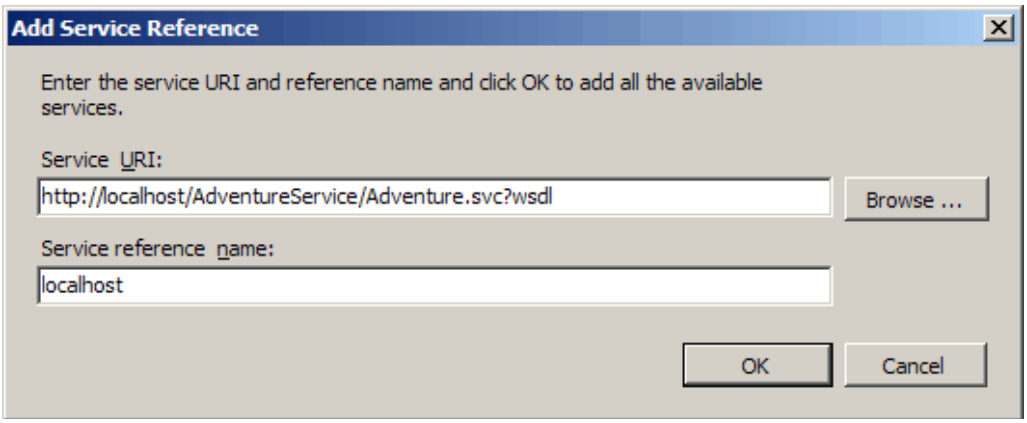


Bu kadar işlemden sonra artık istemcimizi yazmaya başlayabiliriz. İstemci tarafında herhangibir uygulama söz konusu olabilir. Biz mimariyi basit bir şekilde ele almaya çalıştığımızdan bir Console uygulaması geliştireceğiz. Yine Visual Studio 2005' in **Add**

Service Reference seçeneğini kullanarak ilgili servisi otomatik olarak uygulamamıza ekleyebilir ve gerekli proxy sınıfının kolay bir şekilde oluşturulmasını sağlayabiliriz.

NOT : *Visual Studio 2005 dışında, bir WCF servisine ait metadata bilgisini elde edip proxy sınıfını ve istemci için gereken konfigurasyon dosyasını elde etmek amacıyla,* ***svcutil*** *isimli Framework 3.0 aracıda kullanılabilmektedir.*

Console uygulamamızı oluşturduktan sonra projeye sağ tıklayıp **Add Service Reference** seçeneğini kullanarak WCF servisimizi projemize eklememiz gerekmektedir. Burada içeride kullanmak istediğimiz isim alanının adınıda belirtebiliriz. Varsayılan olarak aynı web servislerinde olduğu gibi localhost verilmektedir.

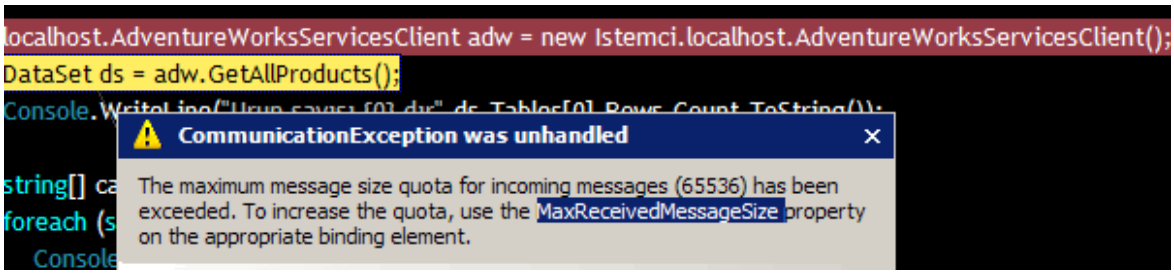


Bu işlemin ardından istemci için gereken proxy sınıfı ve konfigurasyon dosyası otomatik olarak oluşturulacaktır. Artık aşağıdaki kod satırlarını kullanarak sanki sıradan bir web servisini kullanıyormuş gibi WCF Servisimizi ele alabiliriz.

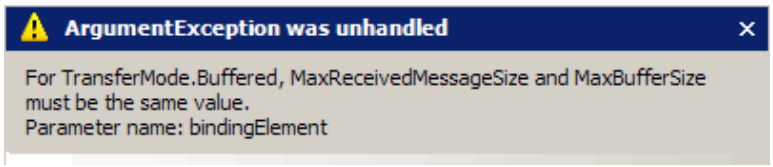
```
using System;
using System.Data;
using Istemci.localhost;

namespace Istemci
{
   class Program
   {
      static void Main(string[] args)
      {
         localhost.AdventureWorksServicesClient adw =
new Istemci.localhost.AdventureWorksServicesClient();
         DataSet ds = adw.GetAllProducts();
         Console.WriteLine("Urun sayısı {0} dır",ds.Tables[0].Rows.Count.ToString());

         string[] categoryNames=adw.GetAllSubCategoryNames();
         foreach (string category in categoryNames)
            Console.WriteLine(category);
```

```
        Employee emp=adw.GetEmployee(1);
        Console.WriteLine(emp.Title + " " + emp.BirthDate.ToShortDateString() + " " +
emp.VacationHours.ToString());

        Console.WriteLine("Ortalama Sub Total
{0}",adw.AverageSubTotalFromHeaders().ToString("C2"));
      }
   }
}
```

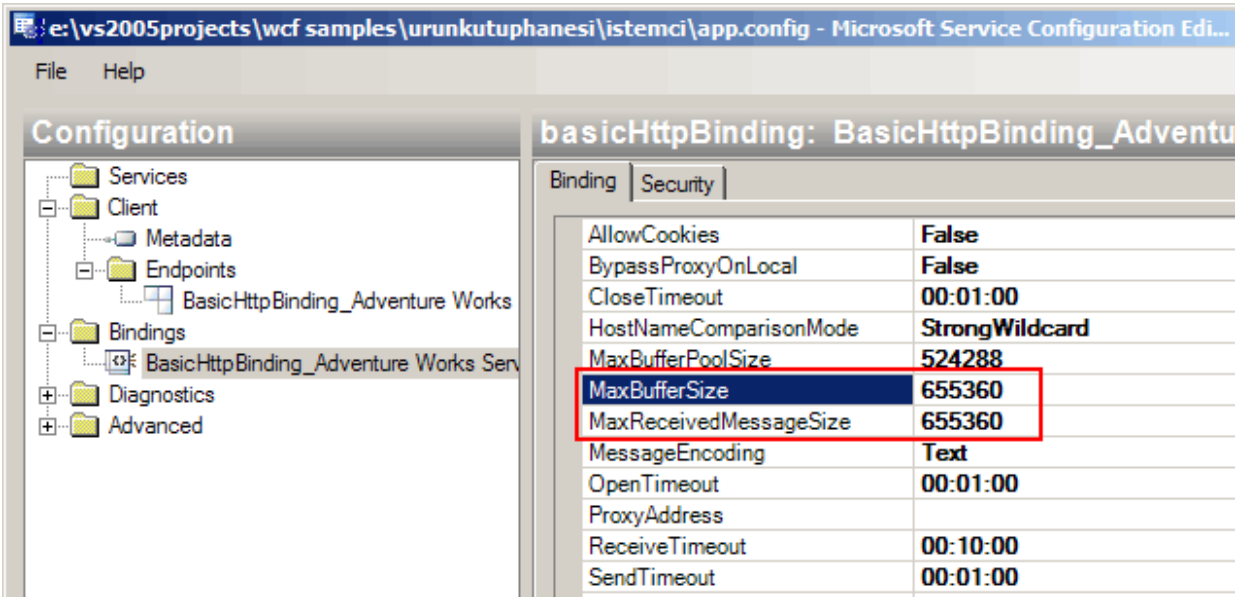
Uygulamayı bu haliyle çalıştırdığımızda GetAllProducts metodu için aşağıdaki gibi çalışma zamanı istisnası (run-time exception) alırız.



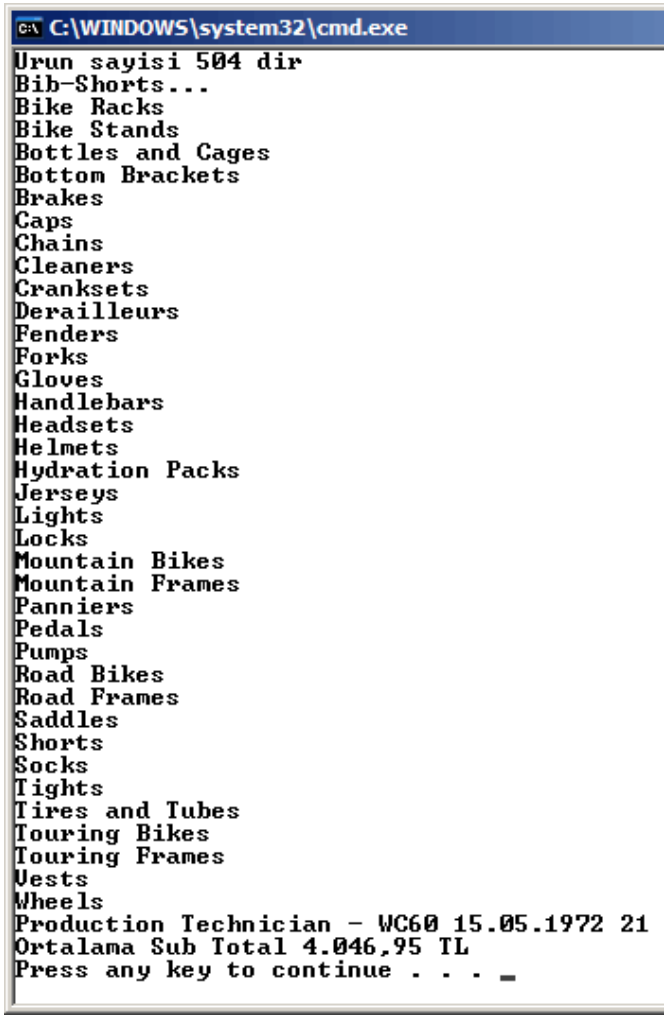
Bunun sebebi istemci tarafında oluşturulan konfigurasyon dosyasında belirtilen **MaxReceivedMessageSize** boyutunun düşük olmasıdır. Nitekim dönen DataSet içeriğinin yer aldığı paketin boyutu çok daha fazladır. Bu nedenle istemci uygulamaya ait App.config dosyasını Edit WCF Configuration ile açıp aşağıdaki ekran görüntüsünde olduğu gibi**MaxReceivedMessageSize** ve **MaxBufferSize** özelliklerinin aynı değere set edilmesi gerekir. Örneğimizde bu değerlerin sonuna bir 0 eklenmiştir. Burada dikkat edilmesi gereken noktalardan birisi de, her iki değerinde aynı olması zorunluluğudur. Aksi takdire çalışma zamanında yine bir istisna alırız.



Bu sebepten ayarlarımız örneğin aşağıdaki gibi olmalıdır.



Örneğimizdeki kodlarımızı incelersek eğer, ilginç olan bir nokta olduğunu farkederiz. **GetAllSubCategoryNames** metodu servis tarafında geriye generic bir List koleksiyonu döndürüyorkan, istemci tarafında metodun dönüşü **string[]** tipinden bir dizi olmuştur. Buda aslında farklı tipteki istemciler için ortak bir standart koyulmaya çalışıldığı şeklinde yorumlanabilir. Uygulamayı çalıştırdığımızda aşağıdakine benzer bir ekran görüntüsü elde ederiz.



Bu makalemizde Windows Communication Foundation çatısı altında, servislerimizi IIS üzerinden nasıl yayınlayıp kullanabileceğimizi incelemeye çalıştık. Bunu yaparken bir servis sözleşmesini**(Service Contract)** nasıl tanımlayabileceğimizi, bu sözleşmeyi uygulayan asıl uzak nesne sınıfının geriye kullanıcı tanımlı bir tip döndürmesi halide veri sözleşmesini**(data contract)** nasıl geliştirmemiz gerektiğini gördük. Bunlara ek olarak özellikle **IIS Hosting** için **svc** uzantılı dosyaların rolünü inceledik ve konfigurasyon dosyalarını daha kolay oluşturabilmemizi sağlayan **Edit WCF Configuration** seçeneğini ele aldık. Böylece geldik bir makalemizin daha sonuna. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Örnek Uygulama İçin Tıklayınız.

Kendi WebPart Kontrolümüzü Geliştirmek - 2 (2007-04-20T20:31:00)

asp.net 2.0,

Kendi web partlarımızı nasıl geliştirebileceğimizi ve bu sayede kişiselleştirilebilir web sunucu kontrollerini nasıl yazabileceğimizi bu konu ile ilgili bir önceki makalemizde incelemeye çalışmıştık. Bu makalemizde ise kendi Web Part bileşenlerimize özel fiillerin (Web Part Verbs) nasıl eklenebileceğini ve söz konusu fillerin ne şekilde ele alınabileceğini incelemeye çalışacağız. Web Part kontrollerini herhangibir **WebPartZone** altında kullandığımızda standart olarak bazı **fiilere(Verbs)** sahip oluruz. Tahmin edeceğiniz gibi bu değerler aslında **WebPartManager** tarafından ele alınmakta ve sayfa üzerinde uygun olan web part alanlarının (Web Part Zone) gösterilmesini sağlamaktadır. Söz gelimi kullanıcı **Edit** isimli fiili(Verb) seçtiğinde **WebPartManager**bileşeni, **EditorZone** kontrolünü aktif hale getirmekte ve söz konusu Web Part kontrolünün değiştirilebilen yada düzenlenebilen özelliklerinin(Properties) bulunduğu bir bileşeni (örneğin**PropertyGridEditorPart** kontrolü) göstermektedir. Bu açıdan bakıldığında fiillerin (Verbs) varsayılan olarak WebPartManager bileşenine ait **Display Mode** değerleri ile yakın bir ilişkide olduklarını söyleyebiliriz. Elbette farklı şekilde davranabilen fiillerde(Verbs) vardır. Örneğin **Minimize** fiili, bulunduğu Web Part' ın içerisinde yer aldığı WebPartZone alanının küçülmesini sağlarken, **Restore** fiili tekrardan eski haline getirilmesine olanak vermektedir.

Kendi Web Part bileşenlerimizi geliştirdiğimizde var olan fiillerin(Verbs) bize yetmediği durumlar söz konusu olabilir. Bu sebepten dolayı istersek kendi Web Part fiillerimizi (Web Part Verbs) oluşturabilir ve kod tarafında ele alarak farklı aksiyonların gerçekleştirilmesini sağlayabiliriz. Burada temel dayanak noktası **WebPartVerb** isimli sınıftır(class). Bu sınıf **IStateManager** isimli arayüzü uyarlayan (implement) bir tiptir. Kullanım amacı, Web Part bileşeni için özel bir fiili tanımlamaktır. Fiile ait isim (Name), açıklama (Description), seçilme durumu(Checked), resmi(ImageUrl) vb gibi bilgileri içerisinde barındıran bir sınıftır. Kullanıcılar bu fiili seçtiklerinde bir aksiyonun gerçekleştirilmesi gerektiği açık bir şekilde ortadadır. Buda doğal olarak bir metodun çalışma zamanında (run time) tetiklenmesi anlamına gelmektedir. İşte bu sebeple tanımlanan fiilerin gerçekleşmesi halinde çalıştırılacak olan metodları işaret eden **WebPartEventHandler**temsilcilerinden (delegates) faydalanılmaktadır. Bu temsilcinin prototipi ise aşağıdaki gibidir.

public delegate void **WebPartEventHandler**(object sender, **WebPartEventArgs** e);

Dikkat ederseniz **WebPartEventHandler** temsilcisi standart bir olay temsilcisidir. İkinci paramete olay metoduna bazı bilgileri taşımakta olan **WebPartEventArgs** sınıfına ait bir nesne örneğidir. Bu tipte doğal olarak **EventArgs** sınıfından türetilmiştir. İlk parametre ise fiili gerçekleştiren referansın bir başka deyişle WebPartVerb nesne örneğinin taşıyıcısıdır.

NOT : *Hatırlayalım ; Temsilciler (Delegates) çalışma zamanında metodların bellek üzerindeki başlangıç adreslerini işaret eden tiplerdir(types). Tanımlandıklarında, işaret edebilecekleri metodun yapısınıda (parametreleri ve dönüş tipi) belirtirler. Olay tabanlı programlamada (**Event based programming**), Asenkron (**Asynchronous**) mimaride yer alan Polling, Callback, WaitHandle gibi modellerde, çok kanallı uygulamalarda (**Multi Thread Applications**) kullanılmaktadırlar*

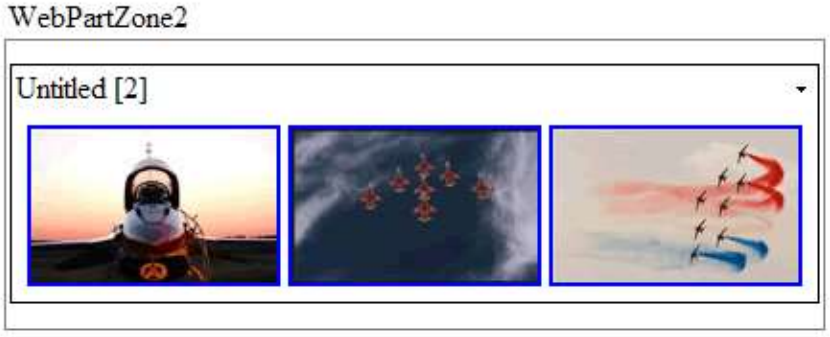
Bize gereken tiplerin neler olduğunu öğrendik. Peki bunları kendi Web Part kontrolümüzde nasıl ele alacağız. Bunun için WebPart sınıfından türetme yoluyla kendi Web Part kontrol sınıfımıza gelen **Verbs** isimli özelliğin ezilmesi (override) gerekmektedir.

public virtual **WebPartVerbCollection** Verbs { get; }

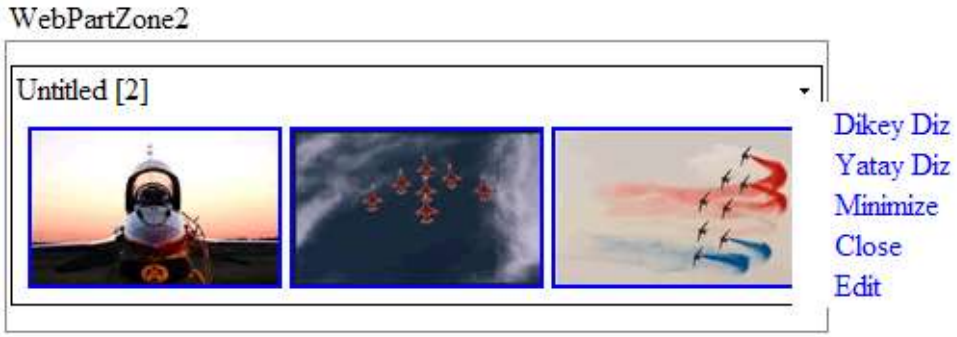
Yukarıda prototipi görünen bu özellik, yanlız okunabilir**(read only)** bir özelliktir ve geriye **WebPartVerbCollection** tipinden bir referans döndürmektedir. WebPartVerbCollection sınıfı türlendirilmiş (strongly typed) bir koleksiyonu temsil etmekte ve Web Part bileşenine eklenecek ekstra fiileri taşımaktadır. Dolayısıyla bu özelliğin get bloğu içerisinde istediğimiz fiileri(Verbs) oluşturmamız ve gereken olay metodu yüklemelerini yapmamız gerekecektir.

Bu makalemizde özellikle üzerinde duracağımız konu kendi fiillerimizi nasıl yazacağımızdır. Konuyu daha iyi anlayabilmek için örnek bir senaryo üzerinden hareket edeceğiz ve göze hoş gelecek bir Web Part kontrolü geliştirmeye çalışacağız. Bu sefer bir önceki Web Part bileşenimizden farklı olarak, **Render** metodu yerine **CreateChildControls** metodunu ezip, bileşen içindeki kontrollerin daha kolay bir şekilde nasıl oluşturulabileceğini de göreceğiz. Dilerseniz amacımızdan bahsederek örneğimizi geliştirelim. Sitemizde çeşitli kategorilerde duvar kağıtları(Wallpapers) olduğunu düşünelim. Siteye giren kullanıcılar seçtikleri kategorideki duvar kağıtlarından kaç tane istiyorlarsa görebilecekler ve istediklerine tıkladıklarında büyük versiyonlarına bakıp bilgisayarlarına indirebilecekler. Bu senaryda Web Part kullanacağımız için resim sayısı ve kategori gibi bilgileri kişselleşetirme(**Personalization**) şansınada sahip olacağız. Web Part kontrolümüz, seçilen kriterlere göre, kontrolün sayfaya her çizilişinde rastgele resimler seçecek ve bunları gösterecektir. Peki kendi fiillerimizi bu senaryo içerisine nasıl katabiliriz? Kullanıcıların isterlerse Web Part kontrolüne eklenen resimleri yatay veya dikey düzende görebileceklerini göz önüne alalım. Bunun için Yatay Diz ve Dikey Diz başlıklı örnek fiilleri Web Part kontrolümüze ekleyip, resimlerin sayfa üzerindeki diziliş yönlerini belirleyebiliriz. Üstelik bu fiillerin değerlerini kişiselleştirirsek, sayfayı son bıraktığımız haliyle elde edebiliriz. Örnek Web Part kontrolümüzü bitirdiğimizde aşağıdaki ekran görüntülerindekine benzer sonuçlar elde edeceğiz.

Örnek olarak Uçak kategorisinde her ziyaretimizde rastgele 3 resmin yanyana gösterilmesi;



Web Part kontrolümüz için geliştireceğimiz fiiller(Verbs);



Dikey Diz başlıklı Verb seçildiğindeki durum;

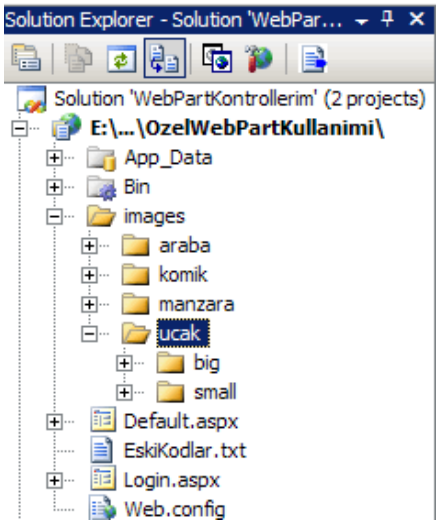
Artık kontrolümüzü geliştrimeye başlayabiliriz. Web Part bileşenimizi yine bir **Web Control Library** kütüphanesinde ele alabiliriz. ResimPart adlı Web Part sınıfımızın WebPart sınıfından türemesi(Inherit) gerektiğini hatırlayalım. Kontrolümüz kendi içerisinde **kişiselleştirilebilir (Personalizable)** 3 özellik barındırmalıdır. Bunlardan birisi ziyaretçinin görmek istediği resim sayısını tutan GosterilecekResimSayisi özelliğidir. Ziyaretçinin görmek istediği kategorinin bilgisini ise ResimKategori isimli özellik ile tutabiliriz. Son olarak ziyaretçinin seçtiği fiile (Verb) uygun olacak şekilde bir değişkenin de kişiselleştirilmesi önemlidir ki bir sonraki ziyarette son bıraktığımız haliyle bir dizilim elde edebilelim. Bu amaçlada ziyaretçinin son seçtiği fiili kişiselleştirilebilir bir özellik olacak şekilde CizimYonu ismiyle saklayacağız. Dilerseniz bahsetmiş olduğumuz özellikleri(Property) aşağıdaki gibi yazarak makalemize devam edelim.

```
[ToolboxData("<{0}:ResimPart runat=server></{0}:ResimPart>")]
public class ResimPart:WebPart
{
   #region Kişiselleştirilebilir özellikler için alan tanımlamaları

   private ResimKategorisi _kategori;
   private int _gosterilecekResimSayisi;
   private Yon _cizimYonu;

   #endregion

   #region Kişiselleştirilebilir Özellikler

   [WebBrowsable(true)]
   [WebDescription("Bakmak istediğimiz resimlerin kategorisi")]
   [WebDisplayName("Resim Kategorisi")]
   [Personalizable(PersonalizationScope.User, false)]
   public ResimKategorisi Kategori
   {
      get { return _kategori; }
      set { _kategori = value; }
   }

   [WebBrowsable(true)]
   [WebDescription("Seçilen kategoride gösterilecek resim sayısı")]
   [WebDisplayName("Resim Sayısı")]
   [Personalizable(PersonalizationScope.User, false)]
   public int GosterilecekResimSayisi
   {
      get{return _gosterilecekResimSayisi <= 0 ? 1 : _gosterilecekResimSayisi;}
      set{_gosterilecekResimSayisi = value <= 0 ? 1 : value;}
   }

   [WebBrowsable(true)]
   [WebDisplayName("Resimlerin Yönü")]
   [WebDescription("Resimler dikey veya yatay gösterilebilmesini sağlar")]
   [Personalizable(PersonalizationScope.User, false)]
   public Yon CizimYonu
   {
      get{return _cizimYonu;}
      set{_cizimYonu = value;}
   }

   #endregion
}
```

Bir önceki makalemizden de hatırlayacağınız gibi, özelliklerimizi kişiselleştirmek için gerekli nitelikler ile işaretliyoruz. Resimlerin yönünü ve var olan resim kategorilerini birer **enum** sabiti içerisinde saklamaktayız.

NOT : *Enum sabitleri bu senaryoda oldukça işe yarar. Ne varki kategori isimlerinin değiştiği yada yeni kategorilerin eklendiği durumlarda koda girip güncelleme yapmak ve uygulamayı tekrardan build etmek gerekecektir. Alternatif bir yol olarak bu tip bir verinin kod dışında bir ortamda, örneğin bir Xml dosyasında veya veritabanındaki bir parametre tablosunda saklanması göz önüne alınabilir.*

Normal şartlarda resimlerin kategorisi için daha farklı bir yöntem izlemek sağlıklı olacaktır. Kendi sistemimizde, aşağıdaki ekran görüntüsünde yer alan klasör yapısını baz alacak bir enum sabiti ele alınmaktadır. Resimlerin bir küçük birde orjinal hallerini tutmak için klasörleme mantığını kullanıyoruz. Buna göre söz konusu resimleri kategori adları şeklinde olan klasörler içerisinde yer alan big ve small alt klasörlerinde ayrıştırmış durumdayız. Web Part kontrolümüz küçük boyutlu resimleri small isimli klasörler altından çekerken, üzerlerine tıklandığında orjinal büyüklüklerindeki versiyonları ise boş bir tarayıcı penceresinde açacak şekilde big isimli alt klasörlerden çekmektedir.



Buna göre kullanıcının enum sabiti yardımıyla seçtiği kategorideki resimleri görebilmesi için, klasör adı ile enum sabiti adının aynı olması gerekir. Bu durumda yanlışlıkla klasörün isminin değiştirilmesi sonucu sistem beklediğimiz şekilde çalışmayacaktır. Özellikle istenen kategoriye bağlı klasör bulunamayacağından çalışma zamanı istisnaları (run time exception) alınması kaçınılmazdır. Bu durumun önüne geçmek için neler yapılabileceğini düşünmekte fayda olacağı kanısındayım. Bize yardımcı olacak enum sabitlerimiz aşağıdaki gibidir.

```
public enum ResimKategorisi
{
   araba,
```

```
   ucak,
   manzara,
   komik
}

public enum Yon
{
   DikeyYon,
   YatayYon
}
```

Şimdide Web Part kontrolümüz için gerekli fiillerimizi (Verbs) geliştirelim. Hatırlayacağınız gibi makalemizin başında bu iş için Verbs özelliğini ezmemiz**(override)** gerektiğini söylemiştik. Bu özellik içerisinde tanımlayacağımız **WebPartVerb** tipinden nesne örneklerinin yapıcı metodları(constructors) içerisinde, ilgili fiil(Verb) seçildiği zaman çalıştırılacak olan metodu işaret edecek bir temsilci tanımı yapılmaktadır. Bunları hesaba katarak Web Part kontrolümüzün içeriğini ilk aşamada aşağıdaki gibi geliştirebiliriz.

```
WebPartVerb vrbYatay, vrbDikey;

public override WebPartVerbCollection Verbs
{
   get
   {
      vrbDikey = new WebPartVerb("DikeyDizilim", new
WebPartEventHandler(DikeyDiz));
      vrbYatay = new WebPartVerb("YatayDizilim", new
WebPartEventHandler(YatayDiz));

      vrbDikey.Text = "Dikey Diz";
      vrbYatay.Text = "Yatay Diz";

      WebPartVerb[] verbs = new WebPartVerb[2];
      verbs[0] = vrbDikey;
      verbs[1] = vrbYatay;
      WebPartVerbCollection verbCollection = new WebPartVerbCollection(verbs);

      return verbCollection;
   }
}

public void YatayDiz(object sender, WebPartEventArgs e)
{
   CizimYonu = Yon.YatayYon;
```

```
}
public void DikeyDiz(object sender, WebPartEventArgs e)
{
   CizimYonu = Yon.DikeyYon;
}
```

Şimdi neler yaptığımıza kısaca bakalım. Kendi yazacağımız fiillerimizi WebPartVerb tipinden tanımladıktan sonra **get** bloğu içerisinde oluşturmaktayız. Bu işlemi yaparken ikinci parametre ile bir **WebPartEventHandler** temsilci örneği tanımladığımıza ve DikeyDiz ile YatayDiz isimli metodları işaret ettiğimize dikkat edelim. Buna göre, kullanıcılar bu fiillerden birisine tıkladığında YatayDiz ve DikeyDiz isimli metodlar çalışacaktır. Bu metodların içerisinde Web Part kontrolümüz için tanımladığımız CizimYonu özelliğinin değerini berlilemekteyiz. Oluşturulan **WebPartVerb** kontrollerini bir dizi içerisinde topladıktan sonra bir **WebPartVerbCollection** koleksiyonunun üretilmesinde kullanıyoruz. Son olarak get bloğundan bu koleksiyonu geri döndürmekteyiz. Peki fiilin seçilmesi sonucunda resimleri yatay veya dikey olarak nasıl yerleştireceğiz?

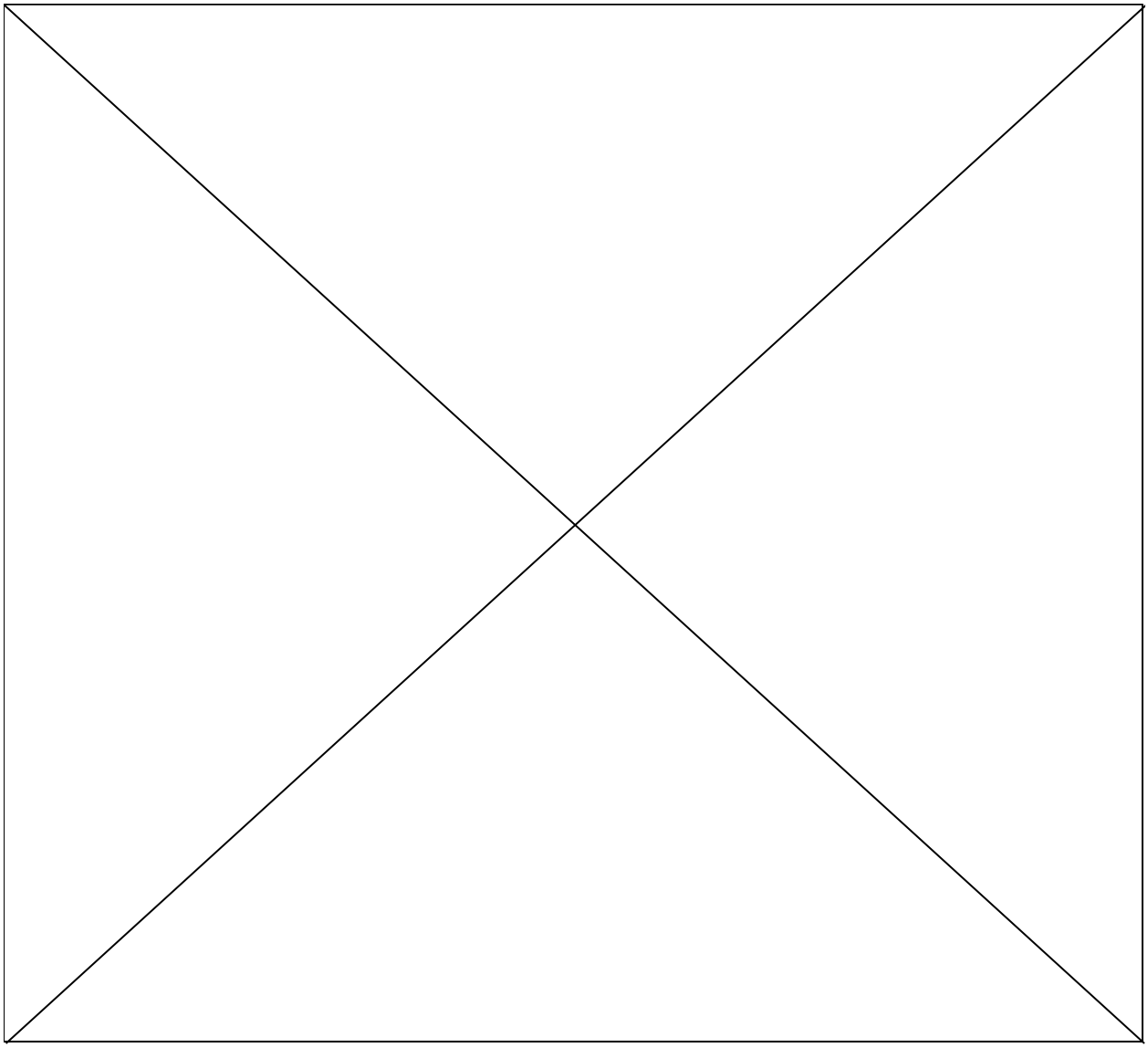
Sonuç itibariyle seçilen resimlerin ekrana alınması ve belirli bir yöne doğru çizilmesi demek, Web Part kontrolünün ekrana çizilmesi sırasında (Render) uygun HTML takılarının(Tag) oluşturulması demektir. Hatırlayacağınız gibi bir önceki makalemizde bu iş için **Render** metodunu kullanmıştık. Render metodu dışında var olan Web sunucu kontrollerinden yararlanaraktanda aynı işlemi gerçekleştirebilmekteyiz. Sonuç itibariyle Web Part kontrolleride birer taşıyıcı (**Container**) olduğundan bir **Controls** koleksiyonuna sahiptir. Dolayısıyla sunucu kontrollerini oluşturup bu koleksiyona ekleyerek HTML elementleri ile fazla uğraşmadan render işlemlerini gerçekleştirebiliriz. Web Part kontrollerinde bu işlem için tek yapmamız gereken**CreateChildControls** metodunu ezmek olacaktır. Kendi örneğimiz için bu metodu aşağıdaki gibi ezebiliriz.

```
protected override void CreateChildControls()
{
   string sanalAdres = HttpContext.Current.Request.Url.ToString();
   sanalAdres = sanalAdres.Substring(0, sanalAdres.LastIndexOf('/'));
   string sanalResimAdresi = sanalAdres + ("/images/") + Kategori.ToString();
   string fizikiKlasor = HttpContext.Current.Request.PhysicalPath;
   fizikiKlasor = fizikiKlasor.Substring(0, fizikiKlasor.LastIndexOf('\\'));
   int siraNo = 0;

   try
   {
      FileInfo[] resimDosyalari = DosyalariAl(fizikiKlasor);
      Random rnd = new Random();
      Table tablo = new Table();

      if (CizimYonu == Yon.DikeyYon)
      {
```

```
            for (int i = 0; i < GosterilecekResimSayisi; i++)
            {
                TableRow satir = new TableRow();
                siraNo = rnd.Next(0, resimDosyalari.Length);
                TableCell hucre = HucreOlustur(sanalResimAdresi, siraNo, resimDosyalari);
                satir.Cells.Add(hucre);
                tablo.Rows.Add(satir);
            }
        }
        else if (CizimYonu == Yon.YatayYon)
        {
            TableRow satir = new TableRow();
            for (int i = 0; i < GosterilecekResimSayisi; i++)
            {
                siraNo = rnd.Next(0, resimDosyalari.Length);
                TableCell hucre=HucreOlustur(sanalResimAdresi, siraNo, resimDosyalari);
                satir.Cells.Add(hucre);
            }
            tablo.Rows.Add(satir);
        }
        Controls.Add(tablo);
    }
    catch
    {
    }
}

private static TableCell HucreOlustur(string sanalResimAdresi, int siraNo, FileInfo[]
resimDosyalari)
{
    string miniResimDosyaAdresi = sanalResimAdresi + "/small/" +
resimDosyalari[siraNo].Name;
    string buyukResimDosyaAdresi = sanalResimAdresi + "/big/" +
resimDosyalari[siraNo].Name;
    TableCell hucre = new TableCell();

    string resim = "<a href='" + buyukResimDosyaAdresi + "' target='_blank'><img
src='" + miniResimDosyaAdresi + "'/></a>";
    hucre.Text = resim;
    return hucre;
}

private FileInfo[] DosyalariAl(string fizikiKlasor)
{
    DirectoryInfo fizikiKlasorBilgisi = new DirectoryInfo(fizikiKlasor + "\\images\\"
```

```
+ Kategori.ToString() + "\\small\\");
   FileInfo[] resimDosyalari = fizikiKlasorBilgisi.GetFiles();
   return resimDosyalari;
}
```

Burada kendimize göre bir algoritma geliştirdik. Temel olarak **CreateChildControls** metodu seçilen fiile göre yatay veya dikey dizilime uygun olacak şekilde bir HTML Table üretmekte ve **Controls**koleksiyonuna eklemektedir. Burada söz konusu olan **Table**, **TableRow** ve **TableCell** tipleri yönetimli kod(managed code) tarafında geliştirilmiş sunucu bileşenleridir ve çalışma zamanında üretilen sayfa içerisinde HTML Table, HTML TR (Satır), HTML TD (Hücre) elementlerine dönüştürülmektedir. Özel olarak, resimlerin küçük hallerini göstermek ve üzerlerine tıklandığında orjinal boyutlarında açmak için **a href** ve **img** HTML elementlerinden faydalanılmaktadır. Sizler bu kod parçasını daha efektif hale getirerek (örneğin optimize ederek) daha ölçeklenebilir bir şekle döndürebilirsiniz. Artık geliştirdiğimiz WebPart bileşenini örnek bir web uygulaması üzerinde deneyebiliriz. Bu amaçla geliştireceğimiz web uygulamasının kişiselleştirmeye destek verebilmesi amacıyla **Membership** ayarlarını içermesi doğru olacaktır. Sonuç olarak aşağıdaki Flash animasyonunda görülen çıktıyı elde ederiz. *(Flash dosyasının boyutu 180 Kb olup yüklenmesi zaman alabilir)*



Yukarıdaki Flash animasyonundanda gördüğünüz gibi, sisteme giren bir kullanıcı kendisine göre istediği kategorideki resimleri gösterebilmekte ve bunları yatay veya dikey olarak dizebilmektedir. Bizim dikkat etmemiz gereken nokta kendi fiillerimizin burada işlenebilir olmasıdır. Yukarıda geliştirdiğimiz ResimPart isimli Web Part kontrolümüzde fiillerimizi ayrı olay metodlarına yönlendirdik. Dilersek tüm fiillerimizi aynı metod içerisinde ele alabiliriz. Bunun için öncelikli olarak **WebPartVerb** bileşenlerimizi oluştururken kullanıdığımız **WebPartEventHandler** temsilcilerini (delegates) aynı metodu işaret edecek şekilde oluşturmalıyız.

```
vrbDikey = new WebPartVerb("DikeyDizilim", new
WebPartEventHandler(VerbUygula));
vrbYatay = new WebPartVerb("YatayDizilim", new
WebPartEventHandler(VerbUygula));
```

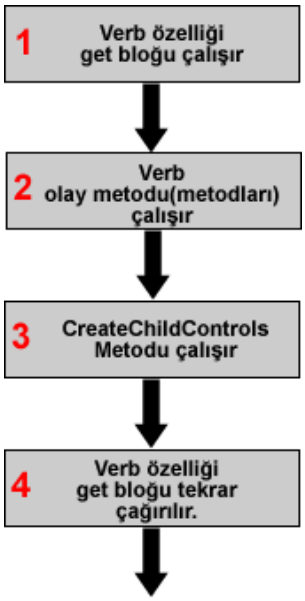
Daha sonra VerbUygula isimli metodumuzu aşağıdaki gibi kodlamamız yeterli olacaktır.

```
public void VerbUygula(object sender, WebPartEventArgs e)
{
   string tetiklenenVerbId=((WebPartVerb)sender).ID;
   if (tetiklenenVerbId == "DikeyDizilim")
      CizimYonu = Yon.DikeyYon;
   else if (tetiklenenVerbId == "YatayDizilim")
      CizimYonu = Yon.YatayYon;
}
```

Tüm fiiller aynı olay metodu içerisinde ele alınacaklarından, olayı meydana getiren **WebPartVerb** nesne örneğinin kim olduğunun bilinmesi gerekmektedir. Bunun için olay metodunun ilk parametresinden yararlanılmıştır. sender isimli değişken, olay metodu içerisinde **WebPartVerb** tipine dönüştürülmüş ve **ID** özelliğinin değerine bakılarak CizimYonu isimli enum sabitine uygun değeri atanmıştır. Uygulamayı bu haliyle test ettiğimizde ilk versiyondaki ile aynı sonuçları elde ederiz.

Çözmemiz gereken bir durum daha vardır. Bu da hangi Verb seçildiyse bunun başına bir check işaret konulmasının sağlanmasıdır. Bu amaçla WebPartVerb sınıfının **Checked**özelliğinin değerinin true veya false olarak değiştirilmesi yeterlidir. Lakin asıl problem bununla ilişki kodun nereye yazılması gerektiğidir. Web sayfaları ve içerisinde yer alan kontrollerin yaşam döngüsü olayı biraz karmaşık hale getirmektedir. Senaryomuzda, işaretleme operasyonu için gerekli kod parçasının **CreateChildControls** metodu içerisine konulması düşünülebilir. Yada bu işlemi fiilleri ele aldığımız olay metodu içerisine koyabiliriz. Ancak hiç birisi işe yaramayacaktır. Bunun sebebi Web Part bileşenimizin, sunucu tarafındaki yaşam döngüsüdür. Dilerseniz yaşam döngüsünü inceleyerek devam edelim. Sayfa ilk kez talep edildiğinde dolayısıyla Web Part kontrolüde ilk kez oluşturulduğundaki olay işleyiş sırası göz önüne alındığında, Web Part kontrolümüzdeki bazı üyelerin işleyiş sırası aşağıdaki gibi olacaktır.

Kullanıcı bir fiil (Verb) seçtikten sonra sayfa sunucuda tekrar oluşturulup, web part kontrolümüzde yeniden oluşturulacaktır. Bu ikinci request sonrasında üylerin işleyiş sırası ise aşağıdaki gibidir.



Görüldüğü gibi her iki yaşam döngüsü sırasında son olarak **Verbs** isimli özelliğe girilmektedir. Dolayısıyla işaretleme kodlarını buraya dahil edebiliriz. Bununla birlikte göze çarpan bir diğer durum, ikinci talep sonrasında Verb özelliğinin iki kez devreye giriyor olmasıdır. Bu sebepten, Verb özelliğinin **get** bloğundaki kodlar aşağıdaki gibi optimize edilmeli ve**WebPartVerb** bileşenleri ile ilgili koleksiyonun oluşturulma işlemlerinin sadece bir kez yapılması garanti altına alınmalıdır.

```
WebPartVerb vrbYatay, vrbDikey;
WebPartVerbCollection verbCollection;

public override WebPartVerbCollection Verbs
{
   get
   {
      if (vrbYatay == null && vrbDikey == null)
      {
         vrbDikey = new WebPartVerb("DikeyDizilim", new
WebPartEventHandler(VerbUygula));
         vrbYatay = new WebPartVerb("YatayDizilim", new
WebPartEventHandler(VerbUygula));

         vrbDikey.Text = "Dikey Diz";
         vrbYatay.Text = "Yatay Diz";

         WebPartVerb[] verbs = new WebPartVerb[2];
         verbs[0] = vrbDikey;
         verbs[1] = vrbYatay;
         verbCollection = new WebPartVerbCollection(verbs);
```

```
        }
        if (CizimYonu == Yon.DikeyYon)
        {
            vrbDikey.Checked = true;
            vrbYatay.Checked = false;
        }
        else if (CizimYonu == Yon.YatayYon)
        {
            vrbDikey.Checked = false;
            vrbYatay.Checked = true;
        }

        return verbCollection;
    }
}
```

Artık çalışma zamanında seçtiğimiz fiillerin yanında tik işaret aşağıdaki resimdekine benzer bir şekilde çıkacaktır.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde kendi Web Part kontrollerimize özel fiilleri (**WebPartVerb** tipinden nesne örneklerini) nasıl ekleyebileceğimizi örnek bir senaryo üzerinden incelemeye çalıştık. Makalemizin sonlarında kontrollerin yaşam döngüsünün ne kadar önemli olabileceğini gördük. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Örnek Uygulama İçin Tıklayınız.(Resimlerin boyutlarının büyük olması nedeni ile Big isimli kasörlerlerdeki resim dosyaları ve App_Data altındaki ASPNETDb.mdf dosyaları silinmiştir. Test ederken bunları göz önüne almayı unutmayınız.)

C# Temelleri - Nitelikleri(Attributes) Kavramak (2007-04-11T20:39:00)

c# temelleri,

Nitelik(Attribute) eninde sonunda her dotNet programcısının kullandığı ve karşılaştığı bir kavramdır. özellikle yansıma (Reflection) konusu ile birlikte anıldığından, .Net Framework içerisinde önemli bir yere sahiptir. .Net Framework içerisinde pek çok modelde

niteliklerden aktif olarak faydalanılmaktadır. Web servislerinden windows kontrollerini geliştirmeye, kendi web part bileşenlerimizi yazmaktan serileştirmeye kadar pek çok alanda işe yaramaktadır. Hatta çok popüler olarak, katmanlı mimarilerde ve nitelik bazlı (attribute based) programlama modellerinde de ele alınmaktadır. İşte bu makalemizde nitelikleri incelemeye çalışacak ve özellikle kendi niteliklerimizi nasıl geliştirebileceğimize değineceğiz.

Herşeyden önce, niteliği (Attribute) tanımlamakta fayda vardır. Nitelikler, uygulandıkları tiplerin (types) yada üyelerin (members) çalışma zamanındaki davranışlarının değiştirilmesine olanak sağlayan sınıflardır. Niteliklerin sınıf(class) olduğu rahatlıkla söylenebilir. Nitekim var olan veya bizim tarafımızdan geliştirilen nitelikler daima **Attribute** sınıfından türemek zorundadırlar. Attribute, **abstract** bir sınıftır. Dolayısıyla örneklenemez ancak bir nitelik sınıfının içermesi gereken temel üyeleri bünyesinde barındırır. Aslında niteliklerin belkide en önemli özelliği, üretilen assembly içerisinde yer alan tip ve üyelere ekstra bilgiler katabilmeleridir. Bir başka deyişle **metadata** içerisine ilave bilgiler eklenebilmesini sağlamaktadır. Bu noktada ortaya önemli bir soru çıkar. Söz konusu ekstra veriler kim tarafından ve nasıl değerlendirilecektir? İşte bu noktada yansıma(Reflection) konusu çok büyük önem taşımaktadır. öyleki, çalışma zamanında(run-time) herhangibir tipin ve üyelerinin hakkında bilgi sahibi olabilme imkanı aynı zamanda metadata içeriğinide elde edebilme anlamına gelmektedir.

NOT : *Nitelikler(Attributes), .Net Framework' de var olan veya geliştiriciler tarafından yazılan tip(type) veya üyelere(members) çalışma zamanında davranışlarının farklı şekillerde ele alınabilmelerini sağlayan ekstra metadata (veri hakkında veri) bilgileri ekler. Bu metadata bilgileri üretilen assembly' lar içerisinde yer alır ve yansıma(Reflection) teknikleri ile çalışma zamanında değerlendirilebilir.*

Niteliklerin faydasını ve ne işe yaradıklarını daha net bir şekilde anlayabilmek için aşağıdaki örnek senaryolar göz önüne alınabilir.

Asp.Net Web Uygulamalarında Kendi Kontrollerimizi Geliştirirken

Daha önceki makalelerimizde kendi web server kontrollerimizi nasıl yazacağımıza kısaca değinmiştik. Şimdi şöyle düşünelim. Yazdığımız bu kontrollerin ele alındığı bir geliştirme ortamı var mı? Cevabın Visual Studio IDE ortamı olduğunu gayet iyi biliyoruz. Peki nasıl oluyorda, bir kontrolü ToolBar üzerinden alıp sayfaya bıraktığımızda, özellikler (Properties) pencersinde, o kontrol sınıfına ait bazı üyeler (özellikler, olaylar) getiriliyor? Demekki, IDE bir çalışma zamanı ortamı olarak sürüklenip bırakılan kontrolün hangi üyelerinin **Properties** penceresinde görünmesi gerektiğini anlayabiliyor. Hatırlayacağınız gibi özelliklerin başına, hatta kontrol sınıfının başına atılan bazı nitelikler(attributes) vardı.

```
[Browsable(true)]
[Description("Hangi Gün?")]
[Bindable(true)]
```

```
[Themeable(true)]
[Category("Tarih Degerleri")]
[DefaultValue("1")]
[Localizable(true)]
public string SeciliGun
{
```

çok basit olarak SeciliGun isimli özelliğin üzerine yazılmış olan nitelikler, **Visual Studio IDE**' si için anlamlıdır. Nitekim Visual Studio çalışan bir uygulama olaraktan, çalışma zamanında (run-time) ilgili niteliklerin değerlerine bakarak bazı hamlelerde bulunur. örneğin **Browsable** niteliğinin true değerine sahip olması, SeciliGun özelliğinin Visual Studio IDE' sinde **Properties**penceresine eklenmesi gerektiği anlamına gelir. **Description** niteliği içerisindeki metinsel bilgiler, IDE tarafından değerlendirilip yine Properties penceresinde gösterilir. *(Diğer niteliklerin ne amaçla kullanıldıklarını Web Server Control Yazmak - 2 isimli makaleden bulabilirsiniz.)*

Kendi Web Part Bileşenlerimizi Geliştirdiğimizde

Bundan bir önceki makalemizde kendi Web Part bileşenlerimizi nasıl geliştirebileceğimiz incelemiştik. Geliştirdiğimiz Web Part bileşenlerinin bazı özelliklerinin kişiselleştirilebilmesi(**personalizable**) ve çalışma zamanında istemcinin bilgisayarında yer alan tarayıcı pencersindeki bir **PropertyGridEditorPart** içerisinde açılıp değiştirilebilmesi için aşağıdaki niteliklerden faydalandık.

```
[WebBrowsable(true)]
[WebDescription("Verilen Url adresine göre Rss bilgisini okur")]
[Personalizable(PersonalizationScope.User)]
[WebDisplayName("Rss Bilgisi Alınacak Url")]
public string Url
{
   get { return _Url; }
   set { _Url = value; }
}
```

İşte buradaki nitelikleri değerlendiren kişi, **Asp.Net Runtime Host**' un ta kendisidir. Yine çalışma zamanındaki bir ortamın karar mekanizmalarında ihtiyaç duyacağı bazı bilgiler metadata içerisine nitelikler yardımıyla eklenmektedir. Buna göre örneğin, Asp.Net Runtime Host, **Personalizable** niteliğinde **PersonalizationScope** isim enum sabitinin değerini **User** olarak gördüğünde takip eden özelliğin kişiselleştirme amaçlı olarak her kullanıcı için ayrı olacak şekilde veritabanına yazılması gerektiğini anlayacaktır. Yine **WebBrowsable** niteliğine true değeri verilmesi sayesinde, ilgili özelliğinde istemcilerin tarayıcı penceresinde görülecek olan **PropertyGridEditorPart** içerisinde ele alınabileceğinide anlayacak ve sayfanın sunucundan istemciye olan hareketinde, render işlemini bu kritere göre değiştirecektir. *(Diğer niteliklerin ne yaptığı ile ilgili*

olaraktan Kendi Web Part Bileşenlerimizi Geliştirmek isimli makalemizden yararlanabilirsiniz.)

Nesneleri Binary Formatta Serileştirmekte

Bildiğiniz gibi bir nesneyi ikili (binary) formatta serileştirmek için BinaryFormatter sınıfının **Serialize** metodundan yararlanırız. Benzer şekilde ters serileştirme işlemi içinde **Deserialize**metodunu kullanırız. Ancak hepimizin yakından tanıdığı bir kural vardır. Bir tipin ikili formatta(binary) serileştirilebilmesi için **Serializable** niteliği ile işaretlenmiş olması gerekir. Binary formatta serileştirmenin olduğu yerler göz önüne alındığında söz konusu niteliğin önemi ortaya çıkmaktadır. örneğin web uygulamalarında **session** bilgilerinin veritabanından tutulmasına karar verildiğinde veya **Profile** bilgilerinde kendi tiplerimizi yada var olan tipleri tablodaki binary alanda tutmak istediğimizde...

Windows Communication Foundation' da Kontratları(Contrats) Hazırlarken

Yakın zamanda, .**Net Framework 3.0** ile gelen ve Microsoft tabanlı dağıtık mimari (distributed architectures) modellerini tek bir çatı altında toplayan **Windows Communication Foundation**' da, bir sınıfın servis olarak yayınlanması için ve sınıf içinden dış dünyaya açılabilecek fonksiyonellikler için yine nitelikleri kullanmaktadır. Aşağıdaki örnek kod parçasında**ServiceContract** ve **OperationContract** isimleriye geliştirilen niteliklerin örnek uygulanış şeklini görmektesiniz. *(Daha detaylı bilgi için WCF Giriş makalesine bakabilirsiniz.)*

```
[ServiceContract]
  public interface IMatematikServis
  {
    [OperationContract]
    double Toplam(double x, double y);

    void DahiliMetod();
  }
```

Web Servislerinde

Bir web servisinin istemci tarafından tüketilebilmesi için çoğunlukla **proxy** sınıflarını kullanıyoruz. Elbette istisnai olarak doğrudan HTTP veya SOAP üzerinden talepte de bulunabilmekteyiz. Nitekim proxy sınıflarının üretilebilmesi içinde, web servisine ait bir **WSDL** dökümanının ele alınması gerekiyor. WSDL (Web Service Description Language) dökümanı bildiğiniz gibi bir web servisinin tanımlamalarının ve fonksiyonelliklerin bir XML içeriği olarak üretilmesini sağlıyor. Peki biz bu belgeyi herhangibir şekilde talep ettiğimizde, bu talebe karşılık XML dökümanı içerisine hangi sınıfların ve hangi metodların koyulacağını sistem nereden biliyor? İşte bu noktada devreye **WebService** ve **WebMethod** gibi nitelikler(attributes) girmektedir. Böylece

WSDL dökümanını hazırlayacak olan **HttpHandler** hangi sınıfı ve hangi metodu xml içerisine alacağını bilecektir.

Katmanlı Mimaride Entity Tiplerinde

özellikle katmanlı mimaride nitelikler çok faydalı olabilmektedir. örneğin, veritabanında yer alan tabloların karşılıklarının tutulduğu sınıflar için otomatik olarak select, insert,update ve delete gibi sorguların hazırlanması istendiği durumlarda çalışma zamanı için ekstra bilgilere (additional metadata) ihtiyaç vardır. İşte çalışma zamanındaki bu ihtiyçaları nitelikler yardımıyla karşılayabiliriz. Söz gelimi entity tipi içerisindeki alan adlarının tablolardaki karşılıklarını, identity olup olmadıklarını yada farklı entity tipleri arasında, tablolar arasındaki ilişkilerin nasıl gerçekleştirilebileceğini belirlemek vb konularda ele alınabilir.

DLINQ (Database Language Integrated Query) de Yer Alan Entity Tiplerinde

Şu anda C# 3.0 ile birlikte adı en çok anılan modellerden biriside **DLINQ(Database Language Integrated Query)** dir. DLINQ temel olarak veritabanından nesnelere indirgenen kümeler üzerinde LINQ sorgularının çalıştırılmasına izin veren bir modeldir. (Daha fazla bilgi için C# 3.0 İlk Bakışta DLINQ makalesine bakabilirsiniz) Aslında model tipik olarak entity katmanlarına dayanan bir yapıya sahiptir. Tablo, alan ve ilişki (relation) eşleştirmeleri vb... için niteliklerden (attributes) faydalanılmaktadır.

```
[Table(Name="Calisanlar")]
class Calisan
{
   [Column(Name="Id",Id=true)]
   public int Id;
```

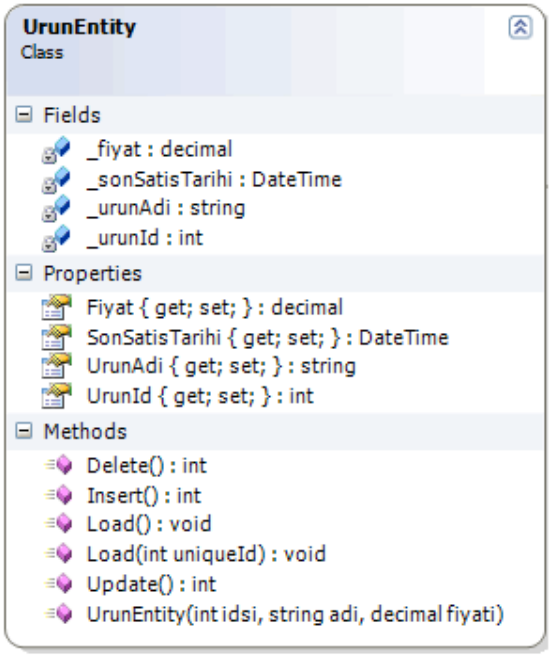
Bir Assembly Hakkında Bilgi Vermek için AssemblyInfo.cs İçeriğinin Değişitirlmesi

Geliştirdiğimiz uygulamaların ürettiği assembly' lara ait genel bilgileri AssemblyInfo.cs dosyası içerisindeki assembly seviyesinde kullanılabilen nitelikler sayesinde metadata içerisine alabiliriz. Bu sayede geliştirdiğimiz Assembly' ın hangi kültüre destek verdiğini(**Culture**), versiyonunu(**Version**), varsa **strong key** bilgisini, başlığını (**Title**), açıklamasını (**Description**) belirtebiliriz. Bu tip bilgiler metadata içerisine alındıktan sonra örneğin **ClickOnce** gibi mimariler tarafından kullanılıp **setup** sayfalarının oluşturulması sırasında kullanılabilir.

Gördüğünüz gibi, nitelikler çalışma zamanında bir takım uygulama parçaları tarafından değerlendirilmekte ve buna göre sonuçlar üretilmektedir.

Bu kısa bilgilerden sonra gelin kendi niteliklerimizi **(Custom Attributes)** nasıl yazabileceğimize bakalım. Kendi niteliklerimizin kıymetlenebilmesi için onları ele alacak bir modelede ihtiyacımız olacaktır. Burada devreye **yansıma(Reflection)** girecek. örneğin

Sql Server 2005 ile birlikte gelen AdventureWorks veritabanındaki Production şemasında (Schema) yer alan Product tablosunun programımız içerisinde bir tip ile ifade edildiğini düşünebiliriz. Bu tip için gerekli insert, update, delete ve select işlemlerinin bu tip içerisindeki metodlar ile yapılmak istendiğini düşünelim. Bu durumda yansımadan faydalanarak özelliklerin adlarından ve o anki değerlerinden yararlanıp bizim için gereken sorguları otomatik olarak hazırlatabiliriz. Ancak dikkat edilmesi gereken noktalar vardır. örneğin ProductId alanı identity tipindendir ve bu nedenlede otomatik olarak artmaktadır. Dolayısıyla otomatik oluşturulacak insert sorgusuna dahil edilmemesi gerekir. Peki çalışma zamanında bu alanı işaret eden sınıf özelliğinin, insert sorgusuna dahil edilmemesi gerektiğini nereden bilebiliriz? İşte bu özellikler için yazacağımız bir nitelik yardımıyla çalışma zamanında davranış değiştirilmesini sağlayabiliriz. Gelin ne demek istediğimiz örnek üzerinden incelemeye çalışalım. Bu amaçla öncelikli olarak bir Product nesnesini temsil edecek bir sınıf tasarlayacağız. Amacımız insert, update, delete ve select sorgularının çalışıp çalışmadığını kontrol etmekten ziyade, bunların oluşturulması sırasında niteliklerin değerini anlamak olduğundan sadece bir kaç temel özelliğin sınıfa dahil edildiğini hatırlatalım. UrunEntity isimli sınıfımızın genel tasarımı şu şekilde olacaktır.

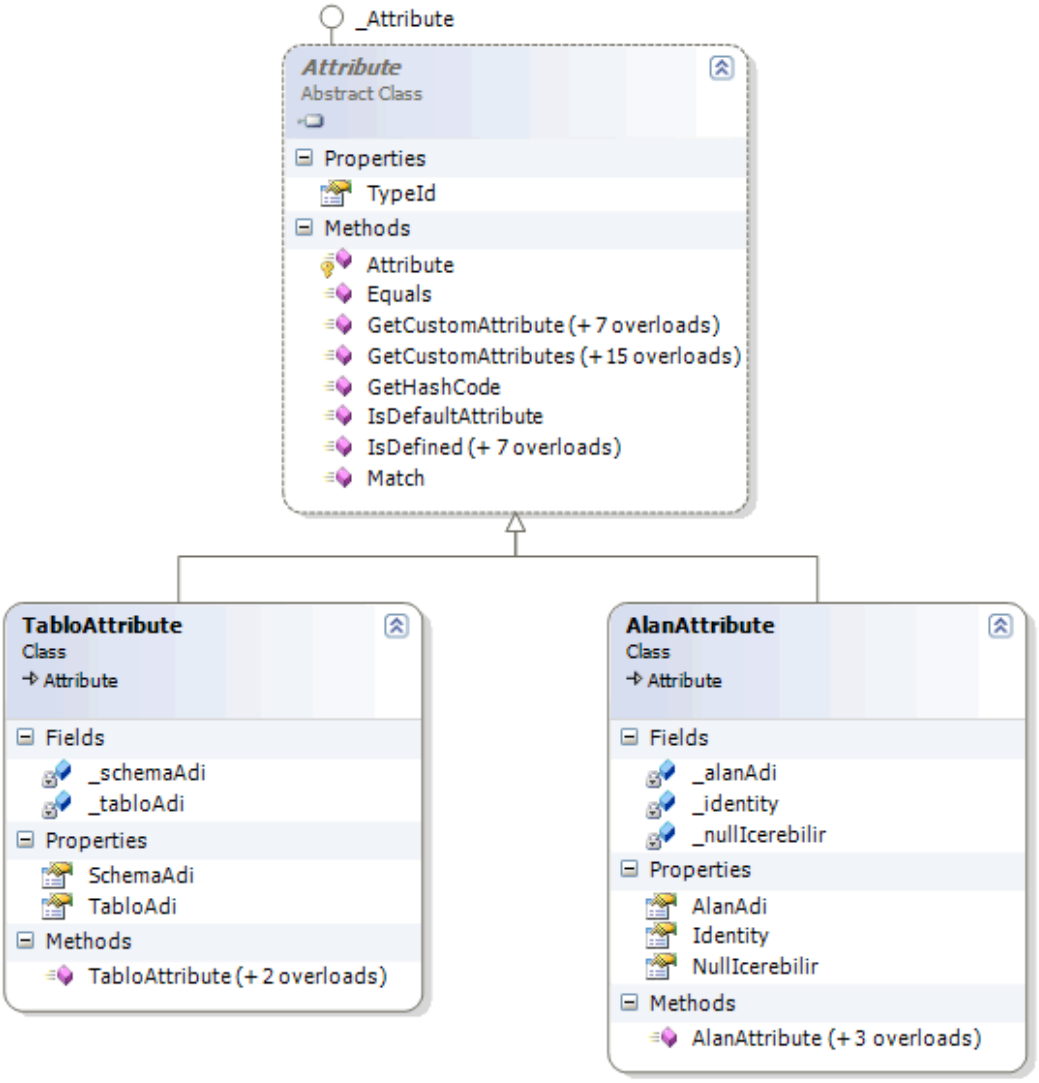


Sınıfımız içerisindeki niteliklerin uygulanmasını ve metodlarımızı ilerleyen kısımlarda geliştireceğiz. Gelelim niteliklerimize. Makalemizin başındada belirttiğimiz gibi bir nitelik mutlaka **Attribute** sınıfından türemelidir ki metadata içerisine eklenebilsin. Bu nedenle sınıfımızın eşleştiği tablo ve kolonları için kullanılacak **TabloAttribute** ve **AlanAttribute** sınıflarını Attribute sınıfından türeterek geliştireceğiz.

NOT : *İsimlendirme standartları oldukça önemlidir. Bu tüm geliştiricilerin aynı tarzda kodlama yapmasını ve kooridanasyon kolaylığını sağlar. örneğin tüm Exception sınıflarının adları* ***Exception*** *kelimesi ile biter veya tüm arayüzlerin (****interfaces****)*

*adları **I** harfi ile başlar. Benzer durum nitelikler içinde geçerlidir. öyleki nitelik sınıflarının adlarıda **Attribute** kelimesi ile bitmektedir. Bu nedenle kendi niteliklerimizi isimlendirirken adlarının Attribute kelimesi ile bitmelerine özen gösterilmelidir.*

Niteliklerimize ait sınıf diagramı ve kodlarımız ise aşağıdaki gibidir.



TabloAttribute.cs;

```
// TabloAttribute isimli niteliğimiz sadece sınıf veya yapılara uygulanabilecektir.
[AttributeUsage(AttributeTargets.Class|AttributeTargets.Struct)]
class TabloAttribute:Attribute
{
   private string _tabloAdi;
   private string _schemaAdi;

   public string TabloAdi
```

```
    {
        get { return _tabloAdi; }
        set { _tabloAdi = value; }
    }
    public string SchemaAdi
    {
        get { return _schemaAdi; }
        set { _schemaAdi = value; }
    }

    public TabloAttribute(string tablonunAdi, string schemaninAdi)
    {
        TabloAdi = tablonunAdi;
        SchemaAdi = schemaninAdi;
    }
    public TabloAttribute(string tablonunAdi)
        : this(tablonunAdi, "dbo")
    {
    }
    public TabloAttribute()
    {
    }
}
```

AlanAttribute;

```
[AttributeUsage(AttributeTargets.Property)]
class AlanAttribute:Attribute
{
    private string _alanAdi;
    private bool _identity;
    private bool _nullIcerebilir;

    public bool NullIcerebilir
    {
        get { return _nullIcerebilir; }
        set { _nullIcerebilir = value; }
    }

    public string AlanAdi
    {
        get { return _alanAdi; }
        set { _alanAdi = value; }
    }
    public bool Identity
```

```
    {
        get { return _identity; }
        set { _identity = value; }
    }

    public AlanAttribute(string alaninAdi, bool identityMi, bool nullIcerirmi)
    {
        AlanAdi = alaninAdi;
        Identity = identityMi;
        NullIcerebilir = nullIcerirmi;
    }
    public AlanAttribute(string alaninAdi, bool identityMi)
        : this(alaninAdi, identityMi, true)
    {
    }
    public AlanAttribute(string alaninAdi)
        : this(alaninAdi, false)
    {
    }
    public AlanAttribute()
    {
    }
}
```

Şu andaki amacımız kendi niteliklerimizi nasıl yazacağımızı görmek olduğundan tam anlamıyla bir entity tipi oluşturmayı hedeflemiyoruz. Bu nedenle TabloAttribute isimli sınıfımız temel olarak eşleştirme amacıyla tablo adı ve bulunduğu şema adını taşıyacak özelliklere sahip. Benzer şekilde AlanAttribute isimli sınıfımızda alan adını, null değer taşınabilip taşınamıyacağını, alanın identity tipinde olup olmadığını belirten özellikler içermektedir. Gördüğünüz gibi Attribute' tan türettiğimiz sınıfların normal sınıflardan farklı bir yazım tarzı bulunmamaktadır. Ancak dikkat ederseniz yazmış olduğumuz nitelik sınıflarımıza **AttributeUsage** isimli başka bir nitelik daha uygulanmaktadır. Bu niteliğin amacı, ilgili niteliğin hangi seviyelere uygulanabileceğini belirlemektir. Bu seviylerin belirtilmesi içinse **AttributeTargets** isimli bir enum sabitini ele almaktadır. örneğin TabloAttribute niteliğimizi sadece sınıf(class) ve yapılara(struct) uygulayabilirken, AlanAttribute isimli niteliğimizi sadece özelliklere(Property) uygulanabilir. Böylece ilgili niteliğin sadece belirtilen tip veya üyelere uygulanabilmesi adına bir zorlama getirilmiş olunur. AttributeTargets isimli enum sabitinin alabileceği tüm değerler ve kısa açıklamaları aşağıdaki tabloda görüldüğü gibidir.

Değer	**Açıklama**
All	Nitelik istenilen tipe veya üyeye uy
Assembly	Nitelik sadece assembly seviyesind

Class	Nitelik sadece sınıflara uygulanabil
Constructor	Nitelik sadece yapıcı metoda uygul
Delegate	Nitelik sadece temsilci tipine uygul
Enum	Nitelik sadece enum sabitine uygul
Event	Nitelik sadece olaya uygulanabilir.
Field	Nitelik sadece alana uygulanabilir.
GenericParameter	Nitelik sadece generic bir parametr
Interface	Nitelik sadece arayüze uygulanabili
Method	Nitelik sadece metoda uygulanabili
Module	Nitelik sadece modül' e uygulanabil olmayışıdır. Yani kastedilen .dll ve
Parameter	Nitelik sadece parametreye uygular
Property	Nitelik sadece özelliğe uygulanabil
ReturnValue	Nitelik sadece dönüş tipine uygular
Struct	Nitelik sadece bir değer türüne bir b

Şimdi bu nitelikleri UrunEntitiy sınıfı içerisinde kullanmaya çalışalım. İlk olarak sınıfımızı aşağıdaki gibi geliştirelim.

```
[Tablo(SchemaAdi="Production",TabloAdi="Product")]
class UrunEntity
{
   private int _urunId;
   private decimal _fiyat;
   private string _urunAdi;
   private DateTime _sonSatisTarihi;

   [Alan(AlanAdi = "ProductID", Identity = true, NullIcerebilir = false)]
   public int UrunId
   {
      get { return _urunId; }
      set { _urunId = value; }
   }

   [Alan("Name", false, false)]
   public string UrunAdi
   {
```

```
        get { return _urunAdi; }
        set { _urunAdi = value; }
    }
    [Alan("ListPrice", Identity = false, NullIcerebilir = false)]
    public decimal Fiyat
    {
        get { return _fiyat; }
        set { _fiyat = value; }
    }

    [Alan("SellStartDate", false, true)]
    public DateTime SonSatisTarihi
    {
        get { return _sonSatisTarihi; }
        set { _sonSatisTarihi = value; }
    }

    public UrunEntity(int idsi, string adi, decimal fiyati)
    {
        UrunId = idsi;
        UrunAdi = adi;
        Fiyat = fiyati;
    }
    public UrunEntity()
    {
    }
}
```

Gördüğünüz gibi UrunEntity sınıfına ve UrunId,UrunAdi, Fiyat ve SonSatisTarihi isimli özelliklerimize TabloAttribute ve Alan Attribute niteliklerimiz uygulanmıştır. Buna göre UrunEntity sınıfının aslında Production şemasındaki Product tablosuna işaret ettiğini anlayabiliriz. Ya da, UrunId isimli özelliğin ProductId isimli alana işaret ettiğini, null değer içeremeyeceğini ve en önemliside Identity bir alan olduğunu anlayabiliriz. Böylece reflection tekniklerini kullanan kodlarımız insert sorgusunu oluştururken ProductId alanını hesabe katmayacağını anlayabilecektir. Elbette bunun geliştirici tarafından kodlanması gerektiğinide unutmayalım. Diğer özellikler içinde benzer uygulamalar yapılmıştır. AlanAttribute ve TabloAttribute isimli sınıflarımız içersinde birden fazla aşırı yüklenmiş yapıcı metod (constructor) kullandığımızdan, niteliklerimizi söz konusu üyelere farklı biçimlerde uygulayabiliriz. Bunlardan birisi **Name=Value** ataması şeklinde olan versiyondur. Bu versiyon doğrudan **public** olan özelliklere(Property) değer atanabilmesini sağlar. Yani özel olarak aşırı yüklenmiş yapıcı metodlar olmasada ilgili niteliğin özellikleri değiştirilebilir.

Artık bundan sonra niteliklerimizi ele alacağımız kodlarımızı yazmamız gerekmektedir. Bu basit kod parçası ile, çalışma zamanında var olan nitelikleride nasıl okuyabileceğimizi ve buna göre nasıl davranış değiştirebileceğimizi görmüş olacağız. Bu amaçla UrunEntity isimli sınıfımızın Insert metodunu aşağıdaki gibi geliştirelim.

```
public int Insert()
{
   Type tip = this.GetType();
   TabloAttribute tblAtr =
((TabloAttribute[])tip.GetCustomAttributes(typeof(TabloAttribute), false))[0];
   string tabloAdi=tblAtr.TabloAdi;
   string schemaAdi =tblAtr.SchemaAdi;
   StringBuilder insertBuilder = new StringBuilder();
   insertBuilder.Append("Insert into ");
   insertBuilder.Append(schemaAdi);
   insertBuilder.Append(".");
   insertBuilder.Append(tabloAdi);
   insertBuilder.Append(" (");

   // Insert sorgusundaki alan adları çekiliyor.
   foreach (PropertyInfo prp in tip.GetProperties())
   {
      AlanAttribute
atr=((AlanAttribute[])prp.GetCustomAttributes(typeof(AlanAttribute), false))[0];
      if (!atr.Identity)
      {
         string alanAdi = atr.AlanAdi;
         insertBuilder.Append(alanAdi);
         insertBuilder.Append(",");
      }
   }
   // Son eklenen virgülü kaldırmak için.
   insertBuilder.Remove(insertBuilder.Length-1, 1);
   insertBuilder.Append(") Values (");

   // insert sorgusundaki değerleri çekiliyor.
   foreach (PropertyInfo prp in tip.GetProperties())
   {
```

```
        AlanAttribute
atr=((AlanAttribute[])prp.GetCustomAttributes(typeof(AlanAttribute), false))[0];
        if (!atr.Identity)
        {
            object alanDegeri = prp.GetValue(this, null);
            if ((prp.PropertyType.Name == "String")
                || (prp.PropertyType.Name == "DateTime"))
                    insertBuilder.Append("'" + prp.GetValue(this, null).ToString() + "',");
            else
                insertBuilder.Append(prp.GetValue(this, null).ToString() + ",");
        }
    }
    insertBuilder.Remove(insertBuilder.Length - 1, 1);
    insertBuilder.Append(")");

    //Insert işlemi için gerekli sorgula çalıştırılır.

    return 0;
}
```

Kod parçası biraz arap saçına dönmüş olabilir. Gelin ne yaptığımıza ve nitelikleri çalışma zamanında nasıl ele alabildiğimize yakından bakalım. Insert metodunun amacı, UrunEntity sınıfı için gerekli olan insert sql sorgu cümlesini otomatik olarak oluşturmaktır. Bu işlemin çalışma zamanında yapılmasını hedeflediğimizden yoğun olarak reflection işlemleri kullanılmaktadır. Bununla birlikte otomatik olarak artan, bir başka deyişle identity tipinde olan bir alanın insert sorgusuna dahil edilmemesi gerekmektedir. öyleyse bu bilgiyi içeren nitelik (attribute) ele alınmalıdır. Benzer şekilde özelliklerin hangi tablo alanlarına denk geldiği veya sınıfın hangi şemadaki hangi tabloya denk geldiği bilgileride niteliklerimizden alınmalıdır. Bu ihtiyaçlar doğrultusunda geliştirilen kodlar göz önüne alındığında herhangibir tipin çalışma zamanında uygulanan niteliğini elde etmek amacıyla **GetCustomAttributes** isimli metod kullanılmaktadır. Bu metod ilk parametre olarak elde edilmek istenen niteliğin tipini alır. Metodun belkide en önemli özelliği geriye **object** tipinden bir dizi döndürüyor oluşudur. Dönen bu dizi içerisinden niteliğe ait özellikleri çekebilmek için bir dönüştürme (cast) işlemi yapılmalıdır. **GetCustomAttributes** metodunun geriye dizi döndürmesinin sebebi, bir tipe veya üyeye birden fazla niteliğin uygulanabilecek olmasıdır. Elde edilen dizi tekrardan uygun nitelik(Attribute) tipine dönüştürüldüğünde indisleme operatörü sayesinde okunmak istenen özelliklere erişilebilir. Ki örneğimizde 0 indisli referanslar çekilmiştir. Insert metodunda kullandığımız aşağıdaki kod parçasında UrunEntity sınıfına uygulanan **TabloAttribute**' tipinin referansı yakalanmaktadır.

```
TabloAttribute tblAtr =
((TabloAttribute[])tip.GetCustomAttributes(typeof(TabloAttribute), false))[0];
```

Burada kullanılan **GetCustomAttributes** metodu tipe aittir. Sınıf içerisindeki üyelere uygulanan nitelikleri elde etmek içinde aynı yol kullanılır. Nitekim, nitelik(attribute) uygulanabilen tüm üyelerin GetCustomAttributes metodu bulunmaktadır. Söz gelimi, UrunEntity sınıfındaki özelliklere(properties) uygulanan nitelikleri çalışma zamanında elde edebilmek için **GetProperties** metodu ile gezilen**PropertyInfo** referanslarına **GetCustomAttributes** metodu aşağıdaki gibi uygulanmıştır.

AlanAttribute atr=((AlanAttribute[])prp.GetCustomAttributes(typeof(AlanAttribute), false))[0];

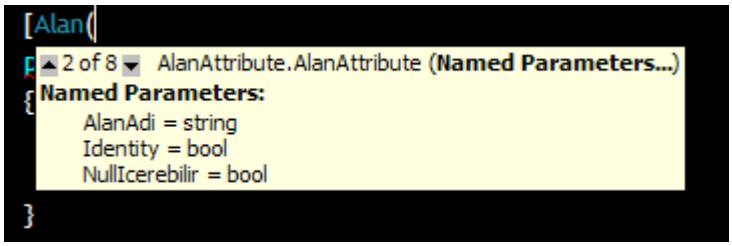
Bu şekilde nitelik referansları elde edildikten sonra söz konusu niteliğin üyelerinin değerlerine bakılabilir.

NOT : *Her ne kadar makalemizin konusu nitelikleri yazmak olsada* ***reflection*** *ile ilgili bazı noktalara da değinmek gerekir. örneğin Insert metodu içerisinde çalışma zamanında o anki UrunEntity nesne örneğinin özelliklerinin değerlerinin elde edilmesi için,* ***GetValue*** *isimli metod kullanılmıştır. Bu metodun ilk parametresi, değerleri taşıyan nesne örneğinin referansıdır.*

örneğimizi herhangibir program içerisinde test etmek için aşağıdaki gibi bir kod parçasından faydalanabiliriz. Bu amaçla örnek bir Console uygulamasını test programı amacıyla kullanabiliriz. Tek yapmamız gereken bir UrunEntity nesne örneği oluşturmak sonrasında ilgili özelliklerinde bazı değerler atamak ve Insert metodunu çağırmaktır.

```
UrunEntity urn = new UrunEntity();
urn.UrunAdi = "Pentium CPU";
urn.Fiyat = 90;
urn.SonSatisTarihi = DateTime.Now.AddDays(30);
urn.Insert();
```

Amacımız nitelikleri kavramak olduğu için, Insert metodunun içerisinde insert sorgusunu çalıştırmak için gereken Data Access Layer çağrıları yazılmamıştır. Ancak sonuçları görmek adına çalışma zamanında Insert metodunun çağırıldığı satıra bir **breakpoint** koyarak adım adım **(step into)** ilerlemekte fayda vardır. Bunun sonucunda aşağıdaki ekran görüntüsünde olduğu gibi Insert sorgusunun doğru bir şekilde oluşturulduğunu görebiliriz. Dikkat ederseniz UrunId özelliği hiç bir şekilde hesaba katılmamıştır. Ayrıca özelliklerin tabloda karşılık olan adlarına bakılarak bu sorgu cümlesi oluşturulmuştur.



Program kodumuzdan üretilen **assembly** içerisine **ildasm.exe** aracı yardımıyla bakmakta fayda vardır. Bunu yaptığımızda yazdığımız niteliklerin o anki bilgileri ile birlikte assembly' ın metadata' sına eklendiğini görebiliriz. *(Ildasm aracında metadata' yı görebilmek için **Ctrl+M** tuş kombinasyonunu kullanırız.)* örnek olark UrunId isimli özelliğimiz için eklenen nitelik metadata içerisinde aşağıdaki şekilde görünecektir.

AlanAttribute' unun Metadata izi;

Property #1 (17000006)

Prop.Name : **UrunId** (17000006)
Flags : [none] (00000000)
CallCnvntn: [PROPERTY]
hasThis
ReturnType: I4
No arguments.
DefltValue:
Setter : (06000015) set_UrunId
Getter : (06000014) get_UrunId
0 Others
CustomAttribute #1 (0c000011)

CustomAttribute Type: 06000013
CustomAttributeName: **AttributeTemelleri.AlanAttribute** :: instance void .ctor()
Length: 54
Value : 01 00 03 00 54 0e 07 41 6c 61 6e 41 64 69 09 50 > T **AlanAdi P**<
: 72 6f 64 75 63 74 49 44 54 02 08 49 64 65 6e 74 >**roductIDT Ident**<
: 69 74 79 01 54 02 0e 4e 75 6c 6c 49 63 65 72 65 >**ity T NullIcere**<
: 62 69 6c 69 72 00 >**bilir** <
ctor args: ()

Burada açık bir şekilde niteliğin(attribute) özelliklerine atanan değerler de görülebilmektedir. Benzer şekilde UrunEntity isimli sınıfımıza uygulanan TabloAttribute niteliğinin metadata içerisine yaptığı katkıyıda görebiliriz.

TabloAttribute' unun Metadata izi;

CustomAttribute #1 (0c000010)

CustomAttribute Type: 06000009
CustomAttributeName: **AttributeTemelleri.TabloAttribute** :: instance void .ctor()
Length: 46
Value : 01 00 02 00 54 0e 09 53 63 68 65 6d 61 41 64 69 > **T SchemaAdi**<
: 0a 50 72 6f 64 75 63 74 69 6f 6e 54 0e 08 54 61 > **ProductionT Ta**<
: 62 6c 6f 41 64 69 07 50 72 6f 64 75 63 74 >**bloAdi Product** <
ctor args: ()

Yine gördüğünüz gibi SchemaAdi özelliğine atanan Production değeri ile TabloAdi' na atanan Product değerleri buraya **Value** olarak alınmıştır.

Program kodlarımızı daha da geliştirmek siz değerli okurlarımızın elindedir. örneğin, Insert, Update, Delete ve Load metodların geliştirebilir bunları gerekirse bir base sınıf içerisinde toplanabilir. Böylece geldik bir makalemizin daha sonuna. Bu makalemizde, nitelikleri(attribute) daha yakından tanımaya çalıştık ve kendi niteliklerimizimi (Custom Attributes) nasıl yazabileceğimizi incelemeye çalıştık. Gördüğünüz gibi niteliklerde birer sınıf olarak düşünüldüklerinde geliştirilmeleri son derece kolay tiplerdir. Ne varki niteliklerin asıl gücü çalışma zamanında reflection kullanıldığında ortaya çıkmaktadır. Tekrardan hatırlatmak gerekirse, amaç çalışma zamanında tiplere ve üyelere nasıl davranılacağına dair kararların verilmesinde assembly içerisindeki ekstra metadata bilgilerinde faydalanılmasıdır. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

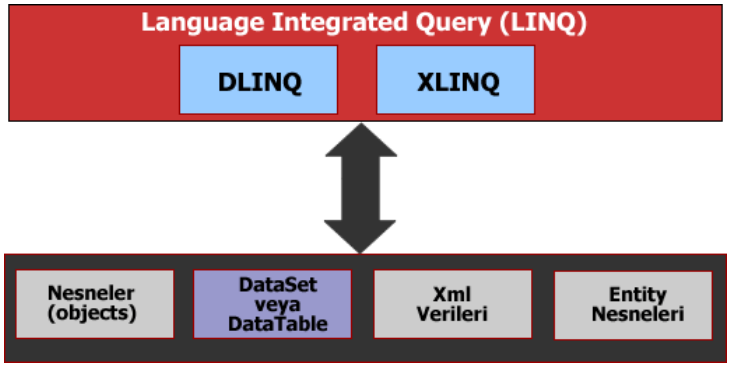
AttributeKullanimi.rar (47,50 kb)

Bağlantısız Katmanda LINQ (2007-04-02T20:44:00)

linq,

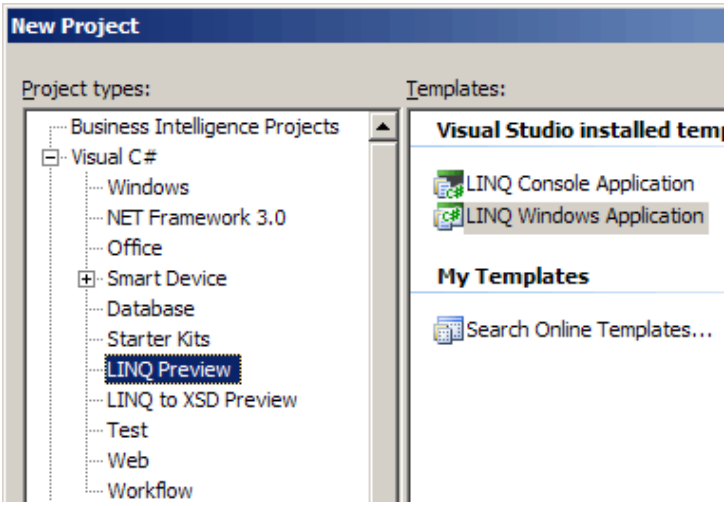
Language Integrated Query (Dil ile tümleştirilmiş sorgu) yardımıyla yapabileceklerimiz saymakla bitmiyor. Aslında LINQ projesinin en önemli çıkış nedeni, **Anders Hejslberg'** ın anlatımıyla veri ve nesne eşitsizliğidir. **(data!=objects)** Bu ifadeyi, TechEd 2006 sunumlarında kullanan Anders Hejslberg, özellikle veri yapılarının programlama ortamına alınması sonrasında, var olan basit sorgu tekniklerinin uygulanamayışından yakınmaktadır. LINQ projesinin aslında en temel amacı, uygulamaların çalışma alanlarında (.Net perspektifinden baktığımızda Application Domain' ler içerisinde), bellek üzerinde konuşlanan nesneler üzerinden bildiğimiz veri sorgulama kurallarını uygulayabilmektir. Bir başka deyişle, nesne(object) üzerinde, var olan veritabanı nesnelerini taşıyabilen **Entity** bileşenleri üzerinde, belleğe alınan **Xml** veri setleri üzerinde, sorgulamaları bilinen alışılagelmiş söz dizimleri ile tek bir standart altında yapabilmektir.

Tüm bu farklı nesnel yapıların ortak bir sorgulama dilini kullanabiliyor olması da LINQ projesinin ana fikirlerinden birisidir aslında.



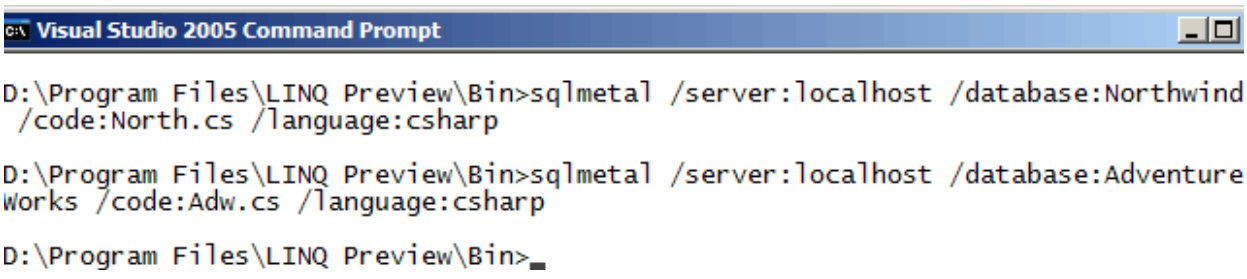
Yukarıdaki grafikte, LINQ projesinin odaklandığı temel modeller ifade edilmeye çalışılmıştır. LINQ sorguları bildiğiniz gibi bellek üzerinde herhangibir şekilde **IEnumerable**arayüzünü uyarlamış olan her tür nesne topluluğuna uygulanabilmektedir. Bu nedenle bellek içi nesnelerden **(in memory objects)**, veritabanı(database) bağlantılı Entity nesnelerine kadar pek çok yerde kullanılabilmektedir.

C# 3.0 ve geleceği ile ilgili olarak önceki makalelerimizde, DLINQ, XLINQ modellerini incelemeye çalışmıştık. Bunların yanında LINQ ile yapabileceklerimizi daha derinlemesine kavrayabilmek maksadıyla bol bol sorgu geliştirdik. Bugünkü makalemizde ise, özellikle **bağlantısız katman (disconnected layer)** nesneleri üzerinde, yani bildiğimiz **DataSet** ve **DataTable** nesne örnekleri üzerinde LINQ sorgularını nasıl yazabileceğimizi basit bir şekilde incelemeye çalışacağız. DataSet ve DataTable gibi bileşenler bildiğiniz gibi herhangibir veri kaynağından yüklenen sonuç kümelerini uygulama belleğinde tutmak amacıyla kullanılmaktadır. Ne varki çalışma zamanında, bağlantısız katman nesneleri üzerindeki verilerde sorgulama yapabilmek için çeşitli yollara başvurmamız gerekir. örneğin bunlardan birisi **Select** metodudur. Bir başka teknikte veri kümelerini **DataView** bileşenlerine alıp filtreleme amacıyla yardımcı fonksiyonellerden faydalanmaktır. LINQ, felsefe olarak yukarıda bahsettiğimiz tüm veri kümeleri için ortak bir sorgulama ortamı sunmaktadır. öyleyse bağlantısız katman nesneleri içinde bu tekilleştirilmiş sorgulama modelini nasıl ele alabiliriz?Dilerseniz hiç vakit kaybetmeden örneğimize başlayalım. Bu seferki örneğimizi **LINQ Windows Application** projesi olarak geliştireceğiz. Nitekim, DataTable içeriğini ekranda görsel olarak ele alabileceğimiz bir ortam olayları daha net algılayabilmemizi sağlayacaktır. Elbetteki bu makalede bahsedilen işlemleri gerçekleştirebilmek için sistemimizde L**INQ Preview** sürümünün yüklü olması gerektiğini unutmayalım.



Herhangibir DataTable üzerinden LINQ sorguları çalıştırabilmemiz için **System.Data.Extensions** isimli kütüphanenin program içerisinde referans edilmiş olması yeterlidir. çalışmakta olduğumuz LINQ Windows uygulaması bu referansı varsayılan olarak içermektedir.

Herşeyden önce uygulamamızın bellek üzerinde DataSet ve DataTable nesnelerine sahip olması gerekiyor. Bu amaçla makalemizde **AdventureWorks** ve **Northwind** veritabanlarından yararlanacağız. DataTable nesnelerimizi doldurmak için başvurabileceğimiz iki yol var. Bunlardan birisi, standart Ado.Net tiplerinden ve fonksiyonelliklerinden yararlanmak. Bir başka deyişle, **DataAdapter** tipi ve **Fill**metodundan bahsediyoruz. Ancak bu kez biraz daha farklı olarak entity nesnelerinden faydalanacağız. Hatırlarsanız **DLINQ** konusunu incelediğimiz makalemizde, bir database ve içerisindeki tablolar için otomatik olarak entity hazırlayabilmemizi sağlayan **SqlMetal** isimli bir aracın LINQ Preview projesi ile birlikte geldiğinden bahsetmiştik. Bizim için gereken Entity sınıflarını oluşturması için SqlMetal aracını aşağıdaki gibi kullanıp üretilen .cs dosyalarını projemize eklememiz yeterli olacaktır. *(SqlMetal aracına, LINQ Preview' u kurduktan sonra, varsayılan olarak D:\Program Files\LINQ Preview\Bin adresinden ulaşabilirsiniz.)*



Dolayısıyla artık entity nesneleri üzerinden DataTable nesne örnekleri içerisine veri doldurma işlemini gerçekleştirebiliriz. Yazacağımız ilk kod parçası, AdventureWorks veritabanı içerisinde yer alan Production şemasındaki Product tablosundan bazı satırların bir DataTable içerisine LINQ sorguları yardımıyla alınması işlemini gerçekleştirecektir. Bu

amaçla aşağıdaki kod parçasında olduğu gibi AdventureWorks isimli sınıfımıza ait bir nesne örneği oluşturmamız gerekmektedir.

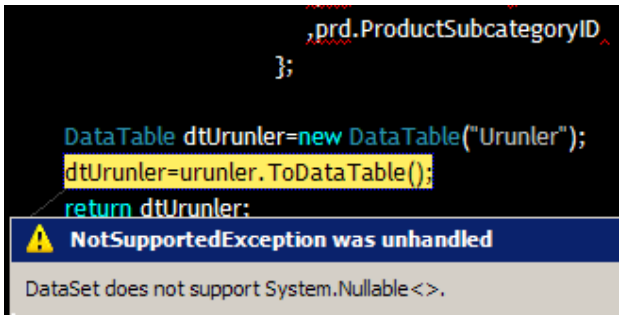
```
AdventureWorks adWorks = new AdventureWorks("data
source=localhost;database=AdventureWorks;integrated security=SSPI");
```

Artık entity sınıflarımıza ait nesne örneklerini, adWorks üzerinden kullanabiliriz. Aşağıdaki metod ile, global olarak tanımladığımız adWorks nesnesini kullanarak, yine global olarak tanımladığımız dtUrunler isimli DataTable isimli nesne örneğine veri doldurma işlemi yapılmaktadır. Biz LINQ sorgumuz içerisinden belirli alanları alıp yeni bir isimsiz tip **(anonymous type)**olarak çekmekteyiz. Elbetteki bu sorgu içerisinde bildiğimiz tüm LINQ imkanlarını kullanabiliriz. Where, order by gibi ifadeler bunlara örnek olarak verilebilir.

```
private DataTable LoadProductsTable()
{
   var urunler =from prd in adWorks.Production.Product
                select new {
                              prd.ProductID
                              ,prd.Name
                              ,prd.ListPrice
                              ,prd.Class
                              ,prd.SellStartDate
                              ,prd.SafetyStockLevel
                              ,prd.StandardCost
                            };

   DataTable dtUrunler=new DataTable("Urunler");
   dtUrunler=urunler.ToDataTable();
   return dtUrunler;
}
```

Burada ilk olarak **adWorks.Production.Product** entity nesnesi üzerinden bir LINQ sorgusu çalıştırılmaktadır. Bunun sonucunda elde edilen veri kümesini bir DataTable içerisine aktarmak için ise tek yapılması gereken **ToDataTable** isimli metodun çağırılmasıdır. *(System.Data.Extensions isim alanı, DataTable ve DataRow' lar için LINQ sorguları hazırlanmasını sağlayan pek çok genişletme metodu içermektedir.)*

***NOT :** Kendi örneklerimizi denerken dikkat etmemiz gereken bir nokta vardır. özellikle null değer alabilen sayısal ve tarihsel formatlı alanlar için LINQ sorguları aşağıdaki ekran görüntüsünde yer alan çalışma zamanı istisnasına neden olabilmektedir. örneğin Product tablosunda sayısal ve null değer alabilen bir alan olarak tanımlanmış olan ProductSubCategoryID için bu istisna mesajı elde edilmektedir.*

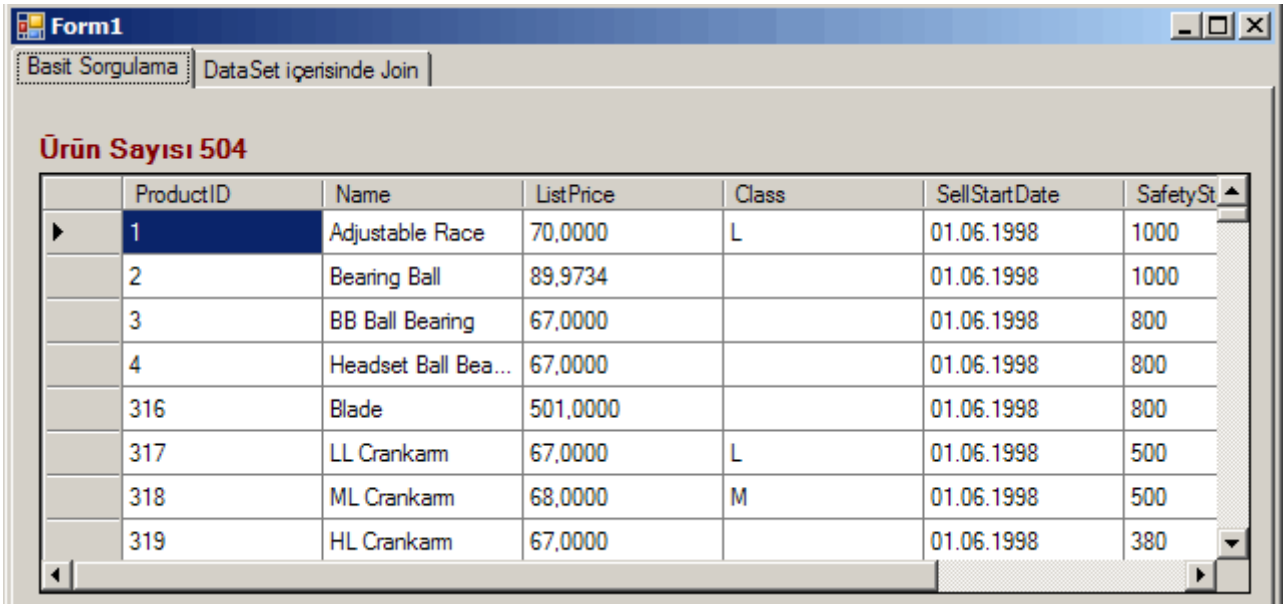


Aynı durum null değerler alabilen varchar, nvarchar tipli alanlar için geçerli değildir. Bunların program ortamı içerisinde yer alan entity sınıfları içerisinde string olarak kullanıldığına ve string' in özellikle referans tipi olduğu için null değer taşıyabildiğine dikkat edelim.

Artık elde ettiğimiz DataTable nesne örneğini herhangibir görsel taşıyıcıya (container) bağlayabiliriz. Bu amaçla .Net 2.0 ile gelen **DataGirdView** kontrolü biçilmiş kaftandır. Uygulamada bu durumu test etmek için ana formumuzun **Load** olay metodu içerisinde aşağıdaki örnek kod parçaları yazılmıştır.

```
adWorks = new AdventureWorks("data
source=localhost;database=AdventureWorks;integrated security=SSPI");

dtUrunler = LoadProductsTable();
dgUrunler.DataSource=dtUrunler;

label1.Text = "ürün Sayısı " + (dgUrunler.Rows.Count-1).ToString();
```

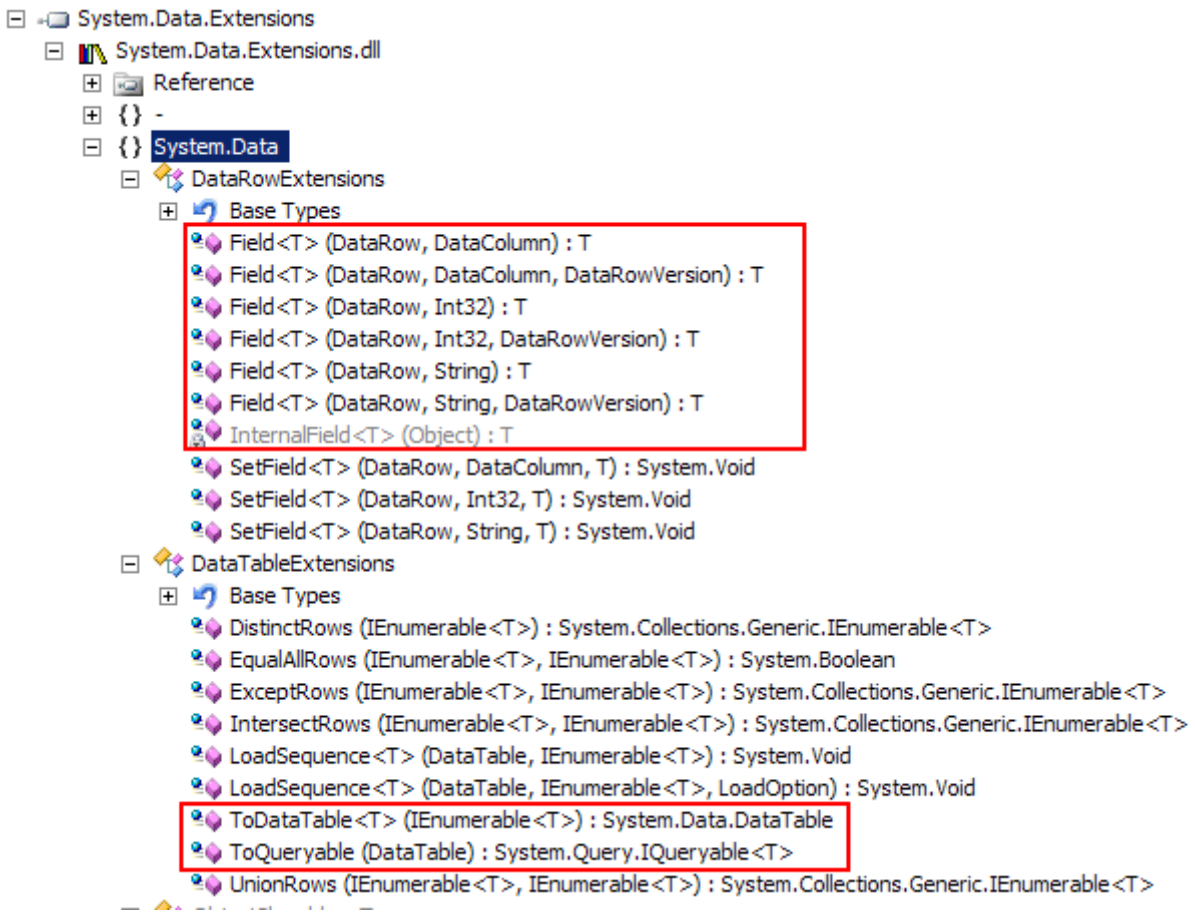
Programın çalışması sonucu aşağıdaki ekran görüntüsünü elde ederiz. Dikkat ederseniz Product tablosundan 504 adet ürün bilgisi yüklenmiştir.



Asıl amacımız elbetteki DataTable nesne örneğini doldurmak değildir. özellikle şunu tekrar belirtmekte fayda vardır. örneğimizde Entity tipleri üzerinden veri çekme işlemi yapılmıştır. Pekala bunu DataAdapter yardımıyla da gerçekleştirebiliriz. Ancak asıl yapmak istediğimiz veriyi bağlantısız katmana nasıl aldığımız değil, bellekte veri taşıyan DataTable üzerinden LINQ sorgularını nasıl çalıştırabileceğimizdir. Nitekim LINQ, DataTable veya DataSet içerisine verinin nasıl çekildiği ile ilgilenmez. Bu amaçla örneğin yukarıdaki sonuçları döndüren DataTable bileşenimizin içerisinde üretim tarihi (SellStartDate alanının değeri) bellirli bir zamandan sonra olanları bulmak istediğimizi düşünelim. Söz konusu sorgu için aşağıdaki gibi bir kod parçasını kullanabiliriz.

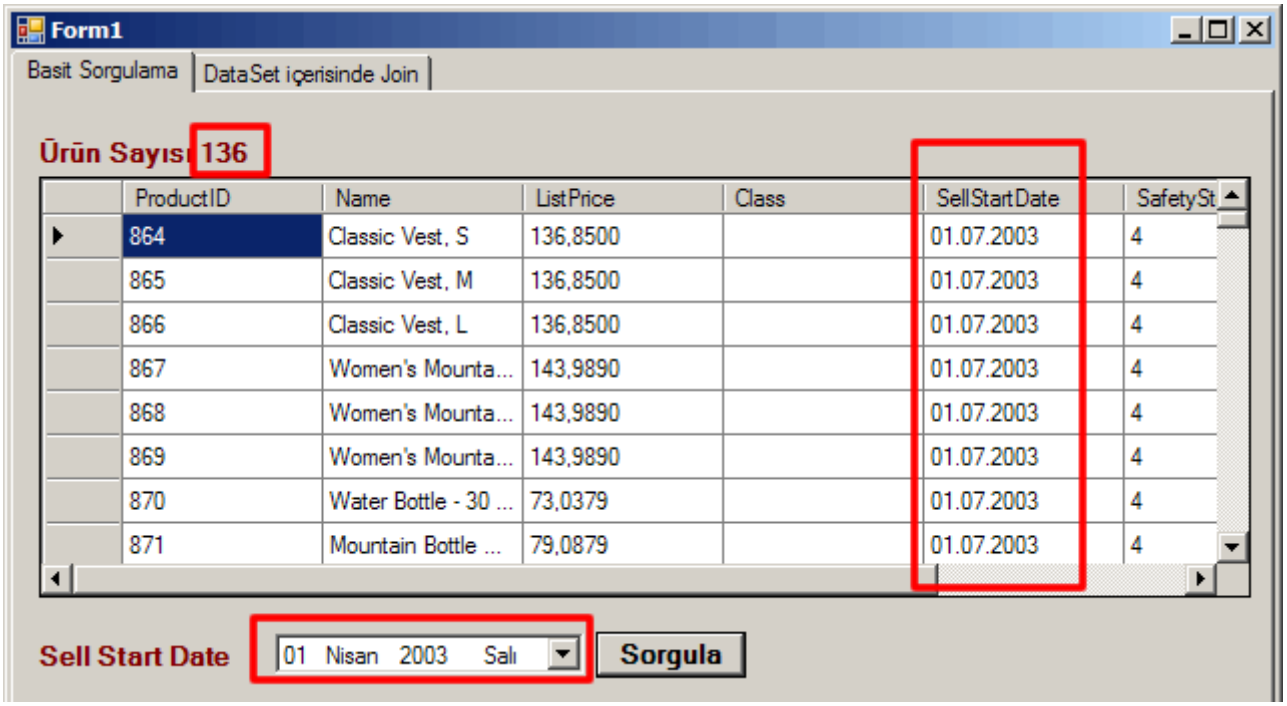
```
var sorgulanabilirUrunler = dtUrunler.ToQueryable();

var sonuclar=from prd in sorgulanabilirUrunler
             where prd.Field<DateTime>("SellStartDate")>=dateTimePicker1.Value
               select new {
                            ProductID=prd.Field<int>("ProductID")
                            ,Name=prd.Field<string>("Name")
                            ,ListPrice=prd.Field<decimal>("ListPrice")
                            ,Class=prd.Field<string>("Class")
                            ,SellStartDate=prd.Field<DateTime>("SellStartDate")
                            ,SafetyStockLevel=prd.Field<short>("SafetyStockLevel")
                            ,StandartCost=prd.Field<decimal>("StandardCost")
                          };

dgUrunler.DataSource=sonuclar.ToDataTable();

label1.Text="ürün Sayısı "+(dgUrunler.Rows.Count-1).ToString();
```

Dikkat edeceğimiz ilk nokta **ToQueryable** metodunun kullanılmasıdır. Bu metodun tek amacı DataTable üzerinde LINQ sorgularının çalıştırılabilmesini sağlamaktır. Aslında**ToQueryable**, **ToDataTable**, **Field<T>** gibi metodlar, **System.Data.Extensions.dll** içerisinde gelen genişletme metodlarıdır. Bunları görmek için her hangibir decompiler aracını kullanabiliriz. örneğiz **XenoCode Fox 2007 Community Edition** aracı yardımıyla System.Data.Extensions.dll içeriğine bakacak olursak aşağıdaki sonuçları alırız.



Gördüğünüz gibi DataTable için **ToQueryable** ve **ToDataTable** metodları, DataRow tipi için **Field<T>** metodu vb... yer almaktadır. Field<T> metodu, sorgulanabilir hale getirlmiş olan DataTable içerisindeki DataRow dizileri üzerinden istenen alanın elde edilebilmesi amacıyla kullanılmaktadır. Dikkat ederseniz generic bir metoddur ve tip olarakta, çekilen alanın veri tipini almaktadır. Bu tipin elbetteki doğru girilmesi şarttır. Aksi takdirde derleme zamanı hataları alırız.

Peki, sorgumuz tam olarak ne yapmaktadır? Tahmin edeceğiniz gibi **where** anahtar kelimesi sayesinde SellStartDate alanının değeri DateTimePicker kontrolünde seçilen tarihten sonra gelen satırlar çekilmektedir. Buradaki where cümlesinde yer alan **prd.Field<DateTime>("SellStartDate")>=dateTimePicker1.Value** ifadesinin söz konusu DataTable içerisindeki her bir **DataRow** için çalıştığını unutmayalım. Bunu daha kolay idrak edebilmek için bu tip bir gereksinimi LINQ olmadan eski usuller ile yazmak istediğinizi düşünün. Tüm satrıları gezeceğimiz bir döngü yazmamız gerektiğini tahmin edebiliriz. Sonuç itibariyle kodumuzu çalıştırdığımızda aşağıdaki veri kümesini elde ederiz.



Sorgularımızı çeşitlendirebiliriz. öyleki artık elimizdeki nesne, DataTable üzerinden elde edilmiş sorgulanabilir bir DataRow kümesinden başka bir şey değildir ve LINQ ifadelerine doğrudan destek vermektedir. Şimdi işlemlerimizi biraz daha ilerletelim. örneğin birbiriyle ilişkili olabilen iki DataTable üzerinde LINQ yardımıyla bir **Join** işlemi gerçekleştirmeye çalışalım. Bu amaçla Northwind veritabanında yer alan Order ve OrderDetails tablolarından faydalalanbiliriz. öncelikle bu tabloları entity nesnelerimize alacağız ve sonrasında ise DataTable nesne örneklerine yükleyeceğiz. Son olarakta bu iki DataTable örneğine ait sorgulanabilir bir nesne üzerinden LINQ yardımıyla bir Join işlemi gerçekleştireceğiz. Bu amaçla programımıza aşağıdaki metodları ekleyelim.

```
private DataTable LoadOrdersTable()
{
   var siparisler=from s in north.Orders
                  select new {
                              s.OrderID
                              ,s.ShipAddress
                              ,s.ShipCity
                              ,s.ShipRegion
                              ,s.ShipPostalCode
                              ,s.ShipCountry
                             };

   return siparisler.ToDataTable();
}

private DataTable LoadOrderDetailsTable()
{
```

```
   var siparisDetaylari=from d in north.OrderDetails
                        select new {
                                     d.OrderID
                                     ,d.UnitPrice
                                     ,d.Quantity
                                   };

   return siparisDetaylari.ToDataTable();

}
```

Metodlarımız sırasıyla north isimli global olarak tanımlanmış entity nesnesi üzerinden hareket ederek Orders ve OrderDetails tablolarından belleğe bazı alanlar için veri çekmektedir. Son olarak elde edilen sonuç kümeleri **ToDataTable** metodu yardımıyla geri döndürülüyor. Şimdi bu iki veri kümesininde OrderID alanları üzerinden birbirlerine bağlı olduğunu biliyoruz. Dolayısıyla birleştirme işlemini gerçekleştireceğimiz sorgu cümesinde bu durumu göz önüne almamız gerekiyor. Bu amaçla aşağıdaki gibi bir kod parçasından faydalanabiliriz.

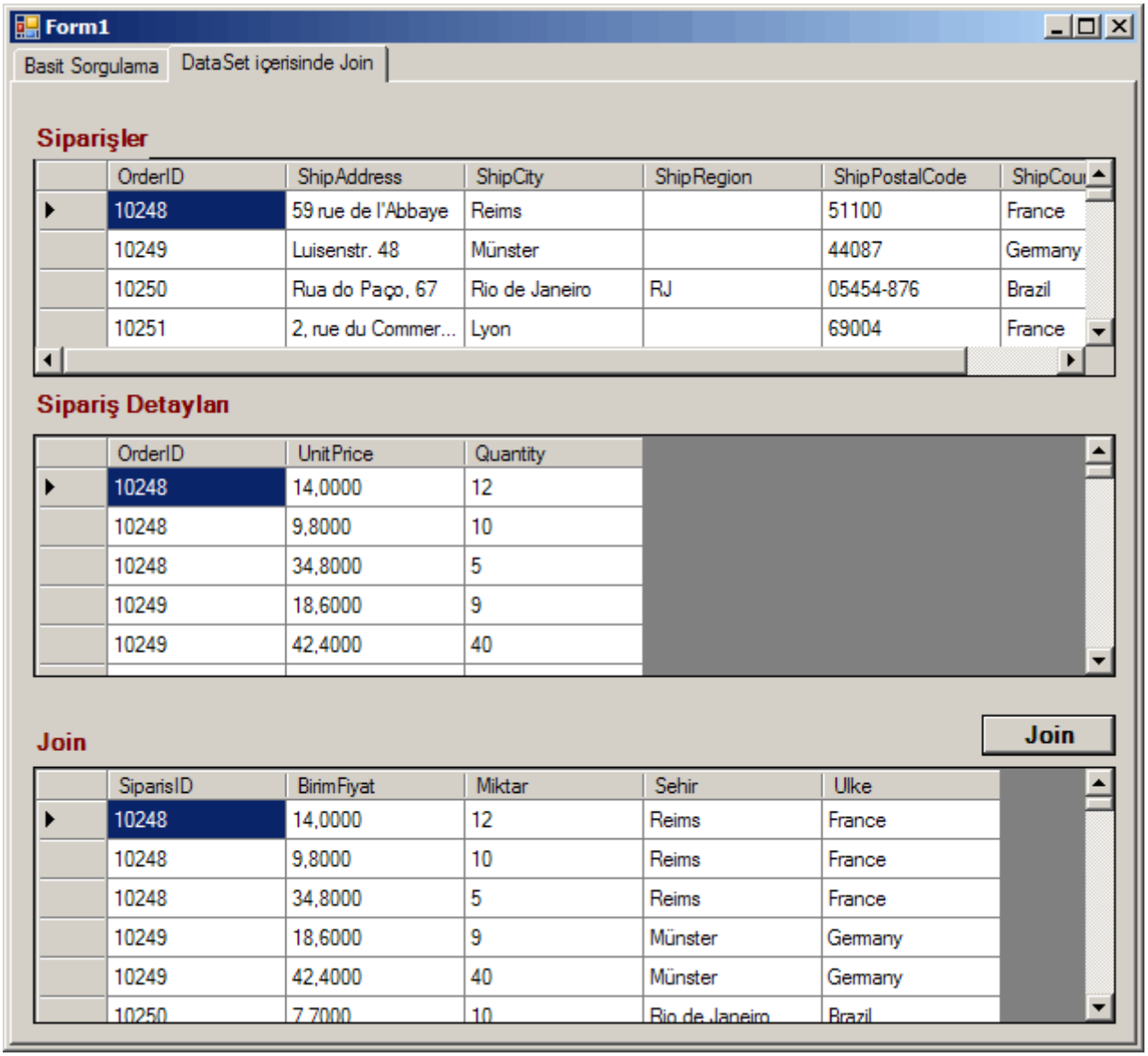
```
AdventureWorks adWorks;
Northwind north;
DataTable dtUrunler, dtSiparisler, dtSiparisDetaylari;

private void Form1_Load(object sender, EventArgs e)
{
   adWorks = new AdventureWorks("data
source=localhost;database=AdventureWorks;integrated security=SSPI");
   north = new Northwind("data source=localhost;database=Northwind;integrated
security=SSPI");

   dtUrunler = LoadProductsTable();
   dgUrunler.DataSource = dtUrunler;

   label1.Text = "ürün Sayısı " + (dgUrunler.Rows.Count - 1).ToString();

   dtSiparisler = LoadOrdersTable();
   dtSiparisDetaylari = LoadOrderDetailsTable();

   dgSiparisler.DataSource = dtSiparisler;
   dgSiparisDetaylari.DataSource = dtSiparisDetaylari;
}

private void btnJoin_Click(object sender, EventArgs e)
{
   var sorgulanabilirOrders = dtSiparisler.ToQueryable();
```

```
   var sorgulanabilirOrderDetails = dtSiparisDetaylari.ToQueryable();

   var sonuclar=from o in sorgulanabilirOrders
                join od in sorgulanabilirOrderDetails
                  on o.Field<int>("OrderID") equals od.Field<int>("OrderID")
                    select new {
                                 SiparisID=o.Field<int>("OrderID")
                                 ,BirimFiyat=od.Field<decimal>("UnitPrice")
                                 ,Miktar=od.Field<short>("Quantity")
                                 ,Sehir=o.Field<string>("ShipCity")
                                 ,Ulke=o.Field<string>("ShipCountry")
                               };

   dgJoin.DataSource=sonuclar.ToDataTable();
}
```

LINQ mimarisinde kullandığımız **Join** kalıbını burada da aynen kullanmaktayız. Tek dikkat etmemiz gereken, generic Field<T> metodunu nasıl ele aldığımızdır. **o** takma adı ile siparişleri tutan sorgulanabilir DataRow nesne dizisini (sorgulanabilirOrders), **od** takma adı ilede sipariş detaylarını tutan sorgulanabilir DataRow nesne dizisini (sorgulanabilirOrderDetails) ifade etmekteyiz. Buna göre join işlemini OrderID alanları üzerinden gerçekleştiren ifademiz aşağıdaki gibidir. Burada her iki DataRow dizisindeki ilgili alanların eşitliğine göre bir kıstas getirilmektedir.

on o.Field<int>("OrderID") equals od.Field<int>("OrderID")

Programımızı çalıştırdığımızda aşağıdakine benzer bir ekran görüntüsü ile karşılaşırız. *(TabPage' in üst tarafında yer alan iki DataGridView bileşeni, sırasıyla Orders ve OrderDetails bilgilerini göstermektedir.)*



Dilersek join ile yazmış olduğumuz sorgumuza **where** ile başka kısıtlamalarda katabiliriz. örneğin, elde edilen sonuç kümesinde **Quantity alanının değeri 10' un üzerinde** olanları elde etmek için tek yapmamız gereken sorgumuzu aşağıdaki gibi genişletmek olacaktır.

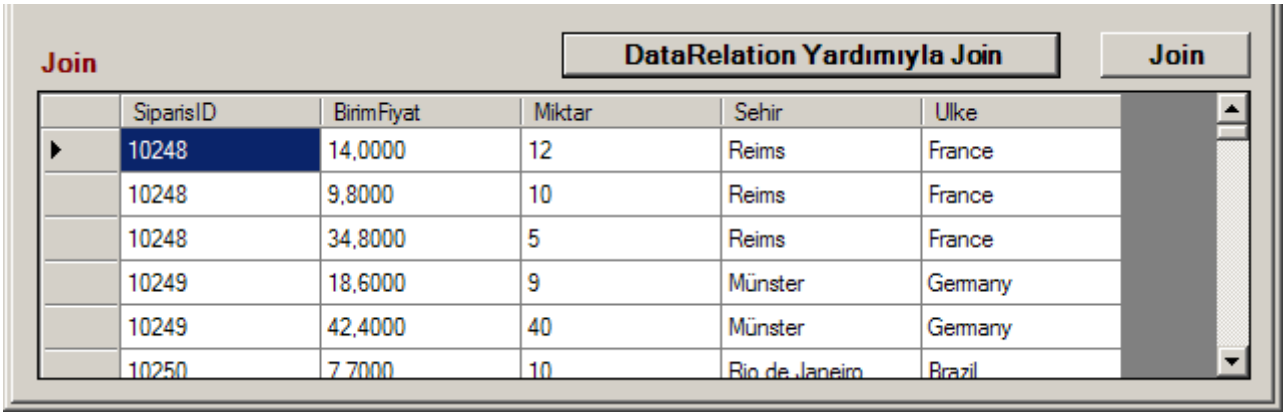
```
var sonuclar=from o in sorgulanabilirOrders
                join od in sorgulanabilirOrderDetails
                  on o.Field<int>("OrderID") equals od.Field<int>("OrderID")
                      where od.Field<short>("Quantity")>10
                  select new {
                              SiparisID=o.Field<int>("OrderID")
                              ,BirimFiyat=od.Field<decimal>("UnitPrice")
                              ,Miktar=od.Field<short>("Quantity")
                              ,Sehir=o.Field<string>("ShipCity")
                              ,Ulke=o.Field<string>("ShipCountry")
                             };
```

Where ifadesinde ilgili alanın değerinin karşılaştırma işlemine tabi tutmak için yine Field<T> generic metodundan faydalandığımızda dikkat edelim.

İstersek join ile yaptığımız birleştirme işlemini, içerisinde **DataRelation** nesnesi barındıran bir DataSet üzerinden de gerçekleştirebiliriz. Bu sefer devreye üst tablodaki herhangibir satıra bağlı alt satırların getirilmesini sağlayacak GetChildRows isimli bir fonksiyonellik gelecektir. Durumu daha iyi anlayabilmek için aşağıdaki kod parçasını göz önüne alabiliriz.

```
DataSet ds = new DataSet();
ds.Tables.Add(dtSiparisler);
ds.Tables.Add(dtSiparisDetaylari);

DataRelation drOrdToDtl = new DataRelation("OrdToDetails",
dtSiparisler.Columns["OrderID"],dtSiparisDetaylari.Columns["OrderID"]);
ds.Relations.Add(drOrdToDtl);

var sorgulanabilirOrders=dtSiparisler.ToQueryable();

var sonuclar=from o in sorgulanabilirOrders
                from od in o.GetChildRows("OrdToDetails")
                   select new {
                                SiparisID=o.Field<int>("OrderID")
                                ,BirimFiyat=od.Field<decimal>("UnitPrice")
                                ,Miktar=od.Field<short>("Quantity")
                                ,Sehir=o.Field<string>("ShipCity")
                                ,Ulke=o.Field<string>("ShipCountry")
                              };

dgJoin.DataSource=sonuclar.ToDataTable();
```

DataSet içerisinde yer alan, dtSiparisler ve dtSiparisDetaylari isimli DataTable nesnelerinin işaret ettiği veri kümeleri arasındaki ilişkimiz OrderID alanları üzerinden Orders' dan OrderDetails' e doğrudur. Bunu DataSet içerisinde tanımlayan ise Ado.Net' in ilk çıkışından beri bildiğimiz **DataRelation** nesnesidir. LINQ sorgumuz, bu nesneyi **GetChildRows** isimli metod içerisinde parametre olarak kullanmaktadır. Böylece **o** takma adı ile temsil edilen sorgulanabilirOrders içerisindeki her bir DataRow için bu ilişki kullanılabilmektedir. Bu da doğal olarak, ilişkinin diğer ucunda yer alan siparişe ait detay bilgisinin elde edilebilmesi anlamına gelmektedir. LINQ sorgumuz iki adet **from** anahtar kelimesi içerdiğinden sonuç doğal olarak bir Join sorgusunun çıktısı ile aynı olacaktır. Uygulamamızı bu haliyle çalıştırdığımızda ilk yazdığımız join sorgusundakine benzer sonuçları elde ederiz.



DataTable ve DataSet' ler üzerinde **ToQueryable, ToDataTable, Field<T>** metodları dışında, **LoadSequence, DistinctRows, EqualAllRows, UnionRows, IntersectRows, ExceptRows, SetField<T>**isimli metodlarda kullanılabilmektedir. Bu metodların temel amacı, DataTable ve DataRow gibi nesneler üzerinde LINQ tekniklerinin daha da genişletilmesini sağlamaktır. örneğin **LoadSequence**metodu sayesinde herhangibir sorgu sonucu elde edilen kümeyi bir var olan bir DataTable içerisine ilave edebiliriz. Bu metod ve diğerleri hakkında daha fazla bilgi almak için LINQ dökümantasyonundan faydalanabilirsiniz.

Böylece geldik bir makalemizin daha sonuna. Bu makalemizde, LINQ' yu DataTable gibi bağlantısız katman nesneleri üzerinde nasıl kullanabileceğimizi incelemeye çalıştık. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

örnek Uygulama İçin Tıklayınız.

Kendi WebPart Kontrolümüzü Geliştirmek - 1 (2007-03-28T20:47:00)

asp.net 2.0,

Web uygulamalarında var olan bileşenlerin yetersiz kaldığı durumlarda kendi kontrollerimizi geliştirme yoluna gidebiliyoruz. Kendi kontrollerimizi geliştirirken seçebileceğimiz yollar bellidir. Var olan bir web bileşeninden türetme yolunu seçebiliriz **(Inherited Controls)**. Bu durumda kontrolün Html çıktısının ne olacağını bir başka deyişle Render işlemlerini çok fazla düşünmemize gerek kalmaz. Tek yapmamız gereken var olan üyeleri ezmek (override) veya yeni üyeler katmaktır. Bir diğer yol birden fazla kontrolü içeren komposit bir bileşen geliştirmektir**(Composite Controls).** Bu tekniğe verilebilecek en güzel örnek kullanıcı web kontrolleridir **(web user controls)**. Özel bileşen geliştirmenin belkide en zor seçeneği, kontrolü sıfırdan yazmaktır. Bu durumda, ilgili bileşenin istemci tarayıcılarındaki Html çıktısını düşünmekle kalmayıp, **ViewState**, **Postback, Event Handling** gibi temel konularında göz önüne alınması ve düşünülmesi gerekir.

Asp.Net 2.0 ile birlikte **WebPart** adı verilen bir Framework gelmiştir. Bu Framework istersek kolay bir şekilde kişiselleştirilebilir **(Personalizable)** web kontrolleri

yazabilmemizede olanak sağlamaktadır. Aslında bahsettiğimiz özelleştirilmiş kontrollerin birer **WebPart** tipi olduğunu söylemek gerekir. Dolayısıyla, kendi WebPart bileşenlerimizi türetme yardımıyla kolay bir şekilde geliştirebilir ve kullanabiliriz. WebPart' lar sıfırdan yazılan bileşen kontrolleri ve web kullanıcı kontrollerine nazaran, WebPart Framework için tam destek sağlarlar. Bu da, kişiselleştirmenin kullanıldığı sayfalarda oldukça önemli bir meziyettir. İşte biz bu makalemizde basit olarak kendi Web Part bileşenlerimizi nasıl geliştirebileceğimizi incelemeye çalışacağız.

Normal şartlar altında sayfada yer alan herhangiri **Web Part Zone** içerisine alınan her bileşen otomatik olarak Web Part muamalesi görür. Lakin kendi Web Part bileşenlerimizi geliştirdiğimizde, daha öncedende belirttiğimiz gibi Web Part Framework' ü içerisindeki özelliklerin ve fonksiyonelliklerin tamamını kullanabilme şansına sahip oluruz. Öyleyse işe **WebPart** tipinden türeyen bir sınıf yazmak ile başlamak lazım.

NOT: *WebPart tipi* ***abstract*** *bir sınıf olup, bir Web Part bileşeni için gerekli tüm temel alt yapıyı sunmaktadır.*

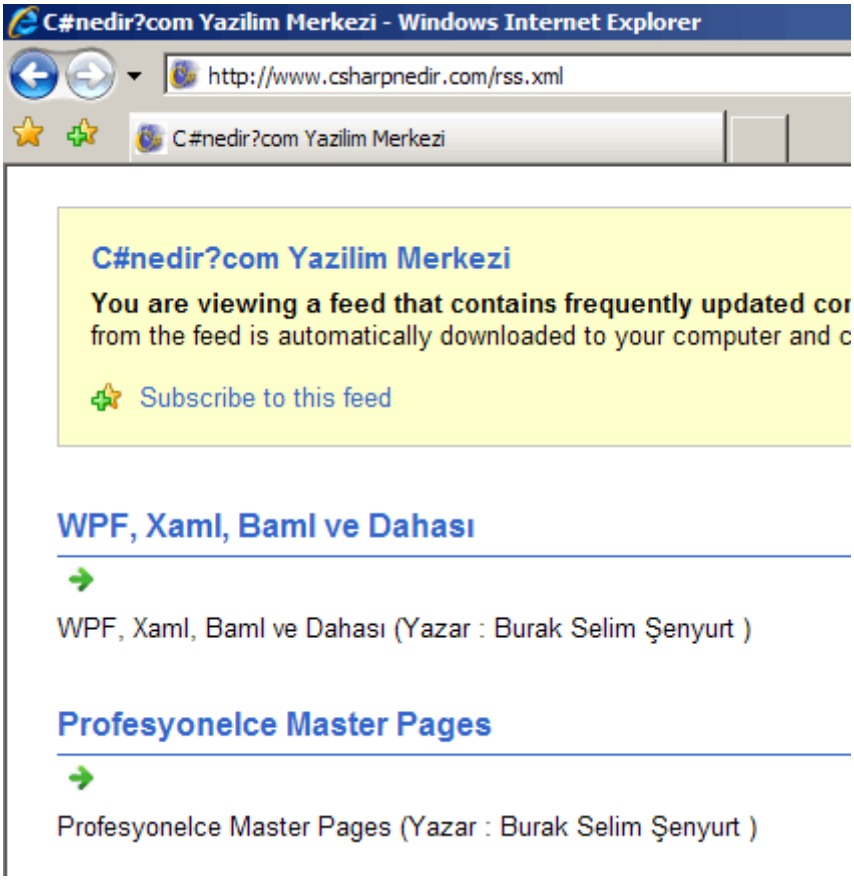
Bildiğiniz gibi, Visual Studio 2005 içerisindeki proje şablonlarından biriside, **Web Control Library** seçeneğidir. Web Control Library, standart olarak örnek bir özel kontrol (Custom Control) sınıfı içerir. Aynı zamanda Web ortamı için gerekli temel referanslarıda hazır olarak barındırır. *(Örneğin System.Web)* Biz Web Part bileşenimizi böyle bir kontrol kütüphanesi içerisine dahil edersek, herhangibir web uygulamasında kolayca kullanabiliriz. Dahası, geliştirdiğimiz Web Part bileşenlerini **Visual Studio ToolBox** içerisinde ele alabiliriz. Bu da Web Part bileşenimizi bir kontrol olarak ToolBox içerisinde tutabileceğimiz anlamına gelmektedir.

Gelelim makalemizin örnek senaryosuna. Kendi Web Part kontrollerimizi geliştirmeyi öğrenirken, standart olarak ele alınan senaryo, **RSS** bilgilerini tutan bir bileşenin yazılmasıdır. Bizde geleneği bozmayıp bu tip bir Web Part bileşenini nasıl yazabileceğimizi incelemeye çalışacağız. Ancak başlamadan önce RSS ile ilişkili olarak biraz bilgi vermekte fayda olacağı kanısındayım. Günümüzde pek çok web sitesi, güncel olarak yayınlamak istedikleri bilgilerin, başkaları tarafından kolay bir şekilde ele alınabilmesi amacıyla, Xml tabanlı içerikler sunarlar. RSS bu anlamda Xml verisini standardize etmektedir. Böylece, her RSS içeriğinin aynı şemaya sahip olması sağlanmış olur. Bizde bu yaklaşımı kullanacağız. Özellikle .Net Framework, Xml üzerinde son derece etkili yönetimli tipler **(managed types)** sunmaktadır. Bu tiplerden faydalanarak her hangibir RSS içeriğini kolay bir şekilde ayrıştırabiliriz **(parsing).** Aşağıdaki ekran görüntüsünde C#Nedir? sitesinin http://www.csharpnedir.com/Rss.xml adresinden yayınlanan RSS dökümanının bir parçasını görmektesiniz.

aspnet_Person...ASPNETDB.MDF) | Default.aspx.cs | **RssCSharpnedir.xml** | RssPart.cs | Default.aspx

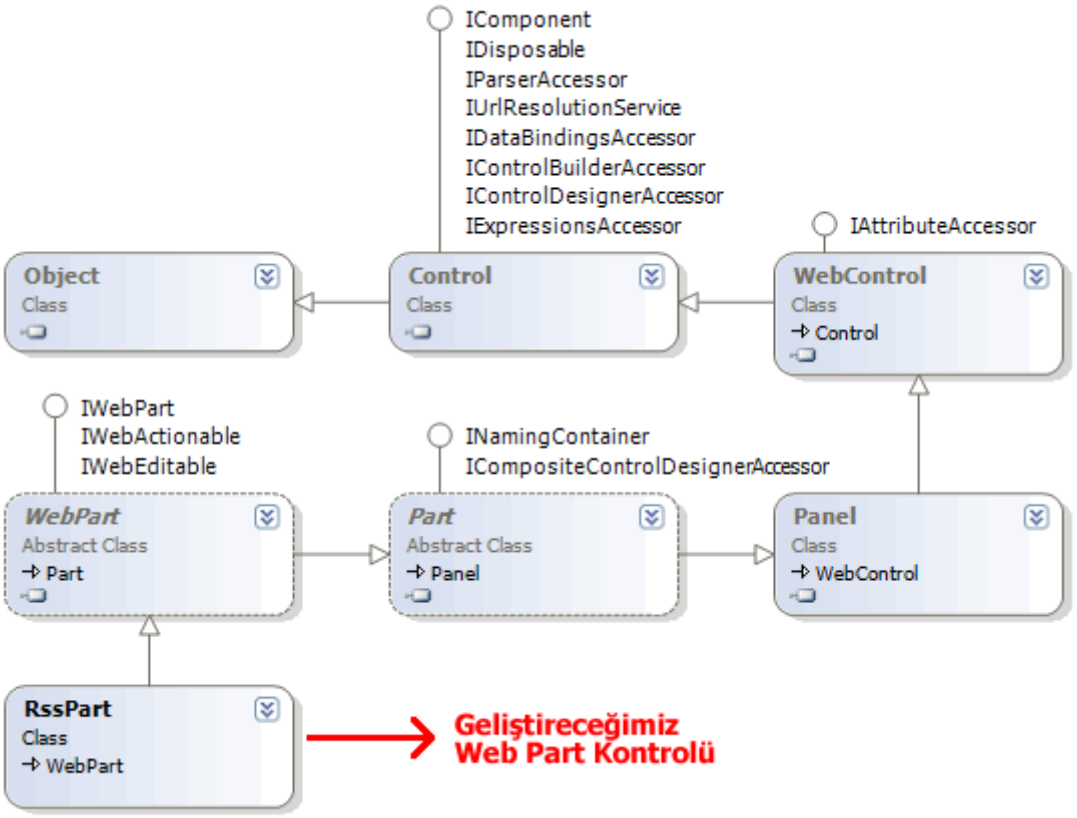
```
<?xml version="1.0" encoding="ISO-8859-9"?>
<rss version ="2.0">
  <channel>
    <title>C#nedir?com Yazilim Merkezi</title>
    <link>http://www.csharpnedir.com</link>
    <description>Türkiye'nin en iyi C# ve .NET Sitesi</description>
    <webMaster>cs@csharpnedir.com</webMaster>
    <managingEditor>algans@csharpnedir.com</managingEditor>
    <item>
      <title>WPF, Xaml, Baml ve Dahası</title>
      <description>WPF, Xaml, Baml ve Dahası (Yazar : Burak Selim Şenyurt )</description>
      <link>http://www.csharpnedir.com/makalegoster.asp?Mid=727</link>
      <pubDate>21.03.2007</pubDate>
    </item>
    <item>
      <title>Profesyonelce Master Pages</title>
      <description>Profesyonelce Master Pages (Yazar : Burak Selim Şenyurt )</description>
      <link>http://www.csharpnedir.com/makalegoster.asp?Mid=726</link>
      <pubDate>13.03.2007</pubDate>
```

Dikkat ederseniz, rss isimli root element içerisinde **channel** isimli tek bir alt boğum **(childe node)** vardır. Bu node içerisinde RSS dökümanın sahibi ile ilişkili çeşitli bilgiler yer alır. Örneğin, RSS konusunu anlatan başlık(title) ve açıklama(description) bilgisi, RSS sahibinin web adresi(link) gibi. Diğer taraftan **item** elementleri içerisindede RSS ile yayınlanmak istenen asıl içerik yer alır. Özetle bu RSS dökümanı ile çalışma zamanında C#Nedir? sitesinde yayınlanan son makalelerin listesini elde edebilir, bağlantıları kullanarak buralara geçiş yapabiliriz. Eğer bu RSS dökümanını internet üzerinden talep edersek aşağıdakine benzer bir ekran görüntüsü alırız.



İşte Web Part kontrolümüz, kişi bazında herhangibir RSS bilgisini sunacak şekilde tasarlanacaktır. Burada RSS' in Url bilgisini ve hatta RSS' in sahibi ile ilgili kısa bir bilgiyi, kişi bazında ayrı ayrı saklayabiliriz. Gelin Web Part kontrolümüzü yazarak kişiselleştirilebilir bir RSS okuyucunun nasıl yapılabileceğini görmeye çalışalım. İlk olarak **Web Control Library** projemize bir sınıf ekleyeceğiz. Sınıfımızın WebPart sınıfından türemiş olması gerekmektedir.

NOT : *Bir sınıfı WebPart abstract sınıfından türettiğimizde, söz konusu yeni tip, Web Part Framework' ünü kullanabileceğimiz üyelerede sahip olur. Bu üyeler WebPart sınfınında türediği çeşitli tipler içerisinde toplanmaktadır. Aşağıdaki şekilde, geliştirdiğimiz örnek web part bileşeninin nesne hiyerarşisini görebilirsiniz.*



İlk olarak RssPart isimli bir sınıfı WebPart tipinden türeyecek şekilde aşağıdaki gibi tanımlayalım.

```
[ToolboxData("<{0}:RssPart runat=server></{0}:RssPart>")]
public class RssPart : WebPart
{
}
```

Geliştirdiğimiz sınıf her ne kadar bir Web Part bileşeni olsada, sunucu taraflı **(server side)** bir kontroldür. Bu nedenle, aspx dosyalarının kaynak kısmında birer takı**(tag)** içerisinde ele alınacaktır. **ToolboxData** isimli niteliğin eklenmesinin sebebi de budur. Buna göre Web Part bileşenimizi herhangibir aspx sayfasına eklediğimizde aşağıdakine benzer bir element içeriği ile karşılaşırız.

```
<cc1:RssPart ID="RssPart1" runat="server"/>
```

Burada cc1 ön ekinin (prefix) nereden geldiğini merak ediyor olabilirsiniz. Aslında ToolBox' tan bir WebPart kontrolünü sayfaya sürükleyip bıraktığımızda otomatik olarak sayfanın başına **Register** direktifi eklenecektir. *(Bu ilk kontrol sürüklenip bırakıldığında otomatik olarak oluşur)* Bu direktif temel olarak, Asp.Net uygulamasında kullanılacak olan Web Part bileşenini içeren **sınıf kütüphanesinin (Contol Library**) referans bilgisinide içerecektir. Kısaca Register direktifimizinde aşağıdaki gibi olacağını söyleyebiliriz.

<%@ **Register Assembly**="MyWebParts" **Namespace**="MyWebParts" **TagPrefix**="cc1" %>

Gelelim Web Part kontrolümüz içerisindeki üyelere. Kontrolümüzün RSS belgesine ait Url bilgisini ve kısa bir başlık bilgisini tutacağını varsayabiliriz. Bu bilgiler için yapacağımız normal şartlar altında birer özellik yazmak olacaktır. Lakin bu sefer özelliklerimizin, kişi bazında özelleştirilebilmesini istiyoruz. Bu nedenle **Personalizable** niteliğini kullanmamız gerekiyor.

```
private string _Url;
private string _RssOwner;

[WebBrowsable(true)] // PropertyGridEditorPart içerisinde bu özelliğin gösterilip gösterilmeyeceğini belirtir.
[WebDescription("Verilen Url adresine göre Rss bilgisini okur")]
[Personalizable(PersonalizationScope.User)] // Özelliğin değerinin Membership tablolarında kullanıcı bazında tutulacağını belirtir.
[WebDisplayName("Rss Bilgisi Alınacak Url")] // PropertyGridEditorPart içerisinde Url özelliği için gösterilecek bilgi
public string Url
{
   get { return _Url; }
   set { _Url = value; }
}

[WebBrowsable(true)]
[WebDescription("Rss sahibine ait bilgiyi içerir")]
[Personalizable( PersonalizationScope.User)]
[WebDisplayName("Rss Yayımcısı")]
public string RssOwner
{
   set { _RssOwner = value; }
   get{return _RssOwner;}
}
```

Url ve RssOwner isimli özelliklerimiz, kişiselleştirilebilir üyelerdir. **Personalizable** niteliğine atanan **PersonalizableScop.User** değeri sayesinde Url ve RssOwner özelliklerinin işaret ettikleri değerlerin, kişi bazında Membership API' si üzerinden ilgili tablolarda tutulabileceği belirtilmektedir. **WebBrowsable** niteliği ile, özelliğin bir **PropertyGridEditorPart** bileşeni içerisinde gösterilip gösterilmeyeceğine karar verilebilir. Buna göre Url ve RssOwner isimli özelliklerimizi, sayfayı ziyaret eden kullanıcılar isterlerse **PropertyGridEditorPart** içerisinden değiştirebilirler. **WebDescription**niteliği(attribute), PropertyGridEditorPart içerisinde gösterilecek olan özelliklerin üzerlerine gelindiğinde kısa açıklama kutucukları

gösterilmesini sağlar. Son olarak **WebDisplayName** niteliği sayesinde, özelliklerin ProperyGridEditorPart içerisinde hangi adlar ile sunulacağı belirtilmektedir.

Kişi bazında tutulacak özelliklerimizide belirttikten sonra, Web Part kontrolümüzün ekrana nasıl çizileceğini ayarlayabiliriz. Bildiğiniz gibi, her sunucu web bileşeni istemci tarafında Html çıktıları haline getirilirler (Render işlemi). Burada da, belirtilen Url adresindeki RSS dökümanını ayrıştırdıktan(parsing) sonra örnek olarak linkler halinde göstermek isteyebiliriz. Bu çıktının düzenli bir formatta olmasını sağlamak içinde Html tablolarına başvurabiliriz. Bir sunucu kontrolünde, Html çıktısını oluşturmak için Render metodu göz önüne alınabilir. Diğer taratfan **CreateChildControls** metoduna başvurup daha kolay bir biçimde çıktı üretebiliriz. CreateChildControls metodunu ilerleyen makalelerimizde ele almaya çalışacağız. Şimdi dilerseniz, Web Part bileşenimiz için **Render** metodunu ezelim **(override)** ve Html çıktısını, RSS içeriğine göre hazırlayalım.

```
protected override void Render(HtmlTextWriter output)
{
   if (!String.IsNullOrEmpty(Url))
   {
      try
      {
         XmlReader reader = XmlReader.Create(Url);

         DataSet ds = new DataSet();
         ds.ReadXml(reader);

         DataTable items = ds.Tables["item"];

         #region Render Table

         // Table elementi render edilmeden önce gerekli style attribute' ları ekleniyor
         output.AddStyleAttribute(HtmlTextWriterStyle.BackgroundColor,
"WhiteSmoke"); // Arka plan rengi
         output.AddStyleAttribute(HtmlTextWriterStyle.Width, "100%"); // genişlik
belirleniyor
         output.RenderBeginTag(HtmlTextWriterTag.Table); // Table için açılış takısı

            output.AddStyleAttribute(HtmlTextWriterStyle.BackgroundColor, "Gold");
            output.RenderBeginTag(HtmlTextWriterTag.Tr); // Tr açılış takısı (satır)
               output.RenderBeginTag(HtmlTextWriterTag.Td); // Td açılış takısı (hücre)
                  output.Write(ds.Tables["channel"].Rows[0]["title"].ToString()); // Td
içerisine Rss dökümanından title bilgisi alınıyor
               output.RenderEndTag(); // Td için kapanış takısı
            output.RenderEndTag(); // Tr için kapanış takısı

            // Rss dökümanındaki her bir item için bir Tr (Table Row) ve içerisinde bir Td
```

```
(hücre) oluşturuluyor
            for (int i = 0; i < items.Rows.Count; i++)
            {
               output.RenderBeginTag(HtmlTextWriterTag.Tr); // Tr açılış takısı
                  output.RenderBeginTag(HtmlTextWriterTag.Td); // Td açlış takısı
                     output.AddAttribute(HtmlTextWriterAttribute.Href,
items.Rows[i]["link"].ToString()); // href isimli attribute sonraki satırda açılacak olan A
takısına ilave edilecek. Değeri ise link alanının içeriği olacak
                     output.RenderBeginTag(HtmlTextWriterTag.A);// A takısı açılıyor
                        output.Write(items.Rows[i]["title"].ToString()); // A takısı içine title
alanının değeri yazılıyor
                     output.RenderEndTag(); // A takısı kapatılıyor
                  output.RenderEndTag(); // Td takısı kapatılıyor
               output.RenderEndTag(); // Tr takısı kapatılıyor
            }

         output.RenderEndTag();// Table' ın bitiş takısı </table>
      }
      catch
      {
         output.Write("Adres çözümlenemedi");
      }

      #endregion
   }
}
```

Render metodunda, daha önceden Web sunucu kontrolü yazmak ile ilişkili makalelerimizde kullandığımızdan biraz daha farklı bir yol tercih ettik. Html içeriğini oluştururken, .Net içerisinde yer alan kuvvetli tiplerden faydalanırsak hatalı Html takısı yazma olasılığımız daha da azalacak ve özellikle tarayıcı farklılıklarını bertaraf edeceğizdir. Bu nedenle Render metodu içerisinde, **HtmlTextWriter** sınıfına ait **RenderBeginTag**, **RenderEndTag**, **AddAttribute**, **AddStyleAttribute** gibi metodlardan yararlanılmıştır. Örneğimize göre içeriğimiz bir Table elementi içerisinde tek sütundan oluşan bir yapıda olacaktır. Bu amaçla bir table takısı açmak için

```
output.RenderBeginTag(HtmlTextWriterTag.Table);
```

kod satırından faydalanılmıştır. Burada RenderBeginTag, bir takının oluşturulmasını sağlarken ne çeşit bir element olacağını parametre olarak gönderdiğimiz**HtmlTextWriterTag** sabiti belirtmektedir. Html içerisinde açılan her takının kapatılması gerektiğini biliyoruz. Bunu kod tarafında ifade ederkende yine HtmlTextWriter sınıfının**RenderEndTag** metodundan faydalanmaktayız. Dikkat edilmesi gereken noktalardan biriside, Html elementlerine niteliklerin nasıl eklendiğidir. Dikkat

ederseniz, bu amaçla **AddStyleAttribute** ve **AddAttribute**metodlarından yararlanılmıştır. Örneğin tablomuza arka plan rengi ve genişlik vermek için

output.**AddStyleAttribute**(**HtmlTextWriterStyle**.**BackgroundColor**, "WhiteSmoke");
output.AddStyleAttribute(**HtmlTextWriterStyle**.**Width**, "100%");

kod satırlarından faydalanılmaktadır. Hangi niteliğin ekleneceğini belirlemek için ise, **HtmlTextWriterStyle** sabitinden yararlanılır. AddAttribute metoduda benzer işlevselliğe sahip olmakla birlikte desteklediği nitelik tipleri daha farklıdır.

Gelelim metodun temel olarak yaptığı işe. İlk olarak kişiselleştirilebilen Url özelliğinden değer alınmakta ve **XmlTextReader** nesnesi elde edilmektedir. Hataların önüne geçmek amacıyla ilk olarak Url özelliğinin değerinin boş olup olmadığına bakılır. Bunun haricinde internet bağlantısının kesik olması gibi hallerde, istenen Url' den RSS bilgileri çekilemeyeceği için bir çalışma zamanı istisnası(runtime exception) alınacaktır. Bununda önüne geçmek için genel bir try-catch bloğu kullanılmıştır. Bu nedenle yükleme ve Render işlemlerini try bloğu içerisinde gerçekleştirmekteyiz. Burada Url adresinden elde edilen Xml içeriğini okumak için farklı yollarda tercih edilebilir. Örneğin doğrudan XmlReader üzerinden hareket edilebilir veya**XPathNavigotor** tipinden faydalanılabilir ki bunlar performans açısından daha etkili yollardır. Biz kodu çok fazla karmaşıklaştırmamak adına doğrudan **DataSet** kontrolüne alıyor ve pahalı bir maliyetinde altına giriyoruz :) DataSet içerisinde yer alan channel ve item tablolarından faydalanarak, RSS içeriğindeki bilgilere erişebiliriz. Örneğin RSS sahibinin belirttiği başlığı (title), Html tablomuzun ilk satırındaki ilk hücreye alıyoruz. Her bir item elementinin belirttiği, başlık ve adres bilgilerini ise sırasıyla title ve link elementlerinden alıp bir **a href** elementi içerisinde gösteriyoruz. Bildiğiniz gibi a href elementi bir link oluşturulmasını sağlamaktadır.

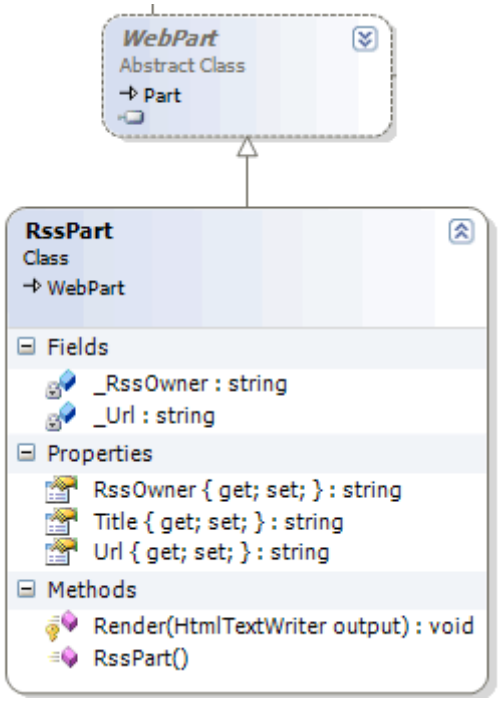
Web Part kontrolümüzün, **WebPart** sınıfından gelen pek çok özelliğinide istersek ezebiliriz (override). Örneğin, Web Part bileşenimizin başlık bilgisi(title), imge bilgisi (title icon image) gibi değerlerini değiştirmek isteyebiliriz. Lakin bu tip üyelerin değerlerinin kontrolün Html çıktısının üretilmesinden önce atanması gerekir. Bu amaçla yapıcı metoddan, özelliğin set bloğundan yada kontrole ait PreInit olay metodundan faydalanabiliriz. Geliştirdiğimiz örnekte biz yapıcı metod ve özellik(property) kullanmayı tercih edeceğiz. Bu amaçla sınıfımıza aşağıdaki üyeleri eklememiz yeterli olacaktır.

```
public override string Title
{
   get
   {
      return _RssOwner;
   }
   set
   {
      _RssOwner= value;
   }
```

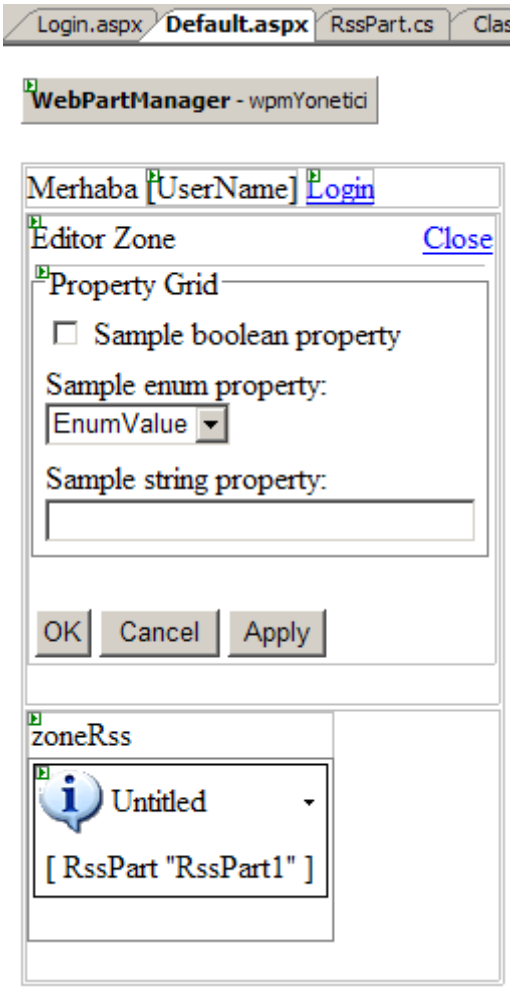
```
}

public RssPart()
{
   TitleIconImageUrl = "Bilgi.gif";
}
```

Yukarıdaki kod parçasında dikkat ederseniz, WebPart sınıfında yer alan Title özelliği ezilmektedir ve kullanıcı tarafından kişiselleştirilebilen RssOwner özelliğinin kullandığı _RssOwner alanının değerini işaret etmektedir. Diğer taraftan, Web Part bileşenimizde hemen başlık kısmının yanında sembolik bir imge göstermek amacıyla yapıcı metod(constructor) içerisinde **TitleIconImageUrl** özelliğine bir değer ataması yapılmıştır. Örnek bir Asp.Net sayfasında Web Part bileşenimizi kullandığımızda tasarım zamanında iken, aşağıdakine benzer bir görüntü ile karşılaşırız.

Burada yer alan Bilgi.gif isimli resim, Web Control Library içerisinde yer almaktadır ve herhangibir Asp.Net projesinde ilgili Web Part bileşeni kullandığında otomatik olarak root klasör içerisine taşınacaktır. Bir başka deyişle eklenen ilk kontrol ile birlikte gelen web kontrol kütüphanesi referansı, beraberinde kaynak olarak bu resim dosyasınıda hedef web uygulaması içerisine taşıyacaktır. Yapılan bu son ekelemeler ile birlikte Web Part bileşenimizin yeni hali aşağıdaki sınıf diagramında gösterildiği gibi olacaktır.



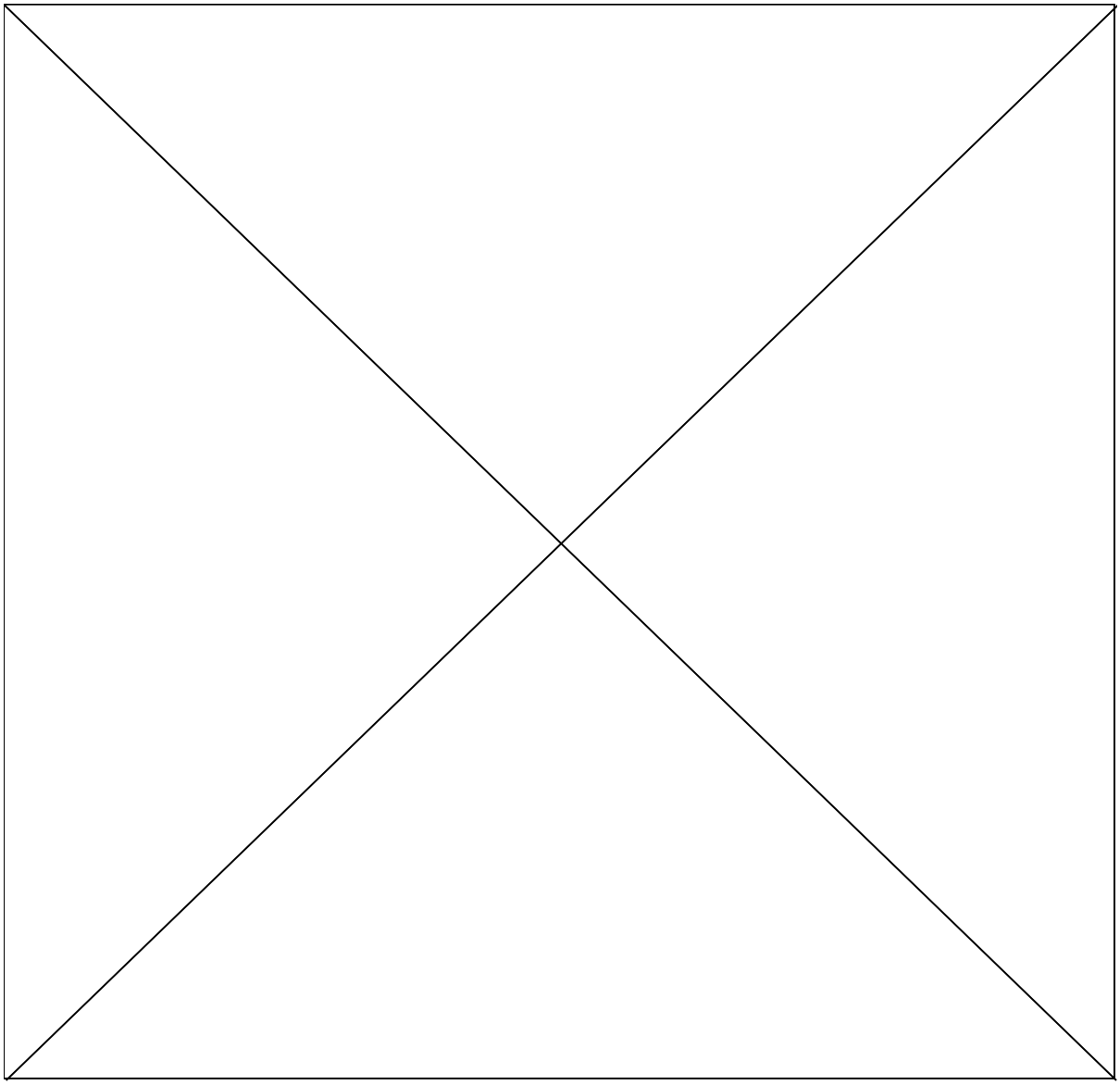
Artık Web Part bileşenimizi bu haliyle test edebiliriz. Kişiselleştirmenin tam olarak etkilerini görebilmek için, bileşenimizi **Membership** ayarları yapılmış örnek bir Asp.Net Web uygulaması üzerinden test etmekte fayda olacaktır. Testleri gerçekleştirmek için ekran görüntüsü aşağıdaki gibi olan bir web sayfasından yararlanabiliriz.



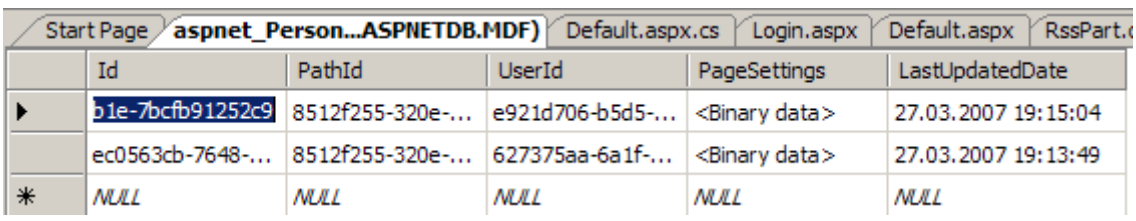
Unutulmamalıdır ki, geliştirdiğimiz Web Part bileşeninin, Web Part Framework özelliklerini etkin bir şekilde kullanabilmesi için bir **Web Part Zone** içerisinde yer alması gerekir. Bu nedenle, RssPart isimli Web Part bileşenimizi zoneRss isimli bir WebPartZone kontrolü içerisinde ele alıyoruz. Diğer taraftan, Web Part kontrolümüz içerisinde yer alan Url ve RssOwner gibi çalışma zamanında kişiselleştirilebilir özelliklerini ele alabilmek için bir **Editor Zone** kontrolümüzde yer almaktadır. Örneğimizde sadece editör kısmını göz önüne alacağımızdan, sayfanın Page_Load olay metodu içerisinde **WebPartManager** bileşenimizin **DisplayMode** özelliği aşağıdaki gibi **EditDisplayMode** olarak ayarlanmıştır. Gerçek bir projede elbetteki kullanıcının diğer modlarıda seçebileceği şekilde kod yazmak gerekir.

wpmYonetici.DisplayMode = WebPartManager.**EditDisplayMode**;

Sonuç olarak uygulamamızı çalıştırdığımızda, aşağıdaki Flash animasyonunda yer alana benzer bir sonuç ile karşılaşırız.*(Flash dosyasının boyutu 380 Kb tır. Bu nedenle yüklenmesi zaman alabilir.)*



Dikkat ederseniz, iki ayrı kullanıcı için iki ayrı RSS bilgisi ele alınabilmektedir. Burak isimli kullanıcı kendisi için Msdn' e ait RSS içeriğini tutarken, Melike isimli kullanıcıda C#Nedir? sitesine ait RSS bilgisini ele alabilmektedir. Bu bilgilerin arka tarafta nereye yazıldığını kontrol etmek istersek, **Membership** bilgilerinin tutulduğu veritabanında *(ki örneğimizde local AspNetDb.mdf dosyası kullanılmaktadır. Bir başka deyişle üyelik bilgileri web uygulamasın ait App_Data klasöründe yer alan AspNetDb.mdf veritabanı içerisinde saklanmaktadır.)* yer alan**AspNet_PersonalizationPerUser** tablosuna bakmamız yeterli olacaktır.



Kendi Web Part bileşenlerimizi geliştirirken başka üst sınıf özelliklerini ezebilir ve istediğimiz şekilde çalışmalarını sağlayabiliriz. Aynı zamanda, bir Web Part bileşenine çalışma zamanında ele alacağı yeni fiili aksiyonlar **(verb)** ekleyebiliriz. Bu ve benzeri diğer konuları ilerleyen makalelerimizde ele almaya çalışacacağız. Böylece geldik bir makalemizin daha sonuna. Bu makalemizde basit olarak kendi Web Part bileşenlerimizi nasıl yazabileceğimizi incelemeye çalıştık. Bunun için,

- kontrolümüzü tanımlayacak olan sınıfı WebPart isimli abstract sınıftan türetmemiz gerektiğini,
- içeride kullanıcı bazında kişiselleştirme yapacağımız özellikler için Personalizable niteliğini (attribute) kullanmamız gerektiğini,
- kontrolün kişiselleştirilebilir özelliklerinin çalışma zamanında ele alınması için PropertyGridEditorPart kullanmamız gerektiğini,
- özelliklerin PropertyGridEditorPart içerisinde gözükmesi için WebBrowsable niteliğinin kullanılması gerektiğini,
- istersek var olan Web Part özelliklerini ezerek değiştirebileceğimizi,
- her zaman olduğu gibi kontrolün Html çıktısını manuel olarak düşünememiz ve yazmamız gerektiğini,

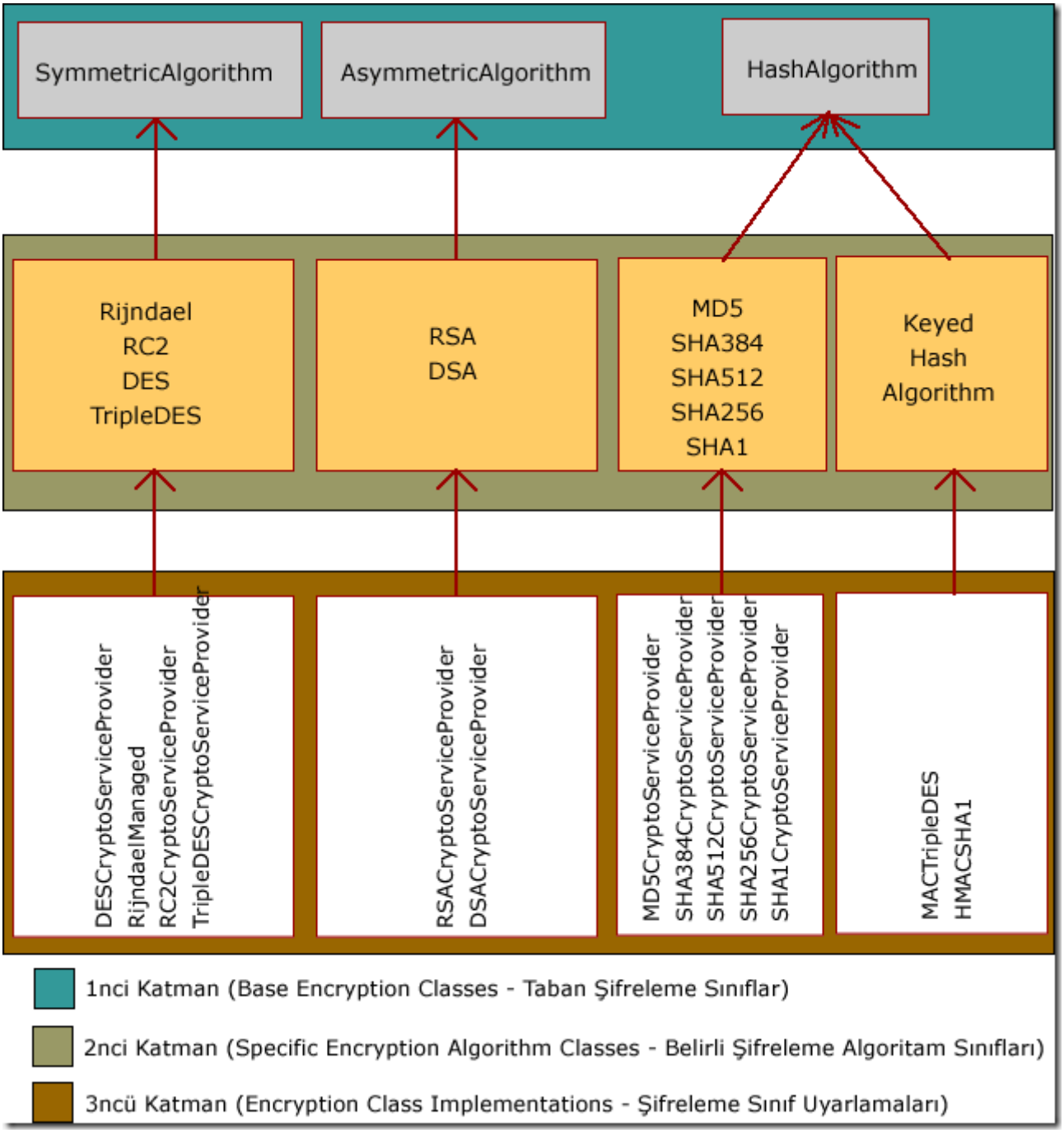
öğrendik. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

Örnek Uygulama İçin Tıklayınız. (Dosya boyutunun küçülmesi için, App_Data klasörüne atılan Aspnetdb.mdf dosyası silinmiştir. Test ederken burada bir AspNetdb.mdf dosyası oluşturmanı gerekecektir. Bunun için Asp.Net Web Site Administration Tool' dan faydalanabilirsiniz.)

RijndaelManaged Vasıtasıyla Encryption(Şifreleme) ve Decryption(Deşifre) (2005-02-23T19:38:00)

c#,.net framework,encryption,decryption,rijndael,

Bu makalemizde, Rijndael Algoritmasını kullanan Managed tiplerden **RijndaelManaged** sınıfı ile şifreleme (encryption) ve deşifre etme (decryption) işlemelerinin nasıl gerçekleştirilebileceğini incelemeye çalışacağız. Konu ile ilgili örneklerimize geçmeden önce .Net Framework içerisinde yer alan Cryptography mimarisinde kısaca bahsetmekte yarar olduğunu düşünüyorum. Aşağıdaki şekil, .Net Framework' te **System.Securtiy.Cryptograhpy** isim alanında yer alan şifreleme hiyerarşisini göstermektedir. Framework mimarisinde şifreleme sistemi ilk olarak üç ana katmandan oluşur. İlk katmanda taban sınıflar (base classes) yer alır. Bunlar SymmetricAlgorithm, AsymmetricAlgorithm ve HashAlgorithm sınıflarıdır. Bu sınıflar kendisinden türeyen ikinci katman sınıfları için temel ve ortak şifreleme özelliklerini içerirler.



SymmetricAlgorithm sınıfını kullanan şifreleme mekanizmalarında herhangi bir anahtar ile şifrelenen veriler, deşifre edilmek istendiklerinde yine aynı anahtarı kullanırlar. Bu özellikle internet gibi herkesin kullanımına açık olan ortamlarda güvenlik açısından tehlike yaratabilir. Nitekim anahtarın herhangi bir şekilde ele geçirilmesi, şifrelenen verinin çözülmesi için yeterli olacaktır. Diğer yandan bu tekniğe göre geliştirilen algoritmalar hızlı ve performanslı çalışırlar.

SymmetricAlgorithm katmanından türeyen Encryption sınıfları ile uygulanan şifreleme algoritmalarında veriyi şifrelerken kullandığımız key(anahtar) ve IV(vektör) değerleri, aynı veriyi deşifre ederken de gereklidir.

SymmetricAlgorithm yapısını kullanan şifreleme mimarilerinin neden olduğu güvenlik sorununun çözümü için AsymmetrciAlgorithm taban sınıfı (base class) geliştirilmiştir. Bu mekanizmada veri şifreleneceği zaman public bir anahtar kullanılır. Bu anahtarın herhangi bir şekilde ele geçirilmesi, verinin deşifre edilebilmesi için yeterli değildir. Nitekim verinin deşifre (decryption) edilebilmesi için karşı tarafın private bir anahtara gereksinimi vardır. Bu avantajının yanında AsymmetricAlgroithm mekanizması SymmetricAlgorithm mekanizmasına göre daha yavaş çalışmaktadır.

AsymmetricAlgorithm katmanından türeyen Encryption sınıfları ile uygulanan şifreleme (encryption) algoritmalarında veriyi şifrelerken public bir key kullanırken, deşifre (decryption) işlemi sırasında farklı olan private key kullanılır.

Birinci katmanda yer alan taban sınıflar, abstract niteliktedir. Dolayısıyla kendisinden türeyen şifreleme sınıflarının içermesi ve uygulaması zorunlu olan üyeler içerirler. Bildiğiniz gibi abstract sınıflardan nesne örnekleri üretilemez. Ancak taban sınıfların static Create metotları yardımıyla bu sınıfları da şifreleme mekanizmalarında kullanabiliriz. İkinci katmanda yer alan sınıflar ise, özellikle belirli şifrelme algoritmalarını işaret ederler. örneğin bu gün işleyeceğimiz Rijndael algoritması 256 bitlik bir anahtar (key) ile şifreleme (deşifre etme) sağlar. Buradaki sınıflar, taban sınıflardan (base classes) türemiştir ve abstract sınıflardır.

Asıl şifreleme metotlarını ve üyelerini bizim için kullanışlı hale getiren sınıflar üçüncü katmanda yer alırlar. Burada dikkat edecek olursanız bazı sınıflar ServiceProvider kelimesi ile biterler. Bu sınıflar Windows' un CryptoApi kütüphanesini kullanan sınıflardır. Diğer yandan Managed kelimesi içeren sınıflar (örneğin RijndaelManaged) özellikle .net için geliştirilmiş yönetimsel uyarlamalardır (Managed Implementations). Buradaki sınıflar sealed olarak tanımlanmıştır. Yani kendilerinden türetme yapılamaz. Buna rağmen eğer istersek ikinci veya birinci katman sınıflarını kullanarak kendi şifreleme algoritma sınıflarımızı veya uyarlamalarımızı geliştirebiliriz. Peki bir veri kümesini şifrelemek için yukarıdaki katmanları ve içeriklerini nasıl kullanabiliriz. Bu makalemizde biz örnek olarak Rijndael algoritmasını kullanan iki örnek geliştireceğiz. .Net içerisinde verileri şifrelemek için izleyeceğimiz yolda anahtar nokta CryptoStream sınıfıdır. Bu sınıfa ait nesne örnekleri yardımıyla verileri stream bazlı olarak, istenen algoritmaya göre şifreleyebilir ya da deşifre edebiliriz. CryptoStream sınıfından bir nesne örneğini aşağıdaki yapıcı metot (constructor) yardımıyla oluşturabiliriz.

public
CryptoStream(Stream <I>stream</I>**,** ICryptoTransform <I>transform</I>**,** CryptoStream Mode <I>mode</I>**);**

Bu metodun ilk parametresine dikkat edecek olursanız bir Stream nesnesidir. CryptoStream sınıfı şifrelenecek veya deşifre edilecek verilerin işlenmesi sırasında Stream nesnelerini kullanılır. Dolayısıyla bellek üzerinde tutulan verileri, fiziki dosyalarda tutulan verileri veya network üzerinden akan verileri ilgili stream nesneleri yardımıyla (MemoryStream, FileStream, NetworkStream vb...) CrpytoStream sınıfına ait bir nesne örneğine

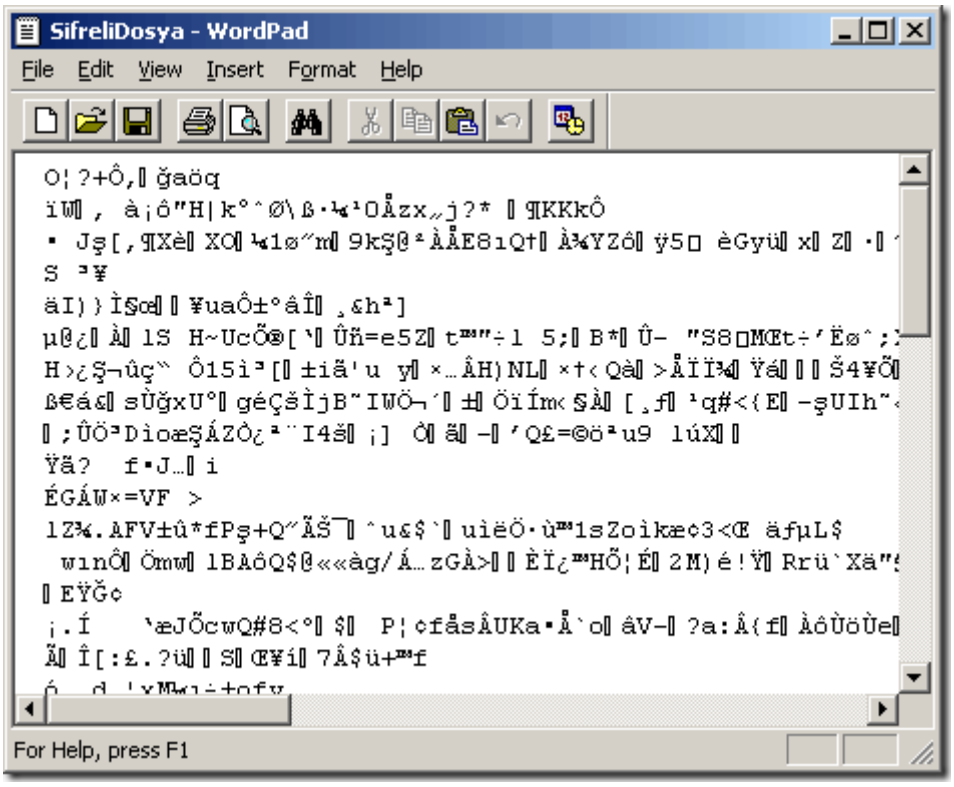
aktarabiliriz. İkinci parametre ise Stream üzerindeki verinin hangi algoritma ile şifreleneceğini (deşifre edileceğini) belirlemek üzere kullanılır. Bu parametre ICryptoTransform ara yüzü tipinden bir nesne örneğidir. üçüncü parametre ise Stream' e yazma veya stream' den okuma yapılacağını belirtir. Stream üzerindeki veriyi şifreleyeceğimiz zaman üçüncü parametre Write değerini alırken, deşifre işlemlerinde Read değerini alır. Karışık gibi görünmesine rağmen örneklerimizden de göreceğiniz gibi, şifreleme(deşifre etme) işlemleri sanıldığı kadar zor değildir. Dilerseniz vakit kaybetmeden örneklerimize geçelim. İlk örneğimizde metin tabanlı bir dosya içeriğini FileStream nesnesinden faydalanarak okuyor ve CyrptoStream sınıfına ait nesne örneği yardımıyla şifreleyerek bir dosyaya yazıyoruz. Daha sona ise şifrelenen bu dosyayı tekrardan çözüyoruz.

```
using System;
using System.IO;
using System.Security.Cryptography;
namespace CryptoStreamS1
{
   class Class1
   {
      [STAThread]
      static void Main(string[] args)
      {
         #region Dosya Şifrelemesi (Rijndael algoritması ile)
         // İlk olarak şifrelemek istediğimiz dosya için bir stream oluşturuyoruz.
         FileStream fs=new
FileStream(@"SifreliDosya.txt",FileMode.OpenOrCreate,FileAccess.Write);
         /* Kullanacağımız şifreleme algoritmasını uygulatabileceğimiz managed Rijndael
sınıfına ait nesne örneğimizi tanımlıyoruz. Şifreleme algoritması olarak Rijndael tekniğini
kullanıyoruz. */
         RijndaelManaged rm=new RijndaelManaged();
         rm.GenerateKey(); // Aalgoritma için gerekli Key üretiliyor.
         /* Şimdi algoritma için gerekli key ve vektör değerlerini üretiyoruz. Burada
kullanılan şifreleme algoritması simetrik yapıda olduğundan şifrelenen verinin açılabilmesi
için (decrypting) aynı key ve vektör değerine ihtiyacımız var. Bu nedenle bunları bir byte
dizisinde tutuyoruz.*/
         // Elde ettiğimiz key değerini bir byte dizisine aktarıyoruz.
         byte[] k=new byte[rm.Key.Length];
         for(int i=0;i<rm.Key.Length;i++)
         {
            Console.Write(rm.Key[i]);
            k[i]=rm.Key[i];
         }
         Console.WriteLine();
         rm.GenerateIV(); // Algoritma için gerekli IV vektör değeri üretiliyor.
         byte[] v=new byte[rm.IV.Length];
```

```
        // Elde ettiğimiz Vektör değerini bir byte dizisine aktarıyoruz.
        for(int i=0;i<rm.IV.Length;i++)
        {
           Console.Write(rm.IV[i]);
           v[i]=rm.IV[i];
        }
        Console.WriteLine();
        /* Belirlediğimiz şifreleme algoritmasını kullanarak, stream üzerinde şifrelemeyi
yapacak CryptoStream nesnemizi oluşturuyoruz. Şifrelemeyi oluşturmak için
RijndaelManaged sınıfından örneklendirdiğimiz nesnemizin CreateEncyrptor metodunu
kullanıyoruz. Oluşturulan şifreli dökümanı ilgili stream' e yazmak
istediğimizdenCryptoStreamMode olarak Write değerini seçiyoruz.*/
        CryptoStream cs=new
CryptoStream(fs,rm.CreateEncryptor(),CryptoStreamMode.Write);
        /*Şimdi şifrelenecek olan byte dizisini almak üzere dosyamız için bir akım
oluşturuyoruz. Nitekim CryptoStream' in aşağıda kullanılan aşırı yüklenmiş versiyonu ilk
parametre olarak şifrelenecek veri yapısını bir byte dizisi halinde alıyor.*/
        FileStream fs2=new FileStream(@"Dosya.txt",FileMode.Open);
        // dosyanın içeriğini byte dizisine aktarıyoruz.
        byte[] veriler=new byte[fs2.Length];
        fs2.Read(veriler,0,(int)fs2.Length);
        // CryptoStream sınıfının write metodu ile dosya.txt' yi okuduğumuz byte dizisinin
içeriğini fs2 ile belirttiğimiz stream' e yazıyoruz.
        cs.Write(veriler,0,veriler.Length);
        fs2.Close();
        cs.Close();
        #endregion
        #region Şifrelenmiş dosyanın örnek olarak ilk satırının decrypt edilerek okunması.
        /* Bu kez işlemleri tersten yapıyoruz. İlk olarak şifrelenmiş ve decrypt edilmek
istenen stream nesnesinin oluşturuyoruz. Ardından bu stream'  deki veriye Rijndael
alogirtmasını uygulayarak Decrypting yapıyoruz. */
        FileStream fsSifreliDosya=new
FileStream(@"SifreliDosya.txt",FileMode.Open,FileAccess.Read);
        RijndaelManaged rm2=new RijndaelManaged();
        //simetrik algoritma kullandığımız için decrypting içinde aynı key ve vektör
değerlerini kullanmamız gerekiyor.
        rm2.Key=k;
        rm2.IV=v;
        CryptoStream cs2=new
CryptoStream(fsSifreliDosya,rm2.CreateDecryptor(),CryptoStreamMode.Read);
        StreamReader sr=new StreamReader(cs2);
        string satir=sr.ReadLine();
        Console.WriteLine(satir);
        #endregion
    }
```

```
    }
}
```

Uygulamayı çalıştırdığımızda orijinal içerikli dosyanın aşağıdaki gibi şifrelendiğini görürüz.



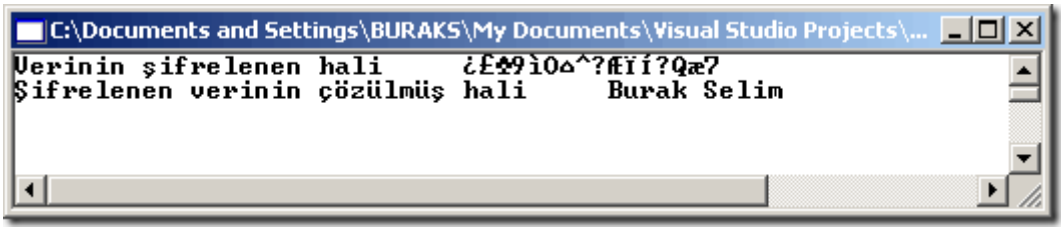
İkinci örneğimizde ise, MemoryStream nesnesinden yararlanacağız. Bu kez, bir veri tablosundan çektiğimiz belli bir alanı bellek üzerinden şifreliyor ve daha sonra şifrelenen verinin orijinal içeriğini elde edecek şekilde deşifre işlemini uyguluyoruz.

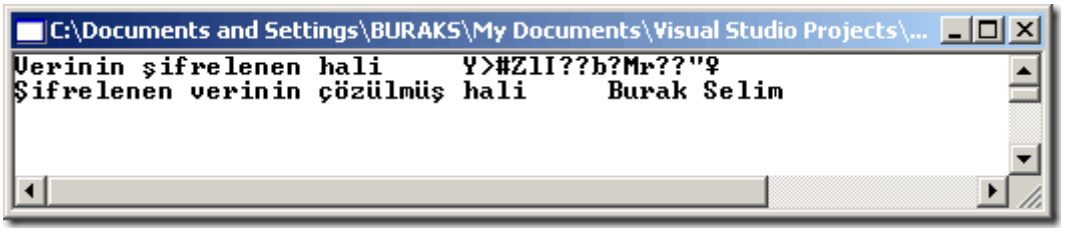
```
using System;
using System.IO;
using System.Data;
using System.Data.SqlClient;
using System.Security.Cryptography;
namespace CryptoStreamS2
{
    class Kriptografi2
    {
        [STAThread]
        static void Main(string[] args)
        {
            #region verinin şifrelenmesi
            /* öncelikle şifrelemek istediğimiz veriyi elde ediyoruz. örnek olarak SQL
Sunucusundaki Ogrenciler tablosundan belirli bir alanı aldık. */
```

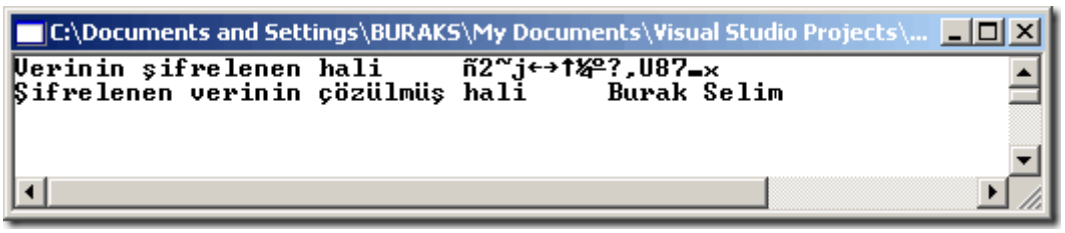
```
        SqlConnection con=new SqlConnection("data
source=BURKI;database=Work;integrated security=SSPI");
        SqlCommand cmd=new SqlCommand("SELECT AD FROM Ogrenciler WHERE
OGRENCINO=1",con);
        con.Open();
        string sifrelenecekVeri=cmd.ExecuteScalar().ToString();
        con.Close();
        /* Şifrelenecek verinin herşeyden önce bir byte dizisi olarak ele alınması
gerekiyor.*/
        byte[] sv=new byte[sifrelenecekVeri.Length];
        for(int i=0;i<sv.Length;i++)
        {
           sv[i]=(byte)sifrelenecekVeri[i];
        }
        /* Şifrelenecek veriyi belleğe yazacağız. Bu nedenle MemoryStream sınıfı tipinden
bir nesne örneği oluşturduk*/
        MemoryStream ms=new MemoryStream();
        /* Şifreleme algoritması olarak Rijnadel tekniğini sağlayan Managed nesne
örneğimizi oluşturuyoruz.*/
        System.Security.Cryptography.RijndaelManaged rm=new RijndaelManaged();
        /* Şifreleme için gerekli anahtar ve vektör değerlerini elde ediyoruz.*/
        rm.GenerateKey();
        rm.GenerateIV();
        /* RijndaelManaged nesnesi tarafından üretilen anahtar ve vektör değerlerini byte
dizilerine alıyoruz. Nitekim karşı tarafın şifrelenen veriyi çözebilmesi için bu anahtar ve
vektör değerlerinin aynılarına ihtiyaçları olacaktır.*/
        byte[] anahtar=rm.Key;
        byte[] vektor=rm.IV;
        /* Veriyi belirttiğimiz algoritmaya göre şifreleyerek parametre olarak verilen
stream' e ki burada MemoryStream' e yazmak için CryptoStream sınıfımızdan nesne
örneğimizi oluşturuyoruz.*/
        CryptoStream cs=new
CryptoStream(ms,rm.CreateEncryptor(anahtar,vektor),CryptoStreamMode.Write);
        /* Veriyi şifreleyerek belleğe yazıyoruz. Başından sonuna kadar.*/
        cs.Write(sv,0,sv.Length);
        cs.FlushFinalBlock();
        Console.Write("Verinin şifrelenen hali ");
        byte[] icerik=ms.ToArray(); /* Belleğe yazdığımız şifrelenmiş veriyi bir byte
dizisine alarak okuyor ve ekrana yazdırıyoruz.*/
        for(int i=0;i<icerik.Length;i++)
        {
           Console.Write((char)icerik[i]);
        }
        Console.WriteLine();
        #endregion
```

```
        #region şifrelenen verinin çözümlenmesi
        /* Bellekte tutulan icerik değerini yani şifrelenmiş olan veriyi parametre alan stream nesnemizi oluşuturuyoruz.*/
        MemoryStream msCoz=new MemoryStream(icerik);
        RijndaelManaged rmCoz=new RijndaelManaged(); // Rijndael algoritmasını kullanarak şifrelenen veriyi çözecek olan provider nesnemizi tanımlıyoruz.*/
        /* SymmetricAlgorithm söz konusu olduğundan RijndaelManaged sınıfına ait nesne örneğinin decryption işlemi için encrypt' te kullanılan key ve IV değerlerine ihtiyacımı var.*/
        rmCoz.Key=anahtar;
        rmCoz.IV=vektor;
        /* Bu kez CryptoStream nesnemiz stream' den okuduğu veri üzerinde Decypting işlemini gerçekleştirecek. Bu nedenle Rijndael nesne örneğimizin CreateDecryptor metodunu çağırıyoruz.*/
        CryptoStream csCoz=new CryptoStream(msCoz,rmCoz.CreateDecryptor(anahtar,vektor),CryptoStreamMode.Read);
        byte[] cozulen=new byte[ms.Length]; // çözülen veriyi tutacak bir byte dizisi oluşturuyoruz.
        csCoz.Read(cozulen,0,icerik.Length); // Şifrelenen veriyi çözümleyerek okuyoruz.
        Console.Write("Şifrelenen verinin çözülmüş hali ");
        /* çözümlenmiş veriyi son olarak ekrana yazdırıyoruz.*/
        for(int i=0;i<cozulen.Length;i++)
        {
           Console.Write((char)cozulen[i]);
        }
        Console.ReadLine();
        #endregion
      }
   }
}
```

Uygulamamızı arka arkaya çalıştırdığımızda aşağıdakine benzer sonuçlar alırız. Dikkat ederseniz deşifre edilen veri her seferinde aynı olmasına rağmen, şifrelenen veri içeriği bir birlerinden farklıdır.







Bu makalemizde kısaca Rijndael algoritmasını kullanan RijndaelManaged sınıfı ile şifreleme ve deşifre işlemlerini incelemeye çalıştık. İlerleyen makalelerimizde, AsymmetricAlgorithm tekniğininin nasıl uygulanabileceğini incelemeye çalışacağız. Bir sonraki makalemizde görüşünceye dek hepinize mutlu günler dilerim.

Cryptography.rar (33,72 kb)

Oyun Programlamaya Giriş (Matrisler Yardımıyla Çarpışma Kontrolü) (2004-12-04T20:09:00)

game programming,c#,

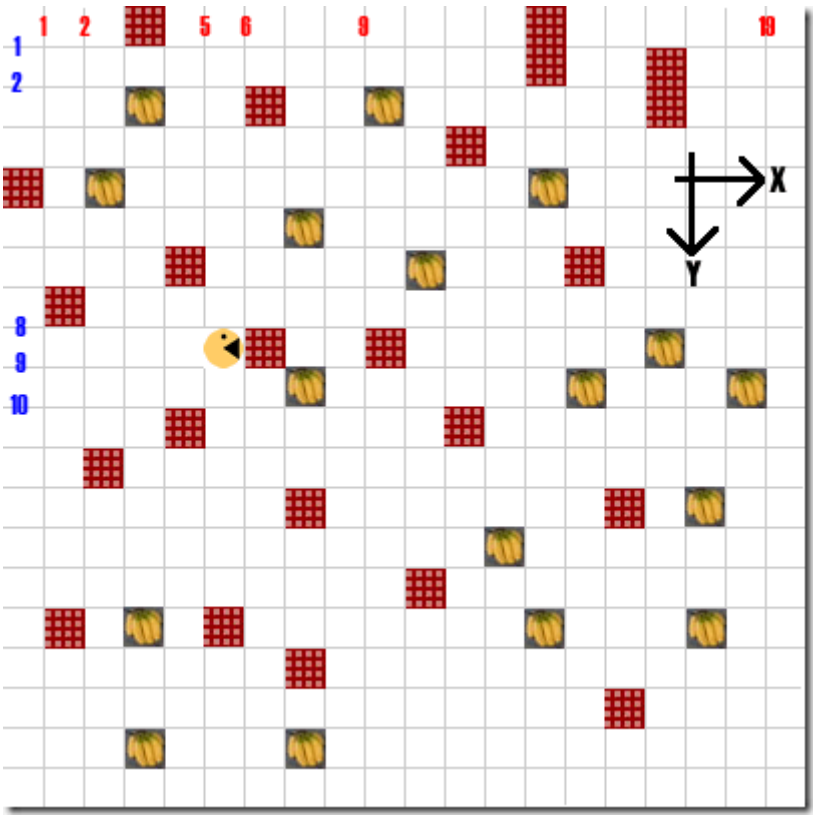
Hafta sonu evde bilgisayarım başında internette gezinirken, tarihi oyunların anlatıldığı bir site ile karşılaştım. Aslında zaten eski oyunları araştırıyordum. Amacım bu oyunlara, oyun oynamak isteyen bir çocuk gözü ile değil, onların yapılarını ve çekirdeklerini anlamaya çalışacak bir yazılımcı gözüyle bakabilmekti. Sonuçta, içimdeki çocuk ağır basıp bir kaç tanesini saatlerce oynadım. Aralarında en çok hoşuma gidenlerden birisi PackManâ€™ di. Packman, doğrusal düzlemde 4 yöne hareket edebilen bir kahramandı. Yolda kendisini rastgele konumlardan gelerek yakalamaya çalışan böceklerden kaçıyor ve bulduğu meyveleri yiyerekte puanlar topluyordu. Tam oyunu bitirmeme az kalmıştıki hiç beklenmedik bir şekilde böceklerden birisi tarafından yendim. Aslında ekrana bir süre donuk gözler ile bakmıştım. Nitekim, oyunu oynarken aklıma geçen gün okuduğum Oyun Programlama kitabı gelmişti.

Kitabın bir bölümünde, ekranda yer alan aynı boyutlu nesnelerin çarpışmalarının kontrolünde iki boyutlu bir matris dizisinden faydalanılıyordu. O anda, çarpışma tekniklerinin farklı bir teoremini Packmanâ€™a benzeyecek bir oyun ile inceleyebileceğimi düşündüm. Her ne kadar amacım tümüyle bir oyunu yazmak olmasada en azından Packmanâ€™ imi ekrandaki duvarların içinden geçirmeyecek şekilde hareket ettirmek istiyordum.

Teori gayet basitti. İlk olarak oyun sahasını, eşit boyutlu karelere bölecektim. Bu karelerin boyutları oyundaki nesnelerin çevresini saran hayali karelerinki ile aynı olacaktı. Böylece bir kare içinde her zaman tek bir oyun elemanı tam olarak sığmış bulunacaktı. Daha sonra ekrandaki elemanları iki boyutlu bir matris dizisi içinde bir şekilde konumlandıracaktım. Son olarak, kahramanın her hareketinde yöne bağlı olaraktan önceki veya sonraki kare alanlarını kontrol edecek ve orada bir Duvar nesnesi var ise çarpışma olduğunu belirterek o yöne olan hareketi kesecektim.

Teoremin kilit noktası, oyun alanındaki karelerde bulunan elemanları, iki boyutlu bir Matris dizisi içerisinde temsil edebilmekti.

İlk olarak aşağıdaki gibi bir oyun alanını düşündüm.



Şekil 1. Oyun Sahası

Öncelikle, Duvar, Muz ve kahramanımız Packoâ€™ ya ait imajları tasvir ettim. Bunların her birisi 20 piksel X 20 Piksel boyutlarındaki bir karenin iç kenarlarına teğer olacak büyüklükteydiler. Daha sonra, oyun sahamı 20 Piksel X 20 Pikselâ€™ lik kareler ile doldurdum. Dolayısıyla artık elimde, 20â€™ ye 20â€™ lik bir Matris vardı. Bu Matris yardımıyla ekrandaki her bir elemanın konumunu bilebilirdim. Tek yapmam gerken Matrisin ilgili elemanına, orada duran nesneyi temsil edecek sayısal bir değer vermekti. Örneğin şu anki haliyle, Packoâ€™ nun Matrisâ€™ deki konumunu aşağıdaki gibi ifade edebilirdim.

Matris[5,8]=1;

Bu durumda, Packoâ€™ nun sağa doğru olan hareketinde herhangibir duvara çarpıp çarpmadığını kontrol etmek için, Y değerinin 1 fazlasına bakmak yeterli olacaktı.

Yani;

eğer Matris[5,8+1]=2 ise (ki Duvarlarıda 2 sayısı ile ifade edebilirdim.)
Çarpışma var, Sağa gitme.
eğer Matris[5,8+1]=3 ise (ki Muzları 3 ile ifade edebilirdim.)
Sağa git, Matris[5,9] daki Muzu ekrandan sil, Puanı arttır.
eğer Matris[5,8+1]=0 ise (ne duvar ne de Muz var ise.)
Çarpışma yoktur. Yola devam et.

Bu kontrolü Packoâ€™ nun yapacağı doğrusal her hareket için uygulayabilirdim. Böylece ,

Hareket yönüne göre bir sonraki adımda yer alan elemanları Matris dizisi içinde bularak çarpışma kontrolünü gerçekleştirebilirdim.

Teoremi kafamda pekiştirdikten sonra, sıra bunu uygulamaya dökmeye gelmişti. Elbetteki bir Windows uygulaması için böyle bir teoremi araştırmaya çalışırken bir takım zorluklar ile karşılaşabilirdim. Örneğin, oyun başladığında Duvarların, Muzların ve Packonun rastgele ekrana konumlandırılması. Ekranda piksel bazında tutulan karesel alanların, Matris dizisi içerisinde nasıl indislendirilebileceği. Öyle ya, 400 pikselâ€™ e 400 pikselâ€™ lik bir Form alanını, 400*400 elemanlı bir Matris dizisinde aynen uygulamak gereksiz yere hafıza tüketimine neden olurdu. Ya da, ekrandaki bir Muzâ€™ un üstünden geçildiğinde o resmin nasıl kaldırılacağı. Bu gibi pek çok sorunu önceden düşünmek ve uygulamayı ona göre planlamak gerekiyordu. Bu amaçla önce düşündüm ve sonra aşağıdaki kodları geliştirdim.

Konumlandırma işlemleri için kullandığım sınıf,

```
using System;
namespace Packo
{
   /* Bu sinif yardimiyla oyun alaninda kullandigimiz nesnelerin X ve Y koordinatlari ve duvar sayilari için rastgele degerler ürettiriyoruz. Ekranimizi 20â€™ ye 20â€™e lik karelere ayirdigimiz için random sinifinin Next metodunu buna uygun sekilde çagiriyoruz. */
   public class Konumlandir
   {
      private int X,Y,duvarSayisi;
      System.Random r;
      public Konumlandir()
      {
         r=new Random();
      }
```

```
        /* X koordinatlari için (herhangibir oyun elemaninin Left özelliginin degeri için) bir
özellik tanimliyoruz. Bu özellik Read-Only formatindadir. */
        public int YerlestirX
        {
           get
           {
              X=r.Next(1,20)*20; /* Oyun alani 400 piksele 400 piksel boyutunda oldugu için,
1 ile 20 arasindaki rakami 20 kat sayisi ile çarpiyoruz.*/
              return X;
           }
        }
        /* Y koordinatlari için (herhangibir oyun elemaninin Top özelliginin degeri için) bir
özellik tanimliyoruz. Bu özellik Read-Only formatindadir. */
        public int YerlestirY
        {
           get
           {
              Y=r.Next(1,20)*20;
              return Y;
           }
        }
        /* Ekrana 10 ile 30 arasinda restgele bir degerde Duvarlar koymak için bu özelligi
kullaniyoruz. Bu özellik Read-Only formatindadir. */
        public int DuvarSayisi
        {
           get
           {
              duvarSayisi=r.Next(10,30);
              return duvarSayisi;
           }
        }
    }
}
```

Ana program;

```
Konumlandir k;
/* Ekrani 20*20 lik bir matris ile ele alacagiz. */
int[,] Matris=new int[20,20];
/* Matristeki her bir elemanin hangi oyun nesnesini (duvar,kahramanimiz packocuk ve
Muz) temsil ettigini daha kolay kontrol edebilmek için sayisal degerleri bir enum sabiti ile
anlamlandiriyoruz.*/
enum AlanSahibi
{
   Pakocuk=1,
```

```
    Duvar=2,
    Muz=3
}
/* Ekrana duvar elemani, rastgele koordinatlara gelecek sekilde ekleniyor.*/
private void DuvarEkle()
{
    /*Bir PictureBox nesnesi tanimlaniyor.*/
    PictureBox pb=new PictureBox();
    /*Nesnemizin içerecegi resim yükleniyor. */
    pb.Image=System.Drawing.Image.FromFile("duvar.jpg");
    /*Nesnemizin ekrandaki yerlesimi için gerekli koordinat ayarlamalari yapiliyor.*/
    pb.Top=k.YerlestirY;
    pb.Left=k.YerlestirX;
    /* Duvarin ekrandaki piksel bazli koordinatlarini Matrisimizdeki elemanlar ile
uyusturabilmek için 20 ile bölüyoruz.     Matrisin bu elemanina Duvar enum sabitinin
degerinin veriyoruz.*/
    Matris[pb.Left/20,pb.Top/20]=(int)AlanSahibi.Duvar;
    /* Nesnemizin boyutlari belirleniyor.*/
    pb.Width=20;
    pb.Height=20;
    pb.SizeMode=PictureBoxSizeMode.StretchImage;
    /* Nesnemiz formumuzun Controls koleksiyonuna ekleniyor.*/
    this.Controls.Add(pb);
}
/* Ekrana bir Muz elemani, rastgele koordinatlara gelecek sekilde yerlestiriliyor.*/
private void MuzEkle()
{
    /*Muz nesnesinin üzerinden geçildiğinde onu ekrandan kaldırabilmek için kontrolün
adını bilmem gerekiyor.Bunun için MuzEkle metodunun adını bulunduğu koordinata göre
tanımlıyoruz.*/
    string ad;
    PictureBox pb=new PictureBox();
    pb.Image=System.Drawing.Image.FromFile("muz.jpg");
    pb.Top=k.YerlestirY;
    pb.Left=k.YerlestirX;
    Matris[pb.Left/20,pb.Top/20]=(int)AlanSahibi.Muz;
    ad="MUZ_"+Convert.ToString((pb.Left/20))+"_"+Convert.ToString((pb.Top/20));
    pb.Name=ad;
    pb.Width=20;
    pb.Height=20;
    pb.SizeMode=PictureBoxSizeMode.StretchImage;
    this.Controls.Add(pb);
}
/* Kahramanimiz Packoâ€™nun ekrandaki konumu, Matris dizisindeki yeri ve saga veya
sola bakacagi resmi belirleniyor. Ayrica, ekrana DuvarSayisi kadar Duvar ve Muz
```

```
elemanlari ekleniyor.*/
private void Baslat()
{
   /* Rastegele X,Y ve duvar sayilari için kullandigimi Konumlandir sinifina ait bir nesne
örnegi olusturuluyor.*/
   k=new Konumlandir();
   resPacko.Left=k.YerlestirX;
   resPacko.Top=k.YerlestirY;
   Matris[resPacko.Left/20,resPacko.Top/20]=(int)AlanSahibi.Pakocuk;
   /*Egere resPacko ekranin sol yarim küresinde ise, Sola bakan resmi gösteriliyor.*/
   if(resPacko.Left<200)
      resPacko.Image=System.Drawing.Image.FromFile("packoSol.jpg");
   /*Egere resPacko ekranin sag yarim küresinde ise, Saga bakan resmi gösteriliyor.*/
   if(resPacko.Left>200)
      resPacko.Image=System.Drawing.Image.FromFile("packoSag.jpg");
   resPacko.Visible=true;
   for(int i=1;i<k.DuvarSayisi;i++)
   {
      DuvarEkle();
      MuzEkle();
   }
}
int carpismaDurumu=0;
/* Eğer üstünden geçtiğimiz nesne Muz ise onu ekrandan kaldırıyoruz. Bunu yaparken
Form üzerindeki kontrollerde gezinip, kontrolün adını buluyor ve Remove metodunu
çağırıyoruz.*/
private void MuzYokEt(int X,int Y)
{
   /*Önce Matirsimizin X,Y elemanının Muz olup olmadığına bakıyoruz.*/
   if(Matris[X,Y]==(int)AlanSahibi.Muz)
   {
      /*Eğer Muz ise dizinin bu elemanının değerini 0 yapıyoruz. Böylece Muz nesnemizi
diziden çıkarmış oluyoruz.*/
      Matris[X,Y]=0;
      /* Daha sonra PictureBoxâ€™ ımızın adını tedarik ediyoruz. */
      string muzX=X.ToString();
      string muzY=Y.ToString();
      string KontrolAdi="MUZ_"+muzX+"_"+muzY;
      /*Döngü ile, Form içindeki tüm kontroller arasında geziniyoruz.*/
      for(int i=0;i<this.Controls.Count;i++)
      {
         /*Eğer güncel kontrolün adı, bizim Muz nesnemizinki ile aynı ise, bu
PictureBoxâ€™ ı Formumuzdan çıkartıyoruz.*/
         if(this.Controls[i].Name.ToString()==KontrolAdi)
         {
```

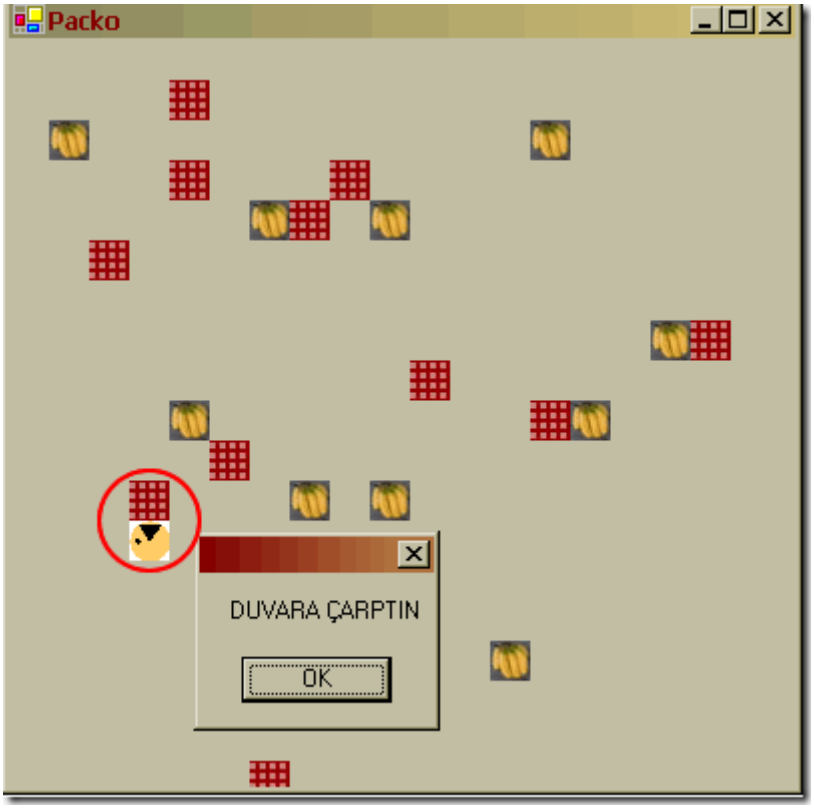
```
            this.Controls.Remove(this.Controls[i]);
          }
        }
    }
}
/* Çarpisma kontrolünün yapildigi metod. Bu metod Packoâ€™nun X, Y koordinatlari ile hareket ettigi yönü (saga,sola,yukariya,asagiya) parametre olarak aliyor. Aldigi X,Y koordinatlari ve yöne göre, bir sonraki kare alaninda bir Duvar nesnesi olup olmadigina bakiyor.*/
private void CarpismaKontrol(int X,int Y,char Yon)
{
    /*Eger sola hareket ediyorsak ve Matrisimizin [X indisinin 1 önceki elemani,Y] AlanSahibi.Duvar enum sabitinin degerine esit ise çarpisma vardir. */
    if(Yon==â€™Aâ€™)
    {
        int alanSahibi=Matris[X-1,Y];
        if(alanSahibi==(int)AlanSahibi.Duvar)
            carpismaDurumu=1;
        else
            carpismaDurumu=0;
        MuzYokEt(X,Y);
    }
    /*Eger saga hareket ediyorsak ve Matrisimizin [X indisinin 1 sonraki elemani,Y] AlanSahibi.Duvar enum sabitinin degerine esit ise çarpisma vardir. */
    if(Yon==â€™Dâ€™)
    {
        int alanSahibi=Matris[X+1,Y];
        if(alanSahibi==(int)AlanSahibi.Duvar)
            carpismaDurumu=1;
        else
            carpismaDurumu=0;
        MuzYokEt(X,Y);
    }
    /*Eger asagi hareket ediyorsak ve Matrisimizin [X,Y indisinin 1 sonraki elemani] AlanSahibi.Duvar enum sabitinin degerine esit ise çarpisma vardir. */
    if(Yon==â€™Sâ€™)
    {
        int alanSahibi=Matris[X,Y+1];
        if(alanSahibi==(int)AlanSahibi.Duvar)
            carpismaDurumu=1;
        else
            carpismaDurumu=0;
        MuzYokEt(X,Y);
    }
/*Eger yukari hareket ediyorsak ve Matrisimizin [X,Y indisinin 1 önceki elemani]
```

```
AlanSahibi.Duvar enum sabitinin degerine esit ise çarpisma vardir. */
   if(Yon==â€™Wâ€™)
   {
      int alanSahibi=Matris[X,Y-1];
      if(alanSahibi==(int)AlanSahibi.Duvar)
         carpismaDurumu=1;
      else
         carpismaDurumu=0;
      MuzYokEt(X,Y);
   }
}
/*Form üzerinde A (sol), S (asagi), D (saga), W (yukari) tuslarina basildikça çarpisma
kontrolü yapiliyor. Eger çarpisma var ise, belirtilen yöndeki dogrusal harekete (20 piksellik
öteleme) izin verilmiyor. Aksi halde harekete devam ediliyor.*/
private void frmPacko_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
{
   int x=resPacko.Left/20;
   int y=resPacko.Top/20;
   if(e.KeyChar==(Char)Keys.A)
   {
      resPacko.Image=System.Drawing.Image.FromFile("packoSol.jpg");
      CarpismaKontrol(x,y,â€™Aâ€™);
      if(carpismaDurumu==1)
         MessageBox.Show("DUVARA ÇARPTIN");
      else
         resPacko.Left-=20;
   }
   if(e.KeyChar==(Char)Keys.D)
   {
      resPacko.Image=System.Drawing.Image.FromFile("packoSag.jpg");
      CarpismaKontrol(x,y,â€™Dâ€™);
      if(carpismaDurumu==1)
         MessageBox.Show("DUVARA ÇARPTIN");
      else
         resPacko.Left+=20;
   }
   if(e.KeyChar==(Char)Keys.S)
   {
      resPacko.Image=System.Drawing.Image.FromFile("packoAsagi.jpg");
      CarpismaKontrol(x,y,â€™Sâ€™);
      if(carpismaDurumu==1)
         MessageBox.Show("DUVARA ÇARPTIN");
      else
         resPacko.Top+=20;
```

```
    }
    if(e.KeyChar==(Char)Keys.W)
    {
        resPacko.Image=System.Drawing.Image.FromFile("packoYukari.jpg");
        CarpismaKontrol(x,y,â€™Wâ€™);
        if(carpismaDurumu==1)
            MessageBox.Show("DUVARA ÇARPTIN");
        else
            resPacko.Top-=20;
    }
}
/*Oyunu kapatmak ve baslatmak kullandigimiz menüleri ContextMenu içerisinde
kullaniyoruz.*/
private void menuItem3_Click(object sender, System.EventArgs e)
{
    Close();
}
private void menuItem1_Click(object sender, System.EventArgs e)
{
    Baslat();
}
/* Kontrol amaciyla, Matris dizisinin degerlerini Text tabanli bir dosyaya da
yazabiliyoruz.*/
private void menuItem4_Click(object sender, System.EventArgs e)
{
    System.IO.FileStream fs=new
System.IO.FileStream("kontrol.txt",System.IO.FileMode.OpenOrCreate,System.IO.FileAc
cess.Write);
    System.IO.StreamWriter sw=new System.IO.StreamWriter(fs);
    for(int i=0;i<20;i++)
    {
        for(int j=0;j<20;j++)
        {
            sw.Write("{0,2},{1,2}={2,3}",i,j,Matris[i,j]);
        }
        sw.WriteLine();
    }
    sw.Flush();
    sw.Close();
    fs.Close();
}
```

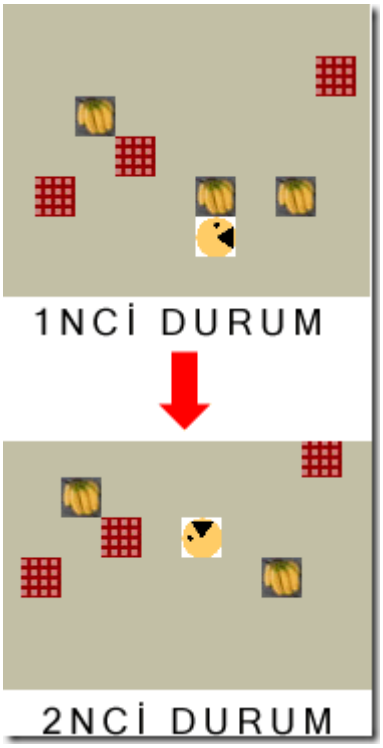
Kodlar her ne kadar uzun görünsede programın tek yaptığı, Packoâ€™yu duvarların içinden geçirmeden hareket ettirmek ve yolda gördüğü Muzâ€™ları toplamasını sağlamak. Örneğin, uygulamayı çalıştırdığımızda bir duvara hangi yönden gelirsek gelelim, Matris

dizimiz içinde Packoâ€™ dan sonraki elemanlar kontrol edilecek ve duvara çarpılıp çarpılmadığına bakılacaktır.



Şekil 2. Duvara Çarpış.

Diğer yandan, eğer Packo bir Muz üzerinden geçerse, bu Muz nesnesini temsil eden PictureBox Form üzerinden kaldırılacak, aynı zamanda Matris dizimizdeki ilgili Muz elemanın değeride sıfırlanacaktır.



Şekil 3. Packo Muzları Yiyebiliyor.

Görüldüğü gibi, Matris tekniği ile eşit karelere bölünmüş sahalardaki nesnelerin birbirleri ile olan çarpışmalarını kontrol edebilmek son derece basit. Bu minik program elbetteki başlangıç aşamasında. Örneğin, her muz yiyişinden sonra puanlama sistemi olması, duvarların seri olacak şekilde ekrana dizilmesi, Packo dışında hareket eden ve Packoâ€™ yu yemeye çalışan böceklerin farklı hareketleri vs... Bu kısımların geliştirilmesinide siz değerli okurlarıma bırakıyorum. Uygulamaya kaldığı yerden devam edebilirsiniz. Bir sonraki makalemizde görüşmek dileğiyle hepinize mutlu günler dilerim.

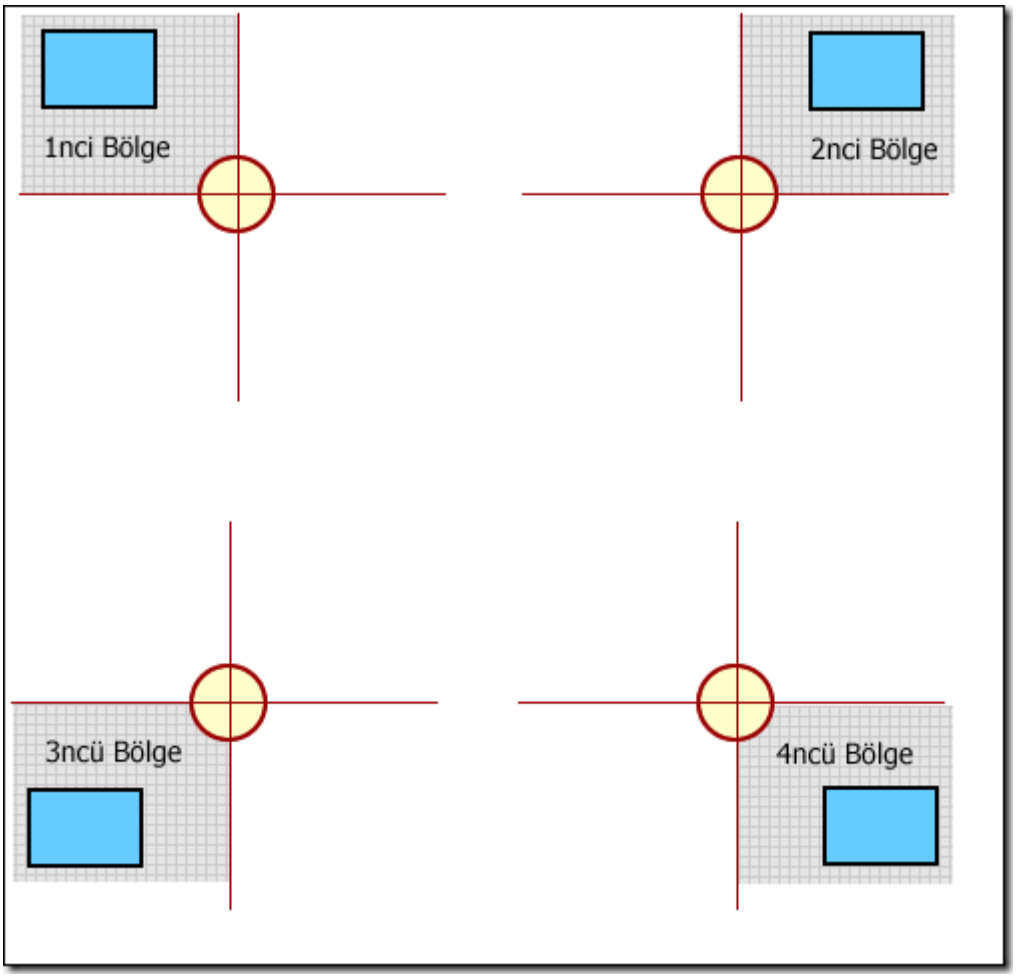
Oyun Programlamaya Giriş (Çarpışma Teknikleri - 3) (2004-11-19T20:06:00)

game programming,c#,

Geçtiğimiz hafta boyunca, Oyun Programcılığı ile ilgili olaraktan aldığım kitapları fırsat buldukça okumaya ve çalışmaya devam ettim. Konular o kadar heyecanlı ve sürükleyici ki araştırmak için zaman kavramı anlmasız hale geliyor. Öyleki, dün gece sabaha karşı saat 03:00 sularında kağıt kalem ile boğuşuyor ve Çarpışma Tekniklerinden birisinin daha matematiksel modelinin C# ile nasıl uygulanabileceğini araştırıyordum. Sonuç olarak işe bir kaç saatlik uykuyla gitmek zorunda kaldım. Ancak buna rağmen tüm gün dinçtim. Çünkü, çarpışma tekniklerinden birisini daha öğrenmiştim. Sıra anlatmaya gelmişti. İşte bugünkü makalemizde 3ncü çarpışma tekniğini incelemeye çalışacağız.

Oyun programcılığında önemli bir yere sahip olan çarpışma tekniklerinde, bu makaleye gelinceye kadar iki ana konuyu inceleme fırsatı bulduk. İlk olarak iki dörtgensel nesnenin bir birleriyle olan çarpışmalarını inceledik. Daha sonraki makalemizde ise, eski dostumuz

Pisagorâ€™u anıp, iki dairesel nesnenin birbirleriyle olan çarpışmalarını araştırdık. Sırada bu iki durumun kombinasyonu var. Yani, bir dörtgen ile dairesel bir nesnenin birbirleriyle olan çarpışmalarının tespit edilmesi. Bu teknikte yine Pisagor teroeminden yararlanacağız. Ancak dikkat etmemiz gereken önemli koordinat noktaları var. Bu durumu daha iyi analiz etmek için aşağıdaki şekli göz önüne alalım.



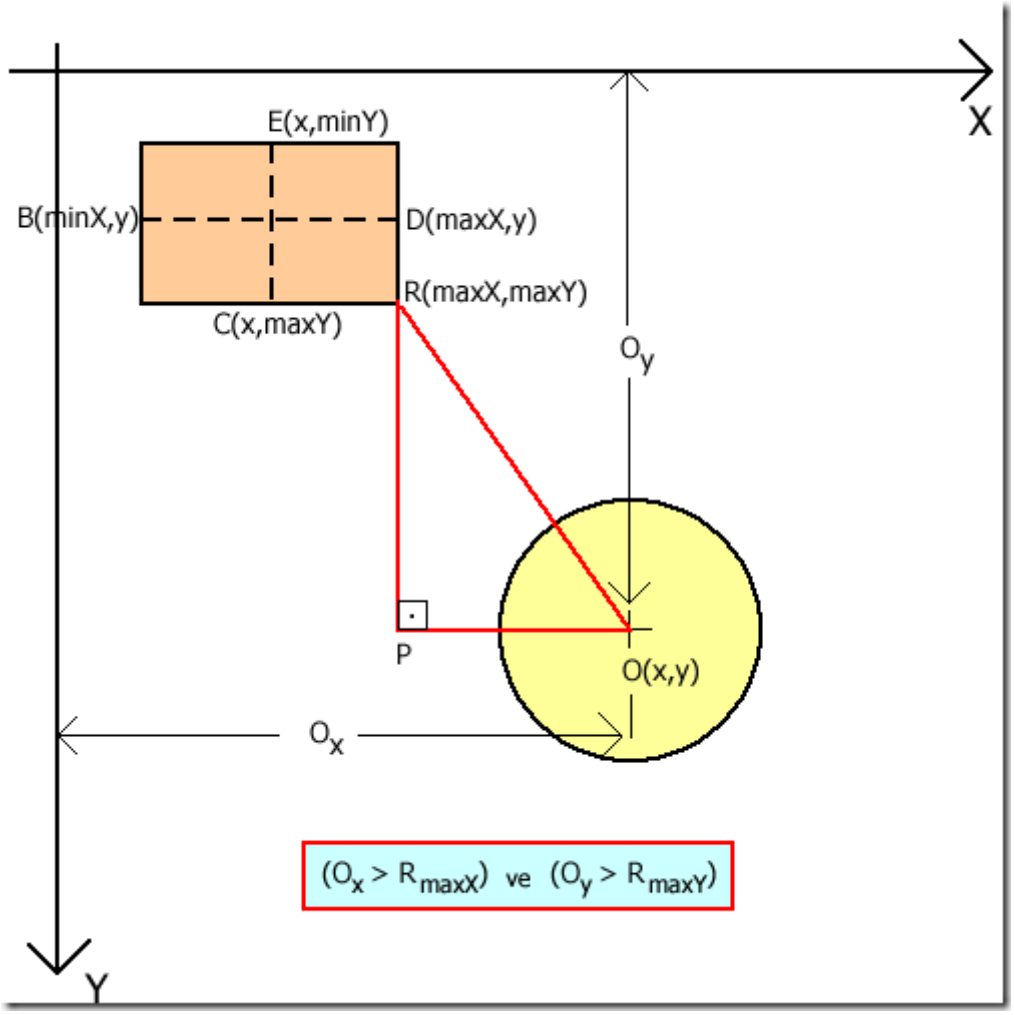
Şekil 1. Dört Bölge.

Bu teoride, dörtgenin dairesel nesneye göre olan konumlarını ele almamız gerekiyor. Bu da bizim, dörtgen nesnesine ait olan maksimum ve minimum sınır noktalarını bilmemiz gerektiğini göstermektedir. Buna göre, dörtgensel nesnenin köşe noktalarının x ve y koordinatları ele alınır. Kısaca, oluşturacağımız pisagor üçgeni için bu maksimum ve minimum x, y koordinatlarını bilmemiz gerekiyor. Çarpışma teoremine gelince;

Dörtgenin dairesel nesneye en yakın olan köşe noktalarından yola çıkılarak oluşturulan dik üçgene ait hipotenüs değeri, dairesel nesnenin yarıçapından küçük ise çarpışma vardır.

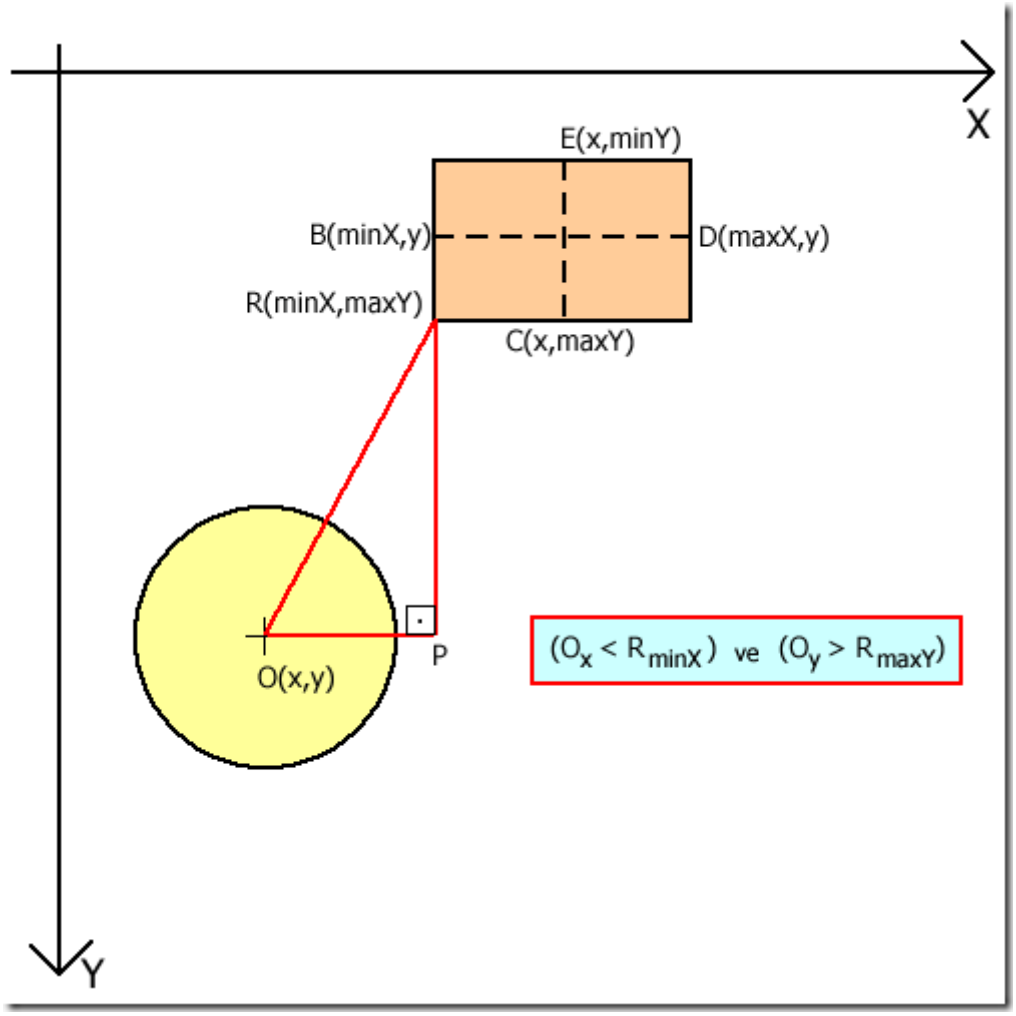
Şimdi buradaki dört bölgeyi kısaca inceleyelim. Örnek olarak 4üncü bölgeyi aşağıdaki şekilde olduğu gibi ele alabiliriz. Esasen tüm bölgelerde, dörtgensel nesneye ait minX,maxX,minY ve maxY koordinatları büyük öneme sahiptir. Örneğin PRO üçgenini

göz önüne aldığımızda, kenarların uzunluklarını bulabilmek için R noktasının koordinatlarından yararlanılmaktadır. R noktasına dikkat edecek olursanız, dörtgenin en uzak X ve en uzak Y değerlerine sahip olduğunu görürsünüz. Dolayısıyla biz, PO uzaklığını bulmak istiyorsak, Ox değerinden RmaxX değerini çıkartmamız gerekecektir. Aynı durum RP kenarının boyunu bulurken de geçerlidir. Bu durumda, Oy değerinden RmaxY değerini çıkartırız. Burada dörtgenin dördüncü bölgeye düştüğünü anlamak için dairenin x,y koordinatları ile R noktasının koordinatları karşılaştırılır. Burada dikkat etmemiz gereken husus, R noktasının koordiantlarını dörtenin daireye olan konumuna göre değişeceğidir.

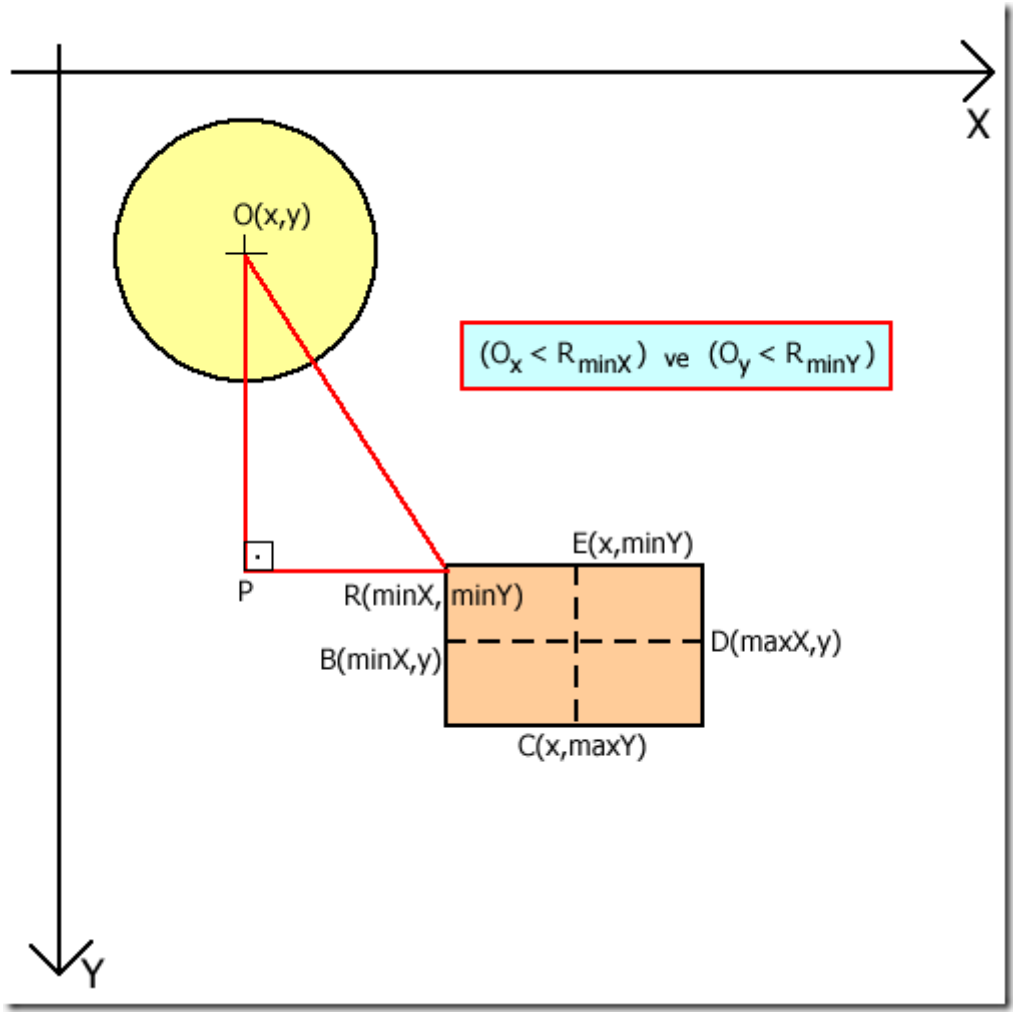


Şekil 2. Dördüncü bölge için durum.

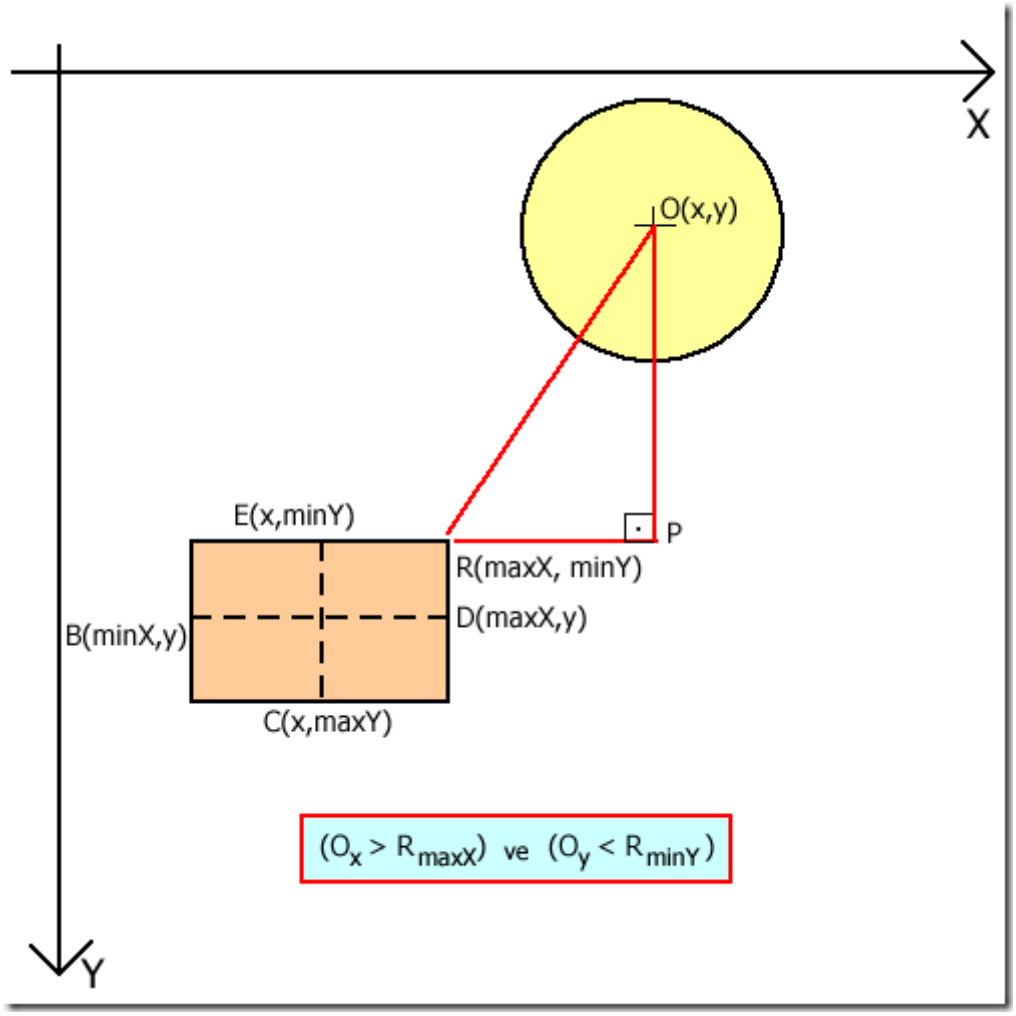
Bu anlattıklarım size biraz karmaşık geldiyse sıkı durun. Diğer bölgelerin grafiksel ifadeleride aşağıda yer almaktadır. Dikkat edecek olursanız, R noktasının koordinatları her bölge için farklıdır. Buda üçgene ait dik kenarlar hesaplanırken uygun R noktası koordinatlarını kullanmamız gerektiğini, başka bir deyişle programlama algoritmasınıda, söz konusu bölgeleri if koşulları ile tespit ederek geliştirmemiz gerektiğini gösterir.



Şekil 3. Üçüncü bölge için durum.

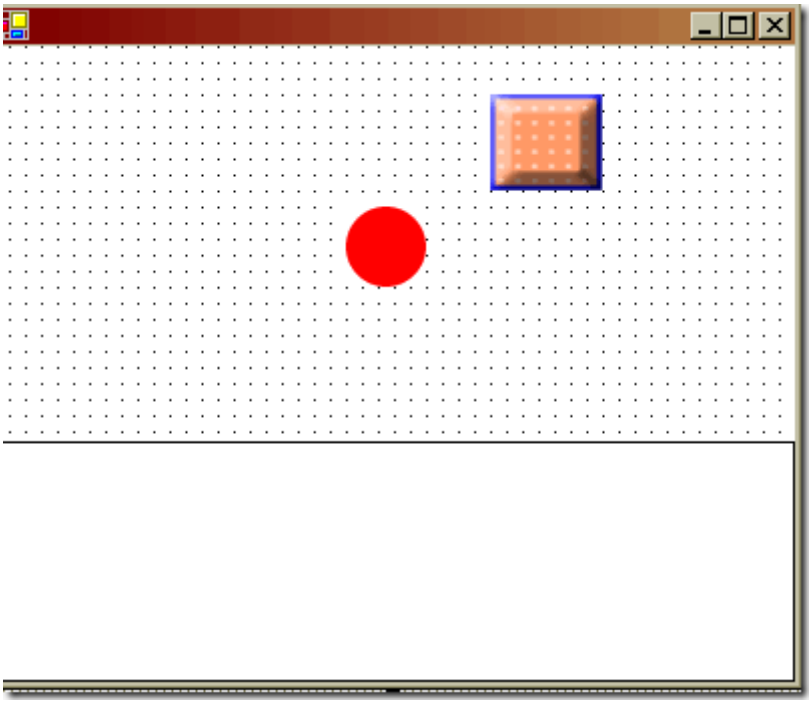


Şekil 4. Birinci bölge için durum.



Şekil 5. İkinci bölge için durum.

Şimdi sıra geldi bu teoremi C# kodları ile gerçekleştirmeye. Bu amaçla aşağıdaki ekran görüntüsüne sahip basit bir windows uygulaması geliştirdim. Her zamanki gibi işlemleri kolaylaştırmak amacıyla, dairesel nesneyi sabit tutup, dörtgensel nesneyi A,S,D,W tuşları ile hareket ettiriyorum. Bizim için önemli olan kısımlar, 4 durumdada geçerli olan üçgene ait dik kenarların FarkX ve FarkY şeklinde hesap edilmeleri ve buradan yola çıkılarak hipotenüs değerlerinin bulunması. Son olarak bu hipotenüs değerini, dairenin yarıçapı ile karşılaştırıp çarpışma olup olmadığını inceliyoruz.



Şekil 6. Uygulama formu.

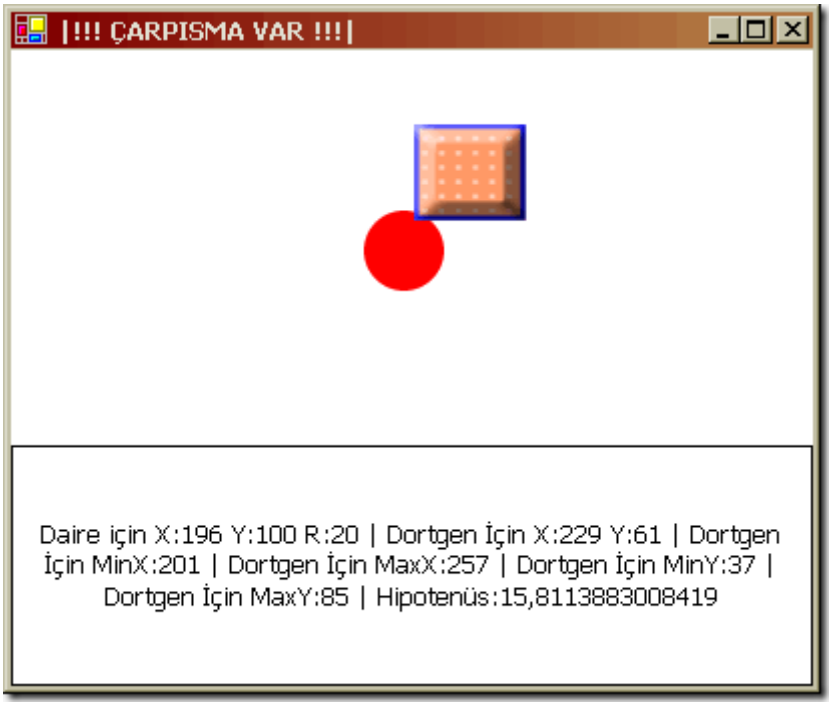
Şimdi gelelim uygulama kodlarımıza.

```
/*global degiskenlerimizi tanimliyoruz. */
float Daire_X,Daire_Y,Dortgen_X,Dortgen_Y,Daire_R;
float Dortgen_MinX,Dortgen_MaxX,Dortgen_MinY,Dortgen_MaxY;
double Hipotenus,FarkX,FarkY;
/*Daire için X,Y koordinatlar, yarıçap Dörtgen için ise X,Y, minimum ve maksimum X,Y
koordinatları hesaplanıyor.*/
private void Hesapla()
{
   Daire_R=Daire.Width/2;
   Daire_X=Daire.Left+Daire_R;
   Daire_Y=Daire.Top+Daire_R;
   Dortgen_X=Dortgen.Left+Dortgen.Width/2;
   Dortgen_Y=Dortgen.Top+Dortgen.Height/2;
   Dortgen_MinX=Dortgen.Left;
   Dortgen_MaxX=Dortgen.Left+Dortgen.Width;
   Dortgen_MinY=Dortgen.Top;
   Dortgen_MaxY=Dortgen.Top+Dortgen.Height;
   /*Burada temel olarak, dörtgenin hangi bölgelere denk düştüğüne bakarak dik üçgene ait
FarkX ve FarkY değerlerini hesap ediyoruz.*/
if(Daire_Y<Dortgen_MinY)
   {
      FarkY=Daire_Y-Dortgen_MinY;
   }
```

```
   if(Daire_Y>Dortgen_MaxY)
   {
      FarkY=Daire_Y-Dortgen_MaxY;
   }
   if(Daire_X>Dortgen_MaxX)
   {
      FarkX=Daire_X-Dortgen_MaxX;
   }
   if(Daire_X<Dortgen_MinX)
   {
      FarkX=Daire_X-Dortgen_MinX;
   }
   /*Dik üçgene ait hipotenüsü buluyoruz.*/
Hipotenus=Math.Sqrt((FarkX*FarkX)+(FarkY*FarkY));
}
/* Ölçümleri ekrana yazdirmak için string deger döndüren bir metod hazirliyoruz.*/
private string Olcumler()
{
   string olcum="Daire için X:"+Daire_X.ToString();
   olcum+=" Y:"+Daire_Y.ToString();
   olcum+=" R:"+Daire_R.ToString();
   olcum+=" | Dortgen İçin X:"+Dortgen_X.ToString();
   olcum+=" Y:"+Dortgen_Y.ToString();
   olcum+=" | Dortgen İçin MinX:"+Dortgen_MinX;
   olcum+=" | Dortgen İçin MaxX:"+Dortgen_MaxX;
   olcum+=" | Dortgen İçin MinY:"+Dortgen_MinY;
   olcum+=" | Dortgen İçin MaxY:"+Dortgen_MaxY;
   olcum+=" | Hipotenüs:"+Hipotenus.ToString();
   return olcum;
}
/* Form üzerinde A,S,D,W tuslarina basildiginda, Dortgen isimli pictureBoxâ€™ in
hareket etmesini sagliyoruz.*/
private void frmCollision2_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
{
   if(e.KeyChar==(Char)Keys.A)
   {
      Dortgen.Left-=1;
      Hesapla();
      lblOlcumler.Text=Olcumler();
   }
   if(e.KeyChar==(Char)Keys.D)
   {
      Dortgen.Left+=1;
      Hesapla();
```

```
        lblOlcumler.Text=Olcumler();
    }
    if(e.KeyChar==(Char)Keys.S)
    {
        Dortgen.Top+=1;
        Hesapla();
        lblOlcumler.Text=Olcumler();
    }
    if(e.KeyChar==(Char)Keys.W)
    {
        Dortgen.Top-=1;
        Hesapla();
        lblOlcumler.Text=Olcumler();
    }
    Kontrol();
}
/* Çarpisma kontrolümüzü yapiyoruz. */
private void Kontrol()
{
    if(Hipotenus<Daire_R)
    {
        this.Text="";
        this.Text+=" |!!! ÇARPISMA VAR !!!| ";
    }
    else
        this.Text="";
}
```

Kodlarımız son derece açık ve kolay. Uygulamamızı çalıştırdığımızda, çarpışma teorimizin başarılı bir şekilde çalıştığını görürüz.



Şekil 7. Çarpışma durumu.

Görüldüğü gibi artık oyun programlamada önemli noktalardan birisi olan çarpışma teorilerinde bayağı bir yol aldık. Ancak her zaman bu teknikleri kullanmayacağız. Örneğin zaman zaman, ekranın eşit karelere bölündüğünü (aynı kareli defterler gibi) ve nesnelerin bu kareler üzerindeki konumlarına göre çarpışıp çarpışmadıklarının belirlendiğini öğrendim. Önümüzdeki hafta büyük bir ihtimalle bu konuyu incelemeye çalışacağım. Kaynaklardan incelediğim kadarı ile bu karelere bölme tekniği tam anlamıyla PackMan tarzı oyunlara yönelik geliştirilmiş bir model. İşin içine bu kez matris dizileri girecek. Bakalım başımıza neler gelicek. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

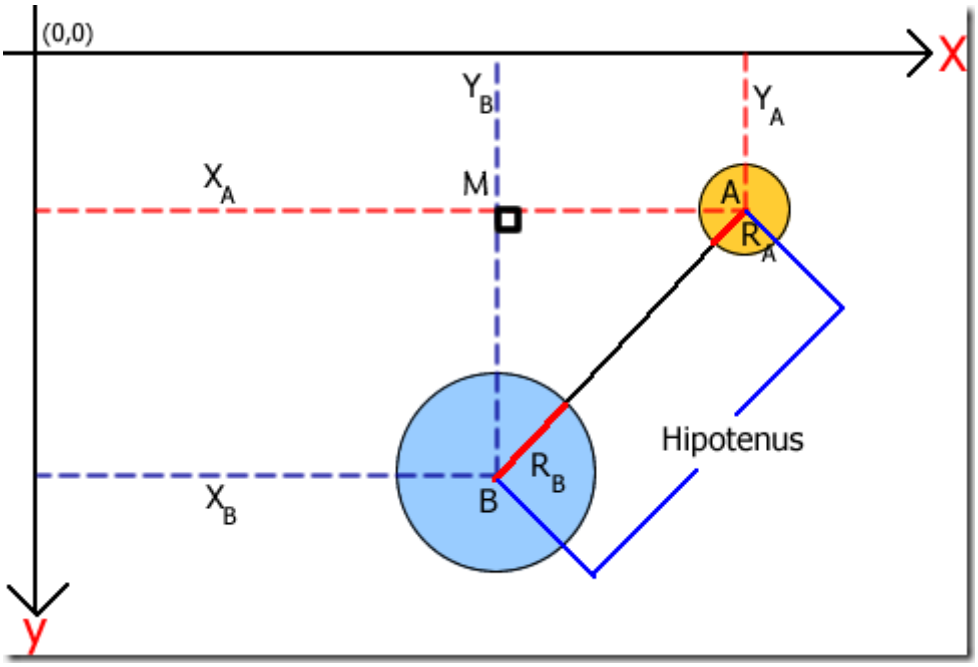
Oyun Programlamaya Giriş (Çarpışma Teknikleri - 2) (2004-11-12T19:56:00)

game programming,c#,

Hatırlayacağınız gibi bir önceki makalemizde, Oyun Programcılığına girmek adına çarpışma tekniklerini incelemeye başlamış ve dörtgenlerin çarpışmalarını ele almıştık. Bugünkü makalemizde ise, dairesel şekillerin birbirleri ile olan çarpışmalarını incelemeye çalışacağız. Dairesel şekillerin çarpışmasına verilebilecek en güzel örnek, kaynaklardan edindiğim bilgiye göre Bilardo oyunlarıdır. Burada gerçekten de mükemmel dairelerin birbirleriyle olan çarpışmaları söz konusudur. Şunuda hatırlatmakta fayda var.

Şu an için teorilerimizi iki boyutlu uzayda inceliyoruz. Elbetteki işin için üç boyutlu cisimler girdiğinde kullanacağımız algoritmalar ve teknikler birazda olsa farklılık gösterecektir. çünkü uzay boyutunda X ve Y koordinatlarına ek olarak Z koordinatlarıda işin içine girecektir. Bu da iki boyutlu bir sistemde Bilardo oyunun tasarlanmasının 3 boyutlu sistemdekine göre daha kolay olduğunu göstermektedir.

Dairesel nesnelerin çarpışmalarını belirlemek için eski dostumuz Pisagor Teoreminden faydalanacağız. Burada ana fikir, dairelerin merkezlerinin birbirlerine olan doğrusal uzaklıkları ile yarıçaplarının toplamlarının karşılaştırılmasıdır. Eğer, dairelerin merkezleri arası doğrusal uzaklık, dairelerin yarıçapları toplamından küçük ise, dairelerin üst üste geldiklerinden dolayısıyla çarpıştıklarından söz edebiliriz. Olayı aşağıdaki şekil ile ele almaya çalışalım.



Şekil 1. Dairelerin çarpışması.

Burada bizim için anahtar şekil, dairelerin merkezleri arasında oluşan BMA dik üçgenidir. Biz bu üçgen yardımıyla, A ve B noktaları arasındaki mesafeyi, başka bir deyişle BMA üçgeninin hipotenüsünü bulabiliriz. Bu bizim için en kilit noktadır. Sonrasında ise, bu mesafeyi Ra ve Rb yarıçaplarının toplamı ile karşılaştırmamız yeterli olacaktır. Dolayısıyla çarpışma formülümüz aşağıdaki gibi olmalıdır.

$$|X_A - X_B| = |MA|$$

$$|Y_A - Y_B| = |MB|$$

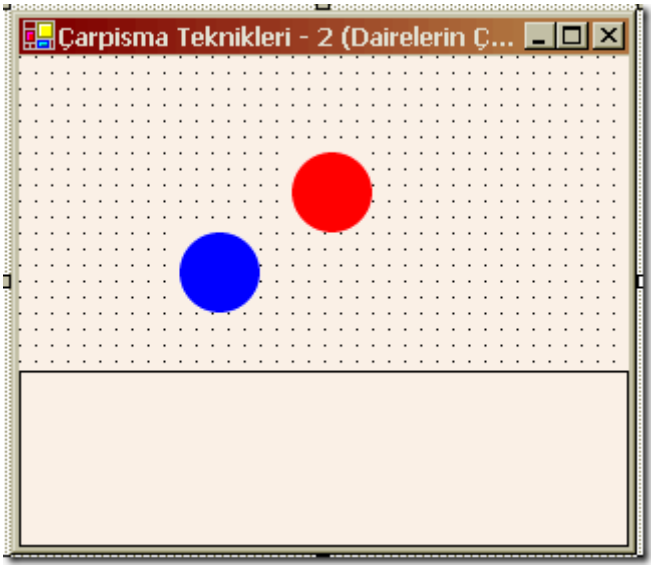
$$|AB| = \sqrt{|MA|^2 + |MB|^2}$$

$$|AB| < R_A + R_B \Rightarrow \text{Çarpışma Var.}$$

Şekil 2. çarpışma Teorimiz.

Görüldüğü gibi yapmamız gereken, X ve Y koordinatları arasındaki mesafelerden yararlanarak, üçgenin hiptoneüsünü (yani iki daire merkezi arasındaki doğrusal uzaklığı) bulmak ve bulduğumuz değeri, yarıçapların toplamı ile karşılaştırmaktır. Şimdi dilerseniz,

bu teoriyi C# ile geliştirilmiş bir windows uygulamasında nasıl simule edeceğimize bakalım. Geliştireceğimiz uygulama her zamanki gibi basit ve anlamsız olacak. Ancak amacımız, yukarıdaki teoremi uygulamak ve sonuçlarını görmek. Bu amaçla aşağıdaki ekran görüntüsüne ait bir windows uygulaması geliştirdim.



Şekil 3. Uygulama tasarımımız.

Kodlarımız ise aşağıdaki gibi olacaktır.

```
/*global değişkenlerimizi tanımlıyoruz. */
float Top1_X,Top1_Y,Top2_X,Top2_Y,Top1_R,Top2_R;
double Hipotenus,YaricapToplam;
/*X,Y Koordinatlari ile yariçaplar belirleniyor.*/
private void Hesapla()
{
   Top1_R=Top1.Width/2;
   Top1_X=Top1.Left+Top1_R;
   Top1_Y=Top1.Top+Top1_R;
   Top2_R=Top2.Width/2;
   Top2_X=Top2.Left+Top2_R;
   Top2_Y=Top2.Top+Top2_R;
   float Fark_X=Math.Abs(Top1_X-Top2_X);
   float Fark_Y=Math.Abs(Top1_Y-Top2_Y);
Hipotenus=Math.Sqrt((Fark_X*Fark_X)+(Fark_Y*Fark_Y));
YaricapToplam=Top1_R+Top2_R;
}
/* ölçümleri ekrana yazdırmak için string değer döndüren bir metod hazırlıyoruz.*/
private string Olcumler()
{
   string olcum="Top 1 için X:"+Top1_X.ToString();
```

```
    olcum+=" Y:"+Top1_Y.ToString();
    olcum+=" R:"+Top1_R.ToString();
    olcum+=" | Top 2 için X:"+Top2_X.ToString();
    olcum+=" Y:"+Top2_Y.ToString();
    olcum+=" R:"+Top2_R.ToString();
    olcum+=" | Hipotenüs:"+Hipotenus.ToString();
    return olcum;
}
private void frmCollision2_Load(object sender, System.EventArgs e)
{
    Hesapla();
    lblOlcumler.Text=Olcumler();
}
/* Form üzerinde A,S,D,W tuşlarına basıldığında, Top2 isimli pictureBox' ın hareket
etmesini sağlıyoruz.*/
private void frmCollision2_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
{
    if(e.KeyChar==(Char)Keys.A)
    {
        Top2.Left-=1;
        Hesapla();
        lblOlcumler.Text=Olcumler();
    }
    if(e.KeyChar==(Char)Keys.D)
    {
        Top2.Left+=1;
        Hesapla();
        lblOlcumler.Text=Olcumler();
    }
    if(e.KeyChar==(Char)Keys.S)
    {
        Top2.Top+=1;
        Hesapla();
        lblOlcumler.Text=Olcumler();
    }
    if(e.KeyChar==(Char)Keys.W)
    {
        Top2.Top-=1;
        Hesapla();
        lblOlcumler.Text=Olcumler();
    }
    Kontrol();
}
/* çarpışma kontrolümüzü yapıyoruz. */
```

```
private void Kontrol()
{
   if(Hipotenus<YaricapToplam)
   {
      this.Text="";
      this.Text+=" |!!! çARPISMA VAR !!!| ";
   }
   else
      this.Text="";
}
```

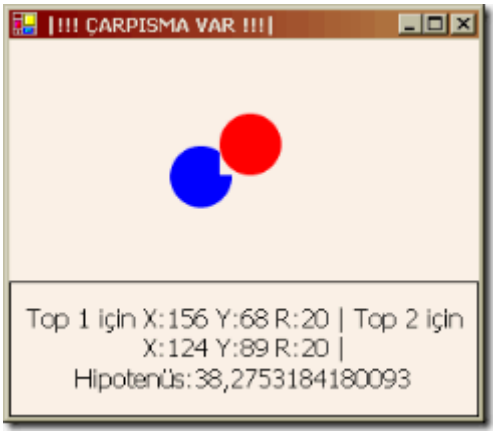
Kodlarımız son derece açık. Dikkat etmemiz gereken noktalardan birisi, Top1 ve Top2 isimli nesnelerin X ve Y koordinatlarının bulunmasıdır. Burada Left ve Top özelliklerinin yanısıra daire merkezlerini tam olarak bulabilmek için, Width veya Height (daire olduklarından X veya Y' ye eklenecek mesafelerin Width veya Height ile hesaplanması farketmez) değerlerinden birisinide göz önüne almamız gerekir. Yani aşağıdaki kodlarda olduğu gibi;

```
Top1_R=Top1.Width/2;
Top1_X=Top1.Left+Top1_R;
Top1_Y=Top1.Top+Top1_R;
Top2_R=Top2.Width/2;
Top2_X=Top2.Left+Top2_R;
Top2_Y=Top2.Top+Top2_R;
```

Bunun dışında Hipotenüs hesaplamasında elbetteki karekök almak için Math sınıfının Sqrt fonksiyonundan yaralanmaktayız.

```
Hipotenus=Math.Sqrt((Fark_X*Fark_X)+(Fark_Y*Fark_Y));
```

Uygulamamızı çalıştırdığımızda, Top2 isimli nesneyi herhangibir yönden Top1 isimli nesne üstüne getirirsek çarpışmanın meydana geldiğini kolayca tespit edebiliriz.



Şekil 4. çarpışmanın çalışma zamanında tespit edilmesi.

çarpışmalar ile ilgili bir diğer önemli durumda, dörtgenler ile dairelerin çarpışmalarının nasıl tespit edilebileceğidir. Bu kez, daire merkez nesne olarak düşünülür ve dörtgenin daireye olan en yakın ve en uzak noktaları değerlendirilerek çarpışmanın olup olmadığına bakılır. Bu teoriyide bir sonraki makalemizde incelemeye çalışacağız. Tekrardan görüşünceye dek hepinize mutlu günler dilerim.

Oyun Programlamaya Giriş (Çarpışma Teknikleri - 1) (2004-11-05T19:53:00)

c#,game programming,

Yaklaşık bir ay kadar önce evde dinlenirken, şu ana kadar yaptığım işleri ve projeleri düşündüm. Kesin olarak şunu söyliyebilirim ki, profesyonel anlamda ilgilendiğim ve kullandığım tek dil C# idi. C# dilini kullanarak, .Net platformu altında veritabanı ağırılıklı olmak üzere çeşitli çalışmalar yaptım. Ancak bir süre sonra farkettim ki, bir Matematik Mühendisi olarak lisans eğitimim sırasında gördüğüm o devasa cebir problemleri, teorem ispatları hiç bir zaman işin içine girmemişti. Matematiğin belkide çok az olmakla birlikte dört işleminin ve bir takım algoritmalar için gerekli iteratif ifadelerinin yer aldığı uygulamalar dışında, onu çok yoğun şekilde kullanmamıştım.

Belki finansal veya istatistiki bir projede yoğun olarak ekonomi teoremlerini ve bunlara bağlı olarak matematiksel denklemleri kullanmak gerekmekteydi ancak çok yoğun bir şekilde bunları kurcalamamıştım. Düşündükçe, günümüz dünyasında artık algoritmalar ile, matematik ile uğraşan yazılımcıların azaldığı kanısına vardım. Hepimizde matematik temeli var. Hepimiz bu konuda çeşitli sınavlardan geçtik. Ama sonuç itibariyle çok az projede, daha önceden gördüğümüz eşsiz teoremleri kullandık. O dakikalarda neden bu yöne eğilmiyorum, biraz eğlenceli hatta matematik yüklü çalışmalara el atmıyorum diye düşünmeye başladım. Derken çözümü son derece eğlenceli ama bir o kadarda önemli bir sahada buldum. Oyun Programcılığı.

çoğumuz, yazılım geliştirirken bir gün büyük bir oyunu yazan ekibin içinde olmayı, onun geliştiricilerinden birisi olarak anılmayı hayal etmişizdir. Bu çok çalışmayı ve birazda zeki olmayı gerektiren bir kişilik ister. Kendi kendime düşünürken, elbette günümüzün popüler oyunlarından birisini yazan herhangibir ekipte olabilmek için çok erken olduğunu zaten biliyordum. Ama en azından bir oyun için gerekli en temel bileşene biraz da olsa aşinalığım vardı. Matematik. Oyun programlama içerisinde, oyun motorlarının geliştirilmesinden aksiyonların uygulanışına, yapay zeka taktiklerinden stratejik karar mekanizmalarına kadar her aşamada Matematiksel algoritmaların yer aldığını gayet iyi biliyordum. Benim için bunları öğrenmeye çalışmak, uygulamak ve denemek, benim için heyecan verici olacaktı. Diğer taraftan eski Matematik günlerimi hatırlamış bir başka deyişle saksıyı biraz daha çalıştırmış olacaktım.

Büyük bir hevesle bir taslak plan hazırladım ve işe koyuldum. Bana öncelikle oyun programlamadaki temel teknikleri başlangıç seviyesinden itibaren anlatacak ve yeri

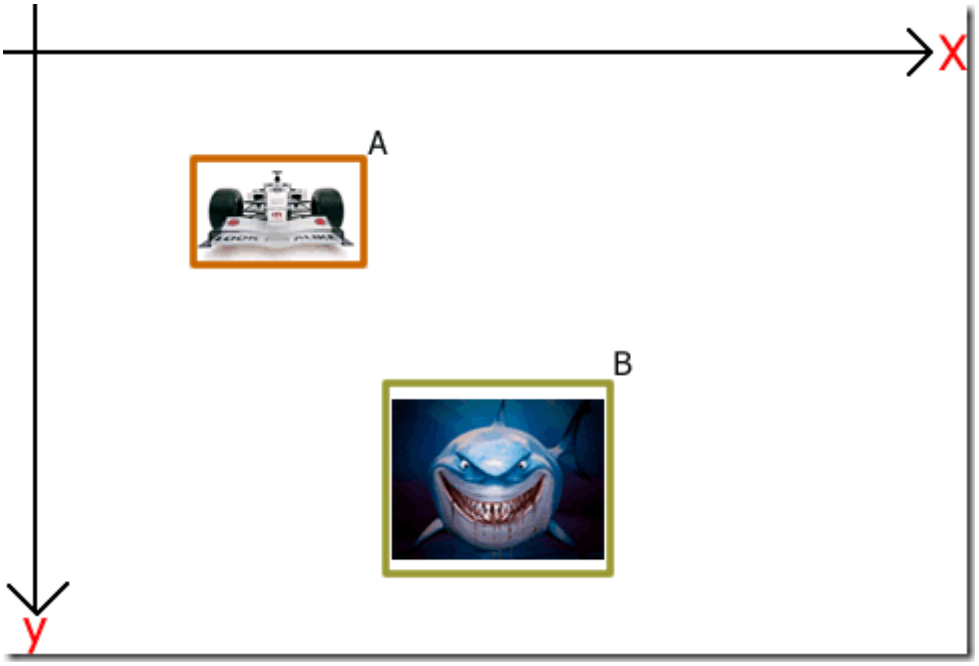
geldiğinde de uzmanlık seviyesine kadar çıkartacak kitaplar gerekliydi. Hemen Amazon.com' da kısa bir araştırmadan sonra aşağıdaki kitapların siparişlerini verdim.

Bu anlattığım olaylar yaklaşık bir ay kadar önce gerçekleşti. Şu anda bu kitaplarda az da olsa ilerlemiş durumdayım. İnanın sevgili okurlarım bu işe giriştiğim için çok ama çok memnunum. Biraz matematik, biraz teori biraz pratik derken bir şeyler kapmaya başladım bile. Herşeyden önce ilk konularda, eski dostumuz Hipotenüs' ü görmek bile son derece güzeldi. Konuları anlamaya başladığıma göre şimdi tek yapmam gereken öğrendiklerimi uygulayarak pekiştirmek ve sizler ile paylaşmak. Artık bu kadar laf kalabalığından sonra, bu makalemizin konusunada değinme zamanı geldi.

Bu gün, oyun programlamanın önemli temellerden birisi olan çarpışma (Collision) tekniklerine giriş yapacağız. Bir oyunda, bir birinden bağımsız öğelerin bir birleriyle çarpışmaları üzerinde duracağımız asıl konu olacak. Bir savaş oyununda tarafların bir birlerine karşı yaptıkları hamleler sonucu kimin kime vurduğunu tespit etmek, vuruşların yönüne veya şiddetine göre, darbeyi alan nesnelerin ne tür hareketlerde bulunacağına karar vermek açısıdından çarpışma teknikleri gerçekten önemlidir. Bu teknikler bir kaç tanedir. Bu gün ben sizlere, 2 boyutlu koordinat sisteminde dörtgensel şekillerin çarpışmalarının nasıl tespit edilebileceğini anlatmaya çalışacağım.

Dortgenlerin baz alındığı bu teknikte amaç, objeleri içine alan ve sınırlayan dörtgensel alanların birbirleri üstüne gelip gelmediklerinin tespit edilebilmesidir.

öncelikle durumu analiz edebilmek amacıyla aşağıdaki şekli göz önüne alalım.

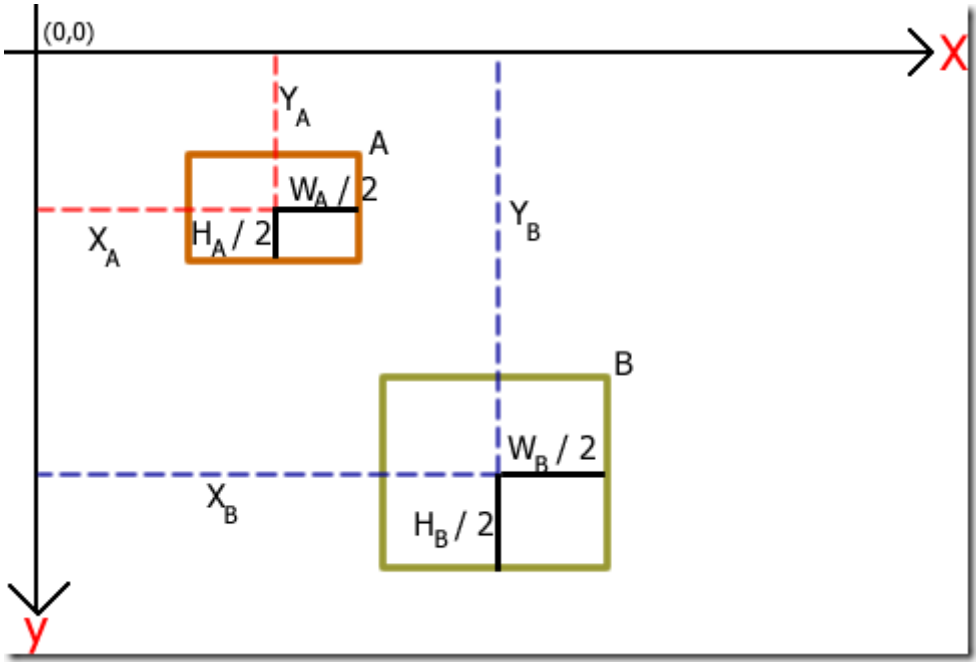


Şekil 1. Senaryo.

Senaryo gereği, köpek balığı ve yarış arabası nesnelerimizin hareketli olduklarını düşünelim. Burada, yarış arabası ve köpek balığı aslında ekranda piksel olarak yer kaplamaktadır. Bu iki piksel topluluğunun çarpışıp çarpışmadıklarını test edebilmek için kullanılabilecek en basit teknik, nesnelerin çevresini saran dörtgenlerin bir birleri üstüne gelip gelmediklerini tespit etmektir. Bu nedenle, senaryomuzda yarış arabımızı A isimli, köpek balığımızı ise B isimli dörtgenler ile çevrelediğimizi düşünelim. İşte bu noktadan sonra işin içine biraz matematik girecek.

Eğer bu iki nesnenin X koordinatları arasındaki fark, genişliklerinin yarılarının toplamından küçük ise ve iki nesnenin Y koordinatları arasındaki fark, yüksekliklerinin yarılarının toplamından küçük ise, bu iki dörtgen üst üste gelmiş dolayısıyla çarpışma (Collision) gerçekleşmiş demektir.

Bunun matematiksel olarak aşağıdaki şekil ile daha iyi anlayabiliriz.



Şekil 2. Koordinat, genişlik ve yükseklik ölçüleri.

Burada, her iki dörtgenin ekranın sol üst köşesine olan uzaklıkları ve genişlik ile yükseklikleri belirtilmektedir. Elbette bir nesnenin X koordinatı uygulamada bu nesnenin Left özelliğinin değerine, Y koordinatı ise Top özelliğinin değerine işaret etmektedir. Aynı şekilde genişlik için Width, yükseklik için ise Height özelliklerinden yararlanılacaktır. Yukarıda bahsettiğimiz çarpışma koşulunun matematiksel ifadesi ise aşağıdaki gibi olacaktır.

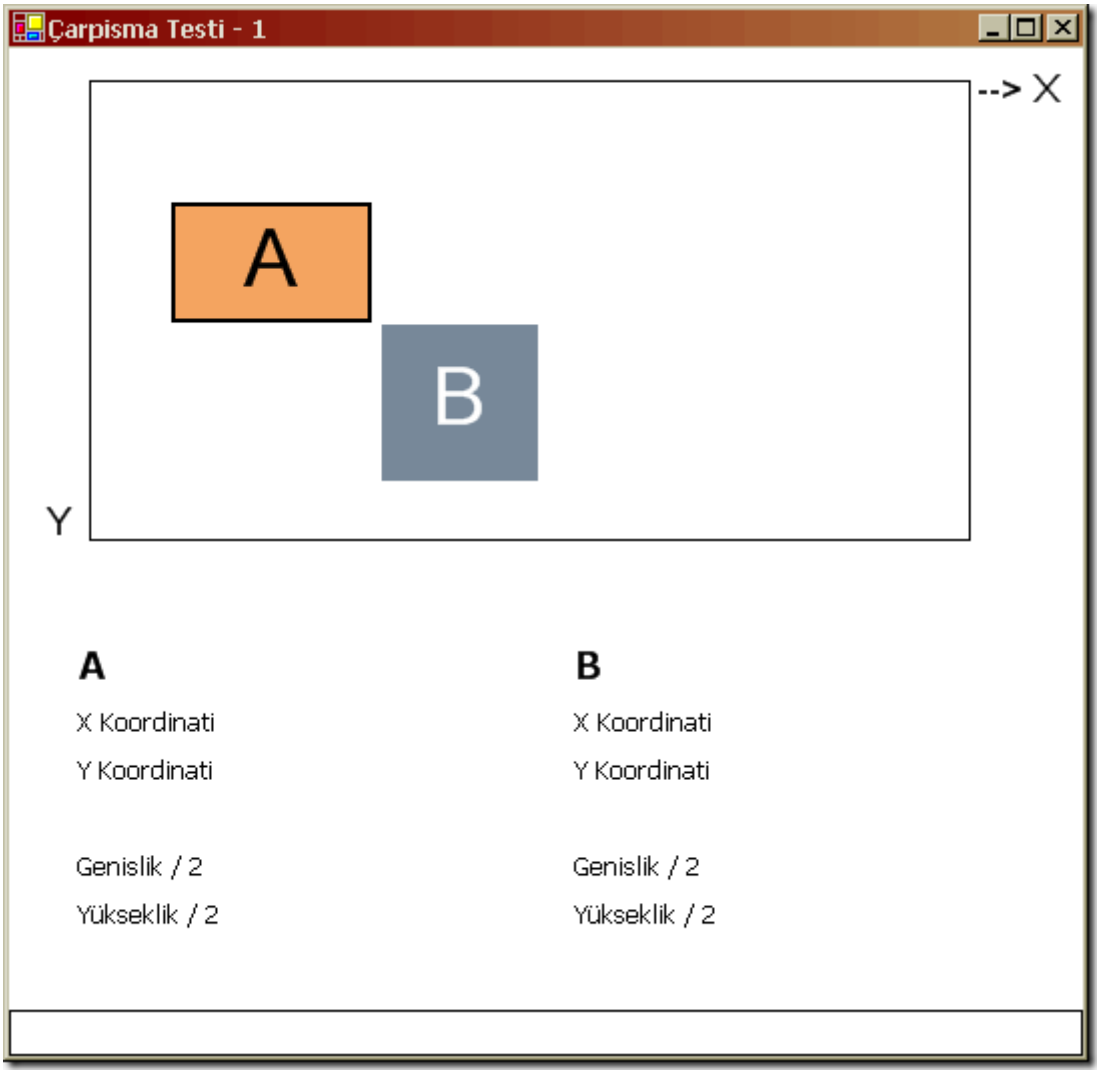
$$|X_A - X_B| < W_A/2 + W_B/2$$

VE

$$|Y_A - Y_B| < H_A/2 + H_B/2$$

Şekil 3. çarpışma (Collision) koşulları.

Dikkat edecek olursanız eşitsizliklerin sol taraflarındaki ifadelerde mutlak değer söz konusudur. Bu nedenle Math sınıfının Abs isimli metodu işimize çok yarayacaktır. Bu ifadedeki her iki eşitsizliğinde gerçekleşmesi halinde, dörtgenlerin üst üste geldiklerinden, dolayısıyla çarpıştıklarından söz edebiliriz. Bu teorimin gerçekliğini kontrol etmenin en güzel yolu uygulama üzerinde olacaktır. Bu amaçla aşağıdaki gibi basit bir windows uygulaması oluşturdum.



Şekil 4. Uygulama.

Bu uygulamada, A isimli dörtgen nesnesini klavyedeki A,S,D,E tuşları ile hareket ettirebileceğiz. Bizim buradaki amacımız, çarpışma olduğu takdirde bunun gerçekleşip gerçekleşmediğini tespit edebilmek. Bu amaçla, nesnelerin ekrandaki koordinatlarınıda izlediğimiz label nesnelerimiz var. Gelelim uygulamamızın kodlarına.

```
private void Yaz()
{
   lblXA.Text=A.Left.ToString();
   lblYA.Text=A.Top.ToString();
   lblWidthA.Text=Convert.ToString((A.Width/2));
   lblHeightA.Text=Convert.ToString((A.Height/2));
   lblXB.Text=B.Left.ToString();
   lblYB.Text=B.Top.ToString();
   lblWidthB.Text=Convert.ToString((B.Width/2));
   lblHeightB.Text=Convert.ToString((B.Height/2));
}
```

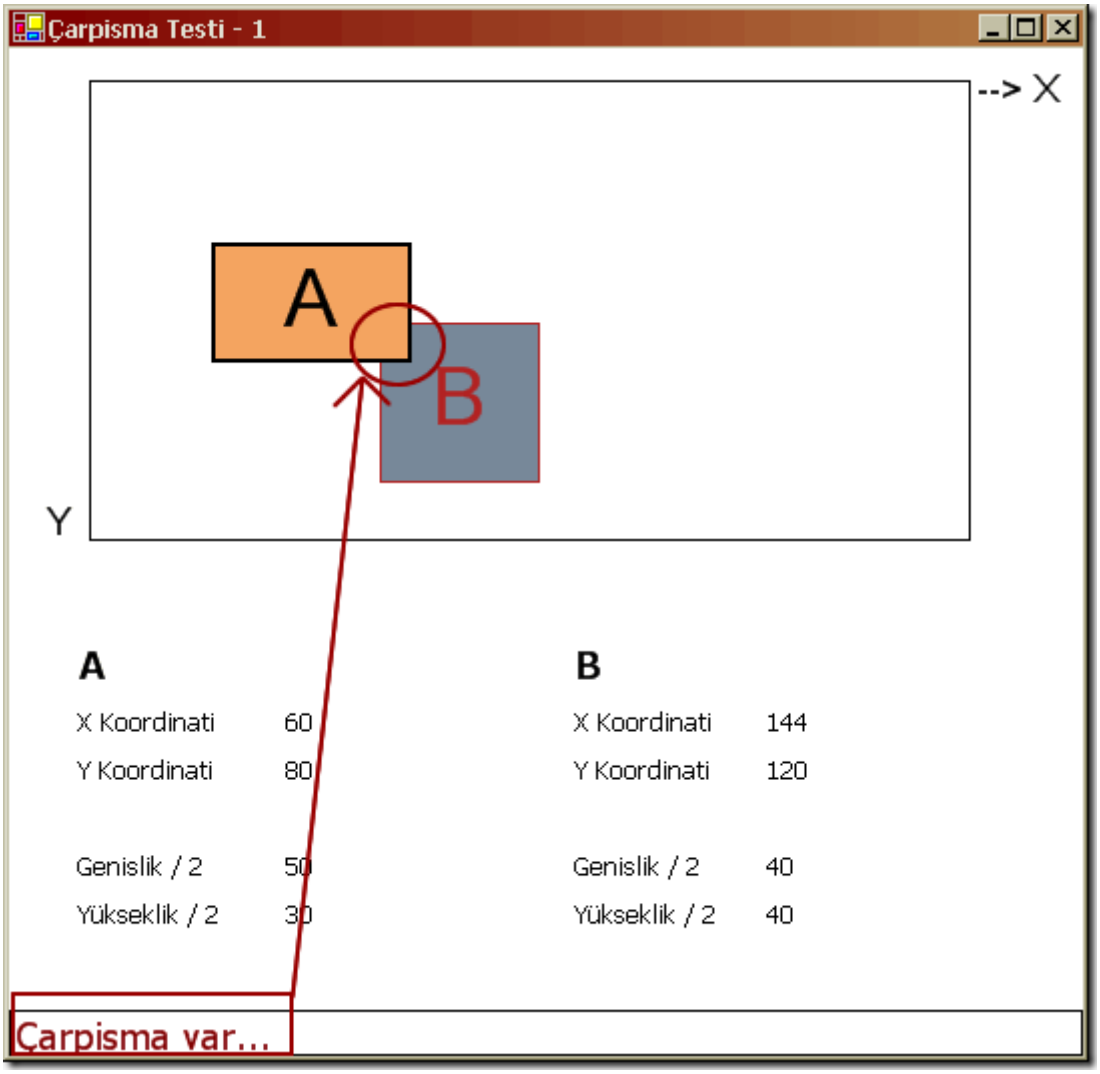
```
private void frmCarpisma_Load(object sender, System.EventArgs e)
{
   Yaz();
}
/* çarpışma Kontrolünün gerçekleştirildiği metod */
private bool CarpismaKontrol()
{
   float mutlakX=Math.Abs((A.Left+(A.Width/2))-(B.Left+(B.Width/2)));
   float mutlakY=Math.Abs((A.Top+(A.Height/2))-(B.Top+(B.Height/2)));
   float farkGenislik=(A.Width/2)+(B.Width/2);
   float farkYukselik=(A.Height/2)+(B.Height/2);
   if((farkGenislik>mutlakX)&&(farkYukselik>mutlakY))
   {
      return true;
   }
   else
      return false;
   }
}
private void frmCarpisma_KeyPress(object sender,
System.Windows.Forms.KeyPressEventArgs e)
{
   if(e.KeyChar==(Char)Keys.D)
   {
      A.Left+=10;
   }
   if(e.KeyChar==(Char)Keys.A)
   {
      A.Left-=10;
   }
   if(e.KeyChar==(Char)Keys.W)
   {
      A.Top-=10;
   }
   if(e.KeyChar==(Char)Keys.S)
   {
      A.Top+=10;
   }
   Yaz();
   if(CarpismaKontrol())
   {
      lblCarpismaKontrol.Text="çarpisma var...";
   }
   else
   {
```

```
      lblCarpismaKontrol.Text="";
   }
}
```

Uygulamada dikkat ederseniz, teoremi aşağıdaki kod satırlarındaki gibi uyguladık.

```
float mutlakX=Math.Abs((A.Left+(A.Width/2))-(B.Left+(B.Width/2))); /* X koordinatları arası farkın mutlak değeri. */
float mutlakY=Math.Abs((A.Top+(A.Height/2))-(B.Top+(B.Height/2)));  /* Y koordinatları arası farkın mutlak değeri . */
float farkGenislik=(A.Width/2)+(B.Width/2); /*Genişliklerin yarılarının toplamı.*/
float farkYukselik=(A.Height/2)+(B.Height/2); /* Yüksekliklerin yarılarının toplamı. */
if((farkGenislik>mutlakX)&&(farkYukselik>mutlakY)) /* Eğer koşul doğru ise çarpışma vardır. */
{
  return true;
}
else
  return false;
}
```

Şimdi uygulamamızı test edelim. A, S, D, W tuşları ile (bunlara basarken Caps Lock açık olmalı) A isimli buton kontrolümüzü 10’ ar birim hareket ettirebilmekteyiz. Eğer kutuları üstüste getirirsek çarpışma teoreminin başarılı bir şekilde gerçekleştiğini görürüz.



Şekil 5. çarpışma gerçekleşti.

Bu teknik en basit Çarpışma modelidir. Nesnelerin dörtgenler iÇerisinde düşünülmesi araba şeklinde olduğu gibi, boşluğa denk düşen alanlarında hesaba katılmasına neden olmaktadır. Ancak iş dairesel nesnelerin çarpışmasına geldiğinde (örneğin tenis toplarının raketler çarpması) hatta, daire ve karesel nesnelerin çarpışmasına geldiğinde dahada karmaşıklaşmaktadır.

Artık ilerleyen zamanlarda bu teoremleri incelemeye çalışacağım. öğrendikçede siz değerli okurlarıma aktaracağım. Artık 24 bölümmü sürer bir tetris oyunu yazmamız, 24 ay mı sürer bilemiyorum. En azından daha önceden bildiğim hatta mutlaka bildiğim ama farkına varamadığım algoritmaları öğrenmek beni oldukça memnun etti. Umuyorum ki sizlerde bu yazı dizisinden hoşnut kalırsınız. Tekrar görüşünceye dek hepinize mutlu günler dilerim.